A new approach to predicting reliable project runtimes via probabilistic model checking

Ulrich Vogl✉ and Markus Siegle✉

Universität der Bundeswehr München, 85577 Neubiberg, Germany,
{ulrich.vogl|markus.siegle}@unibw.de

Abstract. For more than five decades, efforts of calculating exact probabilistic quantiles for generally distributed project runtimes have not been successful due to the tremendous computation requirements, paired with hard restrictions on the available computation power. The methods established today are PERT (Program Evaluation and Review Technique) and CCPM (Critical Chain Project Management). They make simplifying assumptions by focusing on the critical path (PERT) or estimating appropriate buffers (CCPM). In view of this, and since today's machines offer an increased computation power, we have developed a new approach: For the calculation of more exact quantiles or – reversely – of the resulting buffer sizes, we combine the capabilities of classical reduction techniques for series-parallel structures with the capabilities of probabilistic model checking. In order to avoid the state space explosion problem, we propose a heuristic algorithm.

Keywords: project planning, stochastic graph model, series-parallel reduction, probabilistic model checking, PERT, CCPM

1 Introduction

In project planning, predicting the total runtime of activity chains and/or concurrent project activities, is an important task. In most situations, due to the influence of many unpredictable factors, probabilistic methods are more appropriate than deterministic ones. Two well-known representatives, PERT (Program Evaluation and Review Technique [13, p. 303-365]) and CCPM (Critical Chain Project Management [4]), mostly only yield inaccurate results, due to methodical simplifications (taken in order to make them computable).

PERT is limited in principle by focussing on the critical path. Sub-critical paths are completely neglected – even if they appear in a high number or with a significant variance influencing the total runtime distribution. (A more detailed analysis of PERT networks, albeit under Markovian assumptions, has been described in [7].) CCPM works in a more differentiated (but also non-preemptive) way: Using this method, all non-critical paths are augmented by feeding buffers in order to lower the influence of the critical path. For determining the size of the buffer, the so-called Cut & Paste Method and the Root Square Error Method have been suggested. But these methods for buffer sizing handle the variance of the side-paths only in an indirect, usually inaccurate manner, and thus their

effectiveness is hard to quantify. This motivated us to take a review on this topic, seeking for a new, better approach. In particular, since the introduction of both methods in 1958 resp. 1996 the available computation power has increased significantly, and new calculation methods such as probabilistic model checking (pMC) [8], together with efficient tools like PRISM [12], have become available.

We propose a method for complete and accurate calculation of a project's total runtime distribution, where our key ideas are as follows:

- The nodes of a stochastic graph model (SGM), i.e. a directed acyclic graph (DAG), represent the activities of a project. Each node is equipped with a continuous probability distribution, representing its individual runtime.
- The nodes of the graph are reduced in a step-by-step manner, ultimately leading to only a single node, with a related result distribution which represents the project's total runtime.
- We seek to find subgraphs which can be reduced to a single node by serial or parallel reduction, as explained below in Sec. 2.
- When no further series-parallel reduction is possible, we identify the starting and end points of a so-called complex cluster (a generally structured subgraph). We use the concept of syncpoints (see Sec. 3 below) for defining such clusters. The cluster is then reduced by a *complex reduction* to a single node, for which step pMC is employed. In order to avoid state space explosion, it is essential to limit the size of the graph to be fed into pMC. Therefore the clusters analysed by pMC should be as small as possible.
- One faces the challenge of finding an appropriate (heuristic) fitting for the given source distributions, because pMC tools usually only accept exponential distributions. "Fitting" in this context means the approximate modelling of a given distribution by some phase type distribution, for instance by matching the first moments.
- It is an interesting side effect that already one complex reduction step can often eliminate a local complexity hotspot and thereby enable further series-parallel reduction steps.

1.1 Related Work

Melchiors and Kolisch [10, 11] have presented a heuristic approach for establishing and assessing scheduling policies for dynamically arriving, concurring project activities competing for limited resources. In contrast to our approach which works on the operational planning level, their work is targeted at the "tactical" planning level (aka "Macro Process Planning"). It uses aggregated work packages as well as global estimates of runtime and precedence relationships. It assumes high variability project environments, allowing for dynamically emerging activities as well as dynamically changing dependencies between activities. Several heuristics (based on CTMCs and MDPs) are combined, to gain a computable model state space (including a preemptive modelling approach). The final valuation and performance assessment of the priority policy methods is done via simulation.

Kapici [5] presented an approach where complex projects are mapped to a newly developed stochastic model. He then derives statements on the adherance of deadlines, costs or other results. His approach is fully simulation-based.

There are further approaches, most of them based on simulation and/or complexity-reducing heuristics. The destinctive feature of our work, however, is the accurate consideration of the individual project activities and their precedence relations, combined with the power of a probabilistic model checker.

1.2 Paper structure

The rest of this paper is structured as follows: Sec. 2 provides background information on PERT, CCPM and the analysis of stochastic graph models. The shortcomings of these classical methods motivated us to develop an innovative approach, which is presented in detail in Sec. 3. To illustrate our method, Sec. 4 presents some non-trivial examples. Finally, Sec. 5 summarizes the results and describes some ideas for continuing work.

2 State of the art

2.1 Stochastic Graph Models

Stochastic Graph Models are a simple and intuitive formalism for modelling the structure of projects, parallel programs, collections of interdependent tasks, etc.. They have been described in detail, e.g., in [6].

Definition 1. *A Stochastic Graph Model (SGM) is a directed acyclic graph* $G = (V, E, exec)$ *with the following properties:*

1. V *is a finite set of vertices (aka nodes), and* $E \subseteq V \times V$ *is the set of directed edges.* G *is connected and has a single source and a single sink node.*
2. $exec : V \mapsto Distr$ *is a function which assigns to each vertex its associated continuous nonnegative runtime distribution. The runtime distributions of all vertices are mutually independent.*
3. *A tuple* $p = (e_1, e_2, \ldots, e_k) \in E^k$ *of edges is called a directed path, if and only if* $\forall_{1<i\leq k} : start(e_i) = end(e_{i-1})$*, where* $start(e)/end(e)$ *denotes the start/end node of edge* e*. The set of all possible paths in* G *(induced by* E*) is denoted by* $paths(E)$ *.*

A vertex of a SGM starts its execution as soon as all its predecessor vertices have completed. The goal of SGM analysis is to determine the total runtime distribution, i.e. the elapsed time between the start of the source node and the finishing of the sink node.

Definition 2. *En edge* $e \in E$ *is called* ***redundant*** *iff there is a path* $p = (e_1, e_2, \ldots, e_k)$ *of edges with* $e \notin p$*,* $start(e) = start(e_1)$ *and* $end(e) = end(e_k)$*.*

For the rest of this paper, without any loss of information, we assume the SGM to be free of redundant edges (if a particular SGM is not so, they can be discovered and removed easily).

There is a class of SGMs, featuring a series-parallel structure, which is amenable to efficient analysis. This class is characterised by the following definition and theorem.

Definition 3. *Two vertices n_i and n_j of a SGM are said to be* serially connected *iff n_j is the only successor of n_i and n_i is the only predecessor of n_j (or vice versa). A set of vertices $P \subseteq V$ with $|P| \geq 2$ is called* parallelly connected *iff all vertices $n \in P$ have the same set of predecessors and the same set of successors.*

Theorem 1. *(from [6, p. 184]) Two serially connected vertices n_1 and n_2 may be serially reduced to a single vertex n_{12} as follows:*
$pred(n_{12}) = pred(n_1), \quad succ(n_{12}) = succ(n_2), \quad exec(n_{12}) = exec(n_1) * exec(n_2),$
where $pred(n)/succ(n)$ denotes the set of predecessor/successor nodes of n, and $*$ denotes the convolution operator on continuous distributions. Two or more parallelly connected vertices $n_1, n_2, \ldots, n_k$ may be parallelly reduced to a single vertex $n_{1\ldots k}$ as follows:
$pred(n_{1\ldots k}) = pred(n_1), \quad succ(n_{1\ldots k}) = succ(n_1),$
$exec(n_{1\ldots k}) = \max(exec(n_1), \ldots, exec(n_k)),$
where max *denotes the maximum operator on continuous distributions.*

Definition 4. *A SGM G is called series-parallel reducible if it can be reduced to a single node by successive serial and parallel reduction steps.*

In short, serial reduction means that two serially connected nodes are reduced to a single node whose distribution is the convolution of the two operand distributions (the convolution yields the distribution of the sum of the execution times). Parallel reduction means that two or more "parallel" nodes are reduced to a single node which is distributed according to the maximum of the operand runtimes. Series-parallel reduction is a very efficient method for analysing SGMs. However, while many graph structures are series-parallel reducible, in practice many SGMs are not of this class, see e.g. the graph shown in Fig. 1, which is no longer series-parallel reducible if the traverse edge A-D is inserted.

2.2 A simple SGM

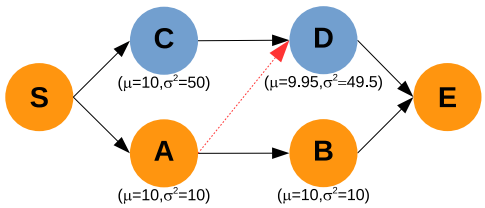

Fig. 1. Example – the traversal edge A-D is initially neglected

We consider a simple SGM consisting of four significant nodes $\{A, B, C, D\}$, enclosed by the source node S and the sink node E (shown in Fig. 1). It is assumed that nodes S and E have negligible runtime, i.e. their runtime is deterministic with value zero. The runtimes of the four significant nodes all follow an Erlang-distribution with n phases and basic rate λ (having mean $\mu = \frac{n}{\lambda}$ and variance

$\sigma^2 = \frac{n}{\lambda^2}$): For A and B with $\lambda = 1, n = 10$, for C with $\lambda = 0.2, n = 2$, and for D with $\lambda = 0.201, n = 2$. The resulting μ and σ^2 values are given in Fig. 1.

For the sake of simplicity, we first neglect the traverse edge from A to D, such that the SGM is series-parallel reducible. Its overall runtime distribution is thus

$$exec(ABCD) = \max(exec(A) * exec(B), exec(C) * exec(D))$$

This density is depicted in Fig. 2 as curve (3) (green).

2.3 PERT

PERT focusses only on the critical path, thereby taking only the mean execution times into account. For the SGM from Fig. 1 (without traversal edge A-D), this results in the critical path (S-A-B-E) (orange coloured nodes), which has a mean value of 20 time units and a variance of 20 (standard deviation 4.47 time units). We calculated exact distributions for two variants and depicted the associated densities in Fig. 2:

(1) only the critical path (S-A-B-E) (curve (1), blue)
(2) only the sub-critical side-path (S-C-D-E) (curve (2), red)

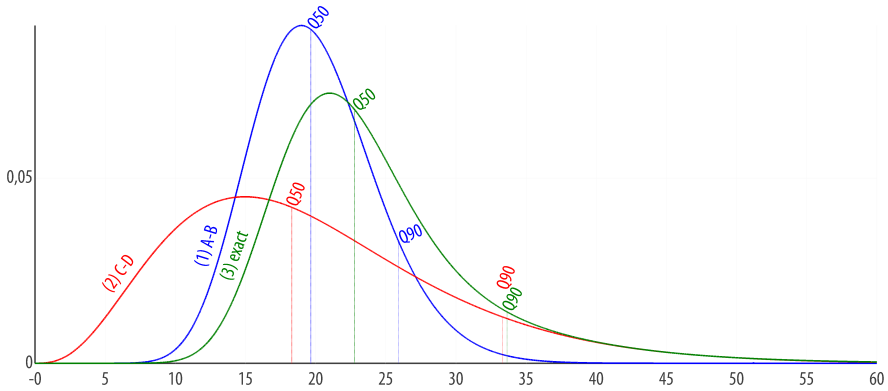

Fig. 2. Comparison of the densities; 50%- and 90%-quantiles

Curves (1) and (2), representing the competing paths, possess indeed similar means (20 resp. 19.95 time units), but quite different variances (20 resp. 99.5 time units squared). The crucial conclusion is gained by comparing curves (1) and curve (3). The former represents the (seeming) PERT view, whereas the latter equals the precise overall distribution. The PERT-caused error becomes most apparent if we focus on the 90% quantiles:

- PERT (curve (1)) suggests, that a runtime of about 25.9 time units would be sufficient to finish the project at a confidence level of 90%.
- The accurate calculation (curve (3)) clarifies, that this confidence level in reality is reached only at about 34 time units.

Fig. 2 also shows that for this particular SGM, the 90%-quantile of the sub-critical path (curve (2), red) – by chance – is almost exact, but its distribution is far from the exact distribution.

2.4 CCPM

Now one will rightly point out that the weaknesses of the PERT method in its almost 60 years of history are sufficiently known. But what about the example, fed into a more modern planning method like CCPM?

The deciding feature of the CCPM method consists of planning all paths at a 50%-quantile level and equipping the critical path as well as all sub-critical side-paths with appropriate buffers. In the literature, two methods for buffer calculation are described: The **Cut & Paste method (C&PM)**, introduced by CCPM inventor E.M. Goldratt [4], proposes for each path to sum up its 50%-quantiles, then take that result as the path's base runtime and add to it an additional buffer of 50% of that. Alternatively, some authors [3] recommend the **Root Square Error Method (RSEM)**, which takes as buffer the square root of the sum of squared differences between the 50%- and the 90% quantiles.

Table 1. CCPM – total runtime (+ buffer) & associated quantiles

	C&PM	***RSEM***
path A-B [time units]	19.34 (+9.67)	19.34 (+6.42)
path C-D [time units]	16.75 (+8.38)	16.75 (+15.60)
total/maximum [time units]	29.01	32.35
relating exact quantile	80.46%	88.01%

Table 1 shows the results of these two variants of the CCPM method applied to the SGM from Fig. 1 (without traversal edge A-D). Looking only at the medians, again path (S-A-B-E) with value 19.34 dominates path (S-C-D-E) which has value 16.75 (these values differ from the medians of Fig. 2 because they are simple sums of the single activity medians and not of the convoluted distributions). But taking into account the buffers, the RSEM method identifies (S-C-D-E) as the critical chain. If we compare the relating exact quantiles of the calculated finishing times to the desired 90% quantile of the accurate model evaluation (at 33.65 time units), we can conclude:

- C&PM yields an "in time"-completion probability of only around 80.5%,
- with RSEM the problematic sub-critical side path (C-D) gains more importance by its dominating feeding buffer. That indeed increases the completion probability to around 88% (which is still below the desired 90% level).

2.5 Non-series-parallel SGMs

We now return to the SGM in Fig. 1, but this time we include the traversal edge A-D into the calculations. This is remarkable, since exactly that edge destroys the series-parallel reducibility of the graph. Therefore, in order to obtain the precise

runtime distribution, we can no longer rely on the convolution and maximum operators, but we need indeed a more powerful computation method. We chose to employ pMC, in particular we use the probabilistic model checker PRISM [12], which under the hood performs a state space analysis (transient analysis by means of uniformization). Since PRISM provides no explicit calculation feature for discrete densities, we use it as follows:

1. First we choose an appropriate discretization step width, e.g. 1% of the smallest occuring mean or standard deviation, as well as an estimation of the upper interval limit, which – for instance – can be gained by a pathwise consideration, thereby taking each single distribution's upper limit.
2. Then for each discrete time value t_i we perform a PRISM call of the form $P(T < t_i) =?$, which delivers the cumulative distribution value for time t_i.
3. PRISM provides a feature to chain such calculations for entire intervals by only one call (given start time, end time and step width); this obviously leads to a tremendous decrease of the calculation effort (granting PRISM the reuse of prior results).
4. The desired density values are eventually gained by a simple numerical differentiation, taking the difference of each two neighbouring distribution values.
5. The calculation time can be further reduced by splitting the PRISM call intervals and distributing them onto several CPU threads.

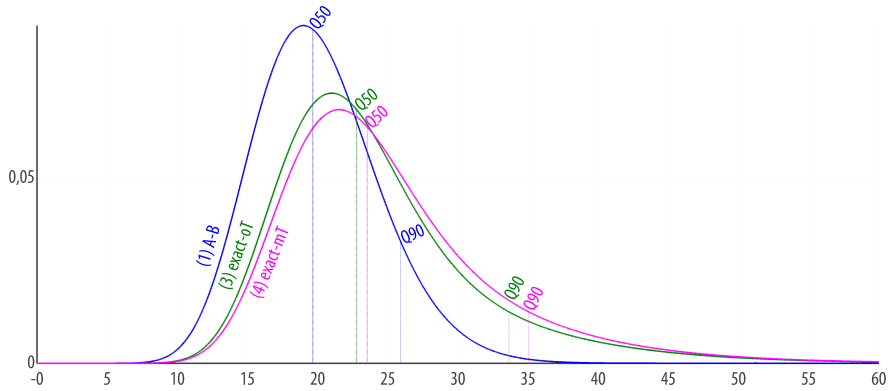

Fig. 3. Comparison of the densities: without (3) resp. with (4) the traverse A-D

Fig. 3 depicts the influence of the traversal edge by comparing the overall densities with / without it. Note that curves (1) (blue) and (3) (green) equal those of Fig. 2. The exact overall density for the SGM with traversal edge, shown as curve (4) (magenta), deviates slightly from curve (3) (without traversal): Now the 90% quantile is reached at 35.01 time units (previously: 33.65). For the sake of comparison, the PERT based density (1), gained by concentrating on the critical path only, is displayed once again (after insertion of the traverse edge, the critical path is still the same!). The 90% quantile of the PERT view lies (at 25.9 time units) more than a quarter below the actual value.

In summary, it can be stated that an accurate calculation of the density offers significant advantages over the established methods PERT and CCPM. In principle, state-space-based methods such as implemented by PRISM are able to produce such accurate distributions, but they are limited to small or medium-sized models because of the arising state space explosion problem. Admittedly, the SGM considered in this section was an extremely simple case – the reality is usually much more complex. Therefore, in the following section, we develop a new method which combines series-parallel reduction and pMC, thereby making it applicable to larger SGMs of realistic size.

3 A new reduction method for analysing project runtimes

Calculating the exact overall runtime density for arbitrarily structured graphs, equipped with general runtime distributions, is in general not feasible for the following reasons:

(a) Series-parallel reduction by use of the convolution and maximum operators quickly leads to very complicated mathematical expressions, if carried out symbolically. Those are difficult to handle, even with advanced tools such as Mathematica [14] or Maple [9].
(b) If the SGM at hand is not series-parallel reducible, purely analytic approaches fail if activity runtimes have general distributions, since state space analysis relies on the memoryless property of the exponential distribution.
(c) Even if all node execution times are exponentially or PH-type distributed, such that state space analysis would be possible in principle, one quickly reaches the limits of computability because of state space explosion.

Our proposed method, presented in this paper, overcomes these problems in the following way: Problem (a) is dealt with by representing general distributions not symbolically, but numerically. That means a general distribution is discretized and represented as a step function, and the convolution and maximum operators are performed on the basis of such numerical representations. A similar numerical approach had previously been described in [6]. Concerning problem (b), we enable state space analysis of SGMs (or subgraphs thereof) with non-exponential execution times by replacing those general distributions with fitted phase-type distributions (see, e.g. [2]). Finally, with regard to problem (c), our scheme avoids performing state space analysis on the overall model. Instead, it combines series-parallel reduction steps with state space analysis of small subgraphs in an iterative manner.

3.1 The iterative reduction algorithm

Fig. 4 illustrates our iterative reduction algorithm to calculate the overall runtime distribution. In each round, the algorithm searches for candidates for serial or parallel reduction and performs the respective reductions, as long as possible. If no further serial or parallel reduction is possible, the algorithm identifies a so-called "complex cluster", which is a generally structured subgraph whose runtime distribution will be analysed with the help of pMC. Since the vertices of the given SGM are associated with generally distributed execution times, which

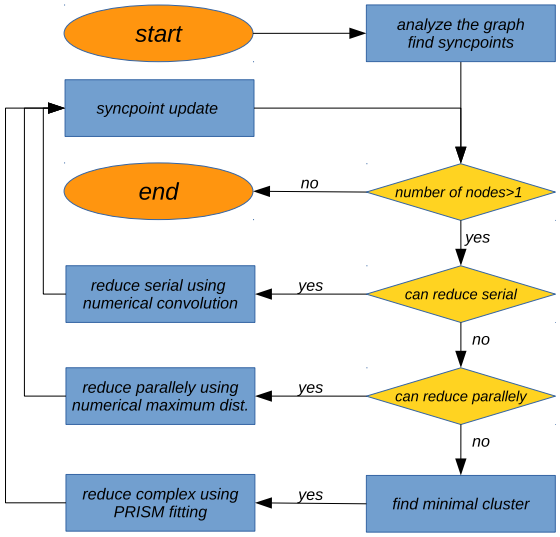

Fig. 4. Flow diagram of the iterative reduction algorithm

cannot be fed immediately into probabilistic model checkers such as PRISM, we need a suitable fitting method to approximate general distributions by exponential phases. This is explained in Sec. 3.3. There remains the problem of how to identify an appropriate cluster? We solve this problem by introducing so-called "syncpoints", as elaborated on in the following subsection.

3.2 An efficient algorithm using syncpoints

When performing the stepwise reduction, our algorithm needs to be able to identify appropriate starting points and end points of clusters to be reduced (either serially or parallelly or by complex reduction). This search can be directed by focusing on particular edge subsets which we call syncpoints:

Definition 5. *Given a SGM $G = (V, E, exec)$. A set $\mathcal{E} \subseteq E$ of edges is called a **full syncpoint** (FSP), if and only if for the node set $\mathcal{P}$ consisting of all the starting points of $\mathcal{E}$ and for the node set $\mathcal{S}$ consisting of all the end points of $\mathcal{E}$ the following three conditions hold:*

1. *Each edge from $\mathcal{P}$ to $\mathcal{S}$ is in $\mathcal{E}$.*
2. *All nodes in $\mathcal{P}$ have the same set of successor nodes, namely $\mathcal{S}$.*
3. *All nodes in $\mathcal{S}$ have the same set of predecessor nodes, namely $\mathcal{P}$.*

*If $|\mathcal{P}| = |\mathcal{S}| = 1$, we call $\mathcal{E}$ a 1-to-1-SP (11SP), a special subclass of FSPs. If only conditions (1) and (2) with $|\mathcal{P}| > 1$ resp. conditions (1) and (3) with $|\mathcal{S}| > 1$ hold, we call $\mathcal{E}$ a **backward** or **forward halfsyncpoint** (BHSP or FHSP). We denote $\mathcal{P}$ as the entrance side and $\mathcal{S}$ as the exit side of any syncpoint type.*

Fig. 5 illustrates the concept of full syncpoints and halfsyncpoints. During SGM reduction, we make use of the syncpoint definition as follows:

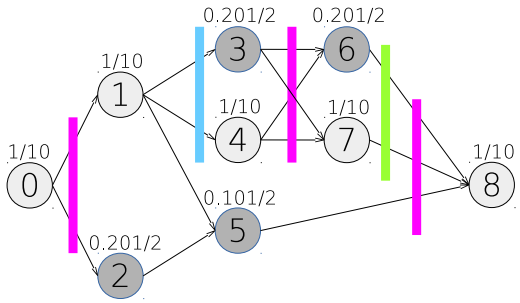

Fig. 5. FSPs (magenta), FHSP (blue), BHSP (green); node labels for later use

- For each 11SP, we can combine its predecessor and its successor node to a single node by serial reduction, convolving the associated densities.
- If a FSP or a FHSP has got exit-sided node set M, which coincides with the entrance-sided node set of another FSP or BHSP, then there is a parallel reduction opportunity for all nodes of M, i.e. the associated densities can be combined by calculating their maximum.
- If all paths emerging from a FSP or FHSP lead to another FSP or BHSP and – vice versa – all paths reaching the latter come from the former one, then the set C of all nodes included in-between forms a cluster. Such a cluster can then be reduced by a complex reduction step using pMC.

Definition 6. *A FHSP resp. BHSP $\mathcal{E}$ is called a* ***subsyncpoint*** *(FSSP resp. BSSP), if and only if there is another set $\mathcal{E}'$ with $\mathcal{E} \subsetneq \mathcal{E}'$ and $\mathcal{E}'$ is either a FSP or a FHSP resp. BHSP.*

It is clear by definition, that the class of FHSPs decomposes in two subclasses, namely the FSSPs and the so-called **real FHSPs**, not beeing a subset of any superordinated syncpoint $\mathcal{E}'$; the same applies to BHSPs / BSSPs.

Each FSP with n entrance- resp. m exit-sided nodes implicitly defines $\binom{n}{k}$ BSSPs of cardinality k at its entrance side, resp. $\binom{m}{l}$ FSSPs of cardinality l at its exit side, with k, l the number of nodes attached at the respective side. Thus a FSP covers in sum $2^n - n - 2$ BSSPs and $2^m - m - 2$ FSSPs $(n, m > 1)$. From a hierarchical point of view, each FSP or HSP with cardinality k $(k > 2)$ immediately covers k SSPs of cardinality $k - 1$ (at the entrance resp. exit side).

The idea now is: The set of all syncpoints / halfsyncpoints in the given SGM induces a new, in general less complex DAG (regarding a kind of precedence to be defined), such that all described reduction steps can be performed on that structure, with a potentially tremendous reduction of calculation effort. In order to be able to identify and manage this, we need some more definitions:

Definition 7. *A set of edges $\mathcal{E} \subseteq E$ is called* ***Z-connected****, if and only if for each two edges $a, b \in \mathcal{E}$ there exist $k \geq 0$ and edges $e_1, e_2, \ldots, e_k \in \mathcal{E}$ such that for $(a, e_1, e_2, \ldots, e_k, b)$ each two neighboring edges have either the same start*

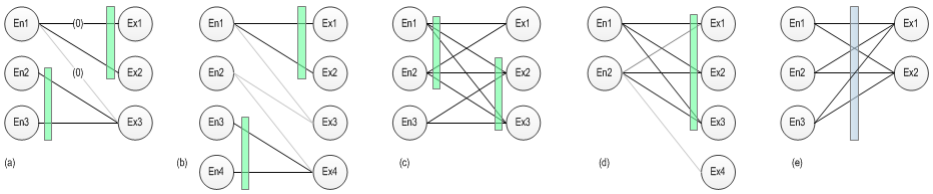

Fig. 6. TLM examples: only (e) is a FSP; (a)... (d) contain HSPs (green).

node or the same end node. $\mathcal{E}$ *is called* ***max-Z-connected*** *if there is no other Z-connected set* $\mathcal{E}' \subseteq E$ *with* $\mathcal{E} \subsetneq \mathcal{E}'$.

Definition 8. *Reusing the notation of Def. 5, a set of edges* $\mathcal{E} \subseteq E$ *is called a* ***top-level manager(TLM)****, if and only if*

1. $\mathcal{E}$ *is max-Z-connected.*
2. $\mathcal{E}$ *contains at least one (full or half) syncpoint.*

Examples of TLMs are shown in Fig. 6. Remarks:

- Condition 2 of Def. 8 is for operational reasons, since TLMs without any contained syncpoint would not be useful.
- Since the SGM is acyclic, each FSP is also a TLM. In Fig. 5, for instance: $FSP(567 <> 8)$ or $FSP(0 <> 12)$.
- In Fig. 5, $FHSP(1 < 34)$ is neither a FSP nor a SSP, hence a real FHSP.

The next definition is needed, since it is possible to construct an acylic DAG without redundant edges where two max-Z-connected sets can be traversed in any order.

Definition 9. *Using the notation of Def. 5, we call a SGM* ***max-Z-acyclic*** *if*

1. $\forall \mathcal{E} \subseteq E$ *max-Z-connected and* $\forall p = (e_1, \ldots, e_k) \in paths(E)$ *holds:* $|\mathcal{E} \cap p| \leq 1$.
2. $\forall \mathcal{E}, \mathcal{F} \subseteq E$ *max-Z-connected,* $\forall p_1, p_2 \in paths(E)$ *: If both* p_1 *and* p_2 *intersect with both* $\mathcal{E}$ *and* $\mathcal{F}$*, then they pass through* $\mathcal{E}$ *and* $\mathcal{F}$ *in the same order.*

Definition 10. *A TLM* $\mathcal{E} \subseteq E$ *is called a* ***predecessor*** *of another TLM* $\mathcal{F} \subseteq E$ *with* $\mathcal{E} \neq \mathcal{F}$ *if and only if there are edges* $a \in \mathcal{E}, b \in \mathcal{F}$ *and a directed path* $(a, e_1, \ldots, e_k, b) \in paths(E)$*. In this case* $\mathcal{F}$ *is called a* ***successor*** *of* $\mathcal{E}$.

Theorem 2. *For each max-Z-acyclic SGM there is a set of TLMs, fulfilling:*

1. *The TLMs are mutually disjoint and the set of TLMs is unique.*
2. *Def. 10 induces a well defined DAG on the TLMs, the* ***TLM-DAG****.*
3. *Each FSP resp. real HSP of the given SGM is covered by exactly one TLM.*

Proof (sketch): In a first step we use the max-Z-connected property of the TLMs to show that they are mutually disjoint and there is only one unique representation of them (independent of the discovering algorithm). Using the properties of Def. 9, we can show the precedence relationship on TLMs to be well defined. As remarked, each FSP is automatically a TLM; on the other hand each real HSP $\mathcal{E}$ *is Z-connected and thus can be expanded to a max-Z-connected* $\mathcal{E}'$*, a TLM.*

The following pseudocode sketches a recursive algorithm for identifying the TLMs and building the TLM-DAG:

```
# top-level call:
recFindTLMDAG(<SGM source node>, <emptyMap>, null)

# recursion method definition:
# @param curRecNode the current recursion subject node
# @param visitedNodeMap map of visited nodes -> already found following TLMs
# @param latestTLMOnStack the latest discovered TLM on recursion stack
# @returns the TLMGraph
TLMGraph recFindTLMDAG(curRecNode,visitedNodeMap,latestTLMOnStack)
  if (visitedNodeMap.contains(curRecNode))  # curRecNode already visited?
    if (visitedNodeMap.get(curRecNode).size()>0) # already found follow. TLMs?
      # update the result by adding edge(s):
      #  connect latest TLM on stack to already found TLMs beneath curRecNode
    else:
      # run recursion call for all successors of curRecNode
  else:
    # determine all Z-connected "children" (S) and "brothers" (P)
    #  by alternating fw/bw expansion, until there are no more new nodes.
    # group S and P by mutual dependencies in both directions:
    #  a) group P by common successors
    #  b) group S by common predecessors
    if (zBrotherGroupMap.size()==1 & zChildrenGroupMap.size()==1) # it's a FSP?
       # create new FSP and add it and the new connecting edge
       # (from latestTLMOnStack) to the result
       # remember all brothers (P) as visited, mapping to the found FSP
       # run recursion call for all children (S)
    else:  # it is not a FSP!
       # if there are HSP(s) between P and S, create a TLM to cover them:
       #  each a)-group with at least 2 p, sharing same common successors
       #  each b)-group with at least 2 s, sharing same common predecessors
       #  induces such a HSP, to be kept by the TLM.
       # remember brothers (P) as visited: wrt. without a found following TLM
       # run recursion call for all children (S)
endMethod
```

Having constructed the TLM-DAG (including all FSPs), the next question is the integration of the halfsyncpoints, especially the subsyncpoints. At this so-called **micro level**, each TLM has to manage three issues:

(2) its subordinated HSPs, ideally in a hierarchical manner,
(2a) the relation between the entrance-/exit-sided nodes and the affecting HSPs,
(2b) the reachability (over paths) between its exit sided nodes and the entrance-sided nodes of the succeeding TLMs.

A solution for this is illustrated in Fig. 7, where the development of the micro linkage is shown based on the two rightmost examples of Fig. 6, assuming that TLM (d) is an immediate predecessor of TLM (e). Fig. 8 illustrates the stepwise reduction of the example from Fig. 5 by relying on the corresponding syncpoint TLM-DAG and its underlying micro structure. The transitions within the figure are reached by (P)arallel, (S)erial and (C)omplex reduction steps.

3.3 Numerical reduction and fitting of general distributions

Whenever a serial or parallel reduction step is performed, our algorithm works on numerical representations of the operand densities. This avoids having to deal with complicated mathematical expressions (they arise very quickly after a couple of such operations have been performed) which could no longer be handled

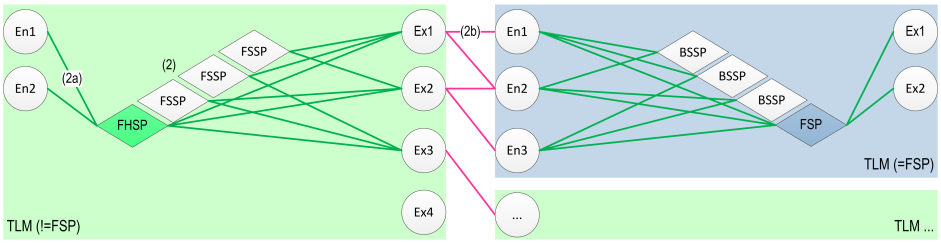

Fig. 7. Micro linkage of Fig. 6 (d,e): each TLM manages its HSPs (2), their usage of entrance-/exit-sided nodes (2a) and the mutual reachability by these nodes (2b).

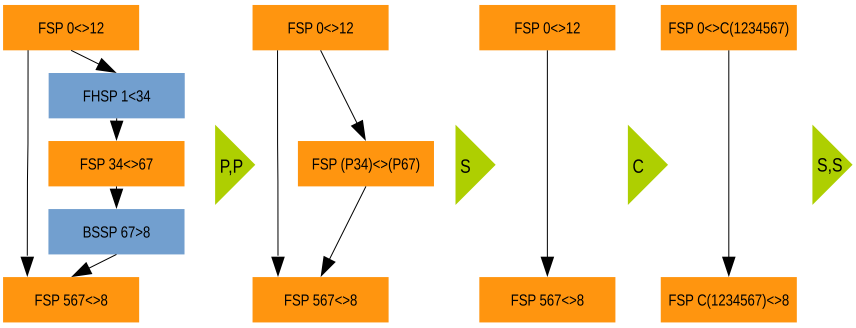

Fig. 8. Reduction steps along the syncpoint DAG: full SPs (orange) and half SPs (blue)

by formula manipulation packages. The precision of the numerical representation increases with increasing number of interpolation points, whereas the calculation effort for these numerical operations grows quadratically with the number of interpolation points. A similar numerical implementation of series-parallel reductions had been described in [6].

For each complex reduction, i.e. before a complex cluster can be analysed by means of pMC, an appropriate PRISM-Model needs to be generated. For this purpose, each single activity's runtime distribution is fitted by a phase-type distribution. At the moment, we advocate a fitting by two convolved Erlang distributions $Erl(\lambda_1, n)$ and $Erl(\lambda_2, m)$. Their parameters can be chosen in such a way that the first two moments match exactly to the original distribution and the error at the third moment is minimized. This fitting provides good results, as long as the coefficient of variation of the original distribution is at most 1, i.e. as long as $\sigma^2/\mu^2 \leq 1$. As a concluding remark on the issue of fitting, note that our algorithm presented in Sec. 3.1 is independent of the particular fitting method used, i.e. other fitting approaches (e.g. [1]) can easily be incorporated.

4 Complex examples

Let us look once again at the example in Fig. 5 and assume that an Erlang distribution is chosen for all activity runtimes (find λ and n values printed directly

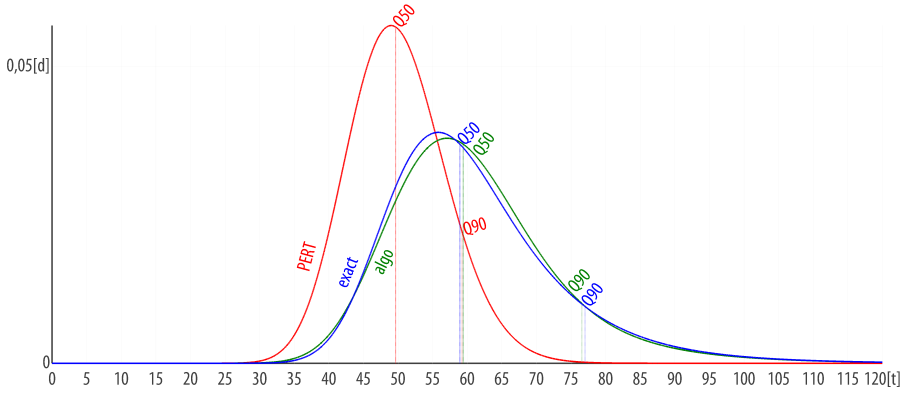

Fig. 9. Densities for the SGM from Fig. 5: PERT (red) vs. exact (blue) and our algorithm (green)

on the nodes). Fig. 9 displays the densities of the critical path (0-1-4-7-8), of the exact distribution (obtained by PRISM) and of the result by our algorithm. While our algorithm works quite accurately (submitting a real 89.57% confidence level as "90%"), PERT presents an actual level of about 50% as "90%"!

But what about CCPM? Table 2 considers the possible paths, assigned with their relating buffer. It is interesting that the more reliable quantile with 84.68% (at 72.53 time units) now comes from the C&PM. But there is still a considerable deviation to the wanted exact 90%-quantile (at 77.04 time units). RSEM now leads – with 79.96% (at 69.65 time units) – to an even worse result.

Table 2. Pathwise CCPM end times (+buffer) and corresponding quantiles

path	*0258*	*0158*	*01478*	*01468*	*01378*	*01368*	*overall*	*quantile*
C&PM	44.31	45.63	**48.35**	47.03	47.03	45.71	**72.53**	**84.68%**
	+22.16	+22.81	**+24.18**	+23.52	+23.52	+22.86		
RSEM	**44.31**	45.63	48.35	47.03	47.03	45.71	**69.65**	**79.96%**
	+25.34	+23.26	+10.15	+14.27	+14.27	+17.44		

For the prototypical calculation in our approach (on an I7-2600K CPU with 4 cores / 8 threads) we chose a stepwidth of $1‰ \cdot \mu_{min} = 0.01$ time units (where μ_{min} is the smallest mean of the node distributions), which resulted in a total machine time of 15.6 s. Another example (SGM with 19 nodes, 3 complex clusters to be analyzed by pMC, $\mu \in [5 \dots 10]$, $\sigma = \lfloor \frac{\mu}{2} \rfloor$) finished within 35.5 s under the same conditions. Increasing the number of nodes (each one now having a normal N(10,4) distribution, with a $1\% \cdot \mu_{min}$ stepwidth) showed, as expected, that the runtime grows strongly with the complexity of the contained clusters: For instance, a randomly generated SGM with 137 nodes (73 caught in clusters, max. cluster size: 5) was finished within 182 s. Another example with 133 nodes (77 caught in clusters, max. cluster size: 16) required 2,269 s to complete.

5 Summary and future work

The presented method offers a remarkable chance to improve the accuracy of established project planning methods by the combined use of exact calculations and heuristic approximations, using pMC. Estimates of the overall runtime distribution – even for complex graph structures – can be calculated more accurately and also – compared to the customary simulation-based approaches – with feasable computation effort. For instance, one of the leading management methods, CCPM, can be enhanced in several ways: For a given project schedule, one obtains a handy calculus of the time-to-finish distribution. Furthermore – reversely – for a given project finalization confidence level (e.g. a 90% quantile) we can determine all buffer sizes (on the critical path and all side paths) in an analytical manner.

This holds even if the scheduling complexity of the CCPM is increased by an additional calculus regarding the resource- or skill-dependencies (an extension already envisaged in our work plan). In that context, we will also use the presented method to solve resource conflicts of the "bad multitasking" type [4] by taking or hedging a founded decision for a particular prioritization. In addition, it might become necessary to use bounding methods (e.g. by inserting or removing edges) in order to make larger SGMs tractable.

References

1. Bobbio, A. et al. (2005). *Matching Three Moments with Minimal Acyclic Phase Type Distrbutions. Stochastic Models 21(2-3), p. 303-326.*
2. O'Cinneide, C.A. (1990). *Characterization of phase-type distributions. Communications in Statistics: Stochastic Models 6(1), p. 1-57.*
3. Dilmaghani, F. (2008). *Critical chain project management (CCPM) at Bosch Security Systems (CCTV) Eindhoven.* Masters Thesis, University of Twente
4. Goldratt, E.M. (1997). *Critical Chain.* The North River Press Publishing Corporation, Great Barrington.
5. Kapici, S. (2005). *A stochastic risk model for complex projects (Dissertation)* Otto-von-Guericke-Universität, Magdeburg
6. Klar, R. et al. (1995). *Messung und Modellierung paralleler und verteilter Rechensysteme.* B.G. Teubner, Stuttgart
7. Kulkarni, V.G and Adlakha, V.G. (1986). *Markov and Markov-regenerative PERT networks. Operations Research 34(5), p. 769-781.*
8. Kwiatkowska, M. et al. (2007). *Stochastic Model-Checking. Vol. 4486 of LNCS: Formal Methods for Performance Evaluation, p. 220-270.* Springer, Heidelberg
9. Maplesoft (2017). *Maple, a symbolic and numeric computing environment.* https://www.maplesoft.com/products/maple/
10. Melchiors, P. and Kolisch, R. (2007). *Scheduling of Multiple R&D Projects in a Dynamic and Stochastic Environment. In B. Fleischmann et al. (Eds), Operations Research Proceedings 2008. pp 135–140.* Springer, Heidelberg
11. Melchiors, P. (2015). *Dynamic and Stochastic Multi-Project Planning. Vol. 673 of Lecture Notes in Economics and Mathematical Systems.* Springer, Heidelberg
12. *The PRISM Model-Checker website.* http://www.prismmodelchecker.org
13. Shtub, A. et al. (2014). *Project Management: Processes, Methodologies and Economics (2nd edition).* Pearson Education Limited, Essex.
14. *Wolfram Mathematica.* https://www.wolfram.com/mathematica/