Proceedings of 12th IOE Graduate Conference
Peer Reviewed
ISSN: 2350-8914 (Online), 2350-8906 (Print)
Year: 2022 Month: October Volume: 12

Task System Generation for Hard-real Time Scheduling on Unrelated Multiprocessor Platforms

Ashish Paudel [a], Nanda B. Adhikari [b]

[a, b] *Department of Electronics and Computer Engineering, Pulchowk Campus, Tribhuvan University*
[a] paudelashish1996@gmail.com, [b] adhikari@ioe.edu.np

Abstract
Analysis of multiprocessor scheduling methods for real-time systems is a challenging domain of research due to the need of strong theoretical and practical guarantees of correctness for the methods to be usable. For the analytical work, a population of tasksystems have to be generated that accurately represents the parameters of practical systems. Although efficient and reliable algorithms are available for identical multiprocessor platforms, such is not the case for unrelated platforms. This work proposes a novel algorithm for generation of hard real-time, implicit-deadline, sporadic and recurrent tasksystems with the context of scheduling them upon unrelated multiprocessor platforms. The algorithm has been designed by generalizing the system parameters of identical platforms to those of unrelated platforms and by simplifying the feasibility testing process, which has made the algorithm efficient. Various system parameters relevant for the understanding of the scheduling problem and the performance of the algorithm are then proposed. Finally, the algorithm is extensively analyzed in terms of the system parameters. The analysis has shown that the proposed algorithm produces tasksystem populations with feasibility ratio almost equal to 1 until a cutoff is reached. The population has also been shown to exhibit growing overall utilization upon increasing problem complexity. Also, the theoretical analysis has shown that the algorithm can be made to have linear time complexity in the average case for scale parameter $\sigma_u \geq 3.0$. These result verify that the algorithm is stable, reliable and efficient in generating task systems.

Keywords
task system generation, multiprocessor scheduling

1. Background

The need of this algorithm arose out of the investigation of the multiprocessor scheduling problem in the real-time domain. Scheduling of such problems is a tricky prospect due their theoretically difficult nature even for relatively simple task system models. Specifically, most of these problems are at least non-deterministic polynomial time hard (NP-hard) in the strong sense: i.e. no polynomial or pseudo-polynomial algorithms exist for solving these problems. A detailed investigation of the problems and their theoretical complexity classes has been performed in [1]. To further complicate the situation, hard real-time systems require strong theoretical guarantee of schedulability: i.e. any schedule proposed by any algorithm must not miss any deadline even if the system is run for countably infinite time steps. This requirement gives rise for the need of optimal algorithms because optimal algorithms guarentee finding a feasible schedule if it exists (which will also be optimal). Fortunately, optimal algorithms exist for the uni-processor case for certain task system models. A detailed investigation of the various uniprocessor scheduling algorithms as well as various aspects of task system models can be found in [2]. The optimality of the uniprocessor scheduling algorithms simplifies the multiprocessor scheduling problem (at least from a logical standpoint): if we can find a feasible partitioning of tasks upon the multiprocessor platform, then the overall multiprocessor schedule thus induced is guarenteed to be feasible.

Due to the tricky nature of the scheduling problem and the strong theoretical guarentees required by hard real-time systems, system designers often rely upon (often simpler) models that have been thoroughly analyzed and which provide the required guarentees instead of more complicated ones which may provide better representation of the system. In order to

analyze real-time system models, an efficient task system generation algorithm is required that can represent the population of task systems that may arise out of practical scenarios.

2. Related Work

Traditionally, task system generation algorithms have been parts of greater bodies of research that mostly focus on scheduling and schedulability analysis. The UUnifast algorithm [3] was introduced in order to generate uniprocessor task systems. The algorithm was fast and it could generate tasksystems with a fixed number of tasks and a fixed total utilization. The idea was to uniformly sample points from a $n-1$ dimensional hyperplane embedded in a n dimensional space (each such point is a task system with n tasks) where the per-task utilization is represented by the projection vectors towards the coordinate axes. This algorithm, however, could not be used for the multiprocessor case as it would entail generating points with total utilization ($u_{total} > 1$), which would cause some task systems to have per-task utilizations > 1 [4].

Other works [5] [6] [7] [8] have gone down a different route: generate a task system of cardinality $K+1$ for number of processors K and go on adding tasks until the total utilization exceeds the overall utilization exceeds K. In [4], it has been claimed that this approach generates tasksystems with biased utilization values.

An observation was made in [9] that the UUnifast algorithm can be used in the multiprocessor setting by simply discarding the task systems with per-task utilization > 1. This modification is known as the UUnifast-Discard algorithm. It has been noted in [4] that this algorithm becomes infeasible for a range of values of number of tasks and overall utilization. The problem of bias has been addressed in [10] as the *RandFixedSum* algorithm. It can generate task systems with given value of number of tasks and overall utilization.

Although the *RandFixedSum* algorithm is incredibly efficient in generating task systems, it is suitable only for identical multiprocessor platforms. Since unrelated multiprocessor platforms frequently occur in many practical systems [11] [12] [13] [14] analysis of the systems requires an effective tasksystem generation algorithm. This has been addressed in this work. The algorithm proposed in this work is a direct generalization of the incremental task system generation algorithm for a constrained-deadline system on identical multiprocessor platforms discussed in [15] without the schedulability test.

3. System Model

The system under consideration is a *hard real-time*, *implicit-deadline* system with *sporadic* and *recurrent* tasks and an *unrelated multiprocessor* platform upon which the tasks have to be scheduled.

A task system to be generated for the model is represented as $\Gamma = \{t_1, t_2, ..., t_n\}$. Such a task t_i is characterized by two parameters: the relative deadline D_i (which is equal to the lower bound on the inter-arrival time, T_i) and a r-tuple of worst-case execution times upon r processors: $C_i = (C_{i1}, C_{i2}, ..., C_{ir})$.

4. Task-set Generation for Identical Multiprocessor Platform

The task system generation algorithm discussed in [15] is outlined below:

1. Generate utilizations (U_i) from exponential distribution with scale σ_u, regenerating tasks that have utilizations > 1.
2. Generate task period from a uniform distribution in $[0, 2000]$ and Worst-case execution time as: $C_i = U_i \times T_i$.
3. Generate deadline from a uniform distribution in $[C_i, T_i]$.
4. In this manner, initially generate a task set of size $r+1$ for r processors.
5. Verify feasibility of the generated task system according to the necessary condition of feasibility proposed in [16]. If the task system is deemed not feasible, goto step 1.
6. Again, verify feasibility of the task system according to one of the sufficient tests of feasibility proposed in [15]. If the task system fails the test, goto step 1.
7. If the task set size (n) is as large as desired, stop. Otherwise add a new task to the task set and goto step 5.

As discussed earlier, this algorithm had been designed for constrained-deadline recurrent, sporadic task system on *identical* multiprocessor platforms. In order to generalize it for unrelated multiprocessor platforms, the first step was to generate r worst-case execution times in step 2 of the algorithm. The other difference between the systems is that the target system is implicit-deadline. Hence, step 3 can be omitted. Furthermore, it was seen that the feasibility test proposed in [16] is pseudo-polynomial in complexity [15] and has not yet been generalized to unrelated multiprocessor platforms. Hence, the test has been omitted in this work. Also, the sufficient conditions for schedulability have a dependency upon global-EDF scheduling policy [15], so they too have been omitted because they would restrict the use of the algorithm for other scheduling scenarios. As will be evident from the results, the algorithm generates high proportion of feasible task systems, so the removal of the schedulability tests is a justified trade-off. Finally, having removed the schedulability tests, the requirement of incremental generation of task systems ceases to exist. Direct generation of task system of desired size can be done. The proposed algorithm with the modifications as discussed above is discussed next.

5. Proposed Algorithm

Let $r \in Z^+$ deonte the number of processors in the multiprocessor platform, $n \in Z^+$ denote the number of tasks in the task system with $r < n$, D_{min}, $D_{max} \in Z^+$ denote the minimum and maximum deadlines that are generated by the algorithm with $D_{min} < D_{max}$, $U(\cdot,\cdot)$ denote the uniform distribution and $Exp(\cdot)$ denote the exponential distribution with scale parameter σ_u. Then, the algorithm proceeds as described in algorithm 1. Since there are novel aspects to the algorithm, both theoretical and practical aspects of the algorithm have been analyzed in the following discussion.

Theorem 1. *Algorithm 1 halts with probability 1.*

Proof. Let $q_{ij}^{(k)} = P(U_{ij} > 1)$ be the probability that the j^{th} entry in the WCET array is > 1 in the k^{th} run of the while loop for a fixed value of i. Also, let $p_{ij}^{(k)} = 1 - q_{ij}^{(k)}$. Then, we have:

$$q_{ij}^{(k)} = P(U_{ij} > 1) = \int_{x=1}^{\infty} \sigma_u e^{-\sigma_u x} dx = e^{-\sigma_u}$$

and $p_{ij}^{(k)} = 1 - q_{ij}^{(k)} = 1 - e^{-\sigma_u}$.

Algorithm 1 Generation of Task-sets for Unrelated Multiprocessor Platforms

procedure GENERATETASKSET($\sigma_u, n, m, T_{min}, T_{max}$)
  for $i \leftarrow 1$ to n **do**
    $U_i \leftarrow [2,2...]_r$
    $C_i \leftarrow [0,0...]_r$
    $D_i \sim U(D_{min}, D_{max})$
    $D_i \leftarrow \lfloor D_i \rfloor$
    while $\exists U_{ij} > 1$ **do**
      $U_{ij} \sim Exp(\sigma_u)$
      $C_{ij} \leftarrow \lfloor U_{ij} \times D_i \rfloor$
    end while
  end for
  $\Gamma \leftarrow \{(C_i, D_i) \mid i \in \{1,2,..n\}\}$
  return Γ
end procedure

Let $p_i^{(k)}$ be the probability that $\forall_j U_{ij} \leq 1$ for the given i in the k^{th} run of the while loop. Then,

$$p_i^{(k)} = \prod_{j=1}^{r} p_{ij}^{(k)} = \prod_{j=1}^{r} (1 - e^{-\sigma_u}) = (1 - e^{-\sigma_u})^r$$

Let X be a random variable that represents the number of while loops that have to be run when the condition $\forall_j U_{ij} \leq 1$ is encountered for the first time for a particular i. Then, X is distributed according to the geometric distribution. Hence,

$$\begin{aligned} P(X = k) &= \left(1 - p_i^{(k)}\right)^{k-1} p_i^{(k)} \\ &= \left(1 - \left(1 - e^{-\sigma_u}\right)^r\right)^{k-1} \left(1 - e^{-\sigma_u}\right)^r \end{aligned}$$

and

$$P(X \leq k) = 1 - \left(1 - p_i^{(k)}\right)^k = 1 - \left(1 - (1 - e^{-\sigma_u})^r\right)^k$$

Let $P(H)$ be the probability that the algorithm halts. We have,

$$P(H) = \lim_{k \to \infty} P(X \leq k) = \lim_{k \to \infty} 1 - \left(1 - (1 - e^{-\sigma_u})^r\right)^k$$

We have, $0 \leq (1 - e^{-\sigma_u}) \leq 1$ (since it is a probability)

or, $0 \leq (1 - e^{-\sigma_u})^r \leq 1$, $(r > 0)$

or, $0 \geq -(1 - e^{-\sigma_u})^r \geq -1$, $(r > 0)$

or, $0 \leq 1 - (1 - e^{-\sigma_u})^r \leq 1$, $(r > 0)$

or, $0 \leq (1 - (1 - e^{-\sigma_u})^r)^k \leq 1$, $(r, k > 0)$.

Thus, $\lim_{k \to \infty} (1 - (1 - e^{-\sigma_u})^r)^k = 0$ and hence:

$$P(H) = \lim_{k \to \infty} 1 - \left(1 - (1 - e^{-\sigma_u})^r\right)^k = 1$$

□

Theorem 2. *Algorithm 1 has an average runtime of* $O\left(\frac{nr}{(1-e^{-\sigma_u})^r}\right)$.

Proof. We have, the expected value of the geometric random variable X as $E[X] = \frac{1}{p_i^{(k)}} = \frac{1}{(1-e^{-\sigma_u})^r}$. This is the expected number of steps for which the while loop has to be run. Since each test of the while loop takes an average of $\frac{r}{2}$ steps (linear search) and the outer loop has to be run for n steps (1 for each task), the average runtime of algorithm 1 is $O\left(\frac{nr}{(1-e^{-\sigma_u})^r}\right)$. □

Corollary 2.1. *Algorithm 1 has an average runtime linear in r at the limit* $\sigma_u \to \infty$.

Proof. We have, $lim_{\sigma_u \to \infty} \frac{nr}{(1-e^{-\sigma_u})^r} = \frac{nr}{(1-0)^r} = nr$ □

It has to be noted that even at moderate values of σ_u (≥ 3.0), the function $\frac{r}{(1-e^{-\sigma_u})^r}$ is almost linear. So, a very fast average performance (linear in the problem size nr) can be expected from the algorithm.

6. Partitioned-EDF Schedulability Test

In order to analyse the algorithm's performance in practical scenarios, it is necessary to determine the effect of the various system parameters on the population of task systems generated. For this purpose, the partitioned earliest-deadline-first (EDF) [2] scheduling policy is used as a test for schedulability. The specifics are discussed next.

The partitioned EDF scheduling problem can be formulated as an integer linear program (ILP) [1]. Consider the set of decision variables x_{ij} with the interpretation that:

$$x_{ij} = \begin{cases} 1 \text{ if task } \tau_i \text{ is assigned to processor } p_j \\ 0 \text{ otherwise} \end{cases} \quad (1)$$

Then, the problem can be formulated as:

$$\text{minimize} \sum_{i=1}^{n} \sum_{j=1}^{m} x_{ij} \cdot \frac{C_{ij}}{D_i} \text{ subject to} \quad (2)$$

$$\sum_{j=1}^{m} x_{ij} \geq 1 \; \forall \; i \in \{1,2,...,n\} \text{ and} \quad (3)$$

$$\sum_{i=1}^{n} x_{ij} \cdot \frac{C_{ij}}{D_i} \leq 1 \; \forall \; j \in \{1,2,...,m\} \quad (4)$$

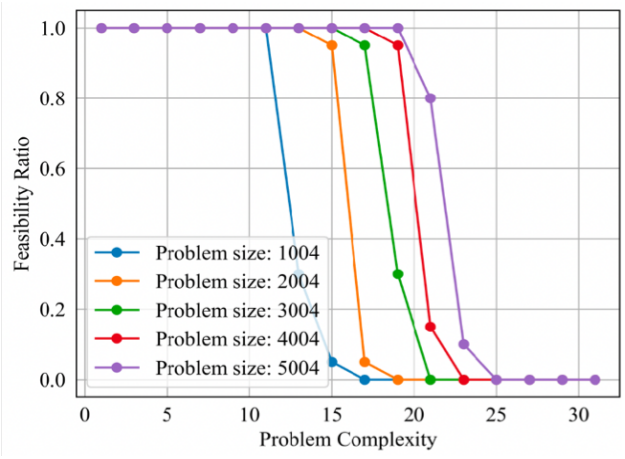

Figure 1: Problem complexity vs feasibility ratio at various values of problem size, $\sigma_u = 1.0$, $dr = 50$

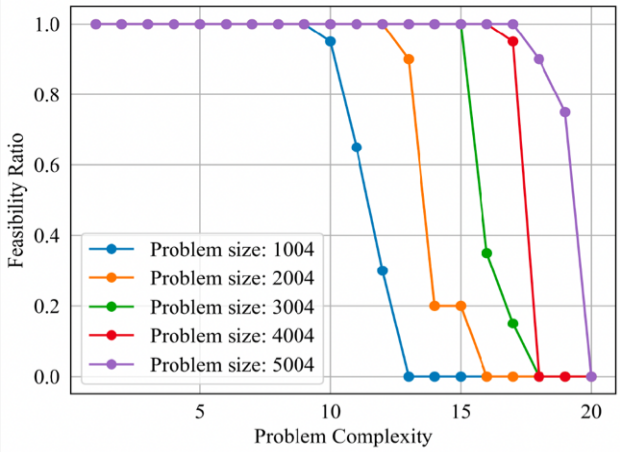

Figure 2: Problem complexity vs feasibility ratio at various values of problem size, $\sigma_u = 2.0$, $dr = 100$

In the above problem, the objective penalizes the overall utilization whereas the first set of constraints ensures that every task is assigned to at least one processor and the second set of constraints ensures that the per-processor utilization does not exceed 1.

Since the domain of the decision variables is $\{0,1\}$ the optimization problem is a special case of the Integer Linear Program and is well known to be NP-hard [17]. The NP-hard nature of the problem prevents exact evaluation of the optimal solutions. However, for feasibility testing, fast metaheuristic algorithms are available which can efficiently find feasible solutions (even though the solutions may not be optimal). For the purpose of this work, Google's branch-and-bound metaheuristic algorithm [18] [19] has been used for partitioning.

After having obtained the partitions, testing for

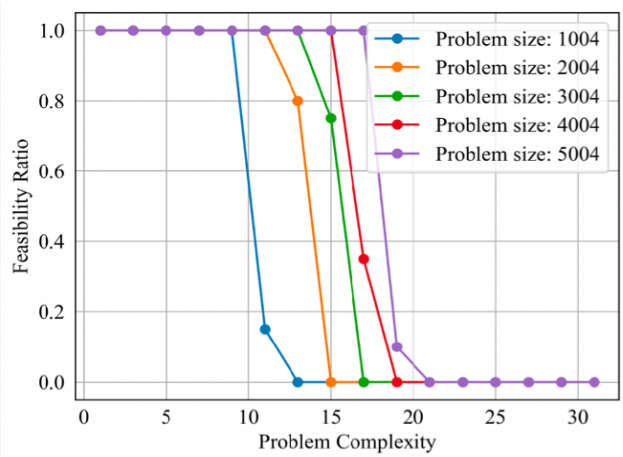

Figure 3: Problem complexity vs feasibility ratio at various values of problem size, $\sigma_u = 3.0,\ dr = 150$

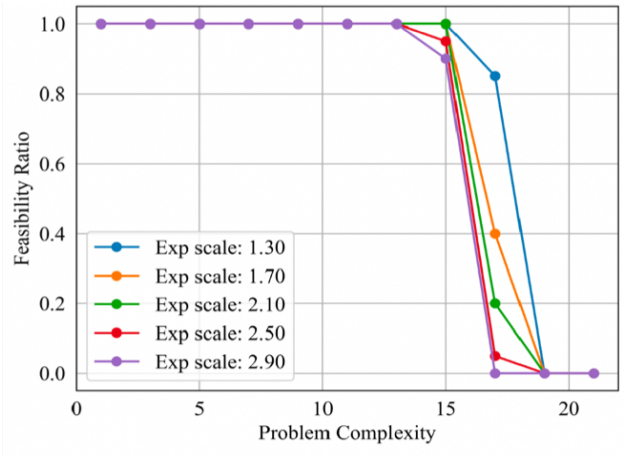

Figure 5: Problem complexity vs feasibility ratio at various values of $\sigma_u,\ s = 3000,\ dr = 150$

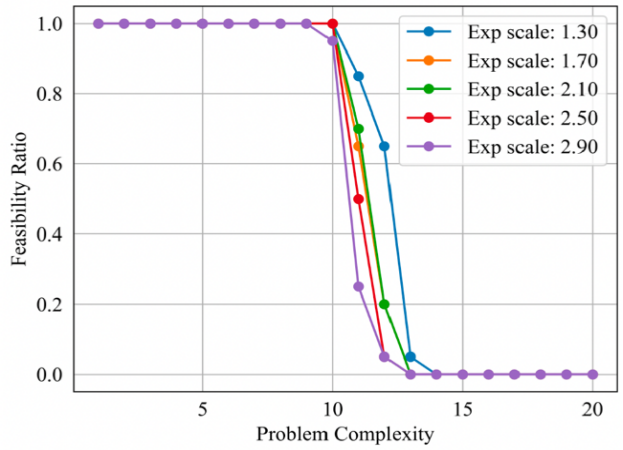

Figure 4: Problem complexity vs feasibility ratio at various values of $\sigma_u,\ s = 1000,\ dr = 100$

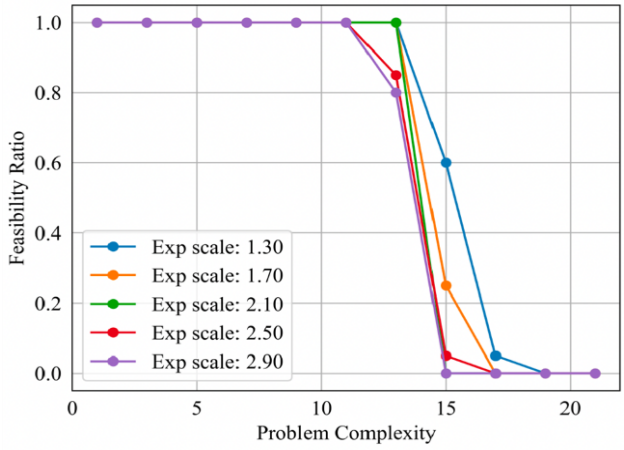

Figure 6: Problem complexity vs feasibility ratio at various values of $\sigma_u,\ s = 2000,\ dr = 50$

feasibility can be done in linear time (in problem size). Since EDF is an optimal uniprocessor scheduling algorithm [2], it guarentees schedulability given the following condition is satisfied for all partitions Γ_p:

$$\Gamma_p \subseteq \Gamma \text{ is EDF-schedulable iff } \sum_{\tau_i \in \Gamma_p} \frac{C_i}{D_i} \leq 1. \quad (5)$$

7. Quantitative Analysis

7.1 System Parameters used for analysis

1. Problem size (s):

   Problem size denotes the number of decision variables involved in the ILP for partitioned-EDF testing. It can be calculated as:

   $$s = n \times r \quad (6)$$

   where n denotes the number of tasks and r denotes the number of processors in the task system.

2. Feasibility ratio (fr):

   The feasbility ratio indicates the proportion of the sample of tasksystems generated that have been deemed feasible by the test. If in a sample of size t, k tasksystems are feasible, the feasibility ratio can be calculated as:

   $$fr = \frac{t}{k} \quad (7)$$

3. Problem Complexity (c):

   Problem complexity intuitively represents the relative hardness of the problem. From a bin-packing analogy, it can be understood that if a large number of items have to be packed into relatively small number of bins, the problem

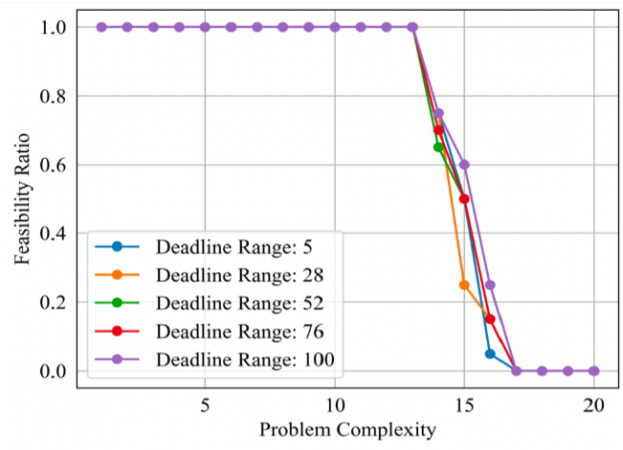

Figure 7: Problem complexity vs feasibility ratio at various values of deadline range, $\sigma_u = 1.3,\ s = 2000$

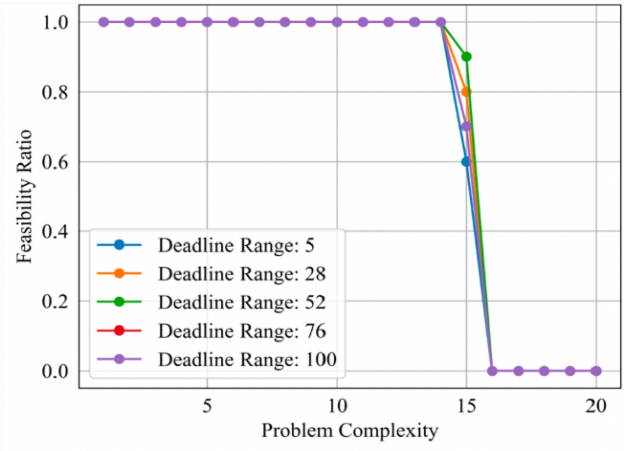

Figure 9: Problem complexity vs feasibility ratio at various values of deadline range, $\sigma_u = 3.0,\ s = 3000$

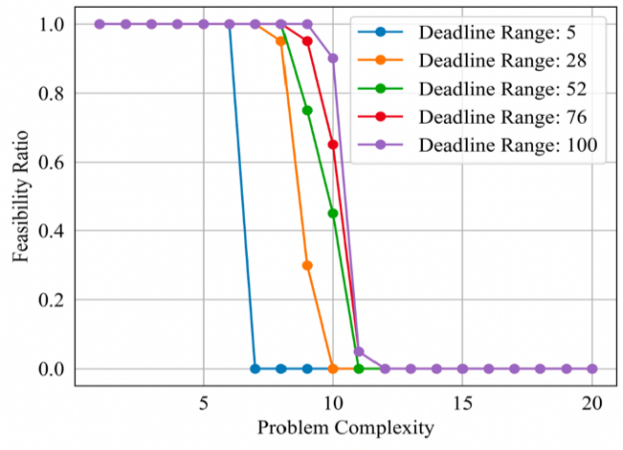

Figure 8: Problem complexity vs feasibility ratio at various values of deadline range, $\sigma_u = 2.0,\ s = 1000$

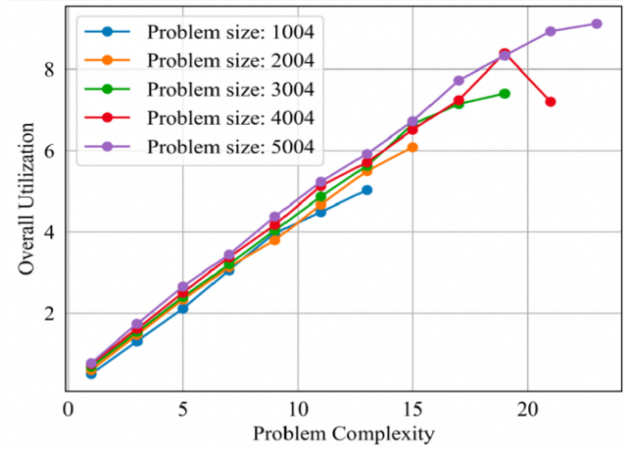

Figure 10: Problem complexity vs overall utilization at various values of problem size, $\sigma_u = 1.0,\ dr = 50$

becomes difficult. Problem complexity can be calculated as:

$$c = \frac{n}{r} \tag{8}$$

4. Deadline Range (dr):

   Deadline range indicates the difference between the maximum and minimum deadline values used by the algorithm for task system generation. It can be calculated as:

   $$dr = max\_deadline - min\_deadline \tag{9}$$

   Intuitively, larger the deadline range, more is the flexibility in selecting deadlines, so there is a higher probability of generating a population with higher feasibility ratio.

7.2 Problem Complexity vs Feasibility Ratio Analysis

Since problem complexity and feasibility ratio are two of the most important system parameters, they have been chosen as the primary variables for the subsequent analysis. The system has been analyzed by varying the parameters as discussed above and the results thus obtained have been presented in figures 1 to 9.

7.2.1 At various values of problem size

The results (figures 1 to 3) clearly show that the algorithm generates populations of tasksystems with very high feasibility ratio until a cut-off is encountered due to the entire problem space being infeasible. The algorithm has also shown to be stable for different values of the system parameters, thus can be reliably used for task system generation. Finally, it

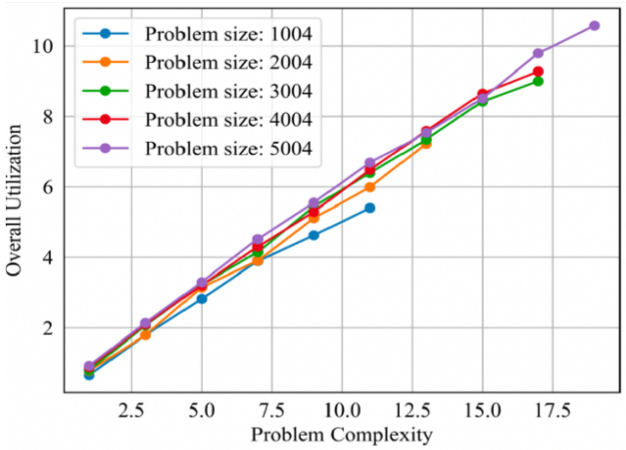

Figure 11: Problem complexity vs overall utilization at various values of problem size, $\sigma_u = 2.0,\ dr = 100$

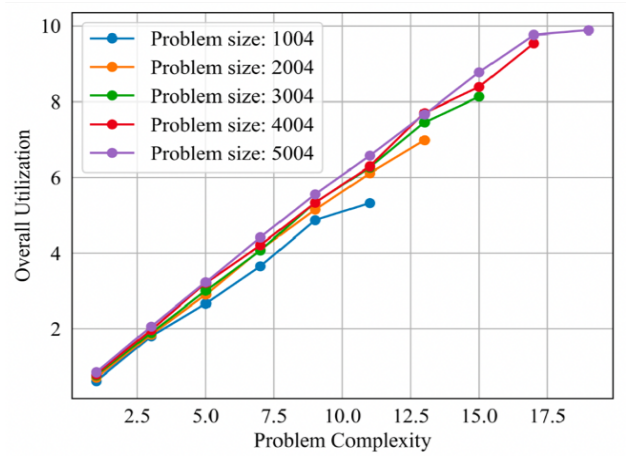

Figure 12: Problem complexity vs overall utilization at various values of problem size, $\sigma_u = 3.0,\ dr = 150$

has been observed that upon increasing the problem size, the feasibility ratio increases for the same value of problem complexity and the cutoff is delayed. This points to a subtle nature of the problem that the change in the number of processors (r) has a greater impact on feasibility ratio than the change in number of tasks (n).

7.2.2 At various values of σ_u

The results (figures 4 to 6) indicate that lowering the value of the scale of the exponential distribution improves the feasibility ratio. This result matches the expected behavior since at lower values of σ_u, task systems with higher utilization behavior are likely to be generated which induce higher feasibility.

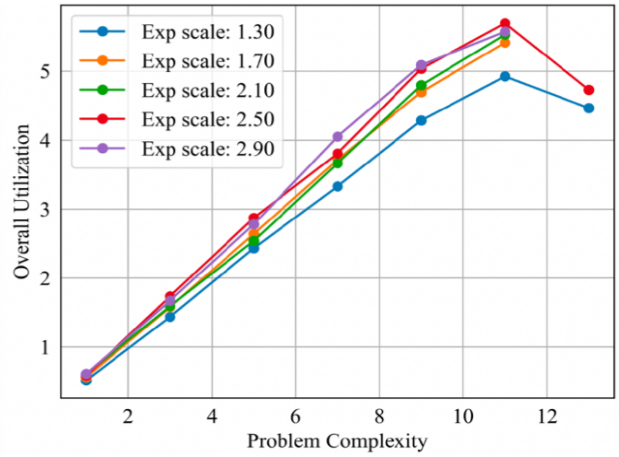

Figure 13: Problem complexity vs overall utilization at various values of σ_u, $s = 1000,\ dr = 50$

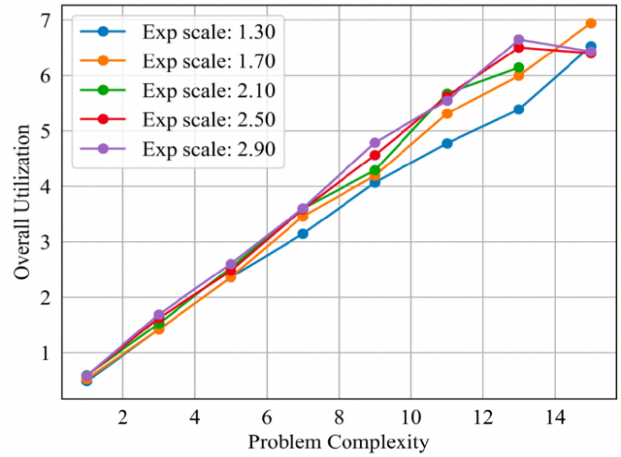

Figure 14: Problem complexity vs overall utilization at various values of σ_u, $s = 2000,\ dr = 100$

7.2.3 At various values of deadline range

The results (figures 7 to 9) indicate that increasing the deadline range improves the feasibility ratio. This is the expected behavior since the generator has a greater degree of freedom in selecting deadlines when the deadline range is larger and the likelihood of selecting deadlines that induce feasible task systems increases.

7.3 Problem Complexity vs Overall Utilization Analysis

Another important system parameter that needs to be considered for the analysis of task system generation algorithm is the overall utilization. The variation of overall utilization with problem complexity at various values of the system parameters have been discussed next.

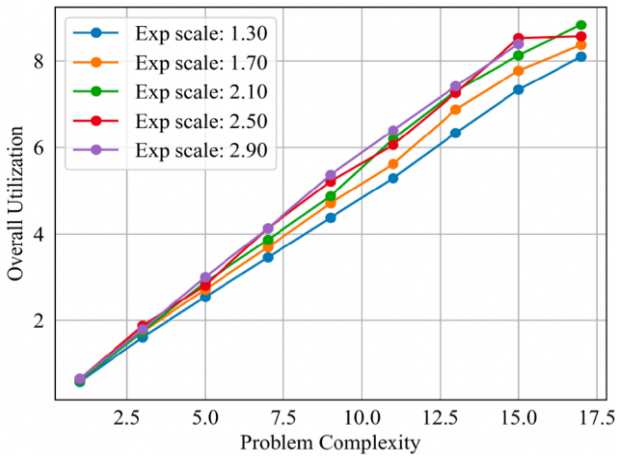

Figure 15: Problem complexity vs overall utilization at various values of σ_u, $s = 3000$, $dr = 50$

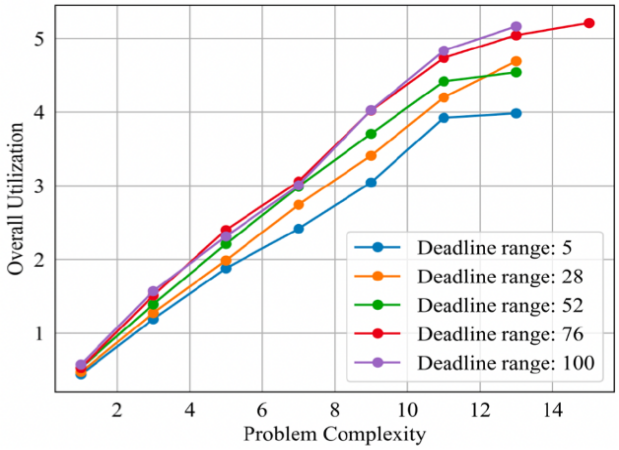

Figure 16: Problem complexity vs overall utilization at various values of deadline range, $\sigma_u = 1.0$, $s = 1000$

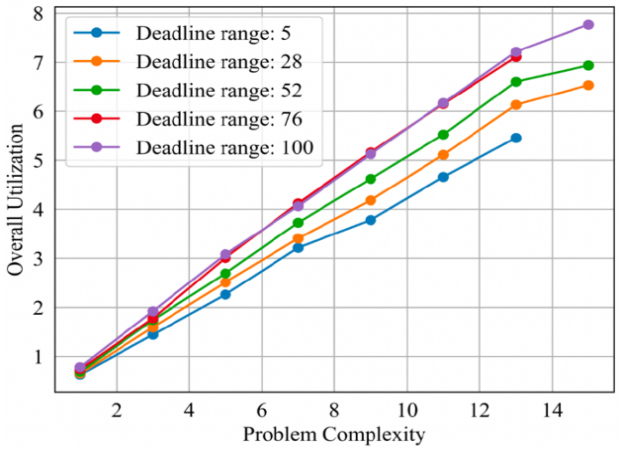

Figure 17: Problem complexity vs overall utilization at various values of deadline range, $\sigma_u = 2.0$, $s = 2000$

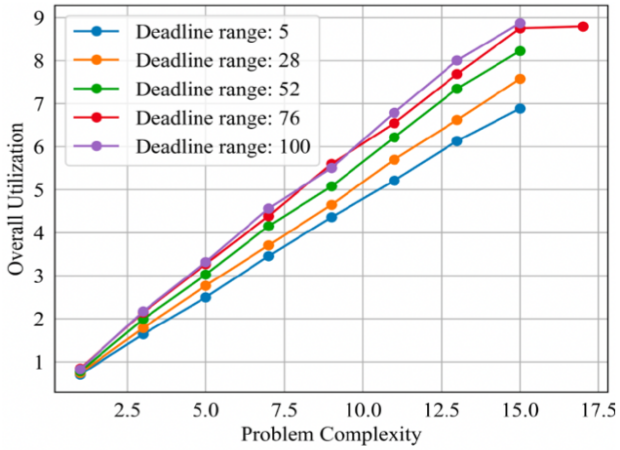

Figure 18: Problem complexity vs overall utilization at various values of deadline range, $\sigma_u = 3.0$, $s = 3000$

7.3.1 At Various Values of Problem Size

A general trend of increasing overall system utilization has been observed upon increasing the problem complexity. At the same complexity, with the increase in problem size, the results (figures 10 to 12) show an increase in overall utilization. With the increase in problem size, the number of constraints in the ILP of the partitioned EDF test increases, which worsens (increases) the optimal value of the objective (the overall utilization).

7.3.2 At various values of σ_u

With the increase in the scale of the exponential distribution, the overall utilization has been observed to increase in the results (figures 13 to 15) at the same value of problem complexity. With the shrinking of the feasible region (as was evident from the feasibility ratio analysis), at higher values of σ_u, the optimal utilization is worsened (increased).

7.3.3 At various values of deadline range

At the same value of problem complexity, the overall utilization has been observed to increase in the results (figures 16 to 18) upon increasing the deadline range. The relation between deadline range and overall utilization is implicit. The task system generation algorithm generates WCETs as $C_{ij} \leftarrow \lfloor U_{ij} \times D_i \rfloor$. With higher value of deadline range, with the minimum deadline held constant, higher values of D_i are generated which increase the values of C_{ij} . Task systems with higher values of WCETs naturally have a higher value of overall utilization.

8. Conclusion

In this work, an algorithm for generation of implicit-deadline, hard real-time, recurrent, sporadic task systems on unrelated multiprocessor platforms was proposed. Both theoretical and quantitative analyses were performed. The quantitative analysis has shown that the proposed algorithm produces tasksystem populations with feasibility ratio almost equal to 1 until a cutoff is reached. The population has also been shown to exhibit growing overall utilization upon increasing problem complexity. These properties align with the theoretical expectation of tasksystem population behavior. Hence, the algorithm is stable and reliable for tasksystem generation. Also, the theoretical analysis has shown that the algorithm can be made to have linear time complexity in the average case for scale parameter $\sigma_u \geq 3.0$. Hence, the algorithm is efficient in terms of runtime for generating task systems.

References

[1] Pontus Ekberg and Sanjoy Baruah. Partitioned scheduling of recurrent real-time tasks. In *2021 IEEE Real-Time Systems Symposium (RTSS)*, pages 356–367. IEEE, 2021.

[2] Sanjoy Baruah, Marko Bertogna, and Giorgio Buttazzo. *Multiprocessor scheduling for real-time systems*. Springer, 2015.

[3] Enrico Bini and Giorgio C Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1):129–154, 2005.

[4] Paul Emberson, Roger Stafford, and Robert I Davis. Techniques for the synthesis of multiprocessor tasksets. In *proceedings 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010)*, pages 6–11, 2010.

[5] Marko Bertogna. Real-time scheduling analysis for multiprocessor platforms. 2008. Phd Thesis.

[6] Marko Bertogna and Michele Cirinei. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, pages 149–160. IEEE, 2007.

[7] Marko Bertogna, Michele Cirinei, and Giuseppe Lipari. Schedulability analysis of global scheduling algorithms on multiprocessor platforms. *IEEE Transactions on parallel and distributed systems*, 20(4):553–566, 2008.

[8] Theodore P Baker, Michele Cirinei, and Marko Bertogna. Edzl scheduling analysis. *Real-Time Systems*, 40(3):264–289, 2008.

[9] Robert I Davis and Alan Burns. Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In *2009 30th IEEE Real-Time Systems Symposium*, pages 398–409. IEEE, 2009.

[10] Roger Stafford. Random vectors with fixed sum. `https://www.mathworks.com/matlabcentral/fileexchange/9700-random-vectors-with-fixed-sum`, 2022.

[11] Antoine Bertout, Joël Goossens, Emmanuel Grolleau, and Xavier Poczekajlo. Workload assignment for global real-time scheduling on unrelated multicore platforms. In *Proceedings of the 28th International Conference on Real-Time Networks and Systems*, pages 139–148, 2020.

[12] Christian El Salloum, Martin Elshuber, Oliver Höftberger, Haris Isakovic, and Armin Wasicek. The across mpsoc–a new generation of multi-core processors designed for safety–critical embedded systems. *Microprocessors and Microsystems*, 37(8):1020–1032, 2013.

[13] Ravikumar V Chakaravarthy, Hyun Kwon, and Hua Jiang. Vision control unit in fully self driving vehicles using xilinx mpsoc and opensource stack. In *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 311–317. IEEE, 2021.

[14] Lucas Mohimont, Mathias Roesler, Luiz Angelo Steffenel, Nathalie Gaveau, Marine Rondeau, François Alin, Clément Pierlot, Rachel Ouvinha de Oliveira, Marcello Coppola, and Philipe Doré. Ai-driven yield estimation using an autonomous robot for data acquisition, 2021.

[15] Marko Bertogna. Evaluation of existing schedulability tests for global edf. In *2009 International Conference on Parallel Processing Workshops*, pages 11–18. IEEE, 2009.

[16] Theodore P Baker and Michele Cirinei. A necessary and sometimes sufficient condition for the feasibility of sets of sporadic hard-deadline tasks. In *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*, pages 178–190. IEEE, 2006.

[17] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[18] Ailsa H Land and Alison G Doig. An automatic method for solving discrete programming problems. In *50 Years of Integer Programming 1958-2008*, pages 105–132. Springer, 2010.

[19] Laurent Perron and Vincent Furnon. Or-tools. `https://developers.google.com/optimization/`.