A Metaheuristic Approach for Best Effort Timing Analysis targeting Complex Legacy Real-Time Systems

Johan Kraft, Yue Lu and Christer Norström
School of Innovation, Design and Engineering
Mälardalen University, Västerås, Sweden
{johan.kraft, yue.lu, christer.norstrom}@mdh.se

Anders Wall
ABB Corporate Research
Västerås, Sweden
anders.wall@se.abb.com

Abstract

Many companies developing real-time systems today have today no means for response time analysis, as their systems violate the assumptions of traditional analytical methods for response-time analysis and are too complex for exhaustive analysis using model checking. This paper presents a novel approach for best effort response time analysis targeting such systems, where probabilistic simulation is guided by a search algorithm of metaheuristic type, similar to genetic algorithms. The best effort approach means that the result is not guaranteed to be the worst-case response time, but also that the method scales to large industrial systems. The proposed method should be regarded as a form of testing, focusing on timing properties. An evaluation is presented which indicates that the proposed approach is significantly more efficient than traditional probabilistic simulation in finding extreme task response times. The paper also presents a method for finding good parameters for the search algorithm, in order to improve its efficiency.

1 Introduction

Many industrial real-time systems are very complex. One example is a control system for industrial robots, developed by ABB Inc. This system contains more than 3 million lines of code distributed on about 100 tasks, which communicate using message queues and semaphores. For many tasks in this system, the behavior and resource usage is highly dependent on events received from other tasks and on the values of certain state variables. Complexity is increased by the typical legacy issues of such large systems, which generally have life-cycles of decades during which many changes are made. Design documentation is often incomplete or inconsistent with the current implementation. Moreover, many industrial systems are open systems; they may receive stimuli from its environment in a variety of ways, e.g. from sensors, user interfaces or other connected computing systems. Systems in the telecom-domain often have similar characteristics.

The existing analytical methods for response time analysis, described in e.g. [5, 12, 16], use a too simplistic system model to allow for analysis of such complex systems. These analysis methods have several assumptions, which many complex embedded systems violate. For instance, tasks may trigger other tasks or change scheduling priority in response to application-specific events.

Since the software in complex embedded systems represents a very large investment in developing time, often hundreds or even thousands of person-years, changing systems of this size to allow for analytical response-time analysis is a huge effort, which can be hard to motivate economically.

Many companies developing complex embedded systems have therefore no means for response time analysis. They are forced to rely on testing to find timing-related problems, which is far from optimal as timing problems are often hard to find through testing. Enabling response-time analysis for complex embedded systems is therefore a problem of high industrial relevance.

Response time analysis for such systems require analysis methods that use a more detailed system model, which describes the tasks' behavior with respect to inter-process communication, usage of CPU time and usage of logical resources. Due to the complexity of the systems concerned, such models typically need to contain probabilistic abstractions, in order to avoid that the model becomes as complex as the real system. Work on extraction of simulation models from complex embedded legacy systems is in progress in a parallel track of our research [4, 14, 15].

The state space of such models are probably too large for exhaustive analysis, e.g., using model checkers like UPPAAL [8] or KRONOS [10]. A model checking expert may be able to optimize such a model to allow for exhaustive analysis, but the average engineer does not have that competence. Best-effort analysis through probabilistic discrete event simulation however scales to industrial-size systems, as it does not explore the whole state space of the model, only random "samples" (simulations) are evaluated. It is therefore not possible to give any guarantees regarding the properties of the modeled system, e.g. the worst-case response time of a task.

Best-effort response time analysis is none-the-less highly useful for developers of complex embedded systems if it is regarded as a form of testing, focusing on timing problems, and used as a complement to traditional testing. A simulation is executed in a fraction of the time required to run the corresponding test cases on the real system, due to the higher level of abstraction in a simulation model, which means that more scenarios can be tested in the same time. As an example, the simulation model used for the evaluation in this paper takes about 2 ms to execute, but corresponds to a 650 ms test case. Several frameworks exist for probabilistic simulation of real-time system models, for instance the commercial tool *VirtualTime*, from Rapita Systems Ltd. [2] and the academic tool *ARTISST* [13].

Traditional probabilistic simulation is however not suitable for finding extreme values of a task's response time. Due to the vast state space of the models concerned, random exploration is unlikely to encounter a task response-time close to the worst case. We therefore propose a novel approach for best-effort response time analysis targeting extreme values in timing properties. An evaluation is presented where we compare this approach to traditional probabilistic simulation, with respect to the discovered response times. The results indicate that the proposed approach is significantly more efficient in finding extreme response times for a particular task than traditional probabilistic simulation.

The proposed approach uses *metaheuristic* search algorithm, named *MABERA*, on top of traditional probabilistic simulation, in order to focus the simulations on parts of the state-space that are considered more interesting, according to a heuristic selection method. Metaheuristics are general high level strategies for iterative approximation of optimization problems. The search technique used by MABERA is related to two commonly used techniques, genetic algorithms and evolution strategies. We do not claim that MABERA is optimal, many improvements are possible, but we demonstrate the potential of extending probabilistic simulation with a metaheuristic search technique for the purpose of best-effort response-time analysis.

The outline of this paper is as follows. Section 2 presents the proposed algorithm in detail. Section 3 explains the parameters of the algorithm and Section 4 presents a method for how to select good values for these parameters. Section 5 presents our implementation of this approach and Section 6 presents the evaluation, including the characteristics of the simulation model used. Section 7 presents related work and Section 8 concludes the paper and presents ideas for future work, including possible improvements of the proposed approach.

2 The Algorithm

The metaheuristic algorithm proposed, MABERA, is defined as a function, according to Definition 1. This algorithm is presented in detail, using pseudo-code, in Section 2.2.

Definition 1. $r = MABERA(M, T, s, p, tt, l)$ *, where r is the highest observed response time of task T in the simulation model* M*. Parameter s is the* population size*,* p *is the* number of parents*,* tt *is the* termination threshold *and* l *is the* simulation length*.* □

The MABERA algorithm is an iterative process, where each iteration consists of a set of simulations, with length l, which produce a *generation* of simulation results. The population size (p) is number of individual simulations in a generation. The first generation is produced by running *s* independent probabilistic simulations, starting in the initial state of the simulation model.

Each generation is analyzed in order to identify *p* number of *parent* simulations, according to a specific selection heuristic described in Section 2.4. Each parent simulation is used to produce *s*/*p* child simulations, which are mutations of the parent simulation. The child simulations will explore a subset of the model's state-space, the *offspring state-space*, which is reachable from a selected state (time instant) in the parent simulation: the *restart time*.

The restart time is randomly selected in a specific time interval of the parent simulation, as described in Section 2.5. A child simulation follows the same path as the parent simulation until the restart time, where it randomly selects a different path in order to explore other parts of the offspring state-space. The offspring state-space is likely to contain a response time for *T* higher than the highest response time for *T* of the parent simulation, unless the parent simulation already have discovered the worst-case response time.

To explain the concept of offspring state-space, think of the state-space of a probabilistic simulation model as a tree, where each node corresponds to a state of the model. Nodes with multiple out-going edges represent non-deterministic selections in the model. An individual simulation is a specific path through the tree, which ends at the state decided by the simulation length. The offspring state-space, which the child simulations are focused on, is the subtree rooted in the state corresponding to the restart time in of the parent simulation.

The child simulations constitute a new generation, from which a new set of parents are selected, and so on. The algorithm iterates in this manner until a termination condition is reached, which depends on the *termination threshold*, tt. The value of tt decides how many "unsuccessful" generations that are allowed before termination, i.e., generations that failed to discover a response-time higher than the highest response time of the previous generations. The termination condition is presented in detail in Section 2.2.

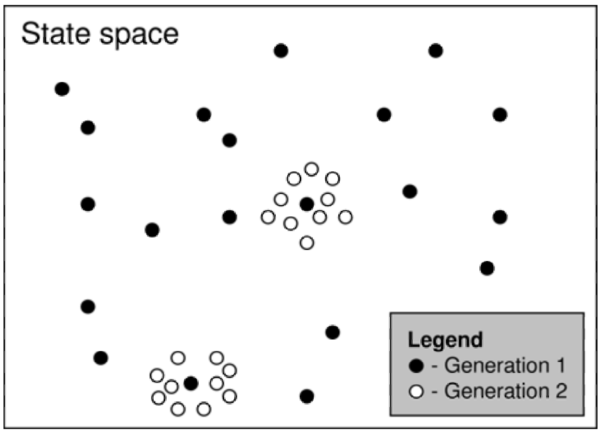

Figure 1. MABERA - conceptual

The state-space exploration of MABERA is illustrated by Figure 1, in this case using a 2-dimensional state-space. In practice, the state-space will have a large number of dimensions and more than two iterations will be made. In this example the population size (s) is 20 and the number of selected parents (p) is 2.

The heuristic selection of parents, described in Section 2.4, is very important for the efficiency of MABERA. There is always a risk of not finding the global maximum, i.e., the worst case response time, as the algorithm might "get stuck" at a local maximum, where no child simulation can be found that is more extreme than the parent. To reduce this risk, the MABERA algorithm selects several parents from each generation, at least 2. Thereby, if one parent "gets stuck" at a local maximum, there is still a chance that the other parents find better results.

The MABERA algorithm is not aware of the detailed state of a simulation. Instead, MABERA specifies the starting state of a child simulation indirect, using the *seed schedule* of the parent simulation and the restart time. The seed schedule specifies the seed-values used for generation of pseudo-random numbers and thereby the outcomes of all non-deterministic selections during the simulation. Thus, for a particular model, the state is completely specified by a seed schedule and the restart time. The simulation can be restarted from this state by running a new simulation using the specified seed schedule from the initial state (time = 0) to the restart time.

Since the connection between the seeds and the resulting simulation is (in practice) unknown in this approach, it is not feasible to optimize the result by selecting "good" seeds for the initial generation; we have no way of telling the "quality" of a seed schedule without running a simulation of it. The first generation of simulations is therefore generated using "randomly" selected seeds, obtained from CPU clock with micro second resolution.

2.1 Definitions

The pseudo-code of the MABERA algorithm, presented in Section 2.2, relies on the following definitions:

Definition 2. *A seed schedule is a list of pairs $\langle t, s\rangle$, where t and s are integer values. The t value specifies the point in simulation time when to apply seed s.* □

Definition 3. *A simulation result is a tuple $\langle rt, t_{rt}, et, t_{et}, pc, t_{pc}, S\rangle$. The elements rt, et, and pc are integer values corresponding to the highest response time, execution time and preemption count, respectively, of the task in focus. The elements t_{rt}, t_{et}, and t_{pc} are integer values specifying the start times of the task instances corresponding to rt, et and pc. S is the seed schedule used to produce this simulation result.* □

Definition 4. *The function $r = MAX\_RT(R)$ returns the highest response time in R, where R is a list of simulation results.* □

The MABERA pseudo-code also uses the functions SIM, SEL and GEN. Their interface is defined below, but their semantics require a more detailed presentation. The details of simulator corresponding to the SIM function is presented in Section 2.3. The functions SEL and GEN are presented using pseudo-code in sections 2.4 and 2.5.

Definition 5. $R = SIM(M, T, I, l)$ *, where R is a list of simulation results from independent simulations of the model M, with focus on task T. The simulations to execute is specified by parameter I, a list of seed schedules. The i:th simulation result in R corresponds to the i:th seed schedule in I. The simulation length is specified by parameter l.* □

Definition 6. $P = SEL(R, p)$ *, where P is a list of simulation results corresponding to the selected parents, a subset of the list of simulation results specified by R. The parents to select is specified by the parameter p, an integer value.* □

Definition 7. $I = GEN(P, s)$ *, where I is a list of seed schedules corresponding to a new generation, based on the simulation results (parents) given in P. The number of elements in I, i.e. the population size, is specified by s.* □

2.2 Pseudo-code for MABERA

Parameters:

M: the simulation model
T: the task in focus
s: integer value – the population size
p: integer value – number of parent to select
tt: integer value – the termination threshold
l: integer value – the simulation length

Returns:

The highest discovered response time of task T

Algorithm 2.1: MABERA(M, T, s, p, tt, l)

```
$tc \leftarrow tt$
$r \leftarrow 0$
$i \leftarrow 0$
$I \leftarrow ()$
while $i < s$
  do $I_i \leftarrow \langle 0, 0 \rangle$
     $i \leftarrow i + 1$
while $tc > 0$
  do $R \leftarrow \text{SIM}(M, T, I, l)$
     if $\text{MAX\_RT}(R) > r$
       then $r \leftarrow \text{MAX\_RT}(R)$
            $tc \leftarrow tt$
       else $tc \leftarrow tc - 1$
     $I \leftarrow \text{GEN}(\text{SEL}(R, p), s)$
return $(r)$
```

2.3 Function SIM

The function SIM represents a discrete event simulator that runs a set of simulations of a particular simulation model, as specified by a list of seed schedules. The result is a list of simulation results, where the i:th simulation result corresponds to the i:th seed schedule.

A global simulation clock, an integer variable, is shared by all tasks. The model may contain probabilistic selection of execution time, inter-arrival time or functional behavior. Such selections are the only source for non-determinism in the simulation of a model and are decided from pseudo-random numbers, which in turn are decided by the seed schedule. Since the simulation is deterministic given a specific seed schedule, it is possible to restart a simulation from a specific state, expressed as a seed schedule and a restart time.

To allow for probabilistic "random" simulation, the random number generator is re-initialized using a seed that is "randomly" selected, by calculating an integer value from a high-resolution hardware clock with microsecond resolution. In our implementation, this is performed when 0 is specified as seed, which is the case for the last seed change event for each simulation. The selected seed replaces the 0 in the seed schedule of the simulation result.

In the context of the MABERA algorithm, the output of the SIM function is a simulation result, as defined in Section 2.1. However, to allow for detailed inspection of a simulation, our simulator implementation also has the possibility to produce a detailed trace, targeting the Tracealyzer tool [1]. Since these traces are quite large, they are not generated during execution of the MABERA algorithm, but can be generated by executing the simulator in the trace-mode, using the specific seed schedule identified by MABERA.

2.4 Function SEL

The SEL function implements the heuristic selection of parent simulations used to produce the next generation of simulations. The selection ranks all simulation results in the current generation with respect to the three properties rt, et and pc, i.e., the highest response time, execution time and preemption count, respectively, of the task in focus.

The execution time and preemption count properties are included in the selection heuristics due to their potential for impacting response time, e.g., a task instance with very high execution time but relatively low response-time is also interesting since a different preemption pattern may result in a higher response time.

The three rank values of each simulation result are multiplied in order to obtain a composite fitness score for the simulation result. The best fitness score is 1, which corresponds to a simulation result that holds the record for all three properties. The returned set of simulation results contains a specified number of simulation results with best (lowest) fitness scores.

The method of combining the three rank values into a total fitness score is not claimed to be optimal. It gives equal importance to the three indicators, response time, execution time and preemption count. It might be possible to improve the selection heuristics by adjusting the relative importance of some of these three indicators. Moreover, the ranking hides the absolute differences in property values between candidates with adjacent ranking. Investigation of other selection heuristics is part of future work.

2.4.1 Definitions for function SEL

The pseudo-code defining the SEL function relies on the following definitions:

Definition 8. *The **ranking** of an element in a list of simulation results is the number of unique values for the specific property that are equal or larger to the specified element. The ranking of the simulation result with highest property value is 1. Simulation results with equal property values receive the same ranking.* □

Definition 9. $v = RANK\_RT(R, i)$ *, where v is the ranking of the i:th simulation result in R, with respect to the response time property of R.* □

Definition 10. $v = RANK\_ET(R, i)$ *, where v is the ranking of the i:th simulation result in R, with respect to the execution time property of R.* □

Definition 11. $v = RANK\_PC(R, i)$ *, where v is the ranking of the i:th simulation result in R, with respect to the preemption count property of R.* □

Definition 12. $P = LOW\_N(R, F, n)$ *, where P is a list of n simulation results, a subset of R. The simulation results included in P is the ones with lowest (i.e. best) corresponding value in the list F. The i:th element in F is the fitness score of the i:th simulation result in R.* □

2.4.2 Pseudo-code for function SEL

Parameters:
 R: a list of simulation results
 p: an integer value – the number of parents to select
Returns:
 A list of simulation results – the selected parents

Algorithm 2.2: SEL(R, p)

```
$F \leftarrow ()$
$i \leftarrow 0$
while $i < |R|$
    do $F_i \leftarrow$ RANK_RT$(R,i) *$ RANK_ET$(R,i) *$ RANK_PC$(R,i)$
       $i \leftarrow i + 1$
return (LOW_N$(R, F, p)$)
```

2.5 Function GEN

The GEN function generates a set of child simulations (seed schedules) corresponding to a new generation, through mutation of a set of parent simulations (simulation results).

Each resulting seed schedule is based on a single parent simulation, but have been extended with one additional seed change event, at the restart time of the parent. When this seed change event occurs, the simulation leaves the path of the parent simulation and follows a randomly selected path. This corresponds to the mutation. The time of this seed change event, the restart time, is in the normal case randomly selected in a time interval, where the lower bound is the restart time of the parent and the upper bound is the *SETI* time of the parent. The SETI time is the *Start time of the earliest Extreme Task Instance*, where *extreme* refers to the task instances that have the highest value of at least one of the following properties: response time, execution time and preemption count.

There is one special case: if the parent's SETI time is earlier than its restart time, this indicates that the parent simulation was less "promising" than the parent's parent. In this case, the restart time of the parent is reused in the child simulation in order to give the parent simulation a second chance.

2.5.1 Definitions for function GEN

The pseudo-code defining the GEN function relies on the following definitions:

Definition 13. $t = RST(R)$ *, where t is the restart time of R, a simulation result, i.e. the t value of the last $\langle s, t\rangle$ pair in the seed schedule of R.* □

Definition 14. $t = SETI(R)$ *, where t is the SETI of the simulation result R, i.e. the minimum of the property values t_{rt}, t_{et} and t_{pc} of R.* □

Definition 15. $S = SS(R)$ *, where S is the seed schedule of the simulation result R.* □

Definition 16. $r = RAND(a, b)$ *, where r is an integer value in the range $a \leq r < b$, randomly selected according to a uniform probability distribution.* □

2.5.2 Pseudo-code for function GEN

Parameters:
 P: a list of simulation results
 s: an integer value – the population size
Returns:
 A list of seed schedules – the next generation

Algorithm 2.3: GEN(P, s)

```
$N \leftarrow ()$
$i \leftarrow 0$
while $i < |P|$
    do $j \leftarrow 0$
       while $j < \lfloor s/|P| \rfloor$
           do if SETI$(P_i) <$ RST$(P_i)$
                do $t \leftarrow$ RST$(P_i)$
                else $t \leftarrow$ RAND(RST$(P_i)$, SETI$(P_i)$)
              $N_i \leftarrow$ APPEND(SS$(P_i)$, $\langle 0, t\rangle$)
              $j \leftarrow j + 1$
       $i \leftarrow i + 1$
return $(N)$
```

3 Parameters of MABERA

The proposed algorithm has a set of parameters, briefly mentioned in Section 2. The parameters of MABERA are:

- l: The **l**ength of each individual simulation.
- p: The number of selected **p**arents from each generation.
- tt: The **t**ermination **t**hreshold.
- s: The population **s**ize.

These parameters impact the thoroughness and runtime of the MABERA algorithm. To maximize the efficiency of MABERA, it is important to select good values for these parameters. This requires that we understand how these parameters impact the behavior of MABERA, and how they relate.

3.1 Parameter l

The simulation length, l, is the value of the simulation clock when the simulation should stop. The l parameter naturally impacts the runtime a simulation and should therefore not be longer than necessary, which depends on the scenario under analysis, e.g. a specific system test case.

Even though longer simulations may find higher response times, as they may contain multiple instances of the relevant scenario, e.g., a system mode change, the resulting increase in runtime can instead be used to increase the population size, s or the termination threshold, tt, which also impacts the runtime.

3.2 Parameter p

The p parameter, i.e., the number of parents to select, decides how much to trust the selection heuristics. If we could trust the heuristics to always point out the truly most "promising" simulation result, i.e., that is closest to the true worst case scenario, we would only need to select that single case for further analysis. However, since the heuristics is not a perfect oracle, several parents should be selected in order to reduce risk of bad heuristic decisions. During experiments with MABERA we have observed that the important property is not the absolute number of parents, but rather the relative amount of parents in relation to the population size, i.e., the p/s quota. This decides the number child simulations based on each parent. The parameter selection process in Section 4 is therefore focused on this quota rather than the absolute number of parents.

3.3 Parameter tt

The tt parameter, the termination threshold, impacts the number of iterations and thus the runtime of the analysis. The meaning of tt is, how many "unsuccessful" iterations that are allowed before the iterative process of MABERA should terminate, where "unsuccessful" means that no higher response times where found, compared to the previous iterations. The algorithm includes a *termination counter* (tc in the pseudo code) which initially is set to the value of tt. Unsuccessful iterations will reduce the value of the termination counter, while a successful iteration resets tc to the value of tt. When the termination counter reaches zero, the iterative process is terminated.

Thus, with a higher tt value, the risk that a good parent is rejected due to "bad luck" is reduced, but the runtime is increased by the extra iterations. It is important to find a balanced value for tt, as the extra runtime required for larger tt values can instead be used to increase the population size.

3.4 Parameter s

The population size, s, is the number of simulations to perform in each iteration. The larger population size, the more thorough analysis. Thus, s should preferably be as large as possible, but since it impacts the runtime of the analysis, which in practice is limited, it is necessary to find an upper bound for s that gives a runtime below (but close to) the desired runtime. When starting a large, over-night analysis, one would like to know that the analysis is finished by the morning, but preferably not much earlier than that in order to best utilize the available analysis time.

4 Selecting parameter values

In this section, we propose a three-step process for finding good parameters values for a specific simulation model. The process contains a set of experiments, which are presented together with examples based on the simulation model used for the evaluation of MABERA, presented in Section 6. One should note that the parameter values presented in this section are not necessarily optimal for other simulation models. Good values parameter values is believed to be dependent on the characteristics of the model under analysis, specifically, the amount and type of probabilistic selections in the model. We have however not yet investigated this aspect of MABERA.

This process is typically performed one time only, after the initial development of the simulation model. The resulting parameter values should be documented for reuse in future analyzes. Large changes of the simulation model may however require repeating the process.

Good values for the four parameters of MABERA are determined as presented in the following sections and in that order.

4.1 Selecting a value for l

The value for the l parameter should be decided first, as it does not depend on any of the other parameters. The l parameter is determined from the model, through direct studies or by studying detailed traces from simulations of the model. For our model, a suitable l value was found to be 650 ms. This length included the system's processing of the events relevant for the scenario under analysis, as well as some idle time afterwards, as a margin.

4.2 Selecting p/s quota and tt value

There is a dependency between the p/s quota and the tt value, as a higher tt value compensates (to some extent) for the negative effect of a higher p/s quota, i.e., the decreased number of child simulations based on each parent. Suitable values for these parameters can therefore not be selected individually, but need to be evaluated together, in combination. This section describes a four-step method for finding a good combination of p/s quota and tt value experimentally.

4.2.1 Step 1 - Specify candidates

The first step is to specify the set of candidate values for p/s and tt. Each combination of these parameter values will be compared in last step of this method. We can not give guidelines for the general case regarding good candidates for these parameters, as we have not yet investigated the impact of the model characteristics on these parameters. We based our candidate selection on our experiences of MABERA analysis of this particular simulation model. We limited the tt candidates to 2, 3 or 4. A value of 1 implies no tolerance, and values above 4 do not seem to improve the performance of MABERA. We have observed best results with p/s quotas below 0.05, so we selected 0.005, 0.01, 0.02 and 0.04 as candidates for the p/s quota.

4.2.2 Step 2 - Determine replication count

The second step is to decide the number of data points we need of each MABERA configuration, to ensure the reliability of the parameter evaluations. We do this experimentally by testing different candidate number of replications using a two-column table, specifying two different parameter combinations. Each cell contains a statistical measure of r independent runs of MABERA, where r is the candidate number of replications. Each row represents a comparison of two different p/s quotas (the columns), with respect to the results from MABERA. The reliability of the results can be studied by creating several such rows where each row is an independent replication of the p/s quota comparison, using the same parameters. If the differences between columns of the same row are significantly larger than the differences between rows of the same column, this indicates that the number of replications gives sufficient reliability.

The most obvious statistical measure to use for this comparison is the mean value. However, the mean value is influenced also by the very lowest values found, which tends to have a relatively large random noise according to our experiences. Since a typical use of the MABERA analysis would imply many replications and focus on the highest result found, we can safely ignore the lowest results in this comparison. We therefore suggest an alternative statistical measure, the mean value of the upper quartile of the MABERA results. We refer to this statistical measure as $Mean_{Q4}$.

The population size can be quite small to speed up this experiment and the tt value is not that important in this case, as it should not impact the reliability significantly, the important parameter here is the number of replications. It should however be constant for all columns.

This is however not a sufficient measure of reliability; we need to verify that the differences indicated by $Mean_{Q4}$ correspond to statistically significant differences between the underlying data sets. If the two data sets of a row are not significantly different, we need to increase the replication count in order to avoid inconclusive results later in the parameter selection process.

An appropriate test for this purpose is the two-sample, two-tailed Kolmogorov-Smirnov test [17] (hereafter the KS test). This test is non-parametric and distribution-free, i.e., it makes no assumptions on the underlying distribution of the data, which is necessary in this case as the response-time data is not normally distributed. The KS test should be applied on the fourth quartile of the MABERA results, in line with the motivation behind the $Mean_{Q4}$ measure.

Table 1. Test of reliability ($Mean_{Q4}$)

	p/s = 0.01	p/s = 0.04
Comparison 1	7886	7932
Comparison 2	7889	7940
Comparison 3	7901	7956

For our model, 200 replications gave very reliable results when we compared p/s quotas of 0.01 and 0.04, as presented in Table 1. Each cell in this table is the $Mean_{Q4}$ measure of the 200 MABERA results. The differences between the two columns (p/s quotas) are much larger than the differences between rows (replications), and when comparing the data sets of each row using the KS test, we found that the differences between the cells of each row are statistically significant at a confidence level of 99.9 %.

4.2.3 Step 3 - Comparable parameters

The third step of this method is to calculate the *cost index* for each combination of candidate values for p/s and tt, which is a relative measure of the average runtime (cost) of MABERA. The purpose of this cost index is to allow for a fair comparison of different parameter combinations, which may have considerably differences in their average runtime. If two parameter combinations produces similar results, but one is considerably faster, it is possible to increase the population size for the faster one and thereby obtain better results.

The first activity in this step is to run MABERA analysis of each parameter combination, replicated the number of times decided in step 2, and collect the average iteration count for each of the parameter combinations. This can be used as a measure of the cost (i.e., runtime), as these are directly proportional. A constant population size should be used in all cases. This should be in a realistic domain and should be a multiple of the number of parents implied by the candidate p/s quotas.

When this experiment was performed on our model, the difference in average iteration count was significant. As presented in Table 2, the most time-consuming combination of p/s and tt (p/s = 0.04, tt = 4) required 88 % more iterations (CPU time) than the least time-consuming combination (p/s = 0.005, tt = 2).

Table 2. Parameter impact on runtime

	tt = 2	tt = 3	tt = 4
p/s = 0.005	6.3	8.4	9.89
p/s = 0.010	6.64	9.16	10.18
p/s = 0.020	7.41	9.43	10.83
p/s = 0.040	7.27	10.06	11.87

The cost index of each candidate parameter combination is calculated by dividing the average iteration count of the specific case with the highest average iteration count of all cases, in our case 11.87. From the cost indices it is possible to calculate a comparable population size, s_c, for each candidate parameter combination. The comparable population size is calculated for each candidate parameter combination in order to give equal runtimes of MABERA, which allows for a fair comparison of the candidate parameter combination. The comparable population size of a parameter combination is calculated by dividing a *reference population size* with the cost index of the parameter combination. To maintain the relative number of parents it is necessary to calculate a comparable number of parents, p_c, by multiplying s_c with the desired p/s quota. Since p_c and s_c should be integer variables and thus needs to be rounded, the p_c/s_c quota will not be identical to the desired p/s quota. However, by selecting the reference population size carefully, it is possible to reduce these errors. In our case, a reference population size of 1000 was found to give quite small errors, below 5 %. For other, smaller, reference population sizes, we observed errors up to 17 %.

Table 3 presents our results from calculating cost index and corresponding p_c and s_c for each candidate combination of tt and (desired) p/s quota, based on the runtime data of Table 2. The reference population size was 1000.

Table 3. Comparable parameters

tt	p/s	Cost index	s_c	p_c	p_c/s_c
2	0.005	0.531	1883	9	0.00478
2	0.01	0.560	1787	18	0.0101
2	0.02	0.624	1602	32	0.0199
2	0.04	0.613	1632	65	0.0398
3	0.005	0.708	1413	7	0.00495
3	0.01	0.772	1295	13	0.0100
3	0.02	0.794	1259	25	0.0199
3	0.04	0.847	1180	47	0.0398
4	0.005	0.834	1200	6	0.005
4	0.01	0.858	1166	12	0.0103
4	0.02	0.912	1096	22	0.0201
4	0.04	1	1000	40	0.04

4.2.4 Step 4 - Comparison

The last step in of this method is to execute MABERA for each candidate parameter combination, using the comparable population size (s_c) and the comparable number of parents (p_c) and the number of replications decided in step 2, in our case 200. The simulation length (l) should be decided as described by Section 4.1. The best parameter combination is decided with respect to $Mean_{Q4}$, i.e., the mean value of the fourth quartile. If the difference between top candidates is small in comparison to the variance indicated by the earlier reliability test, the KS test should be used to verify the statistical significance of the difference. If no significant difference is found, one can either reduce the confidence level of the KS test or perform a more focused comparison of the top candidates, by repeating their evaluation using a higher number of replications.

The results from our experiment is presented in Table 4, which indicates that the best parameters for this model is p/s = 0.01 and tt = 3. The difference between this parameter combination and second best (p/s = 0.005 and tt = 2) was statistically significant according to the KS test, at a confidence level of 75 %.

Table 4. Parameter impact on $Mean_{Q4}$

	tt = 2	tt = 3	tt = 4
p/s = 0.005	8194	8155	8105
p/s = 0.01	8184	8231	8179
p/s = 0.02	8156	8172	8192
p/s = 0.04	8193	8162	8101

4.3 Selecting a value for s

Once suitable values for the other parameters have been established, the last parameter s, ultimately decides the runtime of MABERA. If a runtime of several hours is desired, a trial-and-error approach would be quite time consuming. A better way of determining an appropriate s value that corresponds to a desired (quite long) runtime is through extrapolation of a reference case with a relatively small population size. Given a desired runtime of t and a population size of the reference case, s_r, with the measured runtime t_r, the desired population size s can be extrapolated through $s = (t/t_r) * s_r$. Thus, to perform this extrapolation is simply a matter of selecting the population size for the reference case, measuring the corresponding runtime and deciding a desired runtime. The population size for the reference case can be arbitrarily selected, but should not be too small, as this might reduce the accuracy of the extrapolation. The reference case should have the p/s quota, tt and l values identified in previous steps. The p/s quota is especially important here. If changing s without adjusting p, the changed p/s quota will cause the selected parents to be more or less "promising", according to the selection heuristics. Moreover, each parent will be more or less extensively analyzed. It is likely that these factors impact the number of iterations. This is also supported by Table 2; the average iteration count has a positive correlation with the p/s quota.

5 Implementation

The MABERA implementation consists of two tools, the *SimOpti* tool and a framework for probabilistic discrete event simulation, hereafter referred to as the *simulator*. The SimOpti tool implements all parts of the MABERA algorithm apart from the SIM function of Section 2.3, SimOpti uses the simulator for that purpose. The simulator can also be used for traditional probabilistic simulation, either through the SimOpti tool or as a stand-alone console application.

SimOpti is a .NET application with a GUI that allows for specifying the parameters for MABERA. SimOpti allows for executing a set of MABERA analyses in one run, where each run can have different MABERA parameters. Moreover, it is possible to specify a number of replications of each MABERA analysis, in order to get several observations of each case. This facilitates the experiments proposed in Section 4.

The simulator is an API implemented in C, which corresponds to common operating system services, like task scheduling, inter-process communication and synchronization. The services included in the simulator API is based on the VxWorks real-time operating system [3]. The simulator does not supports analysis of distributed systems yet, only single-CPU systems.

A task is implemented as a C function, which may call other C functions, including standard library routines. An important API function is *execute* which advances the simulation clock. This models a task's consumption of CPU time. A task typically contains at least one execute-statement, but it is possible to emulate environmental stimuli using tasks without execute-statements. Such tasks will not impact the timing of the other tasks in the simulation. An executable file corresponding to the specified model is obtained by compiling and linking the model code together with the simulator API.

The task scheduling, preemptive FPS, is realized using fibers, i.e., lightweight threads scheduled by the application. The combination of native code and fibers makes the simulator very fast. A single simulation of the model used for our evaluation takes only about 2 milliseconds at a simulation length of 650 ms. Thus, the simulator is more than 300 times faster than "real-time" for this model.

The simulator records scheduling and communication events and can produce two types of output, either a parameter set according to Section 2.1 or a detailed trace of the simulation which can be inspected in the Tracealyzer tool [1].

6 Evaluation

In this section, we present an evaluation of MABERA, using a model of a fictive system. We have not been able to use a model of a real system due to the difficulties of reverse engineering such a model from an existing system. A semi-automated tool for model extraction is in development in a parallel research effort [4, 14, 15], but this tool is not yet ready for evaluation. This simulation model is however inspired by a real system we have studied, a control system for industrial robotics developed by ABB Inc., which was briefly described in the introduction.

6.1 The Simulation Model

The scheduling policy of the modeled fictive system is the same as for ABB's robot control system, preemptive

FPS. Just like the ABB system, the tasks of the model violate several assumptions of the existing methods for analytical response time analysis of FPS-based systems. The tasks in the model may:

- trigger the execution of other tasks through communication using message queues,
- be triggered both by timers and events, or a combination of both,
- have different temporal behaviors depending on the contents of received messages and the value of shared state variables,
- be blocked on semaphores, e.g. on sending and receiving of messages, and
- change the scheduling priority of tasks as a response to certain events.

The modeled fictive system controls a set of electric motors based on periodic sensor readings and aperiodic events. The calculations necessary for a real control system is not included in this model, the model mainly describes execution time, communication and other behavior that impact the temporal behavior. The model contains four periodic tasks:

Task	Priority	Period
PLAN	5	40
CTRL	4 or 2	10 or 20
IO	3	5
DRIVE	1	2

The priorities follow the model of the VxWorks real-time operating system [3], used by the robot control system, where lower priority value is more significant (better). The CTRL task may change priority and periodicity in response to two specific events in the model.

The PLAN task is responsible for high level planning of how to move the physical object connected to the motors. It periodically sends coordinates to the CTRL task, which in turn calculates control references for the motors with respect to input from the connected sensors. The resulting motor control references from CTRL are sent to the DRIVE task, which controls the motors. The IO task collects sensor data and aperiodic events from the system's environment and periodically sends this information to CTRL. Depending on the physical state of the controlled system, different numbers of messages are sent. This is modeled in a probabilistic manner. The number of messages from IO impacts the execution time, and thereby response time, of the CTRL task.

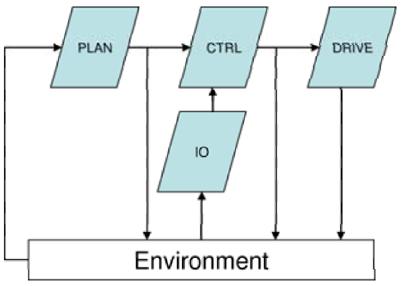

Figure 2. The model used for evaluation

The model also describes sporadic events generated by another connected system – a user interface, which is included in the model of the environment. These events impact the behavior of the model.

There are three types of events: START, STOP and GETSTATUS. These events are received by the PLAN task, which processes them accordingly. The START event will cause the system to start the motor and control it accordingly. The STOP event causes the system to stop the motor and go to idle state. The GETSTATUS event causes all tasks to send a status message to the user interface (modelled as "environment"). These events impact the execution time of the tasks. An overview of the model is given in Figure 2.

6.2 Results

The MABERA approach was applied to the described model using the SimOpti tool. The algorithm parameters were selected according to the process described in Section 4 4, i.e., p/s = 0.01, l = 650 and tt = 3. The desired runtime was set to 7 minutes per replication, which allowed for 200 replications of MABERA in 24 hours on a laptop computer from 2003 (Intel Pentium 4 CPU at 2.4 GHz). This runtime corresponded to a population size of 10000. This setup resulted in 16.240.000 individual simulations. The result of the 200 replications of MABERA is presented in Figure 3, as a histogram of relative frequency of response times, grouped in steps of 50 ms. The task in focus of the analysis is CTRL, which is the one with most complex behavior.

The highest response time discovered by MABERA was 8349, the $Mean_{Q4}$ measure (the mean of the fourth quartile) was 8325 and the mean value was 8045. The highest peak corresponds to values of 8324. Note that 47 % of the replications gave this result, so this result would most likely be detected in only 2 or 3 replications, which only takes about 20 minutes on the relatively slow computer we used.

The same number of simulations was made using traditional probabilistic simulation, without the simulation optimization achieved through MABERA. The corresponding result is presented in Figure 4. The results from probabilistic simulation are significantly lower. The highest response time discovered by probabilistic simulation was 7929, the $Mean_{Q4}$ measure was 7764 and the mean value was 7593. Moreover, the results from probabilistic simulation follows a bell-shaped curve, where most results are found close to the mean value and only 0.5 % of the results are above 7800, while 47 % of the MABERA results are close to the highest discovered value, 8349.

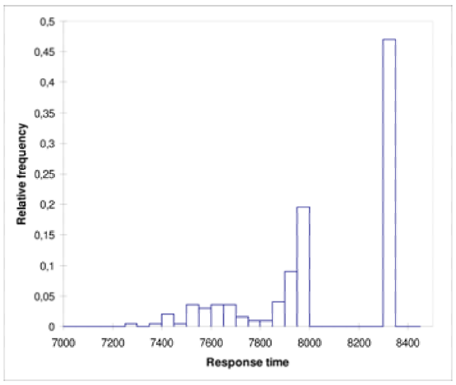

Figure 3. Results - MABERA

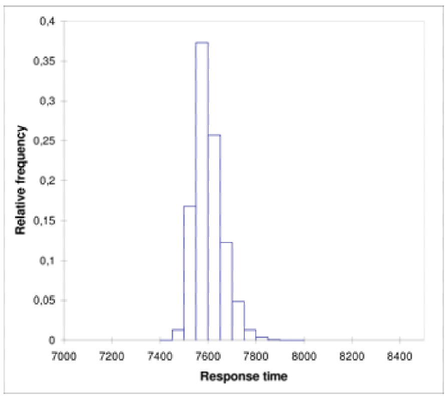

Figure 4. Results - probabilistic simulation

The highest discovered response time from the MABERA case is however not the worst case. Other runs of MABERA have (very rarely) identified response times around 8450. By changing the IO task to always send the maximum number of messages to the CTRL task, we have discovered response times for CTRL above 9000. Such response times are possible also without this manipulation of the model, but are extremely rare. However, for complex models extracted from real systems, such model manipulations may not be trivial and may threaten the validity of the model. We can however conclude that MABERA is significantly more efficient in finding extreme response times, compared to traditional probabilistic simulation.

7 Related work

The proposed metaheuristic algorithm, MABERA, implements a form of simulation optimization, i.e., a technique for finding parameters for a simulation that optimizes the result of a simulation [18]. In this case, the parameters we wish to find is the seed schedule of the simulation, which decides the sequence of pseudo-random numbers, which in turn determines the non-deterministic selections during the simulation. The simulation result that is in focus for optimization is the highest response time of a particular task during the simulation.

Metaheuristics are basically high level strategies for iterative solutions to optimization problems. Several definitions of metaheuristics are quoted in [4]. One definition is "A metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions." [19].

The MABERA algorithm is closely related to Evolutionary Computation (EC), a class of metaheuristics including for instance Genetic Algorithms [11] and Evolution Strategies [6]. Evolutionary computation is inspired by the principle of survival of the fittest in natural evolution. EC algorithms are iterative processes where each iteration produces a *generation* of possible solutions, *individuals*. EC algorithms typically use *crossover* (or *recombination*), which combines properties of two individuals in order to produce new individuals, and *mutation* operators, which cause a self-adaptation of individuals. The driving force is the *selection* of individuals based on their *fitness*, determined by a fitness-function specific for each problem. Individuals with a higher fitness have a higher probability to be chosen as members of the population of the next iteration or as parents for the generation of new individuals [9].

The MABERA approach is similar to the EC approaches, but does not use crossover. This is a possible improvement of MABERA, which is discussed in Section 8. MABERA should be regarded as an instance of an EC approach specialized for a specific problem, as it specifies the selection heuristics and method for mutation, i.e., how to generate child simulations from parent selections.

Evolutionary computation has been proposed for test case generation, for instance in [7], which proposes the use of genetic algorithms for test case generation in mutation-based testing. The MABERA approach can be considered a form of test case generation, where the "test case" is parameters for the simulator. The optimized property of MABERA is however not test coverage, as in [7] but rather the maximum response time discovered by the simulator.

8 Conclusions and future work

This paper has presented a best-effort approach for response time analysis, MABERA, where a metaheuristic algorithm is used as a simulation optimization, on top of traditional probabilistic simulation. MABERA can not identify the worst case response time of a task, but can however identify rare, very high response times of tasks, which are hard to find using testing or traditional probabilistic simulation. MABERA scales to industrial-size systems and should be regarded as a complement to traditional testing. We have demonstrated the potential of this approach through an evaluation based on a model of a fictive but realistic system.

The proposed algorithm is not optimal, but we have several ideas for improvement in future work. One possible improvement is to investigate other heuristic methods for selecting parents. Another improvement is to use crossover, i.e., to base child simulations on more than one parent simulation. This is however not trivial since the MABERA algorithm work on seed schedules and is not aware of the detailed simulation states. Crossover requires an approach where the metaheuristic algorithm is aware of the detailed state of the simulator, i.e., a white-box view of the simulator instead of the black-box view used by MABERA. Moreover, crossover requires that the simulation state can be described as a set of independent variables, chromosomes, where any combination is valid. This is still an open issue.

By using a white-box approach it would also be possible to avoid repeating previously explored simulations. In MABERA, two simulations with different parameters may result in identical simulations, due to the unknown relationship between seed schedules and the resulting scenarios. The white-box approach would be closer to model checking than traditional probabilistic discrete event simulation, even though it is still a best-effort approach.

References

[1] Tracealyzer website, http://www.tracealyzer.se.

[2] Rapita systems website, http://www.rapitasystems.com.

[3] Wind River website, http://www.windriver.com.

[4] J. Andersson, J. Huselius, C. Norström, and A. Wall. Extracting simulation models from complex embedded real-time systems. In *Proceedings of the 2006 International Conference on Software Engineering Advances, ICSEA'06*, Tahiti, French Polynesia, October 2006. IEEE.

[5] N. C. Audsley, A. Burns, R. I. Davis, K. W. Tindell, , and A. J. Wellings. Fixed priority pre-emptive scheduling: An historical perspective. *Real-Time Systems Journal*, 8(2/3):173–198, 1995.

[6] T. Back, F. Hoffmeister, and H. Schwefel. A survey of evolution strategies, 1991.

[7] B. Baudry, F. Fleurey, J.-M. Jézéquel, and Y. L. Traon. Genes and bacteria for automatic test cases optimization in the .net environment. In *Proceedings of the 13th International Symposium on Software Reliability Engineering (ISSRE'02)*, page 195. IEEE Computer Society, 2002.

[8] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Uppaal - a tool suite for automatic verification of real-time systems. In *Proceedings of the 4th DIMACS Workshop on Verification and Control of Hybrid Systems*, 1995.

[9] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, 2003.

[10] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: A Model-Checking Tool for Real-Time Systems. In A. J. Hu and M. Y. Vardi, editors, *Proceedings of the 10th International Conference on Computer Aided Verification, Vancouver, Canada*, volume 1427, pages 546–550. Springer-Verlag, 1998.

[11] H. Bremermann. The evolution of intelligence. the nervous system as a model of its environment. Technical Report 1, Dept. of Mathematics, University of Washington, Seattle, WA, 1958.

[12] G. C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications.* ISBN: 0-7923-9994-3. Kluwer Academic Publisher, 1997.

[13] D. Decotigny and I. Puaut. Artisst: An extensible and modular simulation tool for real-time systems. Technical Report 1423, IRISA, 2001.

[14] J. Huselius and J. Andersson. Model synthesis for real-time systems. In *Proceedings of the 9:th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 52–60, Manchester, UK, March 2005.

[15] J. Huselius, J. Andersson, H. Hansson, and S. Punnekkat. Automatic generation and validation of models of legacy software. In *Proceedings of the 12:th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 342–349, Sydney, Australia, August 2006.

[16] M. H. Klein, T. Ralya, B. Pollak, R. Obenza, and M. G. Harbour. *A Practitioners Handbook for Real-Time Analysis.* ISBN: 0-7923-9361-9. Kluwer Academic Publishers, 1999.

[17] A. M. Law and W. D. Kelton. *Simulation, Modeling and Analysis.* ISBN: 0-07-116537-1. McGraw-Hill, 1993.

[18] S. Ólafsson and J. Kim. Simulation optimization: simulation optimization. In *WSC '02: Proceedings of the 34th conference on Winter simulation*, pages 79–84. Winter Simulation Conference, 2002.

[19] I. Osman and G. Laporte. Metaheuristics: A bibliograph. *Annals of Operations Research*, 63(5):511–623, 1996.