Parametric Critical Path Analysis for Event Networks with Minimal and Maximal Time Lags

Joost van Pinxten*, Marc Geilen*, Martijn Hendriks‡, Twan Basten*‡
*Department of Electrical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands
‡Embedded Systems Innovation, Netherlands Organisation for Applied Scientific Research, Eindhoven, The Netherlands

***Abstract*—High-end manufacturing systems are cyber-physical systems where productivity depends on the close cooperation of mechanical (physical) and scheduling (cyber) aspects. Mechanical and control constraints impose minimal and maximal time differences between events in the product flow. Sequence-dependent constraints are used by a scheduler to optimize system productivity while satisfying operational requirements. The numerous constraints in a schedule are typically related to a relatively small set of parameters, such as speeds, lengths, or settling times. We contribute a parametric critical path algorithm that identifies bottlenecks in terms of the feasible parameter combinations. This algorithm allows analysis of schedules to identify bottlenecks in terms of the underlying cause of constraints. We also contribute a way to find Pareto-optimal cost-performance trade-offs and their associated parameter combinations. These results are used to quantify the impact of relaxing constraints that hinder system productivity.**

I. Introduction

High-end manufacturing systems are cyber-physical systems (CPSs) composed of several cooperating machines, which have strict timing requirements between their operations. These requirements can be modelled as minimal and maximal timing constraints between events. Such constraints often occur due to some physical or computational process, and influence the productivity of the system. Groups of constraints typically share an underlying cause, such as a motor that actuates multiple components simultaneously. The interrelationship of the parameters influencing the constraints are explored during the design phase and one or several trade-offs are selected. It is of interest to quantify the relationship of (component) parameters to the system performance.

Scheduling activities according to a set of constraints is common in engineering [1], research and design [2] projects, and project management [3]. Identifying and alleviating performance bottlenecks is a core activity for improving the performance of schedules. The identified bottlenecks, which are critical paths in the event network, are important hints for reducing the earliest completion time of the schedules. We show that such bottleneck analysis techniques can also be extended to manufacturing CPSs.

This article was presented in the International Conference on Hardware/Software Codesign and System Synthesis 2018 and appears as part of the ESWEEK-TCAD special issue. This is the Author-version of the accepted paper. DOI: 10.1109/TCAD.2018.2857360

In this paper, we show that it is possible and useful to extend the critical path analysis [3] technique with parametric analysis, so that the interdependence between parameters is taken into account. In parametric analysis, the problem is to find solutions, e.g. critical paths, for all possible values of the parameters. Our approach first finds constraints that characterize all feasible combinations of parameters and then, for all feasible combinations of parameters, a symbolic critical path in the form of an expression in terms of the parameters of the event network. We show the effectiveness of our approach and observe that the scalability is primarily determined by the time to find the extremes of a polyhedron. Our method provides system designers with a quantitative approach to evaluate interaction between design parameters. We also show a method to efficiently determine the parameter combinations that yield Pareto-optimal performance-cost trade-offs. System designers can then take informed decisions selecting trade-offs between parameters.

Section II positions our work in the body of existing work. Section III introduces terminology and notation of event networks. Section IV first shows how to find critical paths and extends the terminology with parameters that can describe physical relations; it then shows how to find expressions for critical paths in parametrized event networks. In Section V we show the applicability of the method on two different manufacturing systems: the Twilight system [4], and a Large Scale Printer [5], [6]. These two examples show that we can quantitatively relate relaxation of parameters to productivity gains of the machine. Section VI concludes the paper.

II. Related work

Event and activity networks have been used in project modelling and other kinds of scheduling problems to model minimal and maximal time lags between events or activities. We use the project modelling approach started by Roy [7], and Kerbosch and Schell [8], and further developed by Elmaghraby [9]. The time lags make it possible to describe constraints on the feasible time ranges between events in a CPS. In this paper, we use minimal and maximal time lags to express constraints between events in terms of parameters.

The Critical Path Method (CPM) [3] and generalized precedence relations (GPRs) [9] are used to model event networks, and also to find critical paths in such networks. The CPM and GPRs use the difference between the occurrence of an event in an as-soon-as-possible (ASAP) and as-late-as-possible (ALAP) schedule to determine the slack of

events/activities. Those with zero slack are critical, and any delay in any of these critical events leads to an according delay in the completion of the network. It does not, however, give the designer any insight into how much system productivity is gained when critical constraints are relaxed.

Some critical path methods assume stochastic time lags and are applied to observations of systems, such as the shifting bottleneck detection [10]. In this method, the machines that are most often active are regarded as the bottleneck. This method gives an indication of the bottlenecks over time. However, this approach does not show the impact of deadlines, as imposed by maximal time lags, and can therefore hide the true interaction of different machines, especially when deep pipelining behaviour occurs. Stochastic time lags have also led to investigations of the stochastic criticality index, such as shown in [11]. In this paper, we assume that time lags are deterministic, and focus on the interaction between parameters.

As described in [12], when cost slopes are known for shortening activities (also known as *crashing*), CPM can be used with Linear Programming to give the efficient trade-offs between project duration and project (direct) cost. There are several methods available that can be applied to networks that do not have maximal time lags (Fondahl's method, Siemens method) [12]. The work of [9] generalizes these crashing methods such that maximal time lags can be taken into account. These works, however, do not take into account that the underlying cause for each constraint can be shared. That is, each constraint is independently relaxed, and it is assumed that other relationships remain the same. Ignoring interdependencies between parameters in CPSs leads to incorrect conclusions from a system perspective.

Approaches using parametric analysis for single or multiple parameters have been developed for many kinds of problems. The modified Bellman-Ford-Moore (BFM) algorithm of [13] efficiently finds all critical paths for the feasible values of a *single parameter* when edge weights are integer. Our approach allows an arbitrary number of parameters, with rational numbers as values and weights. Graphs with rational values can be used in the approach of [13] after multiplying all expressions with an integer factor such that all rational numbers become integer values.

Most methods for multiple parameters are based on the observation that parameter combinations occur in convex regions [14], [15], [16]. Parametric analysis of synchronous data-flow graphs has been used to determine, for example, the parameter combinations for which the same maximum cycle mean expression holds [16], [17]. The divide-and-conquer approach from [16] shows that performance expressions can be found even when the parameters do not necessarily take integer values. In this paper we prove that this parametric analysis can be applied to event networks with maximal time lags.

Some work has been carried out to identify for which parameter combinations parametrized models have feasible solutions. The work of Elmaghraby [9] introduces *flexibility* of an activity/constraint in an activity network as the amount of time it can be increased/decreased such that all constraints can still be met in a schedule; i.e. it indicates when networks become infeasible. The convexity of parameter regions has also been used by [18] to determine for which parameter combinations a periodic fixed priority system is schedulable. The interaction between tasks and their relation to system utilization are modelled with fixed priority scheduling rules, and precedence relations between tasks are explicitly not allowed in their work.

Several parametric methods have also been developed for timed automata [15], [19]. In [15] a subclass of parametric timed automata has been identified for which it can decide on the emptiness problem when automaton variables are upper and lower bounded by parameters. A parameter may either occur in some lower bounds, or in some upper bounds, but never in both. The approach of [15] can detect parameter combinations for which the system does not have conflicting requirements, i.e., is feasible. Our extension of the parametric method allows parameters both in lower and upper bounds (minimum and maximum time lags). In [19], the feasibility of activating a set of real time tasks is checked through parametric timed automata. Similar to the work of [18] the approach does not allow precedence constraints, as it models interaction between tasks through periodic activation patterns.

We extend this existing body of work by finding all feasible parameter combinations and their associated performance characterization through parametric temporal analysis of event networks with minimal and maximal time lags. The quantitative analysis allows finding the Pareto-optimal performance-cost trade-offs.

III. Event networks

We adopt the following notation from the work of Elmaghraby and Kamburowski [9]: an *event* is identified by $k \in \mathbb{E} = \{0, \ldots, N+1\} \subset \mathbb{N}$, and is represented by a node in a network graph. The source node 0 represents the start event and the sink node $N+1$ represents the finish event. An example network is shown in Fig. 1. The realization time of event k is denoted t_k. The realization time of the source is fixed to 0.

A *minimal time lag* relation from event i to event j is captured in the standard form: $t_j \geq t_i + D(i,j)$. Such a relation is represented by an edge $(i,j) \in \mathbb{E}^2$ from event i to event j with weight $D(i,j) \in \mathbb{R}$. The interpretation of the minimal time lag $D(i,j)$ depends on its sign. I.e., we allow maximal time lags $L(i,j)$ from event i to event j, by transforming them into standard form [8]:

$$\begin{aligned} t_j \leq t_i + L(i,j) &\iff t_i \geq t_j - L(i,j) \\ &\iff t_i \geq t_j + D(j,i) \end{aligned}$$

That is, a *positive maximal* time lag $L(i,j)$ from i to j is equal to a *negative minimal* time lag $D(j,i) = -L(i,j)$ from j to i. Task graphs with minimal and maximal time lags may introduce cyclic time constraints. To ensure that all events are related to the source, we assume that the source has minimal time relations with zero lag to all other events. Each event, analogously, must be related to the sink, and has a minimal time relation with zero lag to the sink node.

A network graph is equivalent to a system of inequalities. A graph is feasible if and only if there exists a solution

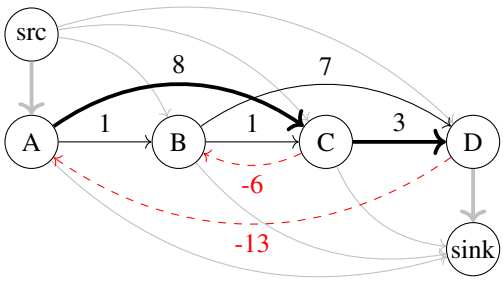


Fig. 1: Example event graph with given time lags between events. Positive minimal time lags are shown with black edges. Negative minimal time lags are shown in dashed red edges and represent maximal time lags or, in other words, relative deadlines. The gray edges have zero time lag. The only critical path src-ACD-sink is shown with thick edges.

to its corresponding system of inequalities. Equivalently, a network is infeasible if and only if it has a cycle with positive cumulative weight. The earliest *feasible* realization time $\underline{t}_k$ of an event k is the smallest number for which the system of inequalities is feasible. $\underline{t}_k$ is equal to the weight of the longest path from the source node to node k in network G:

$$\underline{t}_k = \max_{a \in paths(0 \to k)} \sum_{(i,j) \in a} D(i,j)$$

Definition 1 (makespan)**.** *The makespan M of a graph is the earliest possible realization time of the sink node, $M = \underline{t}_{N+1}$.*

$\bar{t}_k$ is the latest possible realization time of node k. The latest possible realization $\bar{t}_k$ of event k is found by subtracting the longest path from k to the sink from the makespan:

$$\bar{t}_k = M - \max_{a \in paths(k \to N+1)} \sum_{(i,j) \in a} D(i,j)$$

For any graph, the latest possible realization time of the sink node is equal to the makespan: $\bar{t}_{N+1} = M$. The possible realization intervals $[\underline{t}_i, \bar{t}_i]$ of the example in Fig. 1 are shown in Fig. 2. Notably, the realization of event B can start at earliest 2 time units after A, even though the relation AB allows B to start one time unit after A. Path AB has length 1, while path ACB has length $8 - 6 = 2$, i.e. the earliest realization of B is bounded by the maximal time difference between B and C. The makespan of the graph in Fig. 1 is 11, corresponding to the longest path src-ACD-sink. The latest realization of B is 4; the longest path from B to the sink is 7, and it can therefore occur in the range $[2, 4]$ without affecting the makespan.

A relation from i to j has slack $S(i,j) = \bar{t}_j - (\underline{t}_i + D(i,j))$: the difference between the latest realization of the target event j and the earliest realization of the originating event i, taking into account the minimal time between i and j. The relation slack determines how much a relation can be increased without affecting the realization time of the sink node. In the example network, AB has a slack of 3. The relation $D(A,B)$ is allowed to increase by 3 time units before it starts affecting the makespan of the network. AC has slack 0, and it cannot be increased by any amount without affecting the realization time of the sink node.

TABLE I: Path lengths of the example network in Fig. 3.

Path	length	critical if and only if
ABCD	$2q + p$	$p \geq q + 5 \wedge 2q \geq p + 5$
ABD	$3q + 5$	$p \leq q + 5 \wedge p + q \geq 5 \wedge 3q \geq 2p$
ACD	$2p + 5$	$2q \leq p + 5 \wedge 3q \leq 2p \wedge 3p \geq 2q + 5$
ACBD	$-p + 2q + 10$	$p + q \leq 5 \wedge 3p \leq 2q + 5$

Definition 2 (critical relation)**.** *A relation between i and j is critical if and only if there is no relation slack: $S(i,j) = 0$, or equivalently: $\underline{t}_i + D(i,j) = \bar{t}_j$.*

Definition 3 (critical path)**.** *A critical path is a simple sequence of connected critical relations (i.e. no event is traversed more than once), originating from the source and leading to the sink of the network.*

A longest path in a network defines the makespan of the network, and as such is equal to a critical path in the network. At least one critical path exists in the network as the latest allowed realization of the sink node is equal to its earliest possible realization. The makespan of the example network is 11 time units, and the critical path in the example network is src-ACD-sink. In this example, this is the only path which originates in the source, ends in the sink, and has zero slack for each edge in the path.

IV. Parametric Critical Path Analysis

In this section, we generalize the event model such that the time lags can be linear functions of parameters, and then show how to perform critical path analysis on such models. Parameters can correspond to the speed of operations, reconfiguration times, transport times, etc. An example parametrized event network is shown in Fig. 3, where relations between events are denoted as functions of parameters p and q, such as $D(A,C) = 2p + 5$. The example in Fig. 1 is an instance of Fig. 3 with $(p,q) = (3,1)$. There are four simple paths in Fig. 3 from the source to the sink for non-negative values of p and q. The path lengths are listed in Table I, where the last column indicates for which part of the parameter space the path length is maximal. The expression in the last column is constructed as follows. Let $\mathcal{P} = \{\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3, \boldsymbol{e}_4\}$ be the path expressions. Then $\boldsymbol{e}_i \in \mathcal{P}$ is critical at point $\boldsymbol{p} = (p,q)$ if and only if $\boldsymbol{e}_i(\boldsymbol{p}) = \max_{\boldsymbol{e} \in \mathcal{P}}(\boldsymbol{e}(\boldsymbol{p}))$.

A. Overview of the approach

Our approach finds the critical paths for all feasible parameter combinations after it finds which combinations of parameter values have feasible results. The results of our approach for the example event network are illustrated in Fig. 4. Fig. 4 shows four regions in the 2-d parameter space. In one region (blue), the network is infeasible. In each of the three other regions, different paths in the event network are critical. Each path, and hence each region, is associated with its own critical path expression. The makespan of the example in Fig. 1 is 11 time units, which corresponds to the

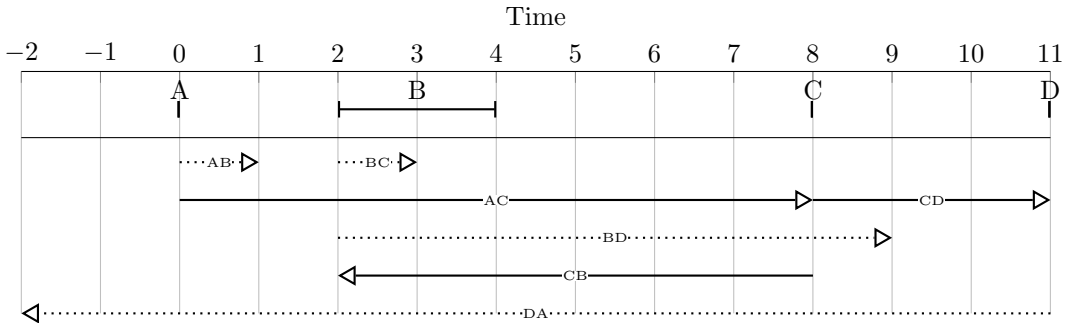


Fig. 2: Possible realization intervals for event graph shown in Fig. 1, shown in the top lane. The constraints are shown in the bottom five lanes as arrows, originating from the earliest realization time of its source event having length $D(i, j)$. Critical and non-critical constraints are represented with solid and dashed lines respectively.

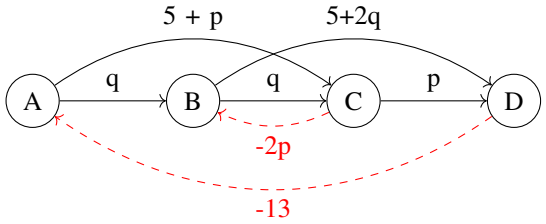


Fig. 3: Example parametrized event network with minimal and maximal time lags. Fig. 1 is an instance of this parametrized network with $(p, q) = (3, 1)$. The source, sink, and their relations have been omitted.

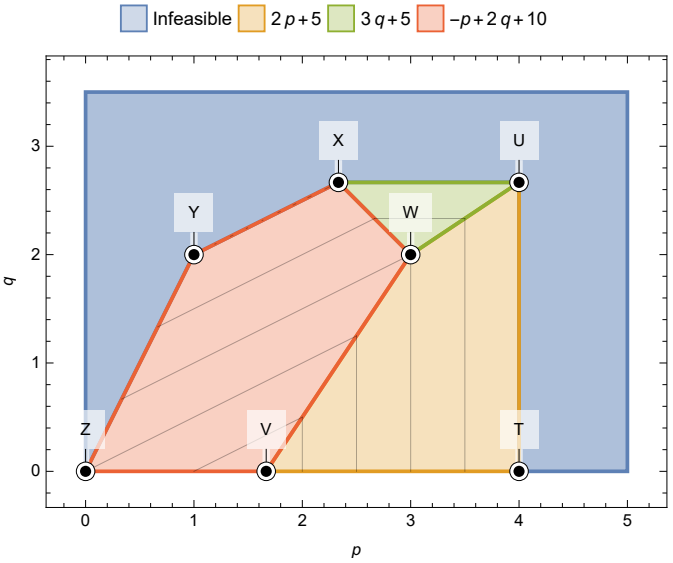


Fig. 4: Critical path expressions and infeasible parameter combinations for the example in Fig. 3. The gray contour lines inside a polyhedron are iso-makespan lines. The extremes of the polyhedra are labelled with a letter.

expression $2p+5$ at $(p, q) = (3, 1)$, which is part of the yellow region in the figure. Parameters often relate to costs, e.g., faster transport is more expensive. Thus the identified regions enable trading off cost and performance. We provide further insight into cost/performance trade-offs in Section IV-F. The rest of this section shows how such expressions can automatically be found for all parameter combinations in a specified range.

We start from an existing algorithm that can be applied to non-parametrized event networks with maximal time lags (Section IV-B). As a first contribution, we use a symbolic version of this method to find a critical path expression for one combination of the parameters (Section IV-C). We prove that such critical path expressions, as well as the conditions for which the network is feasible, are convex. Our second contribution is explained in detail in Section IV-D, where we introduce an algorithm that removes all infeasible parameter combinations from a parameter space for a given event network. This algorithm provides the conditions under which feasible solutions exist. Our third contribution is that we introduce parametric analysis for event networks to find critical path expressions for each of the feasible parameter combinations (Section IV-E). We show that the divide-and-conquer method of [16] can be extended to find all such expressions relatively efficiently. Finally, as a fourth contribution, we show that we can find the Pareto-optimal trade-offs for linear cost functions, and provide some interpretation of the expressions found as a basis for optimization in Section IV-F.

B. Critical Path Analysis for event networks with minimal and maximal time lags

We first show how to calculate earliest and latest realizations for networks with fixed minimal and maximal time lags for a particular parameter point. Event realization times are calculated efficiently using a longest-path algorithm such as the Bellman-Ford-Moore (BFM) algorithm [20], [21]. BFM has been used in the (E)MPM algorithm [7], [8], as well as activity networks with GPRs [9]. These algorithms also detect infeasibility in event networks.

The BFM algorithm can provide both critical paths and infeasible cycles. A network is infeasible when a cycle with positive weight exists in the network. Kerbosch and Schell [8] and Sedgewick and Wayne [22] showed that it is possible to find an arbitrary critical path by doing an ASAP analysis and keeping track of the relaxations per node. In case the network is feasible, the BFM algorithm keeps track of a *parent*

tree [22] that defines which of the incoming nodes was last used to relax a node. This data structure can be efficiently updated while finding the ASAP times of the events in the network. A critical path can be found by following the parent relationships from the sink node up the parent tree until the source node has been reached. Or, in case the network is infeasible, a positive cycle is found by tracking back the parent graph from the sink node until a node is encountered that has been visited already.

C. Parametrized event networks

In networks resulting from CPSs, the relations between events are typically derived from physical or control constraints that the CPS needs to adhere to [4], [5], [23]. It is often possible to parametrize the relations in the event network such that they are linear combinations of some (physical) parameters. The travelling time t of a product at a velocity v, for example, can be modelled as the linear combination of the required displacement x and displacement rate $\delta_x = \frac{1}{v}$, resulting in the linear relation $t = \frac{x}{v} = x \cdot \delta_x$. In the example parametrized network (Fig. 3) p and q are two such displacement rates, for example of two robotic arms moving at different speeds.

We first show how to assess the feasibility and the makespan of an event network for a particular parameter through the critical path analysis detailed in Section IV-C1. We then prove that such critical path and infeasibility expressions relate to geometrical half-spaces and form convex polyhedra in Section IV-C2. We use the half-space representation in a feasibility detection algorithm (Section IV-D), after which we apply a divide-and-conquer to find all expressions in the feasible space (Section IV-E).

1) Relating critical paths and positive cycles to parameters: The parametric weight $D(i,j)(\boldsymbol{p})$ of a relation $r = (i,j)$ is a parametric affine expression $\boldsymbol{e}(\boldsymbol{p}) = \boldsymbol{b}_r \cdot \boldsymbol{p} + c_r$, where $\boldsymbol{p}$ is a vector of parameters, $\boldsymbol{b}_r$ is a vector of weights, c_r is a constant and $\cdot$ denotes a vector inner product. For d parameters, the affine function $\boldsymbol{e}$ can be represented by a vector consisting of coefficients $\boldsymbol{b}_r$ and the constant $\boldsymbol{c}_r$ in the $\mathbb{R}^{d+1}$ space, and the function evaluation at a parameter point $\boldsymbol{p} \in \mathbb{R}^d$ of $\boldsymbol{e}$ becomes $\boldsymbol{e}(\boldsymbol{p}) = [\boldsymbol{b}_r \ c_r] \cdot [\boldsymbol{p} \ 1]$. Consequently, the makespan for a parameter point $\boldsymbol{p}$ becomes:

$$M(\boldsymbol{p}) = \max_{a \in paths(0 \to N+1)} \left(\sum_{(i,j) \in a} D(i,j)(\boldsymbol{p}) \right)$$

A path in the parameter space is critical if the path has the maximal weight of all paths for some combination of parameters. The makespan M can be expressed as a linear combination $\boldsymbol{b} \cdot \boldsymbol{p} + c$ of the occurrences of parameters in some critical path P for all $\boldsymbol{p}$ for which the expression of P is critical:

$$M(\boldsymbol{p}) = \sum_{(i,j) \in P} D(i,j)(\boldsymbol{p}) = \sum_{r=(i,j) \in P} \boldsymbol{b}_r \cdot \boldsymbol{p} + c_r = \boldsymbol{b} \cdot \boldsymbol{p} + c,$$

where $\boldsymbol{b} = \sum_{r \in P} \boldsymbol{b}_r$ and $c = \sum_{r \in P} c_r$.

A critical path expression can only be found in case the network has feasible solutions. Otherwise, instead of a critical path expression, we extract a positive cycle C causing infeasibility at a parameter point $\boldsymbol{p}$ by retrieving the expression V of that cycle C:

$$V(\boldsymbol{p}) = \sum_{r \in C} \boldsymbol{b}_r \cdot \boldsymbol{p} + c_r = \boldsymbol{b} \cdot \boldsymbol{p} + c > 0,$$

where $\boldsymbol{b} = \sum_{r \in C} \boldsymbol{b}_r$ and $c = \sum_{r \in C} c_r$.

Any point $\boldsymbol{p}$ for which a positive cycle exists (i.e. $V(\boldsymbol{p}) > 0$) is infeasible. This inequality therefore defines a half-space for which no feasible solutions exist for the network. When the cycle BCB in Fig. 3 of length $q - 2p$ is greater than zero, the network is infeasible; we therefore recognize that all points (p, q) from the parameter space for which $q - 2p > 0$ are infeasible.

2) Convex polyhedra of critical path expressions: We adapt Proposition 5 from [16] to show that the critical path expressions form convex polyhedra in the parameter space. The proof of that proposition also applies to Proposition 1.

Proposition 1. *$\{\boldsymbol{p} \in \mathbb{R}^d \mid M(\boldsymbol{p}) = \boldsymbol{e}(\boldsymbol{p})\}$ is a convex polyhedron for any critical path expression $\boldsymbol{e}$.*

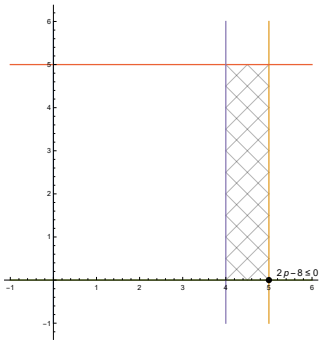
Any half-space s can be described by an affine expression $\boldsymbol{e}$ such that $s(\boldsymbol{e}) = \{\boldsymbol{p} \in \mathbb{R}^d \mid \boldsymbol{e}(\boldsymbol{p}) \geq 0\}$. A convex polyhedron can be represented as the intersection of a finite set of half-spaces, each of which is represented by an expression or vector, i.e. a polygon can be represented by a subset $h \subset \mathbb{R}^{d+1}$. We lift the function s to sets of half-spaces: $s(h) = \bigcap_{\boldsymbol{e} \in h} s(\boldsymbol{e})$. A convex polyhedron can therefore be described by a set of expressions that correspond to half-spaces. Proposition 2 helps to determine the feasible parameter combinations.

Proposition 2. *If all corners of a convex polyhedron with half-space representation h are feasible, then all points in $s(h)$ are feasible.*

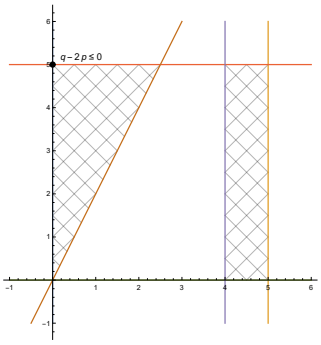
Proof. If there would be a parameter point $\boldsymbol{p}$ in the polyhedron (with half-space representation h) that is infeasible, then at that point a positive cycle with cumulative weight $V(\boldsymbol{x}) = \boldsymbol{b} \cdot \boldsymbol{x} + c$ must exist such that $V(\boldsymbol{p}) > 0$. The inequality $\boldsymbol{b} \cdot \boldsymbol{x} + c$ corresponds to a half-space that contains $\boldsymbol{p}$. The half-space containing $\boldsymbol{p}$ includes at least one corner point of h. Therefore, that corner point must be infeasible too. This is a contradiction and therefore proves the proposition. □

D. Determining feasible parameter combinations

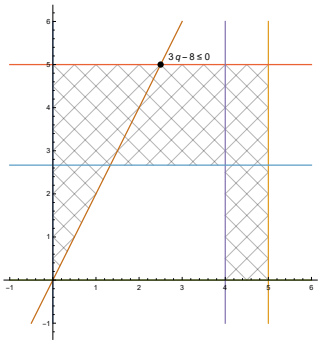
Consider a situation where a convex polyhedron (possibly the entire parameter space) is explored for infeasible points, with the goal to prune these points. Algorithm 1 removes infeasible half-spaces using the information in a positive cycle found in the graph. If BFM (in EVALUATE) finds an infeasible corner point of the polyhedron, it removes the half-space described by the positive cycle expression $V(\boldsymbol{p}) > 0$, as defined in Section IV-C1. It does so by adding the restriction $-V(\boldsymbol{p}) \geq 0$ to the polyhedron. The algorithm continues in a recursive fashion with the remaining space until all corner points are feasible. This process is illustrated in Fig. 5a-5d. The result is returned through the recursive calls. From Proposition 2 it follows that all parameter combinations in that (possibly empty) polyhedron are feasible.



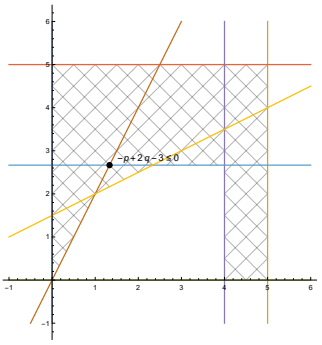
(a) Alg. 1: remove infeasible region (ACDA) $2p - 8 > 0$



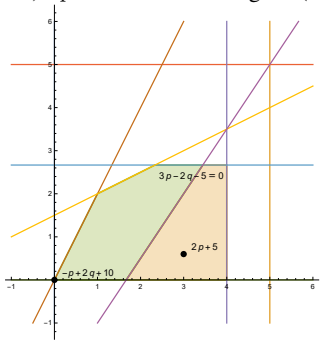
(b) Alg. 1: remove infeasible region (BCB) $q - 2p > 0$



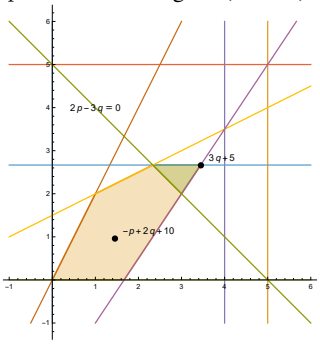
(c) Alg. 1: remove infeasible region (ABDA) $3q - 8 > 0$



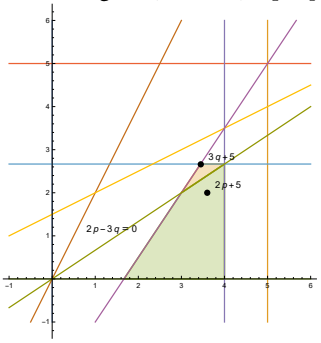
(d) Alg. 1: remove infeasible region (ACBDA) $2q - p - 3 > 0$



(e) Alg. 2: expression $-p + 2q + 10 \neq 2p+5$ at $(3, 3/5)$, split by $3p-2q-5 = 0$



(f) Alg. 2: expression $-p + 2q + 10 \neq 3q+5$ at $(16/11, 24/25)$, split by $-p-q+5 = 0$



(g) Alg. 2: expression $2p + 5 \neq 3q + 5$ at $(36/10, 2)$, split by $2p - 3q = 0$

Fig. 5: Illustration of our method for the example in Figure 3. p and q both range from 0 to 5.

Algorithm 1 Determine feasible parameter combinations

```
1: function DetermineFeasiblePoints(event network G, convex polyhedron CP)
2:     for each corner c_i of s(CP) do
3:         feasible, cycle_expression = Evaluate(G, c_i)
4:         if ¬ feasible then
5:             CP' = CP ∪ {−cycle_expression}
6:             return DetermineFeasiblePoints(G, CP')
7:     return CP // All points in CP are feasible
```

E. Divide-and-conquer

The polyhedron obtained from Algorithm 1 contains only feasible points, and allows the use of the divide-and-conquer approach presented in [16] (Algorithm 2). That approach finds all critical path expressions for all parameter combinations by partitioning the parameter space into smaller and smaller pieces until an expression is found that holds for all corner points of the polyhedron. When an expression $\boldsymbol{e}_i$ found in the interior of a polyhedron does not hold for one of its corner points c with expression $\boldsymbol{e}_c$, the polyhedron is split across the hyperplane $\boldsymbol{e}_i = \boldsymbol{e}_c$; i.e., to one polyhedron the equation $\boldsymbol{e}_i - \boldsymbol{e}_c \geq 0$ is added and to the other $\boldsymbol{e}_i - \boldsymbol{e}_c \leq 0$. For the example in Fig. 3, the algorithm starts from the feasible parameter combinations found in Fig. 5d. It finds expression $M(p, q) = 2p + 5$ for the point $(p, q) = (3, 3/5)$, which does not hold at corner $(0, 0)$ as $M(0, 0) = 10$ (Fig. 5e). The expression $M(p, q) = -p + 2q + 10$ is found at $(0, 0)$, and we therefore split the polyhedron into two pieces, by the equation $-p+2q+10 = 2p+5$. This process is repeated two more times as illustrated in Fig. 5f and 5g.

Once an expression is found that holds for all corner points of a polyhedron, the algorithm stops exploring that part of the parameter space and continues with another polyhedron. For the two polyhedra found in Fig. 5f the algorithm finds two expressions that hold in these polyhedra respectively. It continues with a polyhedron for which no expression has been found yet in 5g. The results of Algorithm 1 and 2 are combined in Fig. 4.

The divide-and-conquer approach of [16] is reproduced in Algorithm 2. This algorithm requires a feasible solution to be found for an arbitrary point inside the polyhedron, along with an expression that holds for that point.

Unfortunately, no efficient algorithm is possible for generating the extreme points (i.e. the corners) of a convex polyhedron [24]. When all parameter combinations are feasible, Algorithm 2 initially needs to transform a non-degenerate set of half-spaces in d dimensions into 2^d corner points. Enumerating an exponential number of corner points is expensive, but it turns out to be feasible for a moderate number of parameters (e.g. ¡ 15). Furthermore, the number of calls to DivideConquer is at least the number of expressions to be identified, and possibly more. The polyhedron for $3q + 5$ for the example of Fig. 3 is found by combining the results of two different sub-problems (Fig. 5f, 5g). Each call to

Algorithm 2 Divide-and-conquer

```
1: function DIVIDECONQUER(event network G, convex polyhedron CP)
2:     p = random point in s(CP)
3:     e_c = FIND_EXPRESSION(G, p)
4:     for each corner c_i of s(CP) do
5:         e_i = FIND_EXPRESSION(G, c_i)
6:         if e_c(c_i) ≠ e_i(c_i) then
7:             // Divide CP into two disjoint polyhedra
8:             CP_1 = CP ∪ {e_i − e_c}
9:             CP_2 = CP ∪ {e_c − e_i}
10:            S_1 = DIVIDECONQUER(G, CP_1)
11:            S_2 = DIVIDECONQUER(G, CP_2)
12:            // Return all expressions found in polyhedra
13:            return S_1 ∪ S_2
14:    return {e_c} // Expression holds for each corner
```

FIND_EXPRESSION costs at most $O(|E||R|)$ where $|E|$ is the number of events (nodes), and $|R|$ the number of relations (edges).

In comparison with the modified BFM algorithm of Levner et al. [13], we see that their approach is more efficient for a single parameter, but it cannot take into account multiple parameters simultaneously. Their modified algorithm finds the critical paths in $O(|R|^2 \cdot |E|)$ time when the parameter coefficient on the relations is chosen from $\{-1, 0, 1\}$. For arbitrary integer values, the approach takes $O((b \cdot |R|)^2 \cdot |E|)$, where b is the largest parameter coefficient occurring on any edge. Our method solves a generalized version of the problem of [13], where parameters can be rational numbers, and multiple parameters are taken into account. For a given number of parameters, the running time of our approach depends on the number of regions to be found, the time to evaluate a particular parameter point, and the time to convert the half-space representation into corner points. The number of regions that will be found depends on the parameter range, which is selected by the designer.

F. Assessing the parameter regions

The results of our approach can be used to determine the quantitative impact of decreasing/increasing parameters. The rate of change per unit of reduction is found in the gradient ∇ of the expression in the region around the current set-point or working point. The gradient component for a single parameter i at a point with expression $\boldsymbol{e}$ is equal to the aggregate contribution $\boldsymbol{b}_i$ of parameter i to the critical path, i.e., $\nabla_i \boldsymbol{e} = \boldsymbol{b}_i$. In Fig. 4, $\nabla M(3,1) = (2,0)$ and $\nabla M(1,1) = (-1,2)$. Changing the parameter by Δ_i will impact the makespan by $\boldsymbol{b}_i \cdot \Delta_i$, as long as the critical path expression still holds for the new parameter point, i.e., it lies in the same region.

We assume that the makespan and a cost function, say $C(p,q) = p - 2q$ in the example of Fig 4, are both to be minimized. Typically, the optimal cost is found at a different parameter combination than the optimal makespan, and therefore trade-offs exist between makespan and cost. For

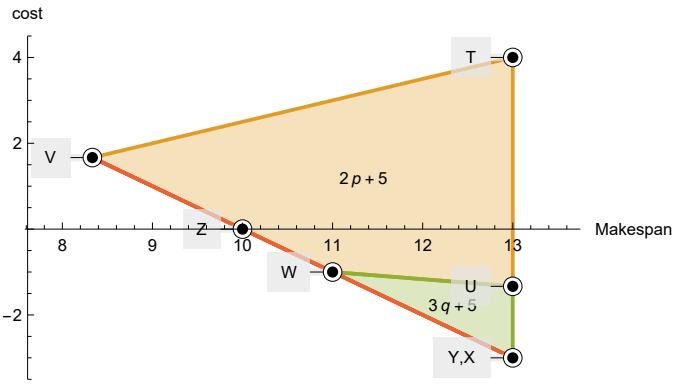


Fig. 6: Cost-makespan trade-off space for $c(p,q) = p - 2q$ for Fig. 3. The labelled extremes correspond to those in Fig. 4.

example, starting from point $X = (7/3, 8/3)$, any point on the line segment XW in Fig. 4 closer to W has shorter makespan $M = -p + 2q + 10$ and higher cost $C = p - 2q$. On the other hand, the cost-makespan trade-off at X is preferred over any of the cost-makespan points found on the line XU. The parameter selection can be improved by following the gradient such that the cost decreases or stays the same, and the makespan decreases, or vice versa. A parameter combination is called dominated iff it is worse in at least one of the two aspects and not better in the other than some other parameter combination.

In general, we want to find all Pareto-optimal cost-performance trade-offs in the parameter space. The Pareto-optimal parameter combinations are those for which no other parameter combinations exist that dominate it. For an arbitrary linear cost function $C(\boldsymbol{x}) = \boldsymbol{\alpha} \cdot \boldsymbol{x}$, the cost and makespan can be computed for each point in the space. All Pareto-optimal parameter combinations can be found by projecting the parameter-space to the cost-performance trade-off space, finding the trade-offs in that space and translating them back to the parameter combinations. Transforming a polyhedron from the parameter space to the cost-makespan space, where $M(\boldsymbol{x}) = \boldsymbol{b} \cdot \boldsymbol{x} + c$ is the associated expression, is defined by:

$$T(\boldsymbol{x}) = \begin{bmatrix} M(\boldsymbol{x}) \\ C(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} \boldsymbol{b} & c \\ \boldsymbol{\alpha} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{\alpha} \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} c \\ 0 \end{bmatrix} = \boldsymbol{P_1 x} + \boldsymbol{p_2}$$

Applying the affine transformation T to all points of a convex polyhedron in the parameter space yields a new convex polyhedron in the cost-makespan space. It is sufficient to transform the extreme points of the polyhedron to obtain the polyhedron in the cost-makespan space. An example of such a projection is shown in Figure 6. The Pareto-optimal corner points are efficiently identified in the cost-makespan space by applying algorithms such as Simple Cull [25] to the finite set of corner points. Each point in a convex polyhedron that lies on a line between two adjacent Pareto-optimal corner points, is Pareto-optimal as well.

Transformation T is not necessarily bijective, i.e., multiple parameter combinations $\boldsymbol{x}$ may map to the same makespan-cost trade-off (M, C). Each Pareto-optimal point (M, C) in a convex region in the trade-off space can be translated back

to the parameter space through the (pseudo-)inverse of the transformation associated with that convex region:

$$\boldsymbol{x} = T^{\dagger}(M, C) = \boldsymbol{P_1}^{\dagger}\left(\begin{bmatrix} M \\ C \end{bmatrix} - \boldsymbol{p_2}\right)$$

Each Pareto-optimal point (M, C) on the Pareto-optimal line segments maps to the space $(T^{\dagger}(M, C) \oplus K(\boldsymbol{P_1})) \cap E$ where E is the polyhedron corresponding to the transformation T, and $K(P1)$ is the kernel of $P1$. For two subspaces A and B, $\oplus$ is their extension: $A \oplus B = \{\boldsymbol{a} + \boldsymbol{b} \mid \boldsymbol{a} \in A, \boldsymbol{b} \in B\}$. Each extension of a kernel and a point in the parameter space is a subspace of the parameter space.

In the example of Fig. 6, the line segment XW of region $3q + 5$ has a transformation T with full rank $\boldsymbol{P_1}$, and each point on the line segment therefore corresponds uniquely to a point on the line segment XW in the parameter space of Fig. 4. Similarly, the line segment VW for region $2p + 5$ maps uniquely to the line segment VW in the parameter space. However, all points of the 2-D polyhedron for the expression $-p+2q+10$ in Fig. 4 have been mapped to points on the line VX in the trade-off space. Its corresponding $\boldsymbol{P_1}$ has less than full rank. The kernel of transformation $\boldsymbol{P_1}$ is the same for all of the Pareto-optimal line-segments, and these line segments map to the intersection of two half-spaces. Each point in the region $-p + 2q + 10$ is therefore a Pareto-optimal parameter combination: for example, the point Y in the cost-makespan space maps to a line $2p + q + d = 0$ such that the line passes through $Z = T^{\dagger}(13, -3) = (8/5, 4/5)$ in the parameter space, i.e. $d = -4$. On this line, the combinations of p and q are such that the makespan remains the same, i.e. they coincide with an iso-makespan line, and also such that the cost does not change. As one point on the line has been shown to be Pareto-optimal, each point in the region that falls onto that line is also Pareto-optimal. The resulting Pareto-optimal parameter combinations are thus the polyhedron for $-p + 2q + 10$.

V. Case Studies

We describe two case studies and perform parametric temporal critical path analysis on them. We investigate the relative speeds of two robot arms for the Twilight system [4], and a specific reconfiguration aspect of the print head of a large-scale printer (LSP) [6]. Algorithms 1 and 2 have been implemented in C++ and run on a 64-bit Ubuntu machine. We have used the C-library of the Double Description method [24] in combination with the GMP library [26].

A. Twilight System

The Twilight System [4] (see Fig. 7) is an example created for the study of controller synthesis and performance analysis of manufacturing systems. The manufacturing system processes balls that need to be drilled. Before drilling is allowed, the ball needs to be conditioned to the right temperature. First, a ball is picked up at the input buffer by the load robot (LR). Once it is brought to the conditioner (COND) it is processed immediately. Once the conditioning of the ball has finished, it immediately needs to be transported by either one of the robots to the drill (DRILL), where it is drilled before the conditioning of the ball expires. Finally, the drilled ball is transported to the output by the unload robot (UR). Figure 8 depicts a simple schedule where the unload robot moves the product from COND to DRILL. Consider that the time it takes for these robots to travel one unit of distance at movement speeds v_{LR} and v_{UR} is $LR = 1/v_{LR}$ and $UR = 1/v_{UR}$ respectively. Increasing LR and UR corresponds to reducing the movement speeds.

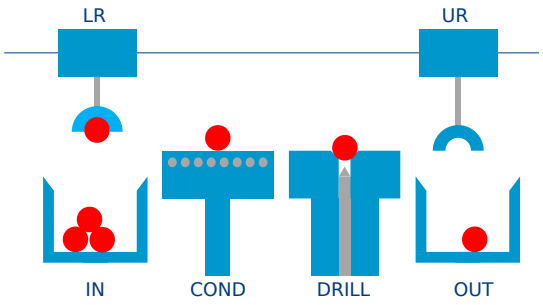


Fig. 7: Twilight manufacturing system, from [4].

The robots can travel either horizontally or vertically, but not diagonally, and always need to move to the highest vertical position before moving horizontally. The horizontal distance between input and conditioning is 10 units, from conditioning to drilling and drilling to output is 5 distance units each. The vertical distance to the input box is 3 units, to the conditioning and drilling platforms 1 unit, and the ball is allowed to be released at any height above the output buffer. Handing over a ball is synchronized and immediate. Conditioning takes exactly 9 time units and expires 8 time units after conditioning has finished. Drilling takes exactly 3 time units. The processing rate of the conditioning, C, and the drilling D are fixed to 1. UR and LR range between 0 and 1.5. Even though they are fixed, C and D are still annotated as parameters in the model to distinguish their contribution to the critical paths.

Figure 9 shows the critical path expressions as function of UR, LR, C, and D. For the given ranges, two positive cycles are detected (blue region). UR and LR become too large to meet the required maximum time between conditioning and finishing the drilling, leading to a positive cycle. In the other cycle, the time it takes for one robot to move away and the other to pick the ball from the conditioner is too large, and the conditioning deadline is violated. Three critical path regions denote the behaviour of the schedule for particular combinations of the robot travelling rates. As LR and UR tend towards zero, the movement speed of the robots becomes infinitely fast, and the makespan becomes $M(0, 0) = 90C+3D+26LR+70UR = 90C+3D$. This is a lower bound on the performance imposed by the time needed for the drilling and conditioning process.

In the green region, changes in UR have the highest impact on performance. The unload robot performs some movements in parallel with the drilling process. Before the unload robot becomes too slow to pick up the ball immediately after drilling ends, there is another cycle that becomes positive. When the unload robot is faster than drilling, the drilling time is always

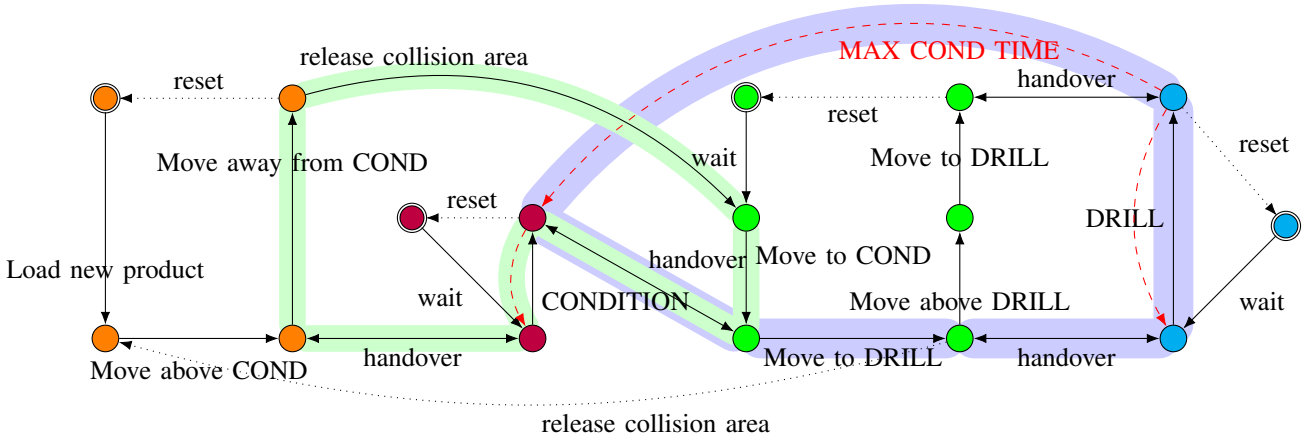


Fig. 8: Example life-cycles and scheduling dependencies for the Twilight System (Figure 7); the Load Robot (orange) loads a product and delivers it to the conditioning stage (purple). The Unload Robot (green) picks it up and needs to deliver it to the Drill (cyan) before the conditioning expires. Dotted edges are dependencies from the current product to the next product. The two positive cycles are shown with green and blue highlighted edges.

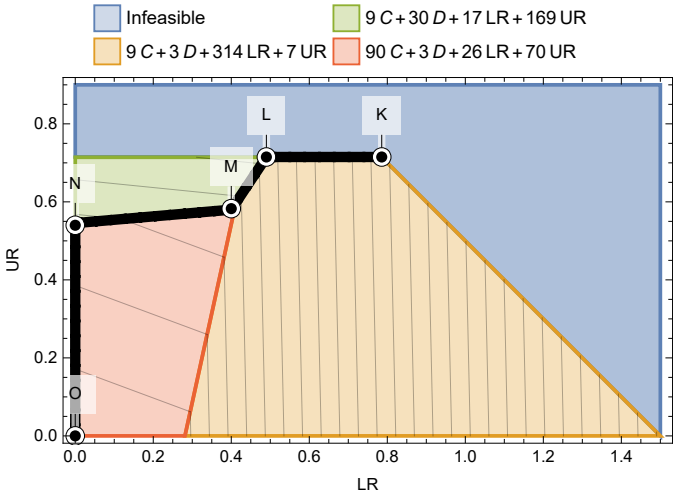


Fig. 9: Expressions for the Twilight system.

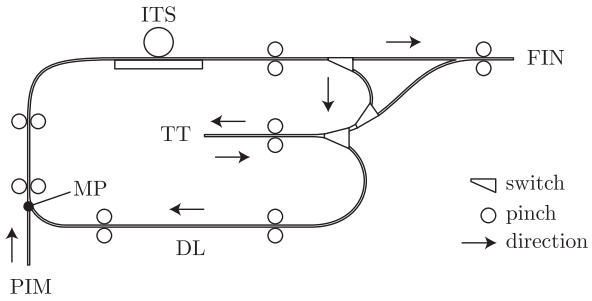


Fig. 10: LSP schematic overview, from [6].

in the critical path, as shown by component $30D$. In the yellow region, the load robot is most often in the critical path. Even though the unload robot performs most actions, our result shows that improving the load robot's speed will give the highest gain. The load robot's speed is in the critical path more often due to the scheduled dependencies. In the red region, the conditioning process becomes the most important bottleneck, especially when LR and UR tend towards zero; the smallest possible makespan for 10 products is $M(0,0) = 93$ time units. This is significantly less than the largest discovered makespan $M(1.5,0) = 483$ time units.

These results show that optimization efforts can be based on the relative and absolute travelling rates of the system. This makes it possible to investigate the interaction of system parameters and performance. Discussing such relations can be valuable when deciding with different stake-holders about the performance of the systems components, such as the robot travelling speeds, conditioning times and interdependencies. Also cost-performance trade-offs are possible. Fig. 9 shows the Pareto-optimal combinations for a cost function $C(LR, UR) = -LR - 3UR$ with a thick black line.

B. Large-Scale Printer

The paper path of a LSP [6] is defined as a path that sheets follow in the printer. The paper path consists of several motors, switches, and functions that perform actions on the sheets. The sheets are guided on a metal track and their speed and acceleration are controlled by pinches. Figure 10 shows the topology of a paper path. The sheets need to move twice through the image transfer station (ITS) before going to the output. Duplex sheets enter the duplex loop (DL), and a turn track (TT) reverses the sheet's direction, for printing on the opposite side. The sheets return from the duplex loop to the merge point (MP) within a pre-defined interval from their first print. The sheets are not allowed to overlap or collide with the sheets coming from the paper input module (PIM). When the sheet has been processed fully, it leaves the printer through the finisher (FIN).

The acceleration profiles of sheets are determined almost completely beforehand; a pre-determined buffer region used

TABLE II: Sheet specifications.

(a) Sheet details

	L	H
A: A4	210	0.25
B: A3	420	0.1
C: A3+	483	0.3

(b) ITS reconfiguration times

		Current		
		A4	A3	A3+
Prev.	A4	0.26	4.51	2.01
	A3	4.78	0.53	6.03
	A3+	2.35	6.10	0.60

to somewhat slow down the sheets. The range of this buffer is encoded as a minimum and maximum travelling time by the vertical edges in the example event network (Fig. 11). Even though the relation between acceleration profiles and minimum and maximum loop times is non-linear, separating these two variables still allows them to be modelled as linear constraints. The horizontal and diagonal edges in the example encode the non-overlapping constraints.

The sheets can have highly varying specifications, and the modules therefore may need to reconfigure themselves to another operating point to achieve the required quality. One such reconfiguration occurs at the ITS; the print head may need to be raised or lowered between sheets to achieve the proper print gap distance for image quality. The print head height H, for example, can be modelled as a linear movement, which can be started after the previous sheet has been fully printed (i.e. has left the ITS). The ITS is ready for the next sheet when it has moved for its full length L, and the print head moved and has stabilized:

$$\begin{aligned} t_{ITS,cur} &= t_{ITS,prev} + \Delta_t \\ &= t_{ITS,prev} + \frac{L_{prev}}{v_{ITS}} + \frac{|H_{prev} - H_{cur}|}{v_{vert}} + \gamma \\ &= t_{ITS,prev} + \alpha L_{prev} + \beta |H_{prev} - H_{cur}| + \gamma \end{aligned} \quad (1)$$

As the speeds v_{ITS} and v_{vert} are constant, we can use parameters $\alpha = 1/v_{ITS}$, $\beta = 1/v_{vert}$. The equation then becomes a linear expression, as H_{prev}, H_{cur}, L_{prev} are all sheet-dependent and assumed constant. The constraint simplifies to the following expression when no head movement is needed $t_{ITS,cur} = t_{ITS,prev} + \alpha L_{prev}$. Decreasing the value of α means increasing the speed at the ITS. Decreasing β means increasing the speed of the vertical movement. Decreasing γ means that oscillations dissipate faster, perhaps due to higher damping in the system.

A repeated pattern of duplex sheets is fed into the printer, i.e. $(ABC)^{60}$. The symbols A, B, C refer to one of the types of sheets denoted in Table IIa and move at the ITS at $v_{ITS} = 800mm/s$. The nominal speed of the head movement is 0.04 units per second.

Let's assume that the ITS is the bottleneck; the PIM and FIN are always ready to provide/receive a sheet. In the event network for the schedule for this print job (Fig. 11), each sheet returns to the merge point in the interval $t_r \in (10, 15)$ from the first time at the merge point. The sequence of first prints and second prints that the scheduler has chosen leads to a requirement on the reconfiguration times between passes in the sequence (diagonal edges).

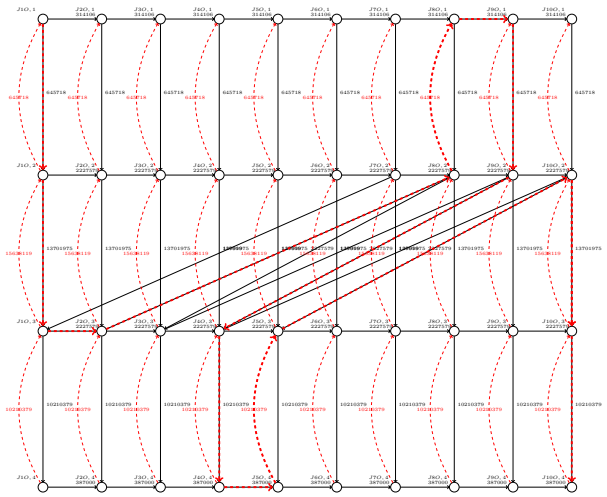


Fig. 11: Example event network and critical path for a LSP flow-shop instance with 10 products. Each column describes a product travelling through the flow-shop, and each row shows the operations on each product. The second and third operation (rows) are mapped onto the same machine. Sequencing edges (diagonal) ensure that at most one product occupies the machine at any time.

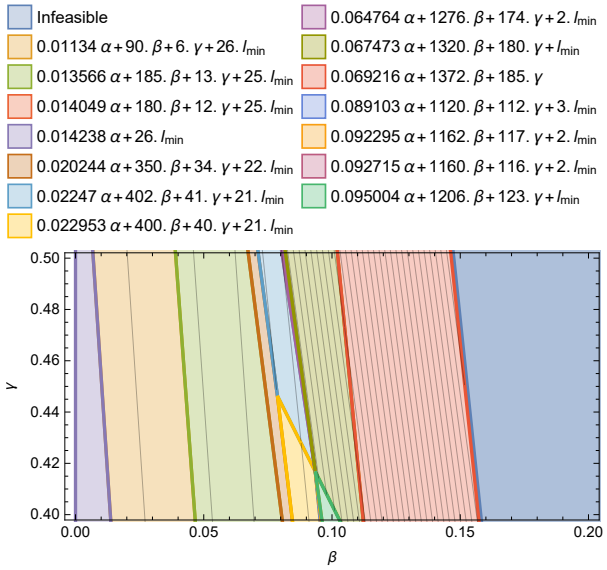


Fig. 12: Example critical path expressions for an LSP.

The critical path expressions associated with different combinations of β and γ are shown in Fig. 12. Depending on β and γ, the majority of the time in the critical path is spent on either: (α) moving the sheet under the ITS, (β) head movements, (γ) oscillations, (l_{min}) travelling through the loop. As β and γ decrease, they also occur less often in the critical path. The performance of the system is lower bounded by the loop time l_{min}, which occurs 26 times in the region with the lowest makespan. The bottleneck changes from the loop time to the head movement parameters as β and γ increase. The expressions show that α becomes relevant for performance, even though only β and γ are varied in the experiment.

Eventually, β and γ become so large that l_{max} is violated.

For $\beta \geq 0.9$, the iso-makespan lines are much closer together, showing that from this point onward, the influence of β and γ are much higher. In these parameter ranges, the makespan is more sensitive to the head movement rate β than to the settling time γ, as can be seen from the contribution of these components in the critical path expressions. One can conclude from this analysis that it is more meaningful to spend development effort on increasing the head movement speed, rather than reducing the settling time.

C. Evaluation

We have evaluated the DETERMINEFEASIBLEPOINTS and DIVIDECONQUER algorithms on several event networks using different parameter ranges. The examples that have been presented so far are summarized in Table III. The illustrated examples have been extended by including more parameters. All experiments are run on an Intel Core i7 950 at 3GHz.

The Packing instances are variations on the Twilight example which only contain minimal time lags. Each product is modelled with 30 events, and dynamic relations exist between events of subsequent products. The time constraints of the relations have been determined stochastically by drawing from several PERT distributions, and are associated with their respective machines, UR or LR. The resulting network is a rather large directed acyclic network without negative constraints, and as such the topological sorting as used in CPM [3] has been used in our experiment, instead of the slower, but more generic BFM.

We observe that the number of critical paths found in the packing cases grows with $O(|E|)$ when only the UR is parametrized, and with $O(|E|^2)$ when both the LR and UR are parametrized. We conjecture that the number of critical paths grows in general with $O(|E|^d)$ where d is the number of parameters.

Due to the growing number of critical paths and corner points of regions for problems with multiple parameters, both the number of splits and the number of evaluations go up. The time taken for each evaluated point remains roughly the same for each network instance, and does not depend on the number of parameters.

VI. CONCLUSIONS

We have shown that it is possible and useful to identify quantitative relationships between system parameters and system performance when the timing constraints in schedules are annotated in linear combinations of design parameters. If the parameters have non-linear relationships, the non-linear model can be split into several models, for which linearised expressions hold around a working point.

We prove that parametric analysis can be applied to the classical Critical Path Method such that regions of critical path expressions and infeasibility expressions are found. These regions and expressions are used to quantify the impact of a parameter change, such as a settling time or a robot travelling rate, on the makespan of the generated schedules. The results of our approach give insight into the interrelationships between design parameters.

We use a divide-and-conquer approach to find the critical path expressions in the parameter space. This paper shows for two manufacturing CPSs how to interpret such relations by combining this information with the original parametrized graph. We also show that Pareto-optimal parameter combinations can be found by transforming the found polyhedra to the cost-makespan space, finding the Pareto-optimal points in that space, and translating the results back to the parameter space. These results can form a sound basis for discussing the trade-offs between cost and performance.

ACKNOWLEDGEMENTS

This work is part of the research programme Robust CPS with project number 12693 which is (partly) financed by the Netherlands Organisation for Scientific Research (NWO). We thank the anonymous reviewers for their helpful comments.

REFERENCES

[1] K. Neumann and C. Schwindt, "Activity-on-node networks with minimal and maximal time lags and their application to make-to-order production," *Operations-Research-Spektrum*, vol. 19, no. 3, pp. 205–217, Sep 1997.

[2] D. G. Malcolm, J. H. Roseboom, C. E. Clark, and W. Fazar, "Application of a technique for research and development program evaluation," *Operations research*, vol. 7, no. 5, pp. 646–669, 1959.

[3] J. E. Kelley, Jr and M. R. Walker, "Critical-path planning and scheduling," in *Eastern Joint IRE-AIEE-ACM '59*. New York, NY, USA: ACM, 1959, pp. 160–173.

[4] B. van der Sanden, J. a. Bastos, J. Voeten, M. Geilen, M. Reniers, T. Basten, J. Jacobs, and R. Schiffelers, "Compositional specification of functionality and timing of manufacturing systems," in *2016 Forum on Specification and Design Languages*, Bremen, Germany, Sep. 2016.

[5] U. Waqas, M. Geilen, J. Kandelaars, L. Somers, T. Basten, S. Stuijk, P. Vestjens, and H. Corporaal, "A re-entrant flowshop heuristic for online scheduling of the paper path in a large scale printer," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015*, 2015, pp. 573–578.

[6] L. Swartjes, L. Etman, J. van de Mortel-Fronczak, J. Rooda, and L. Somers, "Simultaneous analysis and design based optimization for paper path and timing design of a high-volume printer," *Mechatronics*, vol. 41, pp. 82 – 89, 2017.

[7] B. Roy, "Graphes et ordonnancement," *Revue Française de Recherche Opérationnelle*, pp. 323–333, 1962.

[8] J. A. Kerbosch and H. J. Schell, "Network planning by the extended metra potential method (EMPM)," 1975.

[9] S. E. Elmaghraby and J. Kamburowski, "The analysis of activity networks under generalized precedence relations (GPRs)," *Management science*, vol. 38, no. 9, pp. 1245–1263, 1992.

[10] C. Roser, M. Nakano, and M. Tanaka, "Shifting bottleneck detection," in *Proceedings of the Winter Simulation Conference*, vol. 2, Dec 2002, pp. 1079–1086 vol.2.

[11] J. Bastos, B. van der Sanden, O. Donkx, J. Voeten, S. Stuijk, R. Schiffelers, and H. Corporaal, "Identifying bottlenecks in manufacturing systems using stochastic criticality analysis," in *2017 Forum on Specification and Design Languages*, Sept 2017, pp. 1–8.

[12] M. Hajdu, *Network scheduling techniques for construction project management*. Springer Science & Business Media, 2013, vol. 16.

[13] E. Levner and V. Kats, "A parametric critical path problem and an application for cyclic scheduling," *Discrete Applied Mathematics*, vol. 87, no. 1, pp. 149 – 158, 1998.

[14] K. R. Heloue, S. Onaissi, and F. N. Najm, "Efficient block-based parameterized timing analysis covering all potentially critical paths," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 4, pp. 472–484, April 2012.

[15] T. Hune, J. Romijn, M. Stoelinga, and F. Vaandrager, "Linear parametric model checking of timed automata," in *Tools and Algorithms for the Construction and Analysis of Systems*. Berlin, Heidelberg: Springer, 2001, pp. 189–203.

TABLE III: Experimental results of parametric critical path analysis

	Event network		Parametric Critical Path Analysis				
Instance	$\|E\|$	$\|R\|$	Parameter ranges	t (s)	crit. paths	eval.	splits
Example (Fig. 3)	6	15	$p \in (0,1), q \in (0,1)$	0.03	3	19	3
Twilight (Fig. 9)	162	605	$UR \in (0,100), LR \in (0,100)$	0.355	3	32	7
			$UR \in (0,100), LR \in (0,100), C \in (0,1)$	0.631	3	61	12
			$UR \in (0,100), LR \in (0,100), C \in (0,1), D \in (0,1)$	1.502	3	167	27
LSP 180 sheets	362	1785	$\beta \in (0,0.4), \gamma \in (0,1)$	9.315	17	210	60
			$\alpha \in (0,1.25), \beta \in (0,0.4), \gamma \in (0,1)$	26.7	23	574	152
			$\alpha \in (0,1.25), \beta \in (0,0.4), \gamma \in (0,1), l_{min} \in (0,10)$	71.9	40	1709	430
Packing 200 products	6614	21265	$UR \in (0,1)$	59	55	279	92
			$UR \in (0.9,1), LR \in (0.9,1)$	231	99	1078	99
Packing 500 products	16514	53114	$UR \in (0,1)$	373	136	714	237
			$UR \in (0.9,1), LR \in (0.9,1)$	5484	665	9387	2988
Packing 1000 products	33014	106243	$UR \in (0,1)$	1492	262	1404	467
			$UR \in (0.9,1), LR \in (0.9,1)$	49236	2639	39378	12634
Packing 2000 products	66014	212430	$UR \in (0,1)$	6484	551	2976	991
Packing 3000 products	99014	318561	$UR \in (0,1)$	14995	852	4569	1522

[16] A. H. Ghamarian, M. C. W. Geilen, T. Basten, and S. Stuijk, "Parametric throughput analysis of synchronous data flow graphs," in *DATE 2008*, March 2008, pp. 116–121.

[17] M. Damavandpeyma, S. Stuijk, M. Geilen, T. Basten, and H. Corporaal, "Parametric throughput analysis of scenario-aware dataflow graphs," in *2012 IEEE ICCD*, Sept 2012, pp. 219–226.

[18] E. Bini and G. C. Buttazzo, "Schedulability analysis of periodic fixed priority systems," *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1462–1473, Nov 2004.

[19] A. Cimatti, L. Palopoli, and Y. Ramadian, "Symbolic computation of schedulability regions using parametric timed automata," in *2008 Real-Time Systems Symposium*, Nov 2008, pp. 80–89.

[20] R. Bellman, "On a routing problem," *Quarterly of applied mathematics*, vol. 16, no. 1, pp. 87–90, 1958.

[21] L. R. Ford Jr, *Network Flow Theory*. Rand Corp, 1956.

[22] R. Sedgewick and K. Wayne, *Algorithms, 4th Edition*. Addison-Wesley, 2011.

[23] G. Behrmann, E. Brinksma, M. Hendriks, and A. Mader, "Production scheduling by reachability analysis: a case study," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*. Los Alamitos, CA, USA: IEEE, 2005, pp. 140–142.

[24] K. Fukuda and A. Prodon, "Double description method revisited," *Combinatorics and Computer Science*, pp. 91–111, 1996.

[25] M. A. Yukish, "Algorithms to identify pareto points in multi-dimensional data sets," Ph.D. dissertation, The Pennsylvania State University, 2004.

[26] T. Granlund and the GMP development team, *GNU MP: The GNU Multiple Precision Arithmetic Library*, 6th ed., 2016. [Online]. Available: http://gmplib.org/

Joost van Pinxten (S'16) holds a B.Sc. and M.Sc. in Electrical Engineering from Eindhoven University of Technology. He is currently a Ph.D. candidate at Eindhoven University of Technology in the Robust Cyber-Physical Systems project. His research interests include multi-objective combinatorial optimization and scheduling, inter-disciplinary design of cyber-physical manufacturing systems, and model-driven design tools.

Marc Geilen is an assistant professor in the Department of Electrical Engineering at Eindhoven University of Technology. He holds an M.Sc. and a Ph.D. from Eindhoven University of Technology. In 2010, he was a McKay Visiting Professor at the University of California, Berkeley. His research interests include modeling, simulation and programming of multimedia systems, formal models-of-computation, model-based design processes, multiprocessor systems-on-chip, networked embedded systems and cyber-physical systems, and multi-objective optimization and trade-off analysis. He is a member of IEEE. He has been involved with several national and international research projects and programs on the above topics with strong industrial connections. He has served on various TPCs and on organizing committees for several conferences including DATE as a topic chair and member of the executive committee.

Martijn Hendriks photograph and biography not available at time of publication

Twan Basten (M'98-SM'06) received the M.Sc. and Ph.D. degrees in computing science from Eindhoven University of Technology (TU/e), Eindhoven, the Netherlands. He is currently a Professor with the Department of Electrical Engineering, TU/e, where he chairs the Electronic Systems group. He is also a Senior Research Fellow with ESI, TNO, Eindhoven. His current research interests include the design of embedded and cyber-physical systems, dependable computing, and computational models.