Splitting Techniques for Interval Parameters in Performance Models

JOHANNES LÜTHI

CATALINA M. LLADÓ

Bericht Nr. 2000-07

Dezember 2000

Splitting Techniques for Interval Parameters in Performance Models

Johannes Lüthi[1] and Catalina M. Lladó[2]

Technical Report No. 2000-07

[1]Institut für Technische Informatik
Universität der Bundeswehr München
Werner-Heisenberg-Weg 39, 85577 Neubiberg, Germany
email: luethi@informatik.unibw-muenchen.de
URL: www.informatik.unibw-muenchen.de/inst4/luethi

[2]Department of Computing
Imperial College of Science, Technology and Medicine
180 Queens Gate, London SW7 2BZ, United Kingdom
email: cl16@doc.ic.ac.uk

Abstract

During early phases of design and implementation, not all the parameter values of a performance model are usually known exactly. In related research contributions, intervals have been proposed as a means to capture parameter uncertainties. Existing model solution algorithms can be adapted to interval parameters by replacing conventional arithmetic by interval arithmetic. However, the so-called dependency problem may cause extremely wide intervals for the computed performance measures. Interval splitting has been proposed as a technique to overcome this problem. In this work we give an overview of existing splitting algorithms and propose a new selective splitting method that significantly reduces the computational complexity of interval evaluations. Moreover, the exploitation of partial monotonicity properties to further decrease the computational complexity is discussed. The proposed methods are illustrated along the lines of two examples: a small performance model of the MACA-BI protocol for ad-hoc wireless mobile networks and a more complex model of an Enterprise JavaBeans server implementation.

Keywords: Performance models; Analytic modeling; Parameter uncertainties; Interval parameters; Interval splitting

Contents

1 Introduction 2

2 Interval Parameters 4
2.1 Basic Definitions . . . 4
2.2 Interval Extensions . . . 5
2.3 Interval Arithmetic . . . 6
2.4 Dependency Problem . . . 7

3 Interval Splitting 8
3.1 Brute Force Interval Splitting . . . 8
3.2 Selective Interval Splitting . . . 9
3.3 Selective Interval Splitting with Midpoint Evaluation . . . 12
3.4 Exploitation of Single Parameter Occurrence . . . 14
3.5 Exploitation of Partial N-Monotonicity . . . 15
3.6 Summary of Splitting Considerations . . . 16

4 Interval Parameters in a Model for Wireless Mobile Networks 17
4.1 Normalized Throughput for the MACA-BI Protocol . . . 17
4.2 Experimental Results with Interval Model . . . 19

5 Model of an EJB Server Implementation 23
5.1 EJB Submodel to be Adapted to Interval Parameters . . . 23
5.2 EJB Submodel with Interval Parameters . . . 25
5.3 Experimental Results with Interval-Based EJB Model . . . 26

6 Conclusions 29

A Proof of Theorem 2 29

B Proof of Corollary 3 30

1 Introduction

Typically, two types of abstraction are required to build a performance model: firstly, the structural properties of a real system are modelled which can be characterized for example using a corresponding Petri Net or queueing network structure. Secondly, quantitative behavior such as e.g. service or inter-arrival times have to be characterized. The result of this abstraction step is usually a set of model parameters. However, especially in early phases of design and implementation, not every aspect of the real system may be known exactly. Such uncertainties may exist in both, structural as well as parametrical model aspects. This work deals with uncertainties associated with model parameters. The issue of model parameters being subject to uncertainties and variabilities is also addressed in more detail in [15].

The use of intervals to characterize parameter uncertainties in performance models has been introduced in related works [14, 24, 12]. There are many situations where parameter intervals occur naturally: although an exact value for a parameter may not be known, the designer may provide a reasonable range of values for that parameter. If parameters are obtained via measurement, confidence intervals are an important tool to increase the reliability of the results. Parameter intervals may also occur in a situation where bounding analysis is used at one level of a hierarchical model producing input parameter intervals on another level. Parameter intervals are also suitable for worst-case analysis as well as sensitivity studies. Furthermore, the numerical treatment of other approaches to model parameter uncertainties such as e.g. parameter histograms [10], or fuzzy number parameters [11] is based on intervals.

When parameters of an analytical model are characterized by intervals, performance measure intervals can be obtained by adapting existing solution algorithms and formulae for the corresponding model characterized by single value (SV) parameters. This adaptation is done by replacing conventional arithmetic by so-called interval arithmetic. I.e., basic operations and elementary functions for real numbers are replaced by corresponding arithmetic defined for intervals. There are two major advantages of using interval arithmetic as opposed to other techniques for uncertainty analysis like Monte-Carlo [21] and Quasi-Monte-Carlo [19] methods and sensitivity analysis (see for example [6] for a comparison of these two approaches in the context of Markov reward models): (a) results produced by interval analysis are safe performance bounds, i.e., it is guaranteed that the possible range of performance measures is always enclosed by the obtained interval results; (b) if interval splitting is applied, the accuracy of the obtained interval results is automatically known.

However, there is also a major drawback of using interval arithmetic: the so-called *dependency problem* may cause extremely wide intervals for the computed performance measures [18]. Interval splitting as an approach to overcome this problem has been proposed by Majumdar and Ramadoss: In [16], a brute force splitting algorithm is used to obtain reasonable tight performance measure intervals. In [20], following an approach proposed in [22], selective interval splitting with significantly reduced computational complexity is considered. In this paper, a new selective splitting algorithm is presented that uses occasional single value model evaluations to further reduce the overall computational complexity for producing interval results. In the case of so-called N-monotonicity (i.e., monotonicity w.r.t. *all* input parameters), Lüthi and Haring use monotonicity properties to obtain an efficient interval solution without interval splitting [12]. In this work we show that monotonicity w.r.t. just *one* or *more* interval parameters can also be exploited to obtain a more efficient interval splitting solution.

The application of the interval splitting algorithms presented in this paper is demonstrated along the lines of two models that we have adapted to interval parameters: a basic analytical model of the MACA-BI protocol for ad hoc wireless mobile networks and a more complex queueing network model of an Enterprise JavaBeans implementation. The corresponding models with conventional parameters are presented in [3] and [9], respectively.

The paper is organized as follows: since many readers may not be familiar with interval arithmetic, the corresponding mathematical background is summarized in Section 2. An overview of existing splitting approaches as well as our new splitting algorithm is presented in Section 3. This section also includes considerations on possibilities to exploit single parameter occurrence and partial monotonicity properties. Sections 4 and 5 provide two application examples for the proposed techniques. In Section 6, the results are summarized and conclusions are drawn.

2 Interval Parameters

Typically, performance measures in analytical models can be expressed as mathematical functions of a number of input parameters. These input parameters are usually real numbers or integers. Thus, we consider a real function f as follows:

$$\begin{aligned} f : \mathbb{R}^n &\to \mathbb{R}, \\ x = (x_1, \dots, x_n) &\to f(x). \end{aligned}$$

As discussed in Section 1, we are interested in using intervals as input parameters for performance measure functions. In the following, we give some basic definitions of intervals and related terms. A detailed introduction can be found in books like [18, 1, 17].

2.1 Basic Definitions

A real interval is a set of the form

$$X = [\underline{x}, \overline{x}] = \{x \in \mathbb{R} \mid \underline{x} \le x \le \overline{x}\},$$

where $\underline{x}, \overline{x} \in \mathbb{R}$ and $\underline{x} \le \overline{x}$. $\underline{x}$ and $\overline{x}$ are called *endpoints* of the interval. In particular, $\underline{x}$ is called *lower bound*, and $\overline{x}$ is called *upper bound* of the interval $X = [\underline{x}, \overline{x}]$. By $\mathbb{IR}$ we denote the set of all real intervals. An interval $X \in \mathbb{IR}$ is called *thin* if $\underline{x} = \overline{x}$, and it is called *thick*, if $\underline{x} < \overline{x}$. If S is a nonempty bounded subset of $\mathbb{R}$, we denote the *hull* of S by $\square S = [\inf(S), \sup(S)]$. The hull is the tightest interval enclosing S. E.g., $\square\{a, b\} = [a, b]$, if $a \le b$, and $\square\{a, b\} = [b, a]$, if $a > b$. We denote the set of interval vectors $X = (X_1, \dots, X_n)$ with n components by $\mathbb{IR}^n$. An element $X \in \mathbb{IR}^n$ is interpreted as the set of all vectors $x \in \mathbb{R}^n$ such that $x_i \in X_i$, $i = 1, \dots, n$. For example, in the case $n = 2$, this is a rectangle. An interval vector is also referred to as a *box*.

For a real function f, continuous on every closed box on which it is defined, the *range* of a box X is defined as:

$$f^*(X) = \square\{f(x) \mid x \in X\} = \{f(x) \mid x \in X\}.$$

Because of the continuity of f, the range is itself an interval:

$$f^*(X) = [\underline{f}, \overline{f}].$$

In general, the computation of the range is a constrained optimization problem with box constraints. I.e., the global minimum $\underline{f}(x) = \min_{x \in X} f(x)$ and the global maximum $\overline{f}(x) = \max_{x \in X} f(x)$, subject to $x \in X$ have to be found.

In the special case of so-called N-monotonic functions, the range can be computed using only real value evaluations of f with appropriate combinations of parameter interval endpoints as input parameters. To be more specific, let $f(x_1, \ldots, x_n)$ be monotonically increasing w.r.t. all parameters x_i, $i \in I$ and monotonically decreasing w.r.t. all parameters x_i, $i \in D$, where $I \cup D = \{1, \ldots, n\}$. Then the range of f with interval parameters $X_1 = [\underline{x}_1, \overline{x}_1], \ldots, X_n = [\underline{x}_n, \overline{x}_n]$ can be computed as follows:

$$f^*(X_1, \ldots, X_n) \quad = \quad [f(y_1, \ldots, y_n), \, f(z_1, \ldots, z_n)] \, ,$$

where $y_i = \underline{x}_i$, $z_i = \overline{x}_i$ if $i \in I$, and $y_i = \overline{x}_i$, $z_i = \underline{x}_i$ if $i \in D$. In [12], this situation is discussed in detail for the example of the *Mean Value Analysis* (MVA) algorithm (see for example the book [8]) for closed single class queueing networks. A generalized monotonicity theorem, allowing exploitation of partial N-monotonicity is given in Section 3.5.

2.2 Interval Extensions

For many performance measures, monotonicity properties do not hold and general optimization methods are often difficult to apply and of high computational complexity. Sometimes the exact range of a function need not be known, but an interval enclosing the range sufficiently tight may be adequate as well. Thus, in the following we introduce the concept of *interval extensions* and *interval arithmetic* for their efficient computation.

An interval function $F : I\!I\!R^n \rightarrow I\!I\!R$ is an *interval extension* of the real function $f : I\!R^n \rightarrow I\!R$ if (for simplification purposes we leave aside consideration of definition regions) [18]:

$$\begin{aligned} F(x) = f(x) \qquad & \text{for } x \in I\!R, \\ f(x) \in F(X) \qquad & \text{for all } x \in X \in I\!I\!R. \end{aligned} \tag{1}$$

Interval extensions provide enclosures of the range of a real function:

$$F(X) \supseteq \{f(x) | x \in X\}.$$

A property of interval functions that is important for interval splitting techniques that are considered in Section 3, is inclusion isotony. An interval function $F : I\!I\!R^n \rightarrow I\!I\!R$ is called *inclusion isotone* if for all $X, Y \in I\!I\!R$,

$$X \subseteq Y \Rightarrow F(X) \subseteq F(Y).$$

2.3 Interval Arithmetic

An important class of inclusion isotone interval extensions is obtained by interval evaluation of arithmetic expressions. This is done by defining elementary arithmetical operations and functions for intervals. Arithmetic expressions are subsequently defined as recursive combinations of constants, interval variables, elementary operations, and elementary functions. For a formal definition see the book [18]. Since most functions representing performance measures are actually arithmetic expressions, interval arithmetic can serve as a powerful tool to obtain interval extensions of performance measures.

On the set of intervals, the *elementary operations* $\circ \in \{+, -, \cdot, /, \hat{}\,\} =: \Omega$ are defined by setting:

$$X \circ Y = \Box\{x \circ y \mid x \in X, y \in Y\} = \{x \circ y \mid x \in X, y \in Y\}, \quad \forall \circ \in \Omega,$$

for all $X, Y \in I\!I\!R$ such that $x \circ y$ is defined for all $x \in X$, $y \in Y$.

The elements φ of a predefined set Φ of elementary continuous real functions are extended to interval arguments by defining:

$$\varphi(X) = \Box\{\varphi(x) \mid x \in X\} = \{\varphi(x) \mid x \in X\},$$

for all $X \in I\!I\!R$ such that $\varphi(x)$ is defined for all $x \in X$. Such a set Φ of elementary functions may for example include abs (absolute value), the square and square root functions, exp (exponential), ln (natural logarithm), or the trigonometric functions sin, cos, tan.

From monotonicity properties it follows that the elementary operations $\circ \in \{+, -, \cdot, /\}$ can be computed in terms of the end points of the intervals $X = [\underline{x}, \overline{x}], Y = [\underline{y}, \overline{y}] \in I\!I\!R$:

$$X \circ Y = \Box\{\underline{x} \circ \underline{y}, \underline{x} \circ \overline{y}, \overline{x} \circ \underline{y}, \overline{x} \circ \overline{y}\}.$$

In particular,

$$\begin{aligned}
X + Y &= [\underline{x} + \underline{y}, \overline{x} + \overline{y}], \\
X - Y &= [\underline{x} - \overline{y}, \overline{x} - \underline{y}], \\
X \cdot Y &= [\min(\underline{xy}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{xy}), \max(\underline{xy}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{xy})], \\
X/Y &= X \cdot [1/\overline{y}, 1/\underline{y}], \quad \text{if } 0 \notin Y.
\end{aligned}$$

Analogously, (piecewise) monotonicity of the elementary functions can be exploited to define their evaluations along the lines of computations with the interval endpoints of the argument. E.g., because of the monotonicity of the exponentiation function we know that for any $X = [\underline{x}, \overline{x}] \in I\!I\!R$, $\exp(X) = [\exp(\underline{x}), \exp(\overline{x})]$.

Using the interval extensions of elementary operations and functions, an arithmetic expression in n variables can be evaluated with intervals by substituting the variables by the corresponding intervals and step by step application of interval arithmetic. E.g., given the intervals $X_1 = [1, 2]$ and $X_2 = [4, 5]$, the arithmetic expression $f(x_1, x_2) = (2x_1 + x_2)x_1$ is evaluated as follows:

$$\begin{aligned} f(X_1, X_2) &= (2 \cdot [1,2] + [4,5]) \cdot [1,2] \\ &= ([2,4] + [4,5]) \cdot [1,2] \\ &= [6,9] \cdot [1,2] = [6,18]. \end{aligned}$$

Given an arithmetic expression $f(x_1, \dots, x_n)$, in [18] it is shown that the corresponding interval evaluation $f(X_1, \dots, X_n)$ is inclusion isotone and that it provides an enclosure of the range $f^*(X) = f^*(X_1, , \dots, X_n)$ of the real arithmetic expression. I.e.,

$$X_1' \subseteq X_1, \dots, X_n' \subseteq X_n \quad \Longrightarrow \quad f(X_1', \dots, X_n') \subseteq f(X_1, \dots, X_n), \tag{2}$$

and

$$f^*(X) = \{f(x) | x \in X\} \subseteq f(X). \tag{3}$$

Eq. (2) is originally proved in [17], Eq. (3) follows directly from the definitions of interval arithmetic.

2.4 Dependency Problem

As discussed above, interval arithmetic can serve as a tool to obtain interval extensions of real functions. However, due to an effect known as *dependency problem*, equality is often not obtained in Eq. (3). This effect is also known as overestimation. The root of the dependency problem is the memoryless nature of interval arithmetic if a parameter occurs multiple times in an arithmetic expression [18]. For every occurrence of a variable in an expression it is treated independently. For example, the expression $X - X$ is evaluated to $\{x_1 - x_2 \mid x_1, x_2 \in X\} = [\underline{x} - \overline{x}, \overline{x} - \underline{x}]$, instead of $\{x - x \mid x \in X\} = [0, 0]$.

Sometimes an expression can be re-formulated to reduce the number of occurrences of an interval parameter. Examples of this technique are presented in Subsections 4.1 and 5.2. However, in general the dependency problem may often cause crucial overestimation of the actual range of an evaluated function. For example, the iterative nature of the well-known MVA algorithm (N iterations for a queueing network with N jobs) causes an increasing number of parameter occurrences with increasing number of jobs. Fig. 1 shows the overestimation of the response time interval using the interval arithmetical evaluation of the MVA. In this example, a queueing network with two queueing centers is analyzed. One of the service demands and the terminal think time are characterized by interval parameters. The diagram shows the relative width of the interval evaluation in multiples of the actual range for the response time. For example with 10 jobs in the network, the response time interval obtained via interval arithmetic is more than 120 as wide as the actual interval of possible response times.

A way to overcome overestimation due to the dependency problem is to split the original input parameter intervals into subintervals and evaluate the arithmetic expression using these subintervals as input parameters. Approaches in that direction are discussed in the following section.

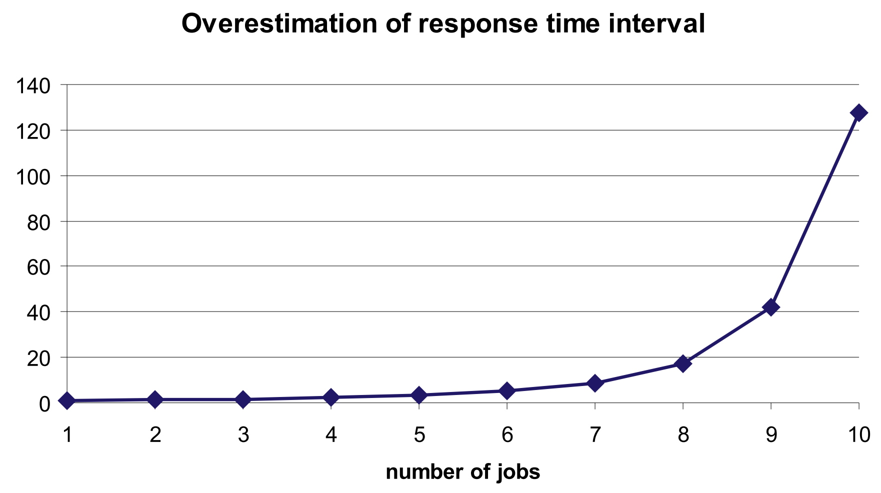


Figure 1: Overestimation of response time intervals with MVA.

3 Interval Splitting

The principal idea for interval splitting is to subdivide the input parameter intervals into several subintervals, compute interval evaluations of the arithmetic expression with the subintervals as input parameters, and find the overall result by computing the minimum of all lower bounds and the maximum of all upper bounds of the intermediate results. Analogously, an interval parameter vector (box) is split into subboxes. The basic idea is illustrated in the following subsection describing a brute-force splitting algorithm. Eq. (2) guarantees that results obtained via interval splitting yield enclosures of the range that are at most as wide as the interval evaluation using the original input parameter intervals. In [22] it is shown that the results obtained from interval splitting converge to the actual range if the width of the subintervals approaches zero. For the sake of readability, in the algorithms discussed in the following sections, we restrict the considerations to a single interval input parameter $X = [\underline{x}, \overline{x}]$. The generalization to multiple interval input parameters is straight forward. Considerations regarding computational complexity are however also included for the case of n interval parameters.

3.1 Brute Force Interval Splitting

In the *brute force splitting* (BFS) algorithm, in every iteration the input parameter intervals are split into two subintervals of equal length. The parameter (sub)intervals considered in iteration s (i.e. splitting degree s) are collected in P^s, the set of potential input parameter intervals. The respective algorithm is depicted in Fig. 2.

In step $S1$ of the BFS algorithm, after the initialization, the interval evaluation of f with the original parameter interval X is computed.

```
     Brute-force Splitting $(\boldsymbol{f}, \boldsymbol{X}, \boldsymbol{\epsilon})$
S1   $s \leftarrow 0$
     $P^0 \leftarrow \{X\}$
     $F^0 = [\underline{f}^0, \overline{f}^0] \leftarrow f(X)$
     do
S2     $s \leftarrow s + 1$
       $P^s \leftarrow \emptyset$
S3     $\forall Z = [\underline{z}, \overline{z}] \in P^{s-1}$ do begin
         $m = (\underline{z} + \overline{z})/2$
         $P^s \leftarrow P^s \cup \{[\underline{z}, m], [m, \overline{z}]\}$
       end
S4     $F^s = [\underline{f}^s, \overline{f}^s] \leftarrow \left[\min_{Z \in P^s} \underline{f(Z)}, \max_{Z \in P^s} \overline{f(Z)}\right]$
S5   until $(\underline{f}^s - \underline{f}^{s-1} < \epsilon)$ and $(\overline{f}^{s-1} - \overline{f}^s < \epsilon)$
```

Figure 2: Brute force splitting algorithm with one interval parameter.

In every iteration, the splitting degree s is incremented and a new set of input parameter intervals under consideration (P^s) is initialized (step $S2$). Subsequently, in step $S3$ of the algorithm, P^s is filled with subintervals of all intervals $X \in P^{s-1}$. Finally, the minimum of all lower bounds as well as the maximum of all upper bounds of evaluations of these subintervals is computed in step $S4$. Steps $S2 - S4$ are iterated until the difference between successive iterations becomes smaller than a predefined stopping criterion ϵ (step $S5$).

Note that the number of subintervals in consideration with splitting degree s is 2^s. More general, if n parameters are characterized as intervals (i.e. we have an n-dimensional input parameter box), it holds that $|P^s| = 2^{sn}$. The application of the BFS algorithm for the solution of interval-based computer performance models is proposed in [16].

3.2 Selective Interval Splitting

In the course of computation in the BFS algorithm, it can be observed that not necessarily every interval in P^s needs to be considered for further splitting. Consider for example the following situation: let Y and Z be two parameter subintervals in P^s. We denote the respective interval evaluations by $f(Y) = [\underline{f(Y)}], \overline{f(Y)}]$ and $f(Z) = [\underline{f(Z)}, \overline{f(Z)}]$. If $\underline{f(Z)} > \overline{f(Y)}$, we know that the actual lower bound $\underline{f}$ of the range $f^*(X) = [\underline{f}, \overline{f}]$ can not be obtained by evaluation of any point $x \in Z$, since we know that $\underline{f} \leq \overline{f(Y)}$.

More general, if we denote the set of parameter subintervals $Z \in P^{s-1} \cup P^s$ that have already been evaluated by P_{ev}, we know that:

$$\underline{f} \leq \min_{Y \in P_{\mathrm{ev}}} \overline{f(Y)}.$$

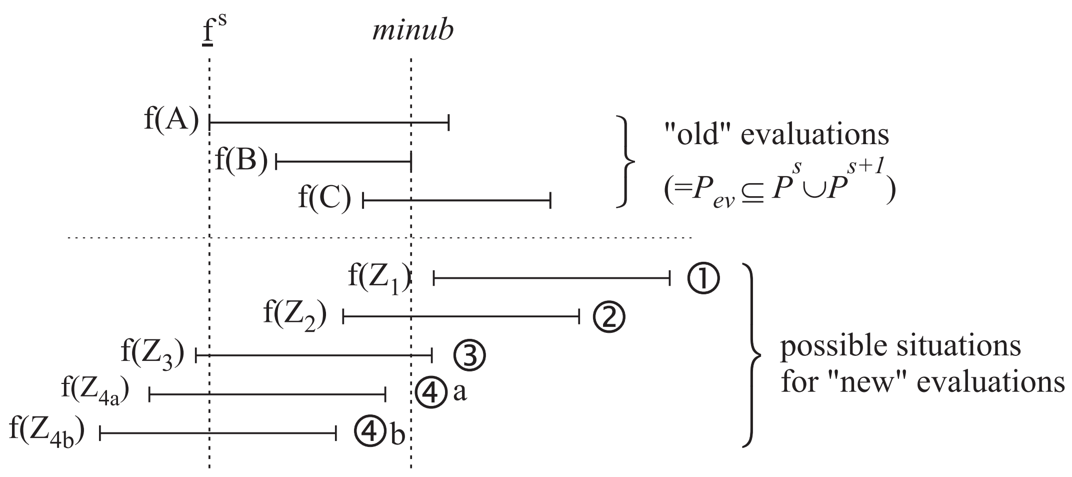


Figure 3: 4+1 situations to be considered in selective interval splitting.

Thus, in the situation described above, the subinterval Z need not be considered for further splitting to find the lower bound of the range $f^*(X)$. This idea of selective interval splitting was introduced by Skelboe in the context of general purpose optimization of rational interval functions [22] and its application to performance models is presented in [20].

In the following we consider the computation of the lower bound $\underline{f}$ of the range f^*. Computation of the upper bound $\overline{f}$ can be done analogously. If an interval Z_i with interval evaluation $f(Z_i) = [\underline{f(Z_i)}, \overline{f(Z_i)}]$ is considered to be included in the set of P^s that may potentially produce the lower bound $\underline{f}$ of the range $f^*(X)$, four situations have to be distinguished for efficient selective interval splitting. These situations are depicted in Fig. 3. In this figure, $\underline{f}^s = \min_{Y \in P_{\text{ev}}} \underline{f(Y)}$, and $minub = \min_{Y \in P_{\text{ev}}} \overline{f(Y)}$:

1. In the first situation, $\underline{f(Z_1)} > minub$. Thus, in the sequel Z_1 can be ignored w.r.t. the search for $\underline{f}$.

2. In the second situation, $\underline{f(Z_2)} \leq minub$. This means that Z_2 is a parameter interval that may eventually produce $\underline{f}$. However, neither $\underline{f}^s$ nor $minub$ are affected by $f(Z_2)$.

3. In the case depicted as situation three, additionally, $\underline{f(Z_3)} < \underline{f}^s$. Thus, by inclusion of Z_3 in P_{ev}, $\underline{f}^s$ has to be updated to the value $\underline{f(Z_3)}$.

4. In the fourth situation, we have $\overline{f(Z_{4ab})} < minub$. Thus, $minub$ has to be updated. Furthermore, there may now eventually be some intervals $Y \in P_{\text{ev}}$ such that $\underline{f(Y)} > minub$. Such Y are no longer of interest for finding $\underline{f}$. Thus, they should be removed from P^s or P^{s-1}. Situation 4a in Fig. 3 shows the case where non of the intervals in P_{ev} is affected, whereas in situation 4b the interval C would be dropped from the set of potential interval parameters since $\underline{f(C)} > \overline{f(Z_{4b})}$.

The situations described above can be exploited to integrate filtering mechanisms into the BFS algorithm that dramatically reduce the number of interval evaluations that have to be

```
        Selective Interval Splitting ($\boldsymbol{f}, \boldsymbol{X}, \boldsymbol{\epsilon}$)
$S1$    $s \leftarrow 0$
        $P^0 \leftarrow \{X\}$
        $F^0 = [\underline{f}^0, \overline{f}^0] \leftarrow f(X)$
        $minub \leftarrow \overline{f}^0$
        do
$S2$        $s \leftarrow s + 1$
            $\underline{f}^s \leftarrow \infty$
            $P^s \leftarrow \emptyset$
$S3$        $\forall Z = [\underline{z}, \overline{z}] \in P^{s-1}$ do begin
$S4$            $m \leftarrow (\underline{z} + \overline{z})/2$
                $Z_1 \leftarrow [\underline{z}, m]$; $Z_2 \leftarrow [m, \overline{z}]$
$S5$            for $i \leftarrow 1, 2$ do begin
$S6$                if ($\underline{f(Z_i)} \leq minub$ then begin
$S7$                    $P^s \leftarrow P^s \cup \{Z_i\}$
$S8$                    if ($\underline{f(Z_i)} < \underline{f}^s$) then $\underline{f}^s \leftarrow \underline{f(Z_i)}$
$S9$                    if ($\overline{f(Z_i)} < minub$ then begin
$S10$                       $minub \leftarrow \overline{f(Z_i)}$
$S11$                       check_lb ($P^{s-1}$); check_lb ($P^s$)
                    end
                end
            end
        end
$S12$   until ($\underline{f}^s - \underline{f}^{s-1} < \epsilon$)
```

Figure 4: Selective splitting algorithm with one interval parameter (considering lower bound only).

```
        check_lb ($\boldsymbol{P}$)
$S1$    $\forall Y \in P$ do
$S2$        if $\underline{f(Y)} > minub$ then $P \leftarrow P \setminus \{Y\}$
```

Figure 5: Checking routine for the selective splitting algorithm.

computed to obtain a sufficiently tight enclosure of the range $f^*(X)$. In Fig. 4, the *selective interval splitting* (SIS) algorithm is depicted. Note, that in the depicted algorithm, only the lower bound of the enclosure of the range is considered. The upper bound can be computed analogously. The initialization (step $S1$) is extended by setting the minimum of all upper bounds to ∞ (i.e., MAXREAL or some equivalent value). In the iteration initialization (step $S2$), the lower bound of the enclosure is also initialized to ∞. As in the BFS algorithm, every parameter interval in P^{s-1} (step $S3$)) is split into two intervals (step $S4$) which are subsequently considered for further treatment (step $S5$). However, situation (1) in Fig. 3 is filtered by the condition in step $S6$ of the SIS algorithm. I.e., only those parameter intervals that may eventually produce the lower bound of the enclosure are considered. This means that in the sequel, we are dealing with situations (2) to (4). In step $S7$, the parameter intervals of interest are included in the next set P^s of potential parameter intervals. Step 8 deals with situation (3) of Fig. 3. I.e., it is decided whether the lower bound $\underline{f^s}$ of the enclosure of iteration s has to be updated. Situation (4) is managed in step $S9$ of the algorithm. The minimum of all upper bounds is eventually updated (step $S10$) and in step $S11$ it is checked (the checking routine is listed in Fig. 5), whether parameter intervals have to be removed from P^{s-1} or from P^s due to the update of *minub*. Steps $S2$ to $S11$ are iterated until the change in the lower bound $\underline{f^s}$ of the enclosure is sufficiently small (step $S12$). In the subroutine depicted in Fig. 5, all intervals Y in the set P are checked (step $S1$) and eventually removed from P if $\underline{f(Y)} > minub$ (step $S2$).

If the lower and upper bounds of the enclosure are computed simultaneously, identical parameter evaluations can be used to optimize the computational effort. To accelerate the decisions in the *check_lb* routine, the sets P^{s-1} and P^s can be implemented as sorted linear lists as it is also suggested in [22]. Note that for multiple (n) interval parameters, every parameter box $X \in P^{s-1}$ has to be split in 2^n subboxes. However, as it is illustrated in Section 5, after some initial iterations the number of interval evaluations eventually increases only linearly with the splitting degree s. This is due to the filtering effect of the SIS algorithm.

3.3 Selective Interval Splitting with Midpoint Evaluation

Given an interval $Z = [\underline{z}, \overline{z}] \in P^{s-1}$, consider the lower bound $\underline{f^*(Z)}$ of the range $f^*(Z)$. In the SIS algorithm discussed in the previous section, we use the fact that $\underline{f^*(Z)} \in f(Z) = [\underline{f(Z)}, \overline{f(Z)}]$ for the selection process. To be more specific, the selection threshold *minub* is chosen as the minimum of the values $\overline{f(Z)}$ for all parameter intervals Z that have already been evaluated. However, by definition we also know that $\underline{f^*(Z)} \leq f(z)$ for any $z \in Z$. I.e., the real function evaluation $f(z)$ for any point $z \in Z$ is also an upper bound for the lower bound of the range $f^*(Z)$. Thus, we know that $\underline{f^*(Z)} \in [\underline{f(Z)}, f(z)]$ for any arbitrary $z \in Z$. This yields indeed a sharper threshold than $\overline{f(Z)}$, since also $f(z) \leq \overline{f(Z)}$ holds for any $z \in Z$.

In the *selective splitting with midpoint evaluation* (SSME) algorithm, depicted in Fig. 6, the interval midpoint $z = (\underline{z} + \overline{z})/2$ of every interval $Z = [\underline{z}, \overline{z}] \in P^{s-1}$ is used to obtain a sharper threshold for the selection decision. Steps $S1$ to $S8$ of the SSME algorithm are

```
        Selective Splitting with Midpoint Evaluation $(\boldsymbol{f}, \boldsymbol{X}, \boldsymbol{\epsilon})$
$S1$    $s \leftarrow 0$
$S2$    $minub \leftarrow f((\underline{x} + \overline{x})/2)$
$\vdots$       $\vdots$
$S9$                $f_m \leftarrow f((\underline{z_i} + \overline{z_i})/2)$
$S10$               if $(f_m < minub)$ then begin
$S11$                 $minub \leftarrow f_m$
$S12$                 check_lb $(P^{s-1})$; check_lb $(P^s)$
                    end
                  end
                end
              end
$S13$   until $(\underline{f}^s - \underline{f}^{s-1} < \epsilon)$
```

Figure 6: Selective splitting algorithm with midpoint evaluation, one interval parameter (considering lower bound only).

identical to the SIS algorithm and are thus not listed in Fig. 6. However, in step $S9$, f_m is assigned the real function evaluation of the interval midpoint. In the sequel (steps $S10$ to $S12$), f_m is used to determine the threshold *minub* which is subsequently used to decide which parameter intervals are of interest for further investigation.

Fig. 7 shows the effect of the sharper decision threshold *minub* obtained via evaluation of the interval midpoints. The bullets in that figure represent the real function evaluations of the respective parameter interval midpoints. Here, the real function evaluations of the midpoint are assumed to be in the middle of the interval evaluations. This does of course not hold in general, but does not effect the principal mechanism of midpoint evaluation as an additional filtering technique. We denote the midpoint of parameter interval Y by m_y. Note that as opposed to Fig. 3, in Fig. 7 the threshold *minub* is determined as the minimum of all midpoint evaluations. Thus, using the SSME algorithm, the parameter interval C would not have been included in P_{ev} in the first place because $\underline{f(C)} > f(m_a) = minub$. Considering the parameter intervals Z_1 to Z_{4b} in Fig. 7 we can observe that both Z_1 and Z_2 need not be split any further, since from $\underline{f(Z_2)} > minub$ we conclude that $\underline{f} \notin f(Z_2)$. I.e., as opposed to the conventional selective splitting algorithm, Z_2 is filtered out due to the sharper selection threshold obtained via midpoint evaluation. Next, consider situation (4a): using conventional SIS, the threshold *minub* is updated, but non of the parameter intervals in P_{ev} is affected by that update. The situation with SSME is different: because $f(m_{Z_{4a}}) < \underline{(f(B)}$, the parameter interval B can be removed from the set of parameter intervals of further

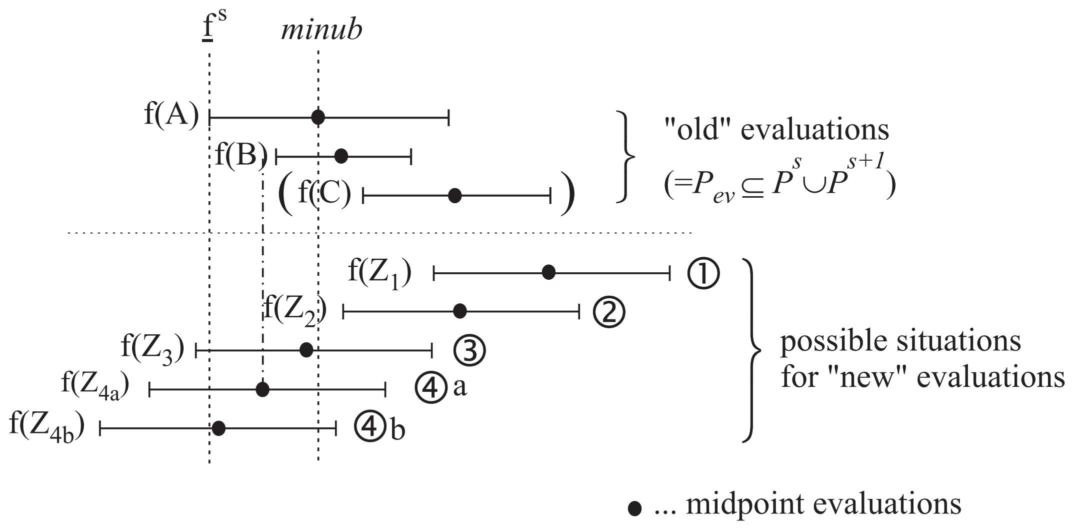


Figure 7: Effect of midpoint evaluation for interval splitting.

interest.

This example as well as the experimental results presented in Sections 4 and 5 illustrates that the additional filtering effect of the SSME algorithm may usually be worth the higher computational expense due to the additional real function evaluation necessary to obtain the tighter threshold.

Due to continuity properties, functions describing performance measures are usually at least piecewise monotonic. Thus, using interval endpoints instead of the interval midpoint to obtain an even tighter threshold may be heuristically argued. However, even with just a single interval parameter, this would double the additional effort due to real function evaluations and experiments have shown that often the improvement of the filtering effect is almost negligible. Furthermore, this approach does not scale well to multiple interval parameters, because in the lack of knowledge of monotonicity behavior, given n interval parameters, real function evaluations on all 2^n corners of the n-dimensional parameter box would have to be computed.

Instead of the stopping criterion of the algorithms listed in Figs. 4 and 6 ($\underline{f}^s - \underline{f}^{s-1} < \epsilon$), an alternative stopping criterion can be used: $\underline{f}^s - minub < \epsilon$. This stopping criterion automatically provides a bound (namely ϵ) on the difference between the lower endpoint $\underline{f}$ of the actual range $f^*(X) = [\underline{f}, \overline{f}]$ and the lower endpoint $\underline{f}^s$ of the enclosure obtained via interval splitting, because the relation $\underline{f} \in [\underline{f}^s, minub]$ holds.

3.4 Exploitation of Single Parameter Occurrence

As it is discussed in Subsection 2.4, the overestimation due to the dependency problem stems from multiple occurrences of one or several input parameters in an expression. However, if not all input parameters occur more than once in the arithmetic expression, the following theorem holds.

Theorem 1 (Moore) *Let $f(\xi_1, \ldots, \xi_n, \nu_1, \ldots, \nu_m) = f(\xi, \nu)$ be an arithmetical expression in $n+m$ variables. Suppose that the variables ν_k, $k = 1, \ldots, m$, occur only once in f. Given interval vectors $X \in I\!I\!R^n$, $Y \in I\!I\!R^m$, it holds that:*

$$f^*(X, Y) = \Box\{f(x, y) \mid x \in X, y \in Y\} = \bigcup_{x \in X} f(x, Y).$$

A proof of this theorem can be found in [17] or in [18]. The interpretation of this theorem is that finding the range of an expression with interval parameters is a classical optimization problem only w.r.t. those parameters that occur multiple times. Parameters that occur only once in the arithmetic expression can be treated with interval arithmetic without producing additional overestimation. Regarding interval splitting, this means that in principal only parameters that occur more than once in the expression have to be split into subintervals. Thus, in the BFS approach, the number of splitting combinations in iteration s is reduced from $2^{s(n+m)}$ to 2^{sn}.

However, in the selective splitting approaches, avoiding to split certain parameter intervals may cause extremely low effectiveness of the filtering mechanism of these algorithms. In many situations, reducing the number of split parameter intervals via exploitation of single occurrence of parameters may cause the selective approaches to behave like the BFS algorithm. I.e., eventually no filtering may take place. Consider the following simple example: $f(X, Y) = X + X + Y$, $X = [0, 1]$, $Y = [0, 10]$. Since Y appears only once in the expression of f, only X might be considered for splitting. Now consider the evaluation of f using the following subintervals of X: $X' = [0, \epsilon]$ and $X'' = [1 - \epsilon, 1]$. Evaluation of f yields: $f(X', Y) = [0, 10 + 2\epsilon]$ and $f(X'', Y) = [2 - 2\epsilon, 12]$ for any arbitrarily small $\epsilon > 0$. Since $2 - 2\epsilon < 10 + 2\epsilon$, even X'' is not filtered out by selective splitting. From this we may conclude that however small the subintervals of X are chosen, non of them is filtered out by selective splitting. Thus, in this case the selective splitting approach behaves like the BFS algorithm. Depending on the desired accuracy, reducing the number of parameter intervals that are split at the cost of switching from selective to (almost) brute force splitting may or may not reduce the total computational complexity. This effect is also illustrated in the experimental results presented in Section 4.

3.5 Exploitation of Partial N-Monotonicity

Monotonicity of the evaluated expression w.r.t. one or more input parameters can be exploited to reduce the number of parameters that have to be split. To obtain the range of an expression, two separate runs of the interval splitting algorithm can be performed. This yields the lower and upper bounds of an enclosure of the range, respectively. In the two runs of the splitting algorithm, parameter intervals for parameters with monotonicity properties can be replaced by appropriate endpoints of the original parameter intervals. This reduces the number of interval parameters in the splitting algorithm, and hence drastically reduces the computational complexity. The following theorem and associated corollary provide the formal justification for this simplification.

Theorem 2 *Let $f(\xi_1, \ldots, \xi_n, \nu_1, \ldots, \nu_m, \mu_1, \ldots, \mu_l) = f(\xi, \nu, \mu)$ be an arithmetical expression in $n + m + l$ variables. Suppose that f is monotonic increasing w.r.t. ν_i, $i = 1, \ldots, m$ if all other parameters are fixed and f is monotonic decreasing w.r.t. μ_j, $j = 1, \ldots, l$ if all other parameters are fixed. Given interval vectors $X \in I\!I\!R^n$, $Y \in I\!I\!R^m$, and $Z \in I\!I\!R^l$ it holds that:*

$$\begin{aligned} f^*(X, Y, Z) &= \square\{f(x, y, z) \mid x \in X, y \in Y\, z \in Z\} \\ &= \left[\inf_{x \in X} f(x, \underline{y}, \overline{z}),\ \sup_{x \in X} f(x, \overline{y}, \underline{z})\right]. \end{aligned}$$

The proof of Theorem 2 can be found in Appendix A.

Corollary 3 *Under the assumptions of Theorem 2 it holds that:*

$$f^*(X, Y, Z) \subseteq \left[\underline{f(X, \underline{y}, \overline{z})},\ \overline{f(X, \overline{y}, \underline{z})}\right].$$

The corollary is proved in Appendix B. Corollary 3 guarantees that an enclosure for the range $f^*(X, Y, Z)$ can be obtained as follows: compute the evaluation of the expression where parameters with monotonic increasing effect are replaced by the respective lower bounds of the parameter intervals, and where parameters with monotonic decreasing effect are replaced by the respective upper bounds of the parameter intervals. This evaluation yields an interval because the input parameters without monotonicity properties are still intervals. The lower bound of this evaluation interval yields the lower bound of the enclosure. The upper bound of the enclosure is computed analogously. The amount of overestimation of that enclosure can be reduced by interval splitting as described in the previous sections. However, Theorem 2 implies that only the parameter intervals $X = (X_1, \ldots, X_n)$ without known monotonicity properties have to be split whereas parameters with monotonicity can be replaced by appropriate single values.

3.6 Summary of Splitting Considerations

In the previous subsections, possibilities to exploit different parameter characteristics to optimize the SIS and SSME algorithms are discussed. In particular, three properties are used to handle input parameters: (1) an input parameter can be an interval or a single value (i.e., thick or thin), (2) it may occur once or multiple times, (3) monotonic behavior of the arithmetic expression may exist (and be known) or not. These parameter properties and associated parameter treatment in the splitting algorithm are summarized in Table 1. In this table, the notation "$-$" stands for "don't care".

Fig. 8 lists the *generalized splitting algorithm* (GSA), which uses calls of the SSME algorithm and considers special parameter properties. In this listing, X_{IvM} denotes a vector of (thick) interval parameters that occur multiple times in the arithmetic expression. X_{IvS} denotes a vector of interval parameters that are not supposed to be split. $X_{\text{Mon}+}$ ($X_{\text{Mon}-}$) denotes a vector of parameter intervals that have monotonic increasing (decreasing) effect.

Table 1: Summary of parameter properties for interval evaluation.

Parameter properties			Treatment in splitting algorithm
Thin	Single occurrence	Monotonicity	
yes	–	–	Treat as real number (no special consideration)
–	yes	–	Treat as interval without splitting or split in interval splitting algorithm (depending on situation)
no	no	yes	Double call of interval splitting with appropriate endpoints as single values
no	no	no	Split in interval splitting algorithm

Finally, x_{SV} denotes a vector of single value parameters (or thin intervals). If there are no parameters with known monotonicity properties, the SSME algorithm is called only once (step $S1$). The notation SSME $(f(X_{\mathrm{IvM}}, X_{\mathrm{IvS}}, x_{\mathrm{SV}}), X_{\mathrm{IvM}}, \epsilon)$ indicates that f is evaluated with all parameters ($X_{\mathrm{Mon+}}$ and $X_{\mathrm{Mon-}}$ are dropped because they do not exist in that case), but only the parameters X_{IvM} are supposed to be split. If there are parameters with known monotonicity properties, the SSME algorithm is called with the lower bounds $\underline{x_{\mathrm{Mon+}}}$ of parameter intervals in $X_{\mathrm{Mon+}}$ and with the upper bounds $\overline{x_{\mathrm{Mon-}}}$ of parameter intervals in $X_{\mathrm{Mon-}}$ to obtain the lower bound $\underline{f}$ of the evaluation (step $S2$). Analogously, the upper bound $\overline{f}$ of the evaluation is obtained by calling the SSME algorithm with the upper bounds $\overline{x_{\mathrm{Mon+}}}$ of parameter intervals in $X_{\mathrm{Mon+}}$ and with the lower bounds $\underline{x_{\mathrm{Mon-}}}$ of parameter intervals in $X_{\mathrm{Mon-}}$ (step $S3$).

4 Interval Parameters in a Model for Wireless Mobile Networks

As an illustrative example we use a model of the MACA-BI (Multiple Access with Collision Avoidance By Invitation) protocol for so-called *ad hoc wireless mobile networks* described by Gerla et al. [3].

4.1 Normalized Throughput for the MACA-BI Protocol

Using the following parameters:

$\lambda \dots$ the aggregate rate of packet generation,

```
        **Generalized Splitting** $(\boldsymbol{f}(\boldsymbol{X}_{\mathrm{IvM}}, \boldsymbol{X}_{\mathrm{IvS}}, \boldsymbol{X}_{\mathrm{Mon}+}, \boldsymbol{X}_{\mathrm{Mon}-}, \boldsymbol{x}_{\mathrm{SV}}), \boldsymbol{\epsilon})$
        if $(X_{\mathrm{Mon}+} \cup X_{\mathrm{Mon}-}) = \emptyset$ then
$S1$      $F = [\underline{f}, \overline{f}] \leftarrow$ SSME $(f(X_{\mathrm{IvM}}, X_{\mathrm{IvS}}, x_{\mathrm{SV}}), X_{\mathrm{IvM}}, \epsilon)$
        else begin
$S2$      $\underline{f} \leftarrow \underline{\mathrm{SSME}}(f(X_{\mathrm{IvM}}, X_{\mathrm{IvS}}, \underline{x_{\mathrm{Mon}+}}, \overline{x_{\mathrm{Mon}-}}, x_{\mathrm{SV}}), X_{\mathrm{IvM}}, \epsilon)$
$S3$      $\overline{f} \leftarrow \overline{\mathrm{SSME}}(f(X_{\mathrm{IvM}}, X_{\mathrm{IvS}}, \overline{x_{\mathrm{Mon}+}}, \underline{x_{\mathrm{Mon}-}}, x_{\mathrm{SV}}), X_{\mathrm{IvM}}, \epsilon)$
        end
```

Figure 8: Generalized interval splitting algorithm considering different parameter types.

$\gamma \ldots$ the control packet length,

$\tau \ldots$ the maximum propagation time,

$\delta \ldots$ the data packet length,

in [3], an expression for the normalized throughput of the single hop case is derived:

$$S = \frac{\delta}{\delta + \frac{2-\mathrm{e}^{-\tau\lambda}}{\lambda} + (\gamma + 2\tau)\mathrm{e}^{\tau\lambda}}.$$

In this expression, the packet length parameter δ occurs twice, λ and τ occur three times, just γ occurs only once. Thus, it is likely that S is subject to the dependency problem causing overestimation of throughput intervals if intervals are used for the model parameters. As discussed in Subsection 2.4, the widening effect of the dependency problem can sometimes be decreased if the expression is rewritten such that interval parameters occur less often. If S is rewritten in the following way:

$$S' = \frac{1}{1 + \left(\frac{2-\mathrm{e}^{-\tau\lambda}}{\lambda} + (\gamma + 2\tau)\mathrm{e}^{\tau\lambda}\right)/\delta},$$

the packet length parameter δ occurs only once whereas the number of occurrences of the other parameters does not change. In the sequel we refer to the expression S as the *original* throughput expression, S' is referred to as the *optimized* expression.

Derivation of the expression w.r.t. the four parameters yields that S is monotonically increasing w.r.t. δ and S is monotonically decreasing w.r.t. τ and γ as long as all parameters are positive. In the following comparisons, these monotonicity properties can be used to reduce the number of interval parameters in the splitting algorithms as discussed in Section 3.5.

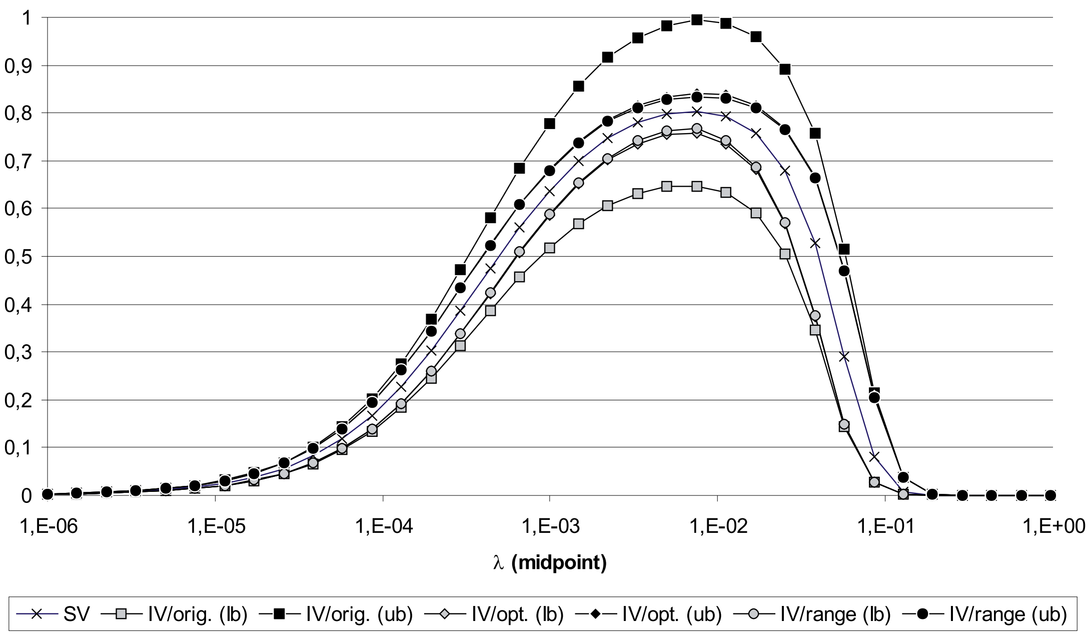


Figure 9: Single value (SV) as well as various interval (IV) results for the normalized throughput of the example model with varying network load λ.

4.2 Experimental Results with Interval Model

For the experiments discussed in this section, we use the single value parameters from [3] (data packet length = 296 bytes, control packet length is 20 bytes, propagation delay is 54 μs, channel speed is 1Mbps), as a bases and assume parameter uncertainties in the way that every parameter ξ is described as $\xi \pm 10\%$, i.e., as the interval $[0.9\xi, 1.1\xi]$. Along the lines of [3], using bits and μs as units, the parameter intervals are:

- $D = [\underline{\delta}, \overline{\delta}] = 2368 \pm 10\% = [2131.2,\ 2604.8]$,
- $G = [\underline{\gamma}, \overline{\gamma}] = 160 \pm 10\% = [144,\ 176]$,
- $T = [\underline{\tau}, \overline{\tau}] = 54 \pm 10\% = [48.6,\ 59.4]$.

The load parameter λ is varied (logarithmically scaled) from 10^{-6} to 1.0 and is also subject to an uncertainty of $\pm 10\%$. In each step, the load factor λ is increased by the factor 1.5. Fig. 9 depicts the results of this experiment, including the single value (SV) normalized throughput, results from interval evaluation of S (IV/orig) and S' (IV/opt) as well as the actual range of the interval evaluation (IV/range). 'lb' denotes lower bounds and 'ub' denotes upper bounds. In this figure, the overestimation caused by the dependency problem can be observed. Direct interval evaluation of S' yields almost exactly the range, whereas evaluation of S yields much wider intervals. However, this fact can only be recognized if the range is known. Thus, interval splitting has to be applied to gain control over the accuracy of the interval results.

Table 2 lists the results for the computational complexity when using the various interval splitting approaches (BFS, SIS, SSME) and different values for the desired accuracy $\epsilon = 10^{-2}, \ldots, \epsilon^{-6}$. For a comparison of the computational complexity of the different splitting algorithms, four variants of the interval model are considered: in the variant denoted as 'NAIVE', monotonicity properties and the single occurrence of parameters are not exploited, i.e., all four parameter intervals are split into subintervals. In the 'ESO' (exploitation of single occurrence) variant, monotonicity properties are not used but parameter intervals that occur only once in the expression are not split (see Subsection 3.4). In the variant denoted by 'MONO 2', monotonicity properties of S w.r.t. δ and γ are exploited, whereas λ and τ are split (see Subsection 3.5). Finally, in the variant 'MONO', all known monotonicity properties are exploited and thus only λ has to be considered for interval splitting. For the 'NAIVE' and 'ESO' variants, two sub-variants are considered: using the original expression S ('orig.') and using the rewritten expression S' ('opt.'). This distinction is not made for the 'MONO2' and 'MONO' variants, because in these two, the parameter δ is treated via its interval endpoints anyway (only the number of occurrences of δ is reduced if using S' instead of S). The table lists the number of necessary interval evaluations for the various splitting algorithms. In the case of the SSME algorithm, also the number of necessary single value (SV) evaluations and the weighted sum $iv + sv/2$ is listed (a SV evaluation is estimated to be of approximately half the computational complexity as an interval evaluation). The values represent the total numbers of evaluations necessary to gain all results depicted in Fig. 9, i.e., results for 35 different λ intervals. Omitted results reflect experiments that had to be aborted because of time and/or memory constraints.

Several observations can be inferred from the results of Table 2: in general, BFS is not an option for efficient interval evaluation as compared to selective interval splitting such as SIS or SSME. Comparing SIS and SSME, the SSME approach always manages to further reduce the number of interval evaluations. In the SSME algorithm, the reduction of interval evaluations is achieved at the cost of additional SV evaluations (the midpoint evaluations). However, almost always the total cost of SSME is smaller than that of SIS. As discussed in Section 3.4, exploitation of single parameter occurrence disables the filtering effect of the selective splitting techniques as soon as the desired accuracy gets small. Thus, avoiding to split parameter intervals that occur only once in the evaluated expression only makes sense for BFS. However, as the results in Table 2 suggest, it is more efficient to use a selective splitting algorithm (SIS or SSME) without exploitation of single parameter occurrence than to use BFS with exploitation of single parameter occurrence. Another important conclusion that can be drawn from the results of these experiments is that rewriting an expression to reduce the number of parameter occurrences may significantly reduce the necessary computational effort for interval splitting. This effect can be seen by comparing the 'Orig.' with the 'Opt.' results. Furthermore, exploitation of monotonicity properties is even more important, because only parameters without known monotonicity properties have to be split. Thus, with exploitation of monotonicity the problem dimension in the splitting algorithms can be

Table 2: Complexity results for example.

Accuracy	Variant	BFS	SIS	SSME		
		IV eval.	IV eval.	IV eval.	SV eval.	IV+SV/2
$\epsilon = 10^{-2}$	NAIVE/Orig.	1,216,678	414,998	178,822	32,952	195,298
	NAIVE/Opt.	1,190	1,190	1,190	802	1,591
	ESO/Orig.	83,830	57,614	31,150	13,166	37,733
	ESO/Opt.	350	350	350	341	521
	MONO 2	350	350	350	341	521
	MONO	210	210	210	178	299
$\epsilon = 10^{-3}$	NAIVE/Orig.	–	4,741,110	1,302,758	167,421	1,386,469
	NAIVE/Opt.	247,206	10,790	4,598	1,701	5,449
	ESO/Orig.	–	–	–	–	–
	ESO/Opt.	2,094	2,094	2,038	1,990	3,033
	MONO 2	2,094	850	634	414	841
	MONO	414	342	320	256	448
$\epsilon = 10^{-4}$	NAIVE/Orig.	–	–	3,831,862	486,351	4,075,038
	NAIVE/Opt.	–	59,702	20,950	5,726	23,813
	ESO/Orig.	–	–	–	–	–
	ESO/Opt.	168,334	164,962	144,590	143,354	216,267
	MONO 2	168,334	2,702	1,742	950	2,217
	MONO	3,110	760	668	511	924
$\epsilon = 10^{-5}$	NAIVE/Orig.	–	–	–	–	–
	NAIVE/Opt.	–	235,302	85,350	21,091	95,896
	ESO/Orig.	–	–	–	–	–
	ESO/Opt.	–	–	–	–	–
	MONO 2	–	7,226	4,426	2,197	5,525
	MONO	28,630	1,652	1,404	1,034	1,921
$\epsilon = 10^{-6}$	NAIVE/Orig.	–	–	–	–	–
	NAIVE/Opt.	–	1,005,974	374,310	90,240	419,430
	ESO/Orig.	–	–	–	–	–
	ESO/Opt.	–	–	–	–	–
	MONO 2	–	21,960	12,882	5,951	15,858
	MONO	301,902	4,718	3,922	2,822	5,333

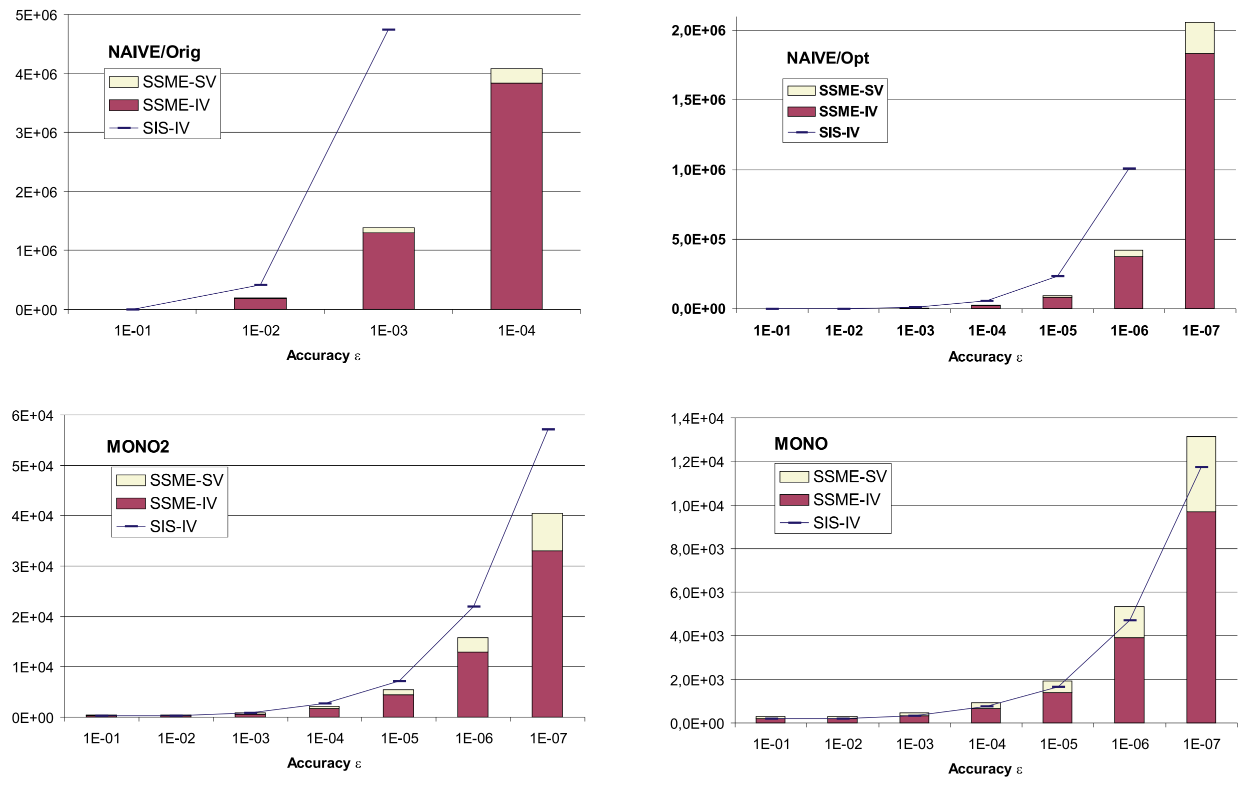


Figure 10: MACA-BI model with interval parameters and interval splitting: computational complexity vs. desired accuracy.

reduced to the number of interval parameters without monotonicity.

In Fig. 10, the comparison of the two selective splitting techniques SIS and SSME is summarized: the number of necessary interval and SV evaluations is depicted for varying values of ϵ (desired accuracy). The four diagrams show results for the variants 'NAIVE/Orig', 'NAIVE/Opt', 'MONO2', and 'MONO'. It can be seen that with the exception of 'MONO' (here, only one parameter is split), the additional cost due to SV evaluations in the SSME algorithm is always more than compensated by the decreased number of interval evaluations in that approach.

Fig. 11 illustrates that the computational complexity for interval splitting may strongly depend on the values of the parameters. This figure depicts the number of expression evaluations necessary during interval splitting for varying midpoints of the λ-interval. For orientation purposes, the SV throughput results are also depicted (labeled on the right y-axes). It is interesting to see that in all cases the splitting effort is highest where S is non-monotonic (i.e., where S takes its maximum). This means, that interval splitting is more costly when it is actually required, i.e., when the upper throughput bound is not obtained by using interval endpoints of the input parameters.

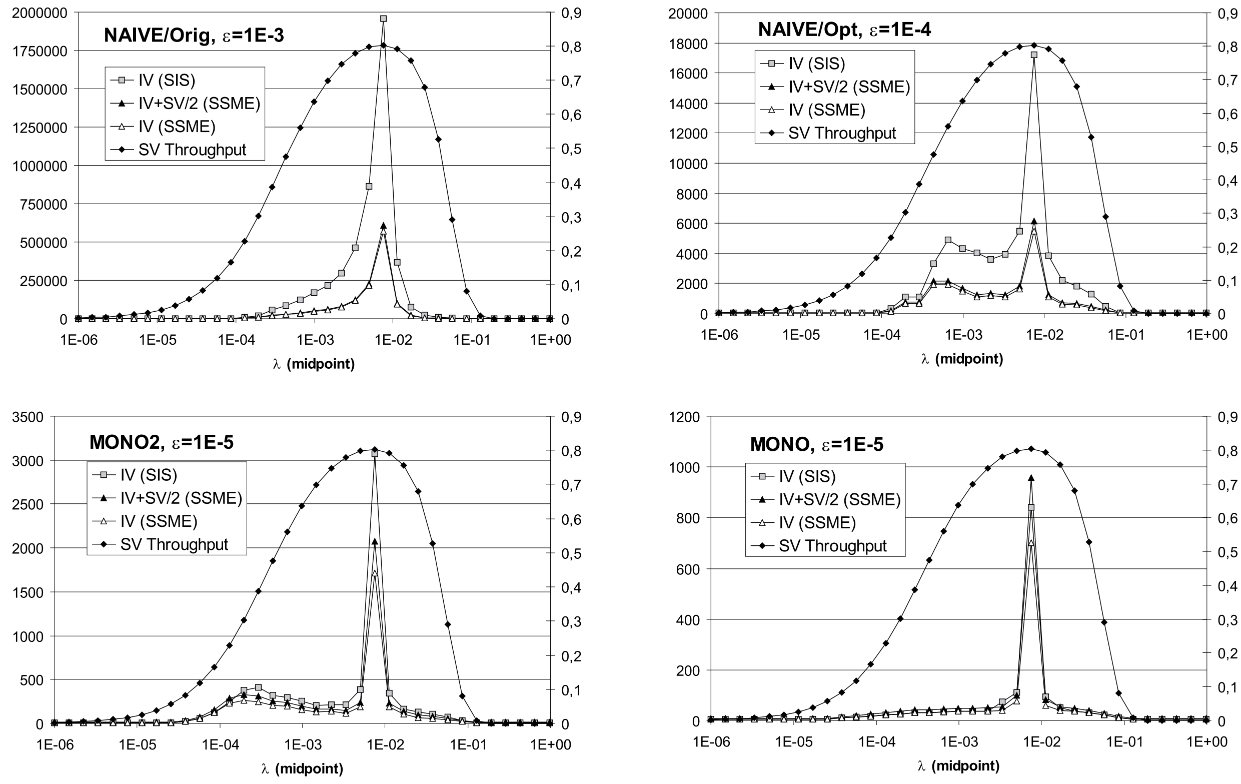


Figure 11: MACA-BI model with interval parameters and interval splitting: computational complexity vs. λ.

5 Model of an EJB Server Implementation

As a more complex example we use a model of an EJB (*Enterprise JavaBeans*) server implementation, which in this case works as the central scheduler of a distributed, three-tier, client-server architecture. The real application modelled is the Kensington Enterprise Data Mining system [7, 2] whose application server (or scheduler) implements the EJB-1.1 specification [23].

Specifically, the behavior of a method execution is modelled since it is the most common operation in the system. Detailed description of this execution and derivation of the model can be found in [9].

5.1 EJB Submodel to be Adapted to Interval Parameters

The model that is used to illustrate the application of interval splitting techniques corresponds to a sub-model of the system described above which is also derived in [9]. This sub-model is shown in Fig. 12 and it was obtained via the application of the Flow Equivalent Server method (FES) (see [5], for example) in order to reduce the complexity of the original

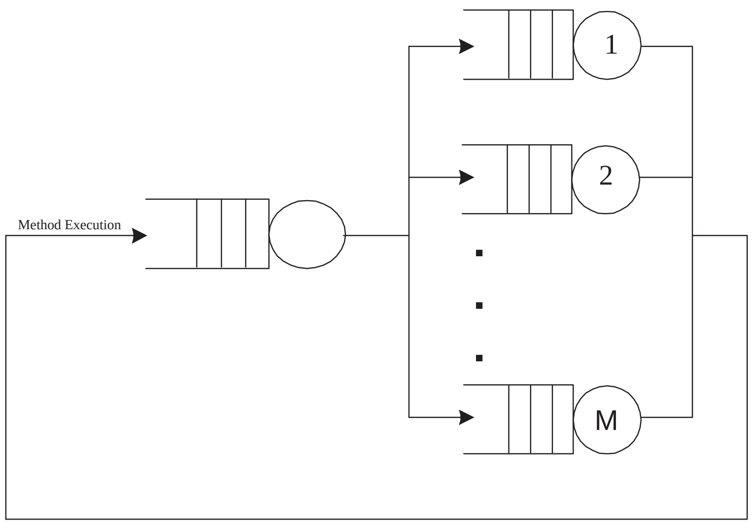


Figure 12: Short-circuited FES sub-network

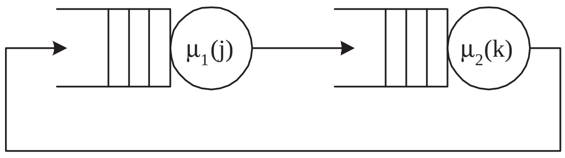


Figure 13: Container (sub)Model

model.

Blocking is the critical non-standard characteristic in this network. A client that has completed service in the outer node may be blocked under certain circumstances (see [9] for details). In this case blocking time is the time required for the first of the M parallel servers to clear its queue in a blocking-after-service discipline.

The M parallel servers are aggregated into a single node as it is shown in Fig. 13. At constant population N (j clients at the outer server and $k = N - j$ at the M parallel servers) the service rate functions $\mu_1(j)$ and $\mu_2(k)$, are estimated as follows:

$$\mu_1(j) = \begin{cases} 1/(m_1 + \beta_{N-j} b(N-j)), & \text{if } (N-j) \geq M, \\ 1/m_1, & \text{otherwise,} \end{cases} \tag{4}$$

where m_1 is the mean service time for server 1 (the outer server) when there is no blocking and β_{N-j} is the dynamic blocking probability. $b(N-j)$ is the mean blocking time when there are j customers at the outer server ($N-j$ customers at the parallel servers) and it is estimated by $b(k) = k/(M^2\mu)$ (see [4]), where μ is the service rate of each of the parallel servers. The computation of the probabilities β_{N-j} is beyond the scope of this paper; the details can be found in [9]. The service rate of the aggregated second server is:

$$\mu_2(k) = \sum_{n=1}^{M} \xi_{kn} n \mu, \tag{5}$$

where the parameter ξ_{kn} is the probability that n out of the M servers are busy, given that there are k customers at the parallel servers altogether. The probabilities ξ_{kn} are computed considering an M-state Markov chain for each population size $k \geq M$ at the parallel servers, where each state corresponds to the number of busy queues in the system. The following recursive function for the corresponding equilibrium probabilities $\pi_k(n)$ is derived from the balance equations determined by the M-state Markov chain:

$$\pi_k(n) = \begin{cases} 1, & \text{if } n = 1, \\ \frac{(I-n+1)(k-1)}{n(n-1)m_1\mu I}\pi_k(n-1), & \text{if } 2 \leq n < M, \\ \frac{(I-n+1)(k-1)(m_1 M^2\mu+(1-\alpha)k)}{M^2 m_1^2 \mu^2 I n(n-1)}\pi_k(n-1), & \text{if } n = M. \end{cases} \tag{6}$$

The parameter I is related to the blocking behavior, a detailed explanation is beyond the scope of this paper. Normalizing the $\pi_k(n)$ gives the probabilities

$$\xi_{kn} = \frac{\pi_k(n)}{\sum_{l=1}^{M} \pi_k(l)} \ . \tag{7}$$

Clearly the visitation rate is the same for both servers (see Fig. 13). The steady state queue length probability distribution for this network – $p(j)$ for the state with j tasks at server 1 and $N - j$ at server 2 – is then calculated as a product form in standard fashion. Finally, the throughput of the FES submodel given N threads in the submodel can be computed:

$$T(N) = \sum_{j=1}^{N} p(j)\mu_1(j) \ . \tag{8}$$

5.2 EJB Submodel with Interval Parameters

The computational steps for the solution of the submodel throughput, given that the service demand parameters μ and m_1 are intervals are adapted to interval arithmetic. During this adaptation several equations are re-formulated to decrease overestimation due to the dependency problem. Moreover, monotonicity properties of combined computation steps are exploited to further reduce the amount of overestimation. As an example for the adaptation process we discuss the interval adaptation of the probabilities ξ_{kn} (see Eqs. (6) and (7)). The complete adaptation process is described in more detail in [13].

Due to the recursion in Eq. (6), m_1 and μ occur multiple times in each of the expressions $\pi_k(n)$. Since with the exception of the case $n = M$, m_1 as well as μ appear only in the denominator, this does not cause overestimation of intervals for $\pi_k(n)$. However, in the normalization step, the dependency problem is in effect, because by having $\pi_k(n)$ in the numerator and the sum $\sum_{l=1}^{M}\pi_k(l)$ in the denominator, m_1 and μ have both increasing as well as decreasing influence on ξ_{kn}.

In the following we rewrite the expressions for $\pi_k(n)$ in a way that allows to cancel as many occurrences of m_1 and μ as possible in the normalization. In a first step, we extract the interval parameters m_1 and μ from the recursion of Eq. (6). This can be done by defining the following recursive expression $\tau_k(n)$ that does not depend on m_1 and μ:

$$\tau_k(n) = \begin{cases} 1, & \text{if } n = 1, \\ \frac{(I-n+1)(k-1)}{n(n-1)I}\tau_k(n-1), & \text{if } 2 \leq n \leq M. \end{cases} \tag{9}$$

Using these $\tau_k(n)$, the probabilities ξ_{kn} can be rewritten to reduce the effect of the dependency problem. For $n < M$, the reciprocals ξ_{kn}^{-1} can be computed as follows:

$$\xi_{kn}^{-1} = 1 + \frac{1}{\tau_k(n)}\left[\sum_{l=1}^{n-1}\tau_k(l)(m_1\mu)^{n-l} + \sum_{l=n+1}^{M}\frac{\tau_k(l)}{(m_1\mu)^{l-n}} + \frac{\tau_k(M)(1-\alpha)k}{M^2(m_1\mu)^{M-n+1}}\right].$$

Note that in this expression for ξ_{kn}^{-1}, $1 \leq n < M$, the computation is separated in a part where m_1 and μ contribute with an increasing effect and a part where m_1 and μ contribute with a decreasing effect, respectively. Within these parts, the parameters m_1 and μ are cancelled as often as possible. This significantly reduces the effect of the dependency problem as compared to the original expressions in Eqs. (6) and (7). The case $n = M$ is treated separately:

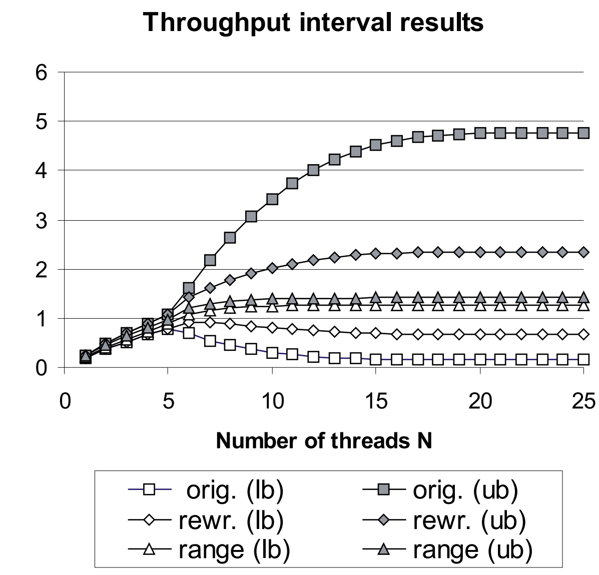
$$\xi_{kM}^{-1} = 1 + \frac{M^2}{\tau_k(M)\left[M^2 + (1-\alpha)k/(m_1\mu)\right]}\sum_{l=1}^{M-1}\tau_k(l)(m_1\mu)^{M-l}.$$

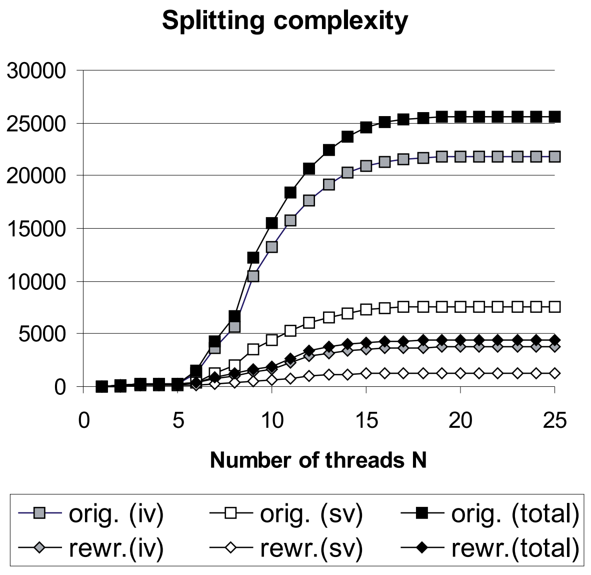
Since in this expression m_1 and μ contribute solely with an increasing effect to ξ_{kM}^{-1}, the probability ξ_{kM} is monotonically decreasing w.r.t. the parameters m_1 and μ. Thus, an interval $X_{kM} = [\underline{\xi}_{kM}, \overline{\xi}_{kM}]$ can be obtained by single value evaluation of ξ_{kM} using the endpoints of m_1 and μ's parameter intervals. I.e., $\underline{\xi}_{kM} = \xi_{kM}(\underline{m}_1, \underline{\mu})$ and $\overline{\xi}_{kM} = \xi_{kM}(\overline{m}_1, \overline{\mu})$.

5.3 Experimental Results with Interval-Based EJB Model

In the following comparison we use two different adaptations of the EJB model to interval arithmetic. In the version denoted as '*orig.*', the normalization step during the computation of the probabilities ξ_{kn} is done along the lines of the original equations given in [9]. For the computation of results labeled '*rewr.*', the rewritten expressions as discussed in Subsection 5.2 are used reducing the dependency problem in this specific computational step. Note that for SV parameters, the original and rewritten equations are mathematically equivalent, the re-formulation only makes a difference for interval evaluations.

In the experiments we use parameter values taken from evaluations described in [9]: $M = 6$ bean servers per container and $I = 20$ different bean instances. The estimates for the service rate of the bean servers μ and the mean service time of the outer server





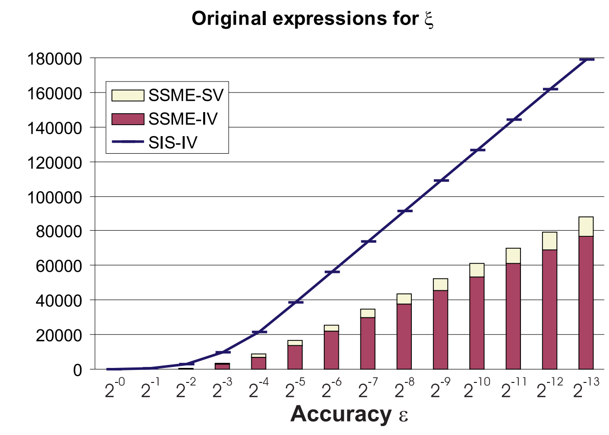
(a) (b)

Figure 14: Comparison of original and rewritten expressions: (a) throughput interval results and (b) computational complexity for interval splitting with varying number of threads (with desired accuracy $\epsilon = 10^{-2}$).

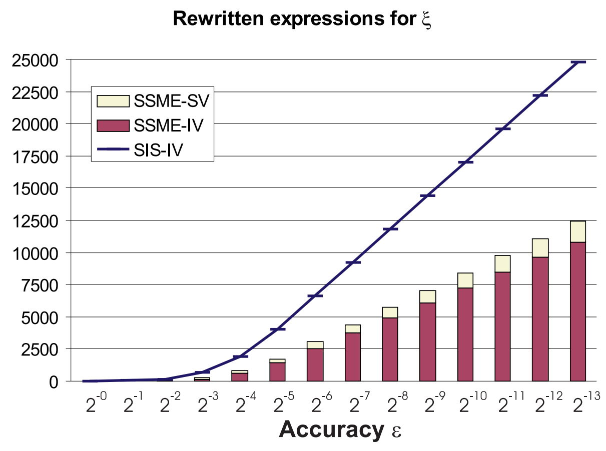
m_1 are subject to uncertainty. Thus, these parameters are characterized by the intervals $\mu^{(iv)} = 1/4.1 \pm 5\% = [0.2317, 0.2561]$ and $m_1^{(iv)} = 0.4 \pm 5\% = [0.38, 0.42]$.

Fig. 14 shows the effect of using the original respectively rewritten expressions for the intermediate probability intervals ξ_{kn} when computing intervals for the submodel throughput $T(N)$. Fig. 14(a) depicts throughput intervals for populations $N = 1, \ldots, 25$. It can be seen in this figure that using the original expressions for ξ_{kn}, the throughput interval is much more overestimated than the throughput interval obtained using the rewritten expressions for ξ_{kn}.

Unfortunately, due to the dependency problem occurring in the computation of $T(N)$, even the throughput intervals obtained by using the rewritten expressions are more than 10 times as wide as the actual range of the throughput (the innermost intervals in Fig. 14(a)). Thus, in both cases, interval splitting has to be applied to obtain reasonable tight enclosures of the throughput range. However, even though interval splitting may be necessary for both, original as well as optimized (w.r.t. interval computation) expressions, the computational effort is significantly reduced when using the rewritten formulae. Fig. 14(b) depicts the computational complexity required to obtain the range for the throughput with an accuracy of $\epsilon < 10^{-2}$. To obtain the range of the throughput, the SSME approach is used, which performs both, interval as well as conventional evaluations . For each version of the expressions (original and rewritten), three different plots are shown: the number of necessary interval evaluations (iv), the number of necessary single value evaluations (sv), and the weighted sum



(a)



(b)

Figure 15: Computational effort for interval splitting for the EJB model with interval parameters and varying accuracy $\epsilon = 2^0, \ldots, 2^{-13}$.

$iv + sv/2$ (total) — the computational complexity for an interval evaluation is approximately twice as high as for a single value evaluation. Note that using the rewritten ξ_{kn}-expressions decreases the number of evaluations during the interval splitting algorithm by a factor of more than 5.

In Fig. 15, the splitting complexity of the SIS algorithm is compared to that of the SSME algorithm. In this comparison, the number of threads is $N = 20$. Fig. 15(a) depicts complexity results using the original expressions for the probabilities ξ_{kn}, whereas Fig. 15(b) shows complexity results obtained using the rewritten expressions. Both diagrams show complexity values for logarithmically scaled accuracy values $\epsilon = 2^0, \ldots, 2^{-13}$ (i.e, deviation of obtained interval results from the actual range). Again, the computational effort is shown in terms of the number of necessary IV evaluations. In both cases, the SSME algorithm out-performs SIS by a factor of about 2. Note that with BFS (not depicted here), the computational complexity would have grown exponentially, whereas the filtering effect of both selective splitting algorithms causes an almost perfectly linear increase of the computational effort when the accuracy is increased.

By comparing Fig. 15(a) and Fig. 15(b) it can also be seen that using expressions that are rewritten in order to reduce overestimation, the computational effort of interval splitting can be significantly reduced. For example, to obtain an accuracy of $\epsilon = 2^{-13}$ with SSME, the equivalent of 87930 interval evaluations is necessary if the original expressions are used, whereas only 12415 interval evaluations are necessary when using the rewritten expressions. Hence this example illustrates that the adaptation of existing solution techniques to interval parameters has to be done with great care. For as many steps as possible, intermediate expressions have to be optimized for an efficient interval computation. I.e., wherever possible, monotonicity properties as well as possibilities to cancel occurrences of interval parameters should be exploited.

6 Conclusions

Recent studies have shown that using intervals as input parameters for models of computer and communication systems is appropriate to represent uncertainties in parameter values that are usually provided as single value numbers. The representation of uncertainties in performance models is of special importance in early phases of system design and implementation and for situations with restrictions to obtain data for input parameters via measurement. If intervals are used to characterize model parameters, any given parameter uncertainty is also reflected in the model output, i.e., in corresponding performance measures. This can be gained by adaptation of an existing conventional solution algorithm to interval parameters: every arithmetical operation of the original solution is replaced by a corresponding interval arithmetic operation.

However, the so-called dependency problem, well-known in interval mathematics, often causes significant overestimation on the obtained performance measure intervals. This means that the actual range of possible results, given the constraints defined by the set of interval parameters, may be a much smaller interval than the one obtained via interval arithmetic. This effect can be overcome if the original input parameter intervals are split into smaller intervals. For every combination of subintervals, the interval solution is computed and the overall minimum and maximum yield the bounds for a tighter performance measure interval.

In this paper we give an overview of existing interval splitting algorithms such as *brute force splitting* and *selective interval splitting.* We introduce a new splitting technique called *selective splitting with midpoint evaluation* (SSME) that combines conventional and interval model evaluations to reduce the overall computational complexity. Furthermore, we show how partial monotonicity properties can be exploited to give a more efficient interval solution. Two example models are included in this work to illustrate the application of the proposed interval splitting techniques. Along the lines of these examples a comparison of the various splitting algorithms is included and it shows that in most situations the new SSME approach performs better than plain selective splitting.

Acknowledgements

The authors would like to thank Shikharesh Majumdar, Carleton University, Ottawa, Canada, for his contributions and discussions related to this work that stem from an ongoing collaboration.

A Proof of Theorem 2

By definition, the range $f^*(X,Y,Z)$ is:

$$\begin{aligned} f^*(X,Y,Z) &= \Box\{f(x,y,z) \mid x \in X, y \in Y\; z \in Z\} \\ &= \left[\inf_{x\in X, y\in Y, z\in Z} f(x,y,z),\quad \sup_{x\in X, y\in Y, z\in Z} f(x,y,z)\right]. \end{aligned}$$

Thus, we have to prove that

$$\inf_{x\in X, y\in Y, z\in Z} f(x,y,z) = \inf_{x\in X} f(x,\underline{y},\overline{z}) \tag{10}$$

and that

$$\sup_{x\in X, y\in Y, z\in Z} f(x,y,z) = \sup_{x\in X} f(x,\overline{y},\underline{z}). \tag{11}$$

Clearly,

$$\inf_{x\in X, y\in Y, z\in Z} f(x,y,z) \le \inf_{x\in X} f(x,\underline{y},\overline{z}).$$

Thus, to proof Eq. (10), it is sufficient to show that:

$$\inf_{x\in X, y\in Y, z\in Z} f(x,y,z) \ge \inf_{x\in X} f(x,\underline{y},\overline{z}) \tag{12}$$

This inequality follows directly from the monotonicity properties of f: let $x = (x_1,\ldots,x_n) \in I\!R^n$, $y = (y_1,\ldots,y_m) \in I\!R^m$, and $z = (z_1,\ldots,z_l) \in I\!R^l$ be arbitrarily chosen. Since f is monotonic increasing w.r.t. y_i, $i = 1,\ldots,m$ it holds:

$$\begin{aligned} f(x,y,z) &= f(x_1,\ldots,x_n,y_1,\ldots,y_m,z_1,\ldots,z_l) \\ &\ge f(x_1,\ldots,x_n,\underline{y_1},y_2,\ldots,y_m,z_1,\ldots,z_l) \\ &\ge \cdots \ge f(x_1,\ldots,x_n,\underline{y_1},\ldots,\underline{y_m},z_1,\ldots,z_l). \end{aligned}$$

Because f is monotonic decreasing w.r.t. z_j, $j = 1,\ldots,l$ we continue:

$$\begin{aligned} f(x_1,\ldots,x_n,\underline{y_1},\ldots,\underline{y_m},z_1,\ldots,z_l) &\ge f(x_1,\ldots,x_n,\underline{y_1},\ldots,\underline{y_m},\overline{z_1},z_2,\ldots,z_l) \\ \ge \cdots &\ge f(x_1,\ldots,x_n,\underline{y_1},\ldots,\underline{y_m},\overline{z_1},\ldots,\overline{z_l}). \end{aligned}$$

I.e., for any $x \in I\!R^n$, $y \in I\!R^m$, $z \in I\!R^l$ we have:

$$f(x,y,z) \ge f(x,\underline{y},\overline{z}),$$

which proves Eq. (12) and hence Eq. (10). Eq. (11) is proved analogously. This completes the proof of Theorem 2.

B Proof of Corollary 3

From the properties of interval extensions (see Eq. (1)) it follows that for all $x \in X$:

$$f(x,\underline{y},\overline{z}) \in f(X,\underline{y},\overline{z}).$$

Thus,

$$\inf_{x\in X} f(x,\underline{y},\overline{z}) \in f(X,\underline{y},\overline{z}) = \left[\underline{f(X,\underline{y},\overline{z})},\ \overline{f(X,\underline{y},\overline{z})}\right].$$

Hence it follows that:

$$\underline{f(X,\underline{y},\overline{z})} \le \inf_{x\in X} f(x,\underline{y},\overline{z}).$$

Analogously,

$$\overline{f(X,\overline{y},\underline{z})} \ge \sup_{x\in X} f(x,\overline{y},\underline{z}),$$

which completes the proof of Corollary 3.

References

[1] G. Alefeld and J. Herzberger. *Introduction to Interval Computations.* Monographs and Textbooks in Computer Science and Applied Mathematics. Academic Press, Inc., New York, London, 1983.

[2] J. Chattratichat, J. Darlington, Y. Guo, S. Hedvall, M. Kohler, and J. Syed. An architecture for Distributed Enterprise Data Mining. In *Proc. 7th Int. Conf. on High Performance Computing and Networking Europe (HPCN Europe'99, April 12–14, Amsterdam, The Netherlands)*, April 1999.

[3] M. Gerla, M. Kazantzidis, G. Pei, F. Talucci, and K. Tang. Ad hoc, wireless, mobile networks: The role of performance modeling and evaluation. In G. Haring, C. Lindemann, and M. Reiser, eds., *Performance Evaluation: Origins and Directions, LNCS 1769*, pages 51–95. Springer Verlag, Berlin, 2000.

[4] P.G. Harrison and C.M. Lladó. Performance evaluation of a distributed enterprise data mining system. In B.R. Haverkort, H.C. Bohnenkomp, and C.U. Smith, eds., *Proc. 11th Int. Conf. on Computer Performance Evaluation Modelling Techniques and Tools (TOOLS 2000, March 27–31, Schaumburg, IL, USA), LNCS 1786*, pages 117–131. Springer Verlag, Berlin, March 2000.

[5] P.G. Harrison and N.M. Patel. *Performance Modelling of Communication Networks and Computer Architectures.* Addison-Wesley, 1993.

[6] B. Haverkort and A.M.H. Meeuwissen. Sensitivity & uncertainty analysis of markov-reward models. *IEEE Trans. on Reliability*, 44(1):147–154, March 1995.

[7] InforSense Ltd. Kensington 2000. http://Kensington.doc.ic.ac.uk.

[8] E.D. Lazowska, J. Zahorjan, G. Scott Graham, and K.C. Sevcik. *Quantitative System Performance – Computer System Analysis Using Queueing Network Models.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1984.

[9] C.M. Lladó and P.G. Harrison. Performance evaluation of an enterprise javabean server implementation. In *Proc. 2nd Int. Workshop on Software and Performance (WOSP 2000, September 17–20, Ottawa, Canada)*, pages 180–188, September 2000.

[10] J. Lüthi. Histogram-based characterization of workload parameters and its consequences on model analysis. In *Proc. MASCOTS'98 Workshop on Workload Characterization in High-Performance Computing Environments, July 19–24, 1998, Montreal, Canada,* pages 1/52 – 1/64, July 1998.

[11] J. Lüthi and G. Haring. Fuzzy queueing network models of computing systems. In *Proc. 13th United Kingdom Workshop on Performance Engineering of Computer and Telecommunication Systems (UKPEW, July 24, 1997, Ilkley, UK)*, July 1997.

[12] J. Lüthi and G. Haring. Mean value analysis for queueing network models with intervals as input parameters. *Performance Evaluation*, 32(3):185–215, April 1998.

[13] J. Lüthi and C.M. Lladó. Interval parameters for capturing uncertainies in an EJB performance model. submitted for publication, preprint available as Technical Report No. 2000-07, Fakultät für Informatik, Universität der Bundeswehr München, D-85577 Neubiberg, Germany, 2000.

[14] S. Majumdar. Interval arithmetic for performance analysis of distributed computing systems. In *Proc. Canadian Conf. on Electrical and Computer Engineering, Quebec, Canada, September 25–27*, September 1991.

[15] S. Majumdar, J. Lüthi, G. Haring, and R. Ramadoss. Characterization and analysis of computer systems with uncertainties and variabilities in workload. In R. Dumke, C. Rautenstrauch, A. Schmietendorf, and A. Scholz, eds., *Performance Engineering – State of the Art and Current Trends.* Springer-Verlag, Berlin, 2001. In print.

[16] S. Majumdar and R. Ramadoss. Interval-based performance analysis of computing systems. In P. Dowd and E. Gelenbe, eds., *Proc. Third Int. Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (Durham, NC, USA, Jan. 18-20, 1995)*, pages 345–351. IEEE Computer Society Press, January 1995.

[17] R.E. Moore. *Interval Analysis.* Prentice-Hall, 1966.

[18] A. Neumaier. *Interval methods for systems of equations*, volume 37 of *Encyclopedia of Mathematics and its Applications.* Cambridge University Press, Cambridge, 1990.

[19] H. Niederreiter. Quasi-monte carlo methods and pseudo-random numbers. *Bulletin of the American Mathematical Society*, 84(6):957–1041, November 1978.

[20] R. Ramadoss. Interval-based performance analysis of computing systems. Master's thesis, Institute for Electrical Engineering, Faculty of Engineering, Department of Systems and Computer Engineering, Carleton University, Ottawa, Ontario, Canada, October 1994.

[21] R.Y. Rubinstein. *Simulation and the Monte Carlo Method.* Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, New York et al., 1981.

[22] S. Skelboe. Computation of rational interval functions. *BIT*, 14:87–95, 1974.

[23] Sun Microsystems. Enterprise JavaBeans 1.1 Architecture. http://java.sun.com/products/ejb, 1999.

[24] C.M Woodside, S. Majumdar, and J.E. Neilson. Interval arithmetic for computing performance guarantees in client-server software. In F. Dehne, F. Fiala, and W.W. Koczkodaj, eds., *Proc. Int. Conf. on Advances in Computing and Information – ICCI '91, LNCS 497*, pages 535–546. Springer Verlag, Berlin, 1991.