European Journal of Operational Research xxx (2013) xxx–xxx

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Interfaces with Other Disciplines

Skill-based framework for optimal software project selection and resource allocation

Fadi A. Zaraket, Majd Olleik, Ali A. Yassine [*]

Faculty of Engineering and Architecture, American University of Beirut, Beirut, Lebanon

ARTICLE INFO

Article history:
Received 8 November 2012
Accepted 24 September 2013
Available online xxxx

Keywords:
Resource allocation
Project selection
Software development
Software skills
Meta-heuristic

ABSTRACT

This paper presents a conceptual framework and a mathematical formulation for software resource allocation and project selection at the level of software skills. First, we introduce a skill-based framework that considers universities, software companies, and potential projects of a country. Based on this framework, we formulate a linear integer program PMax which determines the selection of projects and the allocation of human resources that maximize profit for a certain company. We show that PMax is NP-complete. Therefore, we devise a meta-heuristic, called *Tabu Select and Greedily Allocate* (TSGA), to overcome the computational complexities. When compared to PMax running on CPLEX, TSGA performs 15 times faster with an accuracy of 98% on small to large size problems where CPLEX converges. On larger problems where CPLEX does not return an answer, TSGA computes a feasible solution in the order of minutes.

For demonstration, the proposed skill-based framework and the corresponding mathematical model are applied to Lebanon by performing two surveys on the Lebanese software industry and academia. The case study shows that the proposed framework and mathematical model can be used in practice to improve project selection and resource allocation decisions in software companies.

1. Introduction

The project management literature generously addressed *resource allocation* in general (Brucker, Drexl, Mohring, Neumann, & Pesch, 1999; Herroelen & Leus, 2005; Herroelen, Reyck, & Demeulemeester, 1998; Oezdamar & Ulusoy, 1995) and project portfolio selection (Archer & Ghasemzadeh, 1999; Meade, 2002) as two separate problems. However, there is great room for suggesting models that solve the two interdependent problems simultaneously. In particular, the authors in Gutjahr, Katzensteiner, Reiter, Stummer, and Denk (2010), Yoshimuraa, Fujimia, Izuia, and Nishiwakia (2006) present models to solve the two problems one after the other assuming that the profitability of a chosen portfolio of projects is totally independent from the resources allocated on each one of them. We argue that in reality, the cost of the development of a project highly depends on the human workforce that works on it (Acuna, Juristo, & Moreno, 2006).

Additionally, *software* specific studies concerning project selection and resource allocation are still scarce (Otero, Centeno, Ruiz-Torres, & Otero, 2009). The problem of resource allocation in *software project development* is a unique challenge due to specific characteristics of software projects and software developers (Kan, 1994). The work of Acuna et al. (2006) finds that human resource allocation on software projects is generally left to the judgment of experts such as software team leaders and project managers. Although judgments are educated guesses that work in practice, however, managers lack mathematical tools to develop and assess project schedules and associated human resource allocation (Padberg, 2001). In particular, Plekhanova (1999) and Otero et al. (2009) observe that project managers map each software task to one skill when allocating human resources to tasks. Otero suggests that tasks should be mapped to several skills (Otero et al., 2009). Ngo-The maps a task into a set of skills and optimally allocates resources in release planning (Ngo-The & Ruhe, 2009).

In this paper, we propose a *skill-based framework* (SBF) that considers software projects, software companies and software related academia at the level of basic skills. That is, skills are the common thread that cuts across these three domains and are therefore at the center of our proposed framework. Educational processes, represented by *universities*, generate skills and form software development human resources that can be categorized into *talent classes*. Each talent class: $S \mapsto L$ is characterized as a map from skills S to a strength level $L = \{none, weak, average, good, excellent\}$. We characterize *software projects* by collective strength levels of required skills. The required skills and their associated levels are based on expert estimations where experts are project managers and

* Corresponding author. Tel.: +961 1 350 000.
E-mail addresses: fz11@aub.edu.lb (F.A. Zaraket), mbo01@aub.edu.lb (M. Olleik), ay11@aub.edu.lb (A.A. Yassine).

0377-2217/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.ejor.2013.09.035

software team leaders. We denote by *potential projects* those projects that can be developed in a given country. The process of developing software *requires* skills. Companies in a given country select from a set of potential projects. Each company then allocates its human resources in the form of capita per talent class to develop the selected projects. The company does the selection and allocation with maximum profit as a target. Companies in turn enrich their human resources via (1) strengthening existing skills and (2) producing skills that are not covered in current curricula. We assume that skills produced during development work will eventually make it into curricula due to the interaction between the industry and the academia. We refine our skill set S to include the skills produced by the industry S_I and assume *none* as the strength level for fresh graduates in such skills.

Based on the above framework, we build an integer linear program to optimally select a portfolio of projects and allocate resources (i.e. talents) to them such that the allocated talents satisfy the skills required by the selected projects. PMax differs from previous project selection and resource allocation methods (Gutjahr et al., 2010; Yoshimuraa et al., 2006) in that it treats the resource allocation and project selection as a single problem and presents an integer linear program to solve it. PMax also provides a more realistic cost estimation formula as it considers a company to pay all its talents regardless of whether a talent is allocated or not. Additionally, PMax differs from existing models (Gutjahr et al., 2010; Otero et al., 2009; Yoshimuraa et al., 2006) in that it introduces the concept of critical skills. A critical skill j for a project p is a skill with a minimum level of expertise $\mathbf{\Theta}[p][j]$, where at least one allocated talent must possess to satisfaction a strength in $j \geqslant \mathbf{\Theta}[p][j]$, while other skills can be satisfied by strength levels of several talents.

In this paper, we make several contributions to the software management and operations research literature.

- We present SBF, a skill-based framework, to formalize the relationship between the software academia, the software industry of a given country and the potential projects at the level of skills. We formulate project selection and resource allocation as a mathematical program, PMax, which allows for critical skills and estimates cost more accurately.
- We introduce TSGA, a Tabu-based meta-heuristic, to overcome the computational complexity of PMax since PMax is shown to be NP-complete. We compare the performance of TSGA to CPLEX. TSGA performs 15 times faster than CPLEX and reaches an optimal solution 64% of the time. On average, the profit value obtained from TSGA is 98% of the optimal profit obtained by CPLEX.
- We conduct surveys covering the Lebanese software industry and academia to demonstrate how SBF can be implemented.

2. Literature review

The paper draws upon various streams of research form project management, analyses of skills and competencies required for software product development, and software estimation models and techniques. From the project management literature, we only focus on the literature that combines project portfolio selection, activity scheduling, and resource allocation. Although relevant, the voluminous literature on resource constrained project scheduling problem, including multi-project and multi-mode versions, will be ignored. Additionally, since our mathematical model is built on a new skill-based framework, we also review literature describing various skills and competencies required in a software development environment. Finally, since our proposed model requires the estimation of various input parameters relating to cost and duration of development activities, we also discuss these various software estimation techniques in this section.

2.1. Resource allocation

Gutjahr et al. (2010) present a mixed integer non-linear programming model for project selection and resource allocation while focusing on increasing the competences of the staff through experience. They decomposed the problem by applying a meta heuristic for project selection and then a greedy priority based heuristic for project scheduling and staffing. The main problem resides in the assumption that the cost of development of a certain project is assumed to be given independently of the resources that the model allocates on it. PMax differs by incorporating the cost of the allocated resources in the calculation of the project development cost.

Yoshimuraa et al. (2006) tackle the problems of project selection and resource allocation. They start by selecting the portfolio of projects that maximizes profit. Then they allocate a project leader for each project to end up with allocating the other human resources. In this paper, we consider that project selection and resource allocation are two interdependent problems that should be solved together.

Otero et al. (2009) presented a method that associates a set of required skills with each software task for the completion of the task. The method assigns available human resources to complete the required tasks. Otero's work addresses the situation when the available resources fall short of covering the required skills and minimizes the learning time based on rhetorical relations between missing and available skills. PMax differs in that it maximizes profit, extends skills to projects instead of tasks, and considers project selection concurrently with resource allocation.

Xiao et al. (2009) consider the time and cost optimization problem in project scheduling and present a near optimal genetic algorithm. Ngo-The and Ruhe (2009) consider the release planning problem in software development. They present an optimal allocation of resources that maximizes the value gained from the released features. Their solution does not isolate software release planning from resource allocation across several releases to solve the problem globally. They first compute an optimal solution for a relaxed version of the problem, which they use with a genetic algorithm to compute a near optimal solution for the original problem.

Finally, Barreto et al. present a project manager with utility functions to form a team that fits desired needs using constraint satisfaction approach. The desired needs could optimize several aspects such as expense, performance, or size (Barreto, de O. Barros, & Werner, 2008).

2.2. Cost estimation

There are two main approaches to software development effort estimation: judgment-based methods using group consensus techniques, and model-based methods using formal mathematical models (Boehm, Abts, & Chulani, 2000). Expert judgment techniques involve consulting with a group of software cost estimation experts to use their past experiences and arrive at an estimate (or to a consensus) for the cost and duration of the proposed project (Jorgensen, 2005). Formal models, on the other hand, are designed to provide some mathematical equations to perform effort estimation. These mathematical models could be based on rules-of-thumb, historical data, and analogies, and use inputs such as lines of code, number of functions to perform, and other cost drivers such as language, design methodology, skill-levels, risk assessments. The algorithmic methods have been largely studied and there are a lot of models that have been developed, such as

COCOMO models, the Putnam model, and function points based models (Boehm et al., 2000). No one method is necessarily better or worse than the other; in fact, their strengths and weaknesses are often complimentary to each other (Jorgensen, Boehm, & Rifkin, 2009). Jorgensen shows that formal models have been somewhat erratic, and do not provide higher accuracy than expert judgment (Jorgensen & Shepperd, 2007). Expert judgment-based estimation approaches are, by far, the most common used approaches by the software industry (Jorgensen & Shepperd, 2007). There is no substantial evidence in favor of the use of estimation models. There are even situations where expert estimates are expected to be more accurate than formal estimation models (Jorgensen, 2004). We adopt this point of view and rely on experts to populate the different parameters of PMax while keeping in mind the best practices for expert estimations (Jorgensen, 2004).

2.3. Software skills

Several researchers studied the software skill sets available in the industry to understand the software development dynamics. Andre, Baldoquin, and Acuna (2011) and Colomo-Palacios, Tovar-Caro, Crespo, and Gomez-Berbis (2010) defined several software development job positions and associated each position with a set of required skills. The positions are ordered in a hierarchical fashion, and a number of human resources with the needed skills are mapped to the positions to form a team. This set of skills includes technical skills, as well as a communication and team management skills.

Acuna and Juristo (2004) consider the positions as roles and push the mapping of positions to skills further. They consider psychological traits suitable for playing the role and compute a correspondence between software development roles and a likelihood of needed personality features. Trigo et al. (2010) investigate the most important software skills according to Chief Information Officers (CIO) in Spanish and Portuguese software companies. They consider 10 high level skills and they conclude that the most important skills are *business knowledge* and *help desk/user support* (Trigo et al., 2010). In our skill-based framework, we differ from Colomo-Palacios and Trigo in that we compute a granular set of detailed software related skills based on the study of the academia. In particular, we compute at least an order of magnitude larger skills set, where the skills are well defined formally in terms of course objectives and syllabi.

Lethbridge (2000) and Kitchenham, Budgen, Brereton, and Woodall (2005) try to determine the relevance of software skills taught at academia to the industry needs. They categorize skills into software engineering, scientific and general business and arts skills. They survey software graduates about the importance of their skills. The results reveal that *programming languages* are the most important skills that are rightfully well taught at universities while *mathematical* skills in general are taught more in depth than what the developers actually needed at work. We follow the same framework to generate software skills based on academia but we address CIOs of software companies to collect information about the employees of the companies. We refine our skill set based on the feedback of the CIOs. Intuitively, this provides a better management perspective of the value of skills. We also determine the importance of skills by analyzing the needs of the software projects developed in a certain country. We compare our findings for the Lebanese case with Lethbridge and Kitchenham in the results section.

3. Skill-based framework (SBF)

SBF is a novel framework that considers software projects, software companies and software related academia at the level of basic skills. SBF analyses universities and their curricula, software companies and their human resources, and potential software projects and their requirements in terms of skills. SBF considers the union of the skills taught at universities, strengthened informally at work, produced at work, and required in potential projects as the base skill set.

SBF partitions university graduates and current human resources into talent classes. Each talent class is defined by a talent class strength vector that determines the strength of a talent in all skills. SBF represents the companies as talent class cardinality vectors where an entry in a vector represents the number of human resources in a talent class. Software project requirements are expressed in terms of required skills and strength in required skills.

Fig. 1 shows how universities' students build strength in targeted skills. A university offers software curricula that teach several courses. Each course has specific outcomes that either develop existing software skills or teach completely new ones to students. In SBF, we consider all software courses from local universities to compile a list of all covered software related skills.

Developers working in local companies are likely the graduates of local universities, thus they possess a subset of the global skills taught in the country. In addition, two developers might possess a certain skill but with different strength levels. This might be due to different educational tracks or due to different work experiences. Thus, each developer possesses strength values in the compiled list of skills that reflect his/her proficiency.

We classify developers into several talent classes reflecting their skills and experiences. Developers belonging to the same talent class possess the same strengths in all the skills. For example, all developers belonging to talent class *A* possess a strength of 5 in the human computer interaction (HCI) skill and a strength of 4 in the object oriented programming (OOP) skill. Developers from the same talent class have the same compensation rate.

On the other hand, the analysis of project requirements determines skills and strength levels in those skills that are required for a successful completion of the projects. As shown in the right side of Fig. 1, SBF maps each required skill per project to a required strength level. The team working on a project must possess the required strength level in all required skills to be eligible to work on the project and complete within the set deadline. For example, if project "Navigate" requires strength of 10 in HCI and 7 in OOP, then a team of two developers from talent class *A* can work on it.

3.1. Determining strengths in skills

Academic programs are composed of a hierarchy of courses. Some courses are required and others are left as electives. Each course has a syllabus detailing its learning outcomes. We map a learning outcome to one or more software skills. The term *hosting course* of a set of skills denotes the course that teaches those skills. Students acquire a different strength level in each software related skill depending on several parameters.

- The choice of elective courses that the student takes indicates the skills he/she possesses.
- Additionally, the performance of a student in a course, denoted by *perf*, is directly related to the strength in the skills that the course teaches.
- For a given skill, and a given course that hosts this skill, the number of courses that consider the *hosting course* as a prerequisite, denoted by *prereq*, is indicative of how much the skills taught in the *hosting course* are fundamental.
- Skills are more fundamental if they are required in several courses. Intuitively, this means the skills are further sharpened through repetition.

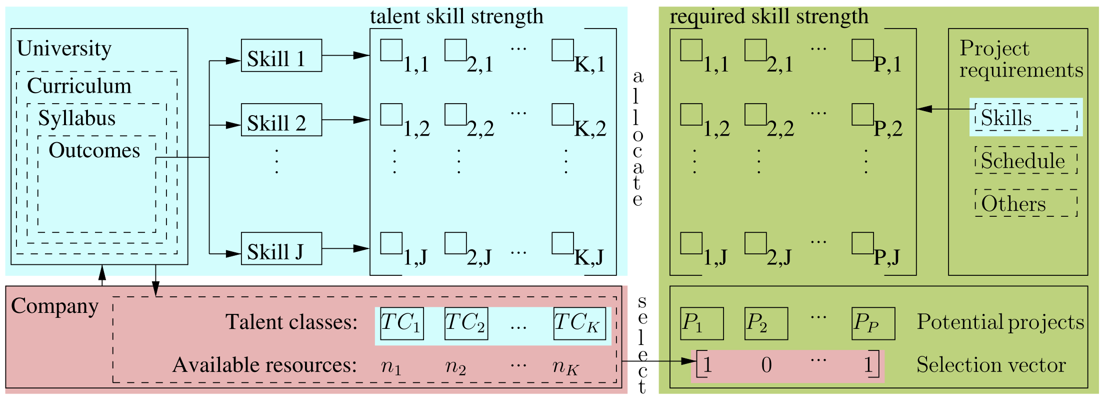

Fig. 1. Skill-based framework for the analysis of software projects, companies, and academia.

- Skills in courses with higher depth in the prerequisites graph are given higher weights because they represent advanced courses taught at advanced stages. The *depth* of a hosting course corresponds to its position in the prerequisites graph and is a good indicator of the level of the unique skills it introduces.

We propose to compute the strength of a student in a given skill as follows.

$$strength = perf \times (1 + prereq) \times depth \tag{1}$$

Eq. (1) describes the strength of a human resource in a skill as a product of performance, frequency of the skill in the curriculum and the introduction level of the skill. The performance of a student *perf* ranges over either passing (1), good (2), or excellent (3). We do not consider weak students as we assume they do not make it into the software development market.

The industry survey determines the performance of existing human resources. The survey also ranks the existing human resources based on experience. We compute the strength of skills for existing human resources using Eq. (1) where *perf* represents performance, *depth* represents rank, and *prereq* represents original skills at hiring.

In SBF we compute the strength as in Eq. (1) and normalize the strength value over the range 0 to H, where H represents the highest normalized strength in a given skill.

3.2. Talent classes

SBF computes several representative talent classes to partition the talents generated by given universities. We considered two groups of students in each software-related academic program. In the first group, students take only the mandatory courses and select not to take software related elective courses. In the second group, students take the mandatory courses and all the software electives. In each group of students there are excellent students, good students and average students. This results in a total of six talent classes for each program where the first extreme represents the average students with only mandatory software related courses, and the other extreme represents the excellent students with all software related courses.

We assume that the rest of the students who do not belong to one of the two groups, fall between them in one of the six talent classes.

Given J skills, and K talent classes, we define $\boldsymbol{M}$ to be the talent classes $K \times J$ matrix such that $\boldsymbol{M}[k][j] \in [0, H]$ represents the strength of a developer from talent class k in skill j, where $1 \leqslant k \leqslant K$, $1 \leqslant j \leqslant J$. We represent a software company as a set of available human resources assumed all to be software developers. The human resources of a company are mapped to the set of talent classes. For each company, we define the resources vector $\boldsymbol{n}$ of size K such that $\boldsymbol{n}[k]$ is the number of software developers belonging to talent class k.

Similarly, each company is assumed to pay salaries for its developers in accordance with the strengths they possess. Therefore, we define the cost vector $\boldsymbol{c}$ of size K such that $\boldsymbol{c}[k]$ represents the money that the company pays per unit time for a software developer belonging to talent class k.

3.3. Representation of projects

A software project with an associated revenue price is defined by a set of requirement specifications communicated by the client. The technical specifications detail the skills required to complete the project, and also fix delivery deadlines. In SBF, we rely on experts to formulate technical requirements into required skills with required strength vectors of size J. We introduce the nominal total strength matrix $\boldsymbol{W}$, where P is the number of projects to consider, such that $\boldsymbol{W}[p][j]$ represents the strength that project p $(1 \leqslant p \leqslant P)$ *requires* in skill j.

In addition, SBF considers important skills by requiring that for each important skill, at least one member of the development team must have more than a required minimum. For this reason, we introduce a critical skills requirements $\boldsymbol{\Theta}$ matrix. The value $\boldsymbol{\Theta}[p][j] \in [0, H]$ indicates that project p *requires* that at least one developer in the development team allocated on p must possess strength in j greater than or equal to $\boldsymbol{\Theta}[p][j]$. This allows our framework to define some expertise requirements in skills depending on project needs.

The vector $\boldsymbol{d}$ of size P represents the deadlines of the projects such that $\boldsymbol{d}[p]$ indicates how many yearly quarters are available at time 0 before the deadline of p. The development team *respects* the deadline of project p by finishing working on p before $\boldsymbol{d}[p]$. Additionally, experts can estimate the nominal duration $\delta[p]$ that the development team *needs* to complete a given project p. We denote by that the nominal duration vector δ. Finally, vector $\boldsymbol{\pi}$ of size P is the revenue vector where $\boldsymbol{\pi}[p]$ is the revenue that project p *generates*.

4. Mathematical formulation – PMax model

Given P potential software projects and a set of available software developers distributed over K software talent classes, our objective is to select the portfolio of projects that maximizes profit and allocates resources optimally amongst them so that all selected projects are successfully completed before their deadlines. Let $T = max(\boldsymbol{d}[1], \ldots, \boldsymbol{d}[p], \ldots, \boldsymbol{d}[P])$ denote the planning horizon, i.e. the last deadline of the set of projects. We use yearly quarters as discrete time steps such that the allocation of resources occurs at each quarter, i.e. resources may be re-allocated quarterly.

Table 1 presents a list of symbols used throughout this paper. The resource allocation matrix $\boldsymbol{A}$ denotes the resources allocated to projects throughout the quarters. $\boldsymbol{A}[t]$ is a matrix of size $P \times K$, i.e. each $\boldsymbol{A}[t][p][k]$ is one element of $\boldsymbol{A}[t]$ that denotes the number of developers from talent class k working on project p during quarter $t, 1 \leqslant t \leqslant T$. Matrix $\boldsymbol{B}$ denotes the total allocated strength matrix such that

$$\forall t, \boldsymbol{B}[t] = \boldsymbol{A}[t].\boldsymbol{M} \tag{2}$$

The binary indicator vector γ of size P denotes the selected projects, such that $\gamma[p]$ is 1 if project p is selected and 0 otherwise. The binary selection matrix $\boldsymbol{S}$ denotes projects selected per quarter. The entry $\boldsymbol{S}[p][t]$ is 1 if resources were allocated to project p in quarter t and 0 otherwise.

Our objective is to maximize profit with available resources. Our decision variables are the entries of the $\boldsymbol{A}$ matrix. We also present the variables in $\boldsymbol{S}$ and γ, that are computable from $\boldsymbol{A}$, as decision variables to make the problem linear and we constrain them to $\boldsymbol{A}$. The total revenue $R = \gamma.\,\boldsymbol{\pi}$ is the revenue associated with the set of the selected and completed projects.

The binary indicator vector $\boldsymbol{b}$ of size T denotes the busy quarters, i.e., $\boldsymbol{b}[t]$ is 1 if the resources were allocated in quarter t and 0 otherwise.

The total cost of development is defined as $\zeta = \boldsymbol{c} \cdot \boldsymbol{n} \cdot \sum_{t=1}^{T} \boldsymbol{b}[t]$ and is calculated as the sum of the costs of each quarter. We assume that all employees receive salaries whether they are allocated to work on a project or not. The cost is added to the total cost if the company is working on at least one project during a chosen quarter. Thus the profit Q becomes:

$$Q = \gamma \cdot \boldsymbol{\pi} - \zeta \tag{3}$$

Constraint (C1) ensures that projects selected for work during a certain quarter have enough allocated resources in terms of total strengths. We compute an integer linear constraint without logic operations Li-1 that is equivalent to (C1). For each $\langle p,t \rangle$ pair, When $\boldsymbol{S}[p][t] = 0$, (C1) is satisfied. Similarly, so is $\forall j.\boldsymbol{B}[t][p][j] \geqslant 0$ from (Li-1) because $\boldsymbol{B}[t][p][j]$ is always greater than or equal to 0. When $\boldsymbol{S}[p][t] = 1$, (C1) and (Li-1) both get simplified to $\boldsymbol{B}[t][p][j] - \boldsymbol{W}[p][j] \geqslant 0$. This proves the equivalence between (C1) and its integer linear form (Li-1).

$$\forall p,t \cdot (\boldsymbol{S}[p][t] = 1) \Rightarrow (\forall j \cdot \boldsymbol{B}[t][p][j] \geqslant \boldsymbol{W}[p][j]) \tag{C1}$$

Table 1
Table of symbols.

J	Total number of skills
K	Total number of talent classes
T	Allocation horizon in quarters
P	Total number of projects
R	Total revenue
$\boldsymbol{n}$	Resources vector of size K
$\boldsymbol{c}$	Cost vector of size K
$\boldsymbol{\delta}$	Nominal duration vector of size K
$\boldsymbol{d}$	Deadline vector of size K
$\boldsymbol{\pi}$	Revenue vector of size P
γ	Project selection vector of size P
$\boldsymbol{b}$	Busy quarter vector of size T
$\boldsymbol{M}$	Talent classes matrix of size $(K \times J)$
$\boldsymbol{W}$	Nominal total strengths matrix of size $(P \times J)$
$\boldsymbol{\Theta}$	Critical skills requirement matrix of size $(P \times J)$
$\boldsymbol{S}$	Project selection matrix of size $(P \times T)$
$\boldsymbol{A}$	Resource allocation matrix of size $(T \times P \times K)$
$\boldsymbol{B}$	Total allocated strengths matrix during of size $(T \times P \times J)$
ε	Very small positive number

$$\forall p,t,j \cdot \boldsymbol{B}[t][p][j] - (\boldsymbol{S}[p][t] \times \boldsymbol{W}[p][j]) \geqslant 0 \tag{Li-1}$$

Constraint (C2) ensures that projects not selected for work during a quarter have no resources allocated to them. We do not provide a linear form of constraint (C2) because we show in Theorem 1 that (C2) is redundant at optimality.

$$\forall p,t \cdot (\boldsymbol{S}[p][t] = 0) \Rightarrow (\forall k, \boldsymbol{A}[t][p][k] = 0) \tag{C2}$$

Constraint (C3) guarantees that when a project is selected, it is actually completed. That is the number of quarters allocated to work on the project is at least equal to the number of nominal quarters needed to complete the project. We compute (Li-3) as an integer linear constraint version of (C3) in a similar manner to (Li-1). When $\gamma[p] = 1$, the bodies of the quantifiers in (C3) and (Li-3) are both reduced to $\sum_{t=1}^{T}(\boldsymbol{S}[p][t]) \geqslant \boldsymbol{\delta}[p]$. When $\gamma[p] = 0$, (C3) and (Li-3) are satisfied since $T \geqslant \boldsymbol{\delta}[p]$ and $\sum_{t=1}^{T}(\boldsymbol{S}[p][t]) \geqslant 0$.

$$\forall p \cdot (\gamma[p] = 1) \Rightarrow \left(\left(\sum_{t=1}^{T} \boldsymbol{S}[p][t] \right) \geqslant \boldsymbol{\delta}[p] \right) \tag{C3}$$

$$\forall p \cdot \sum_{t=1}^{T} \boldsymbol{S}[p][t] + (T \times (1 - \gamma[p])) \geqslant \boldsymbol{\delta}[p] \tag{Li-3}$$

Constraint (C4) guarantees that we do not work on projects that are not selected. We do not provide a linear form for (C4) because we show in Theorem 1 that it is redundant at optimality. Constraint (C5) ensures that allocated resources do not exceed available resources.

$$\forall p \cdot (\gamma[p] = 0) \Rightarrow \left(\sum_{t=1}^{T} [\boldsymbol{S}[p][t] = 0 \right) \tag{C4}$$

$$\forall k,t \cdot \sum_{p=1}^{P} \boldsymbol{A}[t][p][k] \leqslant \boldsymbol{n}[k] \tag{C5}$$

Constraint (C6) ensures that a project has no resources allocated to it after its deadline. Constraint (C7) guarantees that the allocated talents for a project contain at least one talent that satisfies the critical skills required to complete the project.

$$\forall p,t \in [\boldsymbol{d}[p],T) \cdot \boldsymbol{S}[p][t] = 0 \tag{C6}$$

$$\forall p,j,t \cdot (\boldsymbol{S}[p][t] = 1 \wedge \boldsymbol{\Theta}[p][j] > 0) \Rightarrow (\exists k \cdot \boldsymbol{A}[t][p][k] > 0 \wedge \boldsymbol{M}[k][j] \geqslant \boldsymbol{\Theta}[p][j]) \tag{C7}$$

In order to write (C7) as a set of integer linear constraints, we introduce a new four dimensional matrix of binary decision variables $\boldsymbol{f}$. (C7) is equivalent to the following four integer linear constraints.

$$\forall p,k,j,t \cdot \boldsymbol{A}[t][p][k] + \boldsymbol{f}[t][p][k][j] - \boldsymbol{S}[p][t] \geqslant 0 \tag{Li-7-1}$$

$$\forall p,k,j,t \cdot \boldsymbol{M}[k][j] + (\boldsymbol{f}[t][p][k][j] - \boldsymbol{S}[p][t]) \times \boldsymbol{\Theta}[p][j] \geqslant 0 \tag{Li-7-2}$$

$$\forall p,j,t \cdot \sum_{k=1}^{K} \boldsymbol{f}[t][p][k][j] \leqslant K - 1 \tag{Li-7-3}$$

$$\forall p,k,j,t \cdot \boldsymbol{f}[t][p][k][j] \in \{0,1\} \tag{Li-7-4}$$

Constraints (C8) and (C9) enforce the relation between the busy vector and the project selection matrix. In particular, if for a quarter a company is not busy, then it has no projects selected in that quarter. Otherwise, it has at least one project selected.

$$\forall t \cdot (\boldsymbol{b}[t] = 0) \Rightarrow (\forall p \cdot \boldsymbol{S}[p][t] = 0) \tag{C8}$$

$$\forall t \cdot (\boldsymbol{b}[t] = 1) \Rightarrow (\exists p \cdot \boldsymbol{S}[p][t] = 1) \tag{C9}$$

The integer linear form of (C8) is (Li-8). There is no need to present the linear form of (C9) as shown in Theorem 1. When $\boldsymbol{b}[t] = 0$, (C8) is equivalent to $\forall p \cdot \boldsymbol{S}[p][t] = 0$, and (Li-8) is reduced to $\forall t \cdot \sum_{p=1}^{P} \boldsymbol{S}[p][t] \leqslant 0$. Given that $\boldsymbol{S}[p][t] \geqslant 0$, then (Li-8) is equivalent to $\sum_{p=1}^{P} \boldsymbol{S}[p][t] = 0$ and thus equivalent to (C8). When $\boldsymbol{b}[t] = 1$, (C8) is satisfied. Similarly (Li-8) is satisfied since $P \geqslant \sum_{p=1}^{P} \boldsymbol{S}[p][t]$.

$$\forall t \cdot \sum_{p=1}^{P} \boldsymbol{S}[p][t] - (\boldsymbol{b}[t] \times P) \leqslant 0 \tag{Li-8}$$

Constraint (C10) dictates the types of all the decision variables.

$$\forall t, \forall p, \forall k \cdot \boldsymbol{A}[t][p][k] \in \mathbb{N}, \boldsymbol{S}[p][t] \in \{0,1\}, \gamma[p] \in \{0,1\}, \boldsymbol{b}[t] \in \{0,1\} \tag{C10}$$

Our objective is to select a portfolio of projects that the company can complete successfully with maximum profit Q and minimum effort subject to constraints (C1)–(C10). Note that effort $E = \sum_{t=1}^{T}\sum_{p=1}^{P} \boldsymbol{A}[t][p] \cdot \boldsymbol{c}$ is different from the cost ζ since cost considers the cost of employees for each quarter while effort ignores the cost of an employee in quarters where he is not allocated to a project. Thus, we formulate the objective function as shown in Eq. (4) where ε is a very small positive number.

$$OF = \gamma \cdot \boldsymbol{\pi} - \zeta - \varepsilon \cdot E \tag{4}$$

Constraints (C2), (C4), and (C9) are subsumed by the optimization as proven in Theorem 1 below, and consequently we relax our problem and solve for

$$\max_{C1,C3,C5-C8,C10} \gamma \cdot \boldsymbol{\pi} - \zeta - \varepsilon \cdot E \tag{PMax}$$

Theorem 1. *Let System1 be PMax and let E1 be the set of solutions to System1. Let System2 be max$_{C1\text{-}C10}$ (4) and let E2 be the set of solutions to System2. We have E1 = E2.*

Intuitively, maximizing profit guarantees (C9) and minimizing effort guarantees (C2) and (C4), however the formal proof for Theorem 1 is found in the Appendix available online.[1]

5. Complexity of PMax

We argue that PMax is NP-complete by reducing the knapsack (KP) problem to an instance of PMax.

Let the number of skills $J = 1$, all the potential projects $1 \leqslant p \leqslant P$ have duration $\delta[p] = 1$, and let the deadline $\boldsymbol{d}[p] = 1$. Let the number of talent classes $K = 1$ indicating that $\boldsymbol{M}$ has only one entry $\boldsymbol{M}[1][1]$ that we set to 1. Let $\boldsymbol{c}[1] = 0$ (i.e. resources are available for free). This special instance of PMax can be written as

$$\max_{\gamma}(\gamma \cdot \boldsymbol{\pi}) \tag{5}$$

subject to

$$\sum_{p=1}^{P} \gamma[p] \times \boldsymbol{W}[p][1] \leqslant \boldsymbol{n}[1] \tag{6}$$

and where

$$\forall p, \boldsymbol{A}[1][p][1] = \gamma[p] \times \boldsymbol{W}[p][1] \tag{7}$$

This is a 0–1 KP problem where $\boldsymbol{W}[p][1]$ is the weight of project p, $\boldsymbol{\pi}[p]$ is its value and $\boldsymbol{n}[1]$ is the capacity of the knapsack. Any solution to this problem is a solution to the KP problem. Given that KP is NP-complete (Garey & Johnson, 1990), we conclude that PMax has no deterministically polynomial algorithm as long as there is no such a known algorithm for any of the NP-complete problems. Let the number of skills $J = 2$, all the potential projects $1 \leqslant p \leqslant P$ have duration $\delta[p] = 1$, and let the deadline $\boldsymbol{d}[p] = 1$. Let the number of talent classes $K = 2$. Let $M[1][1] = M[2][2] = 1$ and $M[1][2] = M[2][1] = 0$. Let $\boldsymbol{c}[1] = \boldsymbol{c}[2] = 0$ (i.e. resources are available for free). This instance of PMax can be written as

$$\max_{\gamma}(\gamma \cdot \boldsymbol{\pi}) \tag{8}$$

subject to

$$\sum_{p=1}^{P} \gamma[p] \times \boldsymbol{W}[p][1] \leqslant \boldsymbol{n}[1] \tag{9}$$

$$\sum_{p=1}^{P} \gamma[p] \times \boldsymbol{W}[p][2] \leqslant \boldsymbol{n}[2] \tag{10}$$

and where

$$\forall p, \boldsymbol{A}[1][p][1] = \gamma[p] \times \boldsymbol{W}[p][1] \tag{11}$$

$$\forall p, \boldsymbol{A}[1][p][2] = \gamma[p] \times \boldsymbol{W}[p][2] \tag{12}$$

This is a two dimensional KP problem. Therefore, PMax has no known fully polynomial time approximation scheme (FPTAS) (Kellerer, Pferschy, & Pisinger, 2004).

6. Tabu select and greedily allocate

Given the complexity of PMax, solving medium and large size problems optimally is practically infeasible. The memory and run-time requirements can vary tremendously depending on the inherent complexity and the structure of the problem.

Problems of the real case size of $J = 197$, $K = 12$, $P = 35$ (as shown later in Section 7) and with a relatively small number of developers (in the order of tens) are sometimes infeasible if run on an 8 gigabyte quad-core machine using CPLEX. This behavior is justified by the NP-complete characteristic of the problem. For this reason, we devise Tabu Select and Greedily Allocate (TSGA), a method that implements a greedy allocation of resources on a set of chosen projects for completion then performs Tabu iterations on the project selection vector to improve on company profit.

6.1. Tabu select

Tabu Select (TSel) is an application of the Tabu Search metaheuristic on the selection of projects problem. As introduced by Glover (1989), Glover (1990), Tabu search takes a feasible solution, a set of constraints, an objective function, a set of aspiration criteria, and a set of stopping conditions. TSel explores the neighborhood space of the feasible solution beyond local optimality to improve on the value of the objective function. This means that TSel allows intermediate worse objective values in the hope of improving again later. Given a feasible solution of the problem, Tabu search considers it as its current configuration then it investigates the neighboring candidate solutions. The candidate solution that has the best objective and that does not break any Tabu conditions is then considered as the new configuration. Tabu moves are only allowed if the aspiration criteria are met. Meanwhile, TSel keeps track of the solution that generated the best objective so far. The algorithm terminates when the stopping conditions are satisfied.

We make use of TSel to determine the most profitable binary project selection vector γ^*, thus a configuration is an instance of γ. We define a move to a neighboring solution as a single bit flip in the configuration vector. The candidate moves are all the possible one-bit flips of the configuration vector resulting in candidate γ vectors. A Tabu list maintains all visited γ configuration and restricts revisiting it unless the aspiration criterion is met.

For Tabu Select, we define two aspiration criteria. We allow to designate a Tabu candidate move as a new configuration only when one of the following two events happens: (1) The profit associated with this move is higher than the best profit reached so far. (2) All candidate γ vectors are Tabu.

[1] http://webfea.fea.aub.edu.lb/fadi/dkwk/doku.php?id=pmax.

TSel terminates if one of the following stopping conditions takes place:

- If the number of iterations (*maxIterNum*) is exceeded without improving the best stored solution,
- When the allowed run time (*maxRunTime*) is exceeded,
- When all the candidate moves are Tabu for the *maxTabuConfNum* consecutive iteration.

6.1.1. Initial feasible solution

It is generally advisable that the initial feasible solution is computationally easy to generate and of good quality (Glover, 1989). Initially, TSel filters the infeasible projects. A project is feasible if its required strength in each of the J skills does not exceed the combined strength of the company resources in that skill and if its critical requirements are available in the company resources. For each feasible project p, $quarterVal[p]$ represents an estimation of the quarterly profit associated with p. The value $quarterVal[p]$ is equivalent to the revenue of p per quarter of development reduced by an estimate of the cost of the allocated resources on p.

$$quarterVal[p] = \frac{\boldsymbol{\pi}[p]}{\boldsymbol{\delta}[p]} - quarterCost[p] \tag{13}$$

To estimate $quarterCost[p]$, we determine first the skill j_m that maximizes $\boldsymbol{W}[p][j]$ over all skills. We assume that dividing $\boldsymbol{W}[p][j_m]$ by the average developer strength in j_m gives us an estimation of the number of developers that will work on p. We then multiply this estimation of the number of developers by the average salary ($averageC$) of a developer in the company in question.

$$quarterCost[p] = \frac{\boldsymbol{W}[p][j_m]}{averageStr[j_m]} \times averageC \tag{14}$$

Feasible projects are added to the vector of feasible projects *feasibleProj* ordered by the decreasing order of *quarterVal*.

During any phase of the solution, TSGA maintains a matrix $\boldsymbol{F}$ of unallocated resources such that $\boldsymbol{F}[t][k]$ indicates the number of unallocated developers belonging to class k during quarter t.

$$\boldsymbol{F}[t][k] = n[k] - \sum_{p=1}^{P} \boldsymbol{A}[t][p][k] \tag{15}$$

At startup, no allocations are yet made, thus $\forall t$, $\forall k$, $\boldsymbol{F}[t][k] = n[k]$. After selecting a project p, Greedily Allocate (GAlloc) checks if p is feasible given the matrix $\boldsymbol{F}$. If p is feasible, GAlloc allocates resources for p and modifies $\boldsymbol{F}$ removing the newly allocated resources on p. We generate the initial feasible solution γ_i according to the following:

1. Start with $\gamma_i = \boldsymbol{0}$
2. Select the next p belonging to *feasibleProj*
3. If Greedily Allocate (GAlloc) can allocate resources for p, then let $\gamma_i[p] = 1$
4. Goto 2

The initial project selection vector γ_i and the associated initial allocation matrix $\boldsymbol{A}_i$ are then fed to the Tabu Select iterations. We note here that if $\gamma_i = \boldsymbol{0}$, then no project is feasible and thus there is no reason to continue and perform the iterations.

6.1.2. Tabu select iterations

The goal of TSel iterations is to move from one configuration to the other aiming at improving the objective value. Algorithm 1 describes the iteration process. The decision variables γ^* and $\boldsymbol{A}^*$ with the associated profit Q^* denote the best answer that Tabu Select has reached at any point in time.

Initially, the best answer reached is the answer obtained from the initial solution. Therefore $\gamma^* = \gamma_i$, $\boldsymbol{A}^* = \boldsymbol{A}_i$ and Q^* is the profit associated with γ_i and $\boldsymbol{A}_i$ and is calculated as indicated in Eq. (3).

For each configuration vector γ_c, generateMutations () generates the associated list of possible neighboring solutions. Each neighboring (candidate) solution γ_{can} differs by one bit from $\gamma_c \cdot \gamma_{can}$ either adds an additional project to be completed or deselects a previously selected project. In the latter case, freeResources () frees the allocated resources on the earlier selected project by removing them from $\boldsymbol{A}$ and adding them to $\boldsymbol{F}$.

In case γ_{can} adds an additional project, GAlloc () either succeeds in allocating resources for the additional project and returns true, or fails and returns false. The list of feasible neighboring solutions are stored in *candidatesMap*.

If the best neighboring solution has a profit above Q^*, then it becomes the next configuration and $\{Q^*,\gamma^*,\boldsymbol{A}^*\}$ are updated accordingly. Otherwise, the neighboring solution that is not contained in the *tabuList* and that has the highest profit Q is then the next configuration. If no such neighboring solution is found, then one of the Tabu neighbors is chosen at random to be the next configuration by the choice () function.

Any chosen configuration that does not belong to *tabuList* is added to this list. Iterations continue until one of the stopping criteria is met. At the beginning of each iteration, $\boldsymbol{F}$ is reset to represent the non-allocated resources in $\boldsymbol{A}$. The function resetF () resets $\boldsymbol{F}$ according to Eq. (15).

Algorithm 1. (γ^*,A^*,Q^*) TSI (γ_i,A_i)// returns (γ^*,A^*,Q^*)

```
1: $\gamma^* = \gamma_i$, $A^* = A_i$, $Q^* = Q(\gamma_i,A_i)$, $\gamma_c = \gamma_i$, $A_c = A_i$   //initialization
2: $iterNum = 1$, $tabuConfNum = 0$, $tabuList = \{\}$
3: StartTimer (runTime)
4:
5: while ($iterNum < maxIterNum$ && $tabuConfNum < maxTabuConfNum$) do
6:     candidatesMap.clear (), tabuCandidatesMap.clear ()
7:     $list\gamma_{can}$ = generateMutations ($\gamma_c$) // generate vectors one-bit different from $\gamma_c$
8:     for ($\gamma_{can} \in list\gamma_{can}$) do
9:         if ($runTime > maxRunTime$) then
10:            return $(\gamma^*,A^*,Q^*)$
11:        end if
12:        $A = A_c$
13:        $F$ = resetF $(n,A)$
14:        $p$ = indexOfNonZeroElement $(\gamma_{can} - \gamma_c)$
15:        if $((\gamma_{can} - \gamma_c)[p] == -1)$ then     // if the mutation removes a project
16:            freeResources $(p)$     // move resources of $p$ from $A$ to $F$
17:        else if ($\neq$g GAlloc $(p,A)$) then     // GAlloc succeeds and mutation adds a project
18:            continue
19:        end if
20:
21:        $Q_{can} = Q(\gamma_{can},A)$
22:        candidatesMap.insert $(\{Q_{can},\gamma_{can},A\})$
23:    end for
24:
25:    while (candidatesMap.isEmpty () == false) do
26:        $\{Q_{best},\gamma_{best},A_{best}\} = \max_Q candidatesMap$
27:        if $(Q_{best} > Q^*)$ then
28:            $\{Q^*,\gamma^*,A^*\} = \{Q_{best},\gamma_{best},A_{best}\}$
29:            break
```

(continued on next page)

```
30:         end if
31:         if ($\gamma_{best} \in tabuList$) then
32:             $candidatesMap$.remove ($\{Q_{best}, \gamma_{best}, A_{best}\}$)
33:             $tabuCandidatesMap$.add ($\{\gamma_{best}, A_{best}\}$)
34:         end if
35:     end while
36:
37:     if ($candidatesMap$.isEmpty ()) then
38:         $\{\gamma_c, A_c\}$ = choice ($tabuCandidatesMap$)
39:         $tabuConfNum = tabuConfNum + 1$
40:     else
41:         $\gamma_c = \gamma_{best}$, $A_c = A_{best}$
42:         $tabuConfNum = 0$
43:     end if
44:     $iterNum = iterNum + 1$
45: end while
46:
47: return ($\gamma^*, A^*, Q^*$)
```

6.2. Greedily Allocate (GAlloc)

Greedily Allocate is called on Line 17 of Algorithm 1 to decide whether a project p should be added to the list of chosen projects. GAlloc starts by populating the vector of candidate quarters $\boldsymbol{q}$ that includes the quarters during which p can be worked at. A quarter earlier than $\boldsymbol{d}[p]$ is added to $\boldsymbol{q}$ if the combined strength of the unallocated resources in it is at least equal to the strength that p requires for all J skills and if the unallocated resources satisfy the critical skills requirements for p. If the size of $\boldsymbol{q}$ is at least $\delta[p]$, then p is feasible. The complexity of the feasibility check of project p is therefore $O(\delta[p] \times J \times K)$.

GAlloc calculates for each quarter t from $\boldsymbol{q}$ the money value $MVal[t]$ of the resources already allocated during t. GAlloc selects $\delta[p]$ quarters from $\boldsymbol{q}$ starting with the quarters with the highest $MVal$. Ties between two quarters are broken by choosing the one closer to the deadline $\boldsymbol{d}[p]$ first. By selecting quarters with highest $MVal$, GAlloc tries to maximize a quarter usage by allocating as much resources as possible during it before switching to less occupied quarters.

This process has implications on the cost of development since allocating resources for an additional project during a previously non-busy quarter (with $MVal = 0$) adds to the total cost of human resources and makes the company pay salaries for all its developers during that quarter. Breaking ties by choosing quarters closer to the deadline tries to free early quarters for other potential projects with closer deadlines.

For the $\delta[p]$ chosen quarters for working on p, GAlloc allocates all the unallocated resources on p. Then it removes one resource at a time without breaking feasibility conditions starting from the talent class that has the highest salary and moving towards the least costly ones. The allocation process has therefore a polynomial complexity of $O(\delta[p] \times K \times \max(\boldsymbol{n}))$. Therefore, as a whole Greedily Allocate has a polynomial complexity.

7. The Lebanese case study

We evaluate our skill-based framework and apply it on the Lebanese academia and software industry. We perform two surveys, one for the academia and one for the software companies, following the guidelines presented in Section 3 to populate the parameters of PMax.

7.1. Academia survey

In Lebanon, there is one public university and 31 licensed private universities as per the Lebanese Ministry of Education (2012). We conducted surveys on nine universities that have the highest enrollment rates in Lebanon.[2] Each academic institution offers one or more software related programs. We note the distinction between bachelors of engineering (Computer Engineering, Electrical Engineering) and bachelors of science (Computer Science, Information Technology). We study the academic programs offered as indicated in Section 3 and we generate a list of software skills and a set of talent classes.

7.2. Industry survey

According to Dal Hitti, the general manager of the *Association of the Lebanese Software Industry* (ALSI), there are 103 software companies in Lebanon. Half of them are of very small size comprising of 5 employees on average. A quarter of them are composed of 25–30 employees, 22% have an average size of 80 employees and the remaining 3% are companies above 100 employees each (Hitti, 2011).

We contacted 35 software companies and 11 completed the interview based industry survey.[3] For each company, we parametrized the vectors $\boldsymbol{n}$ and $\boldsymbol{c}$. Then each CIO provided us with detailed information about a project that the company completed. This information was used to populate the vectors $\boldsymbol{d}$, $\boldsymbol{\delta}$ and $\boldsymbol{\pi}$ and the matrix $\boldsymbol{W}$. We ended up with 11 projects. Survey data is available online[1].

We generalized and extended the set of projects collected from the surveys by aggregating the required skills and the revenue and formed a total of 35 projects that represent the Lebanese industry. In particular, for each project p, we generated a bounding range of two projects p_1 and p_2 where p_1 required less skills and returned higher revenue and p_2 required more skills and returned less revenue. We also generated two virtual projects to model the projects that the Lebanese industry might attract. One constituted an upper bound on revenue with associated maximum skills requirements, and the other reflected minimum overhead with freelance web development skills working for one yearly quarter. The revenue associated with the latter is set to a small value that reflects market conditions. From the academic survey we were able to identify a comprehensive and granular skill set and a set of sample talent classes that are dispatched into the technical workforce from the academia in Lebanon. The skill set contains 197 skills which cover software and software related skills taught in Lebanese academia. We rank these skills based on need and availability and we report the most important ones in the online Appendix 1.

We classified these skills into 39 categories as displayed in Table 2. They range from core software development to math and electrical engineering. The sample set of talent classes contains 12 classes, 6 for engineering majors, and 6 for science degrees. The talent classes range from taking the bare minimum set of required core courses to taking all the software elective courses. The 11 surveyed companies employ 310 employees, from which 241 are technical employees working in software development.

7.3. Results

For the 11 surveyed companies, we run PMax to determine the optimal project selection vector γ and the optimal resource allocation matrix $\boldsymbol{A}$. PMax provides us with the best performance of the companies with their maximal profit. Table 3 shows the cost and the selected projects for each one of the 11 companies after

[2] The universities are in no specific order: the Lebanese University (LU), the American University of Beirut (AUB), the Saint Joseph University (USJ), the Lebanese American University (LAU), the Beirut Arab University (BAU), the Notre Dame University (NDU), the Hariri Canadian University (HCU), the Lebanese International University (LIU), and the Haigazian University (HU).

[3] It is worth noting that there are 103 software development companies in Lebanon and 11 of them agreed to complete the survey, i.e. 10.6% of the total number.

Table 2
Skill categories.

Intro. to programming	Numerical analysis techniques
Data Structures & Algorithms	Communications Systems
Design & Analysis of Algorithms	Computer Networks
Operation Systems	Computer Organization
Database Systems	Computer Architecture
Programming Language Design & Implementation	Signals & Systems
Software Engineering	Digital System Design
Advanced Algorithms & Data Structures	Analog & Digital Circuits
Theory of Computation	Electronics
Web Programming & Design	Electric Machines & Power Fundamentals
Human Computer Interaction	Electromagnetics
Computer Graphics	Electric Circuits
Computer-aided Geometric Design	General Engineering Tools
Artificial Intelligence	Basic Computer Software
Multimedia Programming	Differential Equations
Network Programming	Discrete Structures
Compiler Construction	Linear Algebra
Multimedia Design	Calculus & Analytic Geometry
Foundations of Digital Media	Introduction to Probability & Random Variables
Digital Imaging	

running PMax on them in the “Cost (PMax)” and “Selected projects” columns respectively.

Column “Difference in cost” estimates the savings resulting from PMAX had the company decided to operate on the selected projects without the advice of PMAX. We can see that even when the optimal portfolio for several companies is the same (as is the case for companies 1, 4 and 5 for instance), the associated cost might be different. This is a direct implication of the difference in human resource allocation between the different companies which directly affects the development costs.

7.4. Discussion

We conducted a second industrial survey where we summarized the results of PMAX for the eleven companies. The first question in the survey queried the representatives on the feasibility of the distribution of skills across selected projects as suggested by PMAX.

The second question queried about the feasibility of the schedule. The third question asked for a ranking of synergistic factors that support or obstruct the results but that are not included in the parameters of PMAX. Finally, the fourth and fifth question asked about technical and know how skills existing in industry that are not covered or cannot be categorized under technical and know how skills acquired at academy.

We passed the survey to representatives from the eleven companies. Two of them replied. We also passed the survey with additional descriptions of the companies to eight software development professionals working in the industry at senior and managerial positions and four of them replied.

All respondents agreed that the results of PMAX are feasible and insightful in terms of allocating talent classes based on the distribution of skills required by the project. They also agreed that the schedules proposed by PMAX were feasible. Informally they expressed their interest in using PMAX once it is developed into an easy to use tool.

They listed and ranked several factors that they think are essential to the success of the strategy and most of them were first concerned with the availability of such projects in the market currently open for the Lebanese companies. Their second important concern was the feasibility of investment in developing the projects. A minority was concerned with the logistics to run several big projects at once in a Lebanese company.

Several of them listed “shipping software” as a technical skill that is not covered in academic training, next in importance they listed “knowledge of specialized development platforms”. They also listed “testing and quality control”, “handling error and exceptional cases” and “handling engineering trade-offs” as skills that are not covered in academic training. In retrospect, we found out that those skills were listed as outcomes and objectives in several advanced technical courses we have surveyed in our academic survey in one form or another.

On know-how and non-technical skills the majority complained about team work. The other complaints were related to risk assessment and experience in client interface. Again, these skills are listed in one form or another in advanced courses we have surveyed in our academic survey. We understand the complaints in the validation surveys as complaints on the strength level in these skills which does not affect the validity of our model. The validation survey and the tables are available online[1].

We also interviewed a selected group of CIOs to validate the results and reflect on the proposed analysis framework. The CIOs thought that they would make use of the model described in this work if it is transformed into an easy to use tool and if it is extended to provide dynamic measures of profit estimations. Concerning the list of 197 software related skills, CIOs thought that additional software skills might come from sources different from academia and thus they think they must be incorporated in the analysis. Additionally, they spelled the importance of non-technical skills in the evaluation and classification of employees. One CIO gave us access to the evaluation sheet of employees at his company and we found out that 13 skills are common between his evaluation form and our skill list. In addition, he uses other meta skills that map into sets of skills in our case. The evaluation sheet contained further non-technical skills that we do not consider in our analysis.

Table 3
Costs of development.

Company	CostPMax	Selected projects	Difference in cost
Company 1	215,440	1,35	261,868
Company 2	1,158,075	1,2,3,31,32,35	249,984
Company 3	752,400	1,28,31,35	1,273,938
Company 4	213,116	31,35	264,192
Company 5	353,464	31,35	123,844
Company 6	604,104	7,35	227,496
Company 7	510,524	31,35	192,333
Company 8	1,575,512	1,7,28,31,32,35	107,789
Company 9	901,948	1,35	282,052
Company 10	2,265,528	1,2,7,28,29,35	821,670
Company 11	1,561,340	1,2,35	869,243

Table 4
Performance comparison between PMax and TSGA.

	RunTimePMax (seconds)	RunTime (TSGA) (seconds)	*time*(*TSGA*)/*timePMax*	*profit*(*TSGA*)/*profitPMax*
Min	15	0	0	81%
Max	5817	68	81%	100% In 64 out of 99
Average	532	15	6.7%	98%

Table 5
Medium and large problems.

Problem size	RunTimePMax	RunTime (TSGA) (second)	*time(TSGA)/timePMax*	*profit(TSGA)/profitPMax*
$P = 70$, $N = 150$	3553 second	51	1.4%	91.2%
$P = 70$, $N = 180$	563 second	76	13.5%	93.4%
$P = 100$, $N = 250$	36,387 second	255	0.7%	94%
$P = 100$, $N = 280$	mem-out	87	NA	NA
$P = 130$, $N = 250$	mem-out	135	NA	NA
$P = 130$, $N = 310$	mem-out	173	NA	NA
$P = 150$, $N = 280$	mem-out	312	NA	NA
$P = 150$, $N = 460$	mem-out	162	NA	NA
$P = 300$, $N = 540$	mem-out	482	NA	NA
$P = 300$, $N = 660$	mem-out	539	NA	NA

Table 6
TSGA initial solution quality.

	RunTime (Initial Solution) (second)	*profit(Initial Solution)/profitPMax* (%)
min	0	32
max	0.16	100
average	0.02	78

8. Evaluation of TSGA

We ran PMax and TSGA, on a 64-bit machine with 8 gigabyte of memory and a quad-core CPU clocked at 2.8 gigahertz. The problems from the Lebanese industry were relatively small. The number of talent classes K was limited to 12, the number of project P was set to 35 and the number of developers in a single company ranged between 6 and 60.

Given these sizes, PMax, when run on CPLEX, required between 15 and 533 second to determine $\boldsymbol{A}$ and γ with an average time of 75 second. Some problems exhausted 5 gigabyte of RAM to run into completion. The problems are available online1.

After carefully tuning the parameters of TSGA, we set $maxIterNum = 1000 \times size(feasibleProj)$, $maxTabuConfNum = size(feasibleProj)$ and $maxRunTime = 600 seconds$. Additionally, We do not put any constraint on the size of the tabuList knowing that the memory needs of TSGA are very modest and almost limited to this list. We then run the same 11 problems using TSGA. We get an optimal solution for 10 of the problems and a 99.75% of the optimal profit on the last one. The run time of TSGA is on average 90% smaller than the CPLEX run time.

To compare the actual performance of TSGA versus PMax, we generated 100 problems with relatively the same size of the Lebanese industry. We kept the same talent classes derived from the Lebanese cases and generated 35 random projects ($P = 35$). We also generated for each problem a random number of talents that we constrained to be less than 100. Then we distributed the talents over the 12 talent classes.

We ran both PMax and TSGA on the 100 generated problems. PMax ran into completion for 99 out of the 100 problems. [12] For one of the problems, and after a run time of 26,800 second, PMax exhausted all the RAM before exiting. For the other 99 problems, Table 4 summarizes the performances of PMax and TSGA.

TSGA performed almost 15 times faster than PMax and generated on average a profit valued at 98% of the optimal profit. Additionally, for the 99 comparative problems, TSGA reached the exact optimal solution 64 times and the worst profit that it found was 19% below the optimal solution. We can see that PMax sometimes needs a considerable amount of time (96 minute) to run into completion while the worst TSGA runtime was 68 second. The results show that TSGA is performing well for small size problems when compared to PMax.

For industries larger than the Lebanese industry, benchmarking TSGA against PMax is not possible because PMax will exhaust memory and runtime resources before returning an optimal solution.

8.1. Medium and large size problems

We generate larger problems that mimic industries larger than the Lebanese industry to check how the performance of TSGA varies with problem sizes. We present problems with increments of the number of projects available P and the total number of talents N, then we run both TSGA and PMax on the generated problems.

Table 5 shows that when PMax returns a solution, TSGA maintains its performance presented in Section 8. CPLEX exhausts memory resources without generating any feasible solution for problems where $P \geqslant 100$ and $N \geqslant 280$. TSGA runtime is still very reasonable and exits on functional stop conditions other than the 10 minute *maxRunTime* allowed for it. The largest two problems we simulated maintained 2551 and 2235 γ configurations in the Tabu list and were solved in 2651 and 2235 iterations that took 28.32 and 32.65 second respectively. The detailed results are found online.

8.2. Quality of TSGA initial solution

When Tabu search is used, it is advisable that the initial solution is quickly computable and of good quality. A better initial solution implies a better overall problem solution in general (Hasle, Lie, Quak, & for industriell og teknisk forskning ved Norges tekniske hogskole, 2007). For TSGA, the initial solution is very quickly generated. After running the 100 benchmarking simulations, the average time for generating the initial solution is 0.02 second. At the same time, on average, the objective value of the profit obtained from the initial solution is 78% of the optimal objective found by CPLEX as shown in Table 6. This high value indicates that the heuristic for generating the initial solution and that includes calling Greedily Allocate is performing well.

9. Conclusion

In this paper, we proposed a skill-based representation of a country over four analysis domains: universities, companies, talents and projects. We showed how we can determine the software skills that a country produces by analyzing the academic programs of its different universities. We then presented a formal methodology for ranking the strength of graduating students in the different software skills.

We also introduced PMax, a project selection and a resource allocation integer linear program that determines the optimal portfolio of projects for a certain company and allocates resources optimally among them.

We studied the complexity of PMax and we found them to be NP-complete. For this reason, we introduced a meta-heuristic based algorithm (TSGA) to outcome the difficulties of using PMax.

We showed how our methodology can be applied by performing two surveys on the Lebanese software industry and academia. We generated a list comprising of 197 software-related skills taught in Lebanon and we compiled the most and the least needed ones in the Lebanese industry. We collected real data about Lebanese companies and software projects developed internally and we ran PMax. We then evaluated the performance of TSGA comparing it to PMax and we concluded that TSGA is running 15 times faster than PMax with 98% accuracy. We showed that as opposed to PMax, TSGA can scale easily when the size of the problems increases.

In practice, our methodology relies on expert opinions to formalize technical software project requirements into skill strength requirements. It also relies on an up-to-date inventory of human resources represented in terms of strengths in skills. This is common practice in software companies where every employee is required to update his/her Curricular Vitae according to a preset template. It will be beneficial if in future work, an established model-based estimation method (for software development cost) is used to validate the expert opinions. In future work, we will consider a dynamic model that allows for skill growth based on resource allocation. If a resource x is allocated on a project p requiring skill s for a period t. At the end of t, the strength of x in s should increase and x might now belong to a different talent class with a higher cost. In general, we hypothesize that an increase in skill strength will increase the cost of the labor force, but at a greater benefit of allowing the company to bid for more challenging and rewarding projects.

Acknowledgments

The authors would like to acknowledge the financial support of the Lebanese national council for scientific research (LNCSR) and Dar Al Handasah Endowment Fund for Research in Engineering. The authors would also like to thank the CIOs of the Lebanese companies that completed the surveys in particular Mr. Dal Hette and Mr. Sami Slim.

Appendix A. Supplementary data

Supplementary data associated with this article can be found, in the online version, at http://dx.doi.org/10.1016/j.ejor.2013.09.035.

References

Acuna, S. T., & Juristo (2004). Assigning people to roles in software projects. *Software Practice and Experience, 34*, 675–696.

Acuna, S. T., Juristo, N., & Moreno, A. M. (2006). Emphasizing human capabilities in software development. *IEEE Software, 23*, 94–101.

Andre, M., Baldoquin, M. G., & Acuna, S. T. (2011). Formal model for assigning human resources to teams in software projects. *Information and Software Technology, 53*, 259–275.

Archer, N., & Ghasemzadeh, F. (1999). An integrated framework for project portfolio selection. *International Journal of Project Management, 17*, 207–216.

Barreto, A., de O. Barros, M., & Werner, C. M. (2008). Staffing a software project: A constraint satisfaction and optimization-based approach. *Computers & Operations Research, 35*, 3073–3089 (Part Special Issue: Search-based Software Engineering).

Boehm, B., Abts, C., & Chulani, S. (2000). Software development cost estimation approaches a survey. Technical report. Annals of Software Engineering.

Brucker, P., Drexl, A., Mohring, R., Neumann, K., & Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research, 112*, 3–41.

Colomo-Palacios, R., Tovar-Caro, E., Crespo, A. G., & Gomez-Berbis, J. M. (2010). Identifying technical competencies of it professionals: The case of software engineers. *International Journal of Human Capital and Information Technology Professionals, 1*, 31–43.

Garey, M. R., & Johnson, D. S. (1990). *Computers and intractability: A guide to the theory of NP-completeness*. New York, NY, USA: W.H. Freeman & Co..

Glover, F. (1989). Tabu search, Part I. *ORSA Journal on Computing, 1*, 190–206.

Glover, F. (1990). Tabu search, Part II. *ORSA Journal on Computing, 2*, 4–32.

Gutjahr, W. J., Katzensteiner, S., Reiter, P., Stummer, C., & Denk, M. (2010). Multi-objective decision analysis for competence-oriented project portfolio selection. *European Journal of Operational Research, 205*, 670–679.

Hasle, G., Lie, K., & Quak, E., & for industriell og teknisk forskning ved Norges tekniske hogskole, S. (2007). Geometric modelling, numerical simulation, and optimization: Applied mathematics at Sintef. Springer Verlag.

Herroelen, W., & Leus, R. (2005). Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research, 165*, 289–306.

Herroelen, W., Reyck, B. D., & Demeulemeester, E. (1998). Resource-constrained project scheduling: A survey of recent developments. *Computers & Operations Research, 25*, 279–302.

Hitti, D. (2011). Personal communication with the manager of the association of the lebanese software industry (ALSI).

Jorgensen, M. (2004). A review of studies on expert estimation of software development effort. *Journal of Systems and Software, 70*, 37–60.

Jorgensen, M. (2005). Practical guidelines for expert-judgment-based software effort estimation. *IEEE Software, 22*, 57–63.

Jorgensen, M., Boehm, B., & Rifkin, S. (2009). Software development effort estimation: Formal models or expert judgment? *IEEE Software, 26*, 14–19.

Jorgensen, M., & Shepperd, M. J. (2007). A systematic review of software development cost estimation studies. *IEEE Transactions on Software Engineering, 33*, 33–53.

Kan, S. H. (1994). *Metrics and models in software quality engineering* (1st ed.). Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc..

Kellerer, H., Pferschy, U., & Pisinger, D. (2004). *Knapsack problems*. Springer.

Kitchenham, B., Budgen, D., Brereton, P., & Woodall, P. (2005). An investigation of software engineering curricula. *Journal of Systems and Software, 74*, 325–335.

Lebanese Ministry of Education, 2012. Universities in lebanon. <http://www.higher-edu.gov.lb/english/default.htm>.

Lethbridge, T. C. (2000). What knowledge is important to a software professional? *Computer, 33*, 44–50.

Meade, L. (2002). R&D project selection using the analytic network process. *IEEE Transactions on Engineering Management, 49*, 59–66.

Ngo-The, A., & Ruhe, G. (2009). Optimized resource allocation for software release planning. *IEEE Transactions on Software Engineering, 35*, 109–123.

Oezdamar, L., & Ulusoy, G. (1995). A survey on the resource-constrained project scheduling problem. *IIE Transactions, 27*, 574–586.

Otero, L. D., Centeno, G., Ruiz-Torres, A. J., & Otero, C. E. (2009). A systematic approach for resource allocation in software projects. *Computers & Industrial Engineering, 56*, 1333–1339.

Padberg, F. (2001). Scheduling software projects to minimize the development time and cost with a given staff. *Asia-Pacific Software Engineering Conference, 0*, 187–194.

Plekhanova, V. (1999). Capability and compatibility measurement in software process improvement. In *2nd European software measurement conference*. Amsterdam, The Netherlands.

Trigo, A., Varajao, J., Soto-Acosta, P., Barroso, J., Molina-Castillo, F. J., & Gonzalvez-Gallego, N. (2010). It professionals: An iberian snapshot. *IJHCITP, 1*, 61–75.

Xiao, J., Wang, Q., Li, M., Yang, Q., Xie, L., & Liu, D. (2009). Value-based multiple software projects scheduling with genetic algorithm. In Q. Wang, V. Garousi, R. Madachy, & D. Pfahl (Eds.). *Trustworthy software development processes of lecture notes in computer science* (Vol. 5543, pp. 50–62). Berlin, Heidelberg: Springer.

Yoshimuraa, M., Fujimia, Y., Izuia, K., & Nishiwakia, S. (2006). Decision-making support system for human resource allocation in product development projects. *International Journal of Production Research*, 44.