2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

Heuristic optimization for software project management with impacts of team efficiency

Nanlin Jin and Xin Yao

Abstract— **Most of the studies on project scheduling problems assume that every assigned participant or every team of the same number of participants, completes tasks with an equal efficiency, but this is usually not the case for real world problems. This paper presents a more realistic and complex model with extra consideration on team efficiency which are quantitatively measured on employee-task assignment. This study demonstrates the impacts of team efficiency in a well-studied software project management problem. Moreover, this study illustrates how a heuristic optimization method, population-based incremental learning, copes with such added complexity. The experimental results show that the resulting near optimal solutions not only satisfy constraints, but also reflect the impacts of team efficiency. The findings will hopefully motivate future studies on comprehensive understandings of the quality and efficiency of team work.**

I. Introduction

THE Project scheduling problem has been extensively studied in literature. Typically such a problem is characterized by resource constraints, precedence constraints, and objectives to be optimized. A project scheduling problem for software projects entails more than one resource constraints, as well as multi-tasking within one day [1] [2], called software project management problem (SPMP). Tasks can be completed if assigned staff have the required IT skills, while each employee has a particular set of IT skills. A good solution assigns employees to tasks, considering constraints on skills capability and requirements, salaries, and task dependencies. A study on the run-time analysis for the SPMP has interpreted the variation of performance on design choices for the evolutionary algorithms (EA) [2].

The SPMP problem assumes that if two employees have the same skills and the same dedication (the percentage of their workload per day), they will finish a task within the same amount of time. Therefore, the only difference of appointing one employee from another depends on the variations of their salaries. This is far from the observations in reality. In practice, employees with the same skills and spending the same dedication to a task can complete it in different durations.

Moreover, the SPMP problem does not differentiate the efficiency caused by different combinations of employees. In fact, the teamwork efficiency is important: various combinations of employees having the same required skill set can complete an identical task in a largely various time scale, especially for complicated tasks [3].

This paper aims to address the weaknesses above by introducing a simple yet illustrative extension to the SPMP problem. The new problem reflects the impacts of team efficiency in project scheduling. Here individual efficiency is treated as a special case of team efficiency.

A. Literature review - team work

It is common that much of the work in a project is completed through a group of participants. A rich body of literature has shown the importance of teamwork. The growing awareness of "good team" and "bad team" raise new questions on how to measure team efficiency [4].

The variations of team efficiency can be the consequences of a wide variety of factors, including experiences, working relationship, team work processes, coordination, conflict management, motivations, team cognitive, and even emotional intelligence.

Team performance has been measured in role-playing games and been predicted based on the previous performance patterns [3]. The individual performances are compared with the performances of a variety of combinations of players which form teams. The performance is measured mainly by the game point gained as well as play time. Researchers have found that teams do not always perform better than individual players, due to a range of factors: such as team size, capability (level) of player(s), level diversity in a team, and task difficulty.

B. Literature review - problem solving methods

The SPMP problem is hard to solve [2]. Therefore heuristic methods are among popular choices for practical problem solving, such as genetic algorithms (GAs) [1], GA with a pre-decision algorithm [5], differential evolution algorithms [6], ant colony optimization algorithms [7][8], artificial bee colony [9], meta heuristic algorithms of simulated annealing-genetic algorithm-Tabu search [10], frog-leaping algorithm [11], branch-and-bound methods [12], hybrid mixed-integer linear programming and constraint programming algorithms [13] [14]. Pareto optimality is used as the multi-objective optimization method for reducing the probability of overtime during software projects [15].

Chan, et al. studied a more realistic software project scheduling problem that is similar to SPMP [16]. They take other factors into consideration, such as employees' experience, speed, efficiencies, and tasks' soft deadline, hard deadline, penalty max head. They also solve their problem

Nanlin Jin is with the Department of Computer Science and Digital Technologies, Northumbria University, Newcastle upon Tyne, NE1 8ST, U.K. (email: nanlin.jin@northumbria.ac.uk).

Xin Yao is with the School of Computer Science, University of Birmingham, Birmingham B15 2TT, U.K. (email: x.yao@cs.bham.ac.uk).

This work was partly sponsored by two EPSRC grants (EP/D052785/1 and EP/J017515/1).

by GA. Wena and Lin, solve multistage human resource allocation for software development by multi-objective GA [17]. Similar to [1], these works also provide the "best fitness so-far" solution(s) for a problem instance.

Population based incremental learning (PBIL) is a heuristic optimization algorithm. PBIL is a special case of Estimation of distribution algorithms (EDA) [18]. Its major difference from a standard genetic algorithm is that probability vectors over possible values to solutions, not candidate solutions, are evolved. Candidate solutions are generated by sampling the evolved probability vectors.

C. Overview of this work

This paper will extend the SPMP problem taking team efficiency into account. To the best of our knowledge, this is the first work that explicitly examines the impact of team efficiency on project scheduling. This work specifies quantitative measures, not qualitative measures on the impact of team efficiency. The new problem in named as T-SPMP. Its instances specify that some combinations of employees increase or decrease the time required for task completion. Secondly, due to the N-P hard nature of the T-SPMP problem, a heuristic optimization method is employed. Finally, the resulting solutions to instances of the SPMP problem and the solutions to instances of the T-SPMP problem will be compared and analyzed.

The rest of this paper is organized as follows: Section II briefs a well-studied software project management problem (SPMP) and its new extension which models team efficiency on tasks (T-SPMP). Section III details the population-based incremental learning (PBIL) algorithm, and the experimental setup. The experimental results are presented in Section IV to show the impact of team work on solutions of task assignment. Finally, Section V concludes the paper.

II. Software project management problem with team efficiency impacts - T-SPMP

A. Notations and solution representation

The software project management problem with team efficiency impacts (T-SPMP) specifies employees and tasks related to a software project. The best solutions should complete the project with the minimal possible cost and time. An employee is identified by his/her employee ID: e_i. An employee is specified by their skill sets, S_{e_i} and salary, P_{e_i}. The set of total employees is $\{e_1, e_2, \ldots, e_m\}$. A task is identified by its ID, k_j. A task is specified by its required skill set S_{k_i}, and its workload (effort), L_{k_j}. The workload is the amount of time that an employee who has all of the skills in S_{k_i} needs to independently finish the task. If two employees work together on this task, the time will be halved if they have the skills needed. The set of tasks is $\{k_1, k_2, \ldots, k_n\}$.

An assignment of all tasks to employees constitutes a candidate solution $x = \{x_{1,1}, x_{1,2}, \ldots, x_{m,n}\}$. Table I shows the representation of a solution. One of its tuple x_{ij} indicates whether employee e_i works on task k_j. x_{ij} is his/her *dedication* on k_j: the time s/he spends in percentage of normal working hours per day, where $x_{ij} \in [0, 1]$. One task can be done by a group of employees working together. An employee can involve in more than one task on one day.

TABLE I
Representation of solution

	k_1	k_j	k_n
e_1	$x_{1,1}$	$x_{1,j}$	$x_{1,n}$
e_i	$x_{i,1}$	$x_{i,j}$	$x_{i,n}$
e_m	$x_{m,1}$	$x_{m,j}$	$x_{m,n}$

B. Project related attributes

TaskDuration is defined as, given a solution x, the amount of time that task k_j is to be finished:

$$t_{k_j} = \frac{L_{k_j}}{\sum_{i=1}^{m} x_{ij}}. \quad (1)$$

This is calculated by dividing the task workload L_{k_j} with the total dedications of employees assigned to this task.

A project often consists of a number of tasks. *ProjectDuration* is the length of time from the start of the first task(s) to the end of the last task(s). Fig.2 shows how to calculate the project duration in pseudo code having a candidate solution and the dependencies among tasks as inputs.

ProjectCost is defined as the total cost in term of the salaries paid for the work done by the employees who are assigned to this project:

$$projectCost = \sum_{i=1}^{m} \sum_{j=1}^{n} P_{e_i} \times x_{ij} \times t_{k_j}. \quad (2)$$

Overwork: although one employee's dedication to one task is constrained by $x_{ij} \in [0, 1]$, there are chances that one employee is assigned to more than one task for a day such that the total dedication of all tasks assigned to s/he on that day is over 1, i.e. s/he overworks: working more than 8 hours on that day. If there is an overwork, the solution is treated as an infeasible solution. Fig. 2 defines how to calculate the overwork in pseudo code with inputs of a candidate solution and the dependencies among tasks. $Overwork_{now}$ in Fig. 2 denotes the current value in calculating $Overwork$.

Task precedence graph (TPG) determines the dependence among tasks. Precedence constraints are studied in constrained scheduling and optimization [19]. This paper only studies the type of "Finish to Start" dependence. Fig. 1 provides an example with 10 tasks and 21 task dependencies. A dependence $k_j \rightarrow k_{j'}$ is interpreted as task $k_{j'}$ cannot be started before task k_j is finished. Table II shows a problem instance with 21 task dependencies. These dependencies between tasks must be satisfied; otherwise the project cannot be completed.

C. Constraints

There are a few important constraints in the T-SPMP problem.

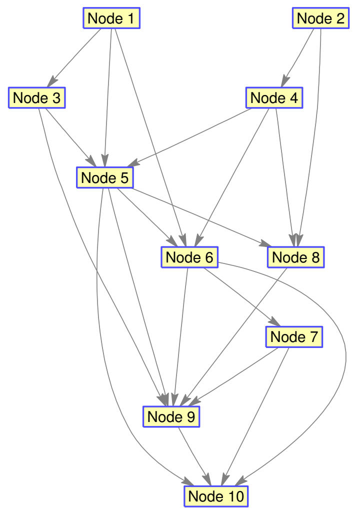

Fig. 1. Task precedence graph: each node represents a task. $Node_j \rightarrow Node_{j'}$ is interpreted as task $k_{j'}$ cannot be started before task k_j is finished.

```
1:  while $TPG \neq \phi$ do
2:    Let V:= unfinished tasks not depending on any other task.
3:    if V = $\phi$ then
4:      return "Unsolvable" and STOP
5:    end if
6:    for all task $k_j \in V$ do
7:      for all employee $e_i$ do
8:        Let $dedication_{ij} = x_{ij}$
9:      end for
10:     if $\sum dedication_i > 1$ then
11:       $overwork_{now} := overwork_{now} + dedication_i - 1$
12:     end if
13:     calculate total dedication $d_j := \sum_{i=1}^{m} dedication_{ij}$
14:     if $d_j := 0$ then
15:       return $(\infty, \infty)$ and STOP
16:     end if
17:   end for
18:   Let $t := MIN_j(L_j/d_j)$
19:   $projectDuration = projectDuration + t$
20:   for all $k_j \in V$ do
21:     Let $L_j := L_j - t \times d_j$
22:     if $L_j := 0$ then
23:       Mark $k_j$ as finished and remove it and its edges from TPG
24:     end if
25:     $overwork = overwork + overwork_{now} \times t$
26:   end for
27: end while
28: Output projectDuration, overwork and stop
```

Fig. 2. The pseudo code for calculating project duration and overwork, given the inputs of TGP and a candidate solution x.

TABLE II
INSTANCE EXAMPLE AND EXPLANATION

Instance values	**Explanation**

Summary:
skill.number=10
task.number=10
employee.number=5
graph.arc.number=21
teamefficiency.number=1
	The total numbers of skills, tasks, employees, dependencies, and team efficiency of an instance.

Attributes and their values about tasks:
task.1.skill.number=2
task.1.skill.1=3
task.1.skill.2=5
task.1.load=7.0
. . .
	The number of skills required for a task and its workload. Task 1 here requires two skills: skills 3 and 5 in the skill set.
	Task 1's workload is 7.

Attributes and their values about employees:
employee.1.skill.number=4
employee.1.skill.1=1
employee.1.skill.2=8
employee.1.skill.3=9
employee.1.skill.4=3
employee.4.salary=10448.13
. . .
	The number of skills that that an employee has and his/her salaries. This example shows that employee 1 has four different skills, numbered 1, 8, 9 and 3 in the skill set.

Attributes and their values about dependencies between tasks:
graph.arc.1=1 3
graph.arc.2=1 2
graph.arc.3=2 4
. . .
	The dependencies among tasks which are numbered as 1, 2, etc:
	"graph.arc.1=1 3" means task 3 cannot start before Task 1 finishes.

e1e2 duration ↓ team efficiency factor $\delta = 0.5$	$e1$ and $e2$ are assigned to a task, the task duration decreases 50%

(1) Every task should be assigned to at least one employee. Here "undt" is used to calculate the number of times that a candidate solution violates this constraint:

$$undt = |\{k_j | \sum_{i=1}^{m} x_{ij} = 0, 1 \leq j \leq n\}|. \quad (3)$$

(2) The union of the skills of the employees who are assigned to a task should have all skills that this task requires. Here "reqsk" denotes the number of the skills unavailable from employees assigned to tasks:

$$reqsk = \sum_{j=1}^{n} |S_{k_j} - \bigcup_{\{i|x_{ij}>0\}} S_{e_i}|. \quad (4)$$

(3) Overwork is not allowed. The maximal working load an employee can take should not be larger than 1. The calculation of overwork is shown in Fig 2. A recent work has showed that to normalize employees dedication for different tasks to ensure they are not working overtime has found the best results in their experiments [2]. As our focus here is given to team efficiency, the original design in [1] is chosen here.

(4) Another constraint is that the solutions which assign employees who have no skills required for a task are unfa-

vorable. This is a soft constraint. The variable "$notwork$" is used to count the number of times when employees are assigned to tasks on which they do not have any skill required:

$$\begin{cases} notwork = |\{i, j | x_{ij} > 0, S_{e_i} \bigcap S_{k_j} = \phi, \\ \qquad 1 \le i \le m, 1 \le j \le n\}|. \end{cases} \tag{5}$$

The essential difference between "$notwork$" and "reqsk" is that any solution whose "reqsk" is larger than 0 implies that this solution is invalid, as at least one task cannot be completed. However a solution whose "$notwork$" is larger than 0 is unnecessarily to be invalid, as long as other employees who are also assigned to this task can finish it. Nevertheless, to assign employee(s) who have no skill required by tasks clearly increases the project cost, and perhaps also project duration.

The above project related attributes and constraints will be used into the design of the fitness function later.

D. Team efficiency

Defining team efficiency can be complicated. There are two types of team efficiency:

- task-specific team efficiency: such type of team efficiency is related to a specific task. A team may obtain different team efficiencies for various tasks;
- general team efficiency: such type of team efficiency applies to all tasks in a project.

Moreover, there are a variety of aspects to quantify team efficiency. This paper only studies the impacts of team efficiency on task duration.

Let a team T has a number of employees. The size of T is w, $w \ge 1$. Here individual efficiency is treated as a special case of team efficiency. The set of employees who are assigned to task k_j in a given (candidate) solution, is notated as E_{k_j}. If T is a subset of E_{k_j}, the team efficiency factor is applied to TaskDuration t_{k_j} by multiplication. δ is T's team efficiency factor, $\delta > 0$.

$$t_{k_j} = \begin{cases} \frac{L_{k_j}}{\sum_{i=1}^{m} x_{ij}} \times \delta, & \text{if } \Pi_{e_i \in E_{k_j}} x_{ij} > 0 \text{ and} \\ & T \subseteq E_{k_j}; \\ \frac{L_{k_j}}{\sum_{i=1}^{m} x_{ij}}, & \text{otherwise.} \end{cases} \tag{6}$$

For simplicity, only one T, i.e. one team efficiency factor, is applied to one problem instance throughout this paper. Given $T = \{e_1, e_2\}$ whose team efficiency factor is δ, δ applies to both $E_{k_j} = \{e_1, e_2, e_3\}$ and $E'_{k_j} = \{e_1, e_2, e_4\}$. However, given $T = \{e_1, e_2\}$ whose team efficiency factor is δ, and $T' = \{e_1, e_3\}$ whose team efficiency factor is δ', in one problem instance $E_{k_j} = \{e_1, e_2, e_3\}$'s team efficiency factor cannot be determined: δ and δ' cannot be applied simultaneously. To study the complexity of team efficiency will be one of the future work.

The definition (6) can be applied to task-specific efficiency or general efficiency.

E. Performance evaluation

T-SPMP is a typical multiple objective optimization problem with constraint satisfaction. The two objectives on pursuing the best solutions for this problem are to minimize the project cost and to minimize the project duration:

$$\begin{array}{ll} \text{minimize} & projectDuration, projectCost \\ \text{subject to} & undt = 0, \\ & reqsk = 0, \\ & overwork = 0. \end{array} \tag{7}$$

The specifications of team efficiency can be viewed as added preferences or soft constraints for multi-objective optimization. Equation (6) indicates that the team efficiency defined in this paper may affect selecting the desirable solutions through the objective of projectDuration.

With the knowledge of the objectives and constraints, candidate solutions are evaluated and biased. This work chooses the definitions of the fitness function in [1] as:

$$f(x) = \begin{cases} \frac{1}{q}, & \text{if } x \text{ is feasible;} \\ \frac{1}{(q+r)}, & \text{otherwise,} \end{cases} \tag{8}$$

where

$$q = 10^{-6} \times projectCost + 0.1 \times projectDuration \tag{9}$$

and

$$r = 100 + 10 \times undt + 10 \times reqsk + 0.1 \times overwork. \tag{10}$$

Function $f(x)$ is to be maximized. It contains two parts: q is the sum of the weighted objective values for feasible solutions; and r is the penalty for infeasible solutions. The search is guided to minimize q among feasible solutions, thus to maximize $f(x)$. This design ensures that feasible solutions are certainly more favorable than any infeasible one in selection. Equation (8) differentiates among infeasible solutions, such that solutions with a smaller number of violations to constraints are more favorable than those violate more constraints. The weights applied to "q" (9) and "r" (10) were chosen through experimentation, as explained in [1].

F. Problem instances

In order to compare with the experimental results in [1], the problems instances which are published at [20] are chosen. Each instance provides the values to the attributes aforementioned. Table II offers an example instance, specifying its attributes of tasks, employees, dependencies and team efficiency and their values. In practice, the instance values on team efficiency are estimated by management from previous records and managers' experiences.

III. PBIL AND EXPERIMENTAL SETUP

A heuristic optimization algorithm is chosen as the problem solving method: population-based incremental learning (PBIL) which generates probabilities over the possible values of candidate solutions. PBIL evolves a probability vector, but not a population of candidate solutions [21].

A. Probability vector representation

A probability distribution of possible values to a tuple x_{ij} is interpreted as employee e_i's dedication on task j: the percentage of e_i's normal working hours per day. For example, let consider the dedication of e_1 on task 2, tuple $x_{1,2}$. The possible values to $x_{1,2}$ are $\{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$. The number of possible dedications, notated by "$NoDdct$", is 11 here. This set of values has been chosen, as more precision will not help problem solving. Here dedication 0 shows that e_1 is not assigned to this task, and dedication 1 shows that s/he is assigned full time on this task. Table III offers three example probability distributions which are e_1's dedication to k_2, e_3's dedication to k_4, and e_5's dedication to k_7 respectively. Task 2 can be assigned to e_1 with a dedication 0.8 with a 20% probability, a dedication 0.9 with a 50% probability, or a full dedication with a 30% probability. So e_1 will certainly be assigned with a high dedication to task 1. Contrastingly, e_3 can be assigned to task k_4 with a dedication 0.1 with a 10% probability, and no dedication with a 90% probability. So, employee e_3 will unlikely be assigned to task 4. The probability distribution for $x_{5,7}$ indicates that as long as e_5 involves in the completion of task 7, it is no matter the value of his/her dedication. The sum of probabilities over all possible dedications for x_{ij} is 1.

TABLE III

THREE PROBABILITY DISTRIBUTIONS OF DEDICATION.

possible dedication value	$pv(x_{1,2})$	$pv(x_{3,4})$	$pv(x_{5,7})$
0	0	0.9	0
0.1	0	0.1	0.1
0.2	0	0	0.1
0.3	0	0	0.1
0.4	0	0	0.1
0.5	0	0	0.1
0.6	0	0	0.1
0.7	0	0	0.1
0.8	0.2	0	0.1
0.9	0.5	0	0.1
1	0.3	0	0.1

Given this representation, there are in total $i \times j \times NoDdct$ numbers to represent all probabilities over dedications of all tuples. A probability vector (pv) which consists of a set of probability distributions, each for a tuple x_{ij}. The relationship between pv and a candidate solution x is that a candidate solution is just one sampling from pv. Conventionally a candidate solution is called a "chromosome" in GA community. The "ChromLength" is $i \times j$. It is also the number of tuples x_{ij}. In this case, a candidate solution or a chromosome determines the dedications to each task. One sampling from the probability distributions in Table III could include $x_{1,2} = 0.9$ that means employee 1 will work on task 2 with a 90% dedication.

B. PBIL learning procedure

PBIL learning procedure follows the steps in Fig.3. Its' functions are implemented in pseudo code in Table IV,

```
1: Initialize ()
2: repeat
3:     Sample(pv)
4:     Evaluate(pv)
5:     Update(pv)
6:     Mutate(pv)
7: until stopping condition met
```

Fig. 3. PBIL learning procedure.

together with the explanation of its parameters. PBIL algorithm starts with initializing a probability vector (pv). At this stage, all tuples are assigned an equal probability, as specified in Function $Initialize()$ in Table IV. The sum of the probabilities over one tuple is 1. Then a population (set) of candidate solutions is generated from the probability vector pv. Due to equal probability, the initial population of sampled candidate solutions is random. The performances of the candidate solutions are evaluated by the fitness function (8).

Given their fitness values, pv is updated using the principle that the probabilities of the possible dedication values chosen by the best performed candidate solution(s) will be increased. For example, one candidate solution is really doing well. Its value on tuple $x_{1,2}$ is 0.8. Then the probability to the value 0.8 of the probability distribution $pv(x_{1,2})$ should increase while the probabilities of the rest possible values, i.e $\{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.9, 1\}$ of $pv(x_{1,2})$ decrease accordingly. The method to update probabilities is specified in the function $Update(pv)$ in Table IV.

TABLE IV

PBIL FUNCTION IMPLEMENTATION AND PARAMETERS. *: LEARNING RATE. **:MUTATION RATE ON PER VALUE OF pv.

Function	Implementation
$Initialize()$	$pv = (1/NOValues)\times$ $ones(NOValues, chromLength)$;
$Sample(pv)$	$if\ (pv(j,i) \geq r(i))$* $chrom(i) = possibleValues(j)$;
$Update(pv)$	$bestpv(m,j) = learningRate$; $pv = (1 - learningRate) \times pv$; $pv = pv + bestpv$;
$Mutate(pv)$	$if\ rand() < mutationRate$ ** $mutatePoints(round(rand()\times NOValues + 0.5), j) = mutationRate$; $pv(:,j) = (1 - mutationRate) \times pv(:,j) +$ $mutatePoints(:,j)$;

Parameter	Explanation
pv	Probability vector
NOValues	The number of possible values to x_{ij}
ChromLength	The length of chromosome
chrom	Chromosome (solution) sampled from pv
p	Population of $chroms$

Mutation plays a role to maintain a certain level of diversity in pv. How to mutate probabilities is specified in the function $Mutate(pv)$ in Table IV. Note that the mutation rate applies on per value of pv. There are in total $i \times j \times NoDdct$ values of a pv.

C. PBIL Parameters

The two major parameters for PBIL are the learning rate and the mutation rate. The sensible values to these two parameters are searched by experimentation. Four 100-runs are tested using the same problem instance and the same random sequences, but using different learning rates and different mutation rates. When learning rate is 0.1 and mutation is 0.1, none of the solutions found is infeasible, while the other learning rates and mutation rates tested have lead to infeasible solutions.

TABLE V

GAs AND PBIL PARAMETERS

Experimental parameters	Values for GAs [1]	Values for PBIL
$NoDdct$	Any real number $\in [0, 1]$	11
Size of population of candidate solutions	64	64
Selection	2-member tournament	Elitist
Recombination	2-D SPX	N/A
Mutation	Bit-Flip (1/length)	as in Table IV (mutation rate on per value of pv)
Replacement	Elitist	Re-sample Learning rate =0.1
Number of iteration	5000	300
Number of runs	100	100

IV. Experimental results

This section analyzes the experimental results on the impacts of team work. Totally four problem instances are tested. For each instance, 20 runs are experimented. starting with different random seeds.

The problem instance in Table II is one of the four instances. It specifies a team efficiency factor Eq.(6) as: if e_1 and e_2 are both assigned to a task, the duration of the task decreases 50%. This is the general type of team efficiency: its efficiency factor applies to all tasks of a project. Table VI specifies the team efficiency factors of the four instances. The following experimental results will demonstrate how such team efficiency factors make impacts on the solutions, comparing with the results on the matching problem instance of SPMP which do not consider team efficiency.

As mentioned earlier, the structure of team efficiency can be more complicated. The number of all combinations of employees can be huge; the values of impacts can be various. In this paper, the instance values on team efficiency presented are relatively simple: a team with two employees, its impact being a 50% variation on the task duration; and one problem instance only applying one of the team efficiency values in Table VI. This decision is made because, firstly a large number of team efficiency values in one instance will generate compound impacts so that the exact impacts of a particular team will be difficult to be investigated; and secondly, small values of team efficiency may not be influential enough to change the solutions, while other reasons, such as salaries and constraints can be more important to the performance of solutions to some problem instances. However this design is illustrative, emphasizing the team impact.

TABLE VI

TEAM EFFICIENCY EXAMPLES

Team efficiency values	**Explanation**
e1e2 duration $\uparrow$ team efficiency factor: $\delta = 1.5$	$e1$ and $e2$ are assigned to a task, the task duration increases 50%
e1e2 duration $\downarrow$ team efficiency factor: $\delta = 0.5$	$e1$ and $e2$ are assigned to a task, the task duration decreases 50%
e4e5 duration $\uparrow$ team efficiency factor: $\delta = 1.5$	$e4$ and $e5$ are assigned to a task, the task duration increases 50%
e4e5 duration $\downarrow$ team efficiency factor: $\delta = 0.5$	$e4$ and $e5$ are assigned to a task, the task duration decreases 50%

A. Overview

Table VII lists the project durations and project costs under five configurations on team efficiency: "10s-5e-10t" does not consider team efficiency. The rest configurations consider team efficiency: "e1e2 duration $\uparrow$" specifies that when e_1 and e_2 work together on a task, the task duration will increase 50%. This models the practical situations where two team members have a poor working relationship and cannot work productively together. Such a poor working relationship delays the task completion, even if there are other team members working on the same task. On the other hand, the configuration "e1e2 duration $\downarrow$" models the situations where two team members can work productively together and speed up the progress of task completion. The rest of configurations are explained in Table VI.

Table VII shows that for the configurations "e1e2 duration $\uparrow$" and "e4e5 duration $\uparrow$", their project durations and project costs have increased significantly, when compared with those of the configuration "10s-5e-10t". On the contrast, under the configurations "e1e2 duration $\downarrow$" and "e4e5 duration $\downarrow$", their project durations and project costs have decreased significantly, when compared with those of the configuration "10s-5e-10t". These demonstrate that team efficiency can be an important factor in assigning employees to tasks. To avoid choosing teams with a "bad combination" and to encourage choosing teams with a "good combination" help achieve objectives.

TABLE VII

PROJECT DURATION AND PROJECT COST WITH AND WITHOUT IMPACTS OF TEAM EFFICIENCY

Configuration	Fig.	project duration	project cost
10s-5e-10t	6(a)	16.44	785770
e1e2 duration $\uparrow$	6(b)	19.43	842440
e1e2 duration $\downarrow$	6(c)	13.10	595810
e4e5 duration $\uparrow$	6(d)	23.96	864460
e4e5 duration $\downarrow$	6(e)	11.00	476880

Fig 4 is the Gantt chart to the best solution found for the configuration “10s-5e-10t”. A near optimal solution to this configuration will complete the project within 16 working days. Fig 5 is the Gantt chart to the best solution found for the configuration “e4e5 duration ⇃”: when $e4$ and $e5$ are assigned to a task, this task will complete much sooner. A near optimal solution to this configuration will complete the project within only 11 working days.

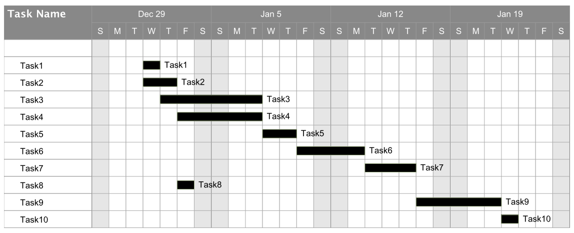

Fig. 4. Gantt chart to the best solution found for the configuration of the problem instance, “10s-5e-10t” which has no consideration on team efficiency.

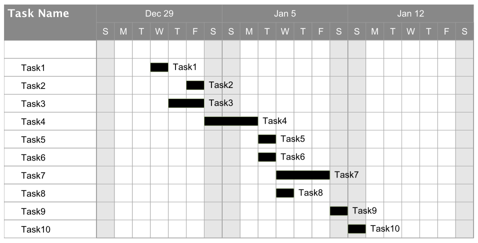

Fig. 5. Gantt chart to the best solution found for the configuration of the problem instance, “e4e5 duration ⇃”.

Fig 6 (a) shows the assignment of 5 employees to 10 tasks. The darkest blocks represent the largest dedication value, ‘1’ and white ones represent the smallest dedication value, ‘0’. For the rest, the darker a block, the higher values on dedication. There are 11 white blocks ($x_{ij} = 0$), meaning a zero dedication of the respective employee e_i on task k_j. The zero dedication is the outcome of various reasons: the employees do not have any skill required to complete the task; and/or employees have been fully assigned to other tasks for which there is no appropriate alternative, otherwise overwork occurred.

B. Team efficiency of e_1 and e_2

Fig 6 (b) shows the assignment under the configuration “e1e2 duration ↾”. Comparing with Fig 6 (a), there are four occurrences where tasks are not longer assigned to the related employees: e_1 is assigned to tasks k_2 and k_8 on (a), but is not assigned to these two tasks on (b); and e_2 is assigned to tasks k_1 and k_{10} on (a), but is not assigned to these two tasks on (b). Consequently, there are 5 tasks for which both e_1 and e_2 work on in (a), but only 1 task, task k_7, for which both e_1 and e_2 work on in (b). This demonstrates that the heuristic search avoids assigning both e_1 and e_2 to a task while optimizing the objectives. However both e_1 and e_2 must work on task k_7 together, because each has a skill required by k_7 which is not practised by any other employee.

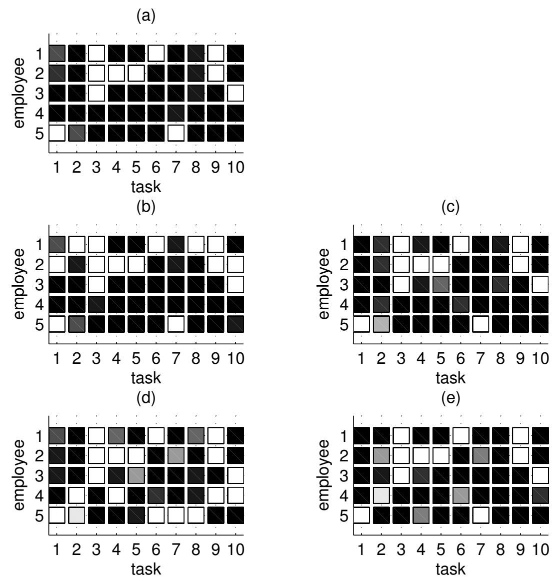

Fig. 6. Employee-task assignment under configurations: (a): “10s-5e-10t”: the assignment considering no team efficiency; (b): “e1e2 duration ↾” whose team efficiency factor: $\delta = 1.5$; (c): “e1e2 duration ⇃” whose team efficiency factor: $\delta = 0.5$; (d): “e4e5 duration ↾” whose team efficiency factor: $\delta = 1.5$; (e): “e4e5 duration ⇃” whose team efficiency factor: $\delta = 0.5$. x axis represents employees and y axis represents tasks. The blocks represents the values of dedication x_{ij}. The darkest blocks represent the largest value, ‘1’ and white blocks represent the smallest one, ‘0’. For the rest, the darker the block, the higher value on dedication.

For the rest, those employees who are assigned to tasks in (a) remain being assigned to the same tasks in (b), although their dedications slightly vary due to the aforementioned differences. In addition, those employees are not assigned to certain tasks remain zero dedication, namely $x_{1,3}, x_{1,6}, x_{1,9}, x_{2,3}, x_{2,4}, x_{2,5}, x_{2,9}, x_{3,3}, x_{3,10}, x_{5,1}$ and $x_{5,7}$ to satisfy the constraints of $undt$ (3), $reqsk$ (4) and $notwork$ (5).

Fig 6 (c) shows the assignment under the configuration “e1e2 duration ⇃”. Fig 6 (c) is similar to (a), although their dedications slightly vary. The shortened durations of the 5 tasks on which both e_1 and e_2 work, make an re-arrangement on dedications necessary. Under this configuration, it is desirable to assign both e_1 and e_2 to as many tasks as possible. However, due to the constraints, e_1 and e_2 will not work together for 5 tasks.

C. Team efficiency of e_4 and e_5

Fig 6 (d) shows the assignment under the configuration “e4e5 duration ↾”. Comparing with Fig 6 (a), there are six occurrences where e_4 or e_5 is assigned to a task on (a), but is not assigned to these tasks on (d). This demonstrates again that the heuristic search avoids assigning both e_4 and e_5 to a task while optimizing the objectives. Both e_4 and e_5 must work together on tasks k_3 and k_5, because each employee has skills required by these two tasks, and these skills are not practised by any other employee.

Fig 6 (e) shows the assignment under the configuration "e4e5 duration $\downarrow$". Fig 6 (e) is similar to (a), although their dedications slightly vary. This is because the shortened durations of 7 tasks make an re-arrangement on dedications necessary. Under this configuration, it is desirable to assign both e_4 and e_5 to as many tasks as possible. However, due to the constraints, e_4 and e_5 will not work for two tasks together. Please note $x_{4,2}$ is 0.1, not zero.

In summary, the resulting solutions under the five comparative configurations demonstrate the compound impacts and trade-off of team efficiency, constraints and objectives. These findings indicate that team efficiency can be influential in project scheduling, especially when the impact of team efficiency overpowers the impact of other influential factors.

V. Conclusions

This paper explores team efficiency in the context of project scheduling. A richer problem (T-SPMP) to the well-studied software project management problem (SPMP) is introduced with added considerations on employees' team efficiency.

As T-SPMP is hard, a heuristic optimization method, population based incremental learning (PBIL), is applied. It evolves probability vectors over possible values to solutions. Candidate solutions are generated by sampling evolved probability vectors.

The experimental results suggest that given a sufficient number of iterations, appropriate learning rates and mutation rates, PBIL was able to find probability vectors which always produce feasible and desirable solutions to the problem instances tested.

This work provides insightful information to managers to make better informed decisions for task-employee allocation in complicated situations. It also shows the worth of comprehensive studies on various configurations on team efficiency.

In future, more comprehensive experimental studies need to carried out on a large and more diverse set of test problem instances, in order to gain a better understanding of why and how the proposed algorithm behaves on different T-SPMPs.

Acknowledgment

Authors thank the team members of the SEBASE and DAASE projects at The Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA), the University of Birmingham, for their support and discussions. Authors would also like to thank the reviewers and Longzhi Yang of Northumbria University for their comments.

References

[1] E. Alba and J. F. Chicano, "Software project management with gas," *Information Sciences*, pp. 2380–2401, 2007.

[2] L. Minku, D. Sudholt, and X. Yao, "Improved evolutionary algorithm design for the project scheduling problem based on runtime analysis," *Software Engineering, IEEE Transactions on*, vol. 40, no. 1, pp. 83–102, Jan 2014.

[3] K. J. Shim and J. Srivastava, "Team performance prediction in massively multiplayer online role-playing games (mmorpgs)," in *Social Computing (SocialCom), 2010 IEEE Second International Conference on*, 2010, pp. 128–136.

[4] M. Hoegl and H. G. Gemuenden, "Teamwork quality and the success of innovative projects: A theoretical concept and empirical evidence," *Organization Science*, vol. 12, no. 4, pp. 435–449, 2001.

[5] I. W. Fung, C. Huang, and C. Tam, "Application of {GA} optimization for solving precedent relationship problem in project scheduling," *Mathematical and Computer Modelling*, vol. 57, no. 910, pp. 2067 – 2081, 2013.

[6] H. Kazemipoor, R. Tavakkoli-Moghaddam, P. Shahnazari-Shahrezaei, and A. Azaron, "A differential evolution algorithm to solve multi-skilled project portfolio scheduling problems," *The International Journal of Advanced Manufacturing Technology*, vol. 64, no. 5-8, pp. 1099–1111, 2013.

[7] J. Xiao, X.-T. Ao, and Y. Tang, "Solving software project scheduling problems with ant colony optimization," *Computers & Operations Research*, vol. 40, no. 1, pp. 33 – 46, 2013.

[8] W.-N. Chen and J. Zhang, "Ant colony optimization for software project scheduling and staffing with an event-based scheduler," *IEEE Transactions on Software Engineering*, vol. 39, no. 1, pp. 1–17, 2013.

[9] V. Zeighami, R. Akbari, I. Akbari, and Y. Biletskiy, "An abc-genetic method to solve resource constrained project scheduling problem," *Artificial Intelligence Research*, vol. 1, no. 2, pp. 185–197, 2012.

[10] R. Golestaneh, A. Jafari, M. Khalilzadeh, and H. Karimi, "Minimizing total resource tardiness penalty costs in the resource constrained project scheduling problem with metaheuristic algorithms," *International Journal of Research in Industrial Engineering*, vol. 2, no. 3, pp. 47–57, 2013.

[11] C. Fang and L. Wang, "An effective shuffled frog-leaping algorithm for resource-constrained project scheduling problem," *Computers & Operations Research*, vol. 39, no. 5, pp. 890 – 901, 2012.

[12] O. Bellenguez-Morineau and E. Nron, "A branch-and-bound method for solving multi-skill project scheduling problem," *RAIRO - Operations Research*, vol. 41, pp. 155–170, 4 2007.

[13] H. Li and K. Womer, "Scheduling projects with multi-skilled personnel by a hybrid milp/cp benders decomposition algorithm," *Journal of Scheduling*, vol. 12, no. 3, pp. 281–298, 2009.

[14] C. Artigues, M. Gendreau, L.-M. Rousseau, and A. Vergnaud, "Solving an integrated employee timetabling and job-shop scheduling problem via hybrid branch-and-bound," *Computers & Operations Research*, vol. 36, no. 8, pp. 2330 – 2340, 2009.

[15] F. Ferrucci, M. Harman, J. Ren, and F. Sarro, "Not going to take this anymore: Multi-objective overtime planning for software engineering projects," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13, 2013, pp. 462–471.

[16] C. K. Chang, H. yi Jiang, Y. Di, D. Zhu, and Y. Ge, "Time-line based model for software project scheduling with genetic algorithms," *Information and Software Technology*, vol. 50, no. 11, pp. 1142 – 1154, 2008.

[17] F. Wena and C.-M. Lin, "Multistage human resource allocation for software development by multiobjective genetic algorithm," *The Open Applied Mathematics Journal*, vol. 2, pp. 95–103, 2008.

[18] Q. Zhang and H. Muhlenbein, "On the convergence of a class of estimation of distribution algorithms," *Evolutionary Computation, IEEE Transactions on*, vol. 8, no. 2, pp. 127–136, April 2004.

[19] E. Tsang, J. Ford, P. Mills, R. Bradwell, R. Williams, and P. Scott, "Towards a practical engineering tool for rostering," *Annals of Operations Research*, vol. 155, no. 1, pp. 257–277, 2007.

[20] E. Alba and J. F. Chicano, "An instance generator for the project scheduling problem," University of Mlaga, Tech. Rep., 2005. [Online]. Available: http://tracer.lcc.uma.es/problems/psp/generator.html

[21] S. Baluja, "Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning," Computer Science Department, Pittsburgh, PA, Tech. Rep. CMU-CS-94-163, 1994.