Contents lists available at ScienceDirect

Computers & Industrial Engineering

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

A polynomial time repeated cuts algorithm for the time cost tradeoff problem: The linear and convex crashing cost deadline problem

Dorit S. Hochbaum [*]

Department of Industrial Engineering and Operations Research, University of California, Berkeley, CA 94720, United States

ARTICLE INFO

Article history:
Received 24 June 2015
Received in revised form 14 January 2016
Accepted 22 February 2016
Available online 3 March 2016

Keywords:
Project management
Time–cost tradeoff
Minimum cut

ABSTRACT

The time cost tradeoff problem in project management, TCTP, is to achieve a given deadline on the project completion time by expediting the normal durations of activities at the cheapest cost possible. The linear TCTP, in which the expediting costs of each activity are linear, as a function of the number of time periods reduced, can be solved using linear programming. We present here an algorithm that solves the linear or convex costs TCTP that runs in polynomial time and calls only for a minimum s, t-cut routine at each iteration. This repeated cuts algorithm is related to the non-polynomial time algorithm by Phillips and Dessouky (1977), the PD-algorithm. The PD-algorithm reduces the project duration, at each iteration, by one time unit, at a minimum cost. The choice of the activities to expedite, in order to reduce the project duration by one unit, is determined by a solution to a minimum s, t-cut in a respective graph.

We present here previously unknown properties of the PD-algorithm, and a new concept of cut-decomposition. These properties are used in devising the repeated cuts algorithm based on scaling. The repeated cuts algorithm solves in polynomial time, the linear as well as the convex TCTP. The algorithm solves the TCTP problem in polynomial time even when the durations and/or the target deadline are not necessarily integers.

1. Introduction

The time–cost tradeoff problem, TCTP, is fundamental in project management and scheduling with precedence constraints. A project consists of activities with prescribed normal durations that must be scheduled subject to precedence constraints. The earliest finish time of the project, also referred to as *makespan* or *project duration*, is the finish time of the project such that all activities are performed subject to precedence constraints, and each activity's duration is the normal duration. A project manager is often confronted with having to meet a certain deadline that is earlier than the normal project duration. Such target deadline may be met by reducing activity durations by assigning extra labor to project activities, in the form of overtime or hiring, or by assigning other types of additional resources such as materials or equipment. The reduction of an activity duration is called *expediting*, or *crashing*, and comes at an increased cost. The decision to reduce the project duration requires the determination of the optimal tradeoff between time and cost. That is, the decision on which activities should have their durations reduced, in consideration of their expediting costs, and by how much, so as to meet the target project duration. There is a large body of literature on project scheduling, scheduling with precedence constraints, and on crashing in the presence of costs for assigning additional resources to specific activities.

TCTP problems vary according to the type of the crashing cost functions: At the normal duration the cost is zero. As the duration of an activity decreases, the cost goes up. If the crashing cost functions are linear or convex, the problem is polynomial time solvable. This follows from the algorithm of Hochbaum and Shanthikumar (1990), that solves a convex separable optimization problem on constraints, with totally unimodular coefficients matrix, in polynomial time. (The standard formulation of TCTP has constraints with totally unimodular coefficients matrix.) If the crashing cost functions are non-linear (and non-convex) the deadline TCTP is NP-hard, as is the case for the *discrete* TCTP. Surveys on the topic of TCTP include (Brucker, Drexl, Möhring, Neumann, & Pesch, 1999; Erenguc, Ahn, & Conway, 2001; Hartmann & Briskorn, 2010; Węglarz, Józefowska, Mika, & Waligóra, 2011).

The time–cost tradeoff problem has been studied extensively in the discrete time–cost context, where the expediting of activities is only permitted at specified discrete levels, and at a prescribed cost for each duration level. This renders the cost function non-convex and the respective discrete problem is known to be NP-hard. This is

[*] Tel.: +1 (510) 642 4998; fax: +1 (510) 642 1403.
E-mail address: hochbaum@ieor.berkeley.edu

http://dx.doi.org/10.1016/j.cie.2016.02.018
0360-8352/© 2016 Elsevier Ltd. All rights reserved.

in contrast to the linear continuous TCTP, where the expediting cost per unit is constant for each activity. The linear TCTP is solved by linear programming and thus in polynomial time, and so is the convex cost version, solved in Ahuja, Hochbaum, and Orlin (2003) and Ahuja, Hochbaum, and Orlin (2004) as the convex dual of minimum cost network flow. In addition to the numerous direct applications of the linear time–cost tradeoff problem, the linear TCTP is also a key relaxation of the discrete versions of the problem, including the resource constrained time–cost tradeoff problem. For instance, linear TCTP was used in approximation algorithms for discrete TCTP in Skutella (1998a,b).

The linear TCTP problem and algorithms for solving it have been studied for over five decades. The first combinatorial algorithm for TCTP, that is not based on linear programming, was a minimum cost network flow approach, discovered in 1961 by Fulkerson (1961) and by Kelley (1961), independently. Phillips and Dessouky (1977) proposed an algorithm, referred to here as the PD-algorithm, that identifies, for each project duration deadline, the least cost critical activities to crash, so as to meet the deadline. The PD-algorithm constructs, for a given set of activity durations and the corresponding project duration, an associated network on the critical activities. The algorithm finds in that network a minimum cut, and the activities on the cut have their durations modified, by one unit, so as to reduce the project duration by one unit, at a minimum cost. This approach is inherently of pseudo-polynomial complexity, as the number of iterations can be as large as the makespan of the project for normal durations. It may appear that the performance of the algorithm can be improved by allowing the "bottleneck" possible amount of project duration reduction, by adjusting the durations of activities on the cut by the maximum amount possible, rather than by one unit. The bottleneck version of the algorithm is discussed in Section 3. Yet, the complexity of the algorithm does not improve by employing the bottleneck variant, since the PD-algorithm necessarily evaluates the optimal solution for at least each *breakpoint* of the *time–cost-tradeoff function*, which is a convex piecewise linear function of the expediting costs versus the makespan of the project (see Fig. 2). Skuttela demonstrated, in Skutella (1998, chap. 1.3), that the number of breakpoints of the time–cost-tradeoff function can be *exponential* in the size of the input. Thus the PD-algorithm, as well as any algorithm that computes the solution for every breakpoint, must have exponential time complexity. A detailed discussion on this topic is provided in Section 5.

As mentioned above, the time–cost tradeoff problem on linear costs can be solved using linear programming. While linear programming is solved in polynomial time, its theoretical complexity and practical performance are uncompetitive when compared to, say, flow algorithms. Indeed, even though the time–cost tradeoff problem can be solved as a linear programming problem, the approach of Phillips and Dessouky is the one commonly used technique for solving TCTP.

Our main contribution here is a repeated cuts algorithm, which is a scaling variant of the PD-algorithm, that runs in polynomial time for both linear and convex expediting costs and uses a minimum s, t-cut routine at each iteration. For a project network on n activities, m precedence constraints and reduction of project duration by a quantity D, the repeated cuts algorithm makes $O(n \log D)$ calls to a minimum s, t-cut procedure in a graph on n nodes and m arcs.

1.1. Paper organization

The paper is organized as follows: In Section 1.2 we introduce notations, basic concepts and definitions related to TCTP. The section includes a discussion of maximum flow minimum cut which ar relevant to the algorithms presented. Section 2 provides a description of the PD-algorithm. It is explained how to construct, what we call here, the crashing graph, and how the minimum cut in that graph is mapped into an optimal cost one unit expediting of the finish time of the project. It is proved, in Observation 2, that the algorithm applies, with the same complexity, also to convex cost functions. A two iterations example illustrates the role of forward arcs and backward arcs of the minimum cut. Section 3 describes the bottleneck improvement to the PD-algorithm where instead of one unit reduction in finish time corresponding to each cut, the reduction is by a quantity that could be considerably larger – the bottleneck of the cut.

The scaling repeated cuts algorithm is given in Section 4. This section introduces the concept of Δ-crashing graph. The cut decomposition theorem is stated and proved; a Δ-stage algorithm is described; and finally the scaling repeated cuts procedure is stated and its complexity is proved. The scaling repeated cuts also applies to convex costs functions with the same complexity as the linear case. Section 5 describes the contrast between the PD-algorithm versus the scaling repeated cuts algorithm in terms of not only their complexity, but in how the repeated cuts algorithm "short-cuts" the progress along the time–cost-tradeoff curve. We then compare, for the family of problems described in Skutella (1998a), the performance of the two algorithms and illustrate the dramatic improvement in run time of the repeated cuts algorithm, over that of the PD-algorithm.

1.2. Preliminaries and notations

In formulating a project network there are two major approaches. One is Activity-on-Arc, and the other Activity-on-Node, known by the acronyms AOA and AON, respectively, e.g. A Guide (2008).

A project network, formulated using AON, has a node for each activity and an arc for each precedence relation. Let V be the sets of activities, and A the set of arcs representing the precedence relations. Let node s be the start node, and node t be the finish node. The project graph is a directed acyclic graph $G = (V \cup \{s,t\}, A)$ with arc $(i,j) \in A$ if activity j has activity i as an immediate predecessor. That is, an activity can only be started once all its predecessors have been completed. The project graph is necessarily a directed acyclic graph, DAG, for a project to be well defined. The earliest finish time of the project is the longest path from s to t, in terms of the sum of the durations of the activities on the path. The longest path is found, using dynamic programming, in linear time $O(|A|)$. For a project graph $G = (V, A)$ we denote the number of activities by $|V| = n$ and the number of precedence arcs by $|A| = m$.

We use here the AOA representation of a project graph which is an s, t-graph $G = (V, A)$ with s the node indicating the start of the project, and t is the node indicating the end of the project. Note that any AON presentation can be mapped to a AOA presentation, e.g. by splitting each activity/node into two nodes and the arc in between the two nodes is then the respective activity arc.

In the time–cost-tradeoff problem each activity has a *normal* duration, and a minimum possible duration. The normal duration of the project is the earliest finish time of the project with the activities at their normal durations. The minimum possible duration of the project is the earliest finish time for the activities at their minimum durations.

Let d_{ij} be the normal duration of activity (i,j), and $\underline{d}_{ij}$ the minimum possible duration of the activity and c_{ij} the cost of expediting the activity by one unit. Dummy activities have a fixed normal duration of 0 and cost per unit change of ∞. Feasible activity durations x_{ij} satisfy that $\underline{d}_{ij} \leqslant x_{ij} \leqslant d_{ij}$. For a feasible durations vector $\mathbf{x}$ we let the associated earliest finish time of the project be denoted by $T(\mathbf{x})$. Let $G_{\mathbf{x}} = (V, A)$, be the project graph for activity durations vector $\mathbf{x}$, and $s_{ij}(\mathbf{x})$ the total slack (float) of activity (i,j) with respect to $T(\mathbf{x})$. Let $A(\mathbf{x})$ be the set of critical activities when the durations vector is $\mathbf{x}$. All critical activities have their total slack equal to 0.

To identify critical activities, slacks and the project duration, a well known dynamic programming, called the **CPM** (Critical Path Method), is used. For completeness, CPM is described next. Let $\mathbf{d}$ be a given durations vector that associates, with each activity arc (i,j) in $G = (V, A)$, the duration d_{ij}. Each node j in the graph gets two labels, $E(j)$ and $L(j)$. $E(j)$ is the earliest time by which each activity originating at j can start, and $L(j)$ is the latest time by which each activity originating at j must start in order to finish by the earliest finish time of the project, $E(t)$.

Setting the boundary condition $E(s) = 0$, the dynamic programming recursion to compute the labels $E(j)$ proceeds according to the topological order in $G = (V, A)$ which is a directed acyclic graph, DAG. That is, it computes the function $E(j)$, once all j's predecessors function values have been computed:

$$E(j) = \max_{i|(i,j)\in A}\{E(i) + d_{ij}\}.$$

The process terminates when $E(t) = T(\mathbf{d})$ has been computed. The backwards dynamic programming recursively computes $L(i)$ for all nodes in reverse topological order. The boundary condition is $L(t) = E(t)$, and the recursive equation is:

$$L(j) = \min_{p|(j,p)\in A}\{L(p) - d_{jp}\}.$$

The **total slack** (also known as "total float") of an activity (i,j) of duration d_{ij} is $s_{ij}(\mathbf{d}) = LT(j) - ET(i) - d_{ij}$. We refer to the total slack here as *slack*. An arc is critical if it lies on a longest path (critical path), which occurs if and only if its slack is 0. A node i that lies on a critical path satisfies that $ET(i) = LT(i)$. We call such nodes *critical nodes*. Notice that both endpoints of a critical arc are critical nodes. We denote the set of critical nodes with respect to durations vector $\mathbf{d}$ by $V_{\mathbf{d}}$.

An $s,\ t$-cut, (S, T), in an $s,\ t$-graph $G = (V, A)$ with arc capacities u_{ij} for all $(i,j) \in A$, is a set of arcs associated with a partition of V to two sets, S and $T = \bar{S}$, such that $s \in S$ and $t \in T$, $(S, T) = \{(i,j) \in A | i \in S, j \in T\}$. For a graph, with all capacity lower bounds equal to 0, the *capacity of the cut* (S, T) is the sum of the weights on the arcs with tails in S and heads in T defined as: $C(S,T) = \sum_{(i,j)\in(S,T)} u_{ij}$. If some arcs (i,j) in A have positive non-zero lower bounds, ℓ_{ij}, then the capacity of the $s,\ t$ cut (S, T) is generalized to, $C(S,T) = \sum_{(i,j)\in A|i\in S,j\in T} u_{ij} - \sum_{(i,j)\in A|i\in T,j\in S} \ell_{ij}$. Note that the cut capacity definition for the 0 lower bounds is a special case of general lower bounds. For a cut (S, T) the arcs (i,j) such that $i \in S,\ j \in T$ are referred to as the *cut-forward* arcs, and the arcs (i,j) such that $i \in T,\ j \in S$ are referred to as the *cut-backward* arcs.

The dual of the minimum $s,\ t$-cut is the maximum flow problem. Once a maximum flow is determined, it takes only linear time to determine the minimum $s,\ t$-cut, and therefore, to find the minimum $s,\ t$-cut in an $s,\ t$-graph $G = (V, A)$, one finds first a maximum flow in the graph. Since the maximum flow in the graph is relevant here, several definitions are introduced.

Let f_{ij} denote the flow value on arc $(i,j) \in A$. A flow vector $\mathbf{f} = \{f_{ij}\}_{(i,j)\in A}$ is said to be *feasible* if it satisfies:

(i) flow balance constraints: for each $j \in V \setminus \{s,t\}$, $\sum_{(i,j)\in A} f_{ij} = \sum_{(j,k)\in A} f_{jk}$ (i.e., inflow(j) = outflow(j)) and
(ii) capacity constraints: for all $(i,j) \in A$, the flow value is between the lower bound and upper bound capacity of the arc, i.e., $\ell_{ij} \leqslant f_{ij} \leqslant u_{ij}$.

For a flow vector $\mathbf{f} = \{f_{ij}\}_{(i,j)\in A}$, we let $f(D_1, D_2) = \sum_{i\in D_1, j\in D_2} f_{ij} - \sum_{p\in D_2, q\in D_1} f_{pq}$. The *value* of the flow is the amount of flow on any $s,\ t$-cut, e.g. $f(\{s\}, V \setminus \{s\})$.

For f_{ij} the variables denoting the amount of flow on arc $(i,j) \in A$ and v the value of the flow, the maximum flow problem is formulated as,

$$\begin{array}{ll} \max & v \\ \text{subject to} & \sum_{i|(k,i)\in A} f_{ki} - \sum_{j|(j,k)\in A} f_{jk} = 0, \quad \forall k \in V \setminus \{s,t\} \\ & \sum_{i|(s,i)\in A} f_{si} = v \\ & -\sum_{j|(j,t)\in A} f_{jt} = -v \\ & \ell_{ij} \leqslant f_{ij} \leqslant u_{ij}\ \forall (i,j) \in A. \end{array}$$

The first set of constraints are the flow balance constraints, and the last set of constraints are the capacity constraints. Note that the problem of finding a maximum flow in a graph with non-zero capacity lower bounds is reducible to finding maximum flow in a graph with 0 lower bounds: One uses a maximum flow procedure on an associated graph to find a feasible flow with respect to the lower and upper bounds, similar to the concept of phase I in linear programming. Once such feasible solution is obtained, the residual graph has 0 lower bounds. Details on this transformation can be found in Ahuja, Magnanti, and Orlin (1993, chap. 6.7, p. 193).

Given a feasible flow $\mathbf{f}$, an arc $(i,j) \in A$ is said to be *saturated* if $f_{ij} = u_{ij}$. An arc (i,j) is said to be a *residual arc* if $(i,j) \in A$ and $f_{ij} < u_{ij}$ or if $(j,i) \in A$ and $f_{ji} > \ell_{ij}$. For $(i,j) \in A$, the *residual capacity* of arc (i,j) with respect to the flow $\mathbf{f}$ is $u^{\mathbf{f}}_{ij} = u_{ij} - f_{ij}$, and the *residual capacity* of the reverse arc (j,i) is $u^{\mathbf{f}}_{ji} = f_{ij} - \ell_{ij}$. We will also denote the residual capacity of arc (i,j) as $r_{\mathbf{f}}(i,j) = u^{\mathbf{f}}_{ij}$.

Let $A^{\mathbf{f}}$ denote the set of residual arcs for flow $\mathbf{f}$ in G which consists of all arcs or reverse arcs with positive residual capacity. Let $G_{\mathbf{f}} = (V, A^{\mathbf{f}})$ denote the *residual network with respect to* $\mathbf{f}$. An augmenting path with respect to $\mathbf{f}$ is a path from s to t in the residual graph $G_{\mathbf{f}}$. The seminal theorem of Ford and Fulkerson (1956) says that a flow $\mathbf{f}$ is maximum if and only if the residual graph $G_{\mathbf{f}}$ has no $s,\ t$ path. Indeed, if a flow is maximum then a minimum cut of capacity equal to the flow value is attained by labeling all nodes reachable from s in the residual graph. Since the flow is maximum, there is no augmenting path in the residual graph, and the set of labeled nodes does not include t. Denoting the set of labeled nodes by S, the cut $(S, \bar{S})$ has capacity equal to the value of the flow and is thus a minimum cut (Ford & Fulkerson, 1956).

Let $T^0 = T(\mathbf{d})$ be the normal project duration, or the earliest project finish time, for activities at their normal durations d_{ij}. Let $T^{min} = T(\underline{\mathbf{d}})$ be the minimum possible project duration corresponding to activity durations at their minimum level $\underline{d}_{ij}$.

Observation 1. For any given project duration value T^*, where $T^* \geqslant T^{min}$, there exists a feasible vector of activity durations that achieve that project duration.

Hence, the Time–Cost-Tradeoff problem, which is to find crashed activity durations of the least cost, so as to achieve the project duration deadline T^*, for $T^* \geqslant T^{min}$, has a feasible solution.

2. The PD-algorithm

The algorithm is based on the following observation: In order to reduce the duration of the project by one unit, all critical paths must have their lengths reduced by one unit. In a constructed $s,\ t$ graph containing only the critical arcs, any $s,\ t$ cut (S, T) is a set of critical arcs so that a unit reduction in the duration of each will result in a unit reduction in the project duration. The key is to assign capacities to the critical arcs so that a modification to the durations of the arcs in a minimum cut will correspond to a minimum cost unit reduction in the project duration.

At each iteration of the PD-algorithm, either the duration of the project is reduced by one unit, or it is proved that such reduction is infeasible. From Observation 1 above, as long as the deadline is no less than the minimum possible project duration, each iteration results in a reduction of the project duration by one unit. At an iteration with a project duration T, a minimum s,t-cut problem is found on an associated graph that we refer to as the *crashing graph*. If $T \geqslant T^{min}$ then the cut capacity is finite and the duration of the project is reduced by one unit at a cost equal to the cut capacity. Otherwise, the project duration may not be reduced any further as it is already at its minimum possible duration, T^{min}.

2.1. Construction of the crashing graph at each iteration

Let the project graph $G = (V, A)$ be given as AOA. Let the cost of reducing the duration of an activity (i,j) per unit, if feasible, be c_{ij}. These cost functions, denoted by $c_{ij}()$, have three segments:

$$c_{ij}(\mathbf{x}) = \begin{cases} \infty & \text{if } x_{ij} \leqslant \underline{d}_{ij} \\ c_{ij}(d_{ij} - x_{ij}) & \text{if } \underline{d}_{ij} < x_{ij} \leqslant d_{ij} \\ 0 & \text{if } d_{ij} \leqslant x_{ij}. \end{cases}$$

Given a feasible durations vector $\mathbf{x}$, the corresponding set of critical activity arcs, $A(\mathbf{x})$, and the set of critical nodes $V_{\mathbf{x}}$, the respective capacitated *crashing graph* $G^c(\mathbf{x}) = (V_{\mathbf{x}}, A)$ is the subgraph of G induced on the critical nodes. Each critical activity arc, $(i,j) \in A(\mathbf{x})$, is assigned lower and upper bounds, $(\ell_{ij}(\mathbf{x}), u_{ij}(\mathbf{x}))$ as follows,

$$(\ell_{ij}(\mathbf{x}), u_{ij}(\mathbf{x})) = \begin{cases} (0, c_{ij}) & \text{if } \underline{d}_{ij} < x_{ij} = d_{ij} \\ (c_{ij}, c_{ij}) & \text{if } \underline{d}_{ij} < x_{ij} < d_{ij} \\ (c_{ij}, \infty) & \text{if } \underline{d}_{ij} = x_{ij} < d_{ij} \\ (0, \infty) & \text{if } \underline{d}_{ij} = x_{ij} = d_{ij}. \end{cases}$$

All precedence arcs have capacity lower and upper bounds, $(0, \infty)$. The following observation generalizes the lower and the upper capacity bounds for nonlinear crashing cost functions.

Observation 2. For convex cost functions, $c_{ij}(x_{ij})$, we define $c^+_{ij}(x_{ij}) = c_{ij}(x_{ij} + h) - c_{ij}(x_{ij})$ to be the right subgradient of $c_{ij}()$ at x_{ij} for h a small enough value. (For the integer durations problem $h = 1$.) Similarly, let $c^-_{ij}(x_{ij}) = c_{ij}(x_{ij}) - c_{ij}(x_{ij} - h)$ to be the left subgradient of $c_{ij}()$ at x_{ij}. Then,

$$(\ell_{ij}(\mathbf{x}), u_{ij}(\mathbf{x})) = \begin{cases} (0, c^-_{ij}) & \text{if } \underline{d}_{ij} < x_{ij} = d_{ij} \\ (c^+_{ij}, c^-_{ij}) & \text{if } \underline{d}_{ij} < x_{ij} < d_{ij} \\ (c^+_{ij}, \infty) & \text{if } \underline{d}_{ij} = x_{ij} < d_{ij} \\ (0, \infty) & \text{if } \underline{d}_{ij} = x_{ij} = d_{ij}. \end{cases}$$

The procedure described includes calls to a routine that finds a minimum s,t-cut in the crashing graph $G^c(\mathbf{x})$, denoted by $(S(\mathbf{x}), T(\mathbf{x}))$, or (S, T) when there is no risk of ambiguity.

The PD-algorithm has the goal of reducing the project finish time from T^0 to T^*. For $T^* \geqslant T^{min}$ there is a feasible setting to the durations of the activities, e.g. at the minimum durations, and hence there is a finite cut at each iteration. Phillips and Dessouky (1977) showed that for a finite s,t-cut partition (S, T) of capacity $\sum_{(i,j)\in A, i\in S, j\in T} u_{ij} - \sum_{(i,j)\in A, i\in T, j\in S} \ell_{ji}$ in the graph $G_{\mathbf{x}}$ the following crashing reduces the duration of the project by one unit:

$$x_{ij} := \begin{cases} x_{ij} - 1 & \text{if } i \in S,\ j \in T \\ x_{ij} + 1 & \text{if } i \in T,\ j \in S \text{ and } \ell_{ij} = c_{ij} > 0. \end{cases}$$

The cost of this one unit reduction in the duration of the project is $\sum_{i\in S, j\in T} c_{ij} - \sum_{j\in T, i\in S} c_{ji}$. Because this cost is also the capacity of the respective cut, (S, T), then the minimum cost reduction of project duration is achieved for a cut of minimum capacity. A formal description of the PD-algorithm is,

```
PROCEDURE PD $(G = (V, A), \mathbf{d}, T^*)$.
Step 0 $\mathbf{x} := \mathbf{d}$; $\overline{T} = T(\mathbf{d})$
Step 1 Find the critical activities $A(\mathbf{x})$ in $G_{\mathbf{x}}$;
Step 2 Construct the $s,t$-graph $G^c(\mathbf{x})$ with capacity lower and upper bounds $(\ell_{ij}(\mathbf{x}), u_{ij}(\mathbf{x}))$;
Step 3 Find a minimum $s,t$-cut in $G^c(\mathbf{x})$, $(S, T)$. Update $\mathbf{x}$ for $(i,j)$ cut-forward arc or cut-backward arc by setting:
    $x_{ij} := \begin{cases} x_{ij} - 1 & \text{if } i \in S,\ j \in T \\ x_{ij} + 1 & \text{if } i \in T,\ j \in S \text{ and } \ell_{ij} = c_{ij} > 0; \end{cases}$
Set $\overline{T} := \overline{T} - 1$. If $\overline{T} = T^*$ terminate. Else, go to Step 1.
```

The procedure is illustrated for a project graph with normal duration of 15 in Fig. 1, where two iterations reduce the finish time to 13.

Lemma 2.1. *The complexity of the PD-algorithm is $O((T^0 - T^*) \cdot T(n,m))$ where $T(n,m)$ is the complexity of finding the minimum s,t-cut problem on a digraph with n nodes and m arcs.*

Proof. The PD-algorithm applies $T^0 - T^*$ iterations in each of which there is a call to a minimum cut procedure in the respective crashing graphs. At each iteration the duration of the project is reduced by one unit at a minimum cost until the duration of the project has been reduce from T^0 to T^*. Hence the complexity of the PD-algorithm is $O((T^0 - T^*) \cdot T(n,m))$, which is exponential in the size of the input. □

A property of the PD-algorithm[1] is proved in the following theorem:

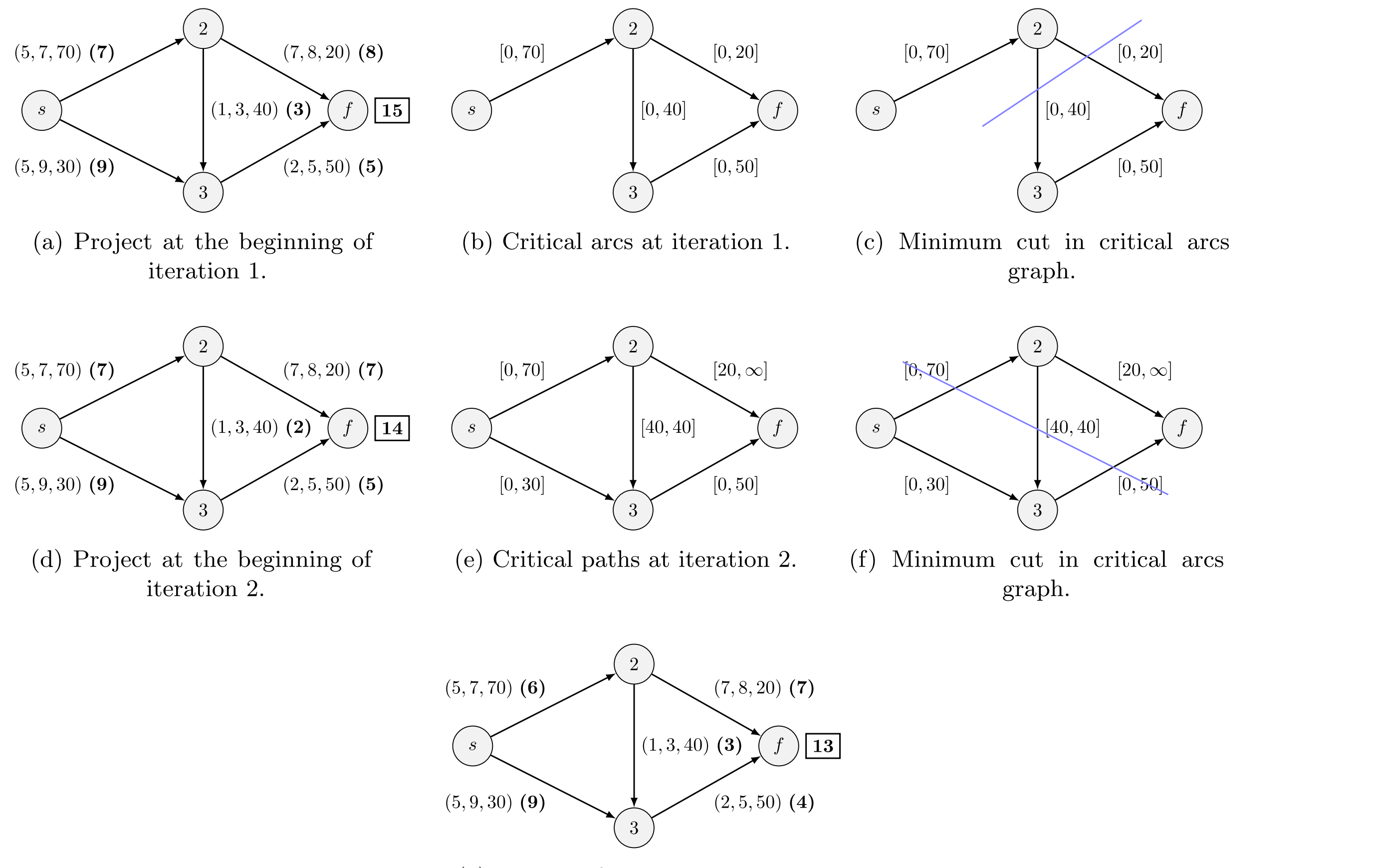
Theorem 2.1. *At the end of an iteration of the PD-algorithm, the flow is feasible for the next iteration.*

Proof. Let $f(\mathbf{x})$ be the maximum flow in $G^c(\mathbf{x})$, at one iteration for duration vector $\mathbf{x}$. The only arcs that may have their capacities modified between subsequent iterations are arcs in the cut. If an arc (i,j) is a cut-forward arc, then it is saturated at the upper bound value, which is finite and equal to the cost, $f_{ij}(\mathbf{x}) = u_{ij}$ and therefore feasible in the next iteration as the capacity upper bound on that arc either does not change, or increases to infinity. If an arc (i,j) is a cut-backward arc then $f_{ij}(\mathbf{x}) = \ell_{ij} = c_{ij}$ and the flow remains feasible in the next iteration where the lower bound either does not change or becomes 0. Therefore the flow on a cut-backward arc remains feasible, as the lower bound for such arc can only go down in the next iteration. The lower bound goes down if the increase in the duration of a cut-backward arc (i,j) is such that $x_{ij} = d_{ij}$. □

3. The bottleneck variant of the PD-algorithm

An obvious way to improve the PD-algorithm is to replace the one unit modification in the durations of cut activities by a larger

[1] Communicated to the author by Dessouky.



(a) Project at the beginning of iteration 1. (b) Critical arcs at iteration 1. (c) Minimum cut in critical arcs graph.

(d) Project at the beginning of iteration 2. (e) Critical paths at iteration 2. (f) Minimum cut in critical arcs graph.

(g) Project after two iterations.

Fig. 1. Two iterations, that reduce project finish time, from 15 to 13. Notation: For project graphs, $(\underline{d}_{ij}, d_{ij}, c_{ij})$, $(\mathbf{x}_{ij})$. For crashing graphs, $[\ell_{ij}, u_{ij}]$.

amount, whenever possible, thus achieving a corresponding larger reduction in project duration.

For cut-forward arcs, the largest possible crashing is the difference between the current duration and the lower bound on the duration. Similarly, for cut-backward arcs the largest possible crashing is the difference between the current duration and the upper bound of the duration. One more issue, that is not considered in the PD-algorithm, is the crashing amount that can transform the status of non-critical arcs to critical. This consideration is redundant for the PD-algorithm, as any non-critical arc has a slack of at least one unit. But if the crashing amount is greater than the slack of a non-critical activity, that arc may not only become critical, but may be reduced beyond the normal duration. In that case, there would be a cost to charge for expediting this activity, a cost that is not considered when finding the cut.

To address that we include in a newly defined crashing graph all arcs, either critical or non-critical. For a feasible durations vector $\mathbf{x}$ we let the equivalent *crashing graph*, defined on all activities, be $G^c(\mathbf{x}) = (V, A)$, with the lower and upper bounds on capacities as follows,

$$(\ell_{ij}(\mathbf{x}), u_{ij}(\mathbf{x})) = \begin{cases} (0,0) & \text{if } x_{ij} = d_{ij} \text{ and } s_{ij}(\mathbf{x}) \geqslant 1 \\ (0, c_{ij}) & \text{if } \underline{d}_{ij} < x_{ij} = d_{ij} \text{ and } s_{ij}(\mathbf{x}) = 0 \\ (c_{ij}, c_{ij}) & \text{if } \underline{d}_{ij} < x_{ij} < d_{ij} \\ (c_{ij}, \infty) & \text{if } \underline{d}_{ij} = x_{ij} < d_{ij} \\ (0, \infty) & \text{if } \underline{d}_{ij} = x_{ij} = d_{ij}. \end{cases}$$

Once a minimum cut is found, the durations of the cut-forward arcs can be decreased down to their respective minimum durations. Therefore, among the cut-forward arcs the bottleneck reduction is,

$$\delta_{\text{forward}} = \min_{(i,j)\in(S,T)} \{x_{ij} - \underline{d}_{ij}\}.$$

Among the cut-backward arcs the bottleneck reduction is,

$$\delta_{\text{backward}} = \min_{(p,q)\in(T,S)} \{d_{pq} - x_{pq}\}.$$

Finally, the reduction in the project finish time may result in non-critical arcs becoming critical. Therefore the reduction in the project finish time cannot exceed (at the same cost) the minimum slack of the non-critical arcs:

$$\delta_{\text{non-critical}} = \min_{(i,j)\in(S,T)\cap(A\setminus A(\mathbf{x}))} s_{ij}(\mathbf{x}).$$

Definition 1. The *bottleneck value* δ of a cut $(S,T) = (S(\mathbf{x}), T(\mathbf{x}))$ is the minimum quantity of the minimum feasible reduction in duration among cut-forward arcs; the minimum feasible increase in the cut-backward arcs, and the smallest positive slack of non-critical cut-forward arcs: $\delta = \min\{\delta_{\text{forward}}, \delta_{\text{backward}}, \delta_{\text{non-critical}}\}$.

We refer to the cut arcs in (S,T) for which the reduction value is equal to the bottleneck value as *bottleneck arcs*.

The bottleneck variant of the PD-algorithm differs from the PD-algorithm in Step 3, where the activities are crashed by the bottleneck amount δ, rather than by one unit only. Step 3 is replaced by Step 3(bottleneck) as given below:

Step 3(bottleneck) Find a minimum s,t-cut in $G^c(\mathbf{x})$, (S,T). Let the bottleneck value of (S,T) be δ. Update $\mathbf{x}$ for (i,j) cut-forward arc or cut-backward arc by setting:

$$x_{ij} := \begin{cases} x_{ij} - \delta & \text{if } i \in S,\ j \in T \\ x_{ij} + \delta & \text{if } i \in T,\ j \in S \text{ and } \ell_{ij} = c_{ij} > 0; \end{cases}$$

Set $\overline{T} := \overline{T} - \delta$. If $\overline{T} = T^*$ terminate. Else, go to Step 1.

As mentioned in the introduction, and discussed further in Section 5, the bottleneck variant does not have a better complexity than the PD-algorithm. Namely, even with the bottleneck reduction in project duration at each iteration, the complexity of the PD-algorithm remains exponential.

4. A scaling repeated cuts algorithm

4.1. A Δ-crashing graph

The scaling algorithm is based on allowing, at each iteration, a reduction of Δ units in the durations of the cut activities, that result in Δ units reduction in the project finish time. We first show that for project finish time T and project deadline T^* there exists a cut of bottleneck value at least $\Delta = \frac{T-T^*}{n}$.

The following definition of Δ-critical arcs applies to non-critical and critical arcs. For the latter, the slack value is zero. (When there is no risk of ambiguity we refer to $s_{ij}(\mathbf{x})$ as s_{ij}.)

Definition 2. An activity arc $(i,j) \in A$ in a project graph with activity durations $\mathbf{x}$ is called Δ-critical if its total slack $s_{ij}(\mathbf{x})$ satisfies $s_{ij}(\mathbf{x}) < \Delta$ and $x_{ij} + s_{ij}(\mathbf{x}) - \underline{d}_{ij} \geqslant \Delta$.

With this definition, all critical arcs $(i,j) \in A(\mathbf{x})$ that can have their duration feasibly reduced by Δ units, namely, $x_{ij} - \underline{d}_{ij} \geqslant \Delta$, are Δ-critical. Also, all non-critical with respect to durations $\mathbf{x}$, that if reduced by Δ would become critical, are Δ-critical. Let $A_\Delta(\mathbf{x})$ be the set of the Δ-critical arcs in the graph with durations $\mathbf{x}$, and a Δ-crashing graph is the subgraph of $G = (V, A)$ that contains only Δ-critical arcs $(V, A_\Delta(\mathbf{x}))$, with the following capacities: The capacity upper bound associated with each arc (i,j) in the graph is the increment in the cost derived from reducing the duration x_{ij} by Δ, or, if $x_{ij} = d_{ij}$ and the arc is not critical with slack $s_{ij} < \Delta$ then the actual reduction in duration is only $\Delta - s_{ij}$. The corresponding cost and capacity upper bound is: $c^-_{ij}(x_{ij}) = c_{ij}(x_{ij} - (\Delta - s_{ij})) - c_{ij}(x_{ij})$.

The capacity lower bounds are non-zero only when $x_{ij} < d_{ij}$ and therefore the arc is critical and the gain if the duration is increased by Δ is: $c^+_{ij}(x_{ij}) = c_{ij}(x_{ij}) - c_{ij}(\min\{x_{ij} + \Delta, d_{ij}\})$.

With this notation the capacity lower and upper bounds in the Δ-critical graph with respect to durations vector $\mathbf{x}$ are:

$$(\ell_{ij}(\mathbf{x}, \Delta), u_{ij}(\mathbf{x}, \Delta)) = \begin{cases} (0, 0) & \text{if } x_{ij} = d_{ij} \text{ and } s_{ij} \geqslant \Delta \\ (0, c^-_{ij}) & \text{if } \underline{d}_{ij} < x_{ij} = d_{ij} \\ (c^+_{ij}, c^-_{ij}) & \text{if } \underline{d}_{ij} < x_{ij} < d_{ij} \\ (c^+_{ij}, \infty) & \text{if } \underline{d}_{ij} = x_{ij} < d_{ij} \\ (0, \infty) & \text{if } \underline{d}_{ij} = x_{ij} = d_{ij}. \end{cases}$$

As in Observation 2, the construction of the Δ crashing graph and capacities is applicable for convex costs functions as well.

We next establish the existence of a cut, of bottleneck value at least $\Delta = \frac{T-T^*}{n}$, in the crashing graph with project duration T, for n the number of activities.

4.2. Cut-decomposition

The scaling repeated cuts algorithm relies on the existence of a cut in the crashing graph of sufficiently high bottleneck capacity. Specifically, for a project with activity durations $\mathbf{x}$ and target deadline on the project duration T^*, it is proved here that there exists at least one cut in the crashing graph of bottleneck capacity at least Δ, for $\Delta = \frac{T(\mathbf{x})-T^*}{n}$.

Let $\mathbf{d}^*$ be an optimal activity durations vector associated with project finish time T^*. Consider the project graph $G = (V, A)$, we define the *residual-durations* graph $G^{rd} = (V, A^{rd})$, with capacities vector $\mathbf{u} = \mathbf{d}^* - \mathbf{x}$, called *residual durations*, defined as follows:

$$u_{ij} = \begin{cases} d^*_{ij} - x_{ij} & \text{if } d_{ij} \geqslant x_{ij} \text{ and } (i,j) \in A \\ x_{ji} - d^*_{ji} & \text{if } d^*_{ji} < x_{ji} \text{ and } (j,i) \in A. \end{cases}$$

That is, the set of arc A^{rd} has one arc for each arc in A, either in the forward direction, or in the reverse direction.

Definition 3. The *cut decomposition* of $G^{rd} = (V, A^{rd})$ with capacities $\mathbf{d}^* - \mathbf{x}$ is a collection of finite capacity cuts, $\{(S_1, \overline{S_1}), (S_2, \overline{S_2}), \ldots, (S_k, \overline{S_k})\}$, called *primitive cuts*, of duration-bottleneck values $\delta_1, \delta_2, \ldots, \delta_k$, such that for each arc $(i,j) \in A$ contained as cut-forward arc in the primitive cuts, $\{(S_{j_1}, \overline{S_{j_1}}), (S_{j_2}, \overline{S_{j_2}}), \ldots, (S_{j_{q_1}}, \overline{S_{j_{q_1}}})\}$, or as cut-backward arc in the primitive cuts $\{(S_{i_1}, \overline{S_{i_1}}), (S_{i_2}, \overline{S_{i_2}}), \ldots, (S_{i_{q_2}}, \overline{S_{i_{q_2}}})\}$,

$$d^*_{ij} = x_{ij} - \sum_{p=1}^{q_1} \delta_{j_p} + \sum_{p=1}^{q_2} \delta_{i_p}.$$

Lemma 4.1. *Let T be the project duration associated with the durations vector $\mathbf{d}$. There is a cut decomposition consisting of at most n primitive cuts, at least one of which has bottleneck value $\geqslant \frac{T-T^*}{n}$.*

Proof. The proof is constructive. We present an algorithm that generates a cut decomposition-one cut at each iteration. At the first iteration the current graph is the *residual-durations* graph $G^{rd} = (V, A^{rd})$ with capacities $\mathbf{d}^* - \mathbf{x}$, for the current durations vector is $\mathbf{x} = \mathbf{d}$. $S_0 = \{s\}$ and $\ell = 0$. Iteration $\ell + 1$ begins with the removal-step, or shrinking-step, of arcs $(i,p) \in (S_\ell, \overline{S_\ell})$ that have capacity of 0, and arcs $(q,i) \in (\overline{S_\ell}, S_\ell)$ that have capacity 0. Shrinking amounts to adding nodes p and q to the set S_ℓ. At the end of the shrinking-step the set S_ℓ with the shrunk nodes is denoted $S_{\ell+1}$. We observe that any precedence arc adjacent to S_ℓ is removed since it has capacity 0 in the residual-durations graph (for every precedence arc (i,j), $x_{ij} = d^*_{ij} = 0$).

Next, the iteration index ℓ is advanced by one unit, $\ell \leftarrow \ell + 1$. The cut $(S_\ell, V \setminus S_\ell)$ in the current graph is then selected as the next primitive cut, and its duration-bottleneck value δ_ℓ is computed. Note that δ_ℓ must be positive as all 0 capacity arcs have been removed. The cut arcs capacities are then updated by the duration-bottleneck value of the cut: For each cut-forward arc, (i,j), the bottleneck value is subtracted, $x_{ij} := x_{ij} - \delta_\ell$, and for each cut-backward arc (p,q) the bottleneck value is added, $x_{pq} := x_{pq} + \delta_\ell$. This updated graph becomes the current graph for the subsequent iteration.

At the end of iteration ℓ, at least one arc adjacent to S_ℓ is a bottleneck arc and thus removed at the beginning of the subsequent iteration. The iterations continue until all arcs have been removed. Since at each iteration at least one arc in A^{rd} is a bottleneck arc, the number of iterations is at most the number of activity arcs in A^{rd} and therefore at most n.

Since each primitive cut corresponds to a cut in the original graph, it follows that there are at most n cuts so that reduction of the bottleneck values of on the arcs of the cuts results in the duration vector $\mathbf{d}^*$ and project finish time of T^*. Hence at least one of these cuts has bottleneck value $\geqslant \frac{T-T^*}{n}$. $\square$

We comment that with this cut-decomposition, every arc with $x_{ij} > d^*_{ij}$ is included only as forward arc in a consecutive sequence of cuts, and every arc with $x_{ij} < d^*_{ij}$ is included only as backward arc in a consecutive sequence of cuts.

Corollary 4.1. *Let $\Delta = \frac{T^0 - T^*}{2n}$ for T^0 the project duration associated with the durations vector $\mathbf{d}^0$, and $T^* \geqslant T^{min}$ the deadline finish time of the project. Let T be the project duration associated with $\mathbf{d} \leqslant \mathbf{d}^0$. While $T \geqslant \frac{1}{2}(T^0 + T^*)$ there exists a cut of Δ-critical activities that can be crashed by Δ so as to achieve a Δ reduction in the project duration, from T to $T - \Delta$.*

Proof. From Lemma 4.1 there is a cut of bottleneck value $\frac{T-T^*}{n}$. Since $T \geqslant \frac{1}{2}(T^0 + T^*)$ then $T - T^* \geqslant \frac{1}{2}(T^0 - T^*)$ and $\frac{T-T^*}{n} \geqslant \frac{1}{2}\frac{T^0-T^*}{n} = \frac{T^0-T^*}{2n}$. □

4.3. The Δ-stage

We are now ready to describe the scaling repeated cuts procedure. For a durations vector $\mathbf{d}$ and the corresponding project duration T, the procedure calls for the Δ-stage in which the PD-algorithm is applied to the Δ-critical graph, reducing one multiple of Δ at a time, until the project duration, T, satisfies $T \leqslant \frac{1}{2}(T^0 + T^*)$. From Corollary 4.1, this requires at most n iterations.

Procedure Δ-stage $(\mathbf{d}, \Delta, T^0 = T(\mathbf{d}))$.
Step 0 $\overline{T} = T(\mathbf{d})$; $\mathbf{x}^1 = \mathbf{d}$; $k = 1$; $s_{ij}(\mathbf{d}) = s^1_{ij}$ the slacks of activity $(i,j) \in A$;
Step 1 Construct the s, t-graph $G^c(\mathbf{x}^k) = (V, A_\Delta(\mathbf{x}^k))$ with capacity lower and upper bounds $(\ell_{ij}(\mathbf{x}^k, \Delta), u_{ij}(\mathbf{x}^k, \Delta))$;
Step 2 Find a minimum s, t-cut in $G^c(\mathbf{x}^k)$ (S, T). Update $\mathbf{x}$ and $\mathbf{s}$ for (i,j) cut-forward arc or cut-backward arc by setting:

$$x^{k+1}_{ij} := \begin{cases} d_{ij} & \text{if } i \in S,\ j \in T \text{ and } s^k_{ij} \geqslant \Delta \\ x^k_{ij} + s^k_{ij} - \Delta & \text{if } i \in S,\ j \in T \text{ and } s^k_{ij} < \Delta \\ \min\{x^k_{ij} + \Delta, d_{ij}\} & \text{if } i \in T,\ j \in S \text{ and } \ell_{ij} = c^+_{ij} > 0. \end{cases}$$

$$s^{k+1}_{ij} := \begin{cases} s^k_{ij} - \Delta & \text{if } i \in S, j \in T \text{ and } s^k_{ij} \geqslant \Delta \\ 0 & \text{if } i \in S, j \in T \text{ and } s^k_{ij} < \Delta \\ \max\{0, x^k_{ij} + \Delta - d_{ij}\} & \text{if } i \in T, j \in S \text{ and } \ell_{ij} = c^+_{ij} > 0, \end{cases}$$

and set $\overline{T} := \overline{T} - \Delta$.
If $k = n$ terminate, output durations $\mathbf{x} = \mathbf{x}^n$, and $T^0 = \overline{T}$. Else, $k := k + 1$ and go to Step 1.

Since each Δ-stage involves exactly n calls to a minimum cut procedure, its complexity is $O(nT(n,m))$. This proves that,

Lemma 4.2. *The complexity of the Δ-stage is $O(nT(n,m))$.*

4.4. The scaling algorithm

At the end of Δ-stage the project finish time, $\overline{T}$, satisfies, $\overline{T} \leqslant \frac{T^0+T^*}{2}$. The scaling procedure then resets $T^0 := \overline{T}$, $\Delta := \frac{1}{2}\Delta$ and calls procedure Δ-stage. When $\Delta \leqslant 1$ the call is for procedure PD until T^* is reached. Since $\Delta \leqslant 1$ that implies that $T^0 - T^* \leqslant 4n$ and at most $2n$ iterations of procedure PD are required to attain an optimal solution. Alternatively 1-stage can be called adding a stopping rule for when $\overline{T} = T^*$.

Scaling Repeated Cuts $(\mathbf{d}, T^0 = T(\mathbf{d}), T^*)$.
Step 0 $\Delta = \frac{T^0 - T^*}{2n}$;
Step 1 If $\Delta \leqslant 1$ call procedure Δ-stage $(\mathbf{d}, \Delta, T^0 = T(\mathbf{d}))$ with output $\mathbf{x}$, else go to Step 3.
Step 2 Set $\Delta := \frac{1}{2}\Delta, \mathbf{d} = \mathbf{x}$ and go to Step 1.
Step 3 {terminate} set $\mathbf{d} = \mathbf{x}$ and call procedure PD $(G = (V, A), \mathbf{d}, T^*)$.

The algorithm makes at most $\log(T^0 - T^*)$ calls to the Δ-stage scaling procedure. From Lemma 4.2, each Δ-stage takes $O(nT(n,m))$ steps. In Step 3 Procedure PD is called when $\Delta \leqslant 1$ in which case the number of units reduction in the duration of the project is at most $2n$. It follows that Step 3 runs in $O(nT(n,m))$ steps. Hence, the total complexity of the procedure is $O(nT(n,m)\log(T^0 - T^*))$. One particularly efficient procedure, in theory, for solving the minimum s, t-cut problem is by Orlin (2013), which has the complexity of $T(n,m) = O(mn)$. In practice however, Hochbaum's PseudoFlow algorithm is the most efficient algorithm for the max-flow min-cut problem Hochbaum (2008) and Chandran and Hochbaum (2009). We conclude that,

Theorem 4.1. *For a project network with n activities, m precedence constraints and an expedited goal of finish time T^*, the scaling repeated cuts algorithm runs in time $O(nT(n,m)\log(T^0 - T^*))$ for $T(n,m)$ the running time for solving the minimum s, t-cut problem.*

5. Comparing the performance of the PD-algorithm with the repeated cuts algorithm

Although the bottleneck variant of the PD-algorithm appears substantially more efficient, it may take a number of iterations to move from one breakpoint to another. This occurs when there are multiple optimal minimum cuts in the crashing graph. In that case, there are different sets of cut activities that can be expedited, at the same cost. Therefore, using the bottleneck variant of the PD-algorithm may take a number of iterations, where in each a different cut is found, but all cuts are of the same cost. In recent work, we describe an algorithm that identifies all minimum cuts of the same cost as a given minimum cut, in the respective crashing graphs, in linear time, Hochbaum (2015).

Skutella (1998a) demonstrated a family of TCTP problems where the number of *breakpoints* is exponential in the size of the

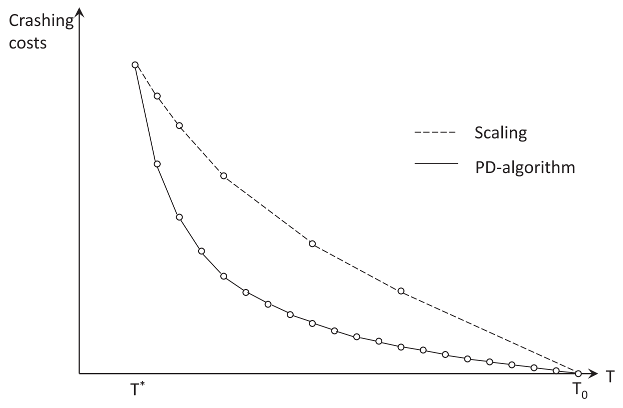


Fig. 2. Breakpoints of PD-algorithm compared to breakpoints of scaling repeated cuts algorithm.

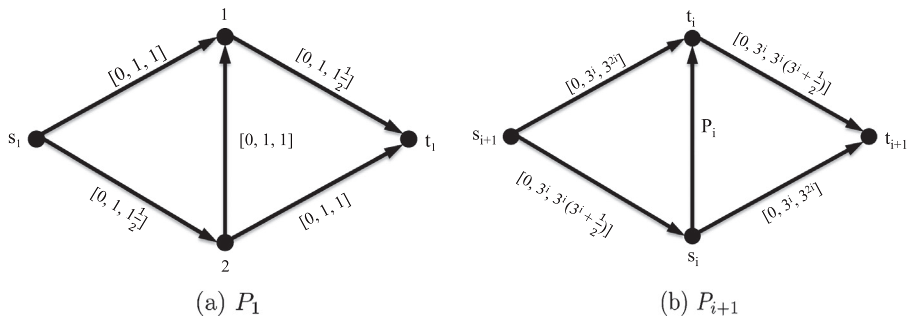


Fig. 3. Family of networks.

input. Hence, if an improved variant of the PD-algorithm that can move from one breakpoint to another in one iteration were to be known, it would still require an exponential number of iterations. An abstract comparison of the behavior of the two algorithms is depicted on the time–cost-tradeoff curve in Fig. 2. Whereas the PD-algorithm finds the optimal solution for every breakpoint (at least), the scaling TCT procedure makes long leaps between Δ-stages, narrowing down the length of these leaps as it gets closer to the target project duration T^*.

We use the family of problems, of Skutella (1998, chap. 1.3), to illustrate the stark performance difference between the PD-algorithm and the scaling TCT algorithm. Fig. 3 shows the recursive construction of the family of project networks for any value of a parameter i. Project network P_1 is depicted in Fig. 3(a). Project network P_{i+1} is constructed recursively from project P_i as shown in Fig. 3(b), and has $2i$ nodes and $4i+1$ activity arcs. Under the normal durations of the activities (the upper bounds) the duration of the project for P_i is 3^{i+1}. For this family of projects, there is a breakpoint in the time cost tradeoff curve for every value of project duration in $\{0, 1, \ldots, 3^{i+1} - 1\}$.

Consider the project network P_k with project duration $T^0 = 3^{k+1}$. Let the target project duration be 3^k. To solve this TCTP, the PD-algorithm calls the minimum cut procedure $3^{k+1} - 3^k$ times, for a total complexity of $3^k \cdot T(2k, 4k+1)$. At each iteration, the PD-algorithm modifies the durations of at least one activity in P_1, since these are the least costly arcs. By contrast, scaling repeated cuts works only on the top layers that contain the Δ-critical arcs, and only reaches the layer of P_1 in the final iteration of the 1-stage. The complexity of scaling repeated cuts for project network P_k is at most $15k^2 \cdot T(2k, 4k+1)$. Even for k as small as 50 (a network on 101 activities) the run time of PD-algorithm for P_{50} exceeds the run time of scaling repeated cuts by at least a factor of $2 \cdot 10^{19}$.

6. Concluding remarks

We demonstrate here that the time–cost-tradeoff problem for linear or convex expediting costs is solved with a scaling repeated cuts algorithm in polynomial time, thus improving dramatically on the exponential run time of Phillips and Dessouky algorithm.

Beyond the polynomial time complexity established here, the repeated cuts algorithm has several insightful properties. The scaling repeated cuts algorithm for uniform cost project network on n activities was recently shown to run in time $O(nm)$, Hochbaum (2015). This has far reaching implications beyond the context of project networks.

Acknowledgement

The author's research was supported in part by NSF – USA award No. CMMI-1200592.

References

A guide to the project management body of knowledge. 4th ed., ANSI/PMI 99-001-2008 (copyright Project Management Institute, Inc. 2008).
Ahuja, R. K., Hochbaum, D. S., & Orlin, J. B. (2003). Solving the convex cost integer dual network flow problem. *Management Science, 49*(7), 950–964.
Ahuja, R. K., Hochbaum, D. S., & Orlin, J. B. (2004). A cut-based algorithm for the convex dual of the minimum cost network flow problem. *Algorithmica, 39*(3), 189–208.
Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1993). *Network flows: Theory, algorithms, and applications*.New Jersey: Prentice-Hall.
Brucker, P., Drexl, A., Möhring, R., Neumann, K., & Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research, 112*, 3–41.
Chandran, B. G., & Hochbaum, D. S. (2009). A computational study of the pseudoflow and push-relabel algorithms for the maximum flow problem. *Operations Research, 57*(2), 358–376.
Erenguc, S., Ahn, T., & Conway, D. G. (2001). The resource constrained project scheduling problem with multiple crashable modes: An exact solution method. *Naval Research Logistics (NRL), 48*(2), 107–127.
Ford, L. R., & Fulkerson, D. R. (1956). Maximal flow through a network. *Canadian Journal of Mathematics, 8*, 339–404.
Fulkerson, D. R. (1961). A network flow computation for project cost curve. *Management Science, 7*(2), 167–178.
Hartmann, S., & Briskorn, D. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research, 207*(1), 1–14.
Hochbaum, D. S. (2008). The pseudoflow algorithm: A new algorithm for the maximum-flow problem. *Operations Research, 56*(4), 992–1009.
Hochbaum, D. S. (2015). *The uniform costs time–cost-tradeoff problem and the assignment problem*. UC Berkeley Manuscript.
Hochbaum, D. S., & Shanthikumar, J. G. (1990). Convex separable optimization is not much harder than linear optimization. *Journal of ACM, 37*(4), 843–862.
Kelley, J. R. Jr., (1961). Critical path planning and scheduling: Mathematical basis. *Operations Research, 9*(3), 296–320.
Orlin, J. B. (2013). Max flows in $O(nm)$ time, or better. symposium on theory of computing, STOC13 (pp. 765–774).
Phillips, S., Jr., & Dessouky, M. I. (1977). Solving the project time/cost tradeoff problem using the minimal cut concept. *Management Science, 24*, 393–400.
Skutella, M. (1998a). Approximation and randomization in scheduling. Ph.D. Thesis, Berlin, Germany: Technische Universität Berlin, Fachbereich Mathematik.
Skutella, M. (1998b). Approximation algorithms for the discrete time–cost tradeoff problem. *Mathematics of Operations Research, 23*(4), 909–929.
Węglarz, J., Józefowska, J., Mika, M., & Waligóra, G. (2011). Project scheduling with finite or infinite number of activity processing modes – A survey. *European Journal of Operational Research, 208*(3), 177–205.