IEICE TRANS. INF. & SYST., VOL.E105–D, NO.10 OCTOBER 2022

PAPER

Evaluation and Comparison of Integer Programming Solvers for Hard Real-Time Scheduling

Ana GUASQUE[†a)] ***and*** **Patricia BALBASTRE**[†b)], ***Nonmembers***

SUMMARY In order to obtain a feasible schedule of a hard real-time system, heuristic based techniques are the solution of choice. In the last few years, optimization solvers have gained attention from research communities due to their capability of handling large number of constraints. Recently, some works have used integer linear programming (ILP) for solving mono processor scheduling of real-time systems. In fact, ILP is commonly used for static scheduling of multiprocessor systems. However, two main solvers are used to solve the problem indistinctly. But, which one is the best for obtaining a schedulable system for hard real-time systems? This paper makes a comparison of two well-known optimization software packages (CPLEX and GUROBI) for the problem of finding a feasible schedule on monoprocessor hard real-time systems.

key words: *integer linear programming, hard real-time scheduling, optimization*

1. Introduction

Modern real-time embedded systems comprise many applications executing on the same computing platform. If a hard real-time task misses any temporal constraint in high criticality applications, it may suppose catastrophic results. This implies that the feasibility of the temporal model must be assured offline. Moreover, task scheduling belong to the class of NP-complete problems and can be solved efficiently only by heuristics. Heuristics are able to find a feasible plan in polynomial time. However, to further minimise or maximise a certain parameter (worst case response time, context switch, power consumption, etc.) there are no heuristics that ensure an optimal result. Brute-force algorithms are one possibility, but they have an exponential cost, especially when the hyperperiod is large. Best-effort algorithms are used, where there is no guarantee that the optimal solution has been found, that is, we can find a feasible plan with low power consumption or response time but there is no way to know if it is the minimum value.

In the last few years, optimization solvers have gained attention from research communities due to their capability of handling large number of constraints. The advantage of integer linear programming (ILP) solutions is that they can optimise an allocation toward a certain goal by changing the optimization criteria. However, ILPs or MILPs (Mixed-Integer Linear Programming) are more used in multi processor than in mono processor systems due to the lack of optimal heuristics having a polynomial complexity in multi processor scheduling. However, optimization techniques can exploit the possibility of optimize a static scheduling plan for a specific performance parameter even in mono processor systems.

Two main solvers are used to solve the problem indistinctly. But, which one is the best for obtaining a schedulable system for hard real-time systems? As far as we know, there is no similar comparison specifically for hard real-time scheduling. Moreover, in the literature, when a MILP solver is used to obtain optimal scheduling, there is no justification as to why one solver is used and not another.

The companies that market the most common solvers present their own benchmarks showing that their product is the fastest and outperforms the others. It is difficult to judge their arguments because it seems that the effectiveness of a solver depends very much on the type of problem it is intended to solve.

This paper makes a comparison of two well-known optimization software packages (CPLEX and GUROBI) for monoprocessor hard real-time systems. This comparison is needed to find out which one offers the best solution in the shortest possible time. This way, with this work we will give to real-time systems engineers a hint about which solver is the best for their purposes.

The rest of the papers is organized as follows: Section 2 presents the main related works on the topic, Sect. 3 describes the main characteristics of the most common solvers used in the real-time systems community. Section 4 presents the task model used. Section 5 describes the mathematical model used to obtain the scheduling plan of a task set in mono processor real-time systems. In Sect. 6 the proposed model is evaluated for Gurobi and CPLEX while in Sect. 7 the main conclusions are pointed out.

2. Related Work

A number of papers have recently been published comparing different optimisation solvers under certain conditions and for different types of applications. The most frequent comparison is between Gurobi and CPLEX. These two solvers are also the most widely used in the area of hard real-time systems.

In [1], Gurobi and CPLEX are compared for a specific problem type. The result of the comparison is that both perform well, although CPLEX performs better when there are many variables and the problem type is quadratic. The

Manuscript received May 5, 2022.
Manuscript publicized July 21, 2022.

[†]The authors are with the Universitat Politècnica de València, Spain.
a) E-mail: anguaor@ai2.upv.es
b) E-mail: patricia@ai2.upv.es

DOI: 10.1587/transinf.2022EDP7073

tabu search method for the boolean optimization problem is compared in [2]. For this specific problem, Gurobi is the fastest solver, and gives slightly better results than CPLEX. CPLEX is a bit faster than Gurobi in terms of proving the optimal solution.

In [3] the so-called Integer Linear Generalized Maximum Multiplicative Programs (IL-GMMP) is studied with the development of three new multi-objective optimization based algorithms. With 57600 experiments, the performance of Gurobi, CPLEX and FICO Xpress is compared when used for solving single-objective integer linear programs. As a conclusion, in general, CPLEX was shown to be the best choice. A mixed-integer linear programming model was proposed to solve the problem of customer scheduling and truck assignment for hospital waste collection in [4]. The model has been implemented and the performance of commercial solvers, GUROBI and CPLEX, to obtain an optimal solution were tested. GUROBI solver's performance was superior to that of the CPLEX solver. In the context of non-convex QP problems, CPLEX and Gurobi are also compared being Gurobi which presents the best results [5]. For solving the problem of mining production planning [6], CPLEX is more efficient than Gurobi solving the proposed model in medium-sized instances. In the case of smaller instances, Gurobi obtains a solution in less time than CPLEX. [7] focus on the performance of the two solvers working in parallel. In this case, CPLEX outperforms Gurobi. Another interesting comparison of both commercial solvers is presented in [8] for quadratic combinatorial optimization problems. Their conclusions were that "CPLEX prefers symmetric or symmetric with a diagonal perturbation yields a positive definite matrix, whereas GUROBI prefers an upper triangular Q matrix". In [9], the comparison is done between CPLEX, FICO XPRESS and Gurobi using a subset of the MIPLIB 2010 library that contains 361 test instances of different hardness (easy, hard, and not solved). According to the proposed methodology GUROBI was identified as the best MILP solver since it reached best results in 60% of instances.

Regarding the works that obtain feasible schedules with ILP techniques in complex real-time models includes, among others, multiprocessor systems [10], power consumption optimization [11], consideration of architectures with local instruction or data caches [12], weakly hard real-time systems [13], mixed criticality [14], etc. More specifically, regarding the use of ILP in real-time systems, [15] use CPLEX to solve the scheduling of periodic real-time tasks with duplication in heterogeneous multiprocessor. In [16], tasks are split into read, execute and write subtasks in order to reduce the contention delay in multicore hard real-time systems, also using CPLEX. Regarding the use of ILP in real-time systems with Gurobi, this solver is used for generating communication schedules in multi-mode distributed real-time applications in [17]. And in [12], Gurobi is used to obtain a schedule that takes into account the effect of cache in the WCET so the length of the schedule is significantly reduced.

As a conclusion of the previous work, although Gurobi performs better in many cases, there is no one solver that dominates completely the others. Furthermore, none of the above-mentioned works compare solvers for the specific domain of real-time systems scheduling.

3. Optimization Solvers

Available optimization solvers differ in many ways. They come with different licenses and different features, for example in terms of how problems can be specified. There are several commercial or free solvers but, taking a look at the literature of the real-time research area, the following are the most used.

The IBM ILOG CPLEX Optimization Studio [18] which is often referred to simply as CPLEX is a commercial solver designed to tackle (among others) large scale (mixed integer) linear problems. CPLEX is now actively developed by IBM. The software also features several interfaces so that it is possible to connect the solver to different program languages and programs. However, also a stand-alone executable is provided. The optimizer is also accessible through modeling systems. CPLEX provides three families of heuristics to find integer solutions at nodes during the branch & cut procedure:

- Node heuristic: employs techniques to try to construct a feasible solution from the current branch & cut node.
- Relaxation Induced Neighborhood Search (RINS) Heuristic: explores a neighborhood of the current incumbent solution to try to find a new, improved incumbent.
- Solution Polishing: can be used to improve the best known solution at the end of branch & cut if optimality has not been proven.
- Feasibility pump ([19]) is a heuristic that finds an initial feasible solution even in certain very hard mixed integer programming problems

With default parameter settings, CPLEX automatically invokes the heuristics when they seem likely to be beneficial. Last version is 20.1.

The Gurobi Optimizer [20] is a modern solver for (mixed integer) linear as well as other related (non-linear, e.g.) mathematical optimization problems. The Gurobi Optimizer is written in C and it is available on all computing platforms and accessible from several programming languages. Standard independent modelling systems can be used to define and to model problems. Gurobi provides general-purpose heuristics. As it is a commercial product, there is not detailed information about how it works but the company claims that quickly finding feasible solutions. By default, the heuristic run at the beginning of a solve, after the LP relaxation and within the branch and cut tree. Last version is 9.5.

Both solvers have interfaces to program in C, C++, Matlab and Python and have academic, free licenses. In the next sections, we are going to present the problem that we want to solve, and the mathematical model presented as a MILP.

4. Task Model

Our task model is composed by n periodic real-time tasks that we assume that are independent (no shared resources nor precedence relationship).

$$\tau = [\tau_1, \ldots, \tau_n] \tag{1}$$

A task τ_i is a tuple with the following parameters:

$$\tau_i = (C_i, D_i, T_i) \tag{2}$$

being C_i the execution time of each activation in the worst case, D_i is the due time or relative deadline and T_i is the task period.

The absolute due time of activation a is $d_{ia} = a \cdot T_i + D_i$. We may assume, without loss of generality, that all previous parameters are integer values. The task utilization U_i is the ratio between the execution time and the period, $U_i = \frac{C_i}{T_i}$. As we focus on hard real-time systems, we assume that $D_i \leq T_i$.

Without loss of generality, we assume that the tasks are ordered by increasing deadline. If the deadline of two tasks coincide, an increasing period order is assumed. We define the hyperperiod (lcm) of the set of tasks as the time at which the release of tasks coincide in time. It corresponds with the least common multiple of the tasks periods.

Another important parameter is the time it takes for a task τ_i to finish execution at a given activation a. This time is w_{ia}. Moreover, it is important to quantify the highest value of all completion times of all activations throughout the hyperperiod. This maximum completion time is called the worst case response time (WCRT) defined and calculated in [21], [22]. It can be calculated as:

$$WCRT_i = \max\{w_{ia}\} \tag{3}$$

On the contrary, the minimum completion time of all activations is defined as the best case response time ($BCRT_i = \min\{w_{ia}\}$). The relationship between both parameters is:

$$BCRT_i \leq WCRT_i \tag{4}$$

And if $WCRT_i \leq D_i$, the task set will be feasible.

Figure 1 shows the parameters that define a real-time periodic task.

We assume that this work is applied to hard real-time systems that are scheduled by means of a static scheduler, i.e. the generation of the plan takes place a priori, in an off-line manner.

The problem to solve is to obtain the schedule of the task set, that is, the instants in which each task has to be executed to meet the deadlines and to optimize some additional parameter.

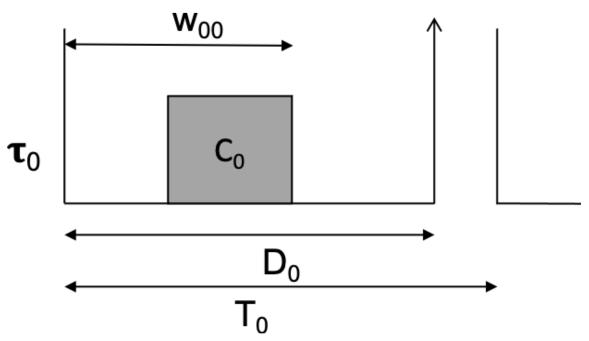

Fig. 1 Real-time task definition

Table 1 Task set example

	C	D	T
τ_1	1	4	4
τ_2	2	5	5
τ_3	2	8	8

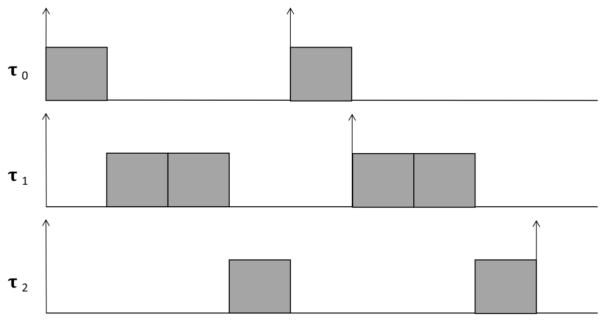

Fig. 2 Chronogram of the example under DM

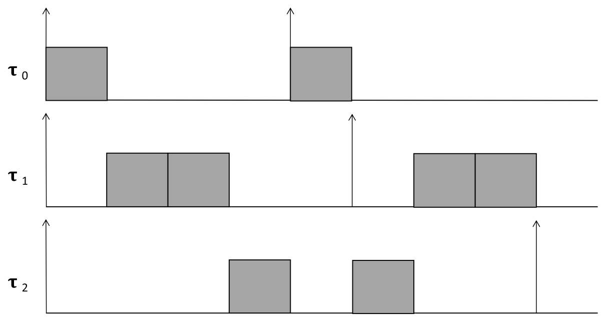

Fig. 3 Chronogram of the example with slight change

4.1 Example

Consider the set of periodic tasks in Table 1.

Figure 2 shows the execution chronogram when the task set is scheduled under a fixed priority algorithm. In this case, priorities are inversely proportional to deadlines (Deadline Monotonic algorithm [23]). When using DM, the WCRT of a task coincides with the response time of the first activation (w_{i0}). As we can observe in the figure, w_{i0} of tasks is 1, 3 and 8.

However, we can see in Fig. 3 how a slight change in the scheduling can improve the WCRT of some tasks. As it can be seen, if task 2 is delayed 1 instant of time, WCRT of task 3 improves considerably. This way, this second schedule obtains 1, 3, and 6 as WCRT. Note that now the WCRT does not coincide with the first activation.

In the next section, we will approach the problem of real-time systems scheduling as an optimization problem with the goal of improving the scheduling obtained with traditional real-time schedulers.

5. MILP Model

In this section, the optimisation problem of obtaining the scheduling plan of a task model as presented in Sect. 4 will be detailed. The goal is to minimise the worst case response time of all tasks or, more precisely, the average worst case response time normalized over the deadline. Low response times have several advantages, for example, in control applications, where the system's stability is assured if variation in response time is low [24].

Table 2 introduces the model details.

$$\min \quad \text{Obj} = \sum_{\forall(i,a_i)} \frac{w_{ia_i}}{D_i}$$

s.t:

$$\sum_{t \in R_{ia_i}} x_{it} = C_i \quad \forall i, a_i \tag{5}$$

$$t \cdot x_{it} \leq d_{ia_i} - 1 \quad \forall t \in R_{ia_i} \tag{6}$$

$$\sum_i x_{it} \leq 1 \quad \forall t \in \{0, 1, \ldots, lcm\} \tag{7}$$

$$w_{ia_i} \geq t \cdot x_{it} - a_i \cdot T_i + 1 \quad \forall t, i, a_i \tag{8}$$

$$x_{it} \in \{0, 1\} \tag{9}$$

$$w_{ia_i} \geq 0 \tag{10}$$

We want to obtain the task execution matrix (x_{it}) which is a binary matrix that represents whether or not a task τ_i executes at time t. With the content of this matrix we are able to know the task response times, which are the variables to minimize. The scheduling has to meet some rules, that are expressed by the constraints. As the model includes both integer and binary variables, it is a MILP problem.

Constraints 5, 6 and 7 are related to the schedulability conditions that the scheduling plan must accomplished. Constraint 5 assures that the task executes C_i units of time inside each activation of period T_i. Constraint 6 ensures that the task is schedulable, that is, the task finishes its execution in all activations before the due time D_i. Constraint 7 is related to the scheduling in mono processor systems since, at each instant of time, no more than one task can execute. Constraint 8 calculates the response time of each task in all activations. Equations (9) and (10) define the decision variable domains.

This model has been implemented in two of the most common used solvers for MILP. The description of the evaluation and obtained results is presented in Sect. 6.

Table 2 MILP model notation

Sets and Indices	
i	Tasks $\tau_i \in \{1, 2, \ldots, n\}$
a_i	Activations of $\tau_i \in \{0, 1, \ldots, N_i - 1\}$
Parameters	
C_i	Worst case computation time of τ_i
D_i	Relative deadline of τ_i
T_i	Period of τ_i
lcm	Hyperperiod of the task set
N_i	Number of activations of τ_i (lcm/T_i)
d_{ia_i}	Due date of τ_i in activation a_i
R_{ia_i}	$[a_i \cdot T_i, (a_i+1) \cdot T_i]$ Possible execution time interval for task i in activation a_i
Decision variables	
x_{it}	Task execution matrix. 1 if τ_i is executed at time t and 0 otherwise.
w_{ia_i}	Response time matrix. Response time of τ_i in activation a_i

6. Evaluation

The simulation process followed in this paper is showed in Fig. 4. It is divided in three steps:

- Synthetic random generation of the task sets.
- Execution of MILP solvers.
- Validation.

The load is generated using a synthetic task generator. A number of tests have been run, specifically, 10000 synthetic task sets have been generated for system utilizations varying from 0.2 to 0.8 in steps of 0.1. The total number of tasks varies from 2 to 6. Each task τ_i was generated by randomly choosing the worst case execution time (C_i) as an integer between 2 and 10 and periods are deduced from the system utilization (U_i) using the following expression (see Sect. 4):

$$T_i = \frac{C_i}{U_i} \tag{11}$$

We assume that deadlines are equal to periods. The total utilization is shared among the tasks using the popular UUniFast discard algorithm [25]. To limit the maximum runtime of the simulations, which is highly dependent on

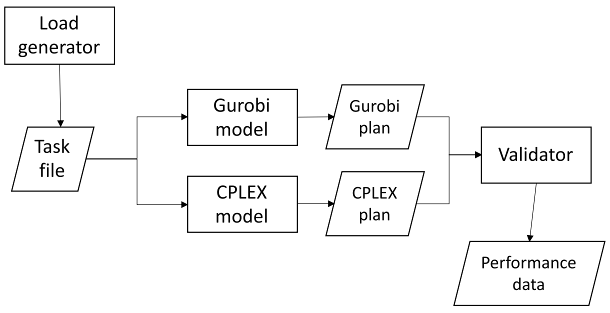

Fig. 4 Experimental evaluation overview

the hyperperiod, we restrict the hyperperiod to be not larger than 200, so we discard the sets that exceed this limit, repeating the generation again until enough feasible instances are found. Although this upper bound of the hyperperiod may seem small, there are techniques to reduce it when this value is larger by modifying the values of the task periods slightly. A very effective technique for drastically reducing the hyperperiod is presented in [26].

The validation consists of two steps: on the one hand, feasibility is checked so all deadlines are met in all task sets. On the other hand, some performance parameters are obtained to compare the two solvers. Specifically, these are the parameters obtained for each set:

- Response time: we calculate the average response time as defined in Sect. 4 respect to the deadline of the task, that is, $\sum_{i,a_i} w_{i,a_i}/D_i$. This is the same expression as the objective function that the MILP problem tries to minimize.

- Solution time: For both solvers, we calculate the time spent to obtain a solution and to confirm that this is the optimum. As simple MILP models are too computationally-intensive as the number of tasks, *lcm* or U_i increases, both solvers have a configurable parameter which is the time limit. After this time, the solution is stored.

- Optimality gap: If the time limit commented previously is reached, the solution has not been proved to be optimal but it might be. Therefore, the distance to a lower bound on the optimal solution is also stored. This distance is called the *gap* hereafter. It is the estimated distance between the lower bound on the best possible objective and best found objective. As looking for proven optimal solutions takes a long time to compute, a common practice is look for a solution that guarantees not worse than x% (*gap*) from the optimal solution. A significant advantage is that most of the gap is often reduced quite early.

For all the experiments, CPLEX 20.1 and GUROBI 9.5 are used with a Free Academic License. Python interface has been used for both solvers. The experiments are executed in a Acer Aspire V5-591G with 2 Intel Core i7-6700HQ 2.60GHz and 16.0 GB RAM memory.

These experiments were run with a time limit of 500 seconds and a reached gap of 1% for both solvers. This means that when the solvers find a solution with a gap less that this reached gap, it stops. The statistics of the whole experiments are shown in Table 3.

As it can be seen in the table, CPLEX presents better results (on average value) in solution times and gap while both present very similar results in response time.

Raw values of solution times are represented in Figs. 5 and 6. We can see that for few tasks CPLEX and Gurobi obtain the optimum quickly. However, as the number of tasks

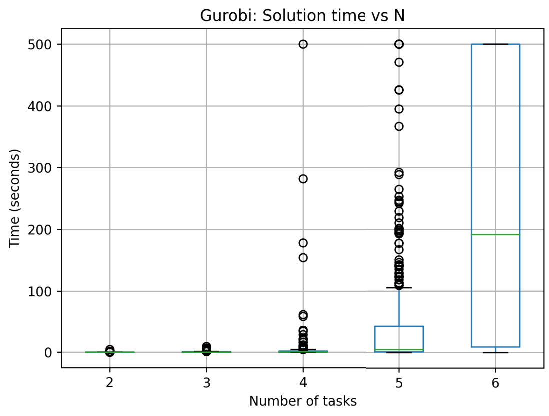

Fig. 5 Solution time vs number of tasks for Gurobi

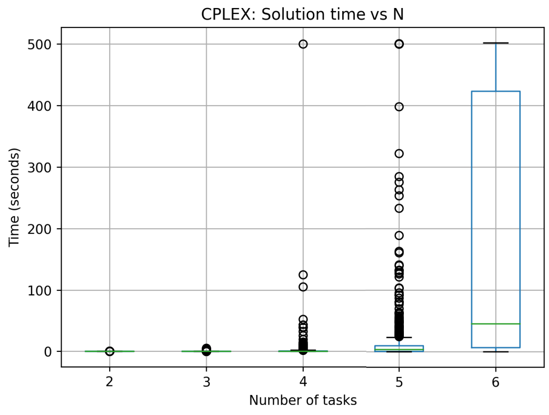

Fig. 6 Solution time vs number of tasks for CPLEX

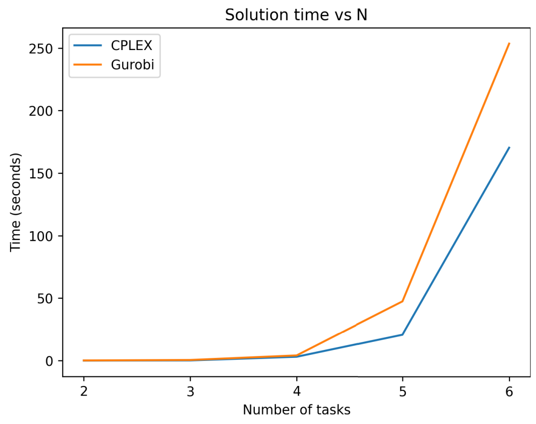

Fig. 7 Solution time vs number of tasks

increases, it becomes more expensive to find the optimal solution for both solvers, although CPLEX is faster.

In order to see the relationship between the obtained values for both solvers, and the temporal parameters of the model, we have graphically represented this relationship in the following figures. All the figures represent the average values of response time, solution time and gap of all the task sets executed. The results regarding solution times are shown in Fig. 7. Figure 7 represents the time that Gurobi and CPLEX need to obtain the solution as a function of the number of tasks. In both solvers, from 4 tasks, the more number of tasks, the more time to find a solution. It shows the same behaviour as in Figs. 5 and 6.

The solution time is also depicted vs. utilization and hyperperiod in Figs. 8 and 9. We choose these parameters (U, N and H) because these parameters are directly related to the dimension of the problem. For x_{it}, N is the number of rows of the matrix, and lcm is the number of columns. Therefore, the higher these two parameters are, the longer it will take to obtain a solution. U is directly proportional to the number of non-zero elements of x_{it}. However, with regard to utilization, CPLEX outperforms Gurobi in all cases. Regarding the hyperperiod, CPLEX is observed to be faster from a hyperperiod of 180. Up to that point, the behaviour is very similar.

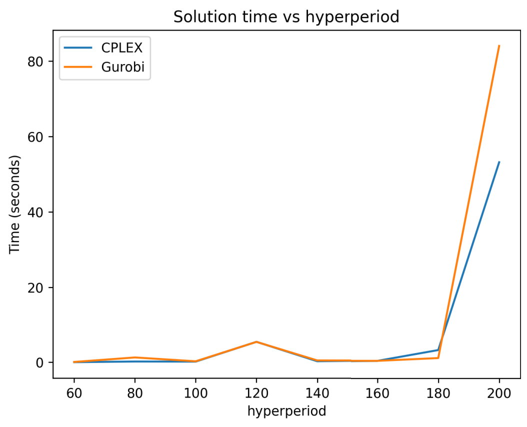

Fig. 8 Solution time vs hyperperiod

To know if this obtained solution is the optimal, we can see the results obtained for the response time, which is the objective that both solvers attempt to minimize. These results are depicted in Fig. 10 with respect to the utilization of the task set. We can see in the graph how the results of Table 3 are confirmed as both solvers obtain the same results. Figure 11 shows the obtained *gap* for both solvers. Again, results are very similar for few tasks but in this case the most significant differences occur where there are 6 tasks. The difference can be explained by the fact that in those cases where the time limit of 500 seconds is reached, the gap will be larger than 1%. Since Gurobi takes longer to find the optimum, presumably there will be more cases where it reaches the time limit and gets a larger gap.

It seems clear that the most significant differences between Gurobi and CPLEX are observed for more than 4 tasks. To investigate behaviour with more tasks, new experiments have been done with 7, 8, 9 and 10 tasks. However, the time to reach the optimal solution is very long (more than 500 seconds in most cases) taking several days to obtain the optimum for a significant number of simulations. To compare both solvers with 7, 8, 9 and 10 tasks, the time limit has been reduced to 10 seconds and the gap and response time obtained between Gurobi and CPLEX will be observed. We know that 10 seconds is not enough time to find the optimum but we want to know how close each solver is to finding it in this time.

So for this second set of experiments, 10000 synthetic task sets have been generated for system utilizations varying from 0.2 to 0.8 in steps of 0.1. The total number of tasks varies from 7 to 10. Each task τ_i was generated by randomly choosing the worst case execution time (C_i) as an integer

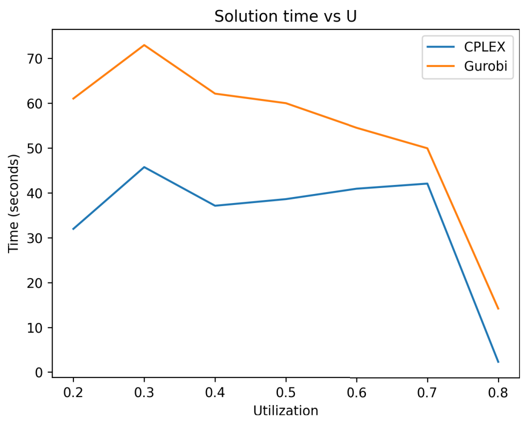

Fig. 9 Solution time vs utilization

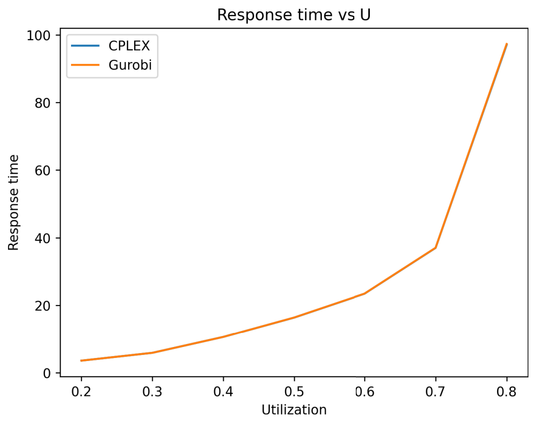

Fig. 10 Response time vs utilization

Table 3 Gurobi and CPLEX results statistics

	Response time		Solution time		gap (%)	
	CPLEX	Gurobi	CPLEX	Gurobi	CPLEX	Gurobi
Mean	14.084	14.1	38.874	61.146	0.8118	1.1104
Std. deviation	19.343	19.406	116.909	148.491	1.0920	2.0995
Min	0.285	0.285	0.0035	0.0016	0	0
Max	150.884	151.43	500	500	1.73172	3.40386

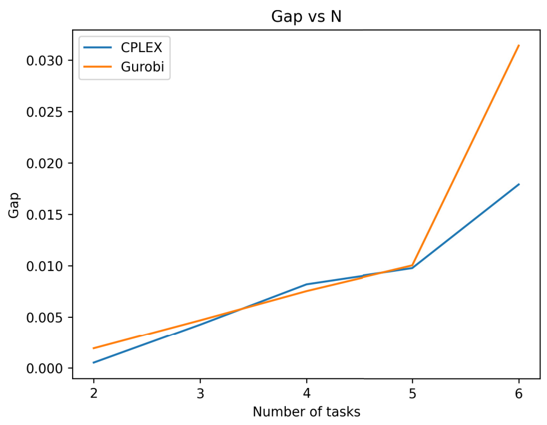

Fig. 11 *gap* vs number of tasks

Table 4 Gurobi and CPLEX results statistics for 7–10 tasks (time limit = 10 seconds)

	Response time		gap (%)	
	CPLEX	Gurobi	CPLEX	Gurobi
Mean	58.767	131.737	52.662	72.518
Std. deviation	59.518	157.765	27.462	2.0995
Min	0.973	1.099	0.917	0.825
Max	550.47	858.794	99.86	99.97

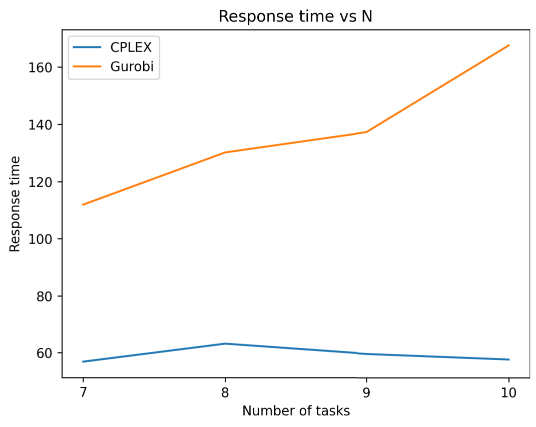

Fig. 12 Response time vs number of tasks for time limit of 10 seconds

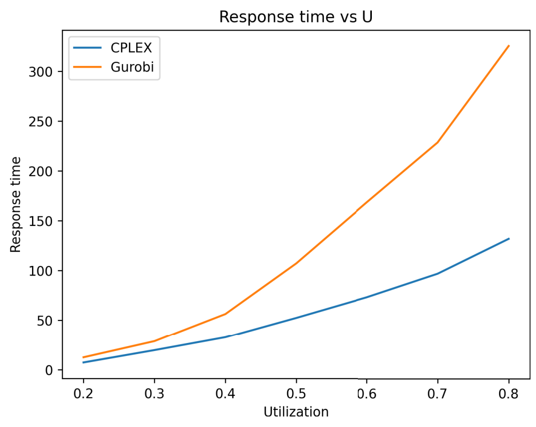

Fig. 13 Response time vs utilization for time limit of 10 seconds

between 1 and 99 and periods are deduced from the system utilization (U_i) using Eq. (11).

The statistics of these experiments are shown in Table 4. We can see how, when there is not enough time to find the optimum, CPLEX is able to find a lower value for the response time than Gurobi.

The response time results vs. utilization and number of tasks are presented in Figs. 12 and 13. The figures confirm the better performance of CPLEX in terms of finding a lower value of the objective function as the number of tasks increases. The difference between the two solvers also becomes more pronounced as utilization increases.

7. Conclusion

In this paper, the problem of finding a feasible schedule of a hard real-time monoprocessor system has been addressed as a mixed integer linear programming problem. The goal of this paper was to compare two optimization solvers (Gurobi and CPLEX) which are the most commonly used in this type of problems. The comparison has been made in terms of time to reach the optimal solution, the solution obtained and the distance to the optimal. The results show how CPLEX performs better than Gurobi on average. Depending on the number of tasks and the hyperperiod, results may vary but, in general, both solvers behave very similarly although CPLEX performs faster than Gurobi in some cases.

As a conclusion to this work, we would recommend using Gurobi or CPLEX interchangeably when the load is low (few tasks or low hyperperiod). However, when there are more than 4 tasks, it is better to use CPLEX because of its shorter time to reach the optimal solution. Even when the time to reach the optimum is limited, CPLEX obtains a lower value of the objective function.

Further work includes to extend the use of MILP to multiprocessor systems in general and to partitioned systems in particular and exploring different mixed optimization functions such as context switches, jitter or power consumption.

Acknowledgments

This work was supported under Grant PLEC2021-007609 funded by MCIN/ AEI/ 10.13039/501100011033 and by the "European Union NextGenerationEU / PRTR".

References

[1] R. Anand, D. Aggarwal, and V. Kumar, "A comparative analysis of optimization solvers," Journal of Statistics and Management Systems, vol.20, no.4, pp.623–635, 2017.

[2] L.M. Hvattum, A. LÃžkketangen, and F. Glover, "Comparisons of commercial mip solvers and an adaptive memory (tabu search) procedure for a class of 0–1 integer programming problems," Algorithmic Operations Research, vol.7, 2012.

[3] P.G. Saghand and H. Charkhgard, "Exact solution approaches for integer linear generalized maximum multiplicative programs through the lens of multi-objective optimization," Computers and Operations Research, vol.137, p.105549, 2022.

[4] R. Linfati, G. Gatica, and J.W. Escobar, "A mathematical model for scheduling and assignment of customers in hospital waste collection routes," Applied Sciences, vol.11, no.22, 2021.
[5] G. Liuzzi, M. Locatelli, V. Piccialli, and S. Rass, "Computing mixed strategies equilibria in presence of switching costs by the solution of nonconvex QP problems," Computational Optimization and Applications, vol.79, no.3, pp.561–599, July 2021.
[6] C. Flores-Fonseca, R. Linfati, and J.W. Escobar, "Exact algorithms for production planning in mining considering the use of stockpiles and sequencing of power shovels in open-pit mines," Operational Research, vol.22, no.3, pp.2529–2553, 2022.
[7] M. González, J.J. López-Espın, and J. Aparicio, "A parallel algorithm for matheuristics: A comparison of optimization solvers," Electronics, vol.9, no.9, 2020.
[8] A.P. Punnen, P. Pandey, and M. Friesen, "Representations of quadratic combinatorial optimization problems: A case study using quadratic set covering and quadratic knapsack problems," Computers and Operations Research, vol.112, p.104769, 2019.
[9] J. Jablonský, "Recent optimization packages and their comparison," Hradec Economic Days, vol.7, no.1, 2017.
[10] S. Baruah, "Feasibility analysis of preemptive real-time systems upon heterogeneous multiprocessor platforms," 25th IEEE International Real-Time Systems Symposium, pp.37–46, 2004.
[11] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M.B. Srivastava, "Power optimization of variable-voltage core-based systems," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol.18, no.12, pp.1702–1714, 1999.
[12] V.A. Nguyen, D. Hardy, and I. Puaut, "Cache-conscious off-line real-time scheduling for multi-core platforms: algorithms and implementation," Real-Time Systems, vol.55, no.4, pp.810–849, 2019.
[13] Y. Sun and M.D. Natale, "Weakly hard schedulability analysis for fixed priority scheduling of periodic real-time tasks," ACM Trans. Embed. Comput. Syst., vol.16, no.5s, pp.1–19, 2017.
[14] T. Fleming and A. Burns, "Investigating mixed criticality cyclic executive schedule generation," Proc. Workshop on Mixed Criticality (WMC), 2015.
[15] W. Zhang, Y. Hu, H. He, Y. Liu, and A. Chen, "Linear and dynamic programming algorithms for real-time task scheduling with task duplication," The Journal of Supercomputing, vol.75, no.2, pp.494–509, 2019.
[16] B. Rouxel, S. Derrien, and I. Puaut, "Tightening contention delays while scheduling parallel applications on multi-core architectures," ACM Trans. Embed. Comput. Syst., vol.16, no.5s, pp.1–20, 2017.
[17] A. Azim, G. Carvajal, R. Pellizzoni, and S. Fischmeister, "Generation of communication schedules for multi-mode distributed real-time applications," Proceedings of Design, Automation and Test in Europe (DATE), Grenoble, France, pp.1–6, March 2014.
[18] I.I. Cplex, "V12. 1: User's manual for cplex," International Business Machines Corporation, 2019.
[19] M. Fischetti, F. Glover, and A. Lodi, "The feasibility pump," Mathematical Programming, vol.104, no.1, pp.91–104, Sept. 2005.
[20] Gurobi, Gurobi optimizer reference manual, Gurobi Optimization, 2019.
[21] P.K. Harter, Jr., "Response times in level-structured systems," ACM Trans. Comput. Syst., vol.5, no.3, pp.232–248, Aug. 1987.
[22] M. Joseph and P. Pandya, "Finding response times in a real-time system," The Computer Journal, vol.29, no.5, pp.390–395, 1986.
[23] C.L. Liu and J.W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," J. ACM, vol.20, no.1, pp.46–61, Jan. 1973.
[24] P. Balbastre, I. Ripoll, J. Vidal, and A. Crespo, "A task model to reduce control delays," Real-Time Syst., vol.27, no.3, pp.215–236, 2004.
[25] E. Bini and G.C. Buttazzo, "Measuring the performance of schedulability tests," Real-Time Systems, vol.30, pp.129–154, 2005.
[26] V. Brocal, P. Balbastre, R. Ballester, and I. Ripoll, "Task period selection to minimize hyperperiod," ETFA2011, pp.1–4, 2011.

Ana Guasque was born in Valencia, Spain, in 1987. She received a B.S. degree in industrial engineering from the Universitat Politècnica de València (UPV) in 2013; and an M.S. degree in automation and industrial computing from the UPV in 2015. She received a Ph.D. degree in industrial engineering from the UPV in 2019. She is currently working as a researcher in the Universitat Politècnica de València. Her main research interests include real-time operating systems, scheduling, and optimization algorithms and real-time control.

Patricia Balbastre is an associate professor of computer engineering at the Universitat Politècnica de València (UPV). She graduated in electronic engineering at the UPV in 1998 and obtained the Ph.D. degree in computer science in 2002. Her main research interests include real-time operating systems, dynamic scheduling algorithms and real-time control, which has resulted in publications in prestigious journals and conferences in the field.