Journal of the Korean
OR/ MS Society
Vol. 18, No. 3, December 1993

A Modular Decomposition Model for Software Project Scheduling

Kiseog Kim* and Barin N. Nag**

Abstract

The high level of activity in the development and maintenance of computer software makes the scheduling of software projects an important factor in reducing operating costs and increasing competitiveness. Software activity is labor intensive. Scheduling management of hours of software work is complicated by the interdependencies between the segments of work, and the uncertainties of the work itself. This paper discusses issues of scheduling in software engineering management, and presents a modular decomposition model for software project scheduling, taking advantage of the facility for decomposition of a software project into relatively independent work segment modules. Modular decomposition makes it possible to treat scheduling as clustering and sequencing in the context of integer programming. A heuristic algorithm for the model is presented with some computational experiments.

1. Introduction

Software engineering has become a major and critical segment of business activity, and at the same time an area in which costs are rapidly increasing. As in other areas of managing business operations, cost reductions can be achieved by efficient scheduling. However, software engineering has some special characteristics which make it difficult to apply established project management methods.

A well-known characteristic of software development is that the work is labor intensive, and the cost of man-hours far exceeds the cost of computer equipment, most of which can be considered as sunk costs. As a labor scheduling problem, the objectives of scheduling software ac-

* Department of Management, Pusan National University
** Department of Management, Towson State University, U.S.A.

tivity is to: (a) reduce the man hours required on a software project; and (b) bring project completion dates as close as possible to given due-dates. Both objectives can be reduced to monetary values of cost or penalty.

Project management in software development and maintenance is largely a problem of optimal assignment of hours from software engineers to jobs in hand. Equipment resources, such as computer time and primary developmental software, are typically unlimited. Thus, software scheduling is a class of labor scheduling problem, a problem that has been extensively studied, and for which several algorithms exist (see [5] for a discussion). In general, labor scheduling is an optimal allocation of resources, where the primary resource is labor hours in various categories.

Software scheduling, like most labor scheduling problems, present: (1) a set of constraints for available time, including both job hours, and a calendar of prior engagements; and (2) a set of constraints for qualifications for the task, some of which may be extendable, or "soft", constraints. In addition, software scheduling brings in considerations of: (3) interdependencies between activities, affecting the activity durations; and (4) the possibility of "part-completion", i.e. the case where a project is delivered a little short of completion. Further, software projects are well-known for the extreme variability, and uncertainty of activity durations. These special characteristics of software scheduling call for scheduling methods beyond established labor scheduling methods.

A special feature of turnkey projects in software engineering is small initial investments, so that software organizations tend to undertake several projects that run concurrently. Significant economies can be gained by efficient utilization of expertise and experience gained from previous projects, or work already completed on concurrent projects. Thus, software modules already developed for one project may be used elsewhere, directly or with minor modifications, with considerable savings in time and cost. Further, experience gathered by a particular software engineer on one project may enhance the work on another project having some similar characteristics. These aspects of software activity suggest a decomposition of software projects into modules. Scheduling can be done as an assignment of modules to individual software engineers, considering previous experience on similar modules. A classification of modules by their usability in multiple projects ensures the completion of key modules. In case the uncertainties of development force the project to extend beyond due dates, it will be the least important modules that are set aside.

In summary, this paper proposes a modular decomposition scheduling technique for software engineering, based upon a prior segmentation of the project activity into small modules. The problem outlined in this paper arises from the second author's own experiences in industry, in

particular at the Research & Test Division (R&T) of the Association of American Railroads (AAR). The R&T group of the AAR is continually faced with the problem of scheduling activity on several concurrent projects, with a limited number of software engineering staff, and still meeting all due dates.

Software scheduling has similarities with PERT techniques. Typically, in software work, there are specific projects to be accomplished within respective due-dates, with a scheduling objective to complete these tasks successfully given that activity durations are uncertain. However, PERT methods focus more upon the activities than upon the agents performing the activities. In the case of software development, varying qualifications of software engineers makes the task assignment problem critical to timely completion. Further, the usability of software segments in other areas makes the durations of developmental activity interdependent. As a result, difficulties arise in using PERT for software scheduling problem. Hence, this paper presents a mathematical model for the problem.

Optimization techniques in software scheduling are extremely difficult to validate. Computer systems maintain logs of user activity. The log lists user identification, hours of use, and specific activity, such as files and programs accessed. Thus, the software scheduling problem is one in which vast quantities of data could have been available from computer logs. Unfortunately, little data exists, if any. The flexible nature of the work and flexible hours discourages software engineers from maintaining rigorous work sheets. For this reason, the proposed decision support model had to be validated with hypothetical data. The paper explains the conceptual framework, and illustrates the working of the model with a hypothetical example.

2. Issues of Software Project Scheduling

2.1 Software Engineering Management and Software Scheduling

Software engineering management may be defined as the management of expectations, computer technology, human skills, time, and money to create a software product that meets the expectations of the client with a satisfactory return to the producer [14]. Issues in the management follow from the definition. The development activity can be characterized as follows :

1. The program development activities are grouped into projects, each having a known desired outcome of building a new system, or with making changes to an existing system.

In either case, the outcome is referred to as the product.

2. The project cannot be done by one person, and is usually done by a group of persons with a variety of specialized skills.
3. The project must meet the expectations of the client, and must be continued until the outcome is satisfactory.
4. Resources must be committed to the project, and to achieve a satisfactory return to the producer, the resources must be managed well. The major portion of committed resources consist of software engineering hours.
5. Conflicts between achieved objectives and expectations create uncertainties, and uncertainty management and adaptability is central to project management.

In a typical software development group, the manager's objective is to complete a project satisfactorily in the allotted time, using the available group of software engineers, each of whom has a certain number of hours available, and certain known qualifications regarding expertise in specific types of software. It is also typical for several projects to run concurrently, some small, some larger. As the resource cost is largely man-hours from software engineers, the project cost can be reduced by optimal scheduling of these hours. The scheduling problem has dual interpretations as follows :

1. Develop a scheduling calendar for each individual software engineer, over a single planning period, such as a week, in which he/she is to accomplish K out of N pending jobs, subject to the availability of T working hours in this period.
2. Develop a scheduling calendar to sequence individual jobs, and distribute them among the engineers, subject to due-dates for the completion of each project.

These conditions appear to be quite reasonable at first glance. The special character of software work is the dependance of activity durations on the sequence in which the activities are performed.

In most cases, the completion and delivery of a project results in a benefit, which can be taken as a "payoff" quantity in monetary terms. A project needs to be partitioned into a number of activity segments for assignment to individual engineers. Although there is no real meaning to a payoff for an activity segment, for modeling purposes an equivalent payoff can be considered such that the payoff for the whole project is the sum of the segment payoffs. Some segments can become especially important, or even critical, when parts of these have impacts on other projects. The real benefits of such segments become enhanced, in part by urgency, and in part by a portion of the revenue from other projects.

Thus, software engineering management presents the classic problem of maximizing payoff at

minimum cost. A solution to this problem ca. be obtained by optimal resource scheduling, where the resources are man-hours, and payoffs result from project completion. Although this is similar to labor scheduling, the case of software scheduling shows some differences, as follows :

1. The time available to complete a project is short, and all work may not be completed.
2. In view of the training time required to bring a software engineer "up to speed" with the type of programs used, hiring temporary personnel to increase available hours is not a feasible proposition.
3. In software development, there is often an association between jobs, i.e. job A is ineffective unless job B is also accomplished. The practical situation is that some elements of software can work only if some other elements are in place. In return-on-investment terms, the payoff from one piece of software is realized only if the other piece is also in place.
4. The assumption of independence of activities, critical to traditional PERT systems, does not hold. For a software engineer, experience on other jobs, that work on similar modules using the same software material, has a major impact on reducing the development time. In fact, the use of segments from the previous task on the same system can greatly reduce the development time of the new segment.

2.2 Modular Decomposition of a Software Project

In view of the associative and interdependent characteristics of software projects, the concept of modular decomposition may be introduced. A characteristic of a software project is that it can be decomposed into modules [12] and the usual experience in software development is that frequently used modules of software are maintained in software libraries to facilitate their use in multiple projects. The sequence of module development becomes an important parameter of scheduling with time advantages for projects if key modules are developed first. Re-use of key modules also affects the expertise level of a software engineer in the scheduling process.

A similarity may be found here with the objectives of the PERT model of a project, with a definite sequence of precedence relationships between activities (see [1] for a review of PERT). While traditional PERT cannot be used when activities are interdependent, some techniques may be borrowed from modified PERT, especially from a network decomposition of PERT [11]. This method decomposes the project network into several subnetworks by removing some linking activities, schedules each subnetwork of independent activities, and then, re-introduces the interlinks between subnetworks.

Modularity may be defined as segmentation into logical groups of computer code, where each

group performs a single specified task. Following a standard Top-Down approach to system design, a desired system performance can be broken down into a set of interacting sub-system performances, which in turn consist of a set of interacting modules. A description of Top-Down systems design, resulting in a breakdown into system modules can be found in a textbook on systems development [13]. The breakup of a system design into modules has immediate advantages as follows (from [8]) :

1. An improved understanding of the logical organization of the system is found by reducing complexity.
2. Maintainability and flexibility, in system updates, is improved.
3. Modules can be made independent of one another, thus reducing complexity of design and implementation.
4. Modules can be made sharable and reusable, thus reducing overall development effort.

It may be seen that modularization in software development, is a way to break down the system design into the design of a set of modules. The entire project can be decomposed into modules, and a set of seed modules, consisting of the critical segments, can be derived. The scheduling process can then be thought of as scheduling a system of modules. As found in [12], the criteria for decomposing systems into modules are found as follows :

1. A module relates to a data structure, its internal linking, accessing procedures and modifying procedures.
2. The sequence of instructions necessary to call a given routine and the routine itself are part of the same module.
3. The formats of control blocks, used in queues in operating systems and similar programs, are hidden within "control block modules", which conventionally form interfaces between program modules.
4. For high flexibility, character codes, alphabetic orderings, and similar data types, within a module, should be hidden.
5. The sequence with which items are processed within the module, should be hidden within the module.

A logical analysis of the above criteria show that they are derived from the usual requirements of systems development, i.e. a system should be transparent to the user, efficient in operation, and allow for flexibility in modification and update.

In addition, decomposition of the system development project according to the above criteria, facilitates the development work by a group, or team, of systems specialists. Systems engineers

often work on difficult and tight schedules. A need to work closely together in developing a system project can actually slow down the project, because of conflicts in scheduling software engineers to work together. Modular decomposition sets engineers free to work individually on segments of a system. There is only the need to interface the developed segments periodically. The decomposition criteria presented above are applicable from the viewpoint of division of work. For example, consider the condition that data structures and linkages are contained within a module. In terms of work management, this means that an engineer need only be given a standard technical specification of data and program interface to other modules, and allowed to work independently on his or her assignment. Thus, modular decomposition alters the needs and techniques of scheduling systems development.

3. Models for Software Project Scheduling

Models of software project scheduling, using the modular decomposition philosophy, are based upon the identification of clusters of software activity modules. The activity components are elemental, and consist of small tasks, such as the development of a single routine. The central theme of cluster formation is the existence of some similarity characteristic of the elemental components, as for example, the need for expertise of a particular type. Thus, clusters are assigned to software engineers, and any sequencing needed can be done within clusters.

The process of decomposition and cluster formation are subject to some assumptions, made as follows :

1. A project has a certain due date, so that the total time available for each available software engineer, within the time period, is known. When several projects run concurrently, the engineer's available time must be allocated between projects in the most productive manner.
2. The returns resulting from the completed project is known, so that when the project is decomposed into modules, the modules can be assigned proportional return values.
3. The qualifications of each software engineer is known, so that estimates can be made of the production time required by an individual engineer on an individual module, given its characteristics, and the engineer's capability. Qualifications vary, and so do module development times. Further, experience on a similar module reduces the development time for an engineer, and the extent of reduction can be estimated.

These assumptions are reasonable for a practical operation with a continuity of projects, where a specific project is given certain priorities, and where a known set of software engineers are available to work on the project.

Knowledge about the project defines the development methodology flow. The sequence and flow of activities in a project can be identified relatively early in the project, along with the qualifications required for each activity, and the expected durations and returns. Thus, tables can be worked out to represent (a) the breakdown of a project into its constituent modules, (b) the return associated with each module, as a fraction of the return from the entire project, and (c) estimated times for each engineer to complete each module. The last table would consider individual engineer's qualifications, as well as the time advantage in re-using a module already developed for another project.

Clusters of activities in modules are similar in the semantic sense, as well as the operational sense, to the cells used in Flexible Manufacturing Systems (FMS). Techniques similar to FMS cell formation methods [15] could be used in module formation in software engineering, if there was no overlap between adjacent cells. Once again, the interdependencies between projects, a character peculiar to software modules, prevent the partitioning of activity clusters for assignment to engineering teams.

3.1 Definitions

Let i,j = 1,……,M, indicate software activity segments, i.e. program modules to be developed, whose number is M. Let k = 1,……,K, indicate the software engineers to perform the development task, and let t = 1,……,T, indicate the time periods within the planning horizon. Variables and parameters are defined as follows

$$y_{ikt} = \begin{cases} 1, & \text{if activity i is selected for engineer k in period t} \\ 0, & \text{otherwise} \end{cases}$$

$$x_{ijkt} = \begin{cases} 1, & \text{if sequence ij is selected for engineer k in period t} \\ 0, & \text{otherwise} \end{cases}$$

T_{kt} = total time available for engineer k in period t

t_{ikt} = expected development time of module i, by engineer k, in period t

s_{ij} = a setup time for module j, when j is preceded by i

v_i = value of activity i

δ_t^-, δ_{kt}^-, θ_{kt}^-, θ_{kt}^+ = deviational variables

3.2 Fixed Time Model

One ordinary objective in the software schedu ing problem is to complete all jobs satisfactorily with the minimum cost in man-hours. When t ne total time available for each worker(T_{kt}) is fixed, however, some jobs may not be com leted due to insufficient time. In this case, similarities with FMS technology[15] enables th e identification of the following two goals :

Goal 1: minimize the total value of the unfini shed modules

Goal 2: minimize the sum of total setup time and development time
(or maximize the total unused time a ailable)

The assumption is that the partial completio n of work is acceptable, although the full completion is more desirable. Then the fixed time model is formulated with two competing goals.

Model : FIX_PC

$$Min\ P_1\{\sum_i v_i\,\delta_i^-\} - P_2\{\sum_k\sum_t \delta_{kt}^-\} \tag{1}$$

subject to

$$\sum_i\sum_j s_{ij}\,x_{ijkt} + \sum_i t_{kt}\,y_{ikt} + \delta_{kt}^- = T_{kt},\ \forall\ k,\ t \tag{2}$$

$$\sum_k\sum_t y_{ikt} + \delta_i^- = 1,\ \forall\ i \tag{3}$$

$$\sum_i x_{ijkt} = y_{jkt},\ \forall\ j,\ k,\ t \tag{4}$$

$$\sum_j x_{ijkt} = y_{ikt},\ \forall\ i,\ k,\ t \tag{5}$$

$$\sum_i\sum_j x_{ijkt} \le |S|-1,\ \forall\ k,\ t\ where\ s \subseteq\{1,\cdots\cdots,M\}\ ;\ 2 \le |S| \le M-1 \tag{6}$$

$$x_{ijkt} = 0\ or\ 1,\ \forall\ i,\ j,\ k,\ t \tag{7}$$

$$y_{ikt} = 0\ or\ 1,\ \forall\ i,\ k,\ t \tag{8}$$

$$\delta_i^- \ge 0,\ \forall\ i \tag{9}$$

$$\delta_{kt}^- \ge 0,\ \forall\ k,\ t \tag{10}$$

The model is an Integer Goal Program, with δ_i^- and δ_{kt}^- as deviational variables, and priorities P1 and P2 assigned to the two goals where P1 $\gg$ P2. Constraint set (2) ensures that time availability conditions are satisfied and r easures the total unused time available(Goal 2). Constraint set (3) determines the deviations f om the first goal. Constraint sets (4)−(7) relate to work assignments for software engineers. I n particular, constraint (6) develops the schedule sequence of jobs. The structure of the const aint avoids subtours, i.e. returning to the initial state before completing all jobs in the cluster or a single time period.

3.3 Flexible Time Models

An alternate situation in project management is that project due-dates are extendable (with associated penalty costs), but a project cannot be delivered unless it is complete. The impact of this condition is to modify the assumptions made in the FIX-PC model to: (1) flexibility for T_{kt}, or extension to available times, and (2) a constraint that all jobs must be completed. It is reasonable to suppose that the extension to available time comes from overtime work at a far greater hourly cost, as well as other costs. Thus, an objective may be taken to be the minimization of maximum overtime usage. In an alternate situation, the wage cost structure may not distinguish between regular hours and overtime hours, and software engineers need to put in extra hours at personal cost when needed. The objective can then be defined as a direct minimization of the total hours used. Models are derived as given below for these two objectives, labeled respectively FLEX-FC 1 and FLEX-FC-2.

Model : FLEX-FC-1

$$Min \{ \underset{kt}{Max}\ \theta^{+}_{kt} \} \tag{1}$$

subject to

$$\sum_i \sum_j s_{ij}\, x_{ijkt} + \sum_i t_{ikt}\, y_{ikt} + \theta^{-}_{kt} - \theta^{+}_{kt} = T_{kt},\ \forall\ k,\ t \tag{2}$$

$$\sum_k \sum_t y_{ikt} = 1,\ \forall\ i \tag{3}$$

$$\sum_i x_{ijkt} = y_{jkt},\ \forall\ j,\ k,\ t \tag{4}$$

$$\sum_j x_{ijkt} = y_{ikt},\ \forall\ i,\ k,\ t \tag{5}$$

$$\sum_i \sum_j x_{ijkt} \le |S| - 1,\ \forall\ k,\ t\ where\ S \subseteq \{1, \cdots\cdots, M\} : 2 \le |S| \le M-1 \tag{6}$$

$$x_{ijkt} = 0\ or\ 1,\ \forall\ i,\ j,\ k,\ t \tag{7}$$

$$y_{ikt} = 0\ or\ 1,\ \forall\ i,\ k,\ t \tag{8}$$

$$\theta_{kt},\ \theta^{+}_{kt} \ge 0,\ \forall\ k,\ t \tag{9}$$

Model : FLEX-FC-2

$$Min \sum_i \sum_j \sum_k \sum_t s_{ij}\, x_{ijkt} + \sum_i \sum_k \sum_t t_{ikt}\, y_{ikt} \tag{1}$$

subject to

$$\sum_i \sum_j s_{ij}\, x_{ijkt} + \sum_i t_{ikt}\, y_{ikt} \le \alpha T_{kt},\ \forall\ k,\ t \tag{2}$$

$$\sum_k \sum_t y_{ikt} = 1,\ \forall\ i \tag{3}$$

$$\sum_t x_{ijkt} = y_{ikt},\ \forall\ j,\ k,\ t \tag{4}$$

$$\sum_{j} x_{ijkt} = y_{ikt}, \ \forall \ i, \ k, \ t \tag{5}$$

$$\sum_{i} \sum_{j} x_{ijkt} \leq |S| - 1, \ \forall \ k, \ t \ where \quad \subseteq \{1, \cdots\cdots, M\} \ ; \ 2 \leq |S| \leq M - 1 \tag{6}$$

$$x_{ijkt} = 0 \ or \ 1, \ \forall \ i, \ j, \ k, \ t \tag{7}$$

$$y_{ikt} = 0 \ or \ 1, \ \forall \ i, \ k, \ t \tag{8}$$

The constraint structure of model FLEX_FC_2 is similar to FLEX_FC_1, but the objective is to minimize the total hours used. To simplify the formulation, the available time parameter T_{kt} is modified by a flexibility coefficient α, where $\alpha > 1$. Note that deviation variables are not includeded in model FLEX_FC_2.

3.4 Discussions

The models described above are 0-1 integer programs(IP), and can be solved as such by any standard available mathematical programming method with appropriate weights applied to the goals. The limitation is on the size of the problem. An IP does not have a polynomial bounded solution time, and the time required grows exponentially with the size of the problem. Further, as the model objective is already approximate and heuristically defined, it is preferable to solve the problem by a heuristic algorithm.

As the assignment groups are built up around the seeds, the selection of good seeds is critical to the successful performance of the heuristic. In the software assignment problem, the seeds are suggested by the nature of the work. These are the modules critical to the tasks on hand. Further, once a module is selected as a seed, there is a preference towards selecting other modules within the same project subgroup, until that particular subgroup is completely assigned. Thus, we would have a tendency towards completing a project subgroup, once it is begun. The benefits of this tendency, in the application to software scheduling is self-evident. The result is a complete assignment of program module development tasks to each software engineer, such that the development costs are minimized.

In some situations, there might be compatibility problems between the software engineers and the activities. Such situations are found when a subset of the engineers have the qualifications needed to accomplish a particular task. Conversely, some engineers have preferences for some types of software activity. The above models can accomodate these compatibilities and/or preferences by simply adjusting the input parameters. A low preference for an activity increases the time cost by a predetermined factor, while an incompatibility increases the time cost by a

large amount. The dynamism of the data is reflective of the real-world condition, i.e. the acquisition of experience on a similar program activity can in the future remove an incompatibility.

The primary constraints in the assignment process are the time constraints of the individual engineer over the period in question. These time schedules can be derived from a prior knowledge of leave schedules and other assignments already made. An individual software engineer has certain preferences for his/her work schedules, and it is relatively simple to incorporate such preferences into an individual scheduling framework. Thus, after the stages of modular decomposition and assignment have been performed, the schedule generated is a complete one, including both activity schedules and individual engineer schedules.

4. Heuristic Solutions for Software Scheduling Models

This section describes some heuristic solution methods for the three software scheduling models given above, one heuristic for each model. Solution time is reduced by solving part of the IP model optimally and using the partial solutions. The heuristics are two-phase, with cluster formation and sequencing phases, similar in some ways to the Generalized Assignment heuristics described in Fisher & Jaikumar, 1981 [4].

The working of the heuristic solutions is illustrated through an example problem in software scheduling. It has been mentioned previously that data is rarely collected on the activity schedules of software engineers. Such data is highly confidential in a competitive environment, and would not be published if it had been available. However, it is not difficult to construct data for a hypothetical software development project. Constructed data can have activity patterns very similar to real project data, although it does not resemble any given project. Tables 1 and 2 show data for a constructed example of software project activity.

Table 1.

INPUT DATA : Activity durations by software engineers(t_{kt}) : Values for individual jobs(V_i) : Available time(T_{kt})

i \ (k,t)	(1,1)	(2,1)	(3,1)	(1,2)	(2,2)	(3,2)	V_i
1	12	9	13	6	5	7	3
2	10	13	9	10	7	5	4
3	6	8	8	13	15	14	2
4	9	9	12	4	4	4	2
5	8	5	6	10	11	10	2
6	9	7	5	11	11	13	1
7	11	14	10	9	6	9	2
8	13	11	14	8	10	6	3
9	9	7	8	13	11	12	2
10	15	14	13	8	10	11	4
11	14	15	14	11	8	10	3
12	15	15	15	10	10	9	2
T_{kt}	20	20	20	20	20	20	

Table 2. INPUT DATA : Setup time(S_{ij}) for module j when j is preceded by i

from i \ to j	1	2	3	4	5	6	7	8	9	10	11	12
1	0	3	1	2	1	0	1	1	1	3	1	2
2	2	0	2	1	0	0	2	1	1	2	2	1
3	1	4	0	2	2	1	1	2	1	3	1	2
4	3	3	2	0	1	1	1	3	2	4	3	1
5	2	2	2	1	0	0	2	2	2	4	3	1
6	2	3	2	2	1	0	2	2	2	4	3	2
7	2	4	1	1	2	1	0	2	1	4	2	2
8	1	2	1	2	1	0	1	0	1	3	2	2
9	2	3	1	2	2	1	1	2	0	3	2	2
10	2	2	1	2	2	1	2	2	1	0	1	1
11	1	3	0	2	2	1	1	1	1	2	0	2
12	3	3	2	1	1	1	2	3	2	3	3	0

4.1 Heuristic for FIX_PC

(Input Data: t_{ikt}, v_i, T_{kt}, s_{ij} for all i,j,k,t)

Ⅰ. Select a seed in each cluster (k,t) :

1. Compute opportunity cost (OC) for each undeleted (k,t)

   Let $(k,t)^* = \{ (k,t) \mid \max_{k,t} [OC(k,t)] \}$

   and $t^*_{i(kt)^*} = \min_i \{ t_{i(kt)^*} \}$

2. Assign seed i* to (k,t)*, and delete row i* and column (k,t)*
3. If all columns have been deleted, GO TO Ⅰ, otherwise GO TO Ⅰ.1

Ⅱ. Assign remaining jobs into clusters :

1. Let Π be the index set of seeds.

   Reset $T_{kt} = T_{kt} - t_{ik*}$ for all $i \in \Pi$.

2. Compute $d_{ikt} = \min \{ s_{i*}, s_{*i} \} + t_{ikt}$ for all k,t and $i \notin \Pi$,

   where * is the seed of (k,t) cluster.

3. Solve Goal Program :

$$Min\ P_1 \{ \sum_i v_i \delta_i^- \} - P_2 \{ \sum_k \sum_t \delta_{kt}^- \}$$

subject to

$$\sum_i d_{ikt}\, y_{ikt} + \delta_{kt}^- = T_{kt} \qquad \forall\ k, t$$

$$\sum_k \sum_t y_{ikt} + \delta_i^- = 1 \qquad \forall\ i \notin \Pi$$

$$y_{ikt} = 0 \text{ or } 1 \qquad \forall\ i, k, t$$

$$\delta_i^- \geq 0 \qquad \forall\ i$$

$$\delta_{kt}^- \geq 0 \qquad \forall\ k, t$$

4. If $y_{ik*}^* = 1$, assign activity i to cluster (k,t).

Ⅲ. Schedule the job sequence in each cluster (by any Traveling Salesman algorithm, or by in spection in small clusters)

Illustrative Example (FIX_PC)

−Input data: Tables 1 and 2

Ⅰ. 1－Ⅰ.3: Assign seeds to clusters:

3→(1,1); 5→(2,1); 6→(3,1); 4→ 1,2); 1→(2,2); 2→(3,2)

Ⅱ. 1－Ⅱ.2: Index set of seeds Π = {3,5,6,4,1,2}

Reset T_{kt} and compute d_{ikt} (TABLE).

Table 3. Values of d_{ikt} and T_{kt}

i∈Π \ (k,t)	(1,1)	(2,1)	(3,)	(1,2)	(2,2)	(3,2)
7	12	16	1	10	7	11
8	14	12	1	10	11	7
9	10	9		15	12	13
10	16	16	1	10	12	13
11	14	17	1	13	9	12
12	17	16	1	11	12	10
T_{kt}	20 ↓ 14	20 ↓ 15	2 ↓ 1	20 ↓ 16	20 ↓ 15	20 ↓ 15

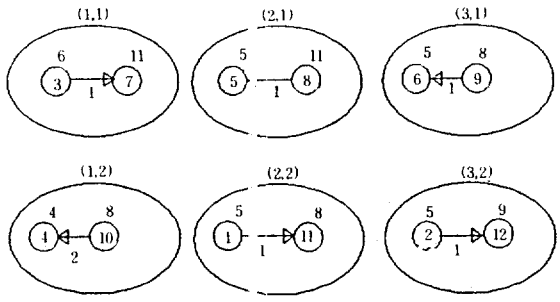
Ⅱ. 3－Ⅱ.4: Solve goal program and assign activities to clusters:

7→(1,1); 8→(2,1); 9→(3,1); 10→(1,2);

11→(2,2); 12→(3,2)

Ⅲ. Schedule sequence over clusters formed.

Figure 1 shows the results.



Note : Numbers in small circles are activity indexes numbers next to circles are activity durations (t_{ikt}), and numbers next to arcs are setup times (S_{ij})

Figure 1. Cluster Assignments from FIX_PC

4.2 Heuristic for FLEX_FC_1

(Input Data: t_{ikt}, v_{it}, T_{kt}, s_{ij} for all i,j,k,t)

[Note : Steps Ⅰ, Ⅱ, 1,2,4 and Ⅲ are the same as FIX_PC. They are repeated here for the reader's convenience.]

Ⅰ. Select a seed in each cluster (k,t) :

1. Compute opportunity cost (OC) for each undeleted (k,t)
   Let $(k,t)^* = \{ (k,t) \mid \max_{k,t} [OC\ (k,t)] \}$
   and $t^*_{i^*(kt)^*} = \min_{i} \{ t_{i(kt)^*} \}$
2. Assign seed i* to $(k,t)^*$, and delete row i* and column $(k,t)^*$
3. If all columns have been deleted, GO TO Ⅱ, otherwise GO TO Ⅰ.1

Ⅱ. Assign remaining jobs into clusters :

1. Let Π be the index set of seeds.
   Reset $T_{kt} = T_{kt} - t_{ikt}$ for all $i \in \Pi$
2. Compute $d_{ikt} = \min \{ s_{i^*}, s_{i} \} + t_{ikt}$ for all k,t and $i \notin \Pi$,
   where * is the seed of (k,t) cluster.
3. Solve Goal Program :

$$Min \{ \underset{kt}{Max}\ \theta^+_{kt} \}$$

subject to

$$\sum_i d_{ikt}\, y_{ikt} + \theta^-_{kt} - \theta^+_{kt} = T_{kt} \qquad \forall\ k,\ t$$

$$\sum_k \sum_t y_{ikt} = 1 \qquad \forall\ i$$

$$y_{ikt} = 0 \text{ or } 1 \qquad \forall\ i,\ k,\ t$$

$$\theta^-_{kt},\ \theta^+_{kt} \geq 0 \qquad \forall\ k,\ t$$

4. If $y_{ikt}^* = 1$, assign activity i to cluster (k,t).

Ⅲ. Schedule the job sequence in each cluster (by any Traveling Salesman algorithm, or by inspection in small clusters)

Illustrative Example (FLEX_FC_1)

- Input data: Tables 1 and 2
- An example solution by FLEX_FC_1 is shown in Figure 2.

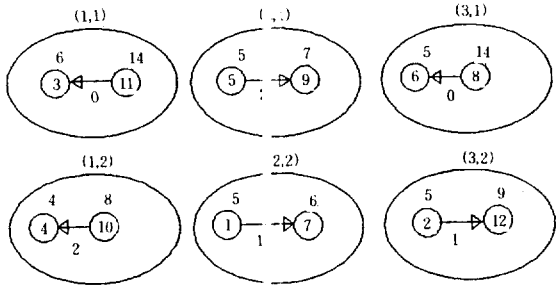


Figure 2. Cluster Assignments from FLEX_FC_1

4.3 Heuristic for FLEX_FC_2

(Input Data: t_{ikt}, v_i, T_{kt}, s_{ij} for all i,j,k,t)

I. Select a seed in each cluster (k,t) :
(As for FLEX_FC_1)

II. Assign remaining jobs into clusters :
1. & 2. (As for FLEX_FC_1)
3. Solve Integer Program :

$$Min \sum_i \sum_k \sum_t d_{ikt}\, y_{ikt}$$

subject to

$$\sum_i d_{ikt}\, y_{ikt} \le T_{kt} \qquad \forall\ k,\ t$$

$$\sum_k \sum_t y_{ikt} = 1 \qquad \forall\ i$$

$$y_{ikt} = 0 \text{ or } 1 \qquad \forall\ i,\ k,\ t$$

4. If $y_{ikt}^* = 1$, assign activity i to cluster (k t).

III. Schedule the job sequence in each cluster by any Traveling Salesman algorithm, or by inspection in small clusters)

Illustrative Example (FLEX_FC_2)

- Input data: Tables 1 and 2
- An example solution by FLEX_FC_2 is shown in Figure 3.

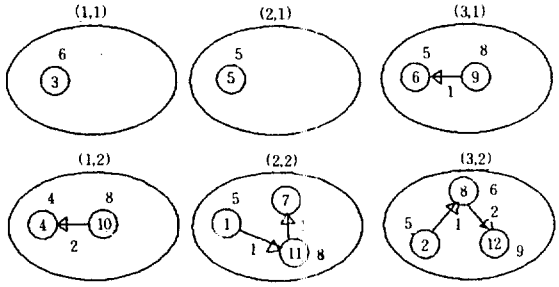


Figure 3. Cluster Assignments from FLEX_FC_2

4.4 Summary of Experiments

A number of experiments were performed with different parameter values, and objective variations. The results are shown in Table 4. It is seen that when a large number of hours are available, as when T_{kt}=20 or even 17, FIX_PC dominates. The total hours used are less and there is no overtime usage. The situation is reversed when available hours are decreased. Under these conditions, FIX_PC leaves work incomplete, which is possibly in practice an undesirable solution. Flexibility of hours and overtime enables the heuristic to find feasible solutions. A preference of FLEX_FC_1 over FLEX_FC_2, or vice versa, is determined by the practical dimensions of work contracts and overtime rate structure. A high cost of overtime might suggest FLEX_FC_1.

It may be noted that the problem data used is typical of practical industry problems, but is still a hypothetical problem. Further, it is well known that heuristic results and performance often depend upon the problem structure. Thus it is entirely possible that on some problems FLEX_FC_1 will be superior to FLEX_FC_2, while the opposite will be true in other cases.

Table 4. Summary of Experiments

model	T_{ki}	Available over-time Hours	Incomplete Jobs	Total Hours Used	Maximum Over-time Usage
FIX_PC	20	0	ϕ **	92**	0≠
FLEX FC_1	20	∞	ϕ ≠	94	0*
FLEX FC_2	20	4	ϕ ≠	82*	3
FIX PC	17	0	ϕ **	92*	0≠
FLEX_FC_1	17	∞	ϕ ≠	98	1*
FLEX_FC 2	17	3	ϕ ≠	92*	1
FIX PC	16	0	A12**	76*	0≠
FLEX_FC 1	16	∞	ϕ ≠	98	1*
FLEX FC 2	16	3	ϕ ≠	92*	1
FIX_PC	15	0	A7 & A12**	65*	0≠
FLEX_FC_1	15	∞	ϕ ≠	94	2*
FLEX FC 2	15	3	ϕ ≠	92*	3

Note : ≠ : due to constraints

* : due to objective function

** : due to objective function (higher priority

5. Conclusions

The purpose of this paper is to discuss the issues involved in scheduling activities in software engineering, and to present a scheduling model based upon these issues. Software projects consist of interdependent activities, and the concept of modular decomposition is presented as a method to overcome the effects of interdependence. Knowledge about a project, and experience on similar projects, are used to decompose a project into elemental activities. At this level of decomposition, interdependencies between segments can be explicitly stated as preferences for some job clusters. A clustering system is proposed, as an integer goal program, to form job clusters for assignment to individual software engineers, with heuristic solution mechanisms for problems of practical size. The method provides for individual sequencing and scheduling, and simplifies the scheduling task of software engineering managers in planning for project due

dates.

Considerable research work is possible in extending the concepts and constraints used in the decomposition-based scheduling model described to conditions more realistic in industry practice. Two areas of model enhancement can be directly seen. The first concerns the variances associated with activity times in software engineering. The accuracy of the schedule should be improved by explicit inclusion of some measure of the variance in the scheduling model. The second possibility arises from the dynamic nature of the software environment. Schedules need to be adjusted, modified, or updated frequently. Model management is a possible answer. Research is needed to make an efficient scheduling method usable by a systems manager in an industry environment.

Acknowledgments

This paper is based on the work done in 1991-1992 while the first author was at the Towson State University as an exchange professor. The authors express their gratitude to Tom Warfield and Tom Guins of the Research & Test Division, Association of American Railroads, Washington, D.C., for their contributions in systems management issues and problems. We also thank the referees and the Departmental Editor for suggesting several improvements in the paper.

References

[1] Adlakha, V. and V.G. Kulkarni, "A Classified Bibliography Of Research On Stochastic PERT Networks: 1966-1987," *INFOR*, Vol. 27(1989), pp.272-296.

[2] Bell, C.E., "Maintaining Project Networks in Automated Artificial Intelligence Planning," *Management Science*, Vol. 35(1989), pp. 1192-1214.

[3] Cusumano, M.A. and C.F. Kemerer, "A Quantitative Analysis of U.S. and Japanese Practice and Performance in Software Development," *Management Science*, Vol. 36(1990), pp. 1384-1406.

[4] Fisher, M. and R. Jaikumar, "A Generalized Assignment Heuristic for Vehicle Routing," *Networks*, Vol. 11(1981), pp. 109-124.

[5] Glover, F., C. McMillan, and R. Glover, "A Heuristic Programming Approach to the Employee Scheduling Problem And Some Thoughts on Managerial Robots", *Journal of Operations Management*, Vol. 4(1984), pp. 113-128.

[6] Granot, D.G. and D. Zuckerman, "Optimal Sequencing And Resource Allocation In Research And Development Projects," *Management Science*, Vol. 37(1991), pp. 140-156.

[7] Khan, M.B. and M.P. Martin, "Managing The Systems Project," *Journal of Systems Management*, Vol. 40(1989), pp. 31-36.

[8] Karimi, J. and B.R. Konsynski, "An Automated Software Design Assistant," *IEEE Transactions on Software Engineering*, Vol. 14(1988), pp. 194-210.

[9] Mannino, M.V., B.S. Greenberg, and S. N. Hong, "Model Libraries: Knowledge Representation and Reasoning," *ORSA Journal on Computing*, Vol. 2(1990), pp. 287-301.

[10] Nag, B., B.L. Golden, and A.A. Assad, "Vehicle Routing with Site Dependencies," *Vehicle Routing: Methods and Studies*, Eds. B.L. Golden and A. A. Assad, North-Holland, Amsterdam/New york, 1988.

[11] Parikh, S.C. and W.S. Jewell, "Decomposition of Project Networks," *Management Science*, Vol. 11(1965), pp. 444-459.

[12] Parnas, D., "On the Criteria To Be Used in Decomposing Systems into Modules," *Communications of the ACM*, Vol. 15(1972), pp. 1053-1058.

[13] Powers, M.J., P.H. Cheney, and G. Crowe, *Structured Systems Development (2nd. Ed.)*, Boyd & Fraser Publishing, Boston, MA, 1990.

[14] Steward, D.V., *Software Engineering: with Systems Analysis and Design*, Brooks/Cole Publishing Company, Monterey, CA, 1987.

[15] Wei, J.C. and N. Gaither, "An Optimal Model for Cell Formation Decisions," *Decision Sciences*, Vol. 21(1990), pp. 416-433.