A Post-Process Optimization Algorithm for Resource Constrained Project Scheduling

Alex Aldas

The Boeing Company
13100 Space Center Blvd.
Houston, Texas 77059
Alex.Aldas@boeing.com

Abstract

As modern engineering projects grow more complex in nature, proper usage of available resources has become increasingly important. Inefficiencies in a scheduling engine could leave a project overdue and over budget. As a result, schedule density and schedule makespan have been drawn to the forefront of the planning and scheduling community. This paper does not try to solve the overall scheduling problem of maximizing resource usage, but rather explores the use of a post-process optimization scheme that attempts to shorten project makespan and increase resource usage through the use of slack distribution.

Introduction

In recent years the National Aeronautics and Space Administration (NASA) has undergone a culture change within its operations to one of "faster, better, cheaper." This philosophy has propagated into every facet of NASA, but arguably none so much as the planning and scheduling community.

The production of large, complex, one-of-a-kind products is costly in-of-itself. Unnecessarily long schedules that do not fully utilize available resources can have a dramatic effect on the overall cost of a given project. A reduction in project makespan by as little as one day has a potential cost savings in the tens to hundreds of thousands of dollars. Therefore, the goal of the scheduler in assembly operations would be one of minimizing project makespan without lowering the quality of the schedule.

Mission planning and scheduling provides a different flavor of the resource constrained project scheduling problem (RCPSP). Here no product is assembled, nor is there a defined network of activity. Rather, mission planning involves the scheduling of individual or groups of activities, which may contain temporal and/or resource constraints. Effective scheduling would try to maximize the number of high priority activities that can be performed during the given mission time frame, while placing lower priority activities in the timeline gaps that remain.

Whether the problem involves spacecraft vehicle assembly or space station mission planning, scheduling has a direct impact on the cost and efficiency of the project. This paper does not attempt to explain the methods used to schedule such projects, but rather it presents a post-process *schedule packing*[1] algorithm which strives to further condense *any* feasible schedule. By packing a schedule, excess slack is redistributed and later extracted. The final schedule has a decreased cycle time and resource idle time, which corresponds to project cost savings.

Resource Constrained Project Scheduling

A resource constrained project scheduling problem is defined by a set of tasks, which operate on a set of finite capacity resources. Each task can contain any number and variety of constraints imposed. The goal of such a problem is to define a schedule in which all constraints are satisfied while meeting some objective function, usually to minimize project makespan. Any schedule which satisfies all the constraints is labeled as *feasible*, while any feasible schedule whose makespan is as short as the shortest known feasible schedule is labeled as *optimal*.

Schedule Packing

Schedule packing is a *post-process* optimization scheme meaning that it attempts to further optimize an existing feasible schedule with respect to assignment density. As such, the algorithm is independent of the method used to generate the feasible schedule. While it does not perform scheduling itself, it *drives* a scheduler to select a task, unschedule it, and re-schedule it on the timeline. Schedule packing can be performed on any feasible schedule so long as the scheduler has the capability of "locking" a schedule and selectively unscheduling and re-scheduling individual tasks. It is therefore the responsibility of the scheduler to insure that all constraints are satisfied in the re-scheduling process.

The easiest way to explain schedule packing is by example. For simplicity, let us take the most trivial of all

[1] The Boeing Company has applied for a patent on this algorithm.

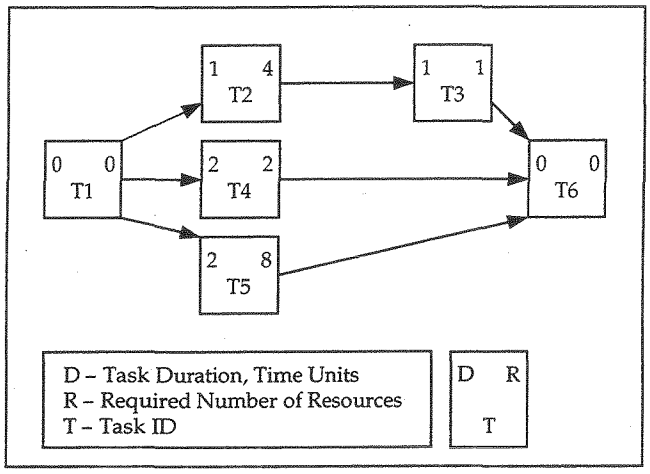

Figure 1: Example project in a precedence diagram.

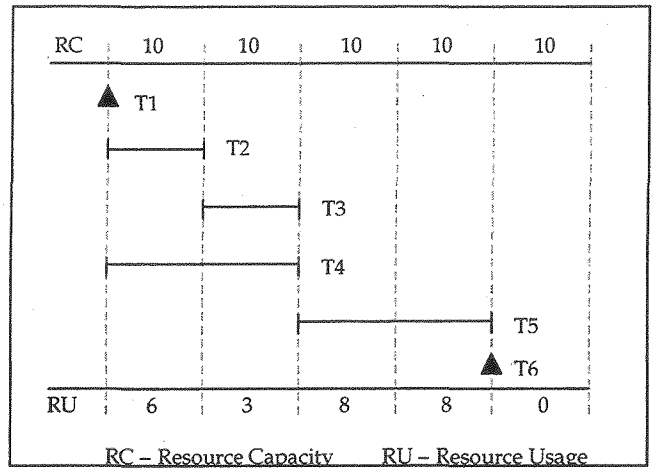

Figure 2: Initial feasible schedule

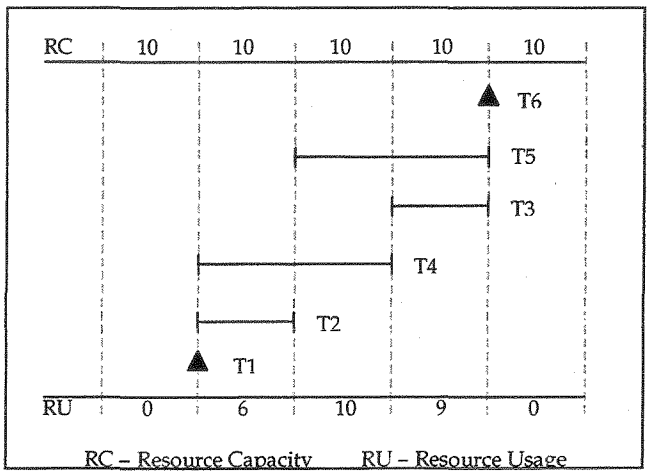

Figure 3: Right shifted schedule

cases: single resource, with constant initial availability over time. For this example, we will define a resource R with a constant capacity of 10. Our project, depicted in Figure 1, consists of 6 tasks with tasks T1 and T6 being milestones. Each task has associated with it a duration, in time units, a required number of resources R, and a task ID. Performing a sort by name, followed by a sort by predecessors on the task list and scheduling each task in the first available interval (left-most feasible interval) results in the feasible schedule illustrated in Figure 2.

Our initial schedule has a makespan of 4 time units. If we were to track our resources, i.e., not enforce resource constraints, we could achieve a schedule makespan of 2 time units. Hence, the added length to the project duration is due to resource constraints rather than technological (precedence) constraints. To shorten the makespan of this schedule we must be able to resolve critical resource constraints within the schedule.

Schedule packing begins with our initial feasible schedule. The algorithm proceeds as follows:

- Select all tasks.
- Sort by latest scheduled end time (secondary sort by latest start time in case of a tie).
- Starting from the top of the task list, unschedule each task and reschedule it into the right most feasible interval.

The results of the above process on our initial feasible schedule are illustrated in Figure 3. What we have done is utilize the slack that exists within our initial schedule to "pack" tasks up against the right hand fence, i.e., the project end time. Through inspection of Figure 3, the actual project duration has been shortened by 1 time unit. Shifting the entire project back to the left by 1 time unit results in a new feasible schedule with a makespan of 3 time units.

The next logical question is can this new schedule be packed further? A good indication of whether a schedule has the potential of being packed is to examine the slack associated with the beginning task(s) of a project. If any of the beginning tasks has a slack time greater than zero, then the project can potentially be packed. Our initial schedule in Figure 2 indicates that task T4, a beginning task since T1 is a milestone, has a slack time of 1 time unit. This indicates that our initial schedule has the potential of being packed. After performing schedule packing on our initial schedule, we find that all of the beginning tasks, T2, T3, and T5 have a slack time of zero. Since all of the beginning tasks have a slack time of zero, the resulting schedule has probably been packed as much as possible.

Schedule Packing with Real World Projects

Real world projects provide a much more complex problem than the example project described above. In the example problem we assumed a single resource with a constant capacity throughout all of time. In real world projects, however, such simplifications are rarely found. In modern

aircraft assembly, the scheduling of a single sub-assembly can have as many as 1,164 tasks scheduled against 226 resources. In addition, the availability of skilled labor is usually a function of shift and contract work.

Our scheme as described above is inadequate for packing complex projects. The primary reason for this short-coming is because we cannot simply pack the schedule against the right-hand fence and then slide the whole schedule back towards the left-hand fence. This would only be possible if all resources used within the project have a constant initial availability. Since most real world projects do not have this property, we must extend the schedule packing scheme to handle the generic case of variable resource availability.

By sorting and rescheduling against the right-hand fence we attempt to distribute slack among tasks so that resource conflicts can be broken. The result is a "packed" schedule crammed against the right-hand fence. If we perform the exact same operation but in the reverse direction, i.e., sort by earliest scheduled start time and schedule into the left most feasible interval, we are effectively "sifting" tasks into "slack holes" left by the shift right operation. Further "sifting" will result in a steady-state schedule which is packed in both directions. Further shifting will produce the same initial schedule.

To summarize the generalized schedule packing algorithm:

- Select all tasks.
- Sort by latest scheduled end time (secondary sort by latest start time in case of a tie).
- Unschedule and re-schedule each task into the right most feasible interval.
- Sort by earliest scheduled start time (secondary sort by earliest finish time in case of a tie).
- Unschedule and re-schedule each task into the left most feasible interval.

The generalized schedule packing scheme is independent of project structure, resource availability, and resource requirements. It can be used on any feasible schedule so long as the underlining scheduler is capable of unscheduling and rescheduling a single task into left-most and right-most feasible intervals.

Benefits of Schedule Packing

The degree to which schedule packing can reduce project makespan is a function of several variables including: 1) quality of the initial schedule and 2) variability in resource availability.

Initial schedules that are near optimal contain a limited amount of slack for which schedule packing can be utilized. Since slack is what the algorithm uses to pack tasks against each of the project fence dates, a schedule with a large amount of slack is more apt to reductions in schedule makespan than a highly optimized initial schedule.

	Makespan, hh:mm	*% Compression*
Benchmark Test 2		
SFI (Baseline)	685h:16m	-
SFI + SP	591h:54m	14.62
Benchmark Test 3		
SFI (Baseline)	861h:25m	-
SFI + SP	696h:25m	21.18
Benchmark Test 4		
SFI (Baseline)	845h:40m	-
SFI + SP	654h:8m	25.54

* Note: Simple Feasible Interval (SFI) scheduler used with a sort by predecessor dispatch order. SP = Schedule Packing

Table 1: Compression analysis for benchmark tests

Schedules that contain a high degree of variability in resource availability usually result in projects with numerous resource conflicts. While schedule packing can eliminate some of these conflicts, its performance is limited.

To examine the degree of compression that schedule packing provides, a set of benchmark tests[2] that emulate real-world assembly projects was chosen. The analysis consisted of tests 2, 3, and 4 within the 12 test data set. Each test contained 575 tasks to be scheduled against 17 resources using simple resource modeling – single mode with precedence, labor and zone constraints. Results from the benchmark tests are presented above in Table 1.

While this data set is a fair example of an assembly process, in practice we have found that the constraints modeled are much more complex than those found in the benchmark tests. Additionally, resource availability is not as "cleanly" defined. Modern assembly processes may move between two and three shift operations and may pull additional resources based upon contract needs. As a result, resource availability histograms appear more like skylines than simple step functions.

To test schedule packing in real world projects, an aircraft sub-assembly with a degree of constraint modeling and resource utilization typical of modern complex assembly operations was chosen. The "live" data set included 169 tasks to be scheduled against 42 resources. Resource requirements were modeled in a multi-modal form and included advanced modeling techniques such as interruptibility, grouping, linking, and timeline exclusivity. The results of the test are displayed in Table 2.

	Makespan, hh:mm	*% Compression*
Baseline Schedule	115h:45m	-
Baseline Schedule + SP	98h:9m	15.2

* Note: Schedule generated by TimePiece©. SP = Schedule Packing.

Table 2: Schedule packing results on aircraft sub-assembly scheduling.

[2] Benchmark tests can be found at http://www.neosoft.com/~benchmrx

From experience, schedule packing can compress a "good" initial schedule by 15-20%. The resulting schedule has a higher resource utilization, which usually corresponds to a decrease in schedule makespan. While schedule packing was described in the context of assembly operations, it has been successfully used within the planning and scheduling operations of facility resource management[3] and on-orbit operations planning[4] with similar results.

Conclusion

Schedule packing is a post-process optimization algorithm that drives a scheduling engine to compress any existing feasible schedule so long as the scheduling engine is capable of enforcing all task constraints. It has been shown in practice to reduce the schedule makespan of "good" initial schedules by 15-20%. In a time where proper resource management is of high priority, schedule packing provides a simple schedule improvement scheme that many of today's schedulers can employ to obtain a better resource utilization rating.

References

Wiest, J. D. 1964. Some Properties of Schedules for Large Projects with Limited Resources. *Operations Research.* 12(3):395-418.

---

[3] F-18 Avionics Integration Lab, The Boeing Company (FRMS©)

[4] SpaceHab Operations (COMPASS©)