Appendix of the paper "Towards a Tractable Exact Test for Global Multiprocessor Fixed Priority Scheduling"

Artem Burmyakov (Innopolis University, Russia)
Enrico Bini (University of Turin, Italy)
Chang-Gun Lee (Seoul National University, South Korea)

January 5, 2022

1 Introduction

This appendix describes a runtime efficient implementation of an exact schedulability test introduced in the following publication:

- Artem Burmyakov, Enrico Bini, Chang-Gun Lee, "Towards a Tractable Exact Test for Global Multiprocessor Fixed Priority Scheduling", IEEE Transactions on Computers journal, accepted in 2021, to appear.

2 Efficient implementation of pruning rules

By relying on Theorem 1 in [2], we proposed the following pruning rules in [2]:

Pruning rule 1

$$\begin{aligned} &\forall g' \in G', \\ &\qquad \mathsf{IF} \quad \exists g \in V: \; c_i \geq c'_i \wedge p_i \leq p'_i, \quad \forall i = 1, \ldots, k \\ &\qquad \mathsf{THEN} \quad G' := G' \setminus \{g'\} \end{aligned} \tag{1}$$

Pruning rule 2

$$\begin{aligned} &\forall g \in G', \quad \forall g' \in V \\ &\qquad \mathsf{IF} \quad c'_i \leq c_i \wedge p'_i \geq p_i, \qquad \forall i = 1, \ldots, k \\ &\qquad \mathsf{THEN} \quad V := V \setminus \{g'\} \end{aligned} \tag{2}$$

The time needed to evaluate (1) and (2) is exponential, since it depends on the size of V. Below, we propose an efficient implementation of these rules.

The states in V are arranged in a table $\mathbb{T}_V$. The columns of $\mathbb{T}_V$ correspond to state parameters $p_1, \ldots, p_k$, $c_1, \ldots, c_k$ respectively, and values in columns are sorted increasingly. See an example in Table 2a, for the graph depicted in Fig. 1.

We next provide an example of using $\mathbb{T}_V$ to check (1). Consider $\mathbb{T}_V$ as depicted in Table 2a. Consider state $g' = \{(1,4),(1,3)\}$. We next check the existence of $g \in V$ for a given g', satisfying (1).

Condition (1) requires $p_1 \leq p'_1$. Thus, we recursively check all rows in $\mathbb{T}_V$ with $p_1 \leq 4$, as $p'_1 = 4$ (these rows are shaded in Table 2b). As rows are sorted by increasing p_i, we terminate the search once condition $p_1 \leq 4$ is violated. We then proceed with p_2. Among all rows with $p_1 \leq 4$, we identify those with $p_2 \leq 3$ (as $p'_2 = 3$). We repeat the procedure recursively for c_1 and c_2. Among all rows with $p_1 \leq 4$ and $p_2 \leq 3$, we identify those with $c_1 \geq 1$ (as $c'_1 = 1$), and finally, those with $c_2 \geq 1$ (as $c'_2 = 1$). The resulted rows correspond to states g satisfying condition (1). As no row left in our case (see Table 2b), condition (1) is violated for a given g'.

The runtime gain of using $\mathbb{T}_V$ over an exhaustive search is achieved thanks to states being ordered by increasing p_i and c_i. Once conditions $p_i \leq p'_i$ or $c_i \geq c'_i$ are violated, the remaining rows are pruned.

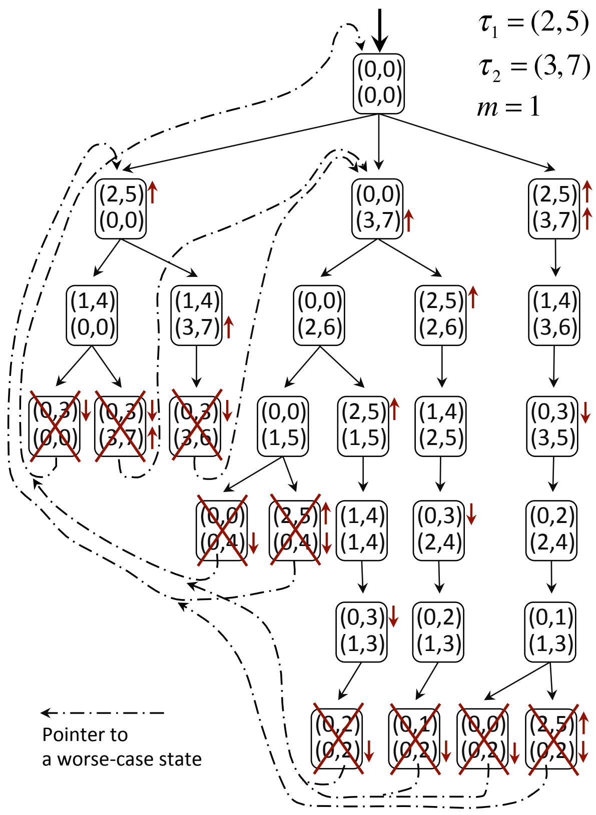

Figure 1: State transition graph, pruned by (1)

p_1	p_2	c_1	c_2
0	0	0	0
	5	0	1
	6	0	2
	7	0	3
1	3	0	1
2	3	0	1
	4	0	2
3	3	0	1
	4	0	2
	5	0	3
4	0	1	0
	4	1	1
	5	1	2
	6	1	3
	7	1	3
5	0	2	0
	5	2	1
	6	2	2
	7	2	3

(a) All states

p_1	p_2	c_1	c_2
0	0	0	0
	5	0	1
	6	0	2
	7	0	3
1	3	0	1
2	3	0	1
	4	0	2
3	3	0	1
	4	0	2
	5	0	3
4	0	1	0
	4	1	1
	5	1	2
	6	1	3
	7	1	3
5	0	2	0
	5	2	1
	6	2	2
	7	2	3

(b) Checking (1)

p_1	p_2	c_1	c_2
0	0	0	0
	5	0	1
	6	0	2
	7	0	3
1	3	0	1
2	3	0	1
	4	0	2
3	3	0	1
	4	0	2
	5	0	3
4	0	1	0
	4	1	1
	5	1	2
	6	1	3
	7	1	3
5	0	2	0
	5	2	1
	6	2	2
	7	2	3

State to be removed

(c) Checking (2)

B^c	B^p	$B^{p,1}$	$\sum_{i=1}^{k} c_i$	$\sum_{i=1}^{k} p_i$	p_1	p_2	c_1	c_2
00	00	00	0	0	0	0	0	0
01	01	01	1	5	0	5	0	1
			2	6	0	6	0	2
			3	7	0	7	0	3
	11	00	1	4	1	3	0	1
				5	2	3	0	1
		01	2	6	2	4	0	2
		10	1	6	3	3	0	1
		11	2	7	3	4	0	2
			3	8	3	5	0	3
10	10	10	1	4	4	0	1	0
			2	5	5	0	2	0
11	11	11	2	8	4	4	1	1
			3	9	4	5	1	2
				10	5	5	2	1
			4	10	4	6	1	3
				11	4	7	1	3
					5	6	2	2
			5	12	5	7	2	3

(d) Extended table; checking (1)

Table 2: Table $\mathbb{T}_V$ for set V, corresponding to the graph in Fig. 1. Examples correspond to $g' = \{(1,4),(1,3)\}$

3 Runtime reduction by using binary search

To speed-up implementation for (1) and (2) further, we extend table $\mathbb{T}_V$, introduced in Section 2, by incorporating additional search keys. Our approach ellaborates the idea of a binary search, and is as follows.

Consider an arbitrary state g. From definition (1) in [2] of a system state, it holds that $c_i \in \{0,\ldots,C_i\}$ and $p_i \in \{0,\ldots,P_i\}$.

For such g, let us compare its values c_i and p_i to the centers of the respective feasibility intervals, that are ${}^{C_i}/_2$ and ${}^{P_i}/_2$, and define boolean parameters b_i^c and b_i^p accordingly:

$$b_i^{\mathsf{x}} = \begin{cases} 1, & \textit{if}\ \mathsf{x}_i > {}^{\mathsf{X}_i}/_2 \\ 0, & \textit{otherwise} \end{cases},$$

where $\mathsf{x} \in \{c, p\}$ and $\mathsf{X} \in \{C, P\}$. Table 2d shows an example, with $B^c = b_1^c \ldots b_n^c$ and $B^p = b_1^p \ldots b_n^p$ computed for each state in the graph from Fig. 1.

By using b_i^c and b_i^p, we optimize the implementation for (1) and (2) as follows. From (1), g and g' must satisfy the following necessary conditions:

$$\begin{cases} b_i^c \ \geq \ b_i^{c'} \\ b_i^p \ \leq \ b_i^{p'} \end{cases}, \qquad i = 1, \ldots, n,$$

with $b_i^{c'}$ and $b_i^{p'}$ corresponding to state g'. Thus, when searching for g satisfying (1), we safely discard all rows in $\mathbb{T}_V$, that violate the conditions above. For example, if checking (1) for $g' = \{(1,4), (1,3)\}$, we examine only those table cells, that are shaded in Table 2d. The same idea applies to (2).

We generalize this approach further. For an arbitrary g, suppose that $b_i^p = 1$, that is $p_i \in ({}^{P_i}/_2, P_i]$. Let us compare p_i to the center of this interval, ${}^{3P_i}/_4$, and let us define boolean parameter $b_i^{p,1} = 1$, if $p_i > {}^{3P_i}/_4$, and $b_i^{p,1} = 0$, if $p_i \leq {}^{3P_i}/_4$. If instead $b_i^p = 0$, that is $p_i \in [0, {}^{P_i}/_2]$, then we compare p_i to the respective center ${}^{P_i}/_4$, and we set $b_i^{p,1} = 0$, if $p_i \leq {}^{P_i}/_4$, and $b_i^{p,1} = 1$, otherwise:

$$\begin{aligned} b_i^{p,1} = 1 \quad &\Leftrightarrow \quad \left[\begin{aligned} &b_i^p = 1 \ \wedge \ p_i > {}^{3P_i}/_4 \\ &b_i^p = 0 \ \wedge \ p_i > {}^{P_i}/_4 \end{aligned}\right., \\ b_i^{p,1} = 0, \quad &\textit{otherwise.} \end{aligned}$$

(The definition above can be extended to parameter c_i as well.) See an example of computing $b_i^{p,1}$ in Table 2d, with $B^{p,1} = b_1^{p,1} \ldots b_n^{p,1}$.

Condition (1) implies the following necessary condition for g and g':

$$b_i^{p'} = 0 \quad \Rightarrow \quad \left(b_i^{p,1} \ \leq \ b_i^{p',1} \right), \tag{3}$$

with $b_i^{p'}$ and $b_i^{p',1}$ corresponding to state g', as well as condition (2) implies that

$$b_i^p = 1 \quad \Rightarrow \quad \left(b_i^{p',1} \ \geq \ b_i^{p,1} \right). \tag{4}$$

Thus, we use the conditions above to prune rows, when traversing $\mathbb{T}_V$.

According to our evaluation, $\mathbb{T}_V$ depicted in Table 2d provides an optimal balance between the runtime reduction and the memory increase, caused by the introduction of redundant data in $\mathbb{T}_V$ (that is columns $B^c, B^p, B^{p,1}$), for the settings defined in Section 5 of [2] . We achieve 3-5 times of runtime reduction at the price of 15-35% memory increase.

4 An optimized order of states traversal

Bonifaci *et al.* [1] examine graph states in the order of increasing state depth (that is the breadth-first traversal [3]). Instead, we examine states in the order of decreasing number of pending jobs at a state. Such an approach yields a much faster schedulability test, because the efficiency of pruning conditions (1) and (2) is maximized in this case[1].

5 Acknowledgement

This research has been financially supported by The Analytical Center for the Government of the Russian Federation (Agreement No. 70-2021-00143 01.11.2021). This work was supported in part by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT).

[1] A refined criteria for the order of states traversal can be found in our code, that is available at `https://github.com/burmyakov/exact-sched-test-gfp.git`

References

[1] Vincenzo Bonifaci and Alberto Marchetti-Spaccamela. Feasibility analysis of sporadic real-time multiprocessor task systems. *Algorithmica*, 2012.

[2] Artem Burmyakov, Enrico Bini, and Chang-Gun Lee. Towards a tractable exact test for global multiprocessor fixed priority scheduling. *IEEE Transactions on Computers*, accepted in 2021, to appear.

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 3 edition, 2009.