Res Eng Design (2016) 27:347–366
DOI 10.1007/s00163-016-0222-7

ORIGINAL PAPER

Optimizing time–cost trade-offs in product development projects with a multi-objective evolutionary algorithm

Christoph Meier[1] · **Ali A. Yassine**[2] · **Tyson R. Browning**[3] · **Ulrich Walter**[1]

Received: 9 July 2015 / Revised: 12 January 2016 / Accepted: 28 February 2016 / Published online: 16 March 2016

Abstract Time–cost trade-offs arise when organizations seek the fastest product development (PD) process subject to a predefined budget, or the lowest-cost PD process within a given project deadline. Most of the engineering and project management literature has addressed this trade-off problem solely in terms of crashing—options to trade cost for time at the individual activity level—and using acyclical networks. Previously (Meier et al. in IEEE Trans Eng Manag 62(2):237–255, 2015), we presented a rich model of the iterative (cyclical) PD process that accounts for crashing, overlapping, and stochastic activity durations and iterations. In this paper, we (1) propose an optimization strategy for the model based on a multi-objective evolutionary algorithm, called ε-MOEA, which identifies the Pareto set of best time–cost trade-off solutions, and (2) demonstrate the approach using an automotive case study. We find that, in addition to crashing, activity overlapping, process architecture, and work policy provide further managerial levers for addressing the time–cost trade-off problem. In particular, managerial work policies guide process cost and duration into particular subsets of the Pareto-optimal solutions. No work policy appeared to be superior to the others in both the cost and duration dimensions; instead, a time–cost trade-off arises due to the choice of work policy. We conclude that it is essential for managers to consider all of the key factors in combination when planning and executing PD projects.

Keywords Time–cost trade-off · Product development · Project management · Iteration · Crashing · Overlapping · Work policy · Optimization · Genetic algorithm

1 Introduction

The management of product development (PD) projects requires addressing many time–cost trade-off problems. Although project time–cost trade-offs have been addressed extensively in the project management literature, this research focused on an approach called crashing (options to trade time and cost at the individual activity level) (Tavares et al. 2002), with only a small number of papers looking beyond to the network-level effects of activity overlapping (e.g., Roemer et al. 2000; Roemer and Ahmadi 2004). Moreover, all of this work presumed acyclical networks, whereas PD projects are iterative/cyclical (e.g., Kline 1985; Cooper 1993; Smith and Eppinger 1997; Browning and Yassine 2016).

In another paper (Meier et al. 2015), we introduced a rich model of the PD process as a cyclical network of activities linked by a variety of inputs and outputs, each modeled with particular characteristics pertaining to the flow of information and/or work products they represent. This model allowed us to explore the effects of crashing, overlapping, rework, and process structure, and it extended our insight into the effects of managerial actions on PD project performance (especially time–cost outcomes) at

Electronic supplementary material The online version of this article (doi:10.1007/s00163-016-0222-7) contains supplementary material, which is available to authorized users.

✉ Ali A. Yassine
ay11@aub.edu.lb

[1] Institute of Astronautics, Technische Universität München, 85748 Garching, Germany

[2] Department of Industrial Engineering and Management, American University of Beirut, Beirut, Lebanon

[3] Neeley School of Business, Texas Christian University, TCU Box 298530, Fort Worth, TX, USA

both the activity and the overall process levels. The model revealed that varying the managerial policies toward crashing, overlapping, and rework has drastic effects on project time and cost. While we were able to identify the model's general behaviors with a set of artificial test problems, the natural next steps are to (1) optimize the model and (2) apply it to an industrial case. Those are the two contributions of this paper. First, we develop a multi-objective evolutionary algorithm (MOEA) to optimize the model. This MOEA requires several special considerations in order to apply it to a complex PD process. We discuss these challenges and their solutions. Second, we apply the MOEA to an industrial example, the development process for an automobile hood at a large automotive company. We discuss the results and their implications for managers—in particular, the impact of work policy (i.e., managerial decisions about when to start activities, including rework) on process duration and cost. We demonstrate that time–cost trade-offs also emerge due to the choice of work policy. Thus, this paper contributes a new optimization procedure for a complex PD process model that accounts for activity overlapping, crashing, and rework, and it increases its validity with a demonstration of its application to an industrial process.

The paper is organized as follows. The next section provides background and literature review. Section 3 gives a brief overview of the model, about which readers can find fuller details elsewhere (Meier et al. 2015). Section 4 describes the new MOEA, and Sect. 5 presents the automotive hood application and its results. The paper concludes in Sect. 6 with a summary of insights and a discussion of limitations and future work.

2 Background

2.1 Related models: simulation and optimization

Many models and techniques have been proposed in the literature to assist managers in planning and controlling PD projects more effectively (Krishnan and Ulrich 2001; Browning and Ramasesh 2007). The design structure matrix (DSM) is one such modeling method that provides a means of representing a complex system (such as a process) based on its constituent elements (activities) and their relationships (Eppinger and Browning 2012). The DSM imparts a significant capability to visualize, analyze, and manage activity iterations, and DSM-based models show promising results in managing complex PD processes (Browning 2001; Yassine and Braha 2003). However, the DSM literature to date focuses on either simulating a particular DSM sequence (e.g., Browning and Eppinger 2002) or optimizing the sequence of DSM elements based on a specific objective (e.g., Meier et al. 2007; Shaja and Sudhakar 2010). Meier et al. (2007) proposed a binary DSM-based procedure using a specialized genetic algorithm to find an optimized sequence for a set of design activities. They showed that the optimality of a solution depends on the objective used for DSM rearrangement. In an activity sequencing context, these objectives vary from reducing iteration/feedback, increasing concurrency, and reducing lead time and cost to some combination of these.

Recent literature has pointed out the potential and success of simulation techniques for managing development processes (Karniel and Reich 2009). Adler et al. (1995) used discrete event simulation (DES) to study performance in companies pursing multiple, concurrent, non-unique PD projects. The simulation allowed them to identify bottleneck activities and several development process characteristics. Baldwin et al. (1999) also used DES to manage the design process in building construction projects. Finally, Browning and Eppinger (2002) and many others (e.g., Cho and Eppinger 2005; Huang and Chen 2006; Lévárdy and Browning 2009) used Monte Carlo DES based on a DSM representation of development projects. These models revealed several interesting process characteristics and performance measures, including expected project duration, cost, and risk and their drivers. Although these models mainly aimed to determine process duration and cost for a given process architecture (i.e., task arrangement), some (e.g., Zhuang and Yassine 2004; Abdelsalam and Bao 2007) also sought to optimize the architecture by comparing the time and cost of various architectures. However, none of these allowed for time–cost trade-off analysis.

The time–cost trade-off problem (TCTP) has been studied extensively in the project scheduling literature (Deckro et al. 1995; Brucker et al. 1999; Hartmann and Briskorn 2010). The TCTP has been treated either as an activity crashing problem, where activities can be shortened (crashed) at additional costs, or as a special case of a multi-mode scheduling problem, where activities can be assigned to different resources at different costs. The latter is referred to as the discrete TCTP (DTCTP) (De et al. 1995). There exist two versions of the DTCTP: deadline (DTCTP-D) and budget (DTCTP-B). While in the deadline problem the total cost is minimized within a given deadline, the DTCTP-B is concerned with minimizing the project duration without exceeding a given budget (Vanhoucke 2015). The stochastic version of the TCTP, which assumes uncertainty in activity duration and cost, was also addressed and referred to as the STCTP (Cohen et al. 2007; Bruni et al. 2015). Stochastic programming, robust optimization, parametric programming, and meta-heuristics are the main optimization approaches used for modeling uncertainty in the STCTP (Herroelen and Leus 2005).

Although this literature is relevant to our proposed model, it lacks fundamental aspects characterizing PD scheduling problems, which are the centerpiece of our model. These aspects include rework potential of activities (i.e., cyclical project networks), process architecture, and work policy. Furthermore, the result of a DTCTP optimization is just one robust schedule but not the entire set of Pareto-optimal solutions, which is provided by our proposed model (Cohen et al. 2007; Hazır et al. 2011, 2015).

2.2 Multi-objective optimization problems

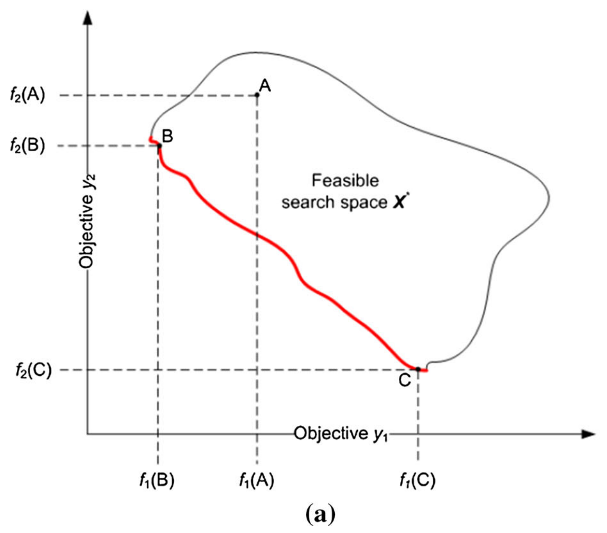
Some research on the time–cost trade-off in PD project management has recognized the increase in costs due to overlapping as a function of additional rework, as well as the increase in costs due to crashing as a function of the amount of extra resource assignments to an activity (Browning and Ramasesh 2007). For example, Roemer et al. (2000) analyzed the time–cost trade-off due to overlapping and bridged the gap between the literature on overlapping and crashing by demonstrating the benefits of jointly applying both strategies (Roemer and Ahmadi 2004). Along similar lines, Berthaut et al. (2014) presented a linear integer optimization model for the resource-constrained project scheduling problem with overlapping modes. Time–cost trade-offs between project duration and overlapping costs are discussed. Their model highlights the importance of interaction between resource constraints and overlapping modes and suggests the relevance of jointly considering them. Finally, Gerk and Qassim (2008) considered a mixed-integer linear programming model for the acceleration of projects, employing the simultaneous crashing, overlapping, and substitution of project activities. Application of the model to industrial case studies highlighted the practicality of using a combination of acceleration strategies in project management. Nevertheless, the models used in these publications did not account for iteration and uncertainty in task duration and cost, which significantly increases the complexity of the time–cost trade-off calculations.

The multi-objective nature of most real-world problems has raised recent interest in the study of simultaneous optimization of multiple, competing objective functions (e.g., Deb 2009; Fujita et al. 1998; Poloni et al. 2000; Coverstone-Carroll et al. 2000; Coello et al. 2007). Unlike in single-objective optimization problems, a single, best solution with respect to all objectives usually does not exist for a multi-objective optimization problem. Instead, one seeks to identify the set of best trade-offs, which consists of elements that are superior to others in at least one of the objectives (e.g., time or cost), while inferior with respect to the others. This set is commonly known as the *Pareto-optimal* set (or *non-dominated* solutions) because it resides in a region of the search space called the Pareto frontier (*Pareto front* for short). Other solutions in the search space are called *dominated* solutions. Figure 1a provides a visual example of a Pareto front for the minimization of two objectives, y_1 and y_2. The curve on the lower left of the feasible search space indicates the Pareto front. Solutions B and C reside on the Pareto front and cannot be dominated by any other feasible point, while point A is dominated by many of the points (such as B) on the Pareto front. Another important multi-objective concept is *ε-dominance*, which reduces the cardinality of the Pareto region by decomposing the objective space into multiple hyper-boxes (Laumanns et al. 2002; Helbig and Pateva 1994). Spanning a length of ε_i in the ith objective, each hyper-box can contain at most one solution. Therefore, the number of solutions in the ε-Pareto front increases with decreasing ε. Figure 1b visualizes ε-dominance for two objectives (cost and duration).

Among the variety of evolutionary algorithms in the literature, genetic algorithms (GAs) (Goldberg 1989) have received the most attention. Originally developed to solve single-objective problems, GAs have been extended by multi-objective operators. Their massive parallel search approach qualifies them to handle multi-objective problems extraordinarily well, because the entire Pareto set (rather than a single Pareto point) can be quickly approximated in a single run. Other meta-heuristics such as ant algorithms (Doerner 2008) and particle swarm algorithms (Coello et al. 2004) have also been enriched by multi-objective versions. However, a general superiority of one approach over the others cannot be claimed due to the "no free lunch" theorems (Wolpert and Macready 1997).

Although multi-objective GAs (MOGAs) retain many of the features of single-objective GAs, the former must be specially designed to ensure (1) the preservation of diversity and (2) the proper definition of a convergence criterion (Laumanns et al. 2002; Kumar and Rockett 2002). Most MOGA designs, especially those proposed in the 1980s and 1990s, do not account for both preservation of diversity and convergence to the true Pareto front, but merely cope with one or the other.

Goldberg (1989) suggested the use of *Pareto ranking* and selection in combination with a *niching* mechanism to move a population toward the Pareto front. Fonseca and Fleming (1998) proposed the MOGA; Srinivas and Deb (1994), the non-dominated sorting GA (NSGA); Deb et al. (2002), the NSGA-2; and Zitzler et al. (2002), the strength of Pareto evolutionary algorithm (SPEA) and SPEA2. All of these approaches focused on a good distribution of solutions without ensuring convergence to the Pareto front. On the other hand, Rudolph and Agapie (2000) and Hanne (1999) developed multi-objective evolutionary algorithms (MOEAs) that guarantee at least some solutions belonging to the global Pareto front. As opposed to the former



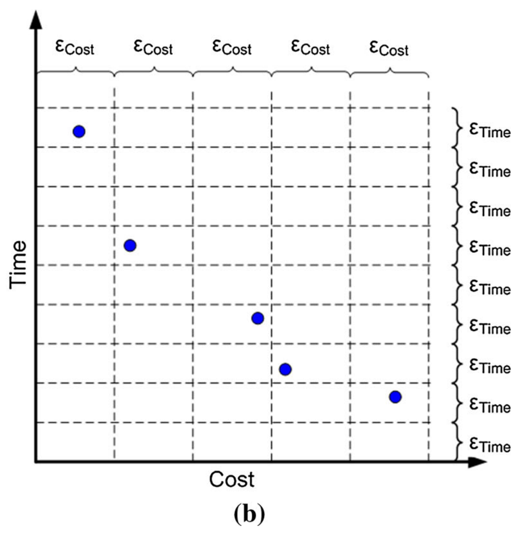


Fig. 1 Illustration of Pareto-optimal solutions. **a** Illustration of Pareto-optimal solutions (points B and C) in a search space. **b** Illustration of ε-dominance

MOGAs, however, both of these algorithms fail to maintain a good distribution of identified solutions, resulting in incomplete coverage of the efficient frontier (Laumanns et al. 2002).

More recent approaches have attempted to address both diversity and convergence. Laumanns et al. (2002) introduced an archiving strategy based on ε-dominance that they claimed guarantees convergence toward the Pareto front as well as a good solution spread. Based on the idea of applying ε-dominance to MOEAs, Deb et al. (2003) proposed further enhancements to Laumanns et al.'s approach.

In this paper, we apply an ε-MOEA to the PD project time–cost trade-off problem because of the encouraging results in (Deb et al. 2005) and its use of the ε-dominance criterion, which enables it to cope with noisy fitness functions (like a simulation) in a better way than most other MOEAs.

3 The PD process model and simulation

We base our optimization study on a rich model of time–cost trade-offs in PD processes that addresses both intra- and inter-activity effects. This section provides a brief overview of the model; full details are available in a companion paper (Meier et al. 2015). The model enables explorations of two foundational concepts, process architecture and work policy, which we will review first.

3.1 Process architecture and work policy

The model treats a PD process as a network of n_A activities and their dependencies, modeled in DSM $\boldsymbol{M}$ in a sequence given by vector $\boldsymbol{S}$. Six DSM layers capture various attributes of the dependencies, which are as follows:

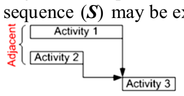
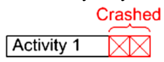
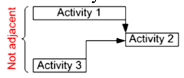
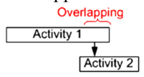
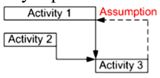
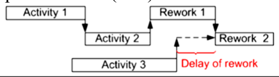
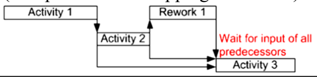
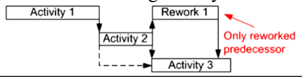
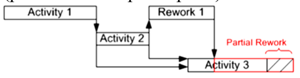
- $m_{1,ij}$ $(i \neq j)$, the probability of activity j causing rework for activity i;
- $m_{2,ij}$ $(i \neq j)$, the fraction of activity i that must be redone if rework occurs;
- $m_{3,ij}$ $(i \neq j)$, the percentage of activity i required to be done before its *final* output is available for j;
- $m_{4,ij}$ $(i \neq j)$, the percentage of j that can occur without penalty before it requires *complete* input from i;
- $m_{5,ij}$ $(i \neq j)$, the percentage of i that can occur without penalty before it can produce any *preliminary* output for j; and
- $m_{6,ij}$ $(i \neq j)$, the percentage of j that can occur without penalty before it must receive *preliminary* information from i.

Each $\boldsymbol{S}$ will result in some of the dependencies appearing below the diagonal in the DSM (feed-forward) and others appearing above (feedback). Changing $\boldsymbol{S}$ causes various dependencies to change from feed-forward to feedback or vice versa, thus changing whether a particular input to an activity comes from an upstream or downstream activity. Therefore, varying $\boldsymbol{S}$ manipulates the *process architecture* (Browning and Eppinger 2002; Eppinger and Browning 2012).

A *work policy* is a set of rules about when to start and stop activities (Browning and Eppinger 2002). It can include rules for crashing and overlapping activities and for waiting on inputs versus proceeding based on assumptions about missing inputs. We investigate the five work policies identified by Meier et al. (2015), designated P_1–P_5 and summarized in Table 1. Policy P_1

Table 1 Summary of work policies studied (Meier et al. 2015)

P_1	Most conservative; limits activity concurrency to that specified in planned order of activities; no crashing or overlapping
P_2	Increases "natural concurrency" by identifying all opportunities to start activities as early as possible; no crashing or overlapping
P_3	P_2 with allowance for crashing and overlapping ("artificial concurrency")
P_4	P_3 with delay of activity rework until all of its predecessors are finished with known rework
P_5	Most aggressive; P_3 with performance of activity rework even if its predecessors are unfinished

Rule	P_1	P_2	P_3	P_4	P_5	Rule	P_1	P_2	P_3	P_4	P_5
Adjacent, independent activities in the activity sequence (***S***) may be executed concurrently: 	✓	✓	✓	✓	✓	An activity may be crashed: 	✗	✗	✓	✓	✓
Non-adjacent activities in ***S*** may be executed concurrently: 	✗	✓	✓	✓	✓	Adjacent activities (in ***S***) may be overlapped: 	✗	✗	✓	✓	✓
Activities may begin with assumptions about any inputs from downstream (in ***S***) activities: 	✓	✓	✓	✓	✓	Downstream rework is deferred until the completion of any rework by all predecessors (in ***S***):  Activity 1 Rework 1 Activity 2 Rework 2 Activity 3 Delay of rework	✗	✗	✗	✓	✗
Activities must wait for all inputs from upstream activities (in ***S***), even if reworking (except where overlapping is allowed):  Activity 1 Rework 1 Activity 2 Wait for input of all predecessors Activity 3	✓	✓	✓	✓	✗	Activities must wait for all inputs from upstream activities (in ***S***), *unless* that input is from a reworking activity:  Activity 1 Rework 1 Activity 2 Only reworked predecessor Activity 3	✗	✗	✗	✗	✓
Upstream rework is accomplished immediately (prioritized and preemptive):  Activity 1 Rework 1 Activity 2 Partial Rework Activity 3	✓	✓	✓	✓	✓						

was adopted from Browning and Eppinger (2002) and can be regarded as a conservative policy, because it limits the amount of activity concurrency by allowing only independent activities in M_1 (i.e., activities for which $m_{1,ij} = 0$) that are also adjacent in ***S*** to work simultaneously. P_2 also permits non-adjacent activities in ***S*** to work concurrently, thereby getting more work done earlier, albeit sometimes at a greater risk of having to be reworked. P_3 adds crashing and overlapping options to P_2. To analyze the impact of additional rules for the timing of rework, we defined a more sequential, rework-deferring policy, P_4, and a more concurrency-oriented policy, P_5. P_4 presumes that it is beneficial to delay rework if any of an activity's predecessors (in ***S***) have not yet finished (see Table 1), because these predecessors might deliver new results that could cause further rework. Delaying the activity's start until the completion of all of its predecessors obviates this problem, but this policy wastes time if an activity's predecessors do not in fact alter its inputs. Hence, P_5 relaxes this inhibition, allowing an activity to proceed despite ongoing rework by its predecessors. This policy would be expected to decrease process duration even further while increasing rework and cost.

3.2 Activity cost and duration

Respectively, c_i and t_i represent the base cost and duration of activity i. The cost and duration with learning curve and rework impact in iteration k_i are then as follows:

$$\tilde{c}_i(k_i) = c_i l_i(k_i) m_{2,ij} \tag{1}$$

$$\tilde{t}_i(k_i) = t_i l_i(k_i) m_{2,ij} \tag{2}$$

where $m_{2,ij}$ is the rework impact and $l_i(k_i)$ is the learning curve effect, modeled as a percentage of the activity's original duration. Three additional effects may alter the cost and duration of activity i in iteration k_i: crashing ($\Delta c_{C_i(k_i)}$ and $\Delta t_{C_i(k_i)}$), overlapping ($\Delta c_{O_i(k_i)}$ and $\Delta t_{O_i(k_i)}$), and rework ($\Delta c_{R_i(k_i)}$ and $\Delta t_{R_i(k_i)}$). We explain these terms below. Putting these terms together yields the following:

$$c_i(k_i) = \tilde{c}_i(k_i) + \Delta c_{O_i(k_i)} + \Delta c_{C_i(k_i)} + \Delta c_{R_i(k_i)} \tag{3}$$

$$t_i(k_i) = \tilde{t}_i(k_i) + \Delta t_{O_i(k_i)} + \Delta t_{C_i(k_i)} + \Delta t_{R_i(k_i)} \tag{4}$$

3.3 Activity overlapping

We use the term *overlapping* to refer to situations of "artificial" concurrency, where an activity begins before the completion of a predecessor activity upon which it depends. (Natural concurrency may also exist when two independent activities proceed in parallel without penalty.) In PD projects, many of the dependencies are determined by information flow, yet many activities may start without all of their inputs by making assumptions instead. In cyclic processes, the source of an activity's input may be a downstream activity providing feedback. Thus, process architecture will affect the efficacy of work policies for overlapping.

As the amount of overlap increases between any two sequentially dependent activities, predecessor activity i and successor (dependent) activity j, the overall duration decreases, but coordination costs and the likelihood of feedback from j causing rework for i increase (Krishnan et al. 1997). As shown in Fig. 2, the final output from the k_ith iteration of i occurs at $T_{f_i(k_i)}$, but the k_jth iteration of activity j prefers input at $T_{s_j(k_j)}$. Overlap implies that $T_{s_j(k_j)} < T_{f_i(k_i)}$. Although this reduces the overall time span, $T_{f_j(k_j)} - T_{s_i(k_i)}$, it may elongate j by rework time $\Delta t_{O_j(k_j)}$, because j began with only preliminary inputs or assumptions about the output from i. The amount of overlap depends on the point in time when i delivers output to j, $T_{D_{ij}(k_i,k_j)} \le T_{f_i(k_i)}$, and the point in time when j receives that input from i, $T_{R_{ij}(k_i,k_j)} \ge T_{s_j(k_j)}$.

Some of an activity's final outputs may be available before the entire activity is finished, and some of an activity's required inputs may not be needed until it is underway. The *final* output of activity i for activity j is available at $T_{A_{ij}(k_i,k_j)} \le T_{f_i(k_i)}$, and activity j needs that input at $T_{N_{ij}(k_i,k_j)} \ge T_{s_i(k_i)}$. The model captures these parameters using $m_{3,ij}$ and $m_{4,ij}$, respectively, such that: $T_{A_{ij}(k_i,k_j)} = T_{s_i(k_i)} + \tilde{t}_i(k_i)m_{3,ij}$ and $T_{N_{ij}(k_i,k_j)} = T_{s_i(k_i)} + \tilde{t}_i(k_i)m_{4,ij}$. In a pure finish-to-start relationship, $m_3 = 1$ and $m_4 = 0$. An overlapped downstream activity j might also be able to begin with only preliminary information from activity i—within limits. The lower bound for $T_{D_{ij}(k_i,k_j)}$ is the point in time specified by the percentage of activity i required to be completed before its *preliminary* output is available for j. This percentage is recorded in $m_{5,ij}$, and $T_{s_i(k_i)} + \tilde{t}_i(k_i)m_{5,ij} \le T_{D_{ij}(k_i,k_j)}$. Any rework due to incomplete predecessor information is assigned to activity j if $T_{D_{ij}(k_i,k_j)} \in \left[T_{s_i(k_i)} + \tilde{t}_i(k_i)m_{5,ij},\ T_{A_{ij}(k_i,k_j)}\right]$. The upper bound for $T_{R_{ij}(k_i,k_j)}$ is the point in time representing the percentage of activity j that can occur before it can benefit from any preliminary input from i. This percentage is recorded in $m_{6,ij}$, so $T_{R_{ij}(k_j)} \le T_{s_j(k_j)} + \tilde{t}_j(k_j)m_{6,ij}$, and j is penalized with rework if $T_{R_{ij}(k_i,k_j)} \in \left[T_{N_{ij}(k_i,k_j)},\ T_{s_j(k_j)} + \tilde{t}_j(k_j)m_{6,ij}\right]$. Thus, $m_{3,ij}$ is the percentage of activity i required to be done before its *final* output is available for j, $m_{4,ij}$ is the percentage of j that can occur without penalty before it requires *complete* input from i, $m_{5,ij}$ is the percentage of i that can occur without penalty before it can produce any *preliminary* output for j, and $m_{6,ij}$ is the percentage of j that can occur without penalty before it must receive *preliminary* information from i. These bounds allow us to compute $T_{D_{ij}(k_i,k_j)}$ and $T_{R_{ij}(k_i,k_j)}$ in any iteration. Then, $t_{O_{ij}(k_i,k_j)}$ represents the entire overlapping duration between activities i and j, and $t'_{O_{ij}(k_i,k_j)}$ is the subset of $t_{O_{ij}(k_i,k_j)}$ that causes rework for activity j due to an imperfect information flow (i.e., when j has to begin with preliminary inputs):

$$t_{O_{ij}(k_i,k_j)} = \min\left\{T_{f_i(k_i)}, T_{f_j(k_j)}\right\} - \max\left\{T_{s_i(k_i)}, T_{s_j(k_j)}\right\} \quad (5)$$

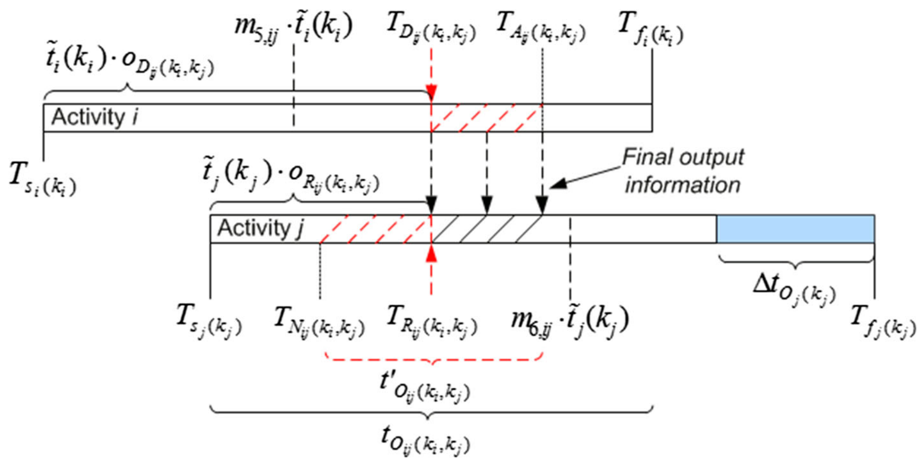


Fig. 2 Example of overlapping two activities, displaying most of the overlapping parameters (Meier et al. 2015)

$$t'_{O_{ij}(k_i,k_j)} = \begin{cases} \min\left\{T_{A_{ij}(k_i,k_j)}, T_{f_j(k_j)}\right\} - \max\left\{T_{s_i(k_i)}, T_{N_{ij}(k_i,k_j)}\right\}, & \text{if activity } j \text{ is not yet active} \\ 0, & \text{if activity } j \text{ is already active} \end{cases} \tag{6}$$

An overlapping function (here, a linear one, although it need not be) determines the amount of rework for activity j caused by overlapping with i:

$$h_{ij}\left(t'_{O_{ij}(k_i,k_j)}\right) = \alpha_{ij} t'_{O_{ij}(k_i,k_j)} \tag{7}$$

where $\alpha_{ij} > 0$ is a rework scaling factor. A typical value is $\alpha = 0.5$, which implies that the duration of the rework due to overlapping requires half of the duration of the overlapping.

If two or more predecessors overlap with activity j and deliver their information to it at the same time, then some of the rework is assumed to be redundant and therefore reduced by $\theta_j = e^{-\beta_j P_j}$ depending on the number of overlapped predecessors ($i \in \boldsymbol{P}_j$) and a cumulative rework factor $\beta_j \geq 0$. A typical value is $\beta_j = 0.5$, which in the (base) case of $|\boldsymbol{P}_j| = 1$ implies that only about 60 % of the cumulative overlapping duration is considered for the computation of overlapping-related rework. Increasing β_j decreases the cumulative rework from a given number of predecessors. Because the maximal pairwise rework between j and any $i \in \boldsymbol{P}_j$ is the minimal amount of cumulative rework, $\hat{i} \in \boldsymbol{P}_j$ is defined as the predecessor with maximal rework time, such that $\hat{h}_{ij}\left(t'_{O_{ij}(k_i,k_j)}\right) = \max\left\{h_{ij}\left(t'_{O_{ij}(k_i,k_j)}\right) | \forall i \in \boldsymbol{P}_j\right\}$. Hence, the overall cumulative change in duration for a downstream activity j due to overlapping is given in Eq. (8), and the associated cost increase is proportional to its increase in duration as shown in Eq. (9).

$$\Delta t_{O_j(k_j)} = \hat{h}_{ij}\left(t'_{O_{ij}(k_i,k_j)}\right) + \theta_i \cdot \left(\sum_{\forall p \in \boldsymbol{P}_j, p \neq \hat{i}} h_{ij}\left(t'_{O_{ij}(k_i,k_j)}\right)\right) \tag{8}$$

$$\Delta c_{O_j(k_j)} = \frac{c_j(k_j)}{t_j(k_j)} \Delta t_{O_j(k_j)} \tag{9}$$

Thus, the model represents the effects of overlapping with an overlapping function (h_{ij}), a rework scaling factor (α_{ij}), and any constraints on particular activity overlaps (governed by m_3–m_6), which determine the points in time in which information is delivered (T_D) and received (T_R). These times are used to determine $\Delta t_{O_j(k_j)}$ and $\Delta c_{O_j(k_j)}$ for each iteration of each activity.

3.4 Activity crashing

$r_i(k_i) \in [0, \hat{r}_i]$ is the *crashing intensity* of activity i in iteration k_i, where $\hat{r}_i \in [0, 1)$ is the maximum crashing intensity allowed. Crashing intensity specifies the minimum activity duration (as a percentage of its regular duration) achievable through the use of crashing. Crashing the k_ith iteration of activity i reduces its duration by:

$$\Delta t_{C_i(k_i)} = -r_i(k_i)\left[\tilde{t}_i(k_i) + \Delta t_{O_i(k_i)}\right] \tag{10}$$

but also increases its cost as a function of the crashing intensity:

$$\Delta c_{C_i(k_i)} = \left(\tilde{c}_i(k_i) + \Delta c_{O_i(k_i)}\right) R_i(r_i(k_i)) \tag{11}$$

where $\tilde{c}_i(k_i)$ is the cost considering learning and rework impact (Eq. 1), $\Delta c_{O_i(k_i)}$ is the cost due to any overlapping (Eq. 9), and in the continuous case:

$$R_i(r_i(k_i)) = \frac{(1 - r_i(k_i))^{-\gamma_i} - 1}{100} \tag{12}$$

where $\gamma_i \geq 0$ is a factor representing the return on crashing efforts for activity i.

The costs of crashing may differ between original and reworked activities. While irrelevant in acyclic processes, where activities are not reworked, in cyclic processes, crashing costs depend on the type of *payment mode*:

(A) *Set-up resource cost* (e.g., setting up a facility or model) must be paid once per full instance of the activity (initial execution or full iteration, but not for partial rework).

(B) *One-time resource cost* (e.g., purchasing a new tool) applies only to the initial execution of the activity and adds no further costs to successive iterations.

(C) *Proportional resource cost* (e.g., staff) must be paid continuously over the activity's duration in any iteration.

3.5 Partial rework of activities

Activity overlapping may also generate *partial rework* for an activity, which we distinguish from regular rework. Partial rework handles the situation when a downstream activity j delivers its output to an upstream activity i after the latest point in time when activity i needed it: $T_{D_{ji}(k_j,k_i)} > m_{6,ji} \cdot \tilde{t}_i(k_i)$ (see Fig. 2). This situation typically emerges if $T_{f_j(k_j)} < T_{f_i(k_i)}$ and $m_{1,ij}$ with $j > i$ holds, which occurs only in iterative processes, such as in the case of iterative overlapping (Krishnan et al. 1997; Browning and Eppinger 2002). In this event, only the duration $T_{D_{ij}(k_i,k_j)} - T_{N_{ij}(k_i,k_j)}$ of activity i must be reworked, because it constitutes the time span between information delivery and the lower bound for information needed (see Fig. 3).

Partial rework modifies the cost and duration of activity i by $\Delta c_{R_i(k_i)}$ and $\Delta t_{R_i(k_i)}$ as shown in Eqs. (3) and (4). If activity i is partially reworked for the zth time, caused by an activity j, then the change in duration due to partial rework is given by the following:

$$\Delta t_{R_i(k_i),z} = l_i(k_i) m_{2,ij} \left(T_{D_{ji}(k_j,k_i)} - T_{N_{ij}(k_i,k_j)} \right) \tag{13}$$

The cost implications depend on the crashing payment mode. For modes A and B,

$$\Delta c_{R_i(k_i),z} = \frac{l_i(k_i) m_{2,ij} \left(T_{D_{ji}(k_j,k_i)} - T_{N_{ij}(k_i,k_j)} \right)}{T_{f_i(k_i)} - T_{S_i(k_i)}} \left(\tilde{c}_i(k_i) + \Delta c_{O_i(k_i)} \right) \tag{14}$$

and for mode C:

$$\Delta c_{R_i(k_i),z} = \frac{l_i(k_i) m_{2,ij} \left(T_{D_{ji}(k_j,k_i)} - T_{N_{ij}(k_i,k_j)} \right)}{T_{f_i(k_i)} - T_{S_i(k_i)}} \left(\tilde{c}_i(k_i) + \Delta c_{O_i(k_i)} + \Delta c_{C_i(k_i)} \right) \tag{15}$$

Generally, an activity i might be partially reworked $n_{R,i}$ times until it is completely reworked—i.e., until k_i changes. Once an activity has been partially reworked, its finish time must be updated. The point in time for the information input that provoked the latest partial rework is designated by $T_{s_{R,i(ki)}}$. Hence, the overall changes in cost and duration due to all partial reworks of activity i are given by the following equations (shown for crashing modes A and B only):

$$\Delta t_{R_i(k_i)} = \sum_{z=1}^{n_{R,i}} \begin{cases} l_i(k_i) m_{2,ij} \left(T_{D_{ji}(k_j,k_i)} - T_{N_{ij}(k_i,k_j)} \right), & \text{if } z = 1 \\ l_i(k_i) m_{2,ij} \left(T_{D_{ji}(k_j,k_i)} - T_{S_{R,i}(k_i)} \right), & \text{if } z > 1 \end{cases} \tag{16}$$

$$\Delta c_{R_i(k_i)} = \sum_{z=1}^{n_{R,i}} \begin{cases} \dfrac{l_i(k_i) m_{2,ij} \left(T_{D_{ji}(k_j,k_i)} - T_{N_{ij}(k_i,k_j)} \right)}{T_{f_i(k_i)} - T_{s_i(k_i)}} \left(\tilde{c}_i(k_i) + \Delta c_{o_i(k_i)} \right), & \text{if } z = 1 \\ \dfrac{l_i(k_i) m_{2,ij} \left(T_{D_{ji}(k_j,k_i)} - T_{N_{ij}(k_i,k_j)} \right)}{T_{f_i(k_i)} - T_{s_i(k_i)}} \left(\tilde{c}_i(k_i) + \Delta c_{o_i(k_i)} + \sum_{l=1}^{z-1} \Delta c_{R_i(k_i),z} \right), & \text{if } z > 1 \end{cases} \tag{17}$$

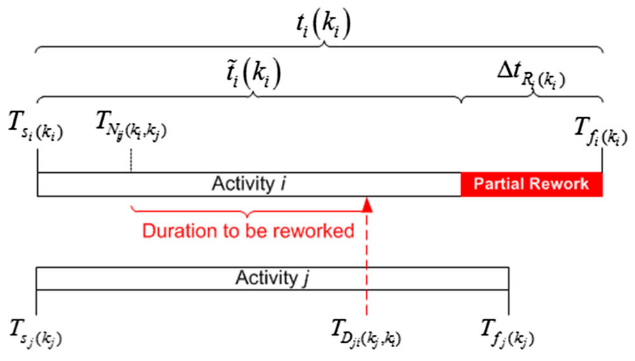


Fig. 3 Illustration of singular partial rework for upstream activity i (Meier et al. 2015)

3.6 Simulation analysis

We adopted a modified form of Browning and Eppinger's (2002) published algorithm for a Monte Carlo DES of a cyclical project network. Summarized briefly, the procedure works as follows (see Fig. 4). First, using work policy rules, the algorithm determines the subset of activities eligible for work. It then advances time and tracks cost until reaching a pertinent time event, T_E, such as one of those shown in Figs. 2, 3. Next, it checks (probabilistically) for any rework caused by feedbacks from any completed activities. When all activities and rework are finished, a single run is complete, providing an overall duration and

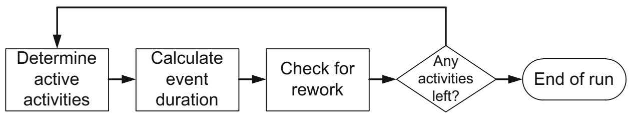


Fig. 4 Event graph for a single simulation run

cost for the project. By conducting many runs, the simulation allows exploration of the frequency distribution (and thus the mean and other parameters) of cost and duration outcomes for any combination of process architecture and work policy. The number of runs required to yield a stable output distribution must be confirmed for each case. We used several validation techniques recommended by Sargent (1999) during the model and simulation's development, and elsewhere (Meier et al. 2015; Meier 2011), we investigated the simulation's performance on a gamut of generated test problems. Overall validity compares favorably with that of other PD process models and simulations reported in the literature (Smith and Morrow 1999).

4 Optimizing the model with an adapted ε-MOEA

The model and simulation presented in the previous section have already been developed in prior works. The contribution of this paper is twofold: the presentation of an optimization procedure tailored to this type of problem (this section) and the validation of the model with a realistic, industrial case study (next section). As noted previously in Sect. 2.2, we develop and apply an ε-MOEA to find the Pareto-optimal set of solutions.

4.1 Flow of the ε-MOEA

Figure 5 summarizes the overall flow of our ε-MOEA design, which is tailored to the previously described time–cost model. In contrast to most MOGAs, which process the entire population through discrete steps, our version of the ε-MOEA is a steady-state evolutionary algorithm (EA)—meaning that, to speed up computation, it processes only a portion of the population in each generation, and each offspring is compared with the parent population immediately after its creation.

Initially, the ε-MOEA randomly generates a certain number of chromosomes, each of which are evaluated for fitness in terms of process cost and duration. All chromosomes are then split up into two distinct populations. Unlike conventional GA designs, the ε-MOEA maintains two coevolving populations during its execution, a parent population, *P*, and an archive population, *A*. While *P* is allowed to contain both dominated and non-dominated chromosomes, *A* contains only non-ε-dominated chromosomes.

Applying the ε-dominance criterion to *A* provides two major benefits (Deb et al. 2005). First, the cardinality of the Pareto region is reduced because ε-dominance decomposes the objective space into multiple hyper-boxes. Second, the ε-dominance approach makes the ε-MOEA interactive with a decision-maker. However, ε-dominance has the disadvantage that ε must be set manually or adaptively, and setting ε inappropriately could result in archiving poor solutions. In the worst case, *A* would contain only one solution that ε-dominates all other solutions.

Provided with the two population sets, the ε-MOEA proceeds as follows. According to a selection method for *P* and *A* (described later in Sect. 4.4), one chromosome is picked out of *P* and one out of *A*. These two chromosomes then undergo the ordinary GA mechanisms of crossover and mutation to create a new chromosome (offspring). We adapt the original ε-MOEA to our model and simulation by adding two further mechanisms. First, we incorporate an efficient repair mechanism to address firm predecessor constraints in the activity sequence. Second, to reduce computational intensity in the simulated determination of fitness, we proceed only with unique offspring. In the final step of the ε-MOEA, acceptance procedures decide if the offspring replaces any members of *P* and/or *A* (Deb et al. 2005).

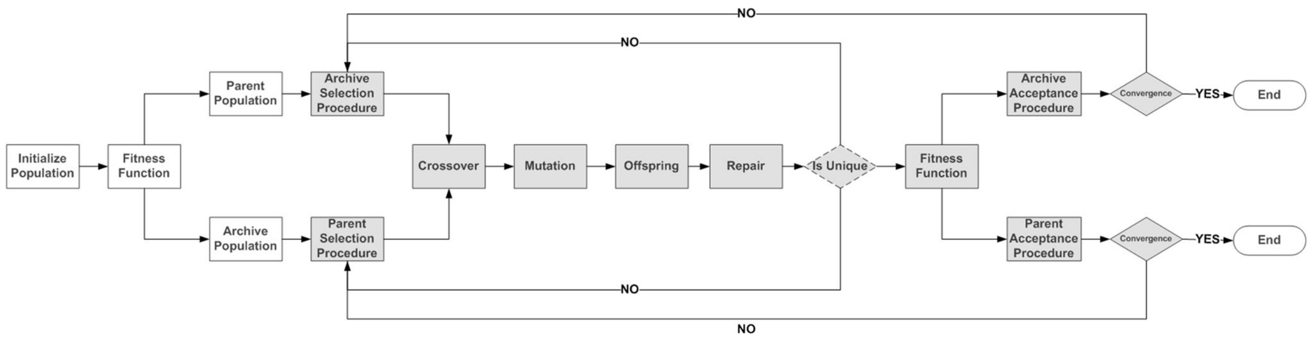


Fig. 5 Tailored ε-MOEA flowchart

Table 2 Optimization parameters (decision variables)

The activity sequence in the DSM (governing process architecture)	$\boldsymbol{S}$
The crashing intensity for each activity i and every iteration k_i	$r_i(k_i)$
The percentage of activity i's duration when it delivers the output in the k_ith iteration for activity j in the k_jth iteration	$o_{D_{ij}(k_i,k_j)}$
The percentage of activity j's duration when it receives the output in the k_jth iteration from activity i in the k_ith iteration	$o_{R_{ij}(k_i,k_j)}$

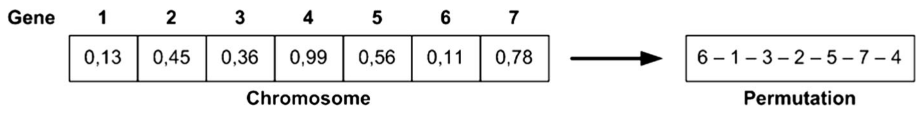


Fig. 6 Chromosome representation example

The overall procedure iterates until any predefined convergence criterion is met, at which point the archive population is expected to contain the Pareto-optimal solution. Defining an appropriate convergence criterion has always been a problem in the design of MOGAs (Laumanns et al. 2002; Kumar and Rockett 2002), and no unique solution has yet been proposed. Since we presume no prior knowledge about the Pareto front, we assume that the ε-MOEA converges when both the absolute number of Pareto solutions remains constant for a certain number of generations and the change in the optimization objectives (duration and cost) does not exceed a predefined threshold (e.g., 5 % of all Pareto solutions) during this period.

4.2 GA components

With the flow of our tailored ε-MOEA established, we now present the problem-specific set-up of its components. Out of the various model parameters, Table 2 shows the variables that are treated as decision variables to be optimized. The parameters $o_{D_{ij}(k_i,k_j)}$ and $o_{R_{ij}(k_i,k_j)}$ determine the amount (duration) of overlapping and related values such as the resulting rework duration, as explained in Sect. 3.

4.2.1 Encoding of activity sequence

$\boldsymbol{S}$ represents a permutation, which is generally represented in a chromosome with integers, although Bean (1994) proposed a *random key* encoding scheme based on real numbers and pointed out its benefits for combinatorial problems. We adopt random keys to encode $\boldsymbol{S}$, treating all optimization parameters as real numbers that can be easily transformed into binary strings with a chosen degree of precision. Typically, the real numbers are used as ascending sorting keys to encode a permutation. These numbers are initially determined randomly and change only under the influence of mutation and crossover. Accordingly, a permutation of length l consists of a vector $\boldsymbol{P} = (r_1, r_2, \ldots, r_l)$ with $r_i \in [0,1]$. Considering the surjective function $\sigma : S_l \to S_l$ with $S_l = \{0, 1, \ldots, l-1\}$, random keys are sorted in ascending order: $r_{\sigma(1)} \leq r_{\sigma(2)} \leq \cdots \leq r_{\sigma(l)}$. A permutation is then encoded by $(\sigma(1), \sigma(2), \ldots, \sigma(l))$. This formalism describes that the key positions within $\boldsymbol{P}$ are ordered according to their absolute value. As an example, consider the chromosome and corresponding encoding shown in Fig. 6. Each gene represents an activity, such that the first segment of a chromosome represents a sequence of activities, $\boldsymbol{S}$.

Theoretically, a key could be generated several times, although the chance is reduced when using sufficient precision for the keys. In this case, a left–right strategy could be used to ensure feasibility. That is, for identical keys (numbers), the key which is further to the left in the chromosome is smaller. Here, each random key is assigned to one distinct gene and consists of 32 bits. Hence, the real number of the key is determined by transforming the binary string to a decimal number divided by 2^{32}.

4.2.2 Encoding of overlapping

Basically, we want the GA to identify the best overlapping intensity between activities in each iteration. The overlapping intensity is a floating point number between 0 and 1. To encode it with bits, we do the same as with the random keys; that is, if we have 8-bit encoding, we transform the 8-bit string into a decimal number and divide it by 2^8. This is done for any overlapping intensity between two activities for all possible iterations.

The overlapping intensity between activities i and j may differ depending on their iteration number, so $o_{D_{ij}(k_i,k_j)}$ and $o_{R_{ij}(k_i,k_j)}$ cannot be encoded in a single bit. Instead, we have to encode these values for all possible combinations of k_i and k_j. Overall, $(\hat{k}_i + 1)(\hat{k}_i + 1)$ combinations exist and must be included in the chromosome. Hence, each combination is binary-encoded through a bit number of any length, n_B, depending on the precision required by the user. An 8-bit encoding $(n_B = 8)$ providing 256 discrete

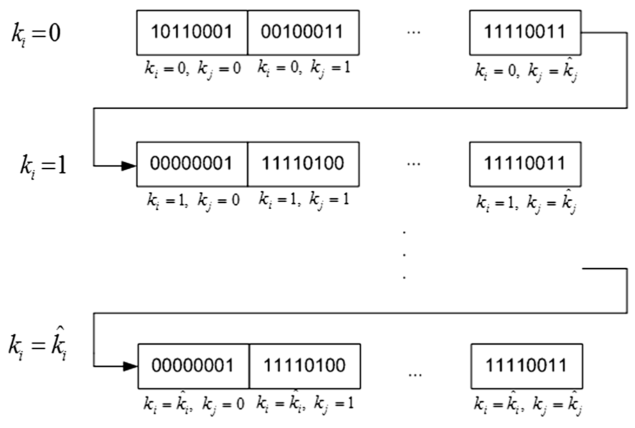


Fig. 7 Illustration of overlapping encoding

intervals between 0 and 100 % overlapping seems sufficient in practice. To reduce computational effort, we only encode values with $o_{D_{ij}(k_i,k_j)} < 1$ and $o_{R_{ij}(k_i,k_j)} > 0$.

The second segment of the chromosome is established by initially identifying the relationships in M_3 and M_5 that meet the aforementioned requirements. Then, we generate an 8-bit number for each (k_i, k_j) pair, as depicted in Fig. 7, and append it to the existing chromosome. To decode the overlapping segment, each of its n_B-bit blocks is transformed to an integer value u in the interval [0, $2^{n_B} - 1$], yielding a value in the interval $\left[m_{3,ij} - \frac{u}{2^{n_B}-1}\left(m_{3,ij} - m_{4,ij}\right), m_{3,ij}\right]$ for $o_{D_{ij}(k_i,k_j)}$ and in the interval $\left[m_{5,ij} + \frac{u}{2^{n_B}-1}\left(m_{6,ij} - m_{5,ij}\right), m_{6,ij}\right]$ for $o_{R_{ij}(k_i,k_j)}$, which represents the third segment.

4.2.3 *Encoding of crashing*

The fourth and last segment of the chromosome addresses crashing. We encode the crashing value of activity i in any of its iterations (up to $\hat{k}_i$) using n_B-bit numbers as well. For evaluation, these bit numbers are transformed to an integer value $u \in [0, 2^{n_B} - 1]$ to generate crashing intensities $r_i(k_i) = \hat{r}_i \frac{u}{2^{n_B}-1}$. Aggregating the four different segments of our data structure so far, we obtain the encoding scheme for a chromosome illustrated in Fig. 8.

4.3 Fitness function

We use time–cost simulation results as the fitness function, although this approach has two major drawbacks. First, evaluating the fitness of each chromosome in a population via simulation is time-consuming. Second, a stochastic simulation provides a "noisy" fitness function because its output may vary slightly each time with identical inputs (e.g., due to stochastic activity durations and iterations). A noisy fitness function will potentially assign different fitness values to identical chromosomes, leading to varying Pareto fronts over successive runs of the ε-MOEA. Moreover, identical chromosomes could become competitors in the population of a single ε-MOEA run due to their different fitness values, allowing them both to take up space in the archive population. In the worst case, the population could consist of multiple instances of the same chromosome with different fitness values. However, we employ two mechanisms to prevent these problems.

The first mechanism is part of the MOEA design: the ε-dominance criterion. If cost and duration outcomes for two identical chromosomes differ by less than ε, then both chromosomes compete for the same hyper-box and one of the two will be discarded, thus ensuring the uniqueness of the other. Nevertheless, the difference between simulation outcomes could exceed ε, particularly if ε is chosen too small. Therefore, the stopping criteria for the number of simulation runs must be coordinated with the choice of ε. If this scenario occurs frequently, then the diversity in the archive, and thus the number of potential Pareto-optimal solutions, decreases. Fortunately, we can easily avoid this case by extending the ε-MOEA with an operator prior to the fitness evaluation (see Fig. 5) to verify the uniqueness of the offspring by comparing its data structure with all members of P and A. Obviously, computational time will increase (since, theoretically, no offspring could be produced for numerous generations), but we nevertheless emphasize the incorporation of this additional mechanism to ensure diversity.

4.4 GA mechanics

4.4.1 *Selection and acceptance procedures*

The ε-MOEA maintains two distinct populations, each requiring a selection operator, and each chromosome has two fitness values, cost and duration. Thus, chromosome selection requires two objective domination checks. We adopt Deb et al.'s (2003) selection procedures for the basic ε-MOEA, described as follows. Within the parent population, P, chromosomes are selected according to a *tournament selection* strategy. During tournament selection, a certain number of chromosomes, depending on the size of the tournament, s, are randomly picked. Generally, the best-fit chromosome wins the tournament with a given probability and overcomes the selection phase. If the tournament contains non-ε-dominated chromosomes (i.e., ε-Pareto optimal), then we select one randomly. Previous research (Goldberg et al. 1991, Goldberg and Deb 1991) found that $s = 4$ performed best on artificial functions. Within A, we just select a chromosome randomly without using a selection operator, because all solutions are Pareto optimal.

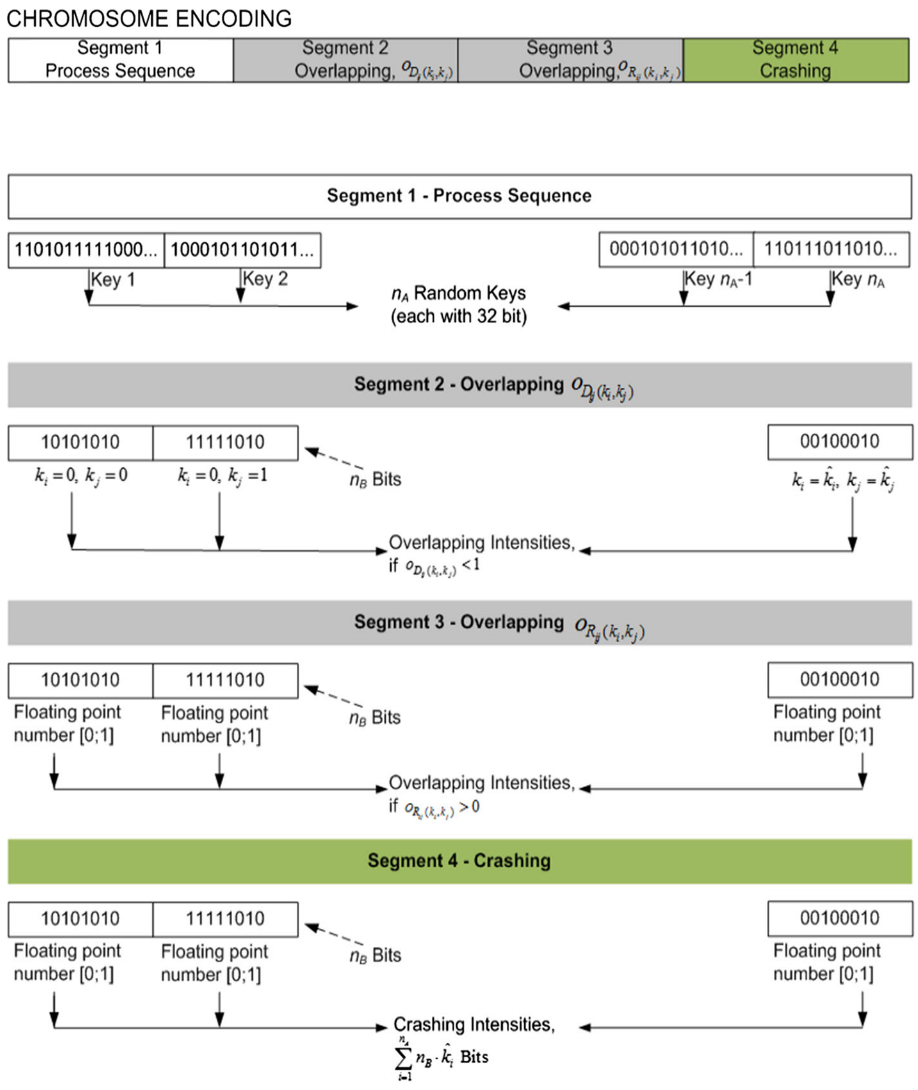

CHROMOSOME ENCODING
Segment 1 Process Sequence
Segment 2 Overlapping, $o_{D_i(k_i,k_j)}$
Segment 3 Overlapping, $o_{R_{ij}(k_i,k_j)}$
Segment 4 Crashing
Segment 1 - Process Sequence
1101011111000...
1000101101011...
000101011010...
110111011010...
Key 1
Key 2
Key n_A-1
Key n_A
n_A Random Keys (each with 32 bit)
Segment 2 - Overlapping $o_{D_{ij}(k_i,k_j)}$
10101010
11111010
00100010
$k_i = 0,\ k_j = 0$
$k_i = 0,\ k_j = 1$
$k_i = \hat{k}_i,\ k_j = \hat{k}_j$
n_B Bits
Overlapping Intensities, if $o_{D_i(k_i,k_j)} < 1$
Segment 3 - Overlapping $o_{R_{ij}(k_i,k_j)}$
Floating point number [0;1]
Overlapping Intensities, if $o_{R_{ij}(k_i,k_j)} > 0$
Segment 4 - Crashing
Crashing Intensities, $\sum_{i=1}^{n_A} n_B \cdot \hat{k}_i$ Bits

Fig. 8 Illustration of chromosome encoding

The ε-MOEA features two acceptance procedures that determine whether the offspring replace any member in P and/or A. If the offspring dominates one or more randomly chosen chromosomes in P, then the dominated member(s) of P are replaced by the offspring. If any member of P dominates the offspring, then it is not accepted. The acceptance procedure for A is more complex, and the reader may refer to the detailed description provided by Deb et al. (2003). In principle, this procedure is similar to the acceptance strategy for P but based on ε-dominance checks.

4.4.2 *Crossover*

Instead of using a single crossover operator on the entire chromosome at once, we apply a separate crossover operator to each segment. This is motivated by the great length of a chromosome (easily comprising several thousand bits) and its time-consuming evaluation via simulation. Applying multiple crossover operations leads to a more intense mixing of the genes, increasing the search speed of the ε-MOEA. While such a strategy risks premature convergence, the computational advantage outweighs any smaller disadvantage in solution quality.

Knjazew (2002) empirically investigated different crossover operators on artificial test functions and claimed superiority for the single-point crossover (Goldberg 1989) over the popular two-point and uniform crossovers. Mainly for this reason, we also incorporated the single-point crossover, which generally works as follows. First, two chromosomes out of the mating pool (i.e., the population after selection) are chosen randomly and undergo crossover with a probability, p_c. Then, a crossover site is chosen randomly, and both chromosomes are sliced at the site into two segments. Finally, the offspring gets two segments from different chromosomes. Figure 9 demonstrates how

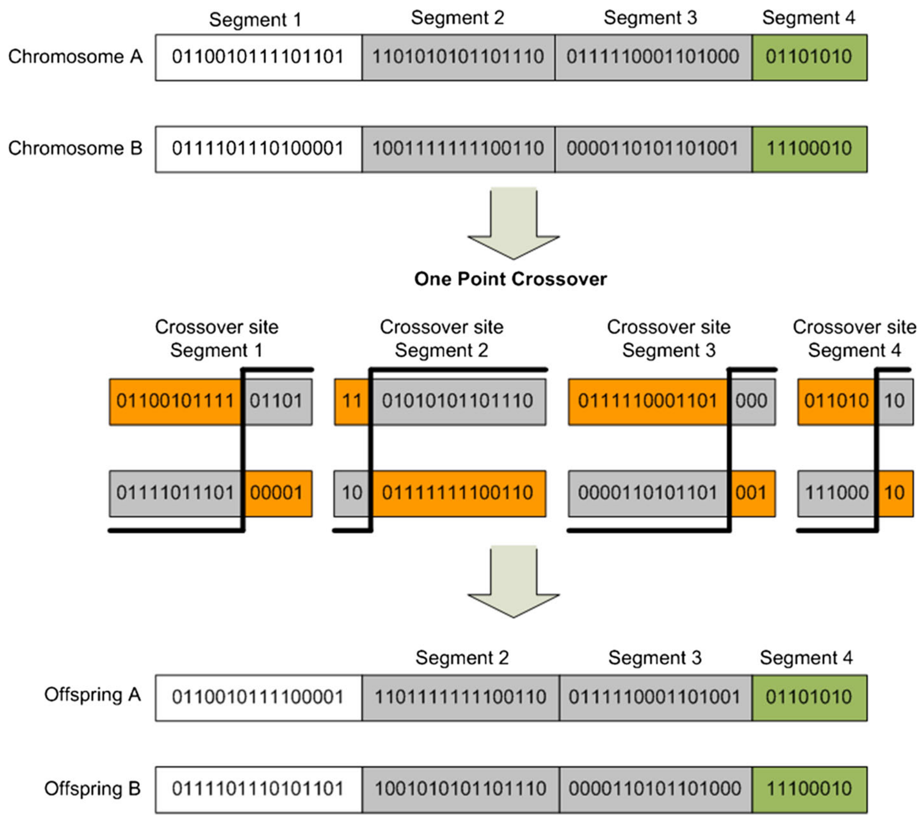


Fig. 9 Demonstration of crossover

the crossover works when applied separately to each of the four sections of our chromosomes.

4.4.3 *Mutation*

As with crossover, we apply mutation to each of the four chromosome segments separately, assuming a certain mutation probability, p_m, which is typically set to a low value. For this purpose, a simple bit-swap mutation (Goldberg 1989)—i.e., swapping a single bit of the corresponding segment—is sufficient.

4.4.4 *Predecessor feasibility*

To ensure feasible activity sequences after crossover and mutation, we must address predecessor constraints. M_4 and M_6 record firm predecessor constraints: Activity j cannot begin before its predecessor i if it requires all input information from i—i.e., if $m_{4,ij} = m_{6,ij} = 0$, assuming $i < j$. Since $m_{6,ij}$ provides an upper bound on $m_{4,ij}$, $m_{6,ij} = 0$ is sufficient to indicate firm predecessor constraints. As random key encoding of the activity sequence prevents duplicate chromosomes, an additional, efficient repair mechanism is required to transform any activity sequence into a precedent-feasible one. A straightforward repair strategy would be to iterate crossover or mutation until all constraints are satisfied. However, depending on the number of constraints, the discrepancy between the immense number of possible permutations ($n_A!$) and the number of feasible solutions could be very large and thus time-consuming. Therefore, we handle the enforcement of predecessor constraints in another way as follows.

Generally, predecessor conflicts do not occur between activities that can be executed concurrently. Assuming we start with an empty activity sequence at time T_0, we can calculate the set of potentially parallel activities at T_0 based on M_6 and subsequently pick an activity out of this set according to a deterministic strategy. For instance, we could scan the (potentially infeasible) activity list of the chromosome from left to right and select the activity number which first matches any activity number in the set of potentially parallel activities. Then, the chosen activity is appended to the end of the feasible activity list and all of its relationships within the network of activities are temporarily deleted. By repeating this procedure until all activities have been assigned to a spot in the activity sequence, we will never violate any predecessor constraints. Meier (2011) describes the repair mechanism in detail and includes its simple algorithm.

As an example, consider the DSM in Fig. 10 representing M_6 and an infeasible activity sequence to the right of it. The M_6 DSM indicates firm precedence relationships between activities with a “0”—e.g., activity 1 must precede activity 3, and activity 2 must precede activities 3 and

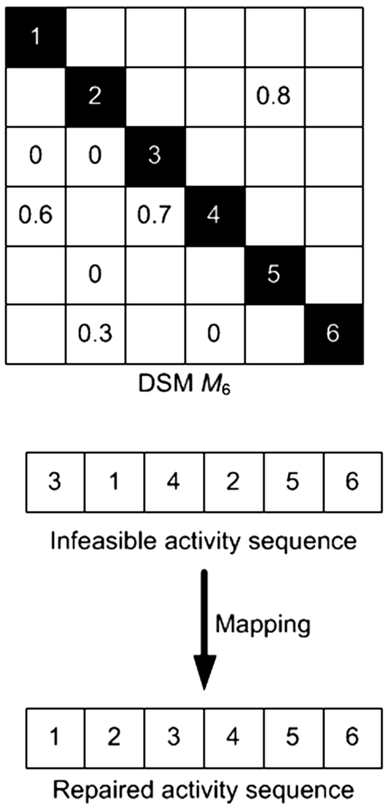


Fig. 10 Preserving predecessor feasibility

5. Hence, the sequence {3–1–4–2–5–6} is infeasible. Applying the repair algorithm leads to the following results. Activities 1 and 2 do not depend on any other activities in the set and thus comprise the initial set of parallel activities. The first value in this set which also occurs in the infeasible sequence is {1}. Thus, the first value of the feasible schedule list must be 1. After deleting the row and column for activity 1 in M_6, the next iteration of the algorithm begins, detecting a new set of parallel activities: {2}. In this set, activity 2 is the earliest one in the infeasible sequence, so it becomes the next activity in the feasible sequence. The row and column for activity 2 are deleted and a new loop starts. Repeating all steps of the algorithm until convergence, we obtain the precedent-feasible sequence {1, 2, 3, 4, 5, 6}.

5 Application: automobile hood development process

We applied the simulation–optimization model to an automotive hood development process. The case study data, based on Zambito (2000) and Yassine et al. (2000), are realistic but disguised to protect company proprietary information. An automobile hood is composed of several individual components. Its development process comprises 43 activities connected via 232 relationships and includes the development not only of the hood components but also of the tooling for assembly and stamping (see activity list and data in Table 3). Also worth mentioning is the occurrence of several verification and validation activities with the potential to cause feedback loops.

The entire set of data for the hood case study is available in Meier (2011) and also in the Appendix (ESM). The activities referring to the development of tooling (activities 26–28) as well as activity 32 can be regarded as the most influential activities. Besides their above-average cost and duration, they feature only minor improvement effects through learning (high values in vector $\boldsymbol{L}$), thus making their rework expensive and long-lasting (see Table 3). According to M_6 (see Appendix of ESM), almost all activities succeeding activity 16 are subject to predecessor constraints. Thus, we expect only marginal differences in process time and cost from the limited degrees of freedom to vary the process architecture.

5.1 Crashing and overlapping

Crashing for the hood process is not completely staff-driven. Instead, some of the aforementioned influential activities may be crashed through the purchase of additional machinery, albeit at a very high price. The max number of iterations per activity was capped at five. This number was determined to be reasonable through sensitivity analysis (Meier 2011).

M_5 and M_6 (see Appendix of ESM) indicate that the possibilities to overlap are actually rather limited in this case. Activities 16–43 require input of essentially final information from their predecessors to begin execution. Therefore, the corresponding cells in M_6 feature low values, while M_5 values are high, suggesting that overlapping will be low. In contrast, the early activities 1-15 are less constrained, thus permitting a considerable amount of overlapping between a few activity pairs. Overall, however, we do not expect a great impact of overlapping on the Pareto front for this case study. The rework penalty for overlapping was calculated according to the linear overlapping function [Eq. (8)] and set to $\alpha_{ij} = m_{1,ij} \cdot m_{2,ij}$—i.e., between 0.05 and 0.25.

5.2 MOEA and simulation parameters

We found that sufficiently stable simulation outputs (for the average cost and duration fitness values of these chromosomes) could be obtained within 1800 simulation runs. Regarding the optimization parameters, although additional constraints (e.g., zero values in M_6) reduce the size of the feasible search space, they also increase the

Table 3 Activity data for the hood development process ($\breve{t}_i$, $\bar{t}_i$, and $\hat{t}_i$ correspond to the "best", "most likely", and "worst" value estimates)

Activities		Time (days)			Costs ($k)			***L***
ID	Name	$\breve{t}_i$	$\bar{t}_i$	$\hat{t}_i$	$\breve{c}_i$	$\bar{c}_i$	$\hat{c}_i$	
1	Strategies for product, mkt, mfg, supply, design, and reusability confirmed (Est. PDL)	0	0	0	0	0	0	–
2	Select power train lineup	0	0	0	0	0	0	–
3	Select materials for all system components	13.5	15	16.5	10	11.3	12.4	0.75
4	Freeze proportions and selected hard points	54	60	66	41	45	49.5	0.75
5	Verify that hard points and structural joint designs are compatible w/program targets	36	40	44	27	30	33	0.75
6	Approve master sections	36	40	44	41	45	49.5	0.85
7	Develop initial design concept (preliminary CAD model)	36	40	44	68	75	82.5	0.10
8	Estimate blank size	0.9	1	1.1	0.7	0.75	0.8	0.10
9	Estimate efforts	0.9	1	1.1	4.1	4.5	4.9	0.10
10	Develop initial attachment scheme	4.5	5	5.5	3.4	3.8	4.13	0.50
11	Estimate latch loads	4.5	5	5.5	20	22.5	24.8	0.10
12	Cheat outer panel surface	9	10	11	10	11.3	12.4	0.50
13	Define hinge concept	18	20	22	20	22.5	24.8	0.50
14	Get prelim. mfg and asy feas. (form, holes, hem, weld patterns, mastic locations, adhesive)	4.5	5	5.5	3.4	3.75	4.13	0.50
15	Perform cost analysis (variable and investment)	1.8	2	2.2	16	18	19.8	0.50
16	Perform swing study	1.8	2	2.2	2	2.3	2.5	0.75
17	Theme approval for interior and exterior appearances (prelim surf available)	13.5	15	16.5	20	22.5	24.8	0.10
18	Marketing commits to net revenue; initial ordering guide	4.5	5	5.5	8.4	9.4	10.3	0.10
19	Program DVPs and FMEAs complete	9	10	11	10	11.3	12.4	0.75
20	Approved theme refined for craftsmanship execution (consistent w/PA objectives)	13.5	15	16.5	30	33.8	37.1	0.10
21	PDN 0—interior and exterior Class 1A surfaces transferred to engineering (±0.5 mm)	2.7	3	3.3	4.1	4.5	5	0.10
22	Conduct cube review and get surface buyoff	18	20	22	54	60	66	0.25
23	Verify mfg and asy feas. (form, holes, hem, weld patterns, mastic locations, adhesive)	9	10	11	64	71	78	0.75
24	Evaluate functional performance (analytically)	13.5	15	16.5	81	90	99	0.50
25	PDN 1—release system design intent level concept to manufacturing	18	20	22	122	135	149	0.50
26	Develop stamping tooling	378	420	462	2835	3150	3465	0.90
27	Develop hemming tooling (if applicable)	57.6	64	70.4	475	528	581	0.75
28	Develop assembly tooling	90	100	110	810	900	990	0.75
29	PDN 2—last Class 1 surface verified and released for major formed parts	18	20	22	176	195	215	0.50
30	PDN 3—final math 1, 2, and 3 data released	18	20	22	189	210	231	0.50
31	CAD files reflect pre-CP verification changes	18	20	22	203	225	248	0.75
32	Make "like production" part and asy tools/ergonomics/process sheets (to extent feasible)	72	80	88	864	960	1056	0.75
33	First CPs available for tuning and durability testing	4.5	5	5.5	57.3	63.8	70.1	1.00
34	Complete CMM analysis of all end items and subassemblies	9	10	11	122	135	149	0.10
35	Perform DV tests (physical)	18	20	22	257	285	314	0.10
36	Verify manufacturing and assembly process capability	4.5	5	5.5	67.5	75	82.5	0.10
37	Complete prelim. ESO for: CP durability testing	13.5	15	16.5	213	236	260	0.10
38	Complete prelim. ESO for: initial set of road tests completed	27	30	33	446	495	545	0.10
39	Complete prelim. ESO for: known changes from CP containable for 1 PP	4.5	5	5.5	77.6	86.3	94.9	0.10
40	Complete prelim. ESO for: design is J1 level—no further changes except No-Blds	4.5	5	5.5	81	90	99	0.10
41	Supplier commitment to support 1 PP w/PSW parts	2.7	3	3.3	50.6	56.3	61.9	0.10
42	Complete prelim. ESO for: Eng. confidence that objectives will be met declared	2.7	3	3.3	52.7	58.5	64.4	0.10
43	Readiness to proceed to tool tryout (TTO), 1 PP and Job #1	9	10	11	182	203	223	0.10

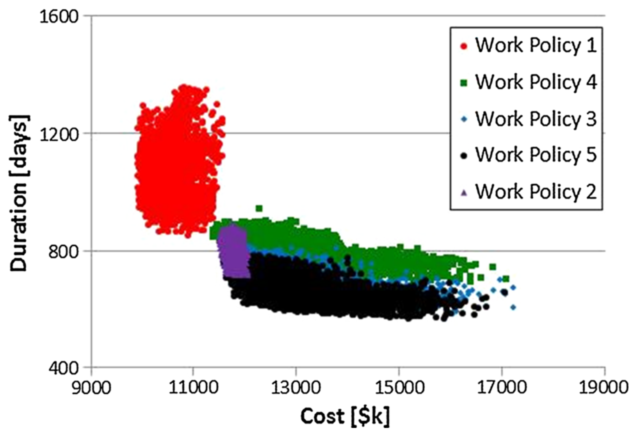


Fig. 11 Random solutions and the Pareto front for the hood process using work policies P_1–P_5 (this figure includes crashing and overlapping)

probability of generating redundant solutions (due to our repair mechanism). Thus, we had to use a fairly large number of generations for the ε-MOEA (while holding population size constant). We let the ε-MOEA run with 10,000 8-bit encoded chromosomes for up to 15,000 generations, simulating the fitness of each chromosome 1800 times per generation.[1]

5.3 Optimization results

Figure 11 shows the Pareto fronts along with 10,000 randomly generated solutions (i.e. initial population of chromosomes for the ε-MOEA), which roughly approximate the solution space, for each of the five work policies. The figure shows the time–cost differences between random and Pareto-optimal (by tracing the lower-left boundary of the contours) solutions for the various work policies. This figure also emphasizes the impact of work policy rules and crashing/overlapping on process duration and cost. For instance, P_1 has more variation with respect to duration (i.e. wider duration range), whereas P_3, P_4, and P_5 tend to be relatively robust in duration but much more variable in cost. We also note that Policy 2 is dominated by policies 3, 4, and 5 and has the least leverage (i.e., duration and cost variance). Finally, when we compare the hood case results with the theoretically approximated solution spaces shown in Fig. 12 (Meier et al. 2015), we can see that the hood development process is in agreement with the general trend for artificial processes (depicted in Fig. 12).[2]

To examine the Pareto front (of Fig. 11) in more detail, we decomposed it into three sectors, as shown in Fig. 13: sector 1 includes the cheapest but slowest processes, generated exclusively by work policy P_1; sector 2 contains Pareto solutions produced by P_3 and P_4; and sector 3 features the collectively fastest processes with the highest cost (generated entirely by P_5). P_2 is dominated by P_3–P_5. The evaluation of certain process parameters for each sector is presented in Table 4. Figure 13 provides PD mangers with clear guidelines for which work policy to use given a specific priority for process duration or cost. That is, cost-sensitive mangers can use policies 1, 3, and 4, while duration-sensitive ones can use P_5.

Figure 13 features an interesting discontinuity in the Pareto front with respect to cost between sectors 1 and 2. This discontinuity is not the result of different feedback densities or longer feedback cycles (distance to the DSM diagonal). Rather, it is due to the work policy rule regarding the parallel execution of non-adjacent activities. Whereas P_2–P_5 allow the simultaneous execution of non-adjacent activities, P_1 does not. Thus, the processes in sectors 2 and 3 (obtained through the application of P_3–P_5) exhibited more parallelism and consequently a higher number of costly reworks. The actual amount of rework depends not only on the process architecture (e.g., number of feedbacks and their locations) but also on the work policy.

Finally, we explored the sensitivity of the Pareto front to the model's complexity (i.e., the inclusion of process architecture, overlapping, and crashing parameters). These effects are clear in Fig. 14, which plots the overall Pareto front (for all work policies) for the hood process assuming work policy P_3 and using different aspects of model sophistication (i.e. architecture, crashing, and overlapping). Using individual managerial levers (architecture, overlapping, and crashing) independently does not traverse the full solution space (Pareto front) compared to the more sophisticated model inclusive of all model elements simultaneously. These results demonstrate the need for a sophisticated model that accounts for all of these features of realistic processes. A comparison of the statistical summary measures for architecture, crashing, and overlapping curves is also shown in the lower part of Fig. 14. It is evident from this summary table that process architecture is not the greatest lever for cost and duration due to the high amount of firm predecessor constraints, which limit the generation of distinct process architectures. Instead, modifications of the individual crashing intensities have the greatest effect on process lead time and cost in this hood development process. (The same analysis was performed for P_4 and P_5, yielding similar conclusions to P_3.)

[1] The entire GA run time for the case study took around 15 min on a workstation equipped with Intel i7 quad-core CPU and 16 GB RAM.

[2] However, the exact values of time and cost differ in both figures due to the difference in activity durations (and rework probabilities and impacts) between the hood development process and the artificial processes.

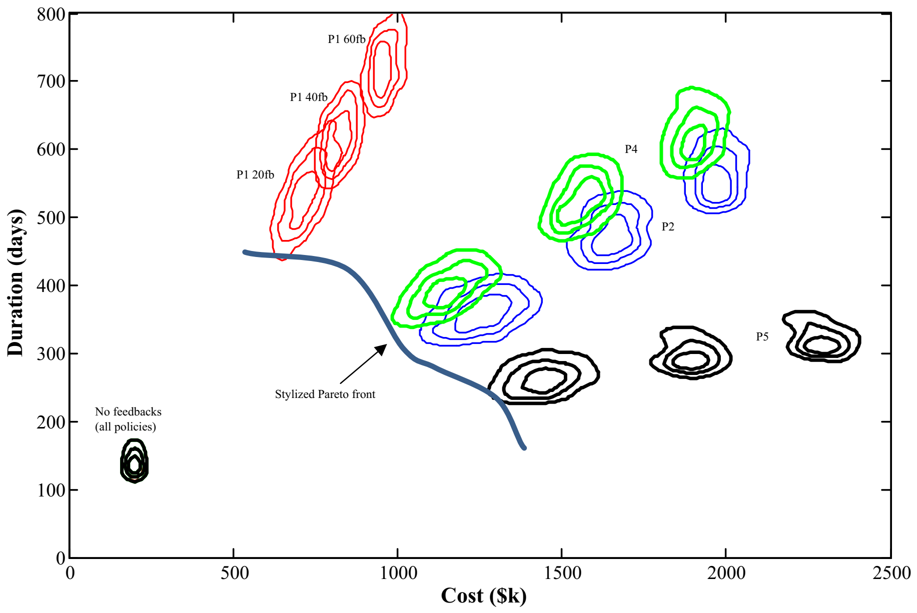


Fig. 12 The theoretically approximated objective spaces from Meier et al. (2015)

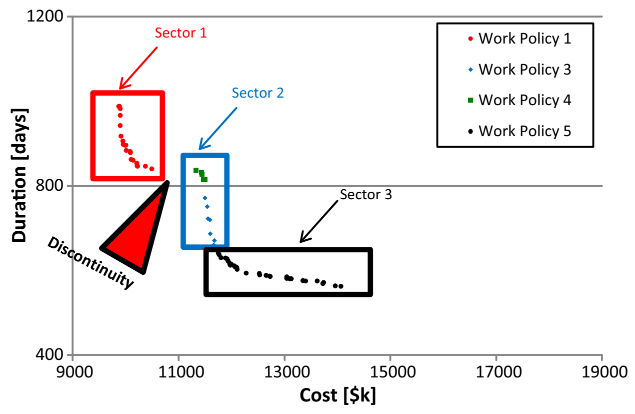


Fig. 13 Overall Pareto front for the hood process using work policies P_1–P_5. Note that P_2 is dominated by P_3–P_5

6 Conclusion

The time–cost trade-off problem for PD processes constitutes a problem of great practical relevance, because PD processes involve an enormous amount of expensive resources employed over a long period of time. Consequently, managers are anxious to minimize cost and duration simultaneously. Past research (primarily based on crashing an acyclical process) pointed out that cost-reducing strategies extend process duration and time-cutting methods increase cost. A trade-off problem thus arises, making efficient process planning mandatory to avoid schedule and cost overruns. In fact, the literature on the time–cost trade-off problem extensively studied the effects of time- and cost-reducing methods and even proposed optimization strategies to identify the set of best trade-offs. But literature to date lacked in analyzing the impact of iterations on these time-/cost-minimizing strategies and on the time–cost solutions themselves—although iteration is pervasive in PD processes and greatly affects time and cost.

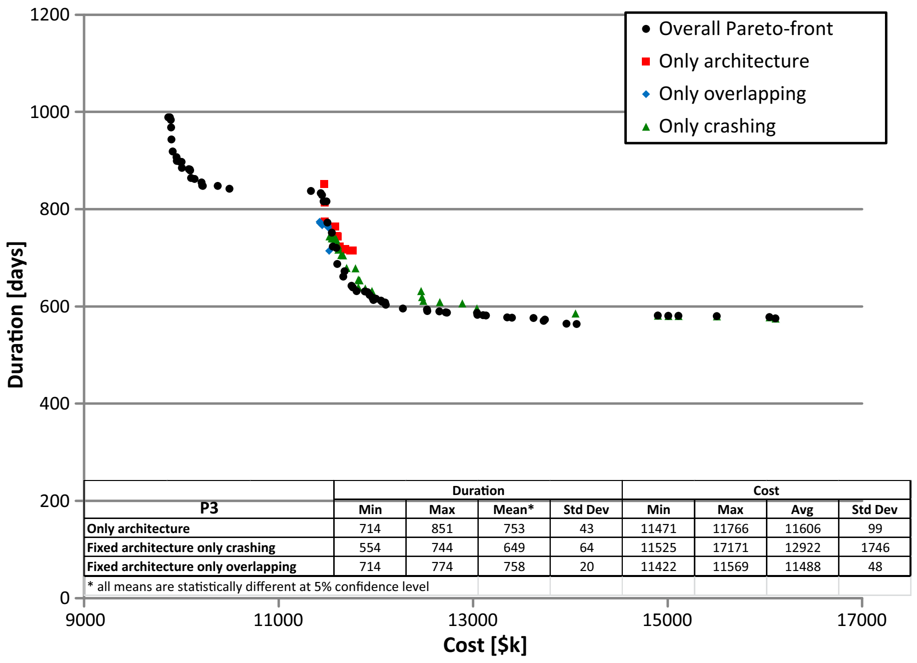
This research gap was the motivation for this paper: to find the most efficient set of trade-offs while considering many aspects at once: crashing, overlapping, cyclical process architecture, and work policy. For this purpose, we used a new PD process model that accounts for feedbacks in the process and jointly considers the process parameters influencing time and cost. We tailored an optimization technique, an ε-MOEA, to this problem and found the Pareto front solutions for a PD process. We applied the new simulation–optimization model to an industrial case study to explore the effects of several work policies.

First, the time–cost simulation–optimization model presented in this paper highlights the need for a sophisticated process model in real-world development processes. Assuming different levels of model complexity (i.e., different work policies along with process architecture, with and without overlapping, and crashing), the corresponding Pareto fronts for the case study development process showed substantial differences in duration and cost. This supports our claim that the combined consideration of several fundamental process parameters for modeling, simulation, and subsequent optimization is pivotal to gaining "globally" Pareto-optimal processes.

Furthermore, the work policies can result in different outcomes for process cost and duration. Interestingly, no work policy appeared to be superior to the others in both the cost and duration dimensions. Instead, a time–cost trade-off arises due to the choice of work policy. Therefore,

Table 4 Selected process parameters for the three disjoint sectors of the Pareto front

Parameter	Sector 1	Sector 2	Sector 3
Dominating work policy	P_1	P_3	P_5
Average number of feedback marks in the DSM	65.4	63.8	77.7
Average distance of a feedback mark to the DSM diagonal	7.16	6.94	9.04
Weighted average distance of a feedback mark to the DSM diagonal (feedback probability times distance to diagonal)	1.92	1.88	2.34
Average number of reworks	59.3	155.3	158.9
Average number of partial reworks	51.3	60.2	81.2
Average crashing intensity (1st iteration) for any activity (min: 0, max: 0.15)	0	0.07	0.08
Average crashing intensity (1st iteration) for critical activities 26, 27, 28, and 32 (min: 0, max: 0.21)	0	0.05	0.09
Average crashing intensity (1st iteration) for most critical activity 26 (min: 0, max: 0.25)	0	0.07	0.18
Average intensity for overlapping available (1st iteration) between two activities (min: 0.86, max: 1.0)	0	0.92	0.93
Average overlapping intensity (1st iteration) between two activities (min: 0, max: 0.2)	0	0.09	0.11



P3	Duration				Cost			
	Min	Max	Mean*	Std Dev	Min	Max	Avg	Std Dev
Only architecture	714	851	753	43	11471	11766	11606	99
Fixed architecture only crashing	554	744	649	64	11525	17171	12922	1746
Fixed architecture only overlapping	714	774	758	20	11422	11569	11488	48

* all means are statistically different at 5% confidence level

Fig. 14 Comparison of the overall Pareto front (all work policies) for the hood process with the Pareto fronts assuming work policy P_3 for different aspects of model sophistication

we suggest extending the time–cost trade-off problem by a further managerial lever (beside crashing, overlapping, and process architecture), namely work policy. The substantial time–cost differences for the work policies should motivate engineering managers to appreciate the impact of this managerial instrument.

Despite its relative sophistication, however, the proposed model does not account for specific resource constraints (only general ones in terms of cost). Specific resources such as staff, machinery, or facilities—just to name a few—must be assigned to PD activities, are typically limited, and typically cause holdups during process execution. Hence, we suggest extending the model by accounting for specific resource constraints and studying their impact on the Pareto front and the efficacy of crashing and overlapping.

In addition to cost and duration, technical performance or quality is another dimension to be considered for PD projects (Liberatore and Pollack-Johnson 2009). We encourage extension of the model, simulation, and optimization with measures of technical performance of the product to be developed. Obviously, quality is expected to increase with a growing number of iterations in the process, and the current model accounts for iterations, albeit only in a probabilistic sense. Thus, we indirectly considered quality issues. However, there remains opportunity to incorporate quality-related parameters explicitly in the model, perhaps along the lines of Lévárdy and Browning (2009).

Finally, we recommend further research on closed-form analysis for an *approximation* of time and cost for cyclic processes (e.g., Nasr et al. 2015). A realistic proxy could be used as a deterministic objective function for the multi-objective optimization and is essential for very large problem sets involving several hundred or thousand activities. Otherwise, using simulation as objective functions for large problem sets would require a high computational effort; GAs are very slow and they may hinder the production process if a high fidelity model is used. However, research toward closed-form solutions is very challenging, and we are not sanguine about its success for arbitrary processes. Nevertheless, the potential benefits of a closed form are worth trying, and our optimized simulation model in this paper helped to identify the most salient characteristics on which to focus in seeking a simplified, closed-form approximation. Particularly, process architecture, crashing, and overlapping intensities, in addition to work policy decisions, are necessary components of any realistic and practical PD process model.

Acknowledgment The first author would like to thank the Bavarian Science Foundation. The second author is grateful for support from the University Research Board (URB) program at the American University of Beirut. The third author is grateful for support from the Neeley Summer Research Award Program from the Neeley School of Business at TCU.

References

Abdelsalam HME, Bao HP (2007) Re-sequencing of design processes with activity stochastic time and cost: an optimization-simulation approach. J Mech Des 129(2):150–157

Adler P, Mandelbaum A, Nguyen V, Schwerer E (1995) From project to process management: an empirically-based framework for analyzing product development time. Manag Sci 41(3):458–484

Baldwin AN, Austin S, Hassan TM, Thorpe A (1999) Modelling information flow during the conceptual and schematic stages of building design. Constr Manag Econ 17:155–167

Bean JC (1994) Genetic algorithms and random keys for sequencing and optimization. J Comput 6(2):154–160

Berthaut F, Pellerin R, Perrier N, Hajji A (2014) Time-cost trade-offs in resource-constraint project scheduling problems with overlapping modes. Int J Proj Organ Manag 6(3):215–236

Browning TR (2001) Applying the design structure matrix to system decomposition and integration problems: a review and new directions. IEEE Trans Eng Manag 48(3):292–306

Browning TR, Eppinger SD (2002) Modeling impacts of process architecture on cost and schedule risk in product development. IEEE Trans Eng Manage 49(4):428–442

Browning TR, Ramasesh RV (2007) A survey of activity network-based process models for managing product development projects. Prod Oper Manag 16(2):217–240

Browning TR, Yassine AA (2016) Managing a portfolio of product development projects under resource constraints. Decis Sci (forthcoming)

Brucker P, Drexl A, Mohring R, Neumann K, Pesch E (1999) Resource-constrained project scheduling: notation, classification, models, and methods. Eur J Oper Res 112:3–41

Bruni ME, Beraldi P, Guerriero F (2015) The stochastic resource-constrained project scheduling problem. In: Schwindt C, Zimmermann J (eds) Handbook on project management and scheduling, vol 2. Springer, Berlin, pp 811–835

Cho S-H, Eppinger SD (2005) A simulation-based process model for managing complex design projects. IEEE Trans Eng Manage 52(3):316–328

Coello CAC, Pulido GT, Lechuga MS (2004) Handling multiple objectives with particle swarm optimization. Evol Comput IEEE Trans 8(3):256-279

Coello CAC, Van Veldhuizen DA, Lamont GB (2002) Evolutionary algorithms for solving multi-objective problems, vol 242. Kluwer Academic, New York

Cohen I, Golany B, Shtub A (2007) The stochastic time–cost tradeoff problem: a robust optimization approach. Networks 49(2):175–188

Cooper KG (1993) The rework cycle: benchmarks for the project manager. Proj Manag J 24(1):17–21

Coverstone-Carroll V, Hartmann JW, Mason WJ (2000) Optimal multi-objective low-thrust spacecraft trajectories. Comput Methods Appl Mech Eng 186(2–4):387–402

De P, Dunne EJ, Ghosh JB, Wells CE (1995) The discrete time-cost trade off problem revisited. Eur J Oper Res 81:225–238

Deb K (2009) Multi-objective optimization using evolutionary algorithms. Wiley, Chichester

Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197

Deb K, Mohan M, Mishra S (2003) Towards a quick computation of well-spread pareto-optimal solutions. In: Evolutionary multi-criterion optimization. Second international conference, EMO 2003, pp 222–236

Deb K, Mohan M, Mishra S (2005) Evaluating the ε-domination based multi-objective evolutionary algorithm for a quick computation of Pareto-optimal solutions. Evol Comput 13(4):501–525

Deckro RF, Hebert JE, Verdini WA, Grimsrud PH, Venkateshwar S (1995) Nonlinear time/cost tradeoff models in project management. Comput Ind Eng 28(2):219–229

Doerner KF, Gutjahr WJ, Hartl RF, Strauss C, Stummer C (2008) Nature-inspired metaheuristics for multiobjective activity crashing. Omega 36(6):1019–1037

Eppinger SD, Browning TR (2012) Design structure matrix methods and applications. MIT Press, Cambridge

Fonseca CM, Fleming PJ (1998) Multiobjective optimization and multiple constraint handling with evolutionary algorithms-part1: a unified formulation. IEEE Trans Syst Man Cybernet Part A Syst Hum 28(1):26–37

Fujita K, Hirokawa N, Akagi S, Kitamura S, Yokohata H (1998) Multi-objective optimal design of automotive engine using genetic algorithms. In: Proceedings of 1998 ASME design engineering technical conferences

Gerk JEV, Qassim RY (2008) Project acceleration via activity crashing, overlapping, and substitution. IEEE Trans Eng Manag 55(4):590–601

Goldberg DE (1989) Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, New York

Goldberg DE, Deb K (1991) A comparative analysis of selection schemes used in genetic algorithms. In: Foundations of genetic algorithms, vol 1, pp 69–93

Goldberg DE, Deb K, Thierens D (1991) Toward a better understanding of mixing in genetic algorithms. In: Proceedings of the 4th international conference on genetic algorithms

Hanne T (1999) On the convergence of multiobjective evolutionary algorithms. Eur J Oper Res 117(3):553–564

Hartmann S, Briskorn D (2010) A survey of variants and extensions of the resource-constrained project scheduling problem. Eur J Oper Res 207(1):1–14

Hazır Ö, Erel E, Günalay Y (2011) Robust optimization models for the discrete time/cost trade-off problem. Int J Prod Econ 130(1):87–95

Hazır Ö, Haouari M, Erel E (2015) Robust optimization for the discrete time-cost tradeoff problem with cost uncertainty. In: Schwindt C, Zimmermann J (eds) Handbook on project management and scheduling, vol 2. Springer, Berlin, pp 865–874

Helbig S, Pateva D (1994) On several concepts for ε-efficiency. OR Spektrum 16(3):179–186

Herroelen W, Leus R (2005) Project scheduling under uncertainty: survey and research potentials. Eur J Oper Res 165:289–306

Huang E, Chen S-JG (2006) Estimation of project completion time and factors analysis for concurrent engineering project management: a simulation approach. Concurr Eng 14(4):329–341

Karniel A, Reich Y (2009) From DSM based planning to design process simulation: a review of process scheme verification issues. IEEE Trans Eng Manag 56(4):636–649

Kline SJ (1985) Innovation is not a linear process. Res Manag 28(2):36–45

Knjazew D (2002) OmeGA: a competent genetic algorithm for solving permutation and scheduling problems. Kluwer Academic Publishers Group, Norwell

Krishnan V, Ulrich KT (2001) Product development decisions: a review of the literature. Manag Sci 47(1):1–21

Krishnan V, Eppinger SD, Whitney DE (1997) A model-based framework to overlap product development activities. Manag Sci 43(4):437–451

Kumar R, Rockett P (2002) Improved sampling of the pareto-front in multiobjective genetic optimizations by steady-state evolution: a pareto converging genetic algorithm. Evol Comput 10(3):283–314

Laumanns M, Thiele L, Deb K, Zitzler E (2002) Combining convergence and diversity in evolutionary multiobjective optimization. Evol Comput 10(3):263–282

Lévárdy V, Browning TR (2009) An adaptive process model to support product development project management. IEEE Trans Eng Manag 56(4):600–620

Liberatore MJ, Pollack-Johnson B (2009). Quality, time, and cost tradeoffs in project management decision making. In: Portland international conference on management of engineering & technology, 2009. PICMET 2009, pp 1323–1329

Meier C (2011) Time-cost tradeoffs in product development processes, Doktor-Ingenieurs (Dr.-Ing.) thesis, Technische Universität München, Munich, Germany

Meier C, Yassine AA, Browning TR (2007) Design process sequencing with competent genetic algorithms. J Mech Des 129(6):566–585

Meier C, Browning TR, Yassine AA, Walter U (2015) The cost of speed: work policies for crashing and overlapping in product development projects. IEEE Trans Eng Manag 62(2):237–255

Nasr W, Yassine A, Abou Kasm O (2015) An analytical approach to estimate the expected duration and variance for iterative product development projects. Res Eng Des 27(1):55–71

Poloni C, Giurgevich A, Onesti L, Pediroda V (2000) Hybridization of a multi-objective genetic algorithm, a neural network and a classical optimizer for complex design problems in fluid dynamics. Comput Methods Appl Mech Eng 186(2–4):403–420

Roemer TA, Ahmadi R (2004) Concurrent crashing and overlapping in product development. Oper Res 52(4):606–622

Roemer TA, Ahmadi R, Wang RH (2000) Time-cost trade-offs in overlapped product development. Oper Res 48(6):858–865

Rudolph G, Agapie A (2000) Convergence properties of some multi-objective evolutionary algorithms. In: Congress on evolutionary computation (CEC 2000), pp 1010–1016

Sargent RG (1999) Validation and verification of simulation models. In: Winter simulation conference, Phoenix, AZ, 5–8 Dec

Shaja AS, Sudhakar K (2010) Optimized sequencing of analysis components in multidisciplinary systems. Res Eng Des 21(3):173–187

Smith RP, Eppinger SD (1997) Identifying controlling features of engineering design iteration. Manag Sci 43(3):276–293

Smith RP, Morrow JA (1999) Product development process modeling. Des Stud 20(3):237–261

Srinivas N, Deb K (1994) Multiobjective optimization using nondominated sorting in genetic algorithms. Evol Comput 2(3):221–248

Tavares VL, Ferreira JA, Coelho JS (2002) A comparative morphologic analysis of benchmark sets of project networks. Int J Project Manag 20(6):475–485

Vanhoucke M (2015) Generalized discrete time-cost tradeoff problems. In: Schwindt C, Zimmermann J (eds) Handbook on project management and scheduling, vol 1. Springer, Berlin, pp 639–658

Wolpert DH, Macready WG (1997) No free lunch theorems for search. IEEE Trans Evol Comput 1(1):67–82

Yassine A, Braha D (2003) Complex concurrent engineering and the design structure matrix method. Concurr Eng Res Appl 11(3):165–176

Yassine A, Whitney D, Lavine J, Zambito T (2000) Do-it-right-first-time (DRFT) approach to DSM restructuring. In: ASME international design engineering technical conferences (Design theory & methodology conference), Baltimore, MD, 10–13 Sept

Zambito T (2000) Using the design structure matrix to structure automotive hood system development. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA

Zhuang M, Yassine AA (2004) Task scheduling of parallel development projects using genetic algorithms. In: ASME international design engineering technical conferences. (Design automation conference), Salt Lake City, Sept 28–Oct 2

Zitzler E, Laumanns M, Thiele L (2002) SPEA2: improving the strength pareto evolutionary algorithm for multiobjective optimization. In: Evolutionary methods for design, optimisation, and control, Barcelona, Spain, pp 19–26