Scheduling to Minimize Downtime in Human-Multirobot Supervisory Control*

Sandra Mau **John M. Dolan**

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA, USA

* This work is partially supported by NASA under Cooperative Agreement No. NNA05CP96A

Abstract

This paper proposes the use of on-line scheduling in human telesupervision of robot teams to determine and maximize the number of robots a human operator can manage at once. To maximize the number of robots in a team, the double-Shifted Shortest Processing Time (dSSPT) algorithm was proposed for this purpose. This scheduling algorithm has the objective of minimizing total downtime (or flow time) for the group. Simulated experimental results show that dSSPT: 1) reduces downtime with respect to existing scheduling algorithms; and 2) increases the achievable span-of-control (i.e., the number of robots supervised) in a human-multirobot team compared to an existing method of prediction [3], especially as the variation in task lengths increases.

I. Introduction

It is a common adage that time is money and in robot planetary exploration, this becomes all the more apparent. For example, 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 [8][9]. This means that it costs NASA approximately $100,000 per day in salaries per year to monitor just two robots. Also, over the course of two years the rovers have traversed less than 10 square miles combined [7]. 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 [10], including a crew of dozens to monitor and control these robots around the clock. It is evident that lowering the human-to-robot ratio would decrease exploration costs [10]. 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 [10].

Why do we need a human in the loop? 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. For example, in the DARPA Grand Challenge (2006) autonomous ground vehicles traversed a significant stretch of desert without human control for the first time, but to an experienced human driver, driving across the Nevada desert is comparatively straightforward. The current state of the art in autonomous robotic planetary exploration is well behind that of the Grand Challenge vehicles. However, robots are still ideal for use for jobs that would endanger humans, such as planetary exploration. 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. But exactly how many robots can a person handle? Crandall et al. [2] point out that it depends on the autonomy of the robots, the complexity of the environment, and even the efficiency of the user interface. Based on those factors, they designed experiments to estimate the number of robots a human can supervise in a group, the so-called "span-of-control" problem. A limitation of their method, which is described in further detail later, is that everything is based on performance averages. It does not explicitly consider cases in which the performance is below or above the mean; for example, cases in which more than one robot may need human assistance simultaneously.

This paper describes a quick and efficient scheduling algorithm to increase the span-of-control (the number of robots a human can handle) beyond an estimate based on pre-computed performance averages. This "double Shifted Shortest Processing Time" (dSSPT) algorithm attempts to minimize a robot team's downtime – the time robots spend inoperative while waiting for and receiving human attention. Additionally, this algorithm's ability to reduce the overall waiting time of tasks in queue makes it a useful tool for other space applications as well, ranging from helping space flight controllers to juggle competing tasks to organizing science experiments on satellites.

Section II of this paper will describe related work in human-multirobot teams and give a brief overview of relevant scheduling algorithms. Section III derives the algorithm. Sections IV and V discusses some experiments and results.

II. Related Work

This section outlines the relevant literature on human-multirobot group interaction as well as some scheduling algorithms and terminology referenced later in the paper. It also argues for the use of scheduling in human-multirobot interaction applications.

A. Human-Multirobot Group Interaction

Although there is a growing body of research on human supervision of multi-robot groups [1], there is little research providing quantitative metrics for the human-robot interaction (HRI) aspect of such systems. The crucial HRI problem that this paper addresses is how many robots one human can handle or the "span-of-control". Crandall et al. [2] asserted that the number of robots in a group depends on the effectiveness of software control interfaces and robot autonomy. He developed a method for determining the maximum number of robots that a single human can supervise based on the "Neglect Time" (NT) and "Interaction Time" (IT) of each robot in an independent heterogenous group. Neglect Time is the (average) amount of time a robot can remain above a desired performance level without needing human assistance. Interaction Time is the (average) amount of time a human requires to bring a robot back to a desired performance level. The maximum number of robots that can be handled by a single human operator is then given by one plus the number of interactions that can fit into the neglect time of a rover, i.e., NT/IT + 1. This is illustrated in Figure 1, where each thick arrow represents the average IT of an individual robot and the average NT is bracketed by the vertical grey lines.

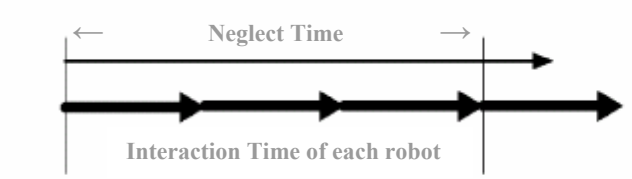


Figure 1: *Human-Robot Team from Crandall [3]*

To determine these values, user experiments were conducted to obtain performance averages for each robot. The goal of the work was to validate the NT and IT concepts via experiment, rather than to optimize human supervisory control. However, the fact that the actual NT and IT values vary leads to the likelihood of instances of overlap, where multiple robots will require human assistance simultaneously. In the worst-case scenario, all robots could potentially require human assistance at once, and those not receiving assistance experience "downtime". This naturally suggests the investigation of an efficient scheduling method that seeks to minimize downtime.

To our knowledge, there has been no work done on scheduling in regards to human-multirobot teams in particular. There has been a lot of research on distributive task allocation for autonomous multi-robot teams [1], but none focused on multirobot scheduling for human supervisor(s). However, there is a huge variety of relevant existing scheduling algorithms in computer processing, networking, management and various other domains, the key relevant concepts from which are described in Section III.

In summary, our work extends Crandall's by proposing that an efficient scheduling algorithm to lower downtime can increase the span-of-control for a human operator. This idea is verified through experiment in Section V.

B. 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). In this subsection, terminology used in the paper is defined and some common approaches are explained.

Scheduling Terminology

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.

Preemption: The idea that a task can be interrupted during processing, placed on hold, and then resumed from the point of interruption. When used to describe a scheduling algorithm, it means that such an algorithm can break tasks up into various chunks and schedule the chunks at various times, not consecutively.

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 [6]. 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 $a_1 <= a_2 <= ... <= a_n$. 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 $r_1+a_1 <= r_2+a_2 <=...<= r_n+a_n$. 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 [4]. 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 non-preemptive algorithms, SRPT was not used.

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

This greedy algorithm schedules tasks in the order they arrive (in other words, in order of release time.) This can be expressed as $r_1<=r_2<=...<=r_n$. The advantage of this algorithm is that the implementation is very simple: a queue. It also minimizes maximum downtime [2]. 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.

III. Derivation of dSSPT Algorithm

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.

Assumptions

Since a human overseeing a group of robots cannot possibly be aware of every robot situation simultaneously, it is assumed 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 non-preemptive and instantaneously static with time, which is further explained in the next subsection.

Scheduling Two Robots

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 2.

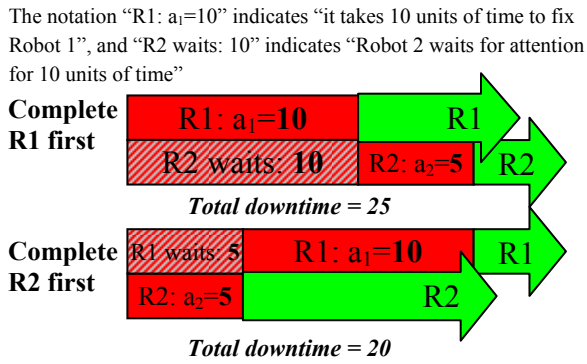


Figure 2: *Simultaneous robot tasks to be ordered*

An example of the basic case is given to 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, a_i 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). 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 $2*a_1-\Delta t+a_2$ whereas if we do swap, downtime would be $2*a_2+\Delta t+a_1$, 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. 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 $a_1+2*r_1 > a_2+2*r_2$ 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 also illustrates the non-preemptive assumption – that if the processing of R1 is interrupted, when it is later readdressed the processing time is assumed

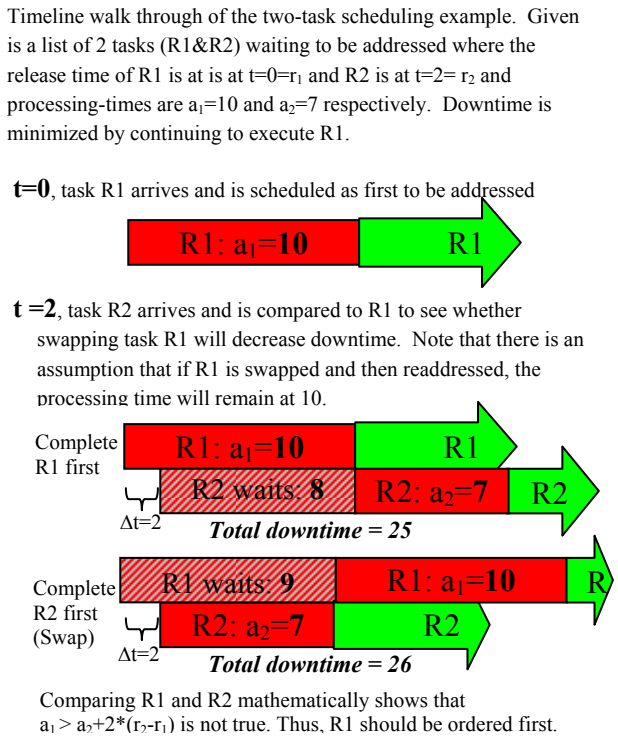


Figure 3: *Timeline of scheduling example*

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 a_1 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.

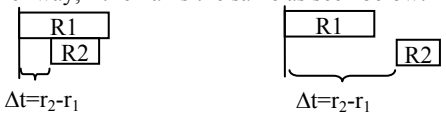
The idea of factoring in the release time resembles the SSPT application that orders tasks such that the processing times for each task is "shifted" by its release time: $r_1+a_1 < r_2+a_2 \rightarrow a_1 < a_2+(r_2-r_1)$ or $a_2+\Delta t$ [4], which is also considered an on-line algorithm. The factor 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.

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 4. 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 4 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 4. When running this on-line, the cases of 4(a) and (c) would mean there are only two tasks remaining in the list (R2 and R3).

For two tasks, with R1 arriving before R2, task R2 can potentially arrive in R1's processing time interval or afterward. Either way, Δt remains the same as seen below:



For three tasks, arriving in the order R1, R2, R3, where R1 is fixed and a_0 is the time at which R1 is finished. A pair-wise comparison in the ordering of R2 and R3 is being made. There are potentially 3 cases to look at for Δt:

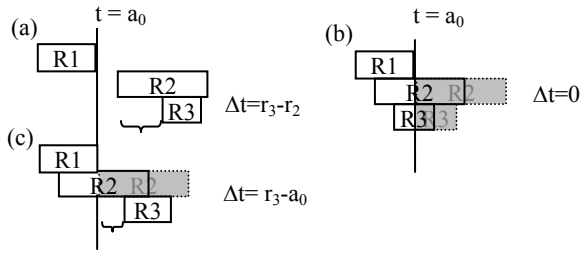


Figure 4: *Time shift for 3 robots*

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 4(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 r3-(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 4(b))
   The downtime for R2 before R3 is:
   $(a_1-\Delta t_2+a_2) + (a_1-\Delta t_3+a_2+a_3)$ *[1a]*
   The downtime for R3 before R2 is:
   $(a_1-\Delta t_3+a_3) + (a_1-\Delta t_2+a_2+a_3)$ *[1b]*

   We want to swap R3 ahead of R2 if its wait time is smaller. Setting [1b]<[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 4(c))
   The downtime for R2 before R3 is:
   $(a_1-\Delta t_2+a_2) + (a_2+a_3-(r_3-a_0))$ *[2a]*
   The wait time for R3 before R2 is:
   $a_3 + (a_1-\Delta t_2+(r_3-a_0)+a_2+a_3)$ *[2b]*

We want to swap R3 a R2 if its wait time is smaller. Setting [2b] < [2a], 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 4(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(a0,r2) as can be seen in Figure 4.

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.

dSSPT Algorithm

To summarize, dSSPT is implemented by sorting each new task into the dSSPT-scheduled list by comparing the new task to existing tasks starting from the back and working towards the front. Given a new task *j*, and an existing task it is being compared to *i* (with ri <= rj), do pairwise swaps down the list according to the following:

Principle: Will swapping minimize processing time?

Evalulation:

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

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

then swap tasks *i* and *j*.

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

Computation Time

The computation time for this greedy scheduling algorithm has the same complexity as other classic algorithms such as SPT, and SSPT of $O(n^2)$, thus making dSSPT an efficient polynomial time alternative to those algorithms.

IV. Experiments 1: Downtime Efficiency of Scheduling Algorithms

The experiments outlined here are designed with human-multirobot group scenarios 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 set of experiments was simulated as on-line and each data point presented below (for varying the number of robots in the group) was an average over 100 trials.

Crandall et al. [3] suggest that in human-multirobot scenarios an average "neglect time" (NT), or mean time to failure, and average "interaction time" (IT), or mean time to fix or mean processing time, can be determined. This is limiting, since it implies that most tasks take around the same amount of time to address on average; however, we used it as a starting point for experiments.

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 6 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.

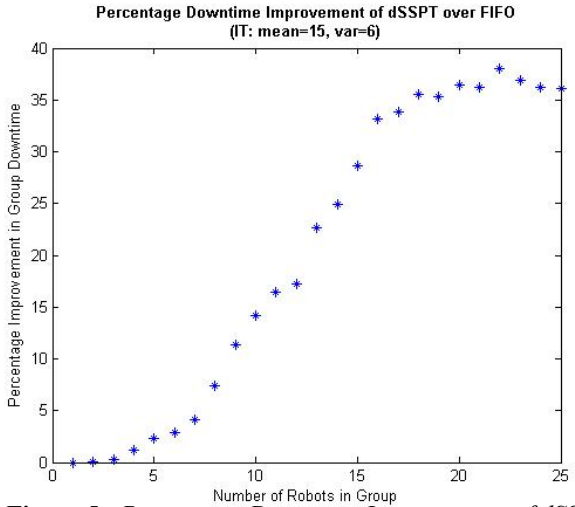


Figure 5: *Percentage Downtime Improvement of dSSPT*

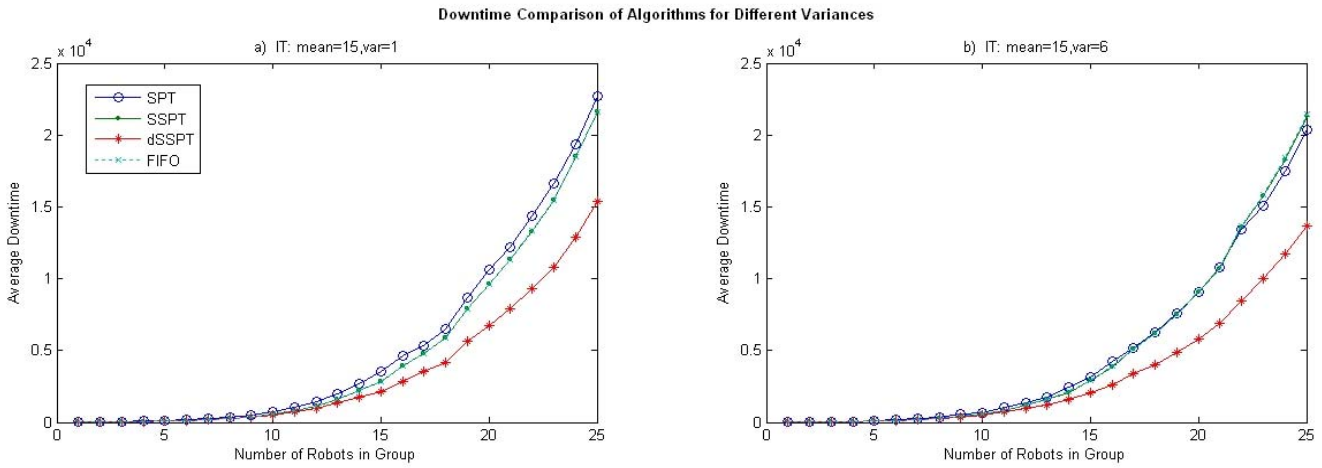


Figure 6: *Downtime comparison for changing variances*

.

For the trial given in Figure 6(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 5 for the trial plotted in Figure 6(c). The shape of this percentage improvement curve is similar for all experiment comparisons, so the other plots were not reproduced here. In Figure 6(b), where the IT had a variance of 6, the dSSPT schedule had

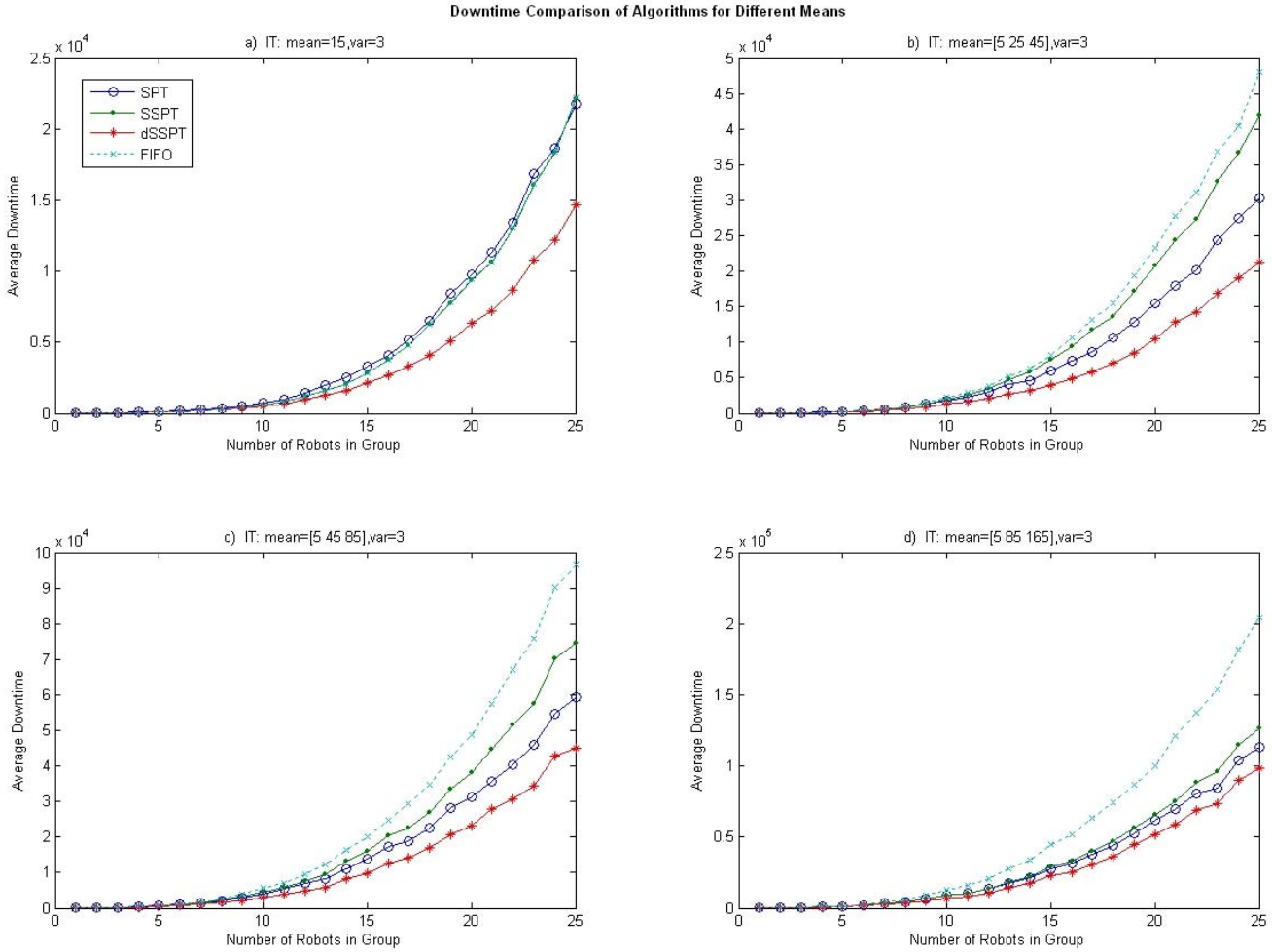


Figure 7: *Downtime comparison for changing mean IT*

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.

It makes intuitive sense that FIFO ($r_1 < r_2 <\ldots< r_n$) and SSPT ($r_1+a_1 < r_2+a_2 <\ldots< r_n+a_n$) performed nearly the same on average if the mean FT (a_1, a_2, …, a_n) is the same on average. The SSPT basically delays every task release time by a constant offset: the mean fix-time ($a_1 \approx\ldots\approx a_n \approx$ 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 FT. Though FT 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 7.

The second set of experiments simulated tasks of three classes with short, medium and long IT, which is more realistic than all tasks having nearly the same IT. This set of experiments varied the mean of the short, medium, and long ITs from 5, 25, and 45 to 5, 85, and 165 respectively, and allowed each IT class size a variance of 3, as seen in Figure 7. 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. 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 7(b), dSSPT shows up to 54% improvement over FIFO, similarly in 7(c), it plateaus around 52%, and in 7(d) it is around 50%. In 7(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 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. For sparse schedules, it uses the improved SSPT and for dense schedules, it uses SPT. 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.

V. Experiments 2: Scheduling in Human-Multirobot Supervisory Control

As mentioned in Section II, research by Crandall 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

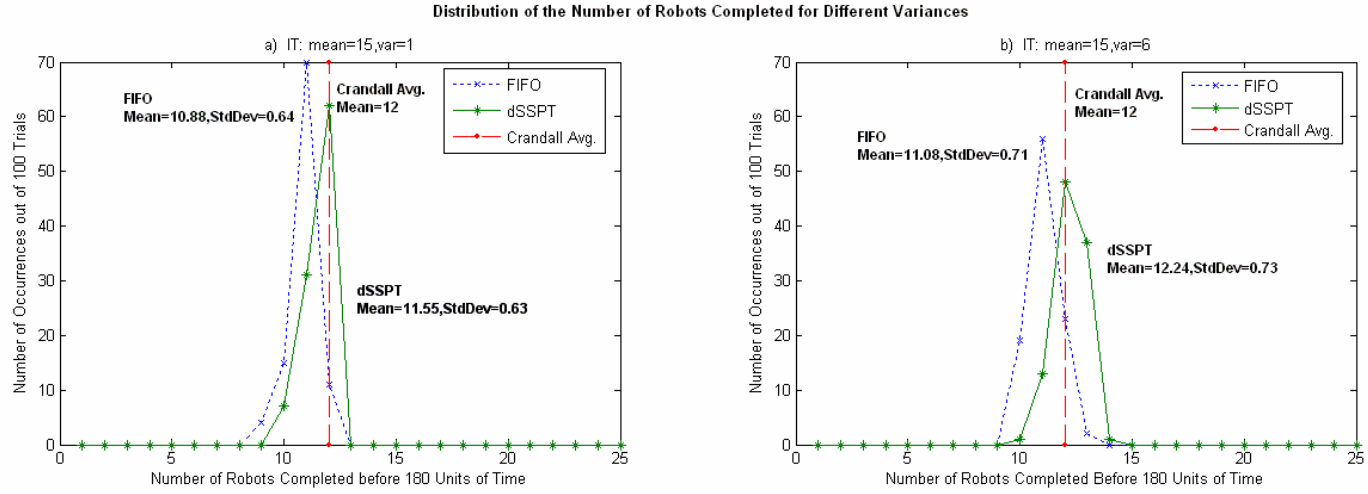


Figure 8: *Number of robots addressed within NT for varying variance*

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 8) as well as mean to simulate long, medium and short tasks (Figure 9). For this section, the main focus is the number of robots that can be scheduled and completed within the span of NT (180.units of time)

According to Crandall et al. [3], 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 8), one would expect according to Crandall and Goodrich 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 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 8(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 9. 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 9(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.

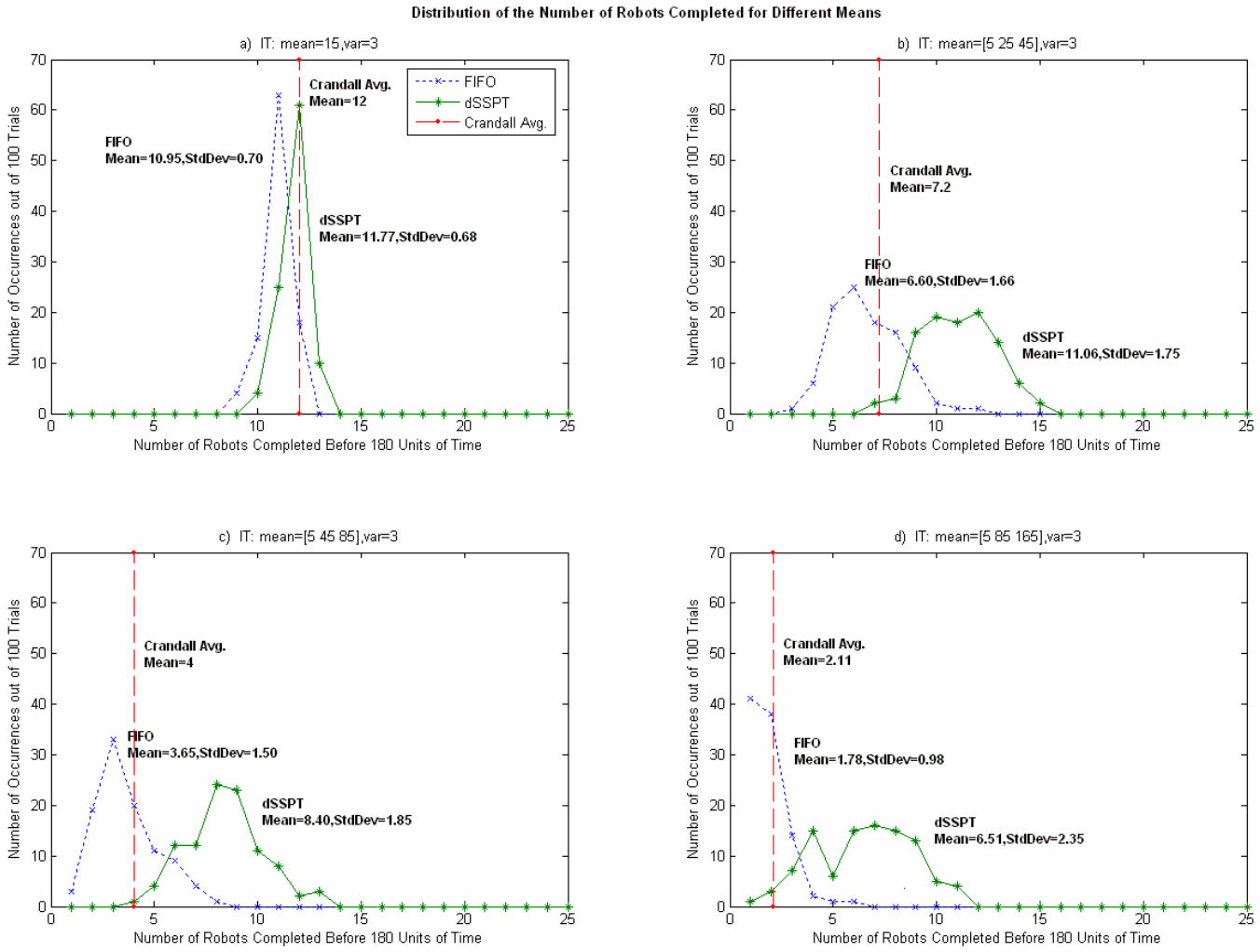


Figure 9: *Number of robots addressed within NT for varying ITs*

With Crandall's method of using average performance of the team, 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 9(c), the mean for FIFO is 3.65 and Crandall's average is 4, compared to dSSPT's mean of 8.40. In 9(d), where the IT of tasks were [5 85 165], FIFO has a mean of 1.78 with a deviation of 0.98 and Crandall's 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 as proposed by Crandall. The results of scheduling would yield exactly Crandall'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, Crandall's static measure of span-of-control does not apply to the general case.

VI. Conclusions

This paper makes two main contributions:

1) *A scheduling algorithm, dSSPT, was developed based on a simple, greedy heuristic that tries to minimize downtime for a team. It outperforms the standard FIFO, SPT and SSPT algorithms and manages to do so in polynomial time.*

As shown in the first set of experiments, the dSSPT algorithm performed better than SPT, SSPT and FIFO in terms of minimizing downtime for a group of tasks. It performed especially well for cases where the IT of tasks had a greater variance. However, one drawback to it is inherited from SPT, in which tasks with larger IT tend to be placed later. If all tasks are of equal importance, this does not matter as much. However, if tasks have different, possibly time-varying importance, one possible solution is to weight the tasks.

Even though the dSSPT scheduling algorithm was originally conceived to augment supervisory control in human-multirobot groups, it can also be used in general for other applications that desire similar qualities of decreasing wait-time or downtime for a group of tasks.

2) *We also show that when scheduling algorithms are applied in human-multirobot group supervisory control a downtime-efficient, dynamic, on-line scheduling algorithm can increase the span-of-control in comparison to that predicted using a static performance averaging method.*

As robots become more autonomous and tasks more complex, the need for scheduling in human-multirobot interactions will become more prevalent. The potential savings in time and money for such a simple implementation is definitely a worthwhile trade-off.

References

[1] Julie A. Adams and Marjorie Skubic. "Introduction to the Special Issue on Human–Robot Interaction". IEEE Transactions on Systems, Man, and Cybernetics—Part A: Systems and Humans, Vol. 35, No. 4, Jul 2005

[2] Nikhil Bansal. "Algorithms for Flow Time Scheduling," Ph.D. thesis. Carnegie Mellon University, Computer Science Department. Dec 2003.

[3] J. W. Crandall, M. A. Goodrich, D. R. Olsen, and C. W. Nielsen. "Validating Human-Robot Interaction Schemes in Multi-Tasking Environments." Systems, Man, and Cybernetics Part-A: Systems and Humans. Volume 35, Issue 4, Pages 438-449. 2005.

[4] A. Elfes, J. Dolan, G. Podnar, S. Mau, and M. Bergerman, "Safe and Efficient Robotic Space Exploration with Tele-Supervised Autonomous Robots", Proceedings of the AAAI Spring Symposium, March, 2006, pp. 104-113.

[5] X. Lu, RA Sitters, L. Stougie, "A class of on-line scheduling. algorithms to minimize total completion time," Oper. Res. Lett. 31 (2003) 232–236.

[6] John A. Stankovic et al., "Implications of Classical Scheduling Results for Real-Time Systems", IEEE Computer, Special Issue on Scheduling and Real-Time Systems, June 1995.

[7] Elizabeth Svoboda. "Astronomers Planning Close-Ups of Mars From (of All Things) a Balloon," The New York Times. 13 Jun 2006. <http://www.nytimes.com/2006/06/13/science/space/13mars.html?pagewanted=1&8dpc>.

[8] Sarah Wang. "One (Martian) Year and Counting," Pasadena Weekly. 12 Jan 2006. <http://www.pasadenaweekly.com/article.php?id=3047&IssueNum=2>.

[9] "Mars Exploration Rover," Wikipedia. 12 Jun 2006. <http://en.wikipedia.org/wiki/Mars_Exploration_Rover_Mission>.

[10] "Mars Exploration Rover Mission: People," NASA, Jet Propulsion Laboratory, Cal Tech. <http://marsrovers.jpl.nasa.gov/people/>.

[11] "The Vision for Space Exploration", NASA, February 2004.