Running Time Analysis for the Project Scheduling Problem

January 19, 2012

1 Preliminaries

We consider evolutionary algorithms for the project scheduling problem as defined by Alba and Chicano (2007).

1.1 Problem Definition

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.

The goal is to assign employees to tasks as to minimize the completion time as well as the costs for the project. Employees can work on several tasks simultaneously, as indicated by their *dedication* to certain tasks. The dedication can be regarded as the fraction of their time devoted to a particular task. The completion time is the time the project is completed. It is computed with a simple iterative algorithm that takes into account the task precedence graph, the effort for each task, and the dedication for all employees.

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 $0\%, 10\%, \ldots, 100\%$. We can think of k being a constant, i. e., independent of the number of employees and tasks. More detailed schedules would not be feasible in practice anyway.

Employees can only work on a task t_j if they have all the skills to do the task. More formally, for each employee i we require

$$\text{req}_j \subseteq \text{skill}_i.$$

If this is not the case, we just treat the corresponding dedication value as if it was 0. Accordingly, we define

$$\bar{x}_{i,j} := \begin{cases} x_{i,j} & \text{if } \text{req}_j \subseteq \text{skill}_i \\ 0 & \text{otherwise.} \end{cases}$$

Alba and Chicano (2007) 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 both part-time jobs as well as the willingness for working overtime. Alternatively, one could introduce the *productivity* of an employee as a general indicator for his performance. The productivity of a part-time employee could then be decreased accordingly. It would also be feasible to associate different productivities with each skill as to reflect how good an employee is at using a particular skill. This was done, for instance, in Kapur, Ngo-The, Ruhe, and Smith (2008). However, for the sake of simplicity we refrain from using a maximum dedication or productivity in this work and instead leave these aspects for future work.

Alba and Chicano (2007) considered the problem of *overtime*: when tasks are executed in parallel, the total dedication of employees can exceed 1. They introduce a penalty that worsens the fitness accordingly.

Here, we follow a different approach. Instead of penalizing overtime, we normalise the dedication to eliminate overtime. 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.

In particular, it allows for much more fine-tuned schedules as employees are supposed to adapt their dedication while working on some task t_i if other tasks they are assigned to start or finish before completing t_i. For instance, assume there are two tasks t_1, t_2 suitable for employee e_1. Due to precedence constraints, t_2 can start only just before t_1 is finished. If we only have a single dedication value $x_{i,j}$ for each employee/task-combination without normalisation, we either have e_1 working overtime (if $x_{1,1} + x_{1,2} > 1$) for the time both t_1 and t_2 need to be pursued. Or we are wasting potential (if $x_{1,1} + x_{1,2} \leq 1$) as e_1 is not allowed to devote 100% of her/his time on a single task, even when she/he only needs to work on a single task for most of the time.

Observe that due to normalisation, the genotype of dedications $x_{i,j}$ does not directly represent the final schedule. The precise dedication of an employee to a task depends on other tasks pursued at the same time. The final schedule or *phenotype* can be computed within a regular fitness evaluation.

1.2 Evaluating Costs and Completion Time

For the sake of notational convenience, we define $0/0 := 0$. The following algorithm computes both cost and completion time according to a(n implicit) Gantt diagram. It first tests whether the genotype is feasible in that every task can be completed in finite time. If it cannot, a large penalty term is returned.

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).

Before giving the pseudocode, we summarise the main ideas. The algorithm iteratively constructs the schedule. We check which tasks can be active at the current point of time. The normalised dedication for all suitable employees is computed for all these tasks. Then we determine the earliest point of time t at which a task is finished. All finished tasks are being marked as finished, 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. This can lead to piecewise evaluations of particular tasks. Note that this is necessary as the total dedication of employees to this task may change as tasks are finished and/or new tasks are started. Along the way, all completion times and costs in all iterations are added up.

Algorithm 1 Evaluate(cost, completiontime, TPG)

1: Let undt = 0.
2: **for** all tasks t_j **do**
3: **if** $\sum_{i=1}^{n} \bar{x}_{i,j} = 0$ **then**
4: undt = undt + 1.
5: **end if**
6: **end for**
7: **if** undt > 0 **then**
8: Output $(\text{undt} \cdot 2\sum_{i=1}^{n}\sum_{j=1}^{m} s_i\text{eff}_j, \text{undt} \cdot 2k\sum_{j=1}^{m}\text{eff}_j)$ and stop.
9: **end if**
10: **while** TPG $\neq \emptyset$ **do**
11: Let V' be the set of all unfinished tasks without incoming edges in TPG.
12: **if** $V' = \emptyset$ **then**
13: Output "Problem instance is not solvable!" and stop.
14: **end if**
15: **for** all tasks t_j in V' **do**
16: **for** all employees e_i **do**
17: Let $d_{i,j} := \frac{\bar{x}_{i,j}}{\max\left(1, \sum_{t_\ell \in V'} \bar{x}_{i,\ell}\right)}$.
18: **end for**
19: Compute the total dedication $d_j := \sum_{i=1}^{n} d_{i,j}$.
20: **end for**
21: Let $t := \min_j(\text{eff}_j/d_j)$.
22: Let cost := cost $+ t\sum_{i=1}^{n} s_i \sum_{j=1}^{m} d_{i,j}$.
23: Let completiontime := completiontime $+ t$.
24: **for** all tasks t_j in V' **do**
25: Let $\text{eff}_j := \text{eff}_j - t \cdot d_j$.
26: **if** $\text{eff}_j = 0$ **then**
27: Mark t_j as finished and remove it and its incident edges from TPG.
28: **end if**
29: **end for**
30: **end while**
31: Output (cost, completiontime) and stop.

In case there is a task for which no employee has the necessary skills, a vector $(\text{undt} \cdot 2k\sum_{j=1}^{m}\text{eff}_j, \text{undt} \cdot 2\sum_{i=1}^{n}\sum_{j=1}^{m} s_i\text{eff}_j)$ is returned. The same holds for solutions where no skilled employee has a positive dedication. This vector is worse than any feasible solution in both completion time and cost. The completion time for any feasible schedule is bounded by $k\sum_{j=1}^{m}\text{eff}_j$. This corresponds to a schedule where all tasks are processed sequentially and for each task there is only one employee working on it, with the minimum dedication of $1/k$. Paying all employees their full salary during the time needed for one employee to work full time yields an upper bound of $\sum_{i=1}^{n}\sum_{j=1}^{m} s_i$ on the costs. Also, any decrease in the number of missing skills strictly decreases both components. This gives strong hints for any search algorithm to reach the feasible region.

The computational complexity of the genotype-phenotype mapping and the fitness evaluation can be bounded by $O(nm^2)$. This is quite efficient, given that the input size is of order $\Theta(nm)$. The feasibility check can be implemented in time $O(nm)$.

The main argument for the remainder of the procedure is that in one iteration of the while loop all steps, excluding Lines 11 and 27 that deal with the TPG, take time $O(|V'| \cdot n) = O(mn)$. As each iteration removes at least one out of m tasks, there can only be $O(m)$ iterations until the evaluation stops. In addition, using appropriate data structures, all executions of Line 27 can be executed in a time proportional to the number of edges in the task precedence graph. This holds since each edge is touched only once. The number of edges is bounded by m^2. Also, while removing edges, counters can be used and updated for the in-degree of each vertex. The counter maintenance causes only constant overhead. This allows to determine the set V' in time $O(m)$. The total effort spent in lines 11 and 27 is hence bounded by $O(m^2)$.

Note that when no normalisation is used an evaluation of cost and completion time can be done more efficiently than in time $O(nm^2)$ if the task precedence graph is sparse. This is because finishing a task does not require recomputing dedication values. In other words, it is not necessary to have piecewise evaluations of tasks. However, Alba and Chicano (2007) also need to compute penalties for overtime and this again requires piecewise task evaluations. Their approach thus seems to require the same computational complexity as the above procedure.

1.3 Representation and Algorithms

Alba and Chicano (2007) used a binary encoding to represent the $x_{i,j}$. Instead, we work directly on 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. This resolves the problem of Hamming cliffs experienced in their approach.

For single-objective optimization, we might get a fitness function f by a weighted combination of cost and completion time. The following describes a variant of the (1+1) EA for our search space.

Algorithm 2 (1+1) EA for project scheduling

1: **repeat**
2: Create x' by copying x.
3: **for** each $1 \le i \le n, 1 \le j \le m$ **do**
4: With probability $1/(nm)$ choose $x'_{i,j}$ uniformly at random from $\{0/k, 1/k, \dots, k/k\} \setminus \{x_{i,j}\}$.
5: **end for**
6: **if** $f(x) \le f(x')$ **then**
7: Let $x := x'$.
8: **end if**
9: **until** happy

Note that the expected number of components changed in one iteration equals 1. As mutation is a global operator, larger changes are possible as well.

We might also consider (global) SEMO, applied to the bi-objective function $(\text{cost}, \text{completiontime})$. SEMO stores all non-dominated solutions in its population. In every generation it generates a new solution by choosing a parent uniformly at random, applying mutation, inserting the offspring into the population, and finally removing all dominated solutions from the population.

For both algorithms and for both formulations we might need to implement a mechanism that guides the algorithms away from infeasible solutions. After all, a returned vector of (∞, ∞) does not give any direction towards feasible values.

Another idea would be to consider *memetic* evolutionary algorithms, i. e., hybridisations with local search. Local search could increase or decrease components $x_{i,j}$ of a solution. This way, it can repair infeasible solutions or refine feasible solutions. The latter might allow for larger values of k, i. e., a more fine-grained representation.

2 Equal Salaries

Let us look at some easy special cases in order to understand what makes the problem difficult. In this section we assume that all salaries are equal. This means that the cost is always fixed, so the task is to minimize the completion time.

2.1 Optimal Completion Times with Normalisation

In every feasible solution for each task the team has the required skills for the task. This also holds if more employees receive a positive dedication for the task. New employees can immediately join in working on a task, regardless of their skills. The following theorem is an immediate conclusion.

Theorem 1. *For every solvable instance, the completion time is minimal if in the corresponding phenotype all employees always work full time. Then the completion time is* $1/n \cdot \sum_{j=1}^{m} \text{eff}_j$.

If normalisation is used, a sufficient condition for minimality is that all dedication values are 1.

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

We also get the following statement, which might prove useful later on.

Lemma 2. *Consider a feasible solution x whose schedule ends with a unique task j being active. If $x_{i,j} < 1$ for some $1 \leq i \leq n$ then the completion time for x is not minimal.*

Proof. Employee i does not work full time during a non-zero time span at the end of the schedule. Hence the completion time is strictly larger than $1/n \cdot \sum_{j=1}^{m} \text{eff}_j$. Along with Theorem 1, the claim follows. □

2.2 A General Lower Bound

Theorem 3. *For any instance with n employees and m tasks such that $nm > 1$ and there is only a single global optimum, the expected optimization time of both RLS and the (1+1) EA is at least $\Omega(knm \log(nm))$.*

More generally, if there are $N > 1$ entries in the dedication matrix for which only one configuration leads to a global optimum, the expected optimization time for both algorithms is at least $\Omega(kN \log N)$.

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 Chernoff bounds, 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

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

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 second claim follows from the same proof when replacing nm with N, ignoring all other entries, and redefining *bad* according to the configuration mentioned in the statement. □

2.3 Time to Feasibility

Theorem 4. *Consider RLS and the (1+1) EA for any single-objective or multi-objective fitness function that is Pareto-compliant (i. e. (x_1, x_2) replaces (y_1, y_2) if $x_1 \leq y_1$ and $x_2 \leq y_2$).*

For any initialization, the expected time until any feasible schedule is found on any solvable instance is at most $enmH(n)$ for the (1+1) EA and at most $nmH(m)$ for RLS, where $H(m) = \sum_{i=1}^{m} 1/i \leq (\ln m)+1$ denotes the m-th harmonic number. There are examples where this bound is tight, provided $k \leq m^{1-\varepsilon}$ for some constant $\varepsilon > 0$.

Proof. Note that, as long as the current solution is infeasible, the fitness is uniquely determined by the number undt of tasks where no skilled employee is assigned. If the number is currently i then there are at least i entries in the matrix where a mutation of this entry (and no other entries) will decrease the number of unassigned tasks. The probability of making such a mutation is at least $i/(nm) \cdot (1-1/(nm))^{nm-1} \geq i/(enm)$ for the (1+1) EA and at least $i/(nm)$ for RLS. The expected waiting time for a decrease is at most enm/i and nm/i, respectively. Summing up waiting times for all possible values yields an upper bound of

$$\sum_{i=1}^{m} \frac{enm}{i} = enm \cdot H(m)$$

for the (1+1) EA and a bound of

$$\sum_{i=1}^{m} \frac{nm}{i} = nm \cdot H(m)$$

for RLS.

The bound is asymptotically tight in general. Consider the instance where the tasks $t_1, \dots, t_m$ have to be performed sequentially and for each task there is only one employee with the required skills. So, feasibility only depends on m specific entries in the matrix. Arguing as in the proof of Theorem 3, with high probability only $m/(2k)$ of these entries are greater than 0 in the initial dedication matrix. Repeating the line of thought from there yields a lower bound of $\Omega(nm \log(m/2k)) = \Omega(nm \log(m^\varepsilon/2)) = \Omega(nm \log(n))$, using the assumption on k. □

2.4 Linear Schedules

Let us consider schedules with a "linear" structure: the task precedence graph is a chain of m vertices, or any directed acyclic graph that contains such a chain. In this case the tasks, w. l. o. g. $t_1, \dots, t_m$ in this order, have to be done sequentially.

Note that normalisation is never actually applied as at each time only one task is processed. In other words, the issue of whether normalisation is used or not is irrelevant for linear schedules.

The completion time is minimized if all employees that are qualified for a task show a maximum dedication for this task. It is easy to derive matching bounds for RLS.

2.4.1 Analysis of RLS

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 is of order $\Theta(knm \ln(nm))$.

Proof. The lower bound follows from Theorem 3.

By Theorem 4 we know that a feasible solution is found in time $O(nm \ln(n))$. 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.

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 $\ell \le nm$ bad entries, yields the upper bound

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

□

2.4.2 Analysis of the (1+1) EA

It is not so easy to do the same for the (1+1) EA.

[Dirk]: This part is still under construction.

Theorem 6 (Variable drift, upper bound Johannsen (2010))**.** *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 $\mathrm{E}\big(X^{(t)} - X^{(t+1)} \mid X^{(t)}\big) \ge h(X^{(t)})$ holds for all $t < T$. Then,*

$$\mathrm{E}\Big(T \mid X^{(0)}\Big) \le \frac{x_{\min}}{h(x_{\min})} + \int_{x_{\min}}^{X^{(0)}} \frac{1}{h(x)} \, \mathrm{d}x$$

From this we easily get a drift theorem for shifted variables. Instead of a fixed target 0, the target in the following process is some value $x^* \ge 0$.

Corollary 7 (Variable drift with translation)**.** *Let $\{X^{(t)}\}_{t\in\mathbb{N}}$ be a sequence of random variables over a finite state space $S \subseteq [x^*, \infty[$ for some $x^* \geq 0$ and let $x_{\min} := \min\{x \in S \colon x > x^*\}$. Furthermore, let T be the random variable that denotes the first point in time $t \in \mathbb{N}$ for which $X^{(t)} = x^*$. Suppose that there exists a continuous and monotone increasing function $h \colon \mathbb{R}_0^+ \to \mathbb{R}^+$ such that $\mathrm{E}\big(X^{(t)} - X^{(t+1)} \mid X^{(t)}\big) \geq h(X^{(t)})$ holds for all $t < T$. Then,*

$$\mathrm{E}\Big(T \mid X^{(0)}\Big) \leq \frac{x_{\min} - x^*}{h(x_{\min})} + \int_{x_{\min}}^{X^{(0)}} \frac{1}{h(x)}\,\mathrm{d}x$$

Proof. Define $\{Y^{(t)}\}_{t\in\mathbb{N}}$ by $Y^{(t)} := X^{(t)} - x^*$. Note that T, as defined in the corollary, also equals the random variable denoting the first point in time $t \in \mathbb{N}$ for which $Y^{(t)} = 0$. Define the function $h'(x) := h(x - x^*)$ and $x'_{\min} := x_{\min} - x^*$ where h and $x_{\min}$ are defined as in the corollary. Note that h' is monotone increasing on $[0, \infty[$ if and only if h is monotone increasing on $[x^*, \infty[$.

We have that $\mathrm{E}\big(Y^{(t)} - Y^{(t+1)} \mid Y^{(t)}\big) \geq h'(Y^{(t)})$ is equivalent to $\mathrm{E}\big(X^{(t)} - X^{(t+1)} \mid X^{(t)}\big) \geq h(X^{(t)})$ for all $t < T$. Applying Theorem 6 to $\{Y^{(t)}\}_{t\in\mathbb{N}}$ we get

$$\mathrm{E}\Big(T \mid Y^{(0)}\Big) \leq \frac{x'_{\min}}{h'(x'_{\min})} + \int_{x'_{\min}}^{Y^{(0)}} \frac{1}{h'(x)}\,\mathrm{d}x = \frac{x_{\min} - x^*}{h(x_{\min})} + \int_{x_{\min}}^{X^{(0)}} \frac{1}{h(x)}\,\mathrm{d}x.$$

As $\mathrm{E}\big(T \mid Y^{(0)}\big) = \mathrm{E}\big(T \mid X^{(0)}\big)$, this proves the claim. □

Theorem 8. *Consider an instance with $m = 1$ tasks and $k = 1$. The expected optimization time of the (1+1) EA is at most $en \ln n + O(n)$.*

> **[Dirk]:** The result is not interesting by itself as the problem is equivalent to OneMax. We get the same bound from results for OneMax. The purpose of this analysis is to use an analysis that can be generalised to more complex settings.

Proof. We can assume w. l. o. g. that the task's effort is 1. Assume that currently $i \geq 1$ employees are assigned to the task. The current completion time is therefore $1/i$. The probability of starting with $i = 0$ is 2^{-n} and then the expected time to reach a feasible solution is $O(1)$. By the law of total expectation, the unconditional expectation differs from the one conditional on $i \geq 1$ only by an additive term $O(2^{-n})$, which is absorbed in the $+O(n)$ term stated in the theorem.

Let us estimate the expected change of the completion time in one iteration. The completion time at the current time t is denoted CT_t. The probability of changing exactly one dedication value is $1/n \cdot (1-1/n)^{n-1} \geq 1/(en)$. There are $n-i$ entries which lead to a decreased completion time of $1/(i+1)$. The difference to the current completion time is $1/i - 1/(i+1) = 1/(i(i+1))$. We pessimistically assume that all other operations do not change the completion time. The expected difference is therefore at least

$$\mathrm{E}(\mathrm{CT}_t - \mathrm{CT}_{t+1} \mid \mathrm{CT}_t = 1/i > 1/n) \geq \frac{n-i}{en} \cdot \frac{1}{i(i+1)}.$$

Define

$$h(x) := \frac{n - 1/x}{en} \cdot \frac{1}{1/x(1/x+1)}.$$

then we get

$$\mathrm{E}(\mathrm{CT}_t - \mathrm{CT}_{t+1} \mid \mathrm{CT}_t > 1/n) \geq h(\mathrm{CT}_t).$$

It is easy to see that h is monotone increasing. Applying Corollary 7 with $x^* := 1/n$ and $x_{\min} := 1/(n-1)$, the expected time until the completion time is decreased to $x^* = 1/n$ is bounded by

$$\begin{aligned}
&\frac{\frac{1}{n-1} - \frac{1}{n}}{\frac{1}{en} \cdot \frac{1}{n(n-1)}} + \int_{1/(n-1)}^{1} \frac{en}{n - 1/x} \cdot 1/x(1/x+1)\,\mathrm{d}x \\
&= en + en \int_{1/(n-1)}^{1} \frac{1}{n - 1/x} \cdot 1/x(1/x+1)\,\mathrm{d}x \\
&= en \left(1 + \ln(n-1) \cdot \frac{n+2}{n}\right) \\
&= en \ln n + O(n).
\end{aligned}$$

□

[Dirk]: To do: generalise this towards $k > 1$.

[Dirk]: To do: generalise this towards $m > 1$ tasks that have to be executed sequentially.

Theorem 9. *Consider the (1+1) EA on any schedule where all salaries are equal, all tasks have to be executed sequentially, and* $\text{eff}_j \geq 1$ *for all* $1 \leq j \leq m$. *Then the expected optimisation time of the (1+1) EA is bounded by* $ek^3n^3m\sum_{j=1}^{m}\text{eff}_j$.

If for each task there are at most q *employees with the necessary skills, we get an upper bound of* $ek^3q^2nm\sum_{j=1}^{m}\text{eff}_j$.

Proof. We only prove the second claim as it implies the first one for $q := n$. 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 $1/(kq^2)$.

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

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

The probability of making this mutation is at least $1/(eknm)$. Then the expected decrease of the completion time is at least $1/(ekq^2nm)$. The upper bound then follows in a straightforward manner from additive drift analysis results. □

The bound from Theorem 9 is quite crude. But it shows that the expected time until the (1+1) EA finds a schedule with minimum completion time is polynomially bounded in k, n, and m.

We do not believe that the (1+1) EA is much more inefficient than RLS, so we conjecture that the upper bound $O(knm\ln(nm))$ for RLS from Theorem 5 also applies to the (1+1) EA. Proving this, however, turns out to be very challenging. Mutations flipping several bits can undo previous "good" mutations. This is because the different tasks are weighted and increasing the dedication for a long task can imply that a mutation is accepted, even when several dedication values on short tasks are decreased. The Hamming distance to the global optimum can increase. Furthermore, increments or decrements of the completion time not only depend on the weight of a task, but also on the total dedication of the other employees. If the total dedication is low, changes have a large effect. If it is high, changes have a small effect. This is difficult to handle in an analysis.

It is easier to analyse the situation where $k = 1$, i. e., only dedication values 0 and 1 are allowed. It is not hard to see that the resulting fitness function is *monotone* in a sense that flipping only 0-bits to 1 and not flipping any 1-bit to 0 results in a strict fitness improvement. By Doerr, Jansen, Sudholt, Winzen, and Zarges (2010) the expected time of the (1+1) EA with mutation probability $1/(nm)$ is bounded by $O((nm)^{3/2})$.

Theorem 10. *In the setting of Theorem 9, if* $k = 1$ *then the expected optimisation time of the (1+1) EA is* $O((nm)^{3/2})$.

For any mutation probability $c/(nm)$ for a constant $0 < c < 1$ the expected time is even bounded by $O(nm\log(nm))$. The last bound is asymptotically tight, as shown by a general lower bound for functions with a unique global optimum, see (Sudholt, 2010, Theorem 5) and an extended version thereof (Sudholt, 2011, Theorem 12).

2.5 Difficult Instances

For "non-linear" schedules, i. e., schedules with more loose precedence constraints, the behaviour of RLS and the (1+1) EA might be different.

It turns out that even on very simple problem instances, RLS can get stuck. The following example instance shows that even when the all-ones matrix is the global optimum, it might be impossible to get there by only local mutations.

The instances contains two tasks such that one task takes a slightly larger effort than the other. Starting with equal dedication values of 1/2, changing only a single component makes the schedule

unbalanced: one task takes significantly longer than the other. As the dedication for the longer task is still 1/2, the employee only works half-time at the end of the project. This increases the completion time.

Theorem 11. *There is an instance with $k = 2$, $m = 2$ tasks and $n = 1$ employee where RLS has an infinite expected optimization time.*

Proof. As there is only one employee, the costs are equal for all feasible solutions. We therefore focus on the completion time.

Define $\text{eff}_1 = 1$ and $\text{eff}_2 = 5/4$ and let the task precedence graph be empty. Assume both tasks require one skill and that the employee has this skill. There is a positive probability that RLS starts with $x_{1,1} = x_{1,2} = 1/2$. The completion time for this schedule is $\text{eff}_2/(1/2) = 5/2 = 2.5$. Table 1 shows the completion times for all 9 possible schedules.

> **[Dirk]:** The value 5/4 is somewhat arbitrary. Any value $1 + \varepsilon$ would work if $\varepsilon > 0$ is not too small.

	$x_{1,2} = 0$	$x_{1,2} = 1/2$	$x_{1,2} = 1$
$x_{1,1} = 0$	18	9	9
$x_{1,1} = 1/2$	9	2.5	2.625
$x_{1,1} = 1$	9	3	2.25

Table 1: Completion times for all possible schedules. The schedule $x_{1,1} = x_{1,2} = 1/2$ in the center is a local optimum. The schedule $x_{1,1} = x_{1,2} = 1$ is a global optimum. The costs are the same for all feasible schedules.

Every schedule that contains 0-entries has a worse fitness. The global optimum $x_{1,1} = x_{1,2} = 1$ has completion time $9/4 = 2.25$.

The solution $x_{1,1} = 1, x_{1,2} = 1/2$ has completion time 3: the first task is finished after 3/2 time steps. An effort of 1/2 has been accomplished for the second task and its remaining effort is 3/4. So, further 3/4 steps are needed to complete the second task.

The last solution $x_{1,1} = 1/2, x_{1,2} = 1$ has the second task finishing first, after $5/4 \cdot 3/2 = 15/8$ time steps. An effort of $15/8 \cdot 1/3 = 5/8$ has been completed for the first task and 3/8 remains. The remaining time to complete the first task is 3/4, leading to a completion time of $15/8 + 3/4 = 21/8 = 2.625$.

As $3 > 2.5$ and $2.625 > 2.5$, no improvement can be reached by changing only one entry in the dedication matrix. The expected time until the global optimum is found is therefore infinite. □

The example is a smallest possible example with regard to the size of the search space. Every instance with $k = 1$, $n = 1$, and $m \in \mathbb{N}$ is solvable by local operations. The same holds for every instance with $m = 1$ and $k, n \in \mathbb{N}$.

Theorem 12. *There is a family of instances parametrized by the number of jobs $m \in \mathbb{N}$ and an initial configuration for the (1+1) EA such that its expected optimisation time is at least $(nm)^{nm}$.*

Proof. Consider the following instance. Set $\text{eff}_1 = \text{eff}_2 = \cdots = \text{eff}_{m-1} = 1$ and $\text{eff}_m = 1 + \varepsilon$ for some $0 < \varepsilon \leq 1/(3m)$. Let the task precedence graph be empty, that is, there are no precedence constraints. Set $k = m$ and $n = 1$, that is, there is only one employee.

The global optimum is to set all dedications to 1, leading to a completion time of $\sum_{j=1}^{m} \text{eff}_j = m + \varepsilon$. The solution where $d_{1,j} = 1/m$ for all $1 \leq j \leq m$ leads to a completion time of $m + m\varepsilon$; after m steps the first $m - 1$ tasks are completed and then a remaining effort of ε for the last one is done with dedication $1/m$.

We claim that every other solution having at least one dedication value $1/m$ has a strictly worse completion time. If the (1+1) EA starts with all dedication values set to $1/m$ then in order to reach a better solution, all entries need to be changed in one mutation. The probability of mutating all entries is $(nm)^{-nm}$. The expected time until this happens is $(nm)^{nm}$. This proves the theorem.

For proving the claim, assume a solution with ℓ dedication entries equal to $1/m$. Temporarily assume that $\varepsilon = 0$, corresponding to all jobs having the same effort. As $k = m$, the next higher dedication value

is twice as large. All tasks with a higher dedication are finished within at most half the time of the ℓ considered tasks. Therefore, there will be a time span of at least OPT /2 where only the latter ℓ tasks will be executed. Considering $\varepsilon > 0$ this time bound is still at least OPT $/(2+2\varepsilon)$. During this time, the employee only works with total dedication ℓ/m, i. e., he/she is idle during a $(m-\ell)/m \geq 1/m$-fraction of the time. This amounts to a total idle time of at least OPT $/(2+2\varepsilon) \cdot 1/m \geq 1/(2+2\varepsilon)$. As the total workload that needs to be completed is $m+\varepsilon$, the completion time is at least

$$m + \varepsilon + \frac{1}{2+2\varepsilon} \geq m + \varepsilon + \frac{3}{4} > m + m\varepsilon.$$

□

This result shows that global optimisation can be very hard. In this example finding a solution of reasonable quality is fairly easy. But there can be a large distance between the global optimum and a local optimum. This makes global optimisation a very challenging task.

The result can also be extended towards more coarse granularity, i. e., $k < m$. If we take the instance from the proof of Theorem 12 and simply add new jobs that have to be executed sequentially before all existing ones, we get an instance with the same characteristics but an arbitrary number of tasks. The following theorem shows that arbitrary polynomial running times can be enforced.

Theorem 13. *For arbitrary $m \geq 2$ and every constant $k \in \mathbb{N}$ there is an instance and an initial configuration for the (1+1) EA such that its expected optimisation time is of order $\Theta((nm)^k)$.*

The upper bound follows from repeating previous considerations of infeasible solutions and linear schedules, applied to the "new" tasks. Note that old and new tasks can be treated independently due to the precedence constraints. For the "old" tasks the optimal configuration can be reached by a lucky mutation. The probability of such a direct hit is $1/(e(knm)^k) = \Omega((nm)^k)$, using that k is constant.

3 Minimising Costs

3.1 Disregarding Completion Times

If we set the weight for the completion time to 0, we are only minimising the costs. Then the task is to find the cheapest qualified employees. If no two employees have the same salary, there is a unique global optimum for the cost where each task is assigned only to the cheapest employee that is qualified for the task. We present a general upper bound on the time until the (1+1) EA finds such a solution.

Theorem 14. *On any instance with weight 0 for the completion time, $k = 1$, and only one task (?) the (1+1) EA has expected optimisation time at most $ek^2n^2m^2 \cdot \sum_{i=1}^{n}\sum_{j=1}^{m} s_i\mathrm{eff}_j/s_\Delta + enmH(m)$ where $s_\Delta := \max\{|s_i - s_j| \mid 1 \leq i,j \leq n, |s_i - s_j| > 0\}$ is the smallest non-zero difference in salaries.*

Proof. By Theorem 4 the (1+1) EA reaches a feasible solution in expected time $enmH(m)$. For the remainder of the run, we claim that for any non-optimal solution the expected decrease of the costs is at least $s_\Delta/(ek^2n^2m^2)$. As the cost is always bounded by $\sum_{i=1}^{n}\sum_{j=1}^{m} s_i\mathrm{eff}_j$, this proves the claim.

[Dirk]: to be continued ...

□

3.2 Linear Schedules

For linear schedules, the cost for any feasible solution x is given by

$$\sum_{i=1}^{n}\sum_{j=1}^{m} s_i \cdot x_{i,j} \cdot \frac{\mathrm{eff}_j}{\sum_{\ell=1}^{n} x_{\ell,j}}.$$

[Dirk]: to be continued ...

4 Conclusions

References

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

B. Doerr, T. Jansen, D. Sudholt, C. Winzen, and C. Zarges. Optimizing monotone functions can be difficult. In *11th International Conference on Parallel Problem Solving from Nature (PPSN 2010)*, volume 6238 of *LNCS*, pages 42–51. Springer, 2010.

D. Johannsen. *Random Combinatorial Structures and Randomized Search Heuristics*. PhD thesis, Universitt des Saarlandes, Saarbrcken, Germany and the Max-Planck-Institut fr Informatik, 2010.

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.

D. Sudholt. General lower bounds for the running time of evolutionary algorithms. In *11th International Conference on Parallel Problem Solving from Nature (PPSN 2010)*, volume 6238 of *LNCS*, pages 124–133. Springer, 2010.

D. Sudholt. A new method for lower bounds on the running time of evolutionary algorithms. Technical report, 2011. URL `http://arxiv.org/abs/1109.1504`.