

Time-line based model for software project scheduling with genetic algorithms

Carl K. Chang^a, Hsin-yi Jiang^{a,*}, Yu Di^b, Dan Zhu^c, Yujia Ge^d

^a Department of Computer Science, Iowa State University, USA

^b Department of Computer Science, University of Illinois at Chicago, USA

^c College of Business, Iowa State University, USA

^d Department of Computer Science, Zhejiang Gongshang University, China

ARTICLE INFO

Article history:

Received 19 July 2007

Revised 23 February 2008

Accepted 2 March 2008

Available online 20 March 2008

Keywords:

Genetic algorithm

Optimization

Project management

Scheduling

Task assignment

ABSTRACT

Effective management of complex software projects depends on the ability to solve complex, subtle optimization problems. Most studies on software project management do not pay enough attention to difficult problems such as employee-to-task assignments, which require optimal schedules and careful use of resources. Commercial tools, such as Microsoft Project, assume that managers as users are capable of assigning tasks to employees to achieve the efficiency of resource utilization, while the project continually evolves. Our earlier work applied genetic algorithms (GAs) to these problems. This paper extends that work, introducing a new, richer model that is capable of more realistically simulating real-world situations. The new model is described along with a new GA that produces optimal or near-optimal schedules. Simulation results show that this new model enhances the ability of GA-based approaches, while providing decision support under more realistic conditions.

© 2008 Elsevier B.V. All rights reserved.

1. Introduction

Experienced software project managers understand the intrinsic difficulty of managing a complex project involving a large team of engineers and developers working together in a dynamic, ever-evolving environment with volatile project parameters. Task assignments and scheduling have always been crucial to the software development process, and the quality of those assignments greatly influences the success of a project.

The main problem with managing a large software project is that it is tedious and error-prone [38], oftentimes requiring specific information to support it. Frequently, the information needed to optimize performance is missing. Common project management tools [1] such as Microsoft Project are not effective at managing optimization problems. One of the main problems of these tools is that they use schedule representations that are “inadequate to model the evolutionary and concurrent nature of software development” [2,3].

To the best of our knowledge, [2] represented the first attempt to develop a richer representation of software project management, one capable of supporting a genetic algorithm based optimization approach. Improvements to the original model (i.e. SPM-Net [2] or PM-Net [4]) were proposed in [5]. Genetic algorithms (GAs), already commonly used in many domains of optimization research

[6,7], provide better solutions to complex project management problems.

The main goal of the current work is to extend and improve the model in [5], which will be briefly reviewed, making it more precise and realistic. GA is still our choice for optimization, but the internal “genome representation”, i.e. the model structure and parameters, has been greatly improved. Complex project management schemes that exist in the real world but were not representable in the old model can now be presented. The added parameters increase the user's control of the model. In addition, concepts from software cost estimation techniques are incorporated into the model.

This paper is organized as follows: Section 2 introduces some basic concepts on GA; Section 3 gives a literature review on scheduling in software project management; Section 4 gives an overview of the model in [5] and suggests improvements in several aspects; Section 5 describes the new and improved model in detail; Section 6 explores properties of the new model and collects numerical data; and Section 7 outlines the directions for future research.

2. Genetic algorithms

GA is a particular type of evolutionary algorithms introduced in the 1970s by John Holland [12]. Such algorithms attempt to simulate natural evolution and form a populous selection of possible solutions by exploring the search space in an attempt to find near-optimal solutions to optimization problems. They do not exhaustively search the entire solution space, but instead start

* Corresponding author. Tel.: +1 5154514458.

E-mail address: hsinyij@iastate.edu (H.-y. Jiang).

with a small population of possible solutions and then evolve towards better solutions. In the end, fairly good though not necessarily optimal solutions for the optimization problem will be produced. The outline of the algorithm is shown below:

1. Find representations of possible solutions, called *genomes*, which are composed of *genes* chosen so that by changing the genes, all possible solutions can be represented.
2. Map each genome to a real number using a *fitness function*. The goal is to find a genome that maximizes the fitness function.
3. Randomly generate a set of genomes called the *initial population*.
4. Repeat the following steps until a *stopping criterion* is met:
 - (a) *Selection*: select a subset of the population for use in the next steps.
 - (b) *Mutation*: produce a new genome from an older one by changing some of its genes.
 - (c) *Crossover*: produce a new genome by combining genes from two older ones.
 - (d) *Combination*: build a new population using some of the old and new genomes.

Some examples of stopping criteria are:

- (a) A certain number of generations have been generated.
- (b) The best individual in the current generation has a high enough fitness value.
- (c) The identity of the best individual has not changed for a number of consecutive generations.

GAs do not guarantee the discovery of the global optimum. However, in many applications a near-optimal solution is often acceptable. The representation chosen for the genome is pivotal to the performance of GAs [37].

3. Related work

Generally speaking, our work is to investigate a strategy to assign employees to tasks, maximize the quality of project schedules, and minimize the total project cost at the same time. Benefiting from the computed schedule of task assignments and durations, thanks to the existence of GA, project managers can carry out the balancing acts to minimize project costs while achieving other project objectives. We will report our survey of the related work in two major categories: software project effort estimation, and scheduling.

3.1. Software project effort estimation

It has been extremely difficult to precisely estimate the total effort demanded for sustaining the lifecycle of a software system. Until now, it is still largely an open issue for researchers although many estimation models had been proposed to predict the effort to construct and maintain software. Among them, models designed with Function Point Analysis (FPA) and Source Lines of Code (SLOC) are commonly seen in the literature.

As a historical note, A.J. Albrecht of IBM devised the function point metric in 1979 [14,15]. FPA became a standard method recognized by ISO, renamed as the Functional Size Measurement, to measure the functional size of an information system. It is calculated based on a requirement-centered view of software and is platform-independent [16,17]. FPA can be employed to determine the software size for cost estimating and to find the testing effort required in the information system [15]. As a result, various organizations adopted FPA as the designated measurement [17–20]. Albrecht and Gaffney also suggested the use of function points (FPs) to estimate SLOC and then utilize SLOC to estimate the work

effort [18]. Later, Felfernig and Salbrechter described the application and extension of FPA for effort estimation in the development of knowledge-based configuration systems [17]. Moreover, Niesink and Ahn et al. proposed a model based on FPA to predict software maintenance effort [19,20].

On the other hand, SLOC is used to measure the size of a software program by counting the number of (non-commentary) lines in the text of the source code of the program. It is typically employed to estimate the effort required to develop a program. A commonly known model called COCOMO calculates the effort (in person-months) based on SLOC [21]. Its second generation model, called COCOMO II, is an improvement of COCOMO. It employs both SLOC and unadjusted function points to express sizes of software [22]. As exemplified from the measurements of project sizes in both the original COCOMO and COCOMO II, most researchers believe that there exists a high degree correlation between FPA and SLOC [18] [22].

Other estimation methods also exist. For example, analogy-based estimation, seen in some of the existing work, calculates the distance between the software project being estimated and the historical software project data [23]. After having calculated the distances, it retrieves data of the most similar projects based on the calculated distance, and estimates the effort based on those similar projects [23]. In recent years, Huang and Chiu applied GAs to determine the appropriate weighted similarity measures of effort drivers used in analogy-based software effort estimation models [24]. Later, they proposed an adjusted analogy-based software effort estimation model which adopted GA to adjust the effort based on the similarity distances [25]. The experimental results obtained from the work of Huang and Chiu suggested that such methods indeed helped improve the accuracy of effort estimation.

Bayesian analysis is another methodology used to predict software effort. The results in Pendharkar's work indicated that the probabilistic forecasting model allows managers to estimate joint probability distribution over different software effort estimations (i.e., the effort estimations in terms of different concerns) [26]. Following that, project managers can use the joint probability distribution to develop a cumulative probability distribution, which in turn helps them estimate the uncertainty of the extent as to how the actual project effort may be greater than the estimated effort.

In fact, most of the existing software project effort estimation models can be employed to support the approach presented in this paper. For the purpose of illustration, without loss of generality, COCOMO will be most often utilized for the rest of the paper.

3.2. Scheduling

Scheduling problems are NP-hard with extremely complex combinatorial optimization issues. Research on finding solutions of resource-constrained project scheduling problems (RCPSPs) has progressed for several decades. The methods form two distinct classes: exact methods and heuristic methods. Exact methods include backtracking, branch and bound, critical path method with its variations, dynamic programming and implicit enumeration. Heuristic methods, which are popular in scheduling, include simulated annealing (SA) [10], Tabu search (TS) [11], and GA [12]. All of these methods may be further categorized into stochastic and deterministic approaches [13]. GA belongs to the class of stochastic search methods.

3.2.1. Project scheduling with genetic algorithms

In recent years, researchers in software engineering found that genetic algorithm (GA) is a feasible optimization method for their problem domains, thus it is used for an increasing number of applications.

Hegazy et al. proposed a model for cost optimization and dynamic project control [36]. The model incorporates an integrated formulation for estimating, scheduling, resource management, and cash-flow analysis. GAs are applied to allocate optional construction methods for each activity. Although resource constraints are also considered in this work, they did not consider much on human factors. Their focus is more on the best combination of construction methods.

In 1996, Chan et al. suggested to solve resource scheduling problems using genetic algorithm [8]. In their paper, an objective function related to resources and constraints related to the precedence of tasks are formed. However, the work was emphasized more on the GA part, and did not discuss much on the successful/failed factors of projects in scheduling.

Hindi et al. demonstrated that the evolutionary algorithm is effective to solve RCPSPs [27]. An empirical study of 2370 instances was conducted. The result showed that the algorithm is capable of finding the best-known solution in 68% of the instances with an average overall error rate of 0.95%.

Valls et al. proposed a hybrid GA for the RCPSP [28]. Although it is helpful for the scheduling problem, the focus on their work is based on the improvement of the algorithm itself, instead of the concerns of the factors with projects.

Alba and Chicano have shown that GAs are quite flexible and accurate for project scheduling, and regarded as an important tool for automatic project management [29]. They provided the basic idea on applying GA for automated task assignments. However, in their model, the experiences and skills of employees are not distinguished. Moreover, changing employees are not allowed during the project duration.

To achieve better performance, researchers continue to focus on issues such as representation (direct and indirect) and operators (initialization, mutation, and crossover). Depending on the specific scheduling problem, the performance of GAs can vary greatly or only by negligible amounts. Our previous model can be considered as an early effort to apply GAs in the software project management environment [5].

3.2.2. Project scheduling considering human resource factors

The philosophy of this paper on considering and analyzing more human resource factors is similar to Plekhanova's work [30]. In that paper, it is asserted that in the mathematical theory of scheduling, resource capability factors are not considered as the factors that influence the schedule since the resources are assumed to be equal (i.e., they possess equal capabilities). However, in practice, the most effective and efficient manner of resource allocation in software projects should be founded on heuristic approaches that consider varying capabilities.

As described in this paper, our new model introduces the timeline axis, considers more human resource factors in project management than our previous model, and has the potential to become a more practical model for project managers to adopt.

4. Overview of previous model

Using a task model, Chang et al. [5] described how to apply GAs to find near-optimal schedules. In the task-based model, the representation of a problem consists of:

- (1) Representation of project: $TPG = (V, E)$
 - The project is represented as a task precedence graph.
 - A directed a cyclic graph where nodes ($v \in V$) represent tasks and edges ($e \in E$) represent task precedence.
 - Each task is associated with an estimated effort and the required skills.

- 2) An employee database D_{emp} with information of skills and salary.
- 3) An objective function to evaluate the performance of a schedule.

Genetic representation is an orthogonal 2D array with one dimension for tasks and the other for employees. GA operators are adopted from the C++ Genetic Algorithm Library (i.e. GALib) [31]. In this approach, scheduling is a two-stage process: the first stage evaluates how a genome satisfies constraints; the second stage evaluates the schedule performance of a genome. The simplest objective function for such evaluations can be defined as:

$$\begin{aligned} \text{Composite objective function} = & \text{Validity}^* (\text{OverLoadWeight} / \text{OverLoad} \\ & + \text{MoneyWeight} / \text{CostMoney} \\ & + \text{TimeWeight} / \text{CostTime}). \end{aligned}$$

Validity (validity of job assignments) is usually scored on a 0/1 basis – 0 if the assignments are invalid and 1 if they are valid. *OverLoad* (minimum level of overtime) is the amount of time worked beyond the individual overtime limit, which is summed over all employees. *CostMoney* (minimum cost) is the total labor cost of performing the project, which is computed using the labor rates of each resource and the hours applied to the tasks. *CostTime* (minimum of time span) is the total time span required to finish the project, from the start of the first task until the end of the last. These three objectives share the same fitness function, to minimize the appropriate input, i.e., overload, project costs, and time. The fitness value is the summation of weighted component objective values. The remainder of this section examines some limitations of the above task-based model.

4.1. Restrictions on tasks

In our earlier model, tasks were the basis of personnel assignments and calculation of the fitness function. The fitness function calculation imposed two restrictions in task assignments:

- A task must begin as early as conditions allow. That is, it begins as soon as all preconditions are satisfied. However, in real-world projects, it may sometimes be advantageous to delay a task.
- A task cannot be interrupted during its execution. In real-world projects, however, there may be another emerging task that is more critical to global fitness. In this case, it may be advantageous to refocus efforts on the other task.

4.2. Restrictions on employee assignments

In the task-based model, it was assumed that employees worked on an assigned task from its beginning to its end. It was further assumed that, once assigned, an employee's total effort would be applied to the task. While it is often advantageous to retain employees on a task to gain experience or maintain project stability, it may sometimes be useful to remove an employee from a task where learning has been significantly slowed or skills are not a good match. The earlier model cannot support such re-assignments and skills are not represented in sufficient detail to facilitate assessment of skill matching.

4.3. "Mongolian Horde" strategy

Implicit in the assignment strategy of the earlier model is the assumption that an unlimited number of suitably skilled employees can be applied to a task for as many hours per week as they can legitimately work. The assumption that we can add people to accelerate a task without any limit, also known as the Mongolian

Horde strategy, is not realistic. Therefore, the effect of different groups of employees and external consultants on task performance and the impacts of coordinating their efforts must be considered for more realistic situations.

4.4. Skill match and experience

To assign people with the skills required to tasks, (instead of applying the “Mongolian Horde” strategy) the model must include a representation of the suitability of employees to perform tasks. The earlier model lacked the ability to assess workers’ expertise. That is, in our earlier work, an employee either possessed a skill or did not, and there was no notion illustrating the level of experience.

4.5. Different scales for objectives

In the fitness calculation, the three objectives of the fitness function – the amount of time for overtime, cost, and total length of time – are combined as a weighted sum. The weight of each component is derived from experience. However, since the scales of the objectives are different, some objectives may be ignored while GA is processed due to the small scale. The model can be enhanced so that the three objectives are directly interpreted on a common scale and can be simply summed with no weighting. For example, overloads for employees can directly result in increased cost and excessive duration (length of time) can be considered with a “penalty”. Moreover, total length of time can be converted to cost.

5. Specification of the new model

To address the aforementioned shortcomings, the task-based model [5] must be changed. Some changes, e.g. the specific characteristics of employees and tasks, are direct enhancements to the model and will be discussed later. A more fundamental change – introduction of a time-line – had the potential for great impact and thus will be discussed here. The “time” expands the two-dimensional (task and employee) model into a three-dimensional one, showing the effort of each employee applied to each task in each time unit. The time-line became necessary due to the requirements to represent re-assignment of employees, learning, the suspension and resumption of tasks, and the introduction of hard and intermediate (that is, task-specific) deadlines.

5.1. Introduction of time-line and discussion of computation burden

Using a time-line (and thereby breaking down a task into smaller components of time-sliced activity) solves the problems described in Sections 4.1 and 4.2 by assigning employees to tasks for discrete time units during the duration of a task, instead of assigning them to the entire duration of the task. In this way, the calculation of the fitness function can be performed at the time of assignment, which allows us to incrementally update various parameters (for example, skills of employees, cf. Section 4.4). In implementing the time-line based model two decisions must be made:

1. Quantum of the time-line. In theory, the granularity of the time-line can be arbitrarily small, and the smaller the unit, the more “precise” the model. Beyond a certain point, however, such “precision” is impractical. Therefore the time unit (i.e. quantum) should not be too small; for a small project, one week or one day is acceptable while for a large project, a month or a calendar quarter might be appropriate. We decided that ϕ , time unit parameter, is to be defined as the number of time units in a month. Thus, if the time unit is a week, $\phi = 4$.

2. Practical upper bound of the time dimension. To be realistic, an upper bound for the duration of a project should be imposed. For example, 1200 months (100 years) is definitely an upper bound of time that no project will reach. For most projects, the upper bound can be much smaller, a few years at most.

The introduction of a time-line to make the model more realistic means it will require more extensive computation than the earlier task-based model. For the same number of tasks and employees, the number of individual assignments in this model will be B/U times greater than that of the task-based model, where B is the upper bound of time, and U is the time unit. Accordingly, the amount of computation required by this model will also be roughly B/U times that of the task-based model. In the experiments described herein, time quanta of months and a maximum duration of three years were used. Smaller granularities are possible, with correspondingly increased computation time. However, the impact is only linear, so the rate of increase of the computational burden is much less than it would be for an exhaustive search method.

5.2. Skill model

Numerical presentations represent skills similar to the model used by Chang et al. [5]. In the new model, an employee can also obtain a skill during the course of a project through training. The definitions of the individual properties of employees are described in the subsequent sections.

5.3. Employee models

An employee is represented by a numerical identifier (ID) and many properties. One property is “type”, of which there are currently two values: “employee” or “contractor” (i.e., external consultants). A contractor cannot receive any training (see Section 5.3.3). Other properties are described in the following sections.

5.3.1. Employee compensation model

A more complex compensation model is used here to convey that employees may be paid at a different rate when working overtime. Three rates are used in this model:

- S_{100} : The base salary for a month of work (where 100% represents one month), including any training time that can be charged to the project. For training, see Section 5.3.3.
- S_{over} : The relative rate for overtime work.
- h_{max} : The maximum number of hours the employee can work, expressed as a percentage. Thus, 175 means the employee cannot work over 175%.

Thus, the total payment for h work hours (measured as a percentage of full-time work hours in one time unit) is:

$$P = \frac{1}{\phi} \times \begin{cases} S_{100} \times h\% & 0 \leq h \leq 100 \\ S_{100} \times 100\% + S_{over} \times (h - 100)\% & 100 \leq h \leq h_{max} \\ \infty & h > h_{max} \end{cases}$$

5.3.2. Employee skill list

An employee’s skill list is an array of proficiency score values, with skill ID as index. A proficiency score value is a number between 0 and 5, with 0 indicating the employee does not possess the skill and 5 indicating that he or she has total mastery of it. Proficiencies do not have to be integers. Fractional scores can also be used. This mechanism is used to evaluate the proficiency of employees in the skill areas required by a project, thus partially solving the problems described in Section 4.4.

5.3.3. Employee training model

In the previous task-based model, it was assumed that the employees' skill levels remain constant throughout the project. The new model introduces "training" as means to improve skills, with values of 0%, 25%, 50%, 75%, and 100% allowed monthly for each employee. At the expense of additional computation overhead, a finer grain of values is possible. Contractors cannot charge for training in this manner. The cumulative training time of an employee for a skill s at the end of time unit m , $T_{\text{train}}(m, s)$, is defined as: $T_{\text{train}}(m, s) = \frac{1}{\phi} \times \sum_{i=0}^m \text{TrainHour}(i, s)$. If an employee has a proficiency of p_0 for a specific skill at the beginning, the proficiency after T hours of training will be: $p = \min \{p_0 + \mu \times T, 5\}$, where μ is the employee's learning speed. This is a very simple model of learning [39] [40]; more complex models can be introduced without altering much the nature of the time-line based model.

5.3.4. Employee experience model

An employee may have low proficiency in a skill but if they work on a task requiring that skill long enough, they will become more proficient as a result of experience. This is often referred to as "on-the-job training". It is assumed that each employee has "Initial Experience" for each skill area.

If an employee's experience level on a task is r at the beginning of a time unit and they work for a fraction of the time (b) on this task, then at the end of the time unit their experience will be $\text{TaskExp}_e = \max(r + \frac{\mu}{\phi} \times b, 5)$. Again, μ is the employee's learning speed. The maximum value for experience is the same as that for proficiency, 5.0.

To avoid confusion, it should be noted that the time-line based model uses the term "experience" differently from other existing models. For other models, the categories of experience are very broad, fixed, and measured in years of work experience on a certain scale (e.g. 1–7). This criterion is used to predict schedules for large, long-term activities, so this broader range is appropriate. In the time-line based model, experience categories are specific to tasks and are used to predict completion dates for small, specific activities. Therefore, in this paper, it is appropriate that we allow the experience level to rise during the course of the project.

5.3.5. Employee availability model

Associated with each employee are two dates t_{begin} and t_{end} , which represent the months they first become and then cease to be available for the project. They can only be assigned to a task or training in a time unit t when $\lceil \frac{t}{\phi} \rceil \geq t_{\text{begin}}$ and $\lceil \frac{t}{\phi} \rceil \leq t_{\text{end}}$. This allows the manager to move employees into and out of the project based on external factors and addresses the problem of restricting employee assignment, noted earlier in Section 4.2.

5.4. Task model

A task is also represented as a numerical identifier (ID) and a set of properties. These properties are described in the following sections.

5.4.1. Employee effort estimation

The effort required to complete a task, measured in person-months, must be known before employees can be assigned to the task and the work can be scheduled. However, the experience level of the employees assigned to a specific task will impact the effort required. The new model allows those experience levels to grow during the course of the tasks. Hence, models, which do not account for learning, can not be used without some modification. A solution is to make the effort parameter a variable so that the effort required can be adjusted based on the experience levels of employees. When employees are assigned to a task, their skills, profi-

ciency, and experience levels are known. We then re-estimate the effort using the employees' known properties. The impact of the task assignment on the fitness function value will now be somewhat different, but a GA with multiple individuals in multiple populations will largely compensate for this effect.

5.4.2. Task importance model

Deadlines are often associated with specific tasks of the entire project. In order to address deadline issues, the concept of "importance" of tasks is introduced by assigning three properties to each task: D_{soft} , D_{hard} , and P . The two deadlines are expressed in time units, while the penalty P is expressed as a cost per unit time. There is no penalty incurred if a task is finished before its soft deadline. If it is finished after the soft deadline but before the hard deadline, a penalty of P is paid for every time unit after the soft deadline. If the task cannot be finished before the hard deadline, the penalty is ∞ , which makes the schedule invalid, as any penalties will be included in the project cost. If $D_{\text{soft}} = D_{\text{hard}} = B$ (the final deadline for the project) for all tasks, then there are no intermediate (task-specific) deadlines.

Thus, if the task is finished on time unit M , the penalty will be:

$$\text{penalty} = \begin{cases} 0 & M \leq D_{\text{soft}} \\ P \times (M - D_{\text{soft}}) & D_{\text{soft}} \leq M \leq D_{\text{hard}} \\ \infty & M \geq D_{\text{hard}} \end{cases}$$

5.4.3. Skill list and ancestor tasks

These are the same as in the task-based model and, indeed, as in most scheduling models based on activity networks. A task still has a list of required skill IDs and a list of direct ancestor task IDs. All ancestor tasks (also known as predecessor tasks) must be completed before the task can begin.

5.4.4. Maximum headcount

In reality, only a limited number of employees can work effectively on any given task. As employees are added, the communication necessary to coordinate their mutual effort grows, usually resulting in lower efficiency [41]. Therefore, there should be a limit to the size of the team assigned to a task. In the new model, this limit is set by the property MaxHead . If desired, the user can supply a value that can be used in all later steps. If not, then the limit can be calculated using any existing model.

For instance, in the COCOMO Model [9,10,21,22], the formula to calculate the development time for a task is: $TDEV = 3 \times (PM)^{(0.33 + 0.2 \times (B - 1.01))}$ where PM is the estimated effort in person-months. Setting $B = 1$ in the early design model, we obtain: $TDEV = 3 \times PM^{0.328}$.

Therefore, the estimated average team size for a task should be $\frac{PM}{TDEV} \approx \frac{1}{3} PM^{0.672}$. For the purpose of this study, we decided that doubling this size (or 1 if PM is too small) can be used as the default size limit, i.e. $\text{MaxHead}_{\text{default}} = \max\{1, \text{round}(\frac{2}{3} PM^{0.672})\}$.

More elaborate models of the effort required to perform a task can be developed. However, replacing this algorithm with one more appropriate to a particular type or category of tasks does not change the fundamental approach.

5.5. Employee-task assignment scheme

The assignment scheme in the time-line based model can be represented by a three-dimensional array. The first dimension is time, divided into time steps, from 1 to an upper limit. For example, if the time step size (also called the time unit) is 1 month and the upper limit is 36 months, the time dimension will range from 1 to 36 in units of one month. The second dimension represents the enumerated project tasks, followed

by dedicated training tasks. The third dimension represents employees. Towards the end of this study, this three-dimensional array is populated with the values {0, 25, 50, 75, 100}, which represent the percentage of time the employee will work on the task during each time unit. Fig. 1 illustrates the representation.

5.6. Calculation of the fitness function

In order to retain the positive property (as required by GALib), the fitness function is expressed as $C - \text{TotalCost}$, where C is a sufficiently large constant. In the experiments reported here, the value $C = 10^{10}$ was used. If the value of C exceeded the length of the mantissa of floating point type, this could result in some loss of precision due to rounding. In practice this can be avoided by calculating a problem-specific, smaller value of C using some heuristics. In the work reported here, as a check, the implementation directly computed the final cost. Our “optimal” solution in this problem is the assignment with minimal cost. No significant problems were seen in the experimental results, as will be reported later.

More serious is the general problem of excessive or invalid assignment schemes (e.g., those containing unacceptable overtime for some employee or those with tasks that cannot be reasonably finished by the hard deadline). For such assignments, the fitness is 0. Such restrictions, while making the model more realistic, also serve to fragment the search space and could, at some point in the search, potentially reduce the available gene pool. In addition, because the dimensionality and granularity of the representation have been increased, the gene pool will be spread more thinly throughout the search space. The solution is to somehow maintain the size of the populations and to increase the number of generations (iterations) in the algorithm beyond those used in the original task-based model. The approach selected in the current research was to include a set of heuristics that convert obviously non-conforming assignments to satisfy the constraints of the problem.

5.6.1. Heuristic

The objectives of these heuristics are to allow the populations to search the space quickly without imposing an excessive processing burden or introducing unrealistic constraints. Since the fitness function is computed using the finer-grained time-line representation, they must be applied during each time step because the properties of both tasks and employees evolve overtime.

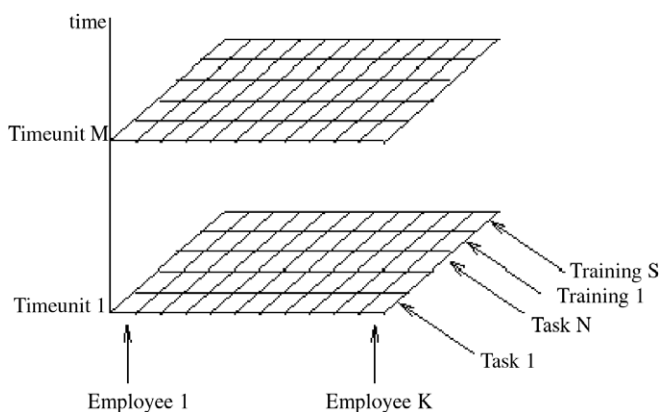


Fig. 1. Assignment scheme in time-line based model.

Heuristic	Explanation
1. Eliminate <i>finished</i> tasks	Remove employees from any task completed in the prior time step (for the definition of finishing, see <i>achievement calculation</i> in Section 5.6.2). In other words, set the corresponding 3-D matrix entries to 0
2. Eliminate <i>untimely</i> tasks	Remove employees from tasks that cannot begin the work during the next time step
3. Eliminate <i>unsuitable</i> assignments	Remove employees from tasks that require skills they do not have at the time the task starts
4. Eliminate <i>unavailable</i> employees	Remove employees from any task on which they become unavailable in the next time step
5. Eliminate contractors from training tasks	Remove any contractors from any training tasks in the next time period
6. Limit employee training	Remove an employee from training tasks if the goal of training is to improve a specific skill for which the employee already has proficiency in excess of 4.5. Certainly, project managers can choose a different threshold
7. Adjust workloads	For each employee, create a list Q of task assignments, sorted by effort expended on each task. The total workload WL of the employee for this time step is the sum of the efforts in the list with respect to the employee. If WL exceeds the maximum value defined for this employee, eliminate the task assignment with the smallest effort for this time step, continue to do so until the new value of WL no longer exceeds the maximum
8. Adjust overstaffed tasks	Examine each task to determine if the team size exceeds its value of $MaxHead$. If $MaxHead$ is exceeded, eliminate the most “unfit” employees from the task until the team size is within the limit. The calculation of “fitness” of an employee e is described as $TaskFit_e$ used in the algorithm shown in Section 5.6.2

After applying these heuristics, the majority of the restrictions pertaining to the model should be satisfied. The assignments may still be invalid in a more subtle way but this is the normal situation when using a GA-approach. The above heuristics are both simple to implement and natural to the scheduling problem.

5.6.2. Achievement calculation

As discussed earlier, the cost drivers of Personnel Capability and Personnel Experience must be calculated twice in the time-line based model. These computations allow for improved performance while the project and individual tasks are being performed. This is done by calculating the “achievement” for each task after each time unit and subtracting this value from the “remaining effort” for the task, which is then used at the beginning of the next time step. If the remaining effort becomes less than or equal to 0, then the task is considered finished. At the start of each task, the

remaining effort is initialized to be the estimated effort. For the purpose of illustration, in the COCOMO II Model, the values of cost drivers for a team–task pair range from 1 to 7, with higher values indicating a linear increase in cost. The calculation of achievement is shown as follows.

5.6.2.1. Algorithm for calculation of achievement in the time-line based model. Given: A task with required skill list S and a list E of employees assigned to it.

1. For each $e \in E$:

$$TaskProf_e = \prod_{s \in S} \frac{SkillProf_{e,s}}{5}$$

where $TaskProf_e$ is the proficiency of employee e for this task, $SkillProf_{e,s}$ is the proficiency of employee e in using skill s . Notice that $TaskProf_e$ is between 0.0 and 1.0.

2. For each $e \in E$:

$$TaskFit_e = \frac{TaskProf_e + \frac{TaskExp_e}{5}}{2}$$

where $TaskFit_e$ is the overall fitness of employee e for this task, $TaskExp_e$ is the experience of employee e for this task. Again, this item is in the range 0.0–1.0.

3. The total fitness F of employee e for this task is:

$$F = \frac{\sum_{e \in E} TaskFit_e \times WorkLoad_e}{\sum_{e \in E} WorkLoad_e}$$

where $WorkLoad_e$ is the work load of employee e for this task during the current time period.

4. Convert F to a cost driver value (1–7, 1 being the most fit) V by:

$$V = 8 - \text{round}(F * 7 + 0.5)$$

if $F = 1.0$ exactly, then set V to 1.

5. The achievement A is then:

$$A = \frac{\sum_{e \in E} WorkLoad_e / 100}{V \times \phi}$$

The division by 100 is necessary because the workloads represent percentages. A is in person-months. ϕ is the time unit parameter.

The algorithm for calculating the fitness function is shown in Fig. 2.

6. Experiments

Our time-line based model was implemented in C++ with GALib [31], an open-source and relatively new toolkit of Genetic Algorithms, in both the Unix systems (Linux and Solaris) and Windows XP environments. In the Windows XP environment, a Tcl/Tk extension package was written to provide a graphical user and database interfaces to the model. Several experiments were conducted to evaluate the correctness and performance of the model.

6.1. Choosing parameters

Population size, generation number, and mutation probability not only influence the time required to perform the GA algorithm but also affect the quality of the result [12,35].

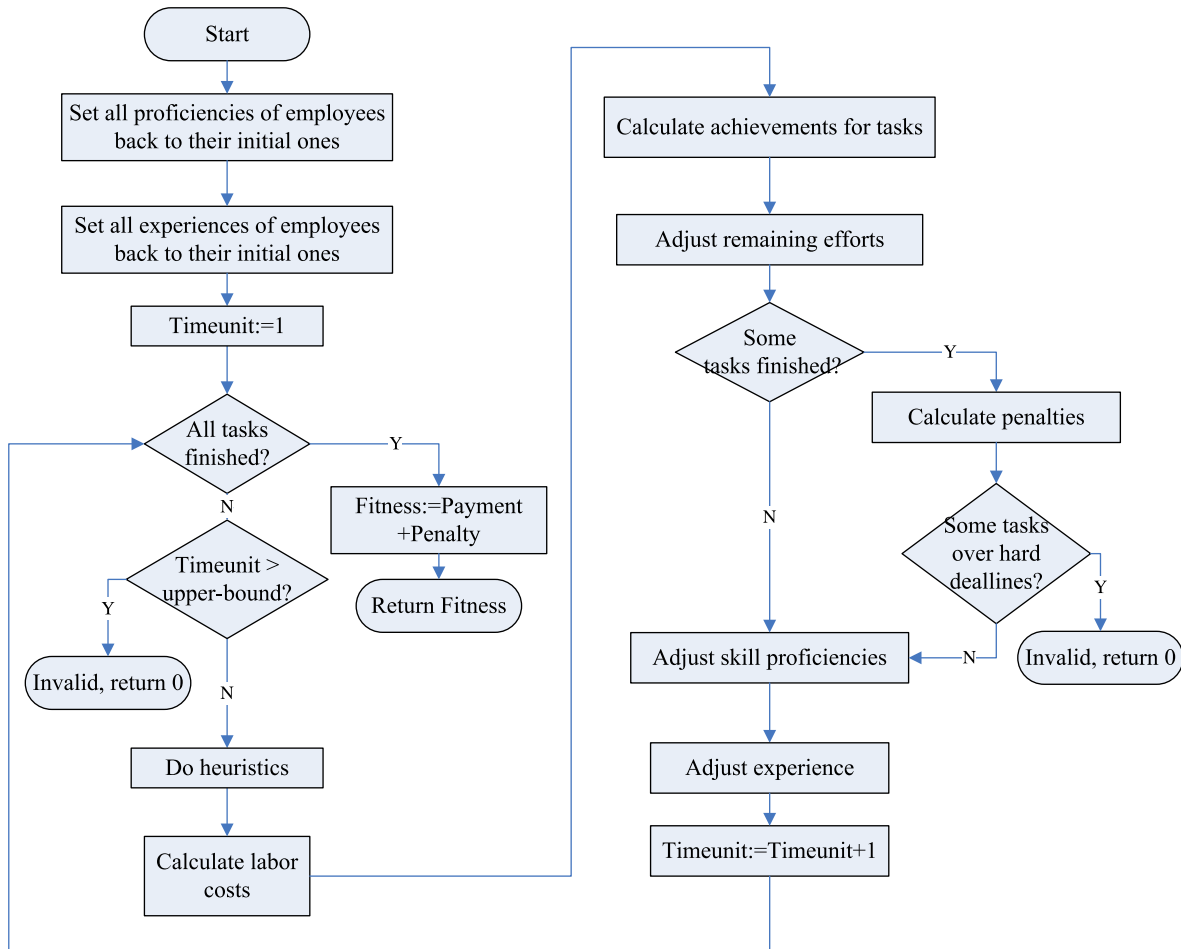


Fig. 2. Calculation flow of fitness function for the time-line-based model.

First, the influences of population size and generation number on the near-optimal solutions were examined in order to determine if the algorithms exhibited any unexpected behaviors, as well as to determine the sizes of later, larger scale tests. Only for the purpose of illustration, an initial set of experiments was performed using small employee- and task-sets with three employees, three tasks, and three skills. The time unit chosen was one month, with an upper limit of 10 months. Table 1 shows the result obtained on this experiment.

From the table, we learn that population-size = 1000 and generation-number = 1000 offered the best results for this small experiment. Although a larger population size and generation number can definitely result in better performance, overall it is still reasonable to expect that those two parameters can be set to 1000 to get positive results for later larger experiments.

The influence of mutation probability on the results of the GA calculation must also be considered upon “fine-tuning” the algorithm. For example, consider an experiment in the later section with parameters of time-unit = week, upper time limit = 60 weeks, population size = 200, generation number = 1000, and a variety of mutation probabilities (see Table 2).

Table 2 suggests that under this scenario, small mutation probabilities will produce better results than larger ones. Such a phenomenon is not unique to this problem or to its implementation. Because the purpose of mutation is to prevent the pre-maturing of GAs, higher mutation probabilities may produce a greater percentage of “unfit” offspring, even when heuristics are used to improve their likelihood of survival. Since the scheduling problem has many restrictions, a high mutation rate would indeed produce a high proportion of such “doomed” offspring. In the experiments reported herein, mutation probabilities between 0.001 and 0.002 produced the best results. Accordingly, the default mutation probability was taken to be 0.001 for the remainder of the work. In fact, most of the GA-related work also utilizes this value. And, as it can be seen in most of the literature, the mutation rate is related to the population size or the number of digits of a genome [34,35].

Constraints (such as tasks with tight, hard deadlines) serve to restrict the search space, making it correspondingly more difficult to develop a large enough population for adequate search space exploration. Severe constraints can adversely influence the performance of the Genetic Algorithm, as each iteration produces few satisfying offspring. As a result, even after many generations, the Genetic Algorithm may not be able to find a good solution or even feasible solution for the simple reason that constraints limit its ability to explore the search space. If constraints can be relaxed, the successive generations of the population can have a larger number of good genomes that span a larger portion of the search space. In such cases, the Genetic Algorithm often produces a final solution that can satisfy the original tight restrictions. For example, in one experiment, when the upper time limit is set to 70, the algorithm was unable to find any feasible solutions. However, with an

Table 1
Performance analysis with population size and generation number in a small example

Population size	Generation number	Best cost
500	100	22528
1000	100	21504
2000	100	18432
3000	100	23552
4000	100	20480
5000	100	22528
1000	500	14336
1000	1000	9216
1000	2000	9216
2000	500	12288
2000	1000	11264

Table 2
Mutation probability changes

Mutation probability	Best cost
0.1	84118528
0.05	85620736
0.01	81460224
0.001	34738176
0.0001	37011456

upper bound of 80, a solution was found with total duration 60, which was less than the earlier, more severe constraint of 70.

6.2. Experimental results

Several large and small experiments were conducted. One consisted of 15 tasks for which 10 employees were available. The employees, in turn, each possessed five skills to a greater or lesser extent of proficiency. Each of the five skills was needed for at least one task and many tasks required multiple skills. It was not immediately obvious what the optimal assignments would be but the algorithm, at the very least, found feasible and near-optimal solutions. In particular, it can be seen that the Genetic Algorithm vastly improved the fitness of the solution from the beginning to the end, and the final solution did not violate the skill restrictions. The detailed scenario and the results of this experiment are given as follows.

6.2.1. Employees

Table 3 shows the properties of the employees available to the project and Table 4 gives their proficiencies in the five skills necessary to complete the project.

6.2.2. Tasks

Fig. 3 shows the dependencies among tasks and Table 5 describes the properties of the tasks.

Table 3
Employee properties

Employee ID	Is contract	Salary	Incr	Max hour	Experience	Speed	Start	End
1	N	6000	100	150	4.8	1.5	1	30
2	N	5300	0	100	4.7	1.3	1	30
3	Y	4800	0	100	4.6	1.3	1	30
4	N	5000	0	75	4.7	0.9	1	30
5	N	5000	0	50	4.6	1.1	1	30
6	N	5800	100	125	4.8	1.4	1	30
7	N	5800	0	100	4.8	1.5	1	30
8	N	5000	0	100	4.5	1.0	1	30
9	Y	4600	0	75	4.3	1.2	1	30
10	N	5300	0	50	4.6	1.3	1	30

Table 4
Employee proficiencies

Employee ID	Proficiency				
1	4.5	5	0	5	4.8
2	0	0	3.5	0	4.8
3	4.3	4	3.5	0	4.8
4	0	4.7	0	0	0
5	4.5	4	3.8	0	5
6	0	4.5	4.3	4	0
7	4.5	4.8	5	0	0
8	0	0	0	4.5	4.6
9	0	0	3.9	4.8	0
10	4.7	4	0	3	4

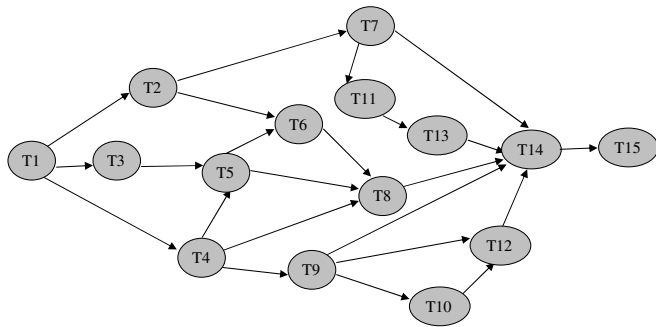


Fig. 3. Task precedence graph.

Table 5
Task descriptions

Task ID	Effort	Soft deadline	Hard deadline	Penalty	Max Head	Skills
1	0.25	0	4	1000	2	1 3
2	0.5	8	12	1000	2	2 3 5
3	0.8	12	15	2000	3	1 2 4 5
4	0.25	0	0	0	3	1 2
5	0.6	0	0	0	3	1 3
6	0.5	0	0	0	3	3 4
7	0.3	0	0	0	2	2 4 5
8	0.4	24	32	2000	2	2 3
9	0.25	0	0	0	2	2 4
10	0.5	0	0	0	2	4 5
11	0.5	0	0	0	2	1 3 5
12	0.25	0	0	0	2	2 4
13	0.8	0	0	0	3	2 3 4
14	0.5	40	52	3000	2	1 2 3 5
15	1	52	60	5000	3	2 3 4 5

6.2.3. Results of the experiment

In this experiment, the time unit was given to be one week, with an upper limit of 60 weeks. A mutation probability of 0.001, a population size of 1000, and a maximum of 1000 generations were used to guide the genetic algorithm. The final (lowest) cost was 25940992. The evolution process is shown in Fig. 4.

6.2.4. More experiments via Experts' efforts

In order to evaluate the new model, we decided to compare GA performance against that of humans. A hypothetical smart home project (Appendix A) was designed for this purpose. We chose this project design because it presented real-life meanings to the project parameters, which would otherwise be too abstract for an expert to do a good job. There are 25 tasks, 12 kinds of skills, and 15 employees available for the project. We recruited two senior software project managers (PM1 and PM2) to assign the employees to tasks. The results we obtained from the managers are shown in Figs. 5 and 6, respectively. Before recently retiring, PM1 had 25

years of experience in software development and over 15 years of project planning and management experience in large-scale telephony systems. PM2 worked on software development projects for 30 years and managed projects for about 20 of those years. He managed over 30 projects ranging in duration from three months to two years with the majority at 18 months. Most of the projects were telecommunications systems; others were real-time, embedded systems such as vehicle location systems.

The experts shared the assumptions they made when arranging schedules. For example, PM1 was less concerned than PM2 about costs when she assigned employees to tasks. However, she paid more attention to the skill matching of the employees. In the end, both experts attempted to arrange the shortest schedules.

Based on their work, we obtained costs of 546353 and 559920, respectively. We ran our model by fixing the mutation probability to 0.001 and the population size to 1000 adjusting the number of generations from 1000 to 2000. The average time the experts spent developing the assignments was 3.0 h excluding preparation time (i.e., the time to understand the project). In contrast, the average run time for our program was only 33 min. In other words, the time it took to generate an assignment by running the program was much faster than the time to generate an assignment by a human-being including the experts. Moreover, the minimum cost of these experiments was 385818. Obviously, the GA outperformed both experts. However, the average cost of all of the experiments we ran was 527104, which was slightly below the experts' results. Two reasons can be cited for the discrepancies (1) the search space of our model included the two schedules generated by the experts and (2) only one expert was concerned with the cost consideration. Hence, the costs of the two schedules were not minimal values. Based on these results, the minimum cost of the experiments could be treated as a lower bound of project cost and the average cost of the experiments could be a useful reference to managers.

In conclusion, the schedules created by the experts were already acceptable but the results from the GA program outperformed the experts' assignments. Therefore, we believe that in order to arrange an itinerary with reduced cost, the schedules generated by our model can be an auxiliary schedule for managers who may or may not have experience in project management. In order to achieve this, the managers can compare the differences between the two schedules (model and self generated), and make some adjustments to the final schedules. Since our model also considers the skills and on-the-job training of employees, the schedule can also help managers inspect the shortage of their assignments, while rearranging those that are already believed to be suitable.

7. Comparisons and discussions

A number of researchers suggested certain kinds of problems where GAs work better than other heuristic methods and provided the criteria to compare different problems and algorithms, such as [32,33]. Mostly, the comparisons focused on GA and hill-climbing. Here, we compare the performance between GA and hill-climbing in this project management environment.

The hill-climbing algorithm that we used works as follows: A population of initial solutions is chosen where the best one is determined as the starting point. This foremost solution is mutated at a randomly chosen single locus and the fitness is evaluated. If the mutation leads to a higher fitness, the new solution replaces the old one. The procedure continues until the optimum is found.

Usually, hill-climbing algorithms are much faster than GA to reach optima. However, the landscape in this problem has many local optima which render the achievement of a global optimum a difficult task for the hill-climbing algorithm. Compared to the same case discussed in the previous section, the mean cost computed by

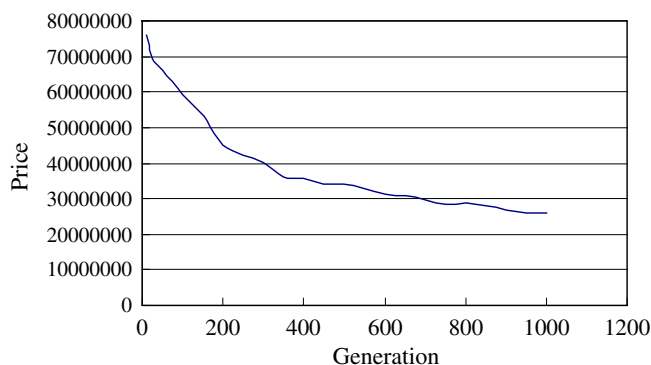


Fig. 4. Evolution process.

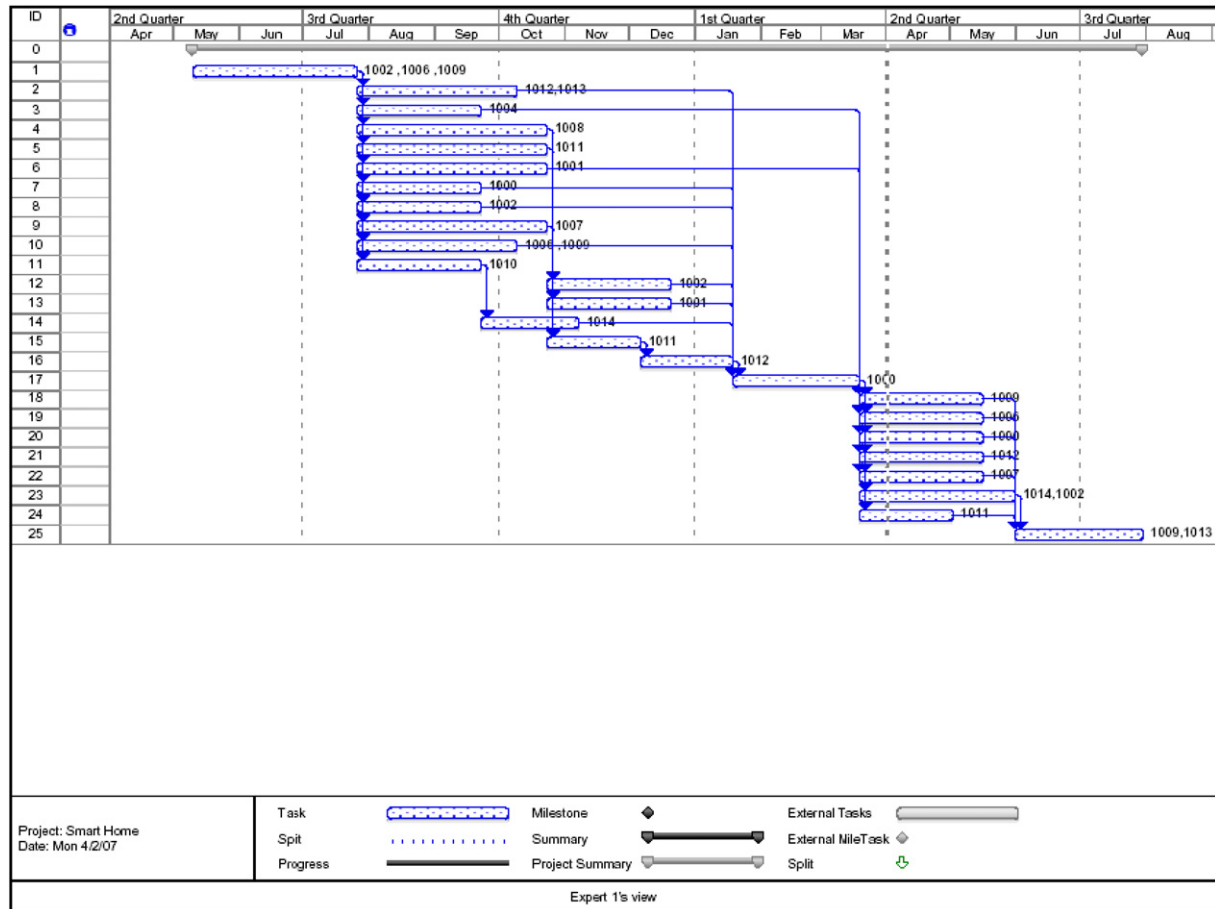


Fig. 5. Assignment of expert 1.

hill-climbing was 34622840 with (initial populations = 1000) while the mean using the GA is 27635360, which outperforms the best fitness achieved by hill-climbing as shown in Table 6.

Using GA to analyze the computational procedure, we can see from Fig. 3 that GA is not efficient in the later stage of computation which, in essence, means it converges very slowly. With GA's ability for global optimization and hill-climbing's ability to locally optimize, a hybrid algorithm combining the two may be the ideal choice.

The table below gives the results from the experiment with two cases. Table 7 shows the result of GA (mutation probability = 0.001, generation number = 1000, population size = 1000) and GA (generation number = 500) with hill-climbing in a small experiment. Table 8 shows the result of GA and GA with hill-climbing in a relatively larger project scheduling problem.

In our model of software project scheduling, on relatively smaller problems, hill-climbing is much better than GA both in time and fitness evaluation. On the other hand, GA will outperform hill-climbing when the problems become more complicated; GA with hill-climbing does not help much in optimization of larger problems. Certainly, we are aware of many real-world considerations that may further complicate the environment. For example, career growth is refined for each employee so that skill match takes a different spin. We consider those "expectations" as more refined constraints that will only enlarge the search space and extend the search time.

8. Conclusions and future work

As explained earlier, traditional and widely accepted project management techniques fail to cope effectively with the evolutionary and dynamic nature of modern software projects. In addition,

these techniques do not deal with large-scale optimization problems. This paper describes a scheduling model that includes some of the nuances that arise in planning actual projects and a genetic algorithm to find optimal and near-optimal solutions. Compared to the efforts by project management experts, our model using GA appeared to be a viable tool to help guide project managers in their daily routines.

Potential topics for future work include the following improvements to the model:

1. Better system dynamics integrated with the modeling of progress of tasks where better training and experience models are included. Can we better model the impact of the synergies between multiple skills of a single individual when they are simultaneously applied to a task, as well as the combined effect of the skills of multiple employees when applied to a task? Moreover, can we explore the impact of team size on these topics?

2. Better representation of skills, employees, and tasks. Can we cluster skills into classes of similar levels of difficulty? Can we introduce different learning rates based on the difficulty, and allow for learning synergies within the classes? Employees can be divided into categories with preferred skills and predefined groupings for training associated with each category. By carefully planning training sessions and task assignments associated with each category, we can accommodate career advancement along a pre-defined sequence of tasks. We can also define categories of tasks and assign levels of criticality of skills to tasks, thereby effectively creating a default list of skills and minimum proficiency levels required. While not a fundamental change to the representation, it would be highly desirable to simplify the task of data entry, which can be quite formidable for the project manager.

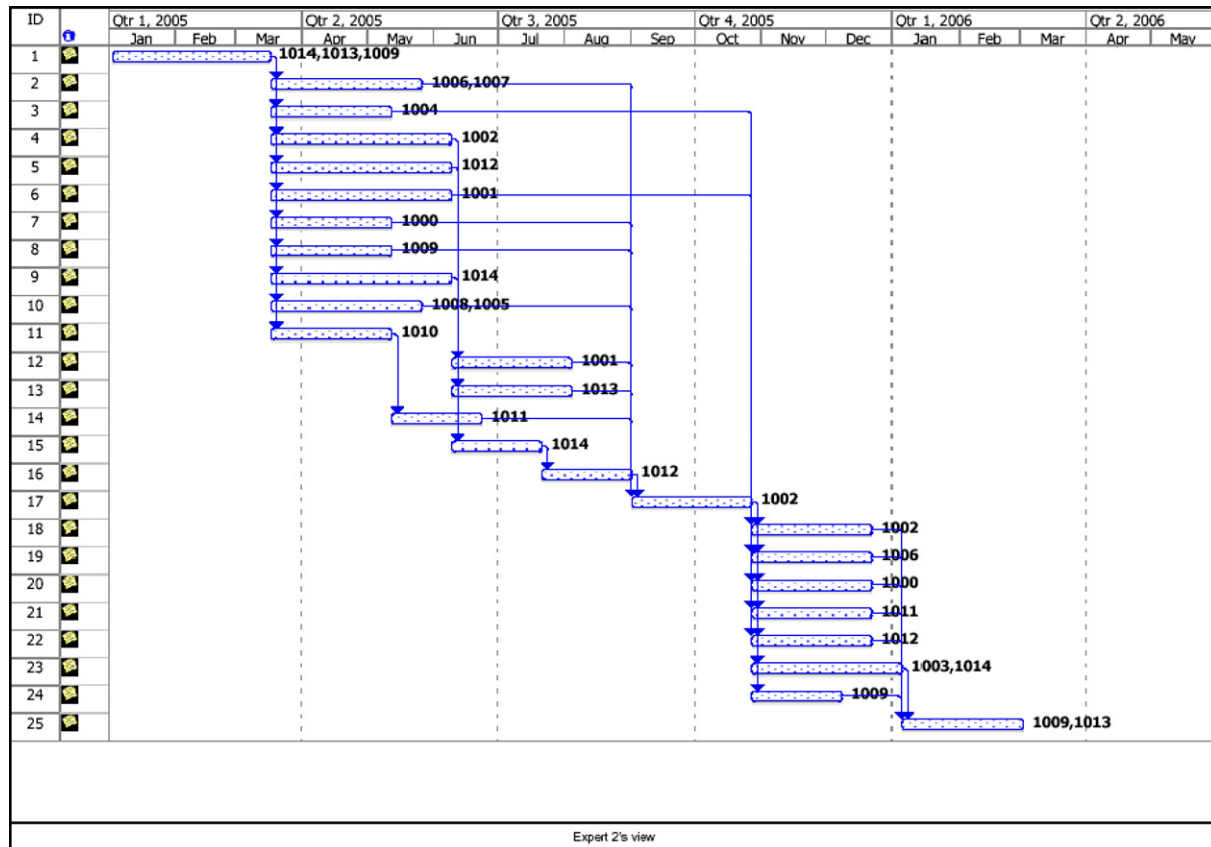


Fig. 6. Assignment of expert 2.

Table 6
Comparison between GA and hill-climbing

	Best	Mean	Worst
Steady GA	25424896	27635360	29941800
Hill-climbing	29883000	34622840	41868400

Table 7
Comparison between GA and GA with HC in a relative small problem (3 tasks, 3 employees)

	Best	Mean	Worst
Steady GA	8192	11298	15360
Steady GA (500 generations) with HC	5286	6983	10024

Table 8
Comparison between GA and GA with HC in a relative big problem (15 tasks, 10 employees)

	Best	Mean	Worst
Steady GA	25424896	27635360	29941800
Steady GA (500 generations) with HC	26761000	28929233	33171600

3. Flow of control options that allow the project manager to easily explore alternative project plans needs to be examined. Some selected tasks can be immediately executable, even if their predecessor tasks are not complete; some selected tasks may be required to be re-done to recover from failure of a specific activity, such as a design review, or a test. The concept of probability can be added to these two options.

4. Explore more search heuristics, such as Tabu search and simulated annealing, and compare their performances in software project scheduling.

5. Consider building a user interface to allow for more convenient feedback from the manager as the project evolves, and to display the status of an on-going project in a manner more suitable to the model. In other words, on top of the GA engine (GALib in our implementation), we can implement a suitable interface for project management. Of course this will become a very extensive project requiring knowledge in other domains such as HCI usability and design.

Acknowledgments

Dr. Mark Christensen, who retired from Northrop Grumman as Vice President of the EDS division, provided very constructive comments during the initial stage of this study. Software project managers Jean Chang, who has over 15 years of solid project planning and management experience in the practitioner's world, and Owen Lavin, who worked on software development projects for 30 years and managed projects for about 20 of those years, invested significant effort in submitting their favorite schedules based on our hypothetical smart home project. We also thank Jane Cleland-Huang for her valuable comments of revising and editing the paper.

Appendix 1. Smart home project

A.1. Purpose and overview of the project

The purpose of Smart Home Project is to develop a home where most devices are controlled by computers so that it can help the disabled or elderly people live independently without the need for an

assistant to be nearby. That is, one of the smart home's primary objectives for the elderly is to enable them to retain their status as an independent homeowner for as long as possible before making the transition into the health care system (e.g., nursing homes). Inside the home, there are sensors to detect the activities of the homeowners and based on their specific actions, or lack thereof, the smart home will react accordingly to deliver necessary automated assistances, including, but is not limited to: medicine application, cooking and general proactive and reactive safety measures.

One exemplary feature of the smart home is its preset automation. Users and researchers alike can define specific context information (defined events, environmental fluctuations, etc.) where the smart home will be triggered into action. One principal example of this is the house's lighting. Depending on the existing amount of natural and synthetic light in any given room, as well as the preset preference of user-defined brightness, the home will adjust window coverings and synthetic light levels to accommodate to the specifications. As a matter of fact, this feature has already been designed and implemented in the Smart Home Lab and at the Department of Computer Science, Iowa State University. Scenarios similar to this can be considered in many other devices within the smart home.

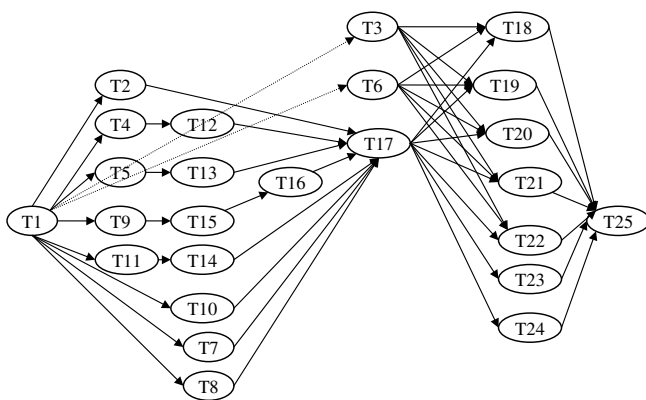
A.2. Features of smart home framework

We focus the features of the smart home framework on three devices (lighting system, motion detector, and a RFID-based heart rate monitor) and one set of actors (researchers). A key point to note is that most of these proposed features are generic and can be easily ported to devices that will be included in future iterations. All of the control software will accept commands from devices such as cell phones, PCs, controllers, etc. This will ensure a greater ease of use for the user in any scenario. Inversely, these devices will also have the capability to receive commands from the control software.

A.2.1. Features of application

The following is an abbreviated list of smart home features to illustrate the expansive breadth of control capability. The general features cover a wide spectrum such as (1) basic control of devices; (2) features on lighting, doors, administration, medication, floors, etc; (3) safety; (4) security and (5) reliability.

The following is the Task Precedence Graph (TPG) extracted from the smart home requirements and its explanation.



Skills:

1. Operating system
2. Programming
3. Network-security

4. Network-wireless (802.11)
5. Network-TCP/IP
6. Math-probability
7. Math-optimization
8. Database
9. Human-computer interaction
10. Language(English)
11. Engineering Basic
12. Network-H.323 (IP Telephony)

Our assumption for the developers' skill levels is that, basically, each skill can be treated as an independent one (that is, there is no dependency between any two skills.) However, we do consider the dependency between skill 4 and skill 5 as it makes sense.

Tasks:

- T1: Design and set up the system server
Skill 1, 2, 5, 8 person months, 3 *MaxHead*
- T2: Set up the security system (Lock system/network system)
Skill 2, 3, 5, 5 person months, 2 *MaxHead*
- ...
- T23: Connect some of the devices to the motion sensors so that the motion of the occupant can be one of the inputs
Skill 1, 5, 11, 5 person months, 2 *MaxHead*
- T24: Connect some of the devices to the sensors so that whatever the sensors get can be the inputs
Skill 1, 5, 11, 1.5 person months, 1 *MaxHead*
- T25: Integrate all the tasks and embed the manual inside the system
Skill 1, 2, 5, 10, 4 person months, 2 *MaxHead*

References

- [1] Project Management Software, <http://www.startwright.com/project1.htm>.
- [2] C. Chao, Software project management net: a new methodology on software management, Ph.D dissertation, University of Illinois at Chicago, Department of Electrical Engineering and Computer Science, 1994.
- [3] K. Schwaber, M. Beedle, Agile Software Development with Scrum, Prentice-Hall, 2002.
- [4] C. Chang, M. Christensen, A net practice for software project management, IEEE Software (1999) 80–88.
- [5] C. Chang, M. Christensen, T. Zhang, Genetic algorithms for project management, Annals of Software Engineering 11 (1) (2001) 107–139.
- [6] D. Fogel, Evolutionary Computation: Toward a New Philosophy of Machine Intelligence, IEEE Press, 1995.
- [7] M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, 1996.
- [8] Weng-Tat Chan, David K.H. Chua, Govindan Kannan, Construction resource scheduling with genetic algorithms, Journal of Construction Engineering and Management (1996) 125–132.
- [9] I. Sommerville, Software Engineering, eighth ed., Addison-Wesley, 2006.
- [10] S. Kirkpatrick, C. Gelatt, M. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.
- [11] F. Glover, Tabu search | part I. ORSA, Journal on Computing 1 (1989) 190–206.
- [12] J. Holland, Adaptation in natural and artificial systems, University of Michigan Press, Ann Arbor, 1975.
- [13] M. Wall, A genetic algorithm for resource-constrained scheduling, Ph.D dissertation, MIT, June 1996.
- [14] Daniel D. Galorath, Michael W. Evans, Software Sizing Estimation and Risk Management, CRC Press, 2006.
- [15] A.J. Albrecht, Measuring application development productivity, in: Proceedings of the Joint SHARE/GUIDE/IBM Application Development Symposium, October 1979, pp. 83–92.
- [16] David Garmus, David Herron, Function Point Analysis: Measurement Practices for Successful Software Projects, Addison Wesley, 2001.
- [17] A. Felfernig, A. Salbrechter, Applying function point analysis to effort estimation in configurator development, in: International Conference on Economic, Technical and organisational aspects of Product Configuration Systems, Copenhagen, Denmark, 2004, pp. 109–119.
- [18] A.J. Albrecht, J.E. Gaffney, Software function source lines of code and development effort prediction, IEEE Transactions on Software Engineering (1983) 639–648.
- [19] Frank Niessink, Hans van Vliet, Predicting maintenance effort with function points, in: International Conference on Software Maintenance, 1997, pp. 32–39.
- [20] The software maintenance project effort estimation model based on function points, Journal of Software Maintenance: Research and Practice 15(2) (2003) 71–85.

- [21] Barry Boehm, Software Engineering Economics, Prentice-Hall Inc., 1981.
- [22] Barry Boehm et al., Software Cost Estimation with COCOMO II, Prentice-Hall PTR, 2000.
- [23] Martin Shepperd, Chris Schofield, Estimating software project effort using analogies, *IEEE Transaction on Software Engineering* 23 (11) (1997) 736–743.
- [24] Sun-Jen Huang, Nan-Hsing Chiu, Optimization of analogy weights by genetic algorithm for software effort estimation, *Information and Software Technology* 48 (11) (2006) 1034–1045.
- [25] Nan-Hsing Chiu, Sun-Jen Huang, The adjusted analogy-based software effort estimation based on similarity distances, *Journal of Systems and Software* 80 (4) (2007) 628–640.
- [26] Parag C. Pendharkar, Girish H. Subramanian, James A. Rodger, A probabilistic model for predicting software development effort, *IEEE Transaction on Software Engineering* 31 (7) (2005) 615–624.
- [27] K.S. Hindi, H. Yang, K. Fleszar, An evolutionary algorithm for resource-constrained project scheduling, *IEEE Transactions on Evolutionary Computation* 6 (5) (2002) 512–518.
- [28] Vicente Valls, Francisco Ballestin, Sacramento Quintanilla, A hybrid genetic algorithm for the resource-constrained project scheduling problem, *European Journal of Operational Research* 185 (2008) 495–508.
- [29] Enrique Alba, J. Francisco Chicano, Software project management with GAs, *Information Sciences* 177 (2007) 2380–2401.
- [30] Valentina Plekhanova, On project management scheduling where human resource is a critical variable, in: *Proceedings of the Sixth European Workshop on Software Process Technology (EWSPT-6)*, Lecture Notes in Computer Science series, Springer-Verlag, London, UK, pp. 116–121.
- [31] M. Wall, Lib: A C++ Library of Algorithm Components, <http://lancet.mit.edu/ga/>.
- [32] M. Mitchell, J. Holland, S. Forrest, When will a genetic algorithm outperform hill-climbing? in: *Advances in Neural Information Processing Systems 6* (7th NIPS Conference), Denver, Colorado, 1994, pp. 51–58.
- [33] J. Horn, D. Goldberg, Genetic algorithms difficulty and the modality of fitness landscapes, *Foundations of Genetic Algorithms* 3 (1995) 243–269.
- [34] Hans-Georg Beyer, Hans-Paul Schwefel, Ingo Wegener, How to analyse evolutionary algorithms, Technical Report No. CI-139/02, University of Dortmund, 2002.
- [35] Jackie Rees, Gary J. Koehler, Learning genetic algorithm parameters using hidden Markov models, *European Journal of Operational Research* 175 (2) (2006) 806–820.
- [36] T. Hegazy, K. Petzold, Genetic optimization for dynamic project control, *Journal of Construction Engineering and Management* 129 (4) (2003) 396–404.
- [37] David A. Coley, *An Introduction to Genetic Algorithms for Scientists and Engineers*, World Scientific Publishing Co. Pte. Ltd., New Jersey, 1999.
- [38] Harold Kerzner, *Project Management Best Practices: Achieving Global Excellence*, John Wiley & Son, Inc., New Jersey, 2006.
- [39] Harold Kerzner, *Project Management: A Systems Approach to Planning, Scheduling, and Controlling*, John Wiley & Son, Inc., New Jersey, 2003.
- [40] G. Abu, J.W. Cangussu, A learning model for software development processes, *Software Engineering (SE)* (455) (2005).
- [41] Frederick P. Brooks, *The Mythical Man-Month: Essays on Software Engineering*, Addison Wesley, Boston, 1995.