Paper 298-2009

Implementing a Repair Heuristic for Unplanned Work in Resource Constrained Project Schedules Using Base SAS® and SAS® PROC CPM

Sandra Archer, Robert L. Armacost, and Julia Pet-Armacost
University of Central Florida, Orlando, FL

ABSTRACT

Scheduling is the process of developing a plan for a set of tasks that compose an overall project that must comply with a set of precedence constraints and optimize a project objective. The area of focus here is the Stochastic Resource Constrained Project Scheduling Problem (SRCPSP) with Stochastic Task Insertion (STI). In order to facilitate experimentation on the development of new predictive techniques for the SRCPSP with STI, an automated process was developed to repair the schedule in "real time" as some tasks occurred and others did not. A right/left-shift repair heuristic is presented here that was formalized and written using Base SAS code and SAS/OR® software's CPM procedure. Base SAS is used to control the schedule data and shift tasks to the right/left-shift as needed. Additionally, PROC CPM is used in an iterative fashion to check the repaired schedule for resource constraint violations. This paper focuses on the experiment environment; however, this repair heuristic has the potential for real-world application by project managers who have the need to run "what-if" scenarios for project planning.

INTRODUCTION

Scheduling is the process of developing a plan for a set of individual tasks, or activities, that comprise an overall job or project. Generally, tasks are ordered to comply with a set of precedence constraints and to optimize an overall project objective. Studies of this important topic in the area of project management expanded rapidly in the latter half of the last century with the increased ability to test various scheduling techniques on variations of the problem using computer technology. The area of focus here is the **Stochastic Resource Constrained Project Scheduling Problem (SRCPSP) with Stochastic Task Insertion (STI)**. The STI problem is a specific form of the SRCPSP, which may be considered to be a cross between two types of problems in the general form: the Stochastic Project Scheduling Problem, and the Resource Constrained Project Scheduling Problem.

Stochastic project scheduling is generally concerned with developing an optimal task order that is robust to uncertainty in certain areas. These areas may include possible resource breakdowns, uncertain resource availability, or variable task durations. Project managers may develop an initial plan before work begins (**predictive**) or react to events as they occur and reschedule tasks as necessary (**reactive**), generally with the objective to minimize expected project duration.

Resource constrained project scheduling is a form of the general resource allocation problem that includes various methods for project managers to deal with limited resources. Resource allocation problems include resource leveling (minimizing the variation in resource levels required over the course of the project) and time/cost trade-off (determining the value of project time completion in terms of resource cost). Variations of resource constrained project scheduling include multi-project scheduling, multi-mode resource constrained project scheduling with and without activity splitting, renewable and non-renewable resources, and preemption and non-preemption constraints. The Deterministic Resource Constrained Project Scheduling Problem (DRCPSP) assumes deterministic resource availability and task duration, and is concerned with the ordering of activities to optimize an objective, usually project duration. There are many heuristic solutions and exact algorithms in the literature for solving various cases of this problem.

The classic Stochastic Resource Constrained Project Scheduling Problem (SRCPSP) has the objective to minimize expected project duration with deterministic resource availability while activity durations occur stochastically. Techniques to solve the problem include predictive scheduling (developing baseline schedules that protect the project duration), and reactive scheduling (providing managers with a plan for rescheduling in real-time with current information). The literature is dominated by those methods to develop a robust baseline schedule.

STOCHASTIC TASK INSERTION PROBLEMS

The Stochastic Task Insertion (STI) problem is a specific form of the SRCPSP. Selim (2002) defined the STI problem as one in which some project activities may or may not occur with a certain probability, and termed these activities as

"unplanned". Under resource constraints and precedence relationships, the occurrence of an unplanned activity may cause other activities to be delayed due to resource usage by the unplanned activity or necessitate a re-sequencing of activities to accommodate additional predecessor constraints. There is limited research on the STI variation of the SRCPSP. Selim (2002) developed a new set of performance measures focusing on a schedule's robustness to disruptions, examined the effect of network, resource and stochastic factors on schedule robustness, and also provided insights into two extreme approaches to obtaining a baseline schedule. Grey (2007) introduced a partial buffering heuristic for the STI, applying concepts from Goldratt's (1997) Critical Chain and Buffer Management method for baseline schedule development for project planning under uncertainty. Grey demonstrated that a partial buffering approach may improve project duration and other metrics for project duration and stability.

While Selim and Grey provided heuristics for developing a robust baseline schedule before any work begins (predictive), other authors focused on how project managers may deal with inserting an unplanned task into an existing schedule (reactive). A polynomial insertion algorithm for the RCPSP is provided by Artigues, Michelon, and Reusser (2003) and used for rescheduling with the occurrence of an unexpected activity. Similarly, Duron, Proth, and Wardi (2001) developed an algorithm to insert a randomly occurring task of stochastic duration into a single-resource constrained schedule with the goal of completing this random task by its due date while minimizing the sum of the delays of the initial task schedule. Ourari and Bouzouia (2003) have also developed algorithms for task insertion on single-machine job shop scheduling problems.

Techniques applied by other researchers to the general form of the SRCPSP may not be appropriate for the STI problem. Therefore, an opportunity to examine various solution methods and heuristics that had been developed for the general SRCPSP problem and test their use on the STI variation of the problem was identified. Opportunities to expand upon the work of Selim (2002) and Grey (2007) by testing their techniques on various configurations of the problem to determine if new techniques are appropriate under these conditions was determined to be an area of research interest.

In order to facilitate experimentation on new predictive techniques for the SRCPSP with STI, an automated process was developed to repair the schedule in "real time" as some tasks occurred and some did not occur. A right/left-shift repair heuristic is presented here that was formalized and written using Base SAS code and SAS/OR PROC CPM. Base SAS is used to control the schedule data and shift tasks to the right or left as needed. Additionally, PROC CPM was used in an iterative fashion to check the repaired schedule for resource constraint violations. This paper focuses on the experiment environment; however, this repair heuristic has the potential for real-world application by project managers who have the need to run "what-if" scenarios for project planning. The program was written with SAS 9.3.1 SP3 in Windows XP.

BACKGROUND

There have been many heuristic reactive solutions proposed for the SRCPSP, including those that offer a full-rescheduling option, depending on the project objective function (Demeulemeester and Herroelen 2002; Herroelen and Leus 2004b)). Schedule repair includes simple control rules such as a right-shift rule (Sadeh, Otsuka, et al. 1993) which involves moving activities affected by the schedule breakdown forward in time. However, Herroelen and Leus (2004b) warn that this reactive strategy may lead to poor results, since it does not re-sequence activities.

In the task insertion problem, the work of Selim (2002) and Grey (2007) focused on the development of **initial baseline (predictive or proactive)** schedules for a project. Existing performance measures did not address flexibility for unplanned work until Selim (2002) developed robustness measures for the SRCPSP with STI. One set of measures was project duration related, comparing the actual "as-run" work schedule (**modified baseline**) to the optimal (**perfect knowledge**) schedule and the original (**baseline**) schedule. The optimal (perfect knowledge) schedule assumes perfect knowledge of all tasks that occurred or did not occur. Another set of measures were re-sequencing related and considered the number of tasks whose start times varied between the modified base and perfect knowledge schedule.

Using these measures, Selim (2002) tested the effect of network factors on robustness including: network topology (order strength and complexity index), resource characteristics (resource factor and resource constrainedness), the location of the stochastic occurrence in the network (early or late), and the number of stochastically occurring tasks (high or low).

RanGen software developed by Demeulemeester, Vanhoucke, et al. (2003) was used by Selim to generate network instances of 32 tasks with order strength (OS), resource factor (RF) and resource constrainedness (RC) as defined by Selim. Selim experimented with high and low settings of stochasticity and location of the stochastic events in the schedules. Selim applied two baseline scheduling techniques: **optimistic** (where none of the stochastically occurring tasks were scheduled) and **pessimistic** (all of the stochastically occurring tasks were scheduled). After identifying which tasks occurred for the optimistic case, a "right-shift" rescheduling policy was implemented to shift all tasks to the right to provide resources and ensure precedence constraint compliance for the unplanned work. Likewise, when

tasks did not occur for the pessimistic case, a “left-shift” rescheduling policy was implemented to shift all tasks to the left to ensure minimization of project duration.

Selim's work provided insights into how the research factors studied affect schedule robustness. This type of information was intended to be used for guiding the development of heuristics for robust scheduling techniques for the SRCPSP with STI.

Inspired by Goldratt’s (1997) concept of inserting buffers to protect a project schedule, Grey (2007) provided an extension of Selim’s (2002) work on the SRCPSP with STI by studying seven new **buffer sizing techniques** for this problem case. Grey replicated Selim’s networks to provide a comparison of results among the methods, and also developed a new measure of robustness to account for the absolute difference in task start times between the schedules. Results allowed Grey to conclude that the four partial buffering heuristics which incorporated knowledge of the stochasticity of the tasks were improvements over the extreme (optimistic and pessimistic) approaches studied by Selim (2002).

RIGHT AND LEFT SHIFT REPAIR POLICIES

Both Selim and Grey implemented a right-shift policy to react to unscheduled tasks that occurred and a left-shift policy to react to scheduled tasks that did not occur. The problem set here assumed 32 tasks of various durations that require various levels of 2 types of resources. It also assumed there were 10 renewable units for each type of resource available.

Leon, Wu, et al. (1994) cited two advantages of using a right-shift policy. First, its implementation is simple and second, deviation is minimized (where deviation is defined as the difference in activity sequence). Selim (2002, pg. 48-53) described the details of implementation in the STI case as follows:

> *The schedule is evaluated at each time period and if a stochastic activity occurs at a certain time period it is inserted into the schedule as soon as resources are available and the precedence relations are not violated. The steps are outlined below:*

- *Start with the optimistic schedule and create a Gantt chart.*
- *Determine which of the stochastic tasks occur and the sequence of their insertion in the optimistic schedule. The insertion sequence is determined by the sequence of tasks in the list of successors. The order of insertion is as follows:*
- *The stochastic task should be inserted as early as possible after the predecessor activity is finished without violating the resource and precedence constraints. The list of successors is used to determine the earliest point at which an activity can be started*
- *The sequence of the tasks in the optimistic schedule should be maintained.*
- *If two tasks can be inserted at the same time period, start with the task with the lower sequence number.*

Selim (2002) also provided a detailed implementation description for a left-shift repair of the STI case as follows:

- *Start with the pessimistic schedule and create a Gantt chart.*
- *Remove the stochastic tasks that do not occur from the Gantt chart representing the pessimistic schedule.*
- *Starting from time period 1, determine which of the tasks that do occur can be started earlier, that is, shifted to the left, while keeping track of the precedence and resource constraints. The sequence of tasks in the pessimistic schedule should be maintained as much as possible.*
- *The schedule sequence can be altered only if there are resources available to schedule a task different from the immediate successor of the stochastic task.*
- *The list of successors was used to determine the earliest point at which an activity can be started if one or more of its predecessors have been eliminated.*
- *If two tasks start at the same time period and they can both be shifted to the left in the compressed pessimistic schedule, start with the task with the lower sequence number.*

The reactive procedures used by Grey (2007) assume a left-shift procedure when removing tasks that did not occur and a right-shift procedure when inserting tasks. Both of these procedures maintain the order of tasks as much as possible to the initial baseline schedule.

It is reasonable that a project manager may implement such policies manually in order to react to schedule changes for a small project with a small number of tasks. However, larger projects will most likely require an automated means by which to re-schedule tasks. An additional motivation for developing a coded version of the right and left shift polices was to provide the ability to replicate the work completed by Selim (2002) and Grey (2007) quickly for comparison to newly developed heuristics.

Instead of two separate heuristics, one combined right/left-shift heuristic has been developed to repair any buffered schedule, including the optimistic (0%) and pessimistic (100%) baselines. This has been implemented successfully using SAS code. The logic for this heuristic follows.

SAS® PROC CPM

In order to automate a right/left-shift repair policy, it is necessary to have the ability to determine if resource violations are occurring when performing the shift. When Selim and Grey performed the repair, resource constraints were enforced manually, specifically, by visually inspecting a 2-dimensional Gantt chart to ensure that more than the number of available resources was not scheduled during any given point during the project. Most Gantt charts are only 1-dimenstional with the horizontal axis providing a scale for the time periods of the project. The vertical axis of the chart usually has no scale; rather, it typically contains a qualitative label describing the specific task. A 2-dimensional Gantt chart also represents time on the horizontal axis, but also represents the number of units of resource units on the vertical axis.

It is relatively easy to use SAS base code to change the start and end time of a task within a project based on a set of logic that performs the shift while ensuring precedence constraints are not violated. However, ensuring that resource constraint violations do not occur is more difficult. This would require code that would sum the resource usage of each concurrently occurring task during every time period. Instead of developing a new set of code to ensure resource constraints are not violated, it was determined that the output data sets of PROC CPM could be used to check for resource constraint violations. Specifically, PROC CPM has the ability to alert us during the right/left-shift schedule repair if more resources are scheduled during any given time period than are available.

The CPM procedure is used for scheduling project activities subject to precedence, time, and resource constraints, allowing the user to choose from a variety of options to control the scheduling process. The following is a summary of the tables available for output from the PROC CPM procedure.

Table 1: PROC CPM Output Data Sets

Relevant Input Data Sets for PROC CPM:	
Activity Data Set	contains all activity-related information including activity name, precedence information, calendar, progress information, baseline (or target schedule) information, resource requirements, and time constraints
Resource Data Set	contains resource types, availabilities, priorities, and if any alternate resources exist
Relevant Output Data Sets from PROC CPM:	
Schedule Data Set	contains the early, late, baseline, resource-constrained, and actual schedules
Resource Schedule Data Set	contains the schedules for each resource used by an activity
Usage Data Set	contains each resource's usage

SAS® PROC CPM uses a serial-parallel method of scheduling to solve a resource constrained schedule. The name of this heuristic describes itself as being "serial in time and parallel in activities". A description of the heuristic that SAS® PROC CPM uses is provided in SAS documentation (SAS Institute 2006), a summary of which follows.

An initial tentative schedule is determined without considering resource constraints, with each activity scheduled to start as early as possible (*e_start*). To correct for resource constraints, time is set equal to the earliest *e_star*t. All activities with *e_star*t equal to time are sorted in a priority order based on their latest start, which is the latest time the activity can be started based on its successor activities' start times. Starting with the earliest of the latest starts, the activity is scheduled to start if enough resources exist. If not, the activity is postponed. If, or when, all the activities on the waiting list are postponed, their new tentative start time (*e_start*) is set to the next time when there is a change in resource availability. The value of time is then set to the earliest *e_start* of these waiting activities, and the process repeats until all activities are scheduled, or an error occurs.

While there are no options within the SAS system such as stopping criteria or increased processing time that may help improve its solutions, there are user-input priority rules that may be changed (such as longest duration, most work remaining, or most slack). There are also some file size options, such as telling SAS how many predecessor constraints, activities, or resources there are. One may also specify to not use a utility table when memory is too full, or if the number of constraints exceeds the number specified (option NOUTIL). These options may have an effect on the final solution or processing time.

Although PROC CPM has the ability to solve a schedule (i.e., determine the start times of the tasks that would minimize project duration while enforcing precedence and resource constraints), PROC CPM was not used here to solve the schedule. The start times of the tasks were an input to the PROC CPM procedure, and the key output used from PROC CPM was the data set showing the number of resources used during every time period of each resource type.

RIGHT/LEFT-SHIFT REPAIR POLICY

The SAS code that was developed to implement a right/left-shift repair policy is described here. The source code is available from the authors upon request.

An initial baseline schedule was first constructed using PROC CPM to determine the start times of all tasks such that precedence and resource constraints were enforced and overall project duration was minimized. The tasks of the initial baseline schedule were sorted chronologically by their scheduled start times. A "real-time" scenario is simulated, such that when a task's start time is reached, the stochastic task will eventuate (occurs or does not occur). Any tasks before this time are assumed historical and only tasks ahead may shift to the left or right, with this point in time the minimum time that any future task may shift left to. When a task does not occur, the duration and resource utilization are set to zero. This logic does not require removal of precedence constraints for non-occurring tasks. For example, consider the precedence constraints for tasks: A → B → C. If A is scheduled to end at time period 10 and then we learn that B does not occur, the earliest time C can shift to is 10. Keeping or removing precedence constraints intact has no effect. Although precedence relationships do not need to be removed, the finish time of all predecessors will need to be updated every time a shifting round is complete before the next stochastic task is removed.

This process is run once for each stochastic task that has been identified. The steps to repair the modified baseline schedule are as follows:

A. Exterior loop "K". Goal = right-shift as necessary to accommodate expanding duration of eventuating task, if necessary. One loop for each stochastically occurring task.

   A.1. Sort the solved schedule by start time and select the first stochastic task, break ties using activity number lowest to highest

   A.2. Identify the initial start time of the stochastic task and refer to it as the "shift time" = t_s

   A.3. Determine if this buffered activity's eventuation (occurs or removed/not occur)

   A.4. If the task is to be removed, there is no right-shifting at this time, only left-shifting; skip to step B.

   A.5. If the task occurs, right shifting must occur to accommodate the additional time this task requires. All tasks that begin after t_s are moved to the right in the schedule. The same large number (currently using 100) is added to each tasks' start and end time.

B. Outer loop "J". Goal = left-shift tasks to ensure minimized overall project duration. One loop for each task that was shifted to the right in step A.5.

   B.1. Identify the first left-shifting task to be the task with the earliest start time after t_s that has not yet been left-shifted, breaking ties with a lower activity number. Set the new start time of this left-shifting task to be t_s or the max of the finish times for all its predecessor tasks, whichever is greater.

C. Inner loop "I". Goal = ensure no resource violations. As many loops as necessary until no resource violations exist.

   C.1. Check the feasibly of this temporary schedule using SAS® PROC CPM for resource constraint violations. If violations exist, increment the start time of this task up by one to the right and test again. Repeat incrementing to the right and testing until no resource violations occur.

   C.2. Repeat step C.1 until all tasks have been shifted to the left and no resource violations occur.

D. Return to step A, that is, identify the next stochastic task, set "shift time"(t_s) equal to it's start time, determine if the activity eventuates, and repeat the cycle.

E. Note that at every step along the way, a second data set containing each activity's list of predecessor start times is continually updated with the predecessors' constantly changing start times until the process is complete.

PROC CPM CODE

The code to run the procedure PROC CPM in step C.1 above is as follows:

```
001  proc cpm data = work.shift
002  out = work.shift_check
003  resourceout = work.rout_check
004  ressched = work.ressched_check
005  resin = work.resources_avail
006  ;
007  activity activity_num;
008  duration Pduration;
009  successor succ1-succ&num_succ.;
010  resource R1-R&num_R. /  period=date obstype=obstype;
011  actual / A_finish = s_finish; *force actual task end time to be s_finish;
```

Line 001 reads the data set "work.shift" into the procedure. This activity data set contains one record for each activity, while the columns contain activity-related information including the activity name, activity durations, successor relationships, resource requirements, and initial start and finish times. The variables that describe this information are defined in lines 007 – 011. Line 011 indicates to the procedure that the actual finish time of each activity is specified by the user as the value of the variable "s_finish". This allows the use of this procedure in a way such that it does not re-sequence activities, rather it only provides the output data sets containing the desired information. Line 005 also defines an input data set; specifically, this data set contains the number of available resources of each type.

Line 002 defines an output data set that contains the early, late, baseline, resource-constrained, and actual schedules. Line 003 defines the output data set for the each resource's usage. Specifically, the data set contains one record for each time period with columns that describe the total number of resources consumed and number of resources still available during this time period. A negative number of available resources in this data set indicates a resource constraint violation during the identified time period. Finally, line 004 defines the output data set for the schedules of each resource used by an activity.

EXAMPLE GANTT CHARTS

The following two Gantt charts demonstrate the beginning state and end result of SAS code implementation for repairing an example buffered schedule. The first Gantt chart (Figure 1) contains the initial baseline schedule. The sections of each Gantt chart represent the two resource types, with the vertical axes representing10 units available each, as well as a third section with no resource utilization. The tasks have been scheduled in the initial baseline schedule with 20% or 80% of their actual duration (if they occur).

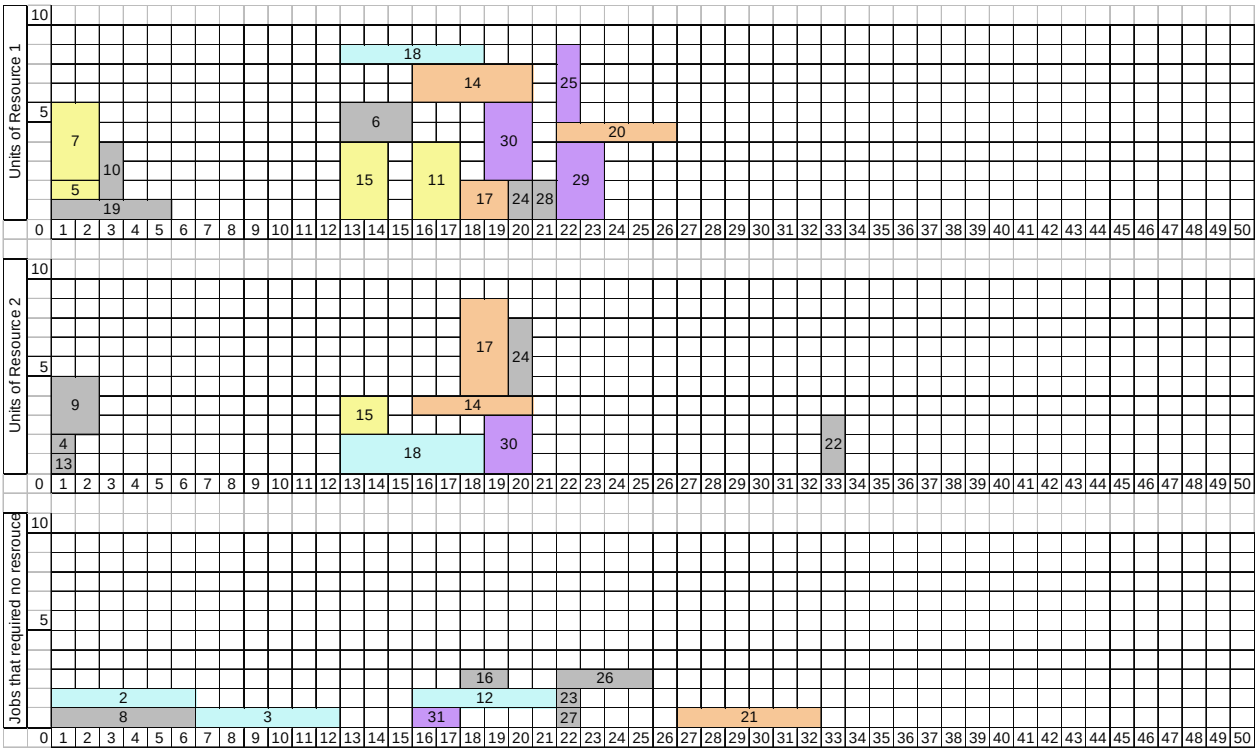

Figure 1: Initial Baseline Schedule

The following summarizes the eventuation of each stochastically occurring task. In the case of a task occurring, all tasks are shifted towards the right to make room for this "expanding" task and then shifted back to the left, one at a time. In the case of a task that does not occur, the task is removed and all tasks are shifted to the left, one at a time. When left shifting, a task is temporarily assigned a new start time that is equal to the current system time, or the latest finish time of its predecessors, which ever is later. If a resource violation occurs, the start time is incremented by one until it starts at a time when no resource violations occur.

- Task 2 – Occurs (80% buffered task expands to 100%)
- Task 5 – Does not occur (20% buffered task is removed)
- Task 7 – Does not occur (20% buffered task expands to 100%)
- Task 3 – Occurs (80% buffered task expands to 100%)
- Task 15 – Does not occur (20% buffered task is removed)
- Task 18 - Occurs (80% buffered task expands to 100%)
- Task 11 – Does not occur (20% buffered task is removed)
- Task 12 – Occurs (80% buffered task expands to 100%)
- Task 14 - Occurs (80% buffered task expands to 100%)
- Task 17 - Occurs (80% buffered task expands to 100%)
- Task 31 - Does not occur (20% buffered task is removed)
- Task 30 – Does not occur (20% buffered task is removed)
- Task 20 – Does not occur (80% buffered task is removed)
- Task 21 – Occurs (80% buffered task expands to 100%)
- Task 25 – Does not occur (20% buffered task is removed)
- Task 29 – Does not occur (20% buffered task is removed)

The final schedule (modified baseline) is shown below in figure 2.

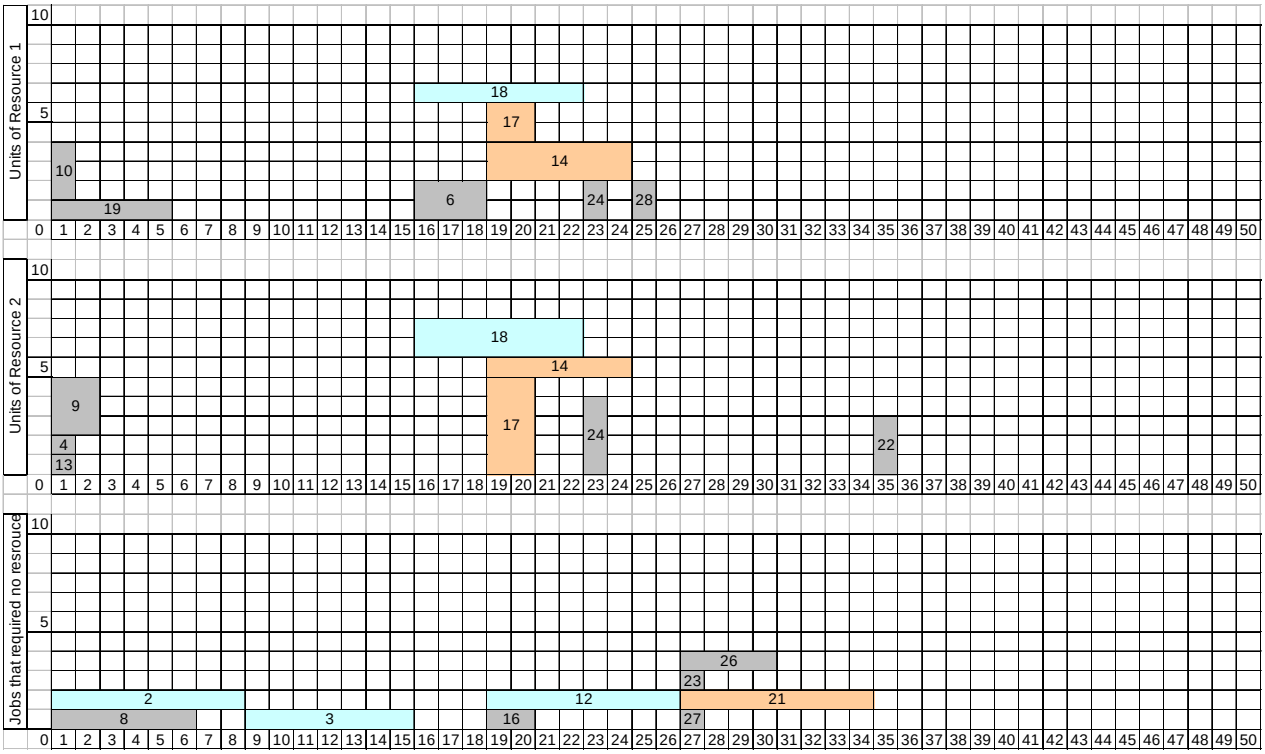

Figure 2: Modified Baseline Schedule with Right/Left-Shift Repair Heuristic Applied

The output 2-dimensional Gantt charts were used to verify the output to the right/left-shift repair and provided a visual documentation of the outcome.

RESULTS

Having the ability to implement an automated right/left-shit repair heuristic laid the foundation for the research of the author's dissertation.

Two new sets of resource metrics were constructed regarding resource utilization and resource flow. Using these new metrics, as well as the Selim/Grey metrics, a new buffering approach was developed that used resource information to size the buffers. The resource-sized buffers did not show to have significant improvement over Grey's 50% buffer used as a benchmark. The new resource metrics were used to validate that the 50% buffering strategy is superior to the 0% or 100% buffering by Selim.

Recognizing that partial buffers appear to be the most promising initial baseline development approach for STI problems, and understanding that experienced project managers may be able to predict stochastic probabilities based on prior projects, the next phase of the research developed a new set of buffering strategies where buffers are inserted that are proportional to the probability of occurrence. The results of this proportional buffering strategy were very positive, with the majority of the metrics (both robustness and resource), except for stability metrics, improved by using the proportional buffer.

Finally, it was recognized that all research thus far for the SRCPSP with STI focused solely on the development of predictive schedules. Therefore, the final phase of this research developed a new reactive strategy that tested three different rescheduling points during schedule eventuation when a complete rescheduling of the latter portion of the schedule would occur. The results of this new reactive technique indicate that rescheduling improves the schedule performance in only a few metrics under very specific network characteristics (those networks with the least restrictive parameters).

This research was conducted with extensive use of Base SAS v9.2 combined with SAS/OR procedures to solve project networks, solve resource flow problems, and implement reactive scheduling heuristics. Additionally, Base SAS code was paired with Visual Basic for Applications in Excel 2003 to implement an automated Gantt chart generator that provided visual inspection for validation of the repair heuristics.

The results of this research when combined with the results of Selim and Grey provide strong guidance for project managers regarding how to develop baseline predictive schedules and how to reschedule the project as stochastic tasks (e.g. unplanned work) do or do not occur. Specifically, the results and recommendations are provided in a summary tabular format that describes the recommended initial baseline development approach if a project manager has a good idea of the level and location of the stochasticity for the network, highlights two cases where rescheduling during schedule eventuation may be beneficial, and shows when buffering proportional to the probability of occurrence is recommended, or not recommended, or the cases where the evidence is inconclusive.

CONCLUSION

The power of Base SAS, paired with SAS/OR tools, provided the means by which to conduct a comprehensive study on various predictive and reactive scheduling techniques for the Stochastic Resource Constrained Project Scheduling Problem (SRCPSP) with Stochastic Task Insertion (STI) problem. An automated process was developed to repair a buffered schedule in “real time” as some tasks occurred and others did not. A right/left-shift repair heuristic was formalized and written using Base SAS code and SAS/OR® software’s CPM procedure. Base SAS was used to control the schedule data and shift tasks to the left/right as needed. Additionally, PROC CPM was used in an iterative fashion to check the repaired schedule for resource constraint violations.

In addition to providing the right/left-shift schedule repair heuristic with the means by which to ensure resource constrains were not violated, the multiple output data sets provided by SAS/OR procedures were very useful over the course of the research. For example, the output data set provided many valuable pieces of information regarding the schedules that were used in the development of new metrics to evaluate the effectiveness of newly developed predictive and reactive techniques. Specific to PROC CPM, the resource utilization data sets were stored for later use in computing two new metrics. The first metric compared the utilization of resources at every time period from an initial baseline schedule to the modified base schedule. The second metric 2 compared the total number of resource time unit (e.g., man-hours) for the initial baseline schedule to the modified base schedule.

This paper focuses on the experiment environment; however, this repair heuristic has the potential for real-world application by project managers who have the need to run “what-if” scenarios for project planning.

REFERENCES

Artigues, C., P. Michelon and S. Reusser (2003). "Insertion techniques for static and dynamic resource-constrained project scheduling." European Journal of Operational Research 149(2): 249.

Demeulemeester, E., M. Vanhoucke and W. Herroelen (2003). "RanGen: A Random Network Generator for Activity-on-the-Node Networks." Journal of Scheduling 6(1): 17.

Demeulemeester, E. L. and W. Herroelen (2002). Project scheduling : a research handbook Boston, Kluwer Academic Publishers.

Duron, C., J.-M. Proth and Y. Wardi (2001). Insertion of a random task in a schedule: A real-time approach, Antibes-Juan les Pins, Institute of Electrical and Electronics Engineers Inc. (Conference Proceedings).

Goldratt, E. (1997). Critical Chain Great Barrington, The North River Press Publishing Corporation.

Grey, J. (2007). Buffer Techniques for Stochastic Resource Constrained Project Scheduling with Stochastic Task Insertions Problems. Florida, United States, University of Central Florida. Ph.D. Dissertation.

Herroelen, W. and R. Leus (2004b). "Robust and reactive project scheduling: a review and classification of procedures." International Journal of Production Research 42(8): 1599.

Leon, V. J., S. D. Wu and R. H. Storer (1994). "Robustness measures and robust scheduling for job shops." IIE Transactions 26(5): 32.

Ourari, S. and B. Bouzouia (2003). "An approach based on operation insertion for one-machine real-time scheduling." International Journal of Robotics and Automation 18(4): 185-190.

Sadeh, N., S. Otsuka and R. Schedlback (1993). Predictive and reactive scheduling with the microboss production scheduling and control system. In: Proceedings of the IJCAI-93 Workshop on Knowledge-Based Production Planning, Scheduling and Control, pp. 293-306.

SAS Institute (2006). SAS/OR 9.1.3 user's guide : project management 2.1. Cary, NC, SAS Institute: 3 v. (xviii, 942 p.).

Selim, B. R. (2002). Robustness measures for stochastic resource constrained project scheduling. Florida, United States University of Central Florida. Ph.D. Dissertation.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Dr. Sandra Archer
Director, University Analysis and Planning Support
University of Central Florida
12424 Research Parkway, Suite 215
Orlando, FL 32826-3207
407-882-0287
archer@mail.ucf.edu
http://uaps.ucf.edu

Dr. Robert L. Armacost
Higher Education Assessment and Planning Technologies
P.O. Box 2237
Flagler Beach, FL 32136
386-439-7486
armacost@mail.ucf.edu

Dr. Julia Pet-Armacost
Associate Dean for Planning and Knowledge Management
College of Medicine
University of Central Florida
(407) 882-0276
jpetarma@mail.ucf.edu