International Journal of
Information Systems and Project Management
ISSN (print):2182-7796, ISSN (online):2182-7788, ISSN (cd-rom):2182-780X
Available online at www.sciencesphere.org/ijispm

Resource allocation in IT projects: using schedule optimization

Michael A. Chilton
Kansas State University
Manhattan, KS 66506
USA
www.shortbio.net/mchilton@ksu.edu

Abstract:
Resource allocation is the process of assigning resources to tasks throughout the life of a project. Despite sophisticated software packages devoted to keeping track of tasks, resources and resource assignments, it is often the case that project managers find some resources over-allocated and therefore unable to complete the assigned work in the allotted amount of time. Most scheduling software has provisions for leveling resources, but the techniques for doing so simply add time to the schedule and may cause delays in tasks that are critical to the project in meeting deadlines. This paper presents a software application that ensures that resources are properly balanced at the beginning of the project and eliminates the situation in which resources become over-allocated. It can be used in a multi-project environment and reused throughout the project as tasks, resource assignments and availability, and the project scope change. The application utilizes the bounded enumeration technique to formulate an optimal schedule for which both the task sequence and resource availability are taken into account. It is run on a database server to reduce the running time and make it a viable application for practitioners.

Keywords:
project management; optimization; resource allocation; bounded enumeration.

DOI: 10.12821/ijispm020303

Manuscript received: 9 April 2014
Manuscript accepted: 15 July 2014

1. Introduction

This paper discusses allocation of renewable resources within Information Technology (IT) software development projects. While the overall process of IT development is similar to managing projects in any other industry, it has some distinct differences. First, IT projects rely almost exclusively on renewable human resources. As personnel are assigned tasks, they complete them in a particular sequence and pass the project on to the next person in the chain. Once freed, the worker then begins work on the next project. Second, IT projects are considered intellectual work in which information about the project must be shared with co-workers, and so adding more workers to speed things up usually does not help, but may make things worse [1]. Third, the direct costs of an IT project are almost exclusively due to labor costs [2]. Some IT applications may require hardware and software acquisitions; however, these are generally not considered direct variable costs, because they are considered part of the firm's IT infrastructure and can be reused for other projects. Additional characteristics of IT projects include project environments in which multiple projects are concurrently active and that many projects are given pre-planned deadlines (i.e., they are time boxed). Time boxing causes project managers to carefully evaluate the project scope in order to meet pre-planned deadlines, while the nature of the work and prior experience may fix the number of personnel assigned to any particular task.

These factors form the basis of this analysis of IT project scheduling and resource allocation methods in this paper. We therefore focus on fixed sets of human resources with different skill sets forming separate labor pools who are attempting to complete a project in the minimum amount of time (minimized make span). Our analysis includes multiple projects that may be competing for the same sets of human resources.

The first step in creating a work plan for a new IT project is to identify and schedule the set of tasks necessary to complete the project. The resulting schedule must observe all of the technical constraints imposed by the project (i.e., the sequence of tasks) and the applied constraints imposed by the number of available resources. In the planning stages, project schedulers often estimate task duration based on the number of resources available and therefore over allocation of resources for a single project is not expected. Once the project gets underway, however, resources can easily become over-allocated due to a number of different factors, such as changes in personnel, lack of availability of personnel due to vacations, sick leaves or changes of employment or because a number of projects may be competing for the same set of resources. The result is that resources assigned to certain tasks may not be able to complete them within the planned schedule and this situation calls for immediate action on the part of the project manager. To be competitive, an organization must attempt to schedule all tasks so that the project is completed in the minimum time.

Scheduling IT projects proceeds as just described and includes identifying tasks, estimating their duration and placing them on a timeline (Gantt chart) that visually displays their sequence, duration, and start and end date. Resources are then assigned to each task from available labor pools, which are separated by skill set. IT projects resemble assembly lines in that different pools of resources perform tasks at specific times during the project. That is, business analysts generally are assigned to the project near the beginning and they build a business case for the application. Next, the systems analysts begin to gather requirements and formulate models of how the software will work. This goes into the design specifications that are passed to programmers who implement the application. The completed application is then sent to quality engineers who provide alpha testing to clear up any bugs and ensure that the application meets the users' requirements. This may be followed by beta testing with a sample of users not involved in the application's development to further refine the program and identify bugs. Once any identified bugs are removed, the completed application is sent to those who are responsible for its deployment and training of users. The duration of the tasks contained in each of these stages is dependent upon what must be done, the complexity of the tasks and the number of resources available at the time. While this may seem like a straightforward process, problems with the planned schedule can occur when multiple projects are being executed at the same time. Problems can also occur when changes are made to the project itself or to the labor pool to which a set of tasks is assigned. Any of these situations can cause delays, which can in turn affect the cost of one or several projects.

Delays can also be caused by the over-allocation of resources. Delays in one project may affect other projects that are being executed concurrently, causing a slowdown in production and fewer projects meeting their deadlines. When

resources become over-allocated, the project managers must devote time to reallocating tasks to additional personnel or to different time periods. Although this can sometimes be done automatically in sophisticated project management software, the process is not particularly straightforward, it consumes a lot of time and the result is usually less than satisfactory. A solution to this problem is to schedule the tasks in all active projects while taking into account the maximum number of resources available. Doing so would prevent over-allocating any resource. We take this approach in this study by introducing a software application that takes the production schedule as input and produces an optimal schedule as output. The software is programmed to recognize the restrictions imposed by the sequence of tasks and by the number of resources from one of several resource pools. Because of this, the resulting schedule yields no over-allocated resources. The application can be used in a multi-project environment and can be run throughout the lifetime of any project, thus providing more current estimates to the project manager.

We have found that over-allocation of resources is a common problem within firms engaged in IT development. In fact, this study was prompted by representatives from a large petro-chemical company who were having difficulty with resource allocation. Project managers in the company often found that they had IT developers who were often over scheduled even though a sophisticated commercial project management program was used to plan, assign and track activities in a large number of concurrent projects. The company asked us to assist them with solving the problem, and so we turned our attention to creating the program that we introduce in this paper.

This paper is organized as follows. The first section is devoted to a detailed explanation of the over-allocation problem and a survey of the literature relevant to it. The next section discusses the software artifact that we produced, including the logic upon which it is based and the results of testing it against a number of artificially generated project networks. The last section outlines the remaining work to be done and provides a description of the experimental design, the variables to be measured, the metrics to be used and the definitions of what would be considered successful findings.

2. Overview and literature review

Scheduling tasks in a project is a complex and time consuming endeavor and has been studied and improved since the Critical Path Method (CPM) and Program Evaluation and Review Technique (PERT) methods were introduced in the 1950s. CPM is known to suffer from its inability to deal with the problem of limited resources [3]. PERT was designed to take into account some of the uncertainty in estimating the duration of tasks that comprise a project [4], by including a set durations for each task based on an optimistic, expected or pessimistic estimate and computing a weighted average to act as the best estimate. CPM and PERT are applied to one project at a time and neither specifically address the restrictions imposed by limited resources [5]. It is doubtful that they were ever intended to be used in a multi-project environment in which different but concurrent projects would compete for the same set of resources.

While the original project management techniques were designed to help better manage large and complex projects, it quickly became apparent that improvements were needed in order to help reduce costs and shorten the time to completion. In addition, because resources are limited in any project endeavor, this constraint needed to be added to the technique. A significant amount of research was accomplished in the ensuing decades, devoted to minimizing the duration of the project and accounting for the limitations imposed by limited resources. Wiest [6] pointed out that although the critical path may represent the longest sequence of tasks in a project, the idea of it being critical becomes meaningless when resources are limited, because any task may be delayed due to the lack of resources. He identified a critical sequence to account for both the required sequencing of tasks and the constraint of limited resources, but warned that the linear programming techniques (at the time) were infeasible for practically sized projects due to their computational requirements. Computational efficiency was analyzed and reported by Davis [7], who discussed the problems encountered when attempting a mathematical formulation of the resource constrained scheduling problem. Herreoelen, De Reyck and Demeulemeester [8], followed in this vein by discussing the computational efficiency of several optimization techniques developed since Davis' [7] analysis.

Gutjahr and Nemhauser [9] also addressed the computational difficulty of the class of scheduling problems by stating that because of the large number of possible solutions, a complete enumeration of them is impractical. They presented a

more efficient technique that eliminates some solutions during the computation because they violate one or more of the constraints imposed. An exact solution can then be determined much more quickly and efficiently than other techniques. Their method was applied to the resource constrained scheduling problem by Davis and Heidorn [10], which we discuss later.

The difficulty of the scheduling problem seems to have challenged researchers into searching for solutions. For example, Moodie and Manville [11] presented an integer linear programming (ILP) technique aimed at solving the assembly line balancing problem, which is similar to finding an optimal schedule for a project. Held and Karp [12] presented a dynamic programming solution to scheduling problems and demonstrate the technique by applying it to both the traveling salesman and the line balancing problems. Dynamic programming has been applied to this problem by numerous researchers since then [cf. e.g., 13, 14]. Goldratt [15] introduced his theory of constraints and the critical chain as a way of viewing, managing and scheduling projects.

Robinson [16] published an algorithm that determines the cost-time function in order to find the project's minimum duration resulting from the optimal allocation of resources to each activity. Implicit in his study is that the number of resources can be adjusted to shorten the duration of any particular task. In software development we find this generally not to be the case because an organization assigns resources (from limited resource pools) to tasks based on the need for specific skills to accomplish the task and because adding more resources to a project tends to slow it down [1]. Adding resources to shorten a project may work well when the work does not involve creative activity or communication among workers; however, in software development, adding more workers is, in Brooks' words, "Like dousing a fire with gasoline…" [1, p. 14].

When limited resources are taken into account the problem is labeled as the Resource Constrained Project Scheduling Problem (RCPSP). This type of scheduling problem is known to be NP hard [17], because the solution space grows much faster than the problem space. It can therefore quickly become intractable for practically sized projects due to the amount of computation required, which is manifested as computing time. Much work has been devoted to this problem, which has generally been addressed either through heuristics or exact solutions [18] and aimed at scheduling around the limitations imposed by scarce resources. Heuristic solutions tend to be favored because they are fast and provide reasonable results, while exact solutions, such as linear and dynamic programming techniques require too much time for practically sized problems. Davis and Patterson [19] compared the results of eight widely used heuristics to an optimum solution produced through the bounded enumeration technique found in [10]. Their results showed a wide variance in the ability of each of the 8 heuristic techniques to produce schedules that were close to optimal, with the average percentage increase above optimal ranging from 5.6% to 16%. Of the 83 test projects used in the analysis, the ability of heuristics to find an optimal schedule also varied widely and ranged from 1 to 24 test cases [19].

Recent work focusing on heuristics has employed such techniques as genetic algorithms and artificial intelligence [cf. e.g., 20] and hybrid techniques [21]. Studies have also analyzed multi-mode problems [22] and have applied meta-heuristics to the multi-mode problem [23]. Additional work has been done recently on exact approaches as well, including mixed integer programming [24] and dynamic programming [25].

Bandelloni and colleagues [26] provided a definitional distinction between resource allocation and resource leveling. They said that resource allocation applies to the case when resources are limited and the scheduling objective is to "keep the project completion time as close as possible to the critical path length such that the resource constraints are met" [26, p. 162]. They further classified resource leveling as a process in which there are no resource limits and the consumption of resources can be controlled to follow a desirable shape. The focus of this study will be on the allocation of resources in a software development project with the goal of scheduling resources to perform specific tasks and ensuring that they are not over-allocated, that is, they are not scheduled to perform so many tasks that they cannot perform them in the time required.

Despite the distinction provided by [26], we should note that the terms resource leveling and resource allocation are often used interchangeably. For example, most project management software programs have options for "resource leveling." This function attempts to solve the allocation problem and can be invoked when the project manager

discovers that a resource is over-allocated. In addition, [27] compared the resource allocation capabilities of seven commercial project management programs, but his analysis seems to address a resource leveling function, because he asked the programs to optimally allocate resources while at the same time find the minimal makespan for a project, suggesting a causal link between the two. In any case, the problem we study here is intended to provide an optimal schedule in the planning phase that incorporates the resource limitations and by doing so, will encounter no over-allocation problems throughout the life of the (unchanging) project. We will also discuss the effects of various types of changes on the project schedule.

When a project manager is faced with an over-allocated resource, he can invoke the resource leveling option in his software. Doing so generally causes one or both of two things to occur: some tasks are reassigned to others who are under-allocated and/or some tasks are delayed until enough resources become available to complete the task(s) affected by the over-allocation. How this is done by the software is not particularly straightforward and each package may use its own proprietary method to do so [27]. The results can be difficult to understand and even more difficult to manage, as it may require a larger time investment by the project manager into activities that are not a part of the project. It may also require that some tasks be split by interrupting work on one task to make resources available for others. It is sometimes recommended that leveling of over-allocated resources be done individually and manually [28].

Our stated goal in this paper is to introduce a software program that produces an optimal schedule that accounts for both the technical and the resource constraints of a project in order to prevent over-allocation of resources. A schedule produced at the beginning of a project that conforms to this definition will have no over-allocated resources; however, the conditions under which software development projects are conducted may change and this may result in resource allocation problems. There are three factors that may affect the allocation of resources during the life of a project. The first is an increase in scope of the project that will necessitate more effort to complete it. This is not an uncommon occurrence in software development, and has been listed as the 14th most often reason for IT project failure [29]. It has also been found to occur in 23% of all IT projects [29]. Secondly, resources may become unavailable during periods in which they were originally thought to be available. This may occur for a variety of reasons (e.g., vacation, sickness, or the loss of people to other jobs). Finally, several projects that compete for the same resources may be executed concurrently. Managing multiple simultaneous or overlapping projects can increase the level of difficulty enormously over the single project case, because management must attempt to balance the resources [30]. This can adversely affect competitiveness since a bidding process must take into account those resources already committed when preparing a bid on a new project, which may be executed concurrently with others.

There have been numerous heuristics developed and used because of the low computational expense, and several exact solutions exist but are not often used in practice for the opposite reason. One exact method that shows promise and is used here, is the bounded enumeration method of [10]. This method takes a set of fixed-duration tasks, which must be completed in a specific sequence and which require a fixed number of resources from one or more resource pools. The first step in the algorithm is to divide them into unit duration tasks. Thus, if the unit duration was one day, a five day task would be divided into 5 one-day tasks. The schedule can then be displayed in a network diagram or as a Gantt chart, either of which makes the technical sequence easy to see. The diagram or chart can be annotated with the number of resources needed for each task from each resource pool. Each resource pool is assumed to contain a maximum number of resources that cannot be exceeded.

The method used in [10] begins evaluating the sequence of activities by entering the first task into the algorithm and determining those sets of tasks that could follow based upon the restriction imposed by the technical sequence. These are termed feasible subsets, but they do not include resource constraints. This process continues so that a set of feasible subsets is created for each stage in the project. The number of stages initially equals the length of the critical path, but may be extended as the program processes the inputs. The next step is to eliminate from consideration any feasible subset that cannot meet the resource constraints. This leaves a set of paths through the project network that will meet all technical and resource constraints. The final step is to determine which path is the shortest, i.e., has the least number of time periods (steps) and therefore conforms to the definition of minimum make span.

3. The software artifact

We created a software application that employs the bounded enumeration technique just described. Its performance is exceptionally fast because it makes use of the speed and power of a database server to reduce the computing time required. To be useful to practitioners, it must be able to find a solution in a reasonable and practical amount of time. We now provide the details of its implementation and in the next section provide a tabulation of the results of its performance.

A predetermined schedule is used as input. This schedule can be taken from a network diagram and it must include the required sequence of activities (the technical constraint), the duration of each task and the number of resources required for each task from each resource pool. A production schedule produced in a commercially available project management software program (e.g., Microsoft Project, Primavera, or SAP's PS module) or a custom scheduling program can provide this input. This data must be treated by an interface, which converts it into a format that can be stored within a relational database. The database consists of 6 separate but related tables. As part of the processing, the actual ask descriptions in the production schedule are replaced with a 5 character code before being entered into the main task table. The entity relationship diagram for the database is shown in Fig. 1.

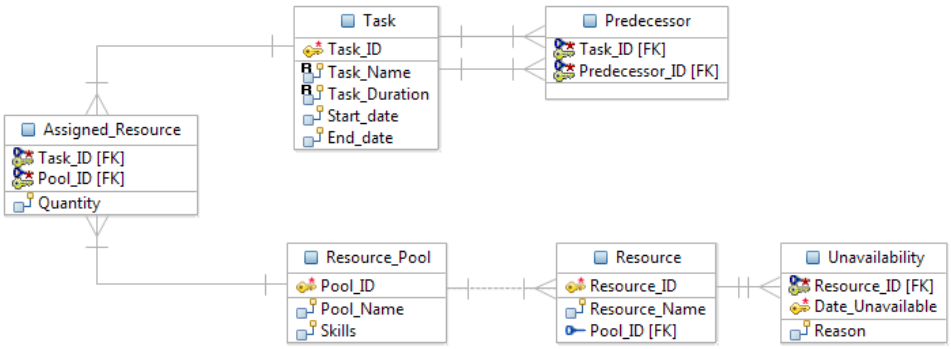

Fig. 1. Relational Schema

The Task table holds data relating to each task in the project and is related to the Predecessor table, which identifies the predecessor of each task. A simple SQL query of these two tables will yield the technical sequence of the project. Resources are recorded in the Resource table by name and are assigned to a specific labor pool. Because the details of each pool are recorded in a separate table (the Resource Pool table), there is no limit to the number of pools that can be included. This can be particularly advantageous because it can be used to separate beginning workers from those with much greater experience and therefore provide a better assessment of the duration of a task. Although both start and end dates are recorded in the Task table, these entries are generally not used in the program because these dates will be determined after the program runs. There are two exceptions. First, the initial start date for the first task is used as a reference from which all other dates can be calculated using the task durations. Second, any tasks that have been completed can be eliminated from the algorithm and a new reference start date can be used. This allows the program to be run in the middle of a project and ignore those tasks that have been completed and will no longer affect its outcome.

There are several temporary tables used to store the data as it is processed by the program that are not shown in Fig. 1. They are described below and are depicted in Fig. 2. The purpose of these tables is to hold data that is obtained by joining the related tables to produce a single storage area for each process used in the program. As each process in the algorithm is executed, the resulting data is reformatted and moved into a new table. While the structure of these tables is permanent, the data contained within them is transient, meaning that it is kept only for the life of the project. As projects

are executed and completed, tasks which have been completed are removed; as new projects are added, their tasks are then stored in the tables. The data from old projects can then be archived for future analysis or simply deleted.

Production data is pre-processed by an interface, which accepts the production schedule as input, encodes the task names as 5 character codes, separates each task into single time-unit tasks and formats it for entry into the six tables in the database. This process is shown in figure 2 as part of the interface. Once formatted, the database server then queries the data within these six tables and processes it as described below. The result is an optimal schedule that can be returned to the production system via an interface that will re-format the output as needed for the production system. Obtaining the optimal schedule is performed as follows:

1) Create a matrix of tasks, their predecessors, duration and the number of resources required for each task from any number of resource pools. This matrix is stored in the first temporary table. This process is very quick. For a 25 task project, it usually takes about 30 to 60 seconds (see table 1).

2) Query the first temporary table and create a list of the project tasks in the order they must be performed (the technical sequence) and store the result in the second temporary table. This list will show a set of tasks that can be performed in one period and all of the tasks that could possibly follow these tasks (called feasible subsets) in the next period in each row of the table. In addition to recording these sets of tasks, the program also records the sum of the resources required to perform them from each resource pool. This table can grow to be quite large even for small projects. As an example, for a project with 25 independent tasks, we found that the program had created over 22,000 entries in this table.

3) Query the second table and create an adjacency matrix ("A" network) and store the results in a third temporary table. This table stores sets of tasks that are compatible, that is, one set can follow another based upon the technical sequence and the resource limitations imposed by each pool. Thus, many of the subsets from the second table are eliminated and this table is generally much smaller. We have found that this process consumes the most amount of time. For the 25 task sample project mentioned previously, the program ran in about 13 to 15 minutes (see table 1).

4) Analyze the third temporary table to determine the optimal path. The optimal path is defined as the one with the fewest number of steps to completion; however, the program also analyzes the cost of each step by summing the number of resources multiplied by number of activities required for each activity within the step. The optimal path is stored in the fourth temporary table, but is not easily interpreted because it accumulates activities as the project progresses. This process ran much quicker than predicted, which was usually less than 10 seconds (see table 1).

The result obtained in step 4) above is sent back to the interface for further processing before it can either be viewed by the project manager and/or returned to the production system to update the production schedule. Ideally, the PM will view this output and determine its suitability before updating the production schedule. Within the interface, the duplicated tasks are removed, all single time-unit tasks are re-combined, the task codes are replaced with their original descriptions and named resources are assigned, and finally, in a multi-project environment, different projects are separated into independent schedules and these optimal schedules are sent back to the production system to replace/update the original schedule(s). We should note that the interface is currently not a part of the program as it is now written, but would be added later once the details of the formatting for the production system are known.

Fig. 2 shows the flow of processing and encapsulates each stage within a rectangle. The production system, which holds the original schedule is shown on the left, the interface is depicted in the middle and the optimizing program is shown on the right of the diagram and is labelled as the Processor. Steps 1 through 4 as described above are labeled in the figure as Process 1 through 4. The data store labeled, "Project Data," is the data contained in the six related tables shown in Fig. 1. The temporary tables are those data stores in the figure shown below each process.

As was mentioned, the Interface portion has not been implemented, so the performance testing of the program was performed by inputting data manually into the database. The program was run in four stages, which correspond to the four steps described previously, and run times were recorded. The output was then checked for accuracy and in some cases, was re-entered back into simple Gantt charts for comparison against the original schedule. We next present the results of our testing.

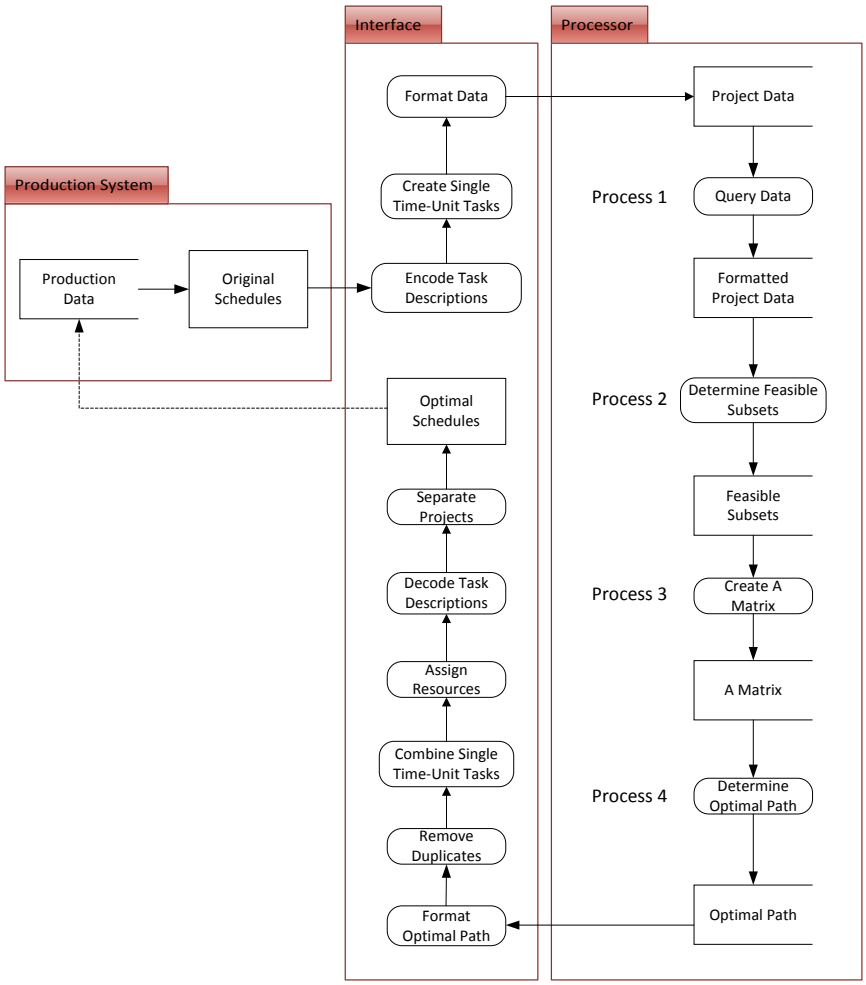

Fig. 2. Data flow to produce an optimal schedule

4. Results

As is well known, the number of possible paths through a network increases rapidly with the size of the network. This fact renders the problem NP hard and it has a major effect on the amount of time required to solve it for an optimal path. Therefore, the next step to help evaluate the utility of this program is to measure its performance as the problem grows in size and complexity. Our initial testing included only small programs, consisting of ten or fewer activities, three resource pools and a maximum availability of 5 resources per pool. This testing was performed to verify the accuracy of the algorithm. The program responded correctly and quickly for each of these tests. For performance under load, however, we needed to increase the size of the projects and the quantities of resources required. To create larger projects, we implemented a small program to randomly generate a set of tasks with durations from 1 to 6 time periods and assign a random number of required resources from each of three resource pools. We initially limited the project size to 25 tasks and gradually increased it to 50 tasks. We also varied the maximum number of resources from 6 to 10, but held the number of resource pools constant to 3. We ran the program on both a workstation and on the database server directly to compare performance. The server had 2 CPUs each with 6 cores running at 3.3 Ghz, 128 GBytes of

RAM and the workstation was equipped with a quad-core CPU running at 2.8 Ghz, with 32 GBytes of RAM. Storage of the data was controlled by DB2 v 9.7, which placed the data into tables on a separate storage area network.

Table 1 shows the run times associated with each process in the program. These processes correspond to the processes numbered 1 through 4 and described above.

Table 1: Program Run Times for 25 Task Example Project

	Workstation		Server	
Resources in each pool	**6**	**10**	**6**	**10**
Process 1	51 sec.	51 sec.	34 sec.	32 sec.
Process 2	15 mins.	13.5 mins.	15.0 mins.	12.7 mins.
Process 3	5.3 sec.	6.5 sec.	5.4 sec.	4.9 sec.
Process 4	1.02 sec.	1.4 sec.	.9 sec.	.85 sec.
Total Run Time:	15.9 mins.	14.5 mins.	15.7 mins.	13.3 mins.

The average run time for the four runs in table 1 was 14 minutes and 51 seconds, which seems to be an acceptable result for practitioners. Interestingly, the hardware differences between the workstation and the server were substantial, yet it seems that the program runs in about the same amount of time. This may be due to the fact that the optimization process is a serial and recursive operation; however, one would think that a server with more memory and faster CPUs would still be able to complete a set of serial processes much faster than a workstation not so equipped. There seems to be a larger difference in performance when the maximum number of resources per pool was increased; however, as stated previously, adding more resources to an IT project will probably not shorten its duration. The exception to this occurs when the resources are working on different projects, and so we varied the maximum number of resources available to simulate this occurrence.

The computation for multiple projects is identical to the computation for single projects. Tasks from different projects need not be analyzed separately since they are held in separate technical paths; that is, each project follows a different technical sequence and so the need for coding them differently during the program execution is absent. Tasks from different projects can be coded by project and added to the database as though it was a single project. The output as produced will then produce an optimal path for all projects that were added to the mix and can be easily discerned when displayed on a Gantt chart. It is only after the computation is complete and the need to view different projects separately becomes apparent that the task codes are grouped by project and treated once again as separate projects either for viewing by the PM or sent back to the production system for updating.

5. Limitations and directions for future research

This study considered only artificially generated projects that were formulated by an application, which was programmed to create tasks of arbitrary duration in a specific sequence. Each project created had a sequence of tasks, some of which were dependent upon others and some of which could be completed concurrently. We also limited the number of labor pools to three and the maximum number of resources in each pool to 6 for one run and 10 for a second run. Although the these projects had good face validity to be representative of production projects, our focus was solely on determining whether the software was able to successfully complete its task of schedule optimization and record the time it takes to do so. There are other factors that practitioners encounter in a production environment that are not dealt with here, and so this represents a limitation for which more research is needed.

We also did not induce changes in the schedules at specific points during the life of a project to check the effects that this might have on the schedule. Our purpose here was to create a software package that determined an optimum schedule and suggest that the technique could be used at any point in a project as things change (e.g., project scope,

number of resources, etc.). We feel confident that the software will perform adequately in such an environment as the concept could easily be applied in a changing environment, but more research is needed to check this.

We did not evaluate the software by assuming a multiple project environment. While this may seem like a major limitation, it should not affect the output of our program, because more than one project can be added simply by identifying the projects in the task code, adding them to the input, running the program and then separating them when the run is complete. Projects that compete for the same resources must be scheduled with regard to those restrictions, and so we feel that the program itself will be unaffected by adding additional projects to the mix. The place that this will have an effect is in the interface where projects have to be correctly coded for input to the program and correctly separated in the output from the program.

Additional research is needed to determine the difference in costs from the output produced by heuristics that are easy to apply and quick to run in an automated environment and the exact solution that we present here. The question of whether creating a program to determine an exact solution can reduce costs enough to justify its investment remains unanswered. Only continued research will tell. However, we should note that an impetus for this study itself was the noticeable lack of control that some supervisors and PMs have over the schedule when it includes limited resources. This continues to be a problem for large companies who may understand the limitations imposed by the number of resources available, but have a difficult time working these limitations into their scheduling process.

6. Conclusion

The length of time that it takes to complete a project can affect the ability of a firm to compete. By completing a project faster than its competitors, a firm will have a competitive edge because the overall project cost is reduced. Optimal scheduling of an IT project is constrained by both the technical sequence of activities and the limitations imposed by having a limited number of resources. Most IT shops in large companies attempt to execute several projects within fixed time periods and these projects can compete for the same resources. Given these conditions, we created a software program based upon previous research that should assist the PM in creating an optimal schedule. Our program has the following advantages:

- The output provides an exact solution. The PM knows that the project cannot be completed any sooner than the schedule produced.
- The program can be run at any time during the project to update the schedule as changes are incorporated or encountered.
- The program completes in a reasonable period of time so that it becomes viable for use in a production environment.
- It can be used in a multi-project environment where projects may compete for the same sets of resources.
- Resources can be placed into separate pools that are grouped together by skill set. Less experienced workers can be placed into pools that are different from experts to help better estimate task durations.
- Resource leveling is no longer required. Some software packages have options that allow the PM to level the resources when it is discovered that they are over-allocated. Because this program takes the resource limitations into account, there can be no over-allocation and the problem can only develop if changes in the number of resources occur during the life of the project. If this does occur, the changes can be incorporated to the program and it can be run again to find a new optimal schedule.

References

[1] F. P. Brooks, Jr., *The Mythical Man-Month: Essays on Software Engineering*. Reading, MA: Addison-Wesley, 1975.

[2] M. Agrawal and K.Chari, "Software effort, quality and cycle time: A study of CMM level 5 projects," *IEEE Transactions on Software Engineering,* vol. 33, no. 3, pp. 145-156, 2007.

[3] D. F. Cooper, "Heuristics for scheduling resource-constrained projects: An experimental investigation," *Management Science*, vol. 22, no. 11, pp. 1186-1194, 1976.

[4] J. D. Wiest, and F. K. Levy, *A Management Guide to PERT/CPM.* Englewood Cliffs, NJ: Prentice-Hall, 1969.

[5] R. C. Ash, "Activity scheduling in the dynamic, multi-project setting: Choosing heuristics through deterministic simulation," in *Proceedings of the 1999 Winter Simulation Conference.* Phoenix, AZ, USA, 1999, pp. 937-941.

[6] J. D. Wiest, "Some properties of schedules for large projects with limited resources," *Operations Research,* vol. 12, pp. 395-418, 1964.

[7] E. W. Davis, "Resource allocation in project network models - A survey," *Journal of Industrial Engineering*, vol 17, no. 4, pp. 177-188, April, 1966.

[8] W. Herroelen, B. De Reyck and E. Demeulemeester, "Resource-constrained project scheduling: A survey of recent developments," *Computers & Operations Research*, vol. 25, no. 4, pp. 279-302, 1998.

[9] A. L. Gutjahr and G. L. Nemhauser, "An algorithm for the line balancing problem," *Management Science*, vol. 11, no. 2, pp. 308-315, 1964.

[10] E. W. Davis and G. E. Heidorn, "An algorithm for optimal project scheduling under multiple resource constraints," *Management Science*, vol. 17, no. 12, pp. B-803-B-816, August, 1971.

[11] C. L. Moodie and D. E. Manville, "Project resource balancing by assembly line balancing techniques," *Journal of Industrial Engineering*, vol. 27, no. 7, pp.377-383, July, 1966.

[12] M. Held and R. M. Harp, "A dynamic programming approach to sequencing problems," *Journal of the Society of Industrial and Applied Mathematics*, vol. 10, no. 1, pp. 196-210, March, 1962.

[13] L. A. Johnson and D. C. Montgomery, *Operations Research in Production Planning, Scheduling and Inventory Control (Vol. 7),* New York, NY, Wiley & Sons, Inc., 1974.

[14] H. N. Psaraftis, "A dynamic programming approach for sequencing of groups of identical jobs, *Operations Research*, vol. 28, no. 6, pp. 1347-1359, 1980.

[15] E. M. Goldratt, *Critical Chain*, Great Barrington, MA, The North River Press, 1997.

[16] D. R. Robinson, "A dynamic programming solution to cost-time tradeoff for CPM," *Management Science*, vol. 22, no. 2, pp. 158-166, 1975

[17] W. Herroelen and R. Leus, "Identification and illumination of popular misconceptions about project scheduling and time buffering in a resource-constrained environment," *Journal of the Operational Research Society,* vol. 56, pp.102-109, 2005.

[18] R. Kolisch, R., "Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation," *European Journal of Operational Research*, vol. 90, no. 2, pp. 320-333, 1996.

[19] E. W. Davis and J. H. Patterson, "A comparison of heuristic and optimum solutions in resource-constrained project scheduling," *Management Science*, vol. 21, no. 8, pp. 944-955, 1975.

[20] J. Zhou, P. E. D. Love, X. Wang, K. L. Teo and Z. Irani, "A review of methods and algorithms for optimizing construction scheduling," *Journal of the Operational Research Society*, vol. 64, pp. 1091-1105, 2013.

[21] I. Pesek, A. Schaerf and J. Zerovnik, "Hybrid local search techniques for the resource-constrained project scheduling problem," *Hybrid Metaheuristics 2007, LNCS 4771*, pp. 57-68, 2007.

[22] T. Messelis, and P. D. Causmaecker, "An automatic algorithm selection approach for the multi-mode resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 233, no. 3, pp. 511-528, 2014.

[23] V. V. Peteghem and M. Vanhoucke, M., "An experimental investigation of metaheuristics for the multi-mode resource-constrained project scheduling problem on new dataset instances," *European Journal of Operational Research*, vol. 235, no. 1, pp. 62-72, 2014.

[24] J. Rieck, J. Zimmermann and T. Gather, "Mixed-integer linear programming for resource leveling problems," *European Journal of Operational Research,* vol.221, no. 1, pp. 27-37, 2012.

[25] H-J. Schütz and R. Kolisch, "Approximate dynamic programming for capacity allocation in the service industry," *European Journal of Operational Research*, vol. 218, no. 1, pp. 239-250, 2012.

[26] M. Bandelloni, M. Tucci and R. Rinaldi, "Optimal resource leveling using non-serial dynamic programming," *European Journal of Operational Research*, vol. 78, pp. 162-177, 1994.

[27] R. Kolisch, "Resource allocation capabilities of commercial project management software packages," *Interfaces*, vol. 29, no. 4, pp. 19-31, 1999.

[28] D. Howard and G. Chefetz, *Ultimate Study Guide: Foundations Microsoft Project 2010*. New York, NY: MSProjectExperts, 2010.

[29] R. R. Nelson, "IT project management: Infamous failures, classic mistakes, and best practices," *MIS Quarterly Executive*, vol. 6, no. 2, pp. 67-78, 2007.

[30] J. W. Olford, "Why is multiple-project management hard and how can we make it easier?" in *Managing Multiple Projects: Planning, Scheduling, and Allocating Resources for Competitive Advantage*, J. S. Pennypacker, and L. Dye, Eds., New York, NY: Marcel Dekker, 2002.

Biographical notes

Michael A. Chilton
Dr. Chilton is an associate professor of Management Information Systems at Kansas State University. He received his Ph.D. from the University of Arkansas. His research interests include project management, systems analysis and design, knowledge management, IT personnel and performance metrics, and IT pedagogy. He has published in a variety of journals including the *Journal of Management Information Systems*, the *Database for Advances in Information Systems* and the *Journal of Information Systems Education*, among others.

www.shortbio.net/mchilton@ksu.edu