Optimized Resource Allocation for Software Release Planning

An Ngo-The and Günther Ruhe, *Senior Member, IEEE*

Abstract—Release planning for incremental software development assigns features to releases such that technical, resource, risk, and budget constraints are met. A feature can be offered as part of a release only if all of its necessary tasks are done before the given release date. We assume a given pool of human resources with different degrees of productivity to perform different types of tasks. In the context of release planning, the question studied in this paper is how to allocate these resources to the tasks of implementing the features such that the value gained from the released features is maximized. Planning of software releases and allocation of resources cannot be handled in isolation. To address the inherent difficulty of this process, we propose a two-phase optimization approach called $\text{OPTIMIZE}_{\text{RASORP}}$ that combines the strength of two existing solution methods. Phase 1 applies integer linear programming to a relaxed version of the full problem. Phase 2 uses genetic programming in a reduced search space to generate operational resource allocation plans. The method is evaluated for a series of 600 randomly generated problems with varying problem parameters. The results are compared with a heuristic that locally allocates resources based on a greedy search.

Index Terms—Release planning, resource allocation, software project management, incremental software development.

1 Introduction

Incremental software development offers products in releases where each release provides additional or modified functionality compared to the previous release. Incremental delivery of software products provides business value earlier and allows for quicker reception of customer feedback. Each release is a collection of features forming a complete system that will be of value to the customer at the time of delivery. This value might change over time. A major problem faced by companies developing or maintaining large and complex systems is determining which elements of a typically large set of candidate features should be assigned to which releases of the software. In addition, there is the question of how to assign resources accordingly. The solution of this complex problem is of pivotal importance for project success. Without good release planning, critical features are not provided at the right time. This might result in dissatisfied customers, time and budget overruns, and decreased competitiveness in the marketplace.

One of the key limitations of current release planning methods is the lack of a systematic process to balance the appropriate delivery of features with the resources available. Greer and Ruhe [17] and van den Akker et al. [40] consider human resources necessary for release only in a cumulative way (i.e., not entering the planning details such as who does what). All other existing release planning models and techniques consider only the cumulative effort or ignore even this (for a more detailed analysis, see [38]).

Release and resource decisions depend on each other. Defining releases without looking into the necessary resources means that the proposed plans are unlikely to be feasible. On the other hand, planning resources without considering the business impact of the product releases can result in missed opportunities for value creation.

In this paper, we augment the scope of release planning by examining details of the resource allocation necessary to actually develop the features. For that, we assume that each feature is decomposed into a sequence of tasks such as design, implementation, and testing. These development tasks can be defined to an even more fine-grained level. In addition, managerial support and/or other tasks can be considered here as well. For these tasks, a pool of human resources is available. For the sake of simplicity, we will refer to these human resources as developers.

It is well known that productivity may vary significantly among developers [1], [28]. Each developer might have different productivity when performing different types of tasks. Good resource allocation in this context takes on even more importance. The resulting problem of optimal assignment of resources to realize the features of a sequence of releases is called Resource Allocation for Software Release Planning, abbreviated by RASORP.

The paper is subdivided into eight sections. Section 2 describes the resource allocation problem in software release planning. A formal problem description is given, which allows the application of optimization algorithms exploiting the special structure of the problem. We show that the problem belongs to the class of NP-complete problems and discuss related results. The fundamental difficulties of the problem are illustrated in Section 3 by a hypothetical case study from the telecommunications

• A. Ngo-The is with Expert Decisions Inc., 54 Royal Oak Cove NW, Calgary, Alberta T3G 4X7, Canada.
E-mail: ngothean@expertdecisions.com.
• G. Ruhe is with the Software Engineering Decision Support Laboratory, University of Calgary, 2500 University Drive NW, Calgary, Alberta T2N 1N4, Canada. E-mail: ruhe@ucalgary.ca.

Manuscript received 3 Apr. 2007; revised 7 Jan. 2008; accepted 7 Feb. 2008; published online 24 Sept. 2008.
Recommended for acceptance by H. Muller.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-2007-04-0123.
Digital Object Identifier no. 10.1109/TSE.2008.80.

domain. The computational complexity of RASORP is addressed by a specialized two-stage optimization method combining integer linear programming and genetic algorithms presented in Section 4. The method is illustrated in Section 5 by the hypothetical case study introduced in Section 3.

The performance of the method was initially evaluated for a series of 600 randomly generated problems with varying problem parameters. For problems with up to 200 features and including up to 600 tasks, the results are compared with a greedy heuristic that makes a locally best choice at each stage of the process. The heuristic plays the role of a baseline procedure as it is assumed to reflect the current state of the practice in making resource decisions. In addition, we compared $\text{OPTIMIZE}_{\text{RASORP}}$ with the direct application of GAs. The results of these investigations are reported in Section 6. In Section 7, we discuss threats to the validity of the results. Finally, Section 8 provides a summary and offers possibilities for future research.

2 Problem Statement and Related Results

2.1 Problem Description

This section discusses the key dimensions of the problem with a semiformal approach. A more formal description is given in Section 4.

2.1.1 Features and Their Assignment to Releases

We consider a feature to be "a logical unit of behavior that is specified by a set of functional and quality requirements" [41]. Features are an essential abstraction that both customers and developers understand. Managing features is then more efficient than directly managing requirements.

In the context of making feature-related release decisions, we also examine the assignment of resources to the tasks necessary to realize these features. Both types of questions are relevant and depend on each other. At the stage of planning, we assume a number of N candidate features $f(1)..f(N)$. The number N of potential new (or revised) features is typically larger than what can be developed with the available resources. The planning horizon K is a parameter telling us how many releases in advance the planning must be conducted. Each release k $(k = 1..K)$ will be made available at predefined release dates t(k). Often, planning is done by looking at the next immediate release $(K = 1)$. However, it is also important for potential customers to know what is planned for future releases.

For each of the features, the question becomes the following: Is the feature planned for one of the next K releases or, if it is not important enough, will it be postponed? More formally, we provide the following definition:

Definition 1. *A release plan is an assignment of features to releases. It is formally described by a matrix* x *of decision variables* x(n, k) *with*

$$\begin{aligned} x(n,k) &= 1 \text{ if and only if feature } f(n) \text{ is offered at} \\ &\text{release } k \ (k = 1..K) \ (\text{and } x(n,k) = 0 \text{ otherwise}). \end{aligned} \tag{1}$$

Assignment of features to releases cannot be done without considering different types of feature dependencies. There are different types of dependencies, as stated by Carlshamre et al. [6], which might impact the planning process. We consider the two most important ones: coupling and precedence relationship among features.

Coupling of features means that a pair of features only makes sense in combination with each other. We assume that coupling will be handled in advance by integrating the respective features into a new more comprehensive whole. Precedence means that a feature is not useful in a release without having another feature already implemented. The precedence relationship is formally defined as follows:

Definition 2. *Feature* f(i) *precedes feature* f(j) *(written as* $(i, j) \in P$*) means that if feature* f(i) *is made available at release* k*, then* f(j) *cannot be made available at any release* $1..k-1$ *earlier than* k*. We will say that features* f(i) *and* f(j) *are in a precedence relationship.*

2.1.2 Planning Objectives

What actually constitutes the planning objective needs careful consideration and cannot be answered in general terms. Often, the objective is to optimize a combination of criteria such as the product's business value, time to market, and risk. Furthermore, there is a dependency of the actual release when the feature is offered. Usually, the cumulative business value created by a feature is larger when it enters the market sooner; perhaps because a product feature might be available before those of a competitor.

For our purposes, we assume a utility function value F(x) expressing the cumulative business value of all features assigned to releases according to plan x. F(x) is composed of individual values v(n, k), where v(n, k) describes the expected value contribution of feature f(n) in case it is assigned to release k. Each value v(n, k) itself is composed from the priorities assigned to features by the (weighted) stakeholders. The prioritization can be done with respect to not only one but a portfolio of (weighted) criteria. Sample prioritization criteria are the following: the estimated market value of feature f(n), urgency of feature f(n), customer satisfaction, and estimated market opportunity. For this paper, we assume that value estimates v(n, k) are given by the project manager. For further details on the composition of the coefficients v(n, k), we refer to [36]. A concrete example of the definition of values v(n, k) is presented in the case study (Table 3).

We will later compare the quality of plans achieved from different types of algorithms.

Definition 3. *The quality of a release plan* x *is described by the degree of optimality that* x *achieves with respect to utility function* F(x)*.*

2.1.3 Feature-Related Tasks

The realization of each feature requires a sequence of tasks. We assume Q different types of tasks. These tasks correspond to the fundamental technical, managerial, and support contributions necessary to develop software. Minimally, these tasks cover design, implementation, and testing. The definition of what constitutes a task is flexible. There might be further differentiation within an activity. For example, testing

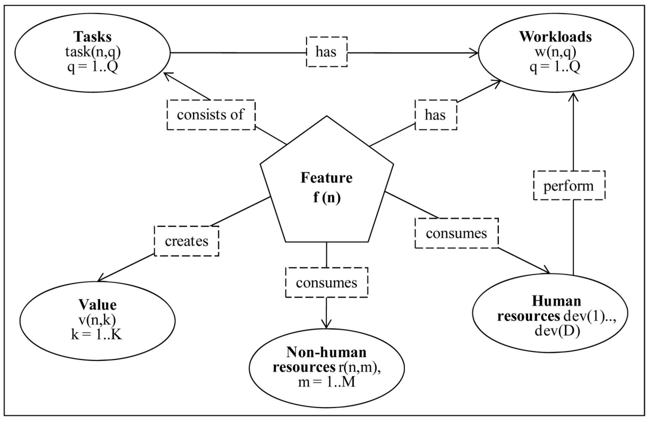

Fig. 1. Data related to a feature f(n).

could be refined into unit testing, integration testing, acceptance testing, and regression testing. While, in principle, the granularity of the definition of a task is flexible, we have to keep our model reasonable in size.

2.1.4 Resources

Human resources (e.g., different types of developers, analysts, external collaborators, etc.) are intended to perform the tasks needed to create the features. For each individual task task(n, q) of type q necessary to provide feature f(n), a workload w(n, q) is defined as the effort needed to fulfill the task. The time needed to perform a task depends not only on the workload but also on the productivity of the developer assigned to this task.

Our resource-allocation process addresses the issue of assigning developers to tasks. We assume that each developer is working on just one task at a time. An additional assumption is that only one developer is allowed to work on one task. In the case where different developers are allowed to work on the task, the original task would need to be further decomposed. Finally, we have to ensure that if a feature is assigned to a specific release k, then all of its tasks need to be completed by release date t(k).

In addition to human resources, nonhuman resources (e.g., capital) are considered for each release as well. We assume M different types of nonhuman resources. Feature f(n) consumes an amount r(n, m) of nonhuman resources of type m. We introduce cap(k, m) with $\mathrm{m} = 1..\mathrm{M}$ and $\mathrm{k} = 1..\mathrm{K}$ as the quantity of the (nonhuman) resource of type m that is available in release k. This results in the following capacity constraints:

$$\begin{aligned}&\Sigma_{\mathrm{h}=1..\mathrm{k}}\ \Sigma_{\mathrm{n}=1..\mathrm{N}}\ \mathrm{r}(\mathrm{n},\mathrm{m})\cdot\mathrm{x}(\mathrm{n},\mathrm{h}) \le \Sigma_{\mathrm{h}=1..\mathrm{k}}\ \mathrm{cap}(\mathrm{h},\mathrm{m})\\ &\quad\text{for all } \mathrm{m}=1 \text{ and } \mathrm{k}=1..\mathrm{K}.\end{aligned} \tag{2}$$

A summary of the key attributes of a feature f(n) is shown in Fig. 1.

2.1.5 Assignment of Tasks to Developers

It is well known that there are major differences in the skills and productivity of software developers [1]. In order to accommodate this, we introduce an average skill level with a normalized productivity factor of 1.0 for each type of task. This allows us to consider more or less skilled developers with a higher or lower productivity factor than 1, respectively. We assume that the project manager can judge the different degrees of productivity of the developers.

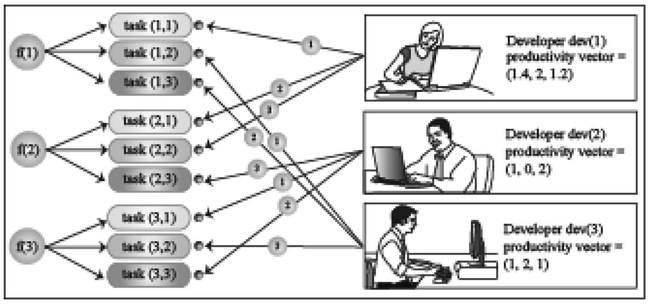
f(1)
task (1,1)
task (1,2)
task (1,3)
f(2)
task (2,1)
task (2,2)
task (2,3)
f(3)
task (3,1)
task (3,2)
task (3,3)
1
2
3
2
1
3
1
2
3
Developer dev(1) productivity vector = (1.4, 2, 1.2)
Developer dev(2) productivity vector = (1, 0, 2)
Developer dev(3) productivity vector = (1, 2, 1)

Fig. 2. Assignment of three developers to all the tasks related to features $\mathrm{f}(1)..\mathrm{f}(3)$.

We consider a pool of D developers, denoted by $\mathrm{dev}(1)..\mathrm{dev}(\mathrm{D})$, performing one or more types of development activities. The productivity factor prod(d, q) of a developer dev(d) for performing a task of type q indicates whether the developer is able to perform the task at all $(\mathrm{prod}(\mathrm{d},\mathrm{q}) \neq 0)$ and, if "yes," how productively (s)he performs that task. In order to summarize information related to the assignment of developers to tasks, we will later use vector u defined as

$$\begin{aligned}&\mathrm{u}(\mathrm{d},\mathrm{t},\mathrm{n},\mathrm{q}) = 1 \text{ if and only if at moment t } (\mathrm{t}\ = 1..\mathrm{t}[\mathrm{K}])\\ &\text{developer d } (\mathrm{d}\ = 1..\mathrm{D}) \text{ is working on task}(\mathrm{n},\mathrm{q})\\ &(\text{and } \mathrm{u}(\mathrm{d},\mathrm{t},\mathrm{n},\mathrm{q}) = 0 \text{ otherwise}).\end{aligned} \tag{3}$$

The assignment of human resources to feature-related tasks is illustrated in Fig. 2. For illustrative purposes, we consider an example of three features with three distinct tasks. A pool of three developers is available to perform the tasks. Each developer has different levels of productivity for performing the individual tasks. The three-dimensional productivity vector indicates the relative productivity to perform three tasks (in this order): design, implementation, and testing. For example, developer dev(2) having productivity vector (1, 0, 2) means that this developer is twice as productive as an average tester (degree of productivity with regards to testing is 2) but cannot perform an implementation task (the degree of productivity with regards to implementation is 0). In Fig. 2, we can also see a possible assignment of developers to the nine tasks under consideration. For each developer, the number at the arc between a developer and a task indicates the order in which the assigned tasks will be performed by the developers. To look at dev(2) again, the first task is the design for feature f(3), followed by testing for f(3). Finally, testing for f(2) is done.

To illustrate the impact of varying productivity values, we consider the estimated workload for task(1, 3) to be $\mathrm{w}(1,3) = 10$ person days. Then, developer dev(3) would need 10 days to perform the tasks. As the productivity of developer dev(2) for the task of testing is assumed to be 2, the same task when performed by this developer would take only five days.

The best possible assignment takes into account the productivity of the developers, their availability, and the possible dependencies between tasks.

2.1.6 Overall Problem Statement

We describe a release plan x and the associated resource allocation u by the combined vector (x, u). We also use the terms "assignment" for x and "detailed schedule" for u. The composite set of all feasible assignments and detailed schedules[1] (x, u) is denoted by the Cartesian product $X \cdot U$ or just (X, U). We note that the part of the problem related to detailed schedules does not directly influence the stated objective of planning but is part of the constraint set that needs to be fulfilled. We can say that the schedule "enables" the features. The objective is to maximize the stated utility function F, which is based on the value parameters v(n, k) introduced in Section 2.1.2. The problem RASORP is formally stated as

$$\begin{aligned}\text{Maximize } \big\{ & F(x) = \Sigma_{n=1..N}\, \Sigma_{k=1..K}\, v(n,k) \cdot x(n,k) \\ & \text{subject to } (x,u) \in (X,U) \big\}.\end{aligned} \tag{4}$$

2.2 Problem Complexity

We study the formal computational complexity of resource allocation for release planning by transforming the knapsack problem (KP) (which is known to be NP-complete [15]) to a special case of RASORP where we look at one release only and all features f(n) have value and effort parameters value(n, 1) and effort(n, 1), respectively.

Decision problem KP is defined as follows: Given a finite set Γ of elements, integers cap^* and value^*, variables value (n, 1) and effort r(n, 1) associated with every element $n \in \Gamma$. Question: Does some subset $\Gamma^* \subset \Gamma$ satisfy the property that $\sum_{n \in \Gamma} \text{value}(n,1) \geq \text{value}^*$ and $\sum_{n \in \Gamma} r(n,1) \leq \text{cap}^*$?

Given KP for a set Γ of N elements, we construct KP as a RASORP problem instance with just $K = 1$ release and $M = 1$ resource as follows: The N elements of Γ are defined as features f(1)..f(N). Using $\text{cap}(1,1) = \text{cap}^*$ and Boolean decision vector x as defined in (1), the question becomes one of finding a subset of features that does not exceed a given resource capacity bound cap(1, 1) and that provides a value of at least value*. Any solution x* of that problem directly transforms into a solution of the KP. From the known status of KP being NP-complete, we conclude that there exists no deterministically polynomial algorithm for this problem, as long as there is no such algorithm for any of the known NP-complete problems.

For another complexity consideration, we consider the so-called two-dimensional KP, which has an additional (resource) constraint. In that case, no fully polynomial time approximation scheme (FPTAS) exists [23]. We can use an analogous transformation as above to conclude that there is no FPTAS for RASOPR in the case where at least two types of resources are considered.

2.3 Related Results

Software project management includes management of human and other resources. Assigning properly skilled developers to appropriate tasks is a complex process, typically including a large number of variables related to schedule times, availability, staffing, and skill profile. If this process is done ad hoc and based on intuition, the results are poor resource utilization and schedule overrun. While intuition is not a bad thing, intuition alone is not sufficient for making complex and crucial decisions. In [33], it was stated that "Software Engineering Project Management's progress has been agonizingly slow in many years, probably it is driven more by human behavior than by technology." In the same way, Padberg [32] pointed out that software engineering currently offers little help to software project managers on how to develop good schedules for their projects.

There are few systematic approaches providing operational guidance in software projects. Chang et al. [7] proposed the application of GAs for software project management. The proposed method focused on schedule minimization and did not examine different productivity levels of developers. More recently, Fenton et al. [13] used Bayesian belief networks as a means to circumvent the inherent uncertainty in estimating effort for resource decisions.

To provide a systematic and quantitatively based solution approach addressing these deficits, we propose a formalized problem description and subsequent application of specialized optimization techniques. The problem under consideration is a combination of a scheduling problem called the Job Shop Scheduling Problem (JSSP) and a variant of KP. In JSSP, a set of tasks (jobs) is implemented by using a given set of available machines. Every job passes every machine exactly once in a prescribed order. There are prescribed job due dates and eventually nonzero release dates of jobs. Precedence relationships need to be addressed between some of the jobs. The objective is to maximize performance, which can involve determining a schedule that minimizes the makespan (i.e., the minimum time to perform the whole project) or minimizes a tardiness function where the tardiness of a job indicates the time span the completion time overshoots the projected due date.

Software engineering problems have been reformulated as search problems in [8] to allow application of powerful metaheuristics such as Simulated Annealing [24] and Tabu Search [16]. These techniques, as well as exact techniques such as Branch and Bound [24], have also been applied to solve JSSP. JSSP has been extensively studied and is known to be NP-complete [15]. Different exact procedures exist for small-scale problems, as well as heuristics for different variants of this problem. A comprehensive survey of job shop scheduling techniques can be found in [21]. Davis [10] was the first to use GAs [19] to solve JSSP. Hartman [18] considers the resource-constrained project scheduling problem with makespan minimization as an objective. A specific implementation of a GA is proposed for its solution. A comparative analysis with other heuristics was conducted, but no quality estimates for the approximation results were obtained. More recently, an efficient GA for JSSP with tardiness objectives was proposed in [26]. The authors introduced a heuristic that cuts down the possible search space and improves performance when the problem size increases.

While related to JSSP, the RASORP problem addressed in this paper is different in terms of the utility function, the constraints, its capability to model different modes of

1. We will use the term "detailed schedule" for vector u to differentiate from the term "schedule" later used for the solution of a relaxed version of the problem.

TABLE 1
Priorities prio(n, j) Assigned to Features f(n) by Stakeholder S(j), j = 1..4

f(n)	Feature Name	S(1)	S(2)	S(3)	S(4)
1	16 sector, 12 carrier BTS for China	5	4	8	7
2	China Feature 1	4	5	5	6
3	China Feature 2	1	3	1	3
4	China Feature 3	3	2	2	3
5	China Feature 4	2	4	1	6
6	China Feature 5	2	3	5	2
7	Common Feature 01	9	8	6	8
8	Common Feature 02	8	8	3	9
9	Common Feature 03	8	2	3	8
10	Common Feature 04	6	7	3	8
11	Cost Reduction of Transceiver	7	3	8	6
12	Expand Memory on BTS Controller	3	2	2	2
13	FCC Out-of-Band Emissions Regulatory Change	8	6	4	4
14	India BTS variant	3	3	3	2
15	India Market Entry Feature 1	3	6	4	8
16	India Market Entry Feature 2	8	5	1	3
17	India Market Entry Feature 3	2	2	6	4
18	Next Generation BTS 'In a Shoebox'	7	6	2	6
19	Pole Mount Packaging	7	3	4	5
20	Software Quality Initiative	9	9	9	9

realization of the tasks, and the packaging component. The problem is not only to schedule a set of jobs or tasks but also to package features into consecutive releases, including the possibility that certain features are not provided with a given time horizon (number of releases). Another difference is in the ability to model different productivity modes for the machines (developers) available to perform the tasks. One developer is able to perform different types of tasks, and the difference in productivity results in different lengths of time needed to perform the tasks.

In KP, the goal is to select items from a given set of items such that the value is maximized and yet the cost does not exceed a certain value. KP and its generalizations have been extensively studied during the last three decades. For a more recent overview, we refer to the work of Kellerer et al. [23]. Many different heuristic solutions have been suggested for KP (see, e.g., the work of Martello and Toth [27] for a specialized greedy approximation algorithm). The solution method presented in this paper generalizes the multi-dimensional KP in that it has more than one container and contains precedence constraints between the items to be packaged.

3 Hypothetical Case Study: Context and Problem Parameters

To illustrate the problem introduced above, we consider a hypothetical case study project from the telecommunications domain that was introduced in [37]. In order to demonstrate the added value from the consideration of resource allocation as part of release planning, the original problem has been extended by a set of tasks for each feature and by a set of developers with different skill profiles. In our case, the goal of planning is to assign human resources to tasks of the 20 features such that the completed and released features offer the best overall value to customers. The customers are represented by four stakeholders S(1)..S(4) with relative degrees of importance defined on a nine-point scale. The concrete degrees of importance are

TABLE 2
Productivity Factors prod(d, q) of Developers dev(d) (d = 1..6) for Different Types q of Tasks

prod(d,q)	Design	Implementation	Testing
dev(1)	1.4	2	1.2
dev(2)	1	0	2
dev(3)	1	2	1
dev(4)	2	0	1
dev(5)	1	1.5	2
dev(6)	2	1	2

$\lambda(1) = 1$ (extremely low) for S(1), $\lambda(2) = 3$ (low) for S(2), $\lambda(3) = 4$ (between low and average) for S(3), and $\lambda(4) = 7$ (high) for S(4). Their assigned degree of priority to the 20 features is given in Table 1. prio(n, j) denotes the priority assigned to feature f(n) by stakeholder S(j). The priorities prio(n, j) are given on a nine-point scale, ranging from 1 (very low priority) to 9 (very high priority). Therein, priority refers to one of the possible prioritization criteria mentioned above. We just assume that it represents the degree of urgency assigned to the feature.

Two releases are considered with release due dates at $t(1) = 12$ and $t(2) = 24$ (in weeks of duration, where efforts are estimated in person weeks with one person week being five person days). In order to make a feature available, it must pass through three related tasks devoted to design, implementation, and testing. For any feature assigned to release 1 (respectively, release 2), its related tasks must be finished before the end of week 12 (respectively, 24). The project provides $D = 6$ developers called dev(1)..dev(6) with the individual skills and productivity profiles described in Table 2. We can see that developer dev(2) is assumed to be of average productivity for design tasks and cannot be assigned to perform development tasks. We also note that dev(2) is twice as productive in testing as average testers are.

All remaining sample project data are summarized in Table 3. Column r(n, 1) describes the use of the nonhuman resources with release capacities of 900 units for both releases. Columns v(n, 1) and v(n, 2) describe the assumed value contributions of features f(n) to the overall utility function F(x) if assigned to releases 1 and 2, respectively. More specifically, with factors $\xi(1)$ and $\xi(2)$ representing the relative degree of importance of releases 1 and 2, respectively, and factors $\lambda(j)$ describing the relative importance of stakeholder $S(j)$ $(j = 1..4)$, we define

$$v(n, 1) = \xi(1)\Sigma_{j=1..4}\lambda(j)\ prio(n, j), \quad (5)$$

$$v(n, 2) = \xi(2)\Sigma_{j=1..4}\lambda(j)\ prio(n, j). \quad (6)$$

Nonnegative factors $\xi(1)$ and $\xi(2)$ are defined on a ratio scale. These factors of relative importance of the releases are used to express the downgrading (or upgrading) factors for a feature released at a later time than release 1. Typically, the contribution of a feature is reduced over time, e.g., $\xi(1) > \xi(2)$. In Table 3, columns v(n, 1) and v(n, 2) show the values of features in releases 1 and 2, respectively. For that, factors $\xi(1) = 1$ and $\xi(2) = 0.5$ have been used. Columns w(n, 1), w(n, 2), and w(n, 3) present the workload necessary

TABLE 3
Feature-Related Project Data

f(n)	Feature Name	r(n,1)	v(n,1)	v(n,2)	w(n,1)	w(n,2)	w(n,3)
1	16 sector, 12 carrier BTS for China	100	392	196	1	6	6
2	China Feature 1	50	324	162	8	4	2
3	China Feature 2	60	140	70	3	6	1
4	China Feature 3	50	152	76	6	2	3
5	China Feature 4	60	240	120	2	9	9
6	China Feature 5	200	180	90	5	6	4
7	Common Feature 01	80	452	226	1	10	8
8	Common Feature 02	100	428	214	1	4	6
9	Common Feature 03	100	328	164	5	4	4
10	Common Feature 04	120	380	190	7	5	1
11	Cost Reduction of Transceiver	110	360	180	10	5	6
12	Expand Memory on BTS Controller	50	124	62	5	1	7
13	FCC Out-of-Band Emissions Regulatory Change	250	280	140	4	9	9
14	India BTS variant	200	152	76	4	10	3
15	India Market Entry Feature 1	70	372	186	8	8	8
16	India Market Entry Feature 2	80	192	96	10	5	2
17	India Market Entry Feature 3	60	240	120	5	6	6
18	Next Generation BTS 'In a Shoebox'	180	300	150	10	10	6
19	Pole Mount Packaging	200	268	134	1	8	9
20	Software Quality Initiative	120	540	270	6	8	10

to implement the feature f(n) in terms of design, implementation, and testing, respectively There are four precedence constraints between features to be considered (see Table 4). For example, feature f(16) (India Market Entry Feature 2) cannot be in an earlier release than f(15) (India Market Entry Feature 1).

This problem, although small in size, is already difficult to solve. Potentially, 60 tasks must be performed by six developers with different productivity profiles. There are various dependencies between the features and between the tasks that must be considered. Finally, a nonhuman resource and its capacities also have to be considered. With D denoting the number of developers available, t(K) denoting the number of time periods considered for K releases, N denoting the number of features under consideration, and Q denoting the number of task types needed to implement features, we obtain as many as $D \cdot t(K) \cdot N \cdot Q = 6 \cdot 24 \cdot 20 \cdot 3 = 8{,}640$ binary variables to solve the overall problem stated in (4). We conclude that special computer support is needed to handle this problem.

4 Solution Approach

4.1 Two-Phase Problem Solution Approach—An Overview

Given the analysis of the problem's complexity, we have concluded that we need a specialized solution approach for RASORP. Our two-phase approach, called $\mathrm{OPTIMIZE_{RASORP}}$, combines the strength of special structure integer linear programming (Phase 1) with the power of GAs (Phase 2). The advantage is twofold. First, from Phase 1, we can generate an upper bound for the maximum value achievable. This allows an evaluation of the solution generated in Phase 2. Second, the solution obtained from Phase 1 is used to restrict the set Π (permutations of N features) to the set Π^* used in the GA of Phase 2. This can significantly reduce the computational effort and allows solution of problems of small and medium size.

Phase 2 can be applied without application of Phase 1, but the search space would be substantially larger in this case. The restriction provided by Phase 1 is heuristic in its nature. That means that it is likely that Π^* (reduced search space) contains a good solution. However, there is no guarantee that it necessarily contains the optimal solution. We will call the direct application of GA to RASORP unfocused search (UFS), while the two-phase method $\mathrm{OPTIMIZE_{RASORP}}$ is also called focused search (FS) (when focusing on the search strategy).

TABLE 4
Precedence Constraints between Features

India Market Entry Feature 1	precedes	India Market Entry Feature 2
India Market Entry Feature 2	precedes	India Market Entry Feature 3
India BTS variant	precedes	Next Generation BTS 'In a Shoebox'
India BTS variant	precedes	Pole Mount Packaging

4.2 Phase 1

To facilitate the process of solving (4), we initially consider a simplified version of RASORP called RASORP*. Instead of looking at all t(K) possible points of time t $(t = 1..t(K))$ for scheduling tasks, in RASORP*, we only look at release dates $t(k)$ $(k = 1..K)$. This leads to a significant reduction of variables and constraints. To formally describe this simplified problem, we use the variables y(d, k, n, q) instead of variables u(d, t, n, q) as defined in (4). Vector u is larger in size than y. The reason for that is that u is defined for each point in time $t = 1..t(K)$ instead of just the release dates $t(k)$ $(k = 1..K)$. More specifically, y is defined as

$$\begin{aligned} &y(d,k,n,q) = 1 \text{ if and only if } \mathrm{task}(n,q) \text{ is assigned to} \\ &\quad \text{developer } d \text{ and is finished in release } k \\ &\quad (\text{and } y(d,k,n,q) = 0 \text{ otherwise}). \end{aligned} \tag{7}$$

If x is given, y can be derived from it. When moving to Phase 2, the granularity of timing (for the assignment of tasks) is increased (from release level to individual time intervals). In general, the assignment y of tasks from Phase 1 would be infeasible for the more fine-grained perspective of Phase 2. That is why the y part generated from Phase 1 is ignored. Instead, a new schedule vector u is created independent from y directly from the solutions maintained in Phase 2.

We have to ensure that, for all developers dev(d) and for all release times t(k), developer dev(d) can work at most t(k) units of time until the end of release k. This leads to[2]

$$\begin{aligned} &\Sigma_{n=1..N}\ \Sigma_{q=1..Q}\ \Sigma_{h=1..k} \lceil w(n,\ q)/\mathrm{prod}(d,\ q)\rceil y(d,\ h,\ n,\ q) \\ &\leq t(k) \quad \text{for all } k = 1..K \text{ and } d = 1..D. \end{aligned} \tag{8}$$

In addition to precedence constraints between features as given in Definition 2, precedence relationships between tasks must be considered as well. These relationships represent the task dependencies that need to be considered for different types of life-cycle development models. In this paper, we are using an end-to-end precedence relationship between tasks. This is to ensure that, e.g., design cannot be finished before testing.

2. For any nonnegative real number w, we denote by $\lceil w \rceil$ the smallest integer not smaller than w.

Definition 4. Weak task precedence. $\mathrm{task}(n, q+1)$ *cannot be finished earlier than the time* task(n, q) *is finished. This is written as* $(\mathrm{task}(n,q), \mathrm{task}(n,q+1)) \in P$ *for all features* n *and all related tasks* $q = 1..Q-1$.

We describe a release plan x and the associated resource allocation procedure y by vector (x, y). We also use the terms assignment for x and schedule for y. The composite set of all feasible assignments and schedules is denoted by the Cartesian product $X \cdot Y$ or just (X, Y). (X, Y) includes all the schedule and resource allocation constraints introduced with respect to (x, y). Overall, the resulting release and resource allocation optimization model RASORP^* is defined as a relaxed version of the original problem RASORP:

$$\text{Maximize } \{F(x) = \Sigma_{k=1..K}\, \Sigma_{n=1..N}\, v(n, k)\, x(n, k) \text{ subject to } (x, y) \in (X, Y)\}. \tag{9}$$

For the solution of (9), we apply branch-and-cut techniques [29] in combination with linear programming (solving the relaxed problem without integrality constraints) to generate upper bounds and use a greedy heuristic to solve the resulting subproblem at each node of the branching tree. At the end of Phase 1, we obtain a solution (x^1, y^1) having value $F(x^1)$. $F(x^1)$ serves as the upper bound for the best possible value achievable in Phase 2 when we use x^1 as the starting point to define a reduced search space $\Pi^*(x^1)$.

4.3 Phase 2

The main idea of Phase 2 is that the optimal or at least a near-optimal solution of the complete RASORP problem can be obtained from the relaxed version RASORP^*. That means that we take solution x^1 and try to adjust it to fulfill the additional constraints. In Phase 2, we use a GA [14] to search for the best solution for assigning human resources to the tasks of the features chosen for release. For a solution x^1 obtained from Phase 1, we define sets of indices of features belonging to the same release.

Definition 5. *The family of indices* $A(x^1, k)$ *of features assigned to release k is called an* x^1*-assignment. Index sets* $A(x^1, k)$ *are defined by*

$$A(x^1, k) = \{n \in 1..N : x^1(n, k) = 1\}(k = 1..K), \tag{10}$$

$$A(x^1, K+1) = \{n \in \{1..N\} : \text{feature } f(n) \text{ postponed according to release plan } x^1\}. \tag{11}$$

In Phase 2, we want to ensure that all features in $A(x^1, 1)$ precede those in $A(x^1, 2)$, which in turn precede those in $A(x^1, 3)$. This process to enlarge the set P of precedence relationships between features is repeated for all consecutive releases defined by x^1. We introduce K artificial features $f(N+1)..f(N+K)$, as well as the following constraints (illustrated in Fig. 3):

$$\text{for all } n \in A(x^1, k),\ k = 1..K, \text{ add constraint "} f(n) \text{ precedes } f(N+k) \text{" to the set P,} \tag{12}$$

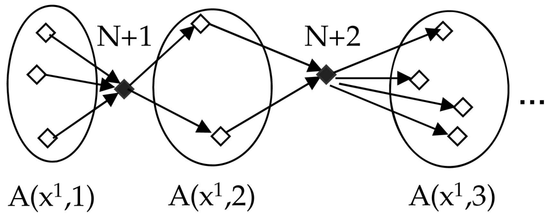

Fig. 3. Artificial features f(N + 1)..f(N + K) and additional precedence constraints generated from x^2.

$$\text{for all } n \in A(x^1, k),\ k = 2..K+1, \text{ add constraint "} f(N+k-1) \text{ precedes } f(n) \text{" to the set P.} \tag{13}$$

The new search space $\Pi^*(x^1)$ is obtained from the set of all permutations Π by adding all the new constraints to P. For each permutation $\pi = (\pi(1), \ldots, \pi(N))$ of feature indices $\{1, \ldots, N\}$, a unique detailed schedule is determined using a serial scheduling scheme, i.e., a procedure to transform a permutation to a schedule. The scheme proceeds along the features in the order defined by permutation π, that is, it starts with $f(\pi(1))$, followed by $f(\pi(2))$, and is repeated until $f(\pi(N))$. For each feature, tasks are handled in the order required. Each task is assigned to the developer that can finish the task the soonest. Once the detailed schedule u is determined, the assignment x can be derived, and the utility function value $F(x) = \sum_{k=1..K} \sum_{n=1..N} v(n,k)\, x(n,k)$ can be computed. This value F(x) is used to define the value of the fitness function of the permutation π used in the GA further specified in Section 6.2.

4.4 Applicability

Our proposed method is intended to qualify release planning, including the inherent planning of human and nonhuman resources and their assignment to tasks. To be successful in that, project and/or product managers need to consider a variety of factors. Not all of them are included in the proposed model (e.g., organizational factors, process factors, or human factors related to staffing were not taken into account). In the sense of evolutionary problem solving as presented in [31], we argue that a good solution to a reasonable approximation of the real-world problem is a step forward in determining a practically applicable solution.

The same arguments are applied to address the inherent uncertainty of data. Effort and productivity estimates are approximations and prone to errors. From running different scenarios (instantiations of the problem with varying parameter settings) of the given problem, the impact of parameter variations can be better understood. This proactive analysis is considered to be an essential step to facilitate better and more objective decisions made by the product and/or project manager.

There is a continuous need for replanning due to changed or new requirements, modified constraints, or priorities [39]. Our paper does not directly address this replanning process. However, once the project data are established, the actual replanning becomes easier to perform and the same two-stage solution method can be applied using the modified parameters.

The main practical benefits of applying the results are expected in terms of more effective resource utilization and improved (higher) total value of the proposed release plans. This value can be seen in financial terms or in a better overall stakeholder satisfaction with the given release plan strategies. In order to achieve these benefits, processes for planning and road-mapping of software releases, effort and resource estimation, definition of the values of features in relation to time, and evaluation of the productivity of human resources need to be in place. That means that the proposed method is applicable for organizations being at level 3 ("Defined") on the five-point maturity scale defined by the Capability Maturity Model Integrated (CMMI) as described in [2].

5 Hypothetical Case Study: Solution of the Stated Problem

We use the two-phase solution approach described in Section 4 to solve the hypothetical case study example introduced in Section 3. To better compare the results, we initially apply a simple greedy search heuristic. This heuristic makes, at any time in the process, the locally best choice for assigning tasks to idle developers. To illustrate the solution method and to compare the different search methods, we consider four different solutions:

- x^1 solution at the end of Phase 1,
- x^2 solution at the end of Phase 2 (FS),
- x^3 solution obtained from the direct application of GA (UFS), and
- x^4 solution obtained from the application of greedy search (see Appendix A).

The release structure of x^4 can be seen in Table 7, where release 1 offers features f(1), f(2), f(3), f(8), f(9), f(10), and f(11). Similarly, release 2 provides features f(4), f(6), f(7), f(14), f(15), and f(20). The resulting value is $F(x^4) = 3,276$. We will later see that the application of $\text{RASORP}_{\text{OPTIMIZE}}$ results in a solution that is about 18 percent better than the greedy search solution.

The application of Phase 1 of $\text{RASORP}_{\text{OPTIMIZE}}$ involves solving the relaxed problem RASORP^*. As a result of that, we obtain a release plan x^1 defined by index sets of features:

- $A(x^1, 1) = \{1, 2, 7, 8, 9, 10, 11, 20\}$ (release 1).
- $A(x^1, 2) = \{3, 4, 13, 14, 15, 16, 17, 19\}$ (release 2).
- $A(x^1, 3) = \{5, 6, 12, 18\}$ (postponed).

In Phase 2, we conduct more fine-grained optimization while maintaining all the precedence relationships between features and tasks. The Gantt chart for the assignment of developers to feature-related tasks provided by x^2 is shown in Table 5. We observe that the budget constraints for both releases are satisfied:

$$\Sigma_{n=1..20} r(n, 1) \cdot x^2(n, 1) = 780 \leq 900 = \text{cap}(1, 1), \quad (14)$$

$$\begin{aligned} &\Sigma_{n=1..20} r(n, 1) \cdot x^2(n, 1) + \Sigma_{i=1..20} r(n, 1) \cdot x^2(n, 2) \\ &= 1,680 \leq 1,800 = \text{cap}(1, 1) + \text{cap}\ (2, 1). \end{aligned} \quad (15)$$

The Gantt chart shown in Table 5 represents both parts of the solution vector (x^2, y^2): the feature assignment to releases x^2 and the assignment of tasks to developers y^2. For each of the three tasks of a feature, we can see when it is proposed to be done. The number in the bar indicates the developer assigned for that task. We observe the following:

- All developers are continuously busy during the weeks. The only exception is developer dev(3) being idle starting week 22 and all remaining developers being idle during week 24.
- Features f(3) and f(19) are offered in the second release, but their development begins at the end of the first release (as there are resources available). This increases efficiency of resource utilization.
- As a result of the more fine-grained optimization of Phase 2, taking into account all the precedence relation ships between features and tasks, feature f(16) can no longer be offered in release 2.
- The assignment of tasks to developers strongly takes into account the developer's skill set. For example, developer dev(1) spends 22 of the 24 weeks performing implementation at which s/he is the most productive. Overall, for 37 of the 45 tasks to be performed, the assignment was the ideal one (i.e., assigned to developers having the highest productivity). This is shown in detail in Table 6.

The quality of the solution resulting from Phase 2 is compared with the upper bound $F(x^1) = 4{,}102$ obtained from Phase 1. The best solution x^2 obtained from FS in $\Pi^*(x^1)$ is the one with a guaranteed quality of 97.7 percent when compared to $F(x^1)$. In reality, the quality can be even

TABLE 5
Gantt Chart of Resource Allocation to Tasks for Release Plan Solution x^2

		Week																							
Feature	Task	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
	(1,1)	3																							
f(1)	(1,2)		3	3	3																				
	(1,3)	5	5		5																				
	(2,1)	6	6	6	6																				
f(2)	(2,2)					1	1																		
	(2,3)						2																		
	(7,1)			5																					
f(7)	(7,2)					3	3	3	3	3															
	(7,3)					5	5	5		5															
	(8,1)				4																				
f(8)	(8,2)							1	1																
	(8,3)					6	6		6																
	(9,1)					4	4	4																	
f(9)	(9,2)									1	1														
	(9,3)							2			2														
	(10,1)								4	4	4	4													
f(10)	(10,2)										3	3	3												
	(10,3)												1												
	(11,1)							6		6	6	6	6												
f(11)	(11,2)								5		5	5	5												
	(11,3)								2	2			2												
	(20,1)	4	4	4																					
f(20)	(20,2)	1	1	1	1																				
	(20,3)	2	2	2	2	2																			
	(3,1)												4	4											
f(3)	(3,2)											1		1	1										
	(3,3)														2										
	(4,1)													6	6	6									
f(4)	(4,2)															1									
	(4,3)													2		2									
	(13,1)														4	4									
f(13)	(13,2)																1	1	1	1	1				
	(13,3)																2	2	2	2	2				
	(14,1)																4	4							
f(14)	(14,2)																	3	3	3	3	3			
	(14,3)																		4	4		4			
	(15,1)																6	6	6	6					
f(15)	(15,2)																		5	5	5	5	5	5	
	(15,3)																				6	6	6	6	
	(17,1)																				4		4	4	
f(17)	(17,2)																					1	1	1	
	(17,3)																					2	2	2	
	(19,1)											2													
f(19)	(19,2)													3	3	3	3								
	(19,3)													5	5	5	5	5							

TABLE 6
Comparison between Actual and Ideal Assignment of Tasks to Developers

Feature	Task	Actual developer assigned	Ideal developer to be assigned	Match
f(1)	(1,1)	3	4, 6	No
	(1,2)	3	1, 3	**Yes**
	(1,3)	5	2, 5, 6	**Yes**
f(2)	(2,1)	6	4, 6	**Yes**
	(2,2)	1	1, 3	**Yes**
	(2,3)	2	2, 5, 6	**Yes**
f(7)	(7,1)	5	4, 6	No
	(7,2)	3	1, 3	**Yes**
	(7,3)	5	2, 5, 6	**Yes**
f(8)	(8,1)	4	4, 6	**Yes**
	(8,2)	1	1, 3	**Yes**
	(8,3)	6	2, 5, 6	**Yes**
f(9)	(9,1)	4	4, 6	**Yes**
	(9,2)	1	1, 3	**Yes**
	(9,3)	2	2, 5, 6	**Yes**
f(10)	(10,1)	4	4, 6	**Yes**
	(10,2)	3	1, 3	**Yes**
	(10,3)	1	2, 5, 6	No
f(11)	(11,1)	6	4, 6	**Yes**
	(11,2)	5	1, 3	No
	(11,3)	2	2, 5, 6	**Yes**
f(20)	(20,1)	4	4, 6	**Yes**
	(20,2)	1	1, 3	**Yes**
	(20,3)	2	2, 5, 6	**Yes**
f(3)	(3,1)	4	4, 6	**Yes**
	(3,2)	1	1, 3	**Yes**
	(3,3)	2	2, 5, 6	**Yes**
f(4)	(4,1)	6	4, 6	**Yes**
	(4,2)	1	1, 3	**Yes**
	(4,3)	2	2, 5, 6	**Yes**
f(13)	(13,1)	4	4, 6	**Yes**
	(13,2)	1	1. 3	**Yes**
	(13,3)	2	2, 5, 6	**Yes**
f(14)	(14,1)	4	4, 6	**Yes**
	(14,2)	3	1, 3	**Yes**
	(14,3)	4	2, 5, 6	No
f(15)	(15,1)	6	4, 6	**Yes**
	(15,2)	5	1, 3	No
	(15,3)	6	2, 5, 6	**Yes**
f(17)	(17,1)	4	4, 6	**Yes**
	(17,2)	1	1, 3	**Yes**
	(17,3)	2	2, 5, 6	**Yes**
f(19)	(19,1)	2	4, 6	No
	(19,2)	3	3	**Yes**
	(19,3)	5	2, 5, 6	**Yes**

better, as we are using an upper bound for the utility function value.

In Table 7, we compare the structure of the solutions x^2, x^3, and x^4. In the table, numbers 1 and 2 refer to assignment of features to releases 1 and 2, respectively. P stands for "postponed." We can see that, for 11 of the 20 features, all three methods provide the same release assignment. However, the greedy solution performed poorer in terms of the number of features provided in releases 1 and 2. This is also confirmed by the value F(x) as defined in (3).

In addition to the utility function value F(x), we compare x^2 and x^4 in terms of the estimated stakeholder satisfaction (ESS) from the proposed plans. Intuitively, a stakeholder S(j) will be more satisfied with a proposed plan x if the highly prioritized (by S(j)) features are provided early. We use (16) for an estimate ESS(S(j), x) for the satisfaction of stakeholder S(j) with respect to solution x:

$$\mathrm{ESS(S(j),\ x)} = [\xi(1)\Sigma_{\mathrm{n:\ x(n)=1}}\ \mathrm{prio(n,j)} + \xi(2)\Sigma_{\mathrm{n:\ x(n)=2}}\ \mathrm{prio(n,j)}]/\xi(1)\Sigma_{\mathrm{n=1..N}}\ \mathrm{prio(n,\ j)}. \tag{16}$$

ESS(S(j), x) is a relative measure normalized to the interval [0, 1] in case $\xi(1) \geq \xi(2)$. For the case study, we assume that $\xi(1) = 1$ and $\xi(2) = 1/2$, e.g., the same feature assigned to release 1 provides twice as much satisfaction as it does when assigned to release 2. Any feature suggested for postponement does not contribute to the satisfaction at all. Fig. 4 clearly shows that x^2 improves the estimated satisfaction by more than 15 percent for all stakeholders related to the greedy solution x^4. Stakeholders S(1) and S(3) would likely be the most satisfied with the FS solution.

When compared with release planning as presented in [37], the case study shows the added values of the approach including resource assignments to tasks. While making objective release suggestions for all features, the proposed results also offers guidance in terms of which tasks are done by which developer and when the task should be performed. With ESS, this is considered an essential input for the final human decision to be made.

6 Empirical Analysis

6.1 Description of Test Cases

To analyze the performance of the proposed method, we randomly generated a set of 600 test cases (also called projects). To keep the size reasonable, we have defined 20 groups out of these 600 projects. The group size was fixed at 30 projects. Within each group, the other parameters were varied randomly within given intervals. The groups with their key parameter values are described in Table 8. The columns are the following:

- Range N describes the range in the number of features in the group projects.
- Average K describes the average number of planning periods considered with $K \in [1, 5]$.
- Average M describes the average number of nonhuman resource types with $M \in [1, 4]$.
- Average Q refers to the average number of tasks considered per feature with $Q \in [3, 4]$.
- Average D refers to the average number of developers considered with $D \in [2, 26]$.
- Average $|P|$[3] refers to the average number of precedence constraints with $|P| \in [0, 19]$.
- Average HR describes the degree of demand for human resources when considering all features (ratio of total capacity and total consumption for all features).
- Average NHR describes the degree of demand for nonhuman resources when considering all features (ratio of total capacity and total consumption for all features).

3. $|P|$ stands for the number of elements of set P.

TABLE 7
Comparison of Plans Generated from FS (x^2), UFS (x^3), and Greedy Search (x^4)

Release Plan	f(1)	f(2)	f(3)	f(4)	f(5)	f(6)	f(7)	f(8)	f(9)	f(10)	f(11)	f(12)	f(13)	f(14)	f(15)	f(16)	f(17)	f(18)	f(19)	f(20)	F(x)
x^2	1	1	2	2	P	P	1	1	1	1	1	P	2	2	2	P	2	P	2	1	4006
x^3	1	1	2	2	P	2	1	1	1	1	2	P	2	2	P	P	2	P	1	1	3864
x^4	1	1	1	2	P	2	2	1	1	1	1	P	P	2	2	P	P	P	P	2	3276

6.2 The Genetic Algorithm

To better understand the strength and limitations of the proposed approach, we performed UFS (in space Π) as well as FS (in space Π^*) for all 600 test cases and compared the results with the application of a greedy algorithm. For performing searches, a genetic algorithm (GA) was used. GAs have proven successful in many search-related software engineering problems (see [8] for an overview and [5] for an example of a successful application). The focus of this empirical evaluation was to evaluate the performance of $\mathrm{OPTIMIZE}_{\mathrm{RASORP}}$, rather than finding the best metaheuristic among the ones possibly applicable.

For the genetic algorithm, we have chosen the open source GA framework given in [14]. The main difference between our approach and the permutation-based GA described by Hartmann [18] is the application of the serial scheduling scheme in $\mathrm{OPTIMIZE}_{\mathrm{RASORP}}$. We have not further investigated the impact of different parameter settings at the performance because this goes beyond the scope of this paper. Instead, we used the following parameter settings, which, on the average, have proven to be good for different customizations of GAs [7]:

- Population size: 100.
- Maximal number of generations: 500.
- Probability of mutation: 1 percent.
- Termination: iIf there is no improvement after 100 consecutive generations or the maximum number of generations is reached.
- Percentage of new random solutions in each new generation: 10 percent.
- Number of generations signaling that the population is stuck at a local optimum: 50.
- Proportion of new individuals when the population is stuck at a local optimum: 80 percent.

In many situations, the use of GA suffers from premature convergence, a situation where the population gets stuck around a local optimum. When this happens, the search process cannot get out of the current local optimum to explore the rest of the search space to find better solutions [12]. We tried to reduce the effect of this phenomenon by adding and adjusting the two last parameters. When the search process gets stuck, the population loses its diversification, all individuals in the population are similar, and no evolution can happen. New individuals are added in order to increase the diversification of the population.

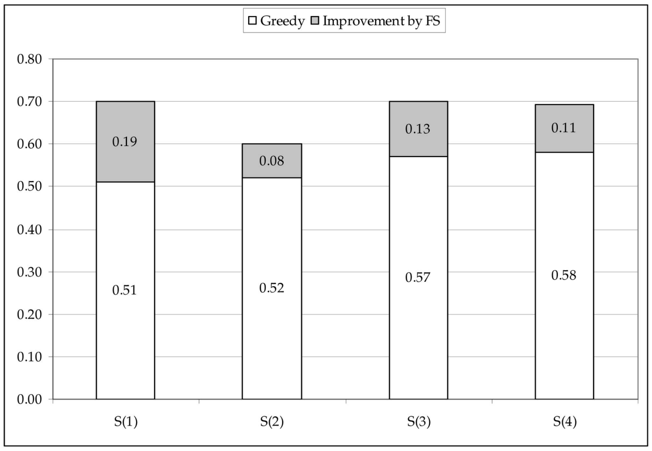

Fig. 4. Improvement of estimated stakeholder satisfaction (in gray) from Focused Search compared to greedy search for stakeholders S(1)..S(4).

6.3 Greedy Search

Greedy search [9] is often used in combinatorial optimization to determine a quick solution. The algorithm called Greedy determines a score for each feature and handles them according to the score in descending order. The arrangement of the N features in this order is indeed a permutation of N, which will be transformed to a feasible permutation, i.e., a permutation satisfying precedence constraints between corresponding features. The application of the serialization procedure on this feasible permutation offers a feasible solution (x, u). The details of heuristic Greedy are presented in Appendix A.

6.4 Experimental Results

Table 9 shows the results of the 600 projects grouped into 20 groups. All computations were made on a Pentium IV 2 GHz computer. The time limit for ILOG CPLEX [20] applied to RASORP^* in Phase 1 of the solution method was set to 30 seconds, which we understood as a good trade-off between the performance of the algorithm and the quality of the solutions obtained. Our decision for the solution time was based on the observation that relatively good solutions can be found quickly, but it takes substantially longer to further improve them or show their optimality.

6.5 Main Findings

6.5.1 Quality of Plans

The quality of plans is measured by the value F(x) of the utility function and its comparison to the determined upper bound $F(x^1)$ determined in Phase 1. Fig. 5 and Table 9 show the results of solving the 600 test problems with UFS based on the direct application of GA [14], focused search FS (i.e., $\mathrm{OPTIMIZE}_{\mathrm{RASORP}}$), and the application of Greedy. The results show the average within each of the groups defined in Table 8. The performance of the focused search FS is clearly better than that of UFS. UFS achieves an average quality of 83.6 percent, while the focused search results in

TABLE 8
Average Parameters for Definition of Groups

Group	Range N	Average K	Average M	Average Q	Average D	Average lPl	Average HR	Average NHR
1	5-14	2.667	1.852	3.333	2.111	0.111	0.689	0.681
2	15-24	2.296	1.704	3.333	2.333	0.481	0.689	0.696
3	25-34	3.037	1.778	3.333	3.111	0.667	0.693	0.707
4	35-44	2.148	1.926	3.333	3.370	1.852	0.715	0.685
5	45-54	2.259	1.741	3.333	3.741	1.889	0.726	0.704
6	55-64	2.630	1.778	3.333	4.741	3.000	0.711	0.726
7	65-74	2.333	1.889	3.333	5.407	2.889	0.689	0.722
8	75-84	2.444	1.741	3.333	5.333	3.704	0.711	0.715
9	85-94	2.815	1.926	3.370	7.074	3.444	0.674	0.704
10	95-104	2.370	1.704	3.333	7.037	5.074	0.693	0.730
11	105-114	2.667	1.926	3.667	8.111	6.185	0.711	0.719
12	115-124	2.370	1.704	3.667	8.926	6.741	0.711	0.719
13	125-134	2.111	1.889	3.667	10.556	7.111	0.685	0.700
14	135-144	2.111	1.926	3.667	10.815	7.259	0.674	0.715
15	145-154	2.148	2.037	3.630	13.519	6.037	0.726	0.719
16	155-164	2.481	1.963	3.630	13.593	7.852	0.707	0.674
17	165-174	2.148	2.000	3.630	15.000	9.444	0.700	0.704
18	175-184	2.074	1.852	3.630	13.963	10.593	0.681	0.693
19	185-194	1.963	1.778	3.519	14.111	9.667	0.696	0.719
20	195-204	1.963	1.926	3.630	17.074	8.963	0.719	0.719

98.5 percent (proven) optimality on the average. The average degree of optimality from applying the greedy heuristic was 63.9 percent.

When comparing UFS and FS, we observe that, for a small N (< 30), the focused search performs similarly to the unfocused search. This can be explained by the fact that when the size of the search space is manageable with the direct use of GA, then reducing the size of the search space does not help. As can be seen in Fig. 5, when N increases, the advantage of the focused search becomes more significant.

TABLE 9
Experimental Results in Terms of the Number of Generations and Quality of Solutions

Group	Average # Iterations UFS	Average # Iterations FS	Average Quality of Solutions UFS	Average Quality of Solutions FS	Average Quality of Solutions Greedy
1	40	2	0.931	0.930	0.522
2	55	3	0.942	0.973	0.623
3	57	13	0.884	0.989	0.571
4	60	15	0.881	0.982	0.618
5	65	40	0.875	0.989	0.655
6	70	41	0.853	0.990	0.647
7	65	40	0.851	0.985	0.649
8	67	45	0.829	0.988	0.651
9	70	50	0.804	0.995	0.617
10	65	55	0.824	0.988	0.658
11	72	60	0.793	0.986	0.635
12	75	50	0.821	0.997	0.670
13	72	55	0.802	0.986	0.650
14	64	58	0.811	0.990	0.669
15	73	62	0.816	0.990	0.667
16	60	40	0.778	0.992	0.634
17	73	60	0.806	0.992	0.675
18	65	57	0.795	0.990	0.663
19	67	55	0.803	0.986	0.689
20	63	56	0.797	0.987	0.669

6.5.2 Number of Generations

We consider the number of generations necessary to generate the final solution as an indicator of the solution time. For all 600 cases, the average number of generations for the unfocused search is 64.3, while the average number for the focused search is 37.7. This suggests that the focused search not only generates better quality solutions but also takes less time to generate them. We also observe that the difference becomes less important when N increases. The differences in the average number of generations for the 20 groups are shown in Fig. 6.

6.5.3 Relative Quality Improvement Comparing Focused Search with Unfocused Search

$$\mathrm{Improve(FS,\ UFS)} = \left[F(x^2) - F(x^3)\right]/F(x^3). \qquad (17)$$

The results of the 600 test cases are shown in Fig. 7. What we observe from this is that FS generally performs better

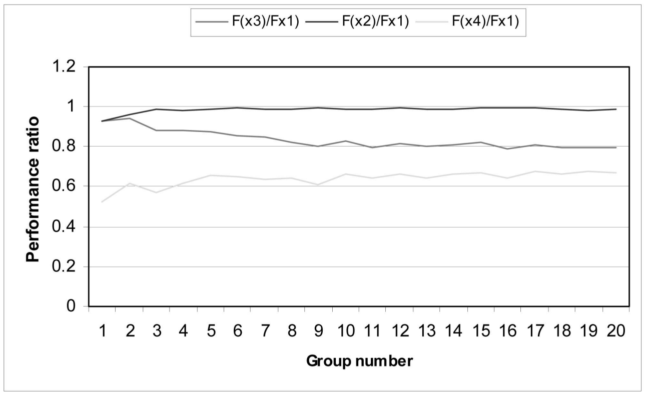

Fig. 5. Comparison between FS, UFS, and greedy search (average performance ratio per group).

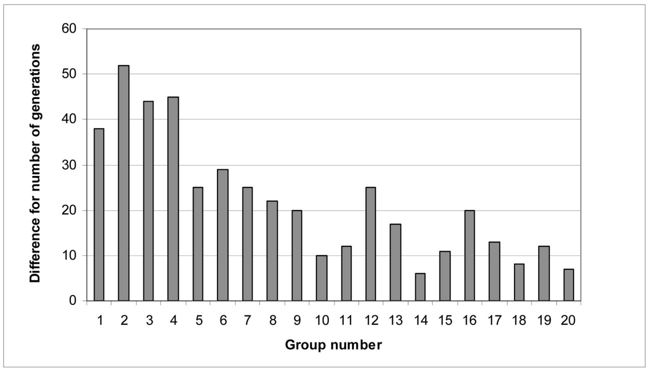

Fig. 6. Difference in the average number of generations between UFS and FS for the 20 groups of test cases.

than UFS, except in the case of small problems. With larger problems, the performance ratio improves toward FS.

6.5.4 Degree of Improvement in Relation to the Number of Releases

As another direction of empirical analysis, we studied the same performance ratio as in Fig. 7, with the only difference that the number K of planning periods was fixed. This results in four scatter plots, as shown in Fig. 8, for $K = 1..4$. The results confirm our hypothesis that the performance ratio gets better with more complex problems. This effect is further strengthened by increasing the number K. This is independent of the number of releases.

6.5.5 Relative Improvement Comparing Focused Search against Greedy Search

The final investigation was carried out to compare focused search with the application of the greedy search heuristic.

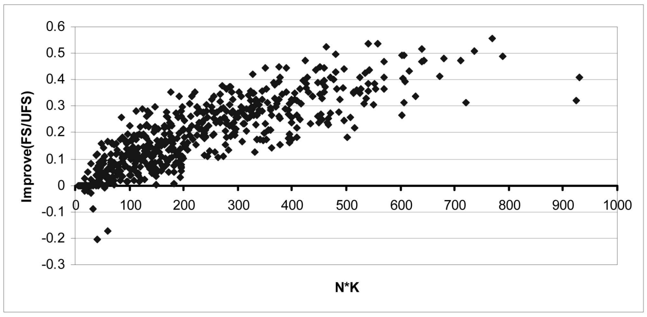

Fig. 7. Scatter plot of relative improvement from unfocused to focused search Improve(FS, UFS) in dependence of the size of the problem N*K.

The relative improvement Improve(FS, Greedy) is defined as the following ratio:

$$\mathrm{Improve(FS,\ Greedy)} = \left[\mathrm{F}(x^2) - \mathrm{F}(x^4)\right]/\mathrm{F}(x^4). \tag{18}$$

In Fig. 9, FS always performs better, and the relative improvement is concentrated in the range between 0.5 and 1 for larger problems.

7 Threats to Validity

The key proposition of this paper is that planning method $\mathrm{RASORP_{OPTIMIZE}}$ is applicable to resource-driven release planning. The solutions obtained from the method improve both the unfocused search and greedy search-based results. We discuss the key threats to the validity of these propositions.

7.1 Construct Validity

The general importance of resource-driven product release planning was confirmed independently in different

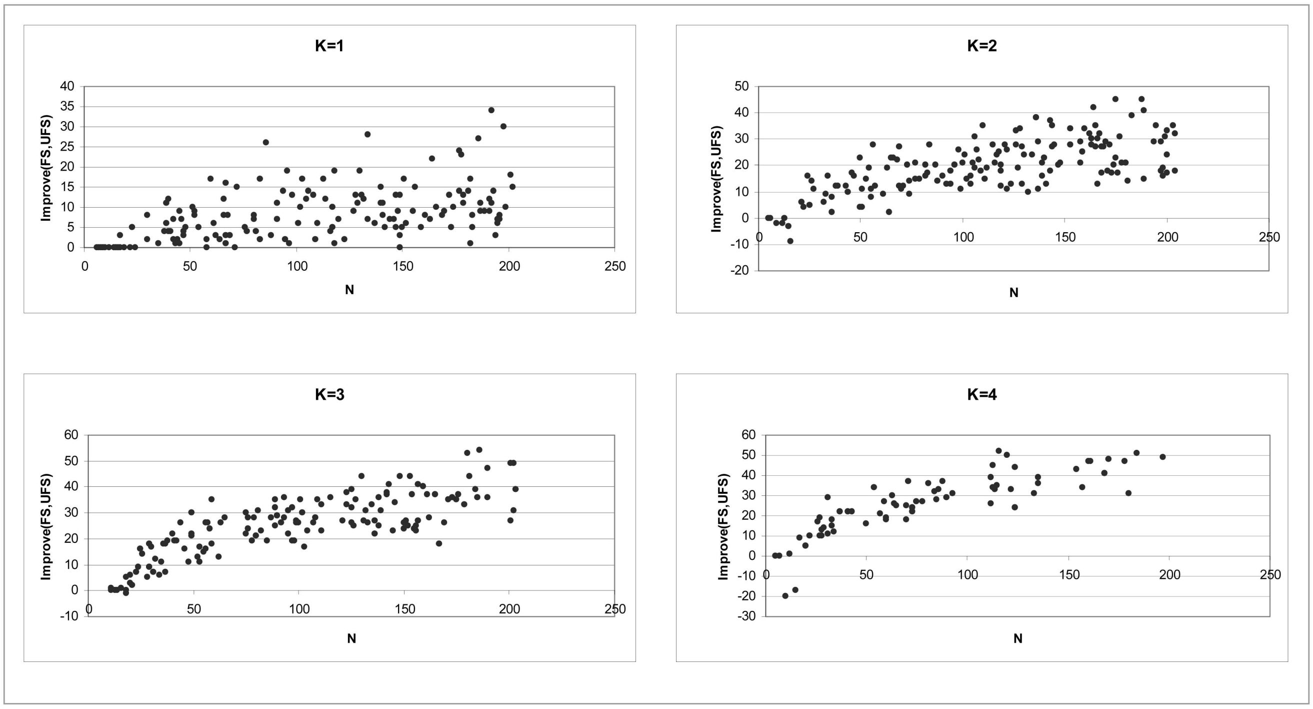

Fig. 8. Improve(FS, UFS) in relation to the number of releases for K - 1..4.

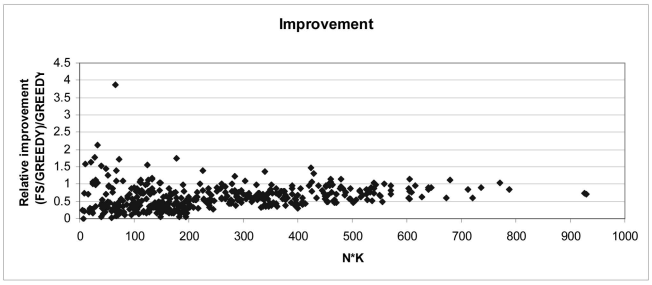

Fig. 9. Improve(FS, Greedy) in the comparison of FS and greedy search.

real-world case studies (e.g., [3], [22], [30]). The model formulated in RASORP ignores organizational, process, and human-related factors. We have made some further simplifications to allow tractability of the problem (such as the assumption that only one developer is working at a task at a time). In general, there is a trade-off between the degree of inclusion of more and more realistic (and thus complex) constraints and the capability to formally handle the resulting problem.

The proposed solutions can be taken as valid input for the final human decision making. This becomes especially true after running a sequence of scenarios exploring possible uncertainties in the model. We argue that a good solution to a reasonable approximation of the real-world problem is a step forward in determining a practically applicable solution and is far better than any ad hoc solution.

To compare results, we have taken the results from greedy search as the baseline. In reality, we usually expect the situation that no rigorous planning method is even applied at all. Instead, decisions are made on a subjective basis. From empirical results obtained from a series of controlled experiments, it was observed that the release planning results obtained from manual planning are worse and even less trusted than those from objective planning [11].

7.2 Internal Validity

The only threats to internal validity have been caused by the inherent randomness of running genetic algorithms. We have reduced this threat by considering the averages of 600 test cases. The average results are considered representative for the performance of genetic search of this class of problems. There were no internal validity threats when solving the relaxed problem RASORP*.

7.3 External Validity

Even though we were running a large set of 600 randomly generated problems with more or less the same (qualitative) results, this does not automatically imply external validity of the results. However, it emphasizes at least that the proposed method is likely to be superior to UFS and even more so for the greedy search. As a tendency, the advantage becomes greater the larger and the more complex the problem is.

There is a range of uncertainty factors in modeling the problem (such as effort estimation, productivity measurement, or the feature value proposition). The results are not claimed to be the one and only way to run the process. Instead, they are qualified planning results supposed to be adjusted in the course of the project. A recent case study conducted at Chartwell Technology demonstrated that the application of the focused search method allows 1) reducing the time needed for generating acceptable resource staffing plans, 2) generating plans of proven quality about 40 percent better than manual plans, and 3) supporting the various types of replanning necessary for varying parameters, budgets, and resource [22].

There is no exact boundary for the size of problems that can be solved in reasonable time with reasonably good quality of solutions. This also means that there might be practical problems of large scale where the method will either need more computational time or not find a good solution at all.

8 Summary and Conclusions

Software release planning including the assignment of resources to perform tasks necessary to implement the features is of significant practical importance for planning and performing incremental software development. The expected practical benefit of the planning method is to provide release plan solutions that achieve a better overall business value (e.g., expressed by the degree of stakeholder satisfaction) by better allocation of resources. Without ignoring the importance of the human expert in this process, the main contribution of the paper is seen in making the overall process more objective and more qualified in terms of the results. This also increases transparency of solutions, especially when compared to making subjective ad hoc decisions.

Plans for resource allocation and provision of features need to be adjusted in the course of a project. It is important to have a qualified starting point for these necessary adjustments caused by different changes in project parameters. From a technical perspective, the importance of our results is based on the following observations:

1. The combined problem of release planning, including resource allocation for related tasks, was never approached systematically before.
2. Consideration of different levels of productivity on the developer side is of key importance in real-life situations.
3. The proposed solution method combines GA and integer linear programming, which reduces the search space and overcomes current weaknesses of GAs in general and solution methods for release planning in particular.
4. The release and resource allocation results have a proven degree of optimality.
5. The method was shown to be applicable to problems with up to about 200 features and 600 tasks.

The problem under consideration is complex and includes additional factors that are not taken into account in this research. In terms of the relationship between features, the strict precedence could be considered in addition to the weak precedence as handled in this paper. On the task level, strict end-start precedence relationship is another extension that could be investigated. In that case, in which a waterfall development process model is assumed, design must be finished before implementation can start. Also, the assumption that just one developer can work on one task at any given time is not always the case. All of these issues would not change the two-phase architecture of the solution method. However, the specific formulation of Phase 1 and/or Phase 2 needs to be modified to accommodate extensions to the model. To achieve a reasonable trade-off between solution time and solution quality, further adjustments in the settings of the GAs would need to be made.

More comprehensive empirical analysis with real-world data is necessary to improve our understanding of the maximum problem size solvable by the proposed method. Overall, we plan to include the proposed method as a new component of the existing decision support system ReleasePlanner® [35]. For that, further fine-tuning of parameters in GA and comparison with other metaheuristics will be done. This integration will also allow us to apply and evaluate the method more comprehensively in the industry.

Appendix A
Greedy Heuristic

The heuristic is based on greedy search. For that purpose, the variables are handled in a determined order. The order is defined by a score, assuming that the best individual variable to handle has the highest score. The heuristic consists of four steps:

- Step 1: Compute a score for each feature using the procedure SCORE.
- Step 2: Sort the features in descending order of score. The result is a permutation π' of N features.
- Step 3: Transform π' to a compatible permutation π using procedure TRANSFORM.
- Step 4: Apply the serial scheduling scheme (procedure SERIALIZE) on the permutation π to obtain the feasible solution (x, u).

Procedure SCORE

Step 1: Compute the demand of each resource using

$$\mathrm{SumMS(m)} = \Sigma_{\mathrm{n=1..N}}\mathrm{r(m,\ n)} \quad \text{for m} = 1..\mathrm{M} \\ \text{(for nonhuman resources)}, \tag{1}$$

$$\mathrm{SumMS(M+q)} = \Sigma_{\mathrm{n=1..N}}\mathrm{w(n,\ q)} \quad \text{for q} = 1..\mathrm{Q} \\ \text{(for human resources)}. \tag{2}$$

Step 2: Compute the capacity of human resource with regards to each task by

$$\mathrm{WC[q]} = \mathrm{t[K]}\ \Sigma_{\mathrm{d=1..D}}\ \mathrm{prod(d,\ q)} \quad \text{for q} = 1..\mathrm{Q}. \tag{3}$$

Step 3: Compute the strictness of each resource:

$$\mathrm{Strictness(m)} = \mathrm{SumMS(m)/c(m,\ 1)} \text{ for m} = 1..\mathrm{M} \\ \text{(for non}-\text{human resources)}, \tag{4}$$

$$\mathrm{Strictness(M+q)} = \mathrm{SumMS(M+q)/WC[q]} \text{ for q} = 1..\mathrm{Q} \\ \text{(for human resources)}. \tag{5}$$

Step 4: Determine the most restrictive resource m^* by

$$\mathrm{m}^* = \mathrm{argmax}_{\mathrm{m=1..M+Q}}\{\mathrm{Strictness(m)}\}. \tag{6}$$

Step 5: Compute the score of each feature using

$$\mathrm{SC(n)} = \mathrm{v(n,\ 1)/r(m^*, n)} \text{ if } 1 \le \mathrm{m}^* \le \mathrm{M}, \tag{7}$$

$$\mathrm{SC(n)} = \mathrm{v(n,\ 1)/w(m^*{-}M,\ n)} \text{ if } \mathrm{M}+1 \le \mathrm{m}^* \le \mathrm{M}+\mathrm{Q}. \tag{8}$$

Procedure TRANSFORM

```
Input: Permutation $\pi'$,
Output: Permutation $\pi$

  Set $n1 = 1$ (index of $\pi'$)
  Set $n2 = 0$ (index of $\pi$)
  While $n2 < N$ do (permutation $\pi$ incomplete)
    If $((\pi'(n1) = -1)$ or
        (there is n such that $(n, \pi(n1)) \in P$ and
          $n \notin \{\pi(1)..\pi(n2)\}))$
    then Set $n1 = n1 + 1$
    If $n1 > N$ then set $n1 = 1$ endif
    Else Set $n2 = n2 + 1$, Set $\pi(n2) = \pi'(n1)$,
        Set $\pi'(n1) = -1$, Set $n1 = n1 + 1$
    If $n1 > N$ then set $n1 = 1$ endif
   Endif
 Endwhile
```

Procedure SERIALIZE

```
Input: Permutation $\pi$
Output: Plan $(x, u)$

For $n = 1$ to $N$ do
  For $q = 1$ to $Q$ do
    Set $d^*(q) = developer\ finishing\ the\ task\ (\pi(n).q)$ at the
    earliest date $t^*(q)$
    If $t^*(q) \le T$ then (still in the time limit)
    Assign $task(\pi(n), q)$ to developer $d^*(q)$
    Endif
  Enddo
    If all task $q$ are assigned then
    Set $x(\pi(n), k^*) = 1$ where $k^* = min_k max_s\{t^*(q)\} \le Tk\}$
    Else Set $x(\pi(n), k) = 0$ for all $k$
    Endif
Enddo
```

Acknowledgments

The authors would like to thank the associate editor and the anonymous referees for their valuable comments and suggestions. Continuous discussions with their colleagues Vahid Garousi, Jim McElroy, and Dietmar Pfahl helped them to reformulate the content to make it more readable. Kornelia Streb has provided them with substantial technical support in the preparation of the paper. The authors also thank the Alberta Informatics Circle of Research Excellence (iCORE) and the Alberta Ingenuity Fund for their financial support of this research.

REFERENCES

[1] S. Acuña, N. Juristo, and A.M. Moreno, "Emphasizing Human Capabilities in Software Development," *IEEE Software,* vol. 23, no. 2, pp. 94-101, Mar./Apr. 2006.

[2] M. Crissis, M. Konrad, and S. Shrum, *CMMI—Guidelines for Process Integration and Product Improvement.* Addison-Wesley, 2006.

[3] A. Amandeep, G. Ruhe, and M. Stanford, "Intelligent Support for Software Release Planning," *Proc. Fifth Int'l Conf. Product Focused Software Process Improvement,* pp. 248-262, 2004.

[4] J. Blazewicz, W. Domschke, and E. Pesch, "The Job Shop Scheduling Problem: Conventional and New Solution Techniques," *European J. Operational Research,* vol. 93, no. 1, pp. 1-33, 1996.

[5] L. Briand, J. Feng, and Y. Labiche, "Experimenting with Genetic Algorithms and Coupling Measures to Devise Optimal Integration Test Orders," *Software Eng. with Computational Intelligence,* pp. 204-234, Kluwer Academic Publishers, 2003.

[6] P. Carlshamre, K. Sandahk, B. Regnell, and J. Nattoch Dag, "An Industrial Survey of Requirements Interdependencies in Software Release Planning," *Proc. Fifth IEEE Int'l Symp. Requirements Eng.,* pp. 84-91, 2001.

[7] C. Chang, M. Christensen, and T. Zhang, "Genetic Algorithms for Project Management," *Annals of Software Eng.,* vol. 11, pp. 107-139, 2001.

[8] J. Clarke, J.J. Dolado, M. Harman, R. Hierons, B. Jones, M. Lumkin, B. Mitchell, S. Mancoridis, K. Rees, M. Roper, and M. Shepperd, "Reformulating Software Engineering as a Search Problem," *IEE Proc. Software,* vol. 150, pp. 161-175, 2003.

[9] T. Cormen, C. Leiserson, and Z. Rivest, "Greedy Algorithms," *Introduction to Algorithms,* chapter 17, p. 329, MIT Press, 1990.

[10] L. Davis, "Job Shop Scheduling with Genetic Algorithms," *Proc. Int'l Conf. Genetic Algorithms and Their Applications,* J.J. Grefenstette, ed., pp. 136-140, 1983.

[11] G. Du, J. McElroy, and G. Ruhe, "A Family of Empirical Studies to Compare Informal and Optimization-Based Planning of Software Releases," *Proc. Fifth ACM-IEEE Int'l Symp. Empirical Software Eng.,* pp. 212-221, Sept. 2006.

[12] L.J. Eshelman and J.D. Schaffer, "Preventing Premature Convergence in Genetic Algorithms by Preventing Incest," *Proc. Fourth Int'l Conf. Genetic Algorithms,* pp. 115-122, 1991.

[13] N. Fenton, W. Marsh, M. Neil, P. Cates, S. Forey, and T. Tailor, "Making Resource Decisions for Software Projects," *Proc. 26th Int'l Conf. Software Eng.,* pp. 397-406, May 2004.

[14] *GALib,* http://lancet.mit.edu/ga/, 2008.

[15] M. Garey and D. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W.H. Freeman, 1990.

[16] F. Glover and M. Laguna, *Tabu Search.* Kluwer Academic, 1997.

[17] D. Greer and G. Ruhe, "Software Release Planning: An Iterative and Evolutionary Approach," *Information and Software Technology,* vol. 46, pp. 243-253, 2004.

[18] S. Hartmann, "A Competitive Genetic Algorithm for Resource-Constrained Project Scheduling," *Naval Research Logistics,* vol. 45, pp. 733-750, 1998.

[19] J.H. Holland, *Adaptation in Natural and Artificial Systems.* Univ. of Michigan Press, 1975.

[20] www.ilog.com, 2008.

[21] A.S. Jain and S. Meeran, "A State-of-the-Art Review of Job-Shop Scheduling Techniques," technical report, Dept. of Applied Physics, Electronics, and Mechanical Eng., Univ. of Dundee, 1998.

[22] P. Kapur, A. Ngo-The, G. Ruhe, and A. Smith, "Optimized Staffing for Product Releases and Its Application at Chartwell Technology," *J. Software Maintenance and Evolution,* vol. 20, pp. 365-386, 2008.

[23] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems.* Springer, 2003.

[24] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, "Optimization by Simulated Annealing," *Science,* vol. 220, pp. 671-680, 1983.

[25] A. Land and A. Doig, "An Automatic Method for Solving Discrete Programming Problems," *Econometrica,* vol. 28, pp. 497-520, 1960.

[26] D. Mattfeld and C. Bierwirth, "An Efficient Genetic Algorithm for Job Shop Scheduling," *European J. Operational Research,* vol. 155, pp. 616-630, 2004.

[27] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations.* John Wiley & Sons, 1990.

[28] G. Meyers, "A Controlled Experiment in Program Testing and Code Walkthroughs/Inspections," *Comm. ACM,* vol. 21, pp. 760-768, 1978.

[29] J.E. Mitchell, "Integer Programming: Branch-and-Cut Algorithms," *Encyclopedia of Optimization,* vol. 2, Kluwer Academic, pp. 519-525, 2001.

[30] J. Momoh and G. Ruhe, "Release Planning Process Improvement —An Industrial Case Study," *Software Process: Improvement and Practice,* vol. 11, pp. 295-307, 2006.

[31] A. Ngo-The and G. Ruhe, "A Systematic Approach for Solving the Wicked Problem of Software Release Planning," *Soft Computing,* special issue on soft computing in software engineering, vol. 12, pp. 95-108, 2008.

[32] F. Padberg, "Scheduling Software Projects to Minimize the Development Time and Cost with a Given Staff," *Proc. Eighth Asia-Pacific Software Eng. Conf.,* pp. 187-194, 2001.

[33] M. Pawlak, "Application of Evolution Program to Resource Demand Optimization in Project Planning," *Proc. IEEE Int'l Conf. Evolutionary Computation,* vol. 1, pp. 435-439, 1995.

[34] A.B. Pyster and R.H. Thayer, "Guest Editors' Introduction: Software Engineering Project Management 20 Years Later," *IEEE Software,* vol. 22, pp. 18-21, 2005.

[35] www.releaseplanner.com, 2008.

[36] G. Ruhe and A. Ngo-The, "Hybrid Intelligence in Software Release Planning," *Hybrid Intelligent Systems,* vol. 1, pp. 99-110, 2004.

[37] G. Ruhe and O. Saliu, "The Art and Science of Software Release Planning," *IEEE Software,* vol. 22, pp. 47-53, 2005.

[38] O. Saliu and G. Ruhe, "Supporting Software Release Planning Decisions for Evolving Systems," *Proc. 29th IEEE/NASA Software Eng. Workshop,* Apr. 2005.

[39] G. Stark, A. Skillicorn, and R. Ameele, "An Examination of the Effects of Requirements Changes on Software Maintenance Releases," *J. Software Maintenance: Research and Practice,* vol. 11, pp. 293-309, 1999.

[40] J.M. J van den Akker, S. Brinkkemper, G. Diepen, and J. Versendaal, "Software Product Release Planning through Optimization and What-If Analysis," *Information and Software Technology,* vol. 50, no. 2008, pp. 101-111, 2005.

[41] J. van Gurp, J. Bosch, and M. Svahnberg, "Managing Variability in Software Product Lines," *Proc. Landelijk Architectuur Congres,* 2000.

An Ngo-The received the BSc degree in mathematics and computer science from the University of Ho-Chi-Minh City, Vietnam, in 1985 and the MBA degree from the French-Vietnamese Center for Management Education, Vietnam, in 1997. He received the DEA (equivalent to MSc) degree in decision support in 1998 and the PhD degree in computer science (decision support in operational research) from LAMSADE, University Paris Dauphine, France thanks to a scholarship from the ESSEC Doctoral Program.

Günther Ruhe is the industrial research chair in software engineering at the University of Calgary. His main results and publications are in software engineering decision support, software release planning, and software project management, as well as software measurement, simulation, optimization, and empirical research. From 1996 to 2001, he was the deputy director of the Fraunhofer Institute for Experimental Software Engineering. He is the author of two books, several book chapters, and more than 120 publications. He is a senior member of the IEEE and a member of the ACM, the IEEE Computer Society, and the German Computer Society (GI).