Evolutionary Algorithms for the Project Scheduling Problem: Runtime Analysis and Improved Design*

Leandro L. Minku
CERCIA
University of Birmingham
Birmingham B15 2TT, UK

Dirk Sudholt
Dept. of Computer Science
University of Sheffield
Sheffield S1 4DP, UK

Xin Yao
CERCIA
University of Birmingham
Birmingham B15 2TT, UK

ABSTRACT

Even though genetic algorithms (GAs) have been used for solving the project scheduling problem (PSP), it is not well understood which problem characteristics make it difficult/easy for GAs. We present the first runtime analysis for the PSP, revealing what problem features can make PSP easy or hard. This allows to assess the performance of GAs and to make informed design choices. Our theory has inspired a new evolutionary design, including normalisation of employees' dedication for different tasks to eliminate the problem of exceeding their maximum dedication. Theoretical and empirical results show that our design is very effective in terms of hit rate and solution quality.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems

Keywords

Search-based software engineering, runtime analysis, project scheduling, theory, evolutionary algorithms

1. INTRODUCTION

Scheduling tasks and assigning resources in large-scale software projects is a very challenging problem. In the project scheduling problem (PSP) [4, 2, 5] the goal is to assign employees to tasks such that all employees have all required skills for their tasks, task precedence constraints among tasks are respected, the completion time is minimised, and the cost in terms of salaries is minimised.

In contrast to other resource-constrained scheduling problems, employees can divide their attention among several tasks at the same time. This is often modeled by dedication values [2] representing the percentage of time an employee spends on a certain task. Employees have a maximum dedication, which must be respected to avoid overwork.

*Authors in alphabetical order.

This renders the PSP more complex than classical scheduling problems. Many researchers resorted to genetic algorithms (GAs) to solve the problem. Different GAs were presented for different variants of the PSP [6, 4, 2, 11, 5]. But it is not well understood which problem characteristics make the problem difficult to solve for evolutionary algorithms (EAs). Some progress was made in the well-known work by Alba and Chicano [2]. They presented an effective GA and a systematic empirical performance analysis on generated benchmark instances, to analyse the impact of problem features on GA's performance.

Our work makes two important contributions. We present the first runtime analysis for the PSP. This theoretical approach has been applied to many problems from combinatorial optimisation [15, 14], and it has led to many interesting results about how EAs perform. We use runtime analysis to gain insight into how simple EAs perform on illustrative instance classes. This sheds light on what instances are easy, and which ones are hard to solve for EAs. It also helps to make more informed design choices for the PSP.

Inspired by theoretical insight, we make a second, more practical contribution: a new mechanism for normalising dedication values. Alba and Chicano [2] showed that GAs spend most of their effort avoiding overwork. This is a major problem even on simple instances, as GAs have a very low hit rate in finding feasible schedules. Our approach normalises dedication values automatically. Normalisation is embedded into the genotype-phenotype mapping, extending the mapping from [2]. Whenever an employee has a total dedication greater than the maximum dedication, all dedication values to active tasks are scaled accordingly. This completely removes overwork from the problem. Instead of struggling to find a solution without overwork, our approach allows EAs to focus on the solution quality. In addition, we introduce a tailored mutation operator and a new way of dealing with infeasible solutions, guiding EAs to reach feasibility.

Both theoretical and empirical results show that our modifications are very effective. A (1+1) EA in our improved design performs better than the existing GA [2]. It reaches feasible solutions in 100% of all tested cases, whereas the GA struggles to reach feasibility. Also in terms of solution quality our approach was better.

This work demonstrates how theoretical research can improve algorithm design in practice.

2. PROBLEM FORMULATION

Our problem formulation is based on existing work [2]. Assume we are given

- a set of employees $e_1, \ldots, e_n$ with salaries $s_1, \ldots, s_n$, and skills $\text{skill}_1, \ldots, \text{skill}_n$, respectively,
- a set of tasks $t_1, \ldots, t_m$ with efforts $\text{eff}_1, \ldots, \text{eff}_m$, and required skills $\text{req}_1, \ldots, \text{req}_m$, respectively, and
- a task precedence graph (TPG), a directed graph with tasks as nodes and task precedence as edges.

The set of tasks and TPG can be referred to as the project to be scheduled. The goal is to assign employees to tasks in order to minimize the completion time as well as the costs for the project (i. e. salaries paid). Employees can work on several tasks simultaneously, as indicated by their *dedication* to certain tasks. The dedication is the fraction of their time devoted to a particular task. The completion time is the time the project is completed.

The amount of dedication of an employee e_i to a task t_j is determined by a value $x_{i,j} \in \{0/k, 1/k, \ldots, k/k\}$, where $k \in \mathbb{N}$ reflects the granularity of the solution. For $k = 1$ we only have dedications 0 and 1. For, say, $k = 10$ we have $k + 1 = 11$ dedications of $0\%, 10\%, \ldots, 100\%$. Employees can only work on a task t_j if all employees working together have all the skills to do the task. More formally, we require

$$\text{req}_j \subseteq \bigcup_{i=1}^{n} \{\text{skill}_i \mid x_{i,j} > 0\}. \qquad (1)$$

Alba and Chicano [2] also considered a maximum dedication for each employee. It reflects how much of a full-time job an employee is able to work. This can reflect part-time jobs as well as the willingness for working overtime. However, for the sake of simplicity we leave this for future work and use the same maximum dedication of 1 for all employees.

3. PREVIOUS WORK

Alba and Chicano's work [2] is probably the most well-known and represents the state-of-the-art in solving PSP using GAs. They represent candidate solutions for the PSP as a matrix of binary values which encode the dedications $x_{i,j}$ for each employee e_i and task t_j. The recombination operator is a 2-D single point crossover, which randomly selects a row and a column and then swaps the elements in the upper left and in the lower right quadrant of the two parents. They used bit-flip mutation, binary tournament selection of two parents for recombination, and survival selection based on elitist replacement of the worst individual.

A candidate solution is considered infeasible if there is a task with no employee associated, or the skills constraint (eq. 1) is not satisfied, or there are employees working overtime. Overtime can happen when tasks are executed in parallel, as the total dedication of employees in the candidate solution can exceed 1. Based on that, the fitness function is defined as follows:

$$f(x) = \begin{cases} 1/q & \text{if the solution is feasible} \\ 1/(q+p) & \text{otherwise,} \end{cases}$$

where $q = w_{\text{cost}} \cdot \text{cost} + w_{\text{time}} \cdot \text{time}$,
$p = w_{\text{penal}} + w_{\text{undt}} \cdot \text{undt} + w_{\text{reqsk}} \cdot \text{reqsk} + w_{\text{over}} \cdot \text{over}$,
and w_{cost}, w_{time}, w_{penal}, w_{undt}, w_{reqsk} and w_{over} are predefined parameters, cost and time are the cost and completion time of the solution, undt is the number of tasks with no employee associated, reqsk is the number of skills still required to perform all project tasks, and over is the total overwork time spent by all employees during the project.

Their fitness function penalizes infeasible solutions, but whether or not it gives hints as to how to reach feasible solutions depends on the chosen values for several pre-defined parameters (w_{penal}, w_{undt}, w_{reqsk} and w_{over}). Moreover, Alba and Chicano's experimental analysis reveals that the constraint overwork can cause their GA to have very low hit rate in finding feasible solutions. This holds even for very simple instances where the skill constraints have been removed (every employee has all skills) and all employees have the same salary.

4. OUR APPROACH

In order to overcome problems related to the hit rate and to improve solution quality, we propose an improved design, which consists of two main points. The first one is normalising employee's dedications (section 4.1) to eliminate the problem of overwork. The second one is to give a clear gradient for searching towards feasibility by introducing a new type of penalty in the evaluation of cost and completion time (section 4.2). We do not aim to improve the GA itself. Instead, we show in our experimental analysis (section 6) that our improved design allows very good results to be achieved even when using a very simple EA such as the one explained in section 4.3. The use of other EAs is left as future work.

4.1 Normalising Dedications

Instead of penalizing overwork and letting an EA search for feasible solutions, we normalise the dedication values. If at some point of time the total dedication of an employee e_i across all active tasks is $d_i > 1$ then her/his dedication for all tasks is divided by d_i. This reflects a very natural way of an employee dividing her/his attention to several tasks.

For instance, assume there are two tasks t_1, t_2 suitable for employee e_1. Assume also that the employee works on both tasks at overlapping time intervals. If $x_{1,1} + x_{1,2} > 1$, the employee works overtime whenever she/he works on both t_1 and t_2 at the same time. If $x_{1,1} + x_{1,2} \leq 1$, on the other hand, resources are wasted when the employee works on a single task. So, no matter the values for $x_{1,1}$ and $x_{1,2}$, there will always be overwork or resources wasted—unless both tasks start and finish at exactly the same time. Note that, depending on $x_{1,1}$ and $x_{1,2}$, there could be even both resources wasted when the employee is working on a single task and overwork when working on both tasks at the same time (whenever $x_{1,1} < 1 \land x_{1,2} < 1 \land x_{1,1} + x_{1,2} > 1$).

Note that we do not normalise "underwork", i. e., total dedications less than 1. This would otherwise remove the possibility of balancing cost vs. completion time.

Normalisation allows for much more fine-grained schedules as employees can automatically re-scale their dedications as soon as tasks are finished or new tasks are started. Examples of schedules with normalisation are given later on in Figure 1. First we describe how the final schedule can be computed from dedication values and the problem instance.

4.2 Evaluating Costs and Completion Time

Algorithm 1 computes both cost and completion time according to an implicit Gantt diagram. For schedules that are infeasible because certain skills are missing, the algorithm returns very high costs and completion times. Both values grow with the number of missing/required skills. When cost and completion times are used in any reasonable single- or multi-objective fitness function, this gives a clear gradient for search algorithms towards feasibility. The algorithm makes the mapping informally described in [2] explicit and it includes the modifications presented here.

The algorithm first tests whether the genotype is feasible in that every task can be completed in finite time (lines 1-3). That is the case when the skills constraint (eq. 1) is not satisfied. Note that this includes the case in which there is a task with no employee designated. So, if the genotype is infeasible, a penalty vector $(\text{reqsk} \cdot 2 \sum_{i=1}^{n} \sum_{j=1}^{m} s_i \text{eff}_j, \text{reqsk} \cdot 2k \sum_{j=1}^{m} \text{eff}_j)$ is returned (line 5), where reqsk is the number of required skills. Both values are higher than for any feasible schedule as the cost is always at most $\sum_{i=1}^{n} \sum_{j=1}^{m} s_i \text{eff}_j$ and the time is always at most $k \sum_{j=1}^{m} \text{eff}_j$. Also, any decrease in the number of required skills strictly decreases both components through reqsk. This gives strong hints for any search algorithm to reach the feasible region.

If the dedication matrix is the *genotype* during optimisation, the final schedule can be called *phenotype*. The final schedule for any feasible genotype can be assessed by storing the $d_{i,j}$-values from each iteration, along with the corresponding time stamps. This defines a mapping that transforms any genotype (i. e., a matrix of dedication values) to a corresponding phenotype (i. e., a schedule).

The algorithm iteratively constructs the schedule. We check which tasks can be active at the current point of time (line 7). The normalised dedication for all suitable employees is computed for all these tasks (line 12). Then we determine the earliest point of time t at which a task is finished (line 14). All finished tasks are being marked as finished (line 20), so that the next iteration can include new tasks, according to the task precedence graph. If there are tasks that have been started, but are not finished yet, their effort is updated to the remaining effort needed for completing them (line 18). This accounts for potentially piecewise evaluations of particular tasks. Along the way, all completion times and costs in all iterations are added up (lines 15-16).

The computational complexity of the genotype-phenotype mapping and the fitness evaluation can be bounded by $O(nm^2)$, with the use of appropriate data structures. The basic idea is to maintain counters for the in-degree of each vertex. The counters and the set V' can be updated efficiently whenever edges are removed from the TPG: if the counter of a vertex reaches 0, it is added to V'. The time complexity is quite low, given that the input size is $\Theta(nm)$. The feasibility check can be implemented in time $O(nm)$.

4.3 Evolutionary Algorithm

As one of our goals is to present the first runtime analysis for the PSP, we follow many previous examples in this area and consider one of the simplest EAs: the (1+1) EA. It is simple because it neither uses a population nor recombination (crossover). One generation consists of mutation, and then selection checks whether the mutation has found an improvement.

Alba and Chicano [2] used a binary encoding to represent the $x_{i,j}$. Instead, we work directly in the search space $\{0/k, \dots, k/k\}^{nm}$. Mutation chooses which components to change and then for each one it picks a new value uniformly at random. Algorithm 2 describes the (1+1) EA for our search space, for minimising some fitness function f.

The fitness function to be used in our experimental analysis is $f(x) = w_{\text{cost}} \cdot \text{cost} + w_{\text{time}} \cdot \text{time}$, where cost and time are obtained from algorithm 1. The runtime analysis is not restricted to this fitness function.

Algorithm 1 Evaluate(cost, time, TPG)
Output: (cost, time)

```
1: Let reqsk = 0.
2: for all tasks $t_j$ do
3:     reqsk = reqsk + $|\text{req}_j \setminus \bigcup_{i=1}^{n} \{\text{skill}_i \mid x_{i,j} > 0\}|$.
4: if reqsk > 0 then
5:     Output $\left(\text{reqsk} \cdot 2 \sum_{i=1}^{n} \sum_{j=1}^{m} s_i \text{eff}_j, \text{reqsk} \cdot 2k \sum_{j=1}^{m} \text{eff}_j\right)$;
       stop.
6: while TPG $\neq \emptyset$ do
7:     Let $V'$ be the set of all unfinished tasks without incoming edges in TPG.
8:     if $V' = \emptyset$ then
9:         Output "Problem instance not solvable!" and stop.
10:    for all tasks $t_j$ in $V'$ do
11:        for all employees $e_i$ do
12:            Let $d_{i,j} := \frac{x_{i,j}}{\max\left(1, \sum_{t_\ell \in V'} x_{i,\ell}\right)}$.
13:        Compute the total dedication $d_j := \sum_{i=1}^{n} d_{i,j}$.
14:    Let $t := \min_j(\text{eff}_j / d_j)$.
15:    Let cost := cost + $t \sum_{i=1}^{n} s_i \sum_{j=1}^{m} d_{i,j}$.
16:    Let time := time + $t$.
17:    for all tasks $t_j$ in $V'$ do
18:        Let $\text{eff}_j := \text{eff}_j - t \cdot d_j$.
19:        if $\text{eff}_j = 0$ then
20:            Mark $t_j$ as finished and remove it and its incident edges from TPG.
21: Output (cost, time) and stop.
```

Algorithm 2 (1+1) EA for project scheduling

```
1: repeat
2:     Create $x'$ by copying $x$.
3:     for each $1 \leq i \leq n, 1 \leq j \leq m$ do
4:         With probability $1/(nm)$ replace $x'_{i,j}$ by a value chosen u. a. r. from $\{0/k, 1/k, \dots, k/k\} \setminus \{x_{i,j}\}$.
5:     if $f(x') \leq f(x)$ then $x := x'$.
6: until happy
```

5. RUNTIME ANALYSIS

In the following, we estimate the *optimisation time* of the (1+1) EA, defined as the first generation in which a global optimum is found. We will also investigate *randomised local search (RLS)*. It differs from the (1+1) EA in that during mutation exactly one dedication value is changed. The entry is chosen uniformly at random.

We make few assumptions about the fitness function f to keep the analyses as general as possible. We only assume that f is *Pareto-compliant* in a strict sense: if x' Pareto-dominates x (i. e., $\text{cost}(x') \leq \text{cost}(x) \wedge \text{time}(x') \leq \text{time}(x)$) and x does not Pareto-dominate x' then $f(x') < f(x)$. That is, any improvement in one or both objectives also improves the fitness f. This applies to any weighted combination of cost and completion time. In the special case where all employees have the same salary, the costs are always the same. Then f boils down to minimising the completion time.

5.1 Optimal Completion Times

We start with a structural result. The two goals of minimising costs and completion time are often conflicting. We first look at the extreme case of minimising the completion time only. In every feasible solution for each task the team has the required skills for the task. This also holds if more employees join in working on a task, regardless of their skills. The following theorem is an immediate conclusion.

THEOREM 1. *For every solvable PSP instance, the completion time is minimal if in the schedule all employees always work full time. Then the completion time is* $1/n \cdot \sum_{j=1}^{m} \mathrm{eff}_j$.
If normalisation is used, a sufficient condition for minimality is that all dedication values are 1.

The second statement is not true without normalisation. Without it, the difficulty for optimization is to find the ideal balance between different tasks, while avoiding overwork. When normalisation is used, this difficulty disappears.

5.2 A General Lower Bound

We first present a lower bound on the expected running time, indicating the least time we should allow for running an EA on the problem. The only requirement is that there is a single globally optimal dedication matrix with respect to the chosen fitness function. This also applies in settings where there is a single Pareto-optimal solution for minimising cost and completion time. In a more general sense, the analysis applies to the time for finding any fixed target point.

THEOREM 2. *Consider a PSP instance with* n *employees and* m *tasks,* $nm > 1$, *with a single global optimum (a fixed target) in the fitness function used. The expected optimization time (time to hit the target) of RLS and the (1+1) EA, with or without normalisation, is at least* $\Omega(knm \log(nm))$.

Proof. Call an entry of the dedication matrix *bad* if it disagrees with the global optimum. Observe that in order to find the optimum it is necessary to change all bad entries at least once in a mutation. Each entry is bad at initialization with probability $k/(k+1)$. The expected number of bad initial entries is $knm/(k+1)$. By classical Chernoff bounds [13], with probability $e^{-\Omega(nm)}$ the initial number of bad entries is at least $nm/3$. Assume an initial number of $nm/3$ bad entries and define $t := (knm - 1)\ln(nm/3)$. The probability of not changing a particular entry in t mutations is $(1 - 1/(nm))^t$. The probability that there is a bad entry which is never turned good during t mutations is at least

$$1 - \left(1 - \left(1 - \frac{1}{knm}\right)^t\right)^{nm/3} \geq 1 - \left(1 - e^{-\ln(nm/3)}\right)^{nm/3}$$
$$\geq 1 - \left(1 - \frac{3}{nm}\right)^{nm/3} \geq 1 - e^{-1}.$$

Using the union bound for the initialization, with probability at least $1 - e^{-1} - e^{-\Omega(nm)} = \Omega(1)$ the algorithm has not found an optimum after $t = \Omega(knm \cdot \ln(nm))$ steps. This establishes the bound $\Omega(1) \cdot t = \Omega(knm \cdot \log(nm))$. □

The lower bound increases linearly with the granularity parameter k. This makes sense as every entry needs to be set to one out of k possible choices, and mutation makes these choices uniformly. The term $nm \ln(nm)$ reflects the time until mutation has affected all wrong dedication values.

5.3 Time to Feasibility

In order to see how efficiently our treatment of infeasible solutions guides evolution towards feasible search points, we estimate the expected time until feasibility is reached.

To this end, we first cite a recent result from drift analysis. It is used to determine the expected time until a Markov chain $\{X_t\}_{t \in \mathbb{N}}$ reaches a designated target state 0.

THEOREM 3 (JOHANNSEN'S VARIABLE DRIFT THM.[10]). *Let* $\{X_t\}_{t\in\mathbb{N}}$ *be a sequence of random variables over a finite state space* $S \subseteq \mathbb{R}_0^+$ *and let* $x_{\min} := \min\{x \in S \colon x > 0\}$. *Furthermore, let* T *be the random variable that denotes the first point in time* $t \in \mathbb{N}$ *for which* $X_t = 0$. *Suppose that there exists a continuous and monotone increasing function* $h \colon \mathbb{R}_0^+ \to \mathbb{R}^+$ *such that* $\mathrm{E}(X_t - X_{t+1} \mid X_t) \geq h(X_t)$ *holds for all* $t < T$. *Then,*

$$\mathrm{E}(T \mid X_0) \leq \frac{x_{\min}}{h(x_{\min})} + \int_{x_{\min}}^{X_0} \frac{1}{h(x)}\, \mathrm{d}x$$

Using this variable drift, we get the following with a very reasonable assumption: the total number of skills is not larger than some polynomial in nm.

THEOREM 4. *Consider RLS or the (1+1) EA, with or without normalisation, on any solvable instance where* $\sum_{i=1}^{n} |\mathrm{skill}_i| \leq (nm)^\delta$ *for some constant* $\delta \geq 0$. *The expected time until some feasible schedule is found is* $O(nm \log(nm))$.

The upper bound is by a factor of k smaller than the lower bound from Theorem 2. This indicates that the time to feasiblity is only a small fraction of the optimisation time.

Proof of Theorem 4. Note that, as long as the current solution is infeasible, the fitness is uniquely determined by the number of required skills. Let $X_0, X_1, \ldots$ describe this number and its development over time. A feasible solution is found once this value reaches 0. We claim that

$$\mathrm{E}(X_{t+1} - X_t \mid X_t > 0) \geq X_t \cdot \frac{1}{enm}$$

for both algorithms (for RLS one can remove the factor e). Applying the variable drift theorem (Theorem 3) with $h(x) = x/(enm)$ yields an upper bound of

$$\frac{1}{1 - \frac{1}{enm}} + \int_1^{X_0} \frac{enm}{x}\, \mathrm{d}x = O(1) + enm \cdot \ln(X_0)$$

and this is $O(nm \log(nm))$ as $X_0 \leq \sum_{i=1}^{n} |\mathrm{skill}_i| \leq (nm)^\delta$, so $\log(X_0) \leq \delta nm = O(nm)$.

It remains to prove the claim. Fix any task t_j and let $m_j := |\mathrm{req}_j \setminus \bigcup_{i=1}^{n} \{\mathrm{skill}_i \mid x_{i,j} > 0\}|$ describe the number of required skills with respect to t_j. As the instance is solvable, making all employees work on t_j reduces the number of required skills by m_j. A mutation that only increases the dedication of a fixed employee with $x_{i,j} = 0$ has probability at least $1/(nm) \cdot (1 - 1/(nm))^{nm-1} \geq 1/(enm)$. Hence, the expected reduction of $|\mathrm{req}_j \setminus \bigcup_{i=1}^{n} \{\mathrm{skill}_i \mid x_{i,j} > 0\}|$ in one generation is at least $m_j \cdot 1/(enm)$. As this holds for all tasks and $X_t = \sum_{j=1}^{m} m_j$, we get $\mathrm{E}(X_{t+1} - X_t \mid X_t > 0) \leq X_t \cdot 1/(enm)$. This implies the claim and the upper bound. □

5.4 Easy Instances: Linear Schedules

Now we turn to a class of illustrative "easy" instances. Assessing how effective an EA is on easy instances is an excellent starting point for an analysis. It gives a good baseline for comparisons with other results on the running time, and most importantly we learn about the search behaviour of an EA. We need a good understanding of easy instances before we can move on to more complicated instance classes.

As easy cases we consider schedules with a "linear" structure: the task precedence graph is a chain of m vertices, or more generally any directed acyclic graph that contains such a chain. In this case all tasks have to be completed sequentially. Note that if normalisation is switched on, it is never actually applied as at each time only one task is processed. The issue of whether normalisation is used or not is irrelevant for linear schedules.

We also assume that all salaries are equal. So the problem boils down to minimising the completion time. We first give a result for RLS as it is easier to analyse.

THEOREM 5. *Consider an instance with n employees with equal salaries and m tasks with arbitrary positive efforts. Let the task precedence graph contain an m-vertex chain as subgraph. Then the expected optimization time of RLS, with or without normalisation, is of order $\Theta(knm\ln(nm))$.*

Proof. The lower bound follows from Theorem 2.

By Theorem 4 we know that a feasible solution is found in time $O(nm\ln(nm))$. Afterwards, increasing the dedication of any employee strictly decreases the time until this task, and hence the whole schedule, is completed. The remaining expected optimization hence equals the expected time until all entries of the dedication matrix are 1 (cf. Theorem 1).

As no entry can be decreased in RLS, we can use the following argument. Call an entry $x_{i,j}$ *good* if $x_{i,j} = 1$ and *bad* otherwise. The number of bad entries is monotone decreasing over time. If the current number is ℓ, the probability of further decreasing it equals $\ell/(knm)$ as any entry $x_{i,j} < 1$ is set to $x_{i,j} = 1$ with probability $1/k$. The expected time until this happens is knm/ℓ. Summing all expected times, starting with at most nm bad entries, yields the upper bound

$$\sum_{\ell=1}^{nm} \frac{knm}{\ell} = knm \cdot \sum_{\ell=1}^{nm} \frac{1}{\ell} = O(knm\log(nm)). \qquad \square$$

We also prove a polynomial upper bound for the (1+1) EA, if all efforts are polynomial in n, m.

THEOREM 6. *In the setting of Theorem 5, if additionally $\text{eff}_j \geq 1$ for all $1 \leq j \leq m$ then the expected optimisation time of the (1+1) EA, with or without normalisation, is bounded by $ek^3n^3m\sum_{j=1}^{m}\text{eff}_j$.*

Proof. Clearly, the completion time is at most $k\sum_{j=1}^{m}\text{eff}_j$. We claim that for each non-optimal schedule there is always at least one mutation that decreases the completion time by at least $1/(kn^2)$.

Consider a task j where not all employees have full dedication. The decrease of the completion is minimal in case $n-1$ employees have full dedication and the remaining employee has dedication $(k-1)/k$. Then a mutation increasing the latter value to 1 decreases the time for this task by

$$\frac{\text{eff}_j}{n - 1/k} - \frac{\text{eff}_j}{n} = \frac{\text{eff}_j/k}{n^2 - n/k} \geq \frac{1}{kn^2}.$$

The probability of making this mutation is at least $1/(eknm)$. Then the expected decrease of the completion time is at least $1/(ekn^3m)$. The upper bound then follows from drift analysis (Theorem 3): we take X_t as the difference to the optimal completion time, note $X_0 \leq k\sum_{j=1}^{m}\text{eff}_j$ and put $h := 1/(ekn^3m)$. $\square$

The upper bound for the (1+1) EA is much higher than the upper bound for RLS. We conjecture that an upper bound of $O(knm\ln(nm))$ also applies for the (1+1) EA, i. e., both algorithms have the same asymptotic running time.

Proving this, however, is tough. Even though the problem looks similar to the simple problem OneMax, many obstacles make it much more complex: arbitrary weights for tasks, a non-linear relation between single dedication values and their contribution to fitness, and a representation with strings of more than 2 values.

If $k = 1$, i. e., only dedications 0 and 1 are allowed, the analysis becomes easier. It is not hard to see that then the fitness function is *monotone*: flipping only 0-bits to 1 and not flipping any 1-bit to 0 results in a strict fitness improvement. By Doerr *et al.* [8] this yields the following.

THEOREM 7. *In the setting of Theorem 6, if $k = 1$ then the expected optimisation time of the (1+1) EA is $O((nm)^{3/2})$. Additionally, if the mutation probability is changed to $c/(nm)$ for a constant $0 < c < 1$ the expected time is even bounded by $O(nm\log(nm))$.*

This supports our conjecture that the (1+1) EA is as effective as RLS for linear schedules.

5.5 Difficult Instances

We also look at difficult instances to get insight into what makes the PSP hard. Linear schedules are easy to solve, so we consider settings with tasks being processed in parallel. We have already seen in section 4.1 that without normalisation an EA struggles in finding an optimal balance between dedications for different tasks. With normalisation this problem becomes a lot easier. But we show in the following that even with normalisation, RLS and the (1+1) EA can struggle in finding an optimal balance.

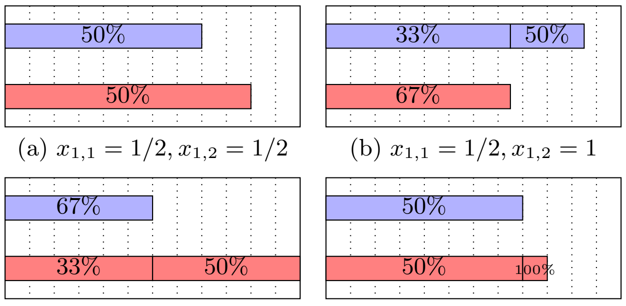
In particular, RLS can get stuck in local optima even on a very simple and tiny problem instance where costs are irrelevant. The instance contains two tasks with efforts 4 and 5, respectively. There is only one employee, so trivially we always get a fixed cost for any feasible schedule. Starting with equal dedication values of 1/2, both tasks finish at similar times. The task with higher effort takes two more time steps, throughout which the employee only works half time, see Figure 1 (a). This increases the completion time, compared to the optimal schedule where both dedications are 1, see Figure 1 (d). Any local operation either creates an infeasible schedule or it increases the imbalance between the two tasks. This increases the time period at the end where the employee only works half time and it makes the schedule even worse, see Figure 1 (b), (c).

RLS has a positive probability of starting in the local optimum, and no local mutation is accepted. Thus, we get:

THEOREM 8. *There is an instance with $k = 2$, only $m = 2$ tasks and just $n = 1$ employee (see Figure 1) where RLS with normalisation has an infinite expected optimization time.*

This example shows that the global mutation operator used in the (1+1) EA is important in general.

The above instance can also be generalised towards more than two tasks. Set $\text{eff}_1 = \text{eff}_2 = \cdots = \text{eff}_{m-1} = 2m$ and



(a) $x_{1,1} = 1/2, x_{1,2} = 1/2$ (b) $x_{1,1} = 1/2, x_{1,2} = 1$
(c) $x_{1,1} = 1, x_{1,2} = 1/2$ (d) $x_{1,1} = 1, x_{1,2} = 1$

Figure 1: Gantt diagrams of all feasible schedules for the example from Theorem 8 and the employee's dedication. (a) is a local optimum, (d) is the only global optimum.

$\text{eff}_m = 2m + 1$. There are no precedence constraints. Set $k = m$ and $n = 1$, that is, there is only one employee.

This yields a setting which can also become hard for the (1+1) EA. Similar to the instance from Theorem 8, setting all dedication values to $1/k = 1/m$ yields a local optimum. All other solutions are worse, as they create an imbalance between tasks—except for solutions where all dedications are equal and larger than $1/m$. In order to escape from the local optimum, all dedication values need to be changed in a single mutation. The expected running time then increases exponentially in the number of tasks.

THEOREM 9. *For every $m \in \mathbb{N}$ there is an instance with m tasks and one employee where the (1+1) EA with normalisation has expected optimisation time at least $\frac{1}{m}\left(\frac{m^2}{m+1}\right)^m$.*

We omit a formal proof due to space constraints. This result shows that global optimisation can be very hard, even if finding a solution of reasonable quality might be easy.

In particular, even though normalisation makes it easier to balance dedications, there is still a risk of non-optimal equilibria between dedications for tasks processed in parallel. This can present a major obstacle for EAs as many dedications might need to be changed in a single mutation.

6. EXPERIMENTAL ANALYSIS

This section presents an experimental analysis of our algorithm with the objective of further analysing the effects of normalisation. In order to do so, we compare our algorithm against a state-of-the-art GA [2] and a (1+1) EA without normalisation[1]. The latter was used to check whether normalisation is the main reason for the differences in the results obtained by our algorithm and the GA.

The (1+1) EA without normalisation works similarly to our algorithm with normalisation, but with an extra condition on the fitness function to consider overwork. In this case, the fitness is $f(x) = w_{pess} + over$ if the skills constraint (eq. 1) is satisfied but there is overwork. The value *over* is the total amount of overwork time spent by all employees during the project [2] and
$w_{pess} = w_{\text{cost}} \cdot 2\sum_{i=1}^{n}\sum_{j=1}^{m} s_i\text{eff}_j + w_{\text{time}} \cdot 2k\sum_{j=1}^{m}\text{eff}_j$.

This section is further divided as follows: section 6.1 presents the data sets used in the experiments; section 6.2 presents the parameters; section 6.3 presents the results in terms of hit rate (number of runs in which a feasible solution was found); and section 6.4 in terms of solution quality.

[1]Our implementation of the (1+1) EA with and without normalisation was based on the Opt4J framework [12].

6.1 Data Sets

In order to make a fair comparison against the GA, we used the same 48 instances (benchmarks 1-5) of the PSP generated by Alba and Chicano for their experimental analysis [2]. As the number of instances is high, we avoid biasing the conclusions and remove the possibility of hand-tuning the algorithm to a particular problem instance. Moreover, we can verify how the algorithms are affected by problem features such as number of employees, number of tasks, number of employees' skills and number of project demanded skills.

Benchmarks 1-3 were used to analyse the effect of varying each of three problem features (number of employees, number of tasks, and number of employees' skills) while maintaining the other features fixed. These data sets use the same salary for all employees ($10,000), so that, given a project, the cost of all solutions for this project is always the same. In this way, the ideal cost per unit of time is known and it is possible to evaluate how close a given solution is to the optimum in terms of completion time.

Benchmark 1 is composed of 4 instances varying the number of employees among 5, 10, 15 and 20. Benchmark 2 is composed of 3 instances varying the number of tasks among 10, 20 and 30. Benchmark 3 is composed of 5 instances varying the number of employees' skills among 2, 4, 6, 8 and 10 skills, which are randomly selected from a set of 10 project skills. Each task requires five different skills in this benchmark. In benchmarks 1 and 2, all employees have all necessary skills, i. e., the skills constraint (eq. 1) is always satisfied. Instances within each of the benchmarks 1 and 3 represent the same project to be developed (i. e., they have the same tasks and TPG) with the number of tasks fixed as 10. Each instance of benchmark 2 represents a different project, as the number of tasks is different. In benchmarks 2 and 3, the number of employees is fixed as 5.

Benchmarks 4 and 5 are composed of instances that represent different projects and each employee has a different salary. Each benchmark is composed of 18 instances which vary all the previous problem features. The number of employees can be 5, 10 or 15 and the number of tasks 10, 20 or 30. In benchmark 4, the total number of project skills is 10, and two ranges were considered separately for the number of employees' skills: 4-5 and 6-7. In benchmark 5 the number of project demanded skills can be 5 or 10, and the number of skills per task and employee is in the range 2-3.

6.2 Setup

For a fair comparison between our algorithm and the GA, we used the following parameters, which correspond to the parameters used previously [2]: constant for the granularity of the solution $k = 7$; $w_{\text{cost}} = 10^{-6}$; $w_{\text{time}} = 10^{-1}$; number of generations 5064 (= number of fitness evaluations considering their initial population of size 64); and the number of independent runs per problem instance 100.

6.3 Hit Rate

Table 1 shows the hit rates for all benchmarks. Our (1+1) EA with normalisation always achieved hit rate 100, i. e., all runs always found a feasible solution. The modified Wald confidence interval [1] with 95% of confidence for that is [96.83,100.00]. This is a significant improvement in comparison to the GA, which frequently presented much lower hit rates, sometimes even hit rates of zero. It is worth noting that the upper limits of the confidence intervals for hit rates of 90 or less are lower than the lower limit for hit rates of 100.

In order to check whether normalisation plays a significant role in improving the hit rates, the (1+1) EA with normalisation was compared to the (1+1) EA without normalisation. The hit rates for the latter algorithm were almost always lower, showing that normalisation is important for improving hit rates. The (1+1) EA without normalisation sometimes achieved better and sometimes worse hit rates than the GA, showing that the (1+1) EA itself can sometimes be beneficial and sometimes detrimental for the hit rates in comparison to the GA.

Another interesting observation is that our (1+1) EA with normalisation always managed to achieve hit rate of 100 independent of the problem features. The study performed by Alba and Chicano [2] revealed that their GA's hit rate varied depending on the problem features. For instance, it was lower when there were more tasks, less employees' skills or more project demanded skills. Our experiments using (1+1) EA without normalisation also present different hit rates for different instances. So, normalisation again plays an important role in making the hit rate less dependent on the problem features. In other words, normalisation helps to improve the robustness of the EA.

6.4 Solution Quality

In this section, we analyse the quality of the feasible solutions in terms of average fitness, cost and completion time. Only feasible solutions were used for computing the values analysed in this section.

Following Demšar's recommendation for comparing two algorithms over multiple data sets [7], we used Wilcoxon sign rank statistical test to compare the average fitness of the (1+1) EA with and without normalisation over all problem instances. The test shows statistically significant difference at the level of significance of 0.05 (p-value of $8.2911 \cdot 10^{-6}$). The average fitness obtained by the (1+1) EA with normalisation wins for all problem instances. So, normalisation plays a key role in improving the fitness of the solutions. It is not possible to perform this type of comparison against Alba and Chicano's GA because relevant parts of their numerical results were not presented in their paper.

Besides the better average fitness across problem instances, the variances in cost and completion time across solutions for a same problem instance were also very low when normalisation was used. The variance in cost was never more than 0.03 percent of the average cost and the variance in completion time was never more than 0.18 percent of the average completion time of the project. When normalisation was not used, the variance in cost was from 0.25–1.90 percent of the total cost and the variance in completion time was 4.65–52.19 percent of the total completion time whenever there were more than three feasible solutions.

6.4.1 Benchmarks 1-3

Table 2 shows some values descriptive of the solution quality for benchmarks 1-3. The raw completion time for each algorithm and problem instance was omitted due to space restrictions. The cost for all solutions of benchmarks 1, 3 and the first instance of benchmark 2 were \$980,000. The cost for the second and third instances of benchmark 2 were \$2,600,000 and \$2,700,000, respectively. This is expected considering the instances with the same project to be developed and the fact that all employees have the same salary.

The cost per unit of time (cost/time) for benchmarks 2 and 3 is optimal at \$50,000 (all employees working full time), as there are 5 employees and all have the same salary. Each problem instance of benchmark 1 has the optimal cost/time increased by \$50,000 in relation to the previous instance, as the number of employees is increased by 5. As we can see from table 2, our (1+1) EA with normalisation obtained near optimal solutions for all these benchmarks. For benchmarks 2 and 3, we can see that the cost/time was closer to the optimum than the other algorithms'. So, for the same project cost, our (1+1) EA with normalisation obtained solutions with lower completion time. The same happened for benchmark 1, as shown through the lower product of the number of employees by the completion time ($n \cdot \text{time}$) in table 2. As the (1+1) EA without normalisation always presented worse completion time than the GA, normalisation plays a key role in improving solution quality.

The results also reveal that the solution quality of our (1+1) EA with normalisation was less affected by variations in the number of employees, tasks and employees' skills than the other algorithms'. This is shown by its more similar product $n \cdot \text{time}$ across instances of benchmark 1, and by its more similar cost/time across instances of benchmarks 2 and 3. These values were more different across instances of a same benchmark for the (1+1) EA without normalisation. So, normalisation plays again an important role in making the EA more robust.

6.4.2 Benchmarks 4-5

Even though we cannot compare cost and completion time numerical values of our (1+1) EA with normalisation against the GA for benchmarks 4 and 5, we analyse them against the (1+1) EA without normalisation. In this case, even though our (1+1) EA with normalisation always obtained considerably better completion time (solutions took from 39.49 to 90.92 percent of the time spent by solutions generated without normalisation), it obtained slightly worse cost (from 100.17 to 103.43 percent of the cost of solutions without normalisation). Nevertheless, the slightly worse cost reflects the choice of weights w_{cost} and w_{time} in the fitness function. The fitness values produced by the (1+1) EA with normalisation were better, representing better solution quality considering the given w_{cost} and w_{time}. As future work, other weights or a multi-objective evolutionary algorithm should be tested.

7. CONCLUSIONS

We have presented novel theoretical insight into the performance of EAs for the PSP. This theory inspired improvements in the design of EAs, including normalisation of dedication values, a tailored mutation operator, and fitness functions with a strong gradient towards feasible solutions. Normalisation removes the problem of overwork and allows an EA to focus on the solution quality. It facilitates finding the right balance between dedication values for different tasks and allows employees to adapt their workload whenever other tasks are started or finished.

Runtime analyses for EAs with and without normalisation have covered easy and difficult instances, and how long it

Table 1: Hit rate out of 100 runs for the (1+1) EA without normalisation, and the GA (obtained from [2]). The hit rate for the (1+1) EA with normalisation was always 100.

Benchmark 1			Benchmark 2			Benchmark 3		
Employees	No Norm	GA	Tasks	No Norm	GA	Employees' skills	No Norm	GA
5	97	87	10	97	73	2	0	39
10	100	65	20	84	33	4	0	53
15	97	49	30	3	0	6	24	77
20	96	51	–	–	–	8	11	66
–	–	–	–	–	–	10	100	75

Tasks	Benchmark 4				Benchmark 5			
	4-5 employees' skills 5,10,15 employees		6-7 employees' skills 5,10,15 employees		5 project skills 5,10,15 employees		10 project skills 5,10,15 employees	
	No Norm	GA	No Norm	GA	No Norm	GA	No Norm	GA
10	2,0,89	94,97,97	9,100,100	84,100,97	10,49,90	98,99,100	0,0,0	61,85,85
20	0,2,17	0,6,43	0,78,11	0,76,0	0,2,67	6,9,12	0,0,0	8,1,6
30	0,0,0	0,0,0	0,6,0	0,0,0	0,0,1	0,0,0	0,0,0	0,0,0

Table 2: Quality of feasible solution for (1+1) EA with and without normalisation, and the GA (obtained from [2]). The averages were calculated considering only the runs in which a feasible solution was found. The best values are in bold. n, m and sk are the number of employees, tasks and employees' skills.

Benchmark 1					Benchmark 2				Benchmark 3			
n	Avg. $n \cdot$ time			Avg. cost/time[1]	m	Avg. cost/time			sk	Avg. cost/time		
	Norm	No Norm	GA	Norm		Norm	No Norm	GA		Norm	No Norm	GA
5	**98.04**	113.96	109.40	**49,978**	10	**49,978**	36,579	44,944	2	**49,983**	–	45,230
10	**98.06**	129.68	112.70	**99,940**	20	**49,980**	35,273	44,748	4	**49,983**	–	45,069
15	**98.07**	128.06	115.95	**149,900**	30	**49,990**	18,236	–	6	**49,980**	38,762	44,651
20	**98.08**	129.54	117.60	**199,830**	–	–	–	–	8	**49,979**	36,518	44,617
–	–	–	–	–	–	–	–	–	10	**49,978**	37,182	44,427

[1] Cost per time for benchmark 1 was omitted for the (1+1) EA without normalisation and the GA due to space constraints.

takes to find feasible solutions. For linear schedules both the (1+1) EA and RLS are effective. However, despite using normalisation they still struggle to escape from local optima where many dedication values form an equilibrium.

Our empirical study confirmed that normalisation is very effective in improving the hit rate, the solution quality and making the EA more robust.

Future work includes experimental analysis of the runtime or generation-to-success distributions [9, 3], use of other weights for the fitness function and other EAs such as multi-objective algorithms.

Acknowledgments: The authors thank Enrique Alba for providing instances from [2] and the anonymous reviewers for their constructive comments. This research was supported by EPSRC grant EP/D052785/1.

8. REFERENCES

[1] A. Agresti and B. Coull. Approximate is better than "exact" for interval estimation of binomial proportions. *The American Statistician*, 52:119–126, 1998.

[2] E. Alba and J. F. Chicano. Software project management with GAs. *Information Sciences*, 177:2380–2401, 2007.

[3] D. Barrero, B. Castaño, M. R-Moreno, and D. Camacho. Statistical distribution of generation-to-success in GP: Application to model accumulated success probability. In *EuroDP '11*, pages 155–166. Springer, 2011.

[4] C. K. Chang, M. J. Christensen, and T. Zhang. Genetic algorithms for project management. *Annals of Software Engineering*, 11:107–139, 2001.

[5] C. K. Chang, H. Jiang, Y. Di, D. Zhu, and Y. Ge. Time-line based model for software project scheduling with genetic algorithms. *Information and Software Technology*, 50(11):1142 – 1154, 2008.

[6] C. Chao. *SPMNET: A New Methodology for Software Management.* PhD thesis, The University of Illinois at Chicago, 1995.

[7] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *JMLR*, 7:1–30, 2006.

[8] B. Doerr, T. Jansen, D. Sudholt, C. Winzen, and C. Zarges. Optimizing monotone functions can be difficult. In *PPSN '10*, pages 42–51. Springer, 2010.

[9] H. Hoos and T. Stützle. Local search algorithms for SAT: An empirical evaluation. *Journal of Automated Reasoning*, 24(4):421–481, 2000.

[10] D. Johannsen. *Random Combinatorial Structures and Randomized Search Heuristics.* PhD thesis, Universität des Saarlandes, Saarbrücken, Germany and the Max-Planck-Institut für Informatik, 2010.

[11] P. Kapur, A. Ngo-The, G. Ruhe, and A. Smith. Optimized staffing for product releases and its application at Chartwell Technology. *Journal of Software Maintenance and Evolution: Research and Practice*, 20(5):365–386, 2008.

[12] M. Lukasiewycz, M. Glaß, F. Reimann, and J. Teich. Opt4J - A Modular Framework for Meta-heuristic Optimization. In *Proc. of GECCO '11*, pages 1723–1730. ACM, 2011.

[13] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[14] F. Neumann and C. Witt. *Bioinspired Computation in Combinatorial Optimization – Algorithms and Their Computational Complexity.* Springer, 2010.

[15] P. S. Oliveto, J. He, and X. Yao. Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results. *Int'l Journal of Automation and Computing*, 4(3):281–293, 2007.