# A General Scheduling Framework for Multi-objective Real-time Systems

Sen Wang[1], Ashrarul Haq Sifat[1], Xuanliang Deng[1], Shao-Yu Huang[2], Changhee Jung[2], Jia-Bin Huang[3], Ryan Williams[1], and Haibo Zeng[1]

[1]Virginia Polytechnic Institute and State University
[2]Purdue University
[3]The University of Maryland, College Park

***Abstract*—We regard the industry challenge as a multi-objective real-time scheduling problem. Different from traditional methods, we model it as a nonlinear program (NLP) and use gradient-based methods for efficient solutions. Different requirements can be freely added as constraints, which gives the proposed method effective for a large range of problems. Furthermore, the proposed NLP scheduling method can utilize available scheduling algorithm as initialization method, and improve their performance even further. Preliminary evaluation shows advantages against simple heuristic methods in complicated problems.**

## I. Introduction

In this brief paper, based on the scheduling problem posed by RTSS 2021 industry challenge [1], we model it as a multi-objective scheduling problem and propose a general scheduling method. Specifically, given a computing system and multiple scheduling requirements, our method returns an efficient scheduling algorithm that satisfies all the requirements at the worst cost of pseudo-polynomial complexity. We envision that our approach can address the scheduling problem in a broad range of real-time systems. In the following, we use the original industry challenge as an example to illustrate our methods. Necessary notations are introduced in this section.

We consider a single directed acyclic graph (DAG) model $\mathcal{G} = (V, E)$ to describe computation works. Each node $v_i$ has their own period $T_i$, worst-case execution time $c_i$, relative deadline $d_i$. An edge $E_{ij}$, $(v_i, v_j)$, goes from node $v_i$ to node $V_j$ means $v_j$'s input depends on $v_i$'s output. The overall DAG graph $\mathcal{G}$ is not necessarily fully connected. The hyper-period, the least common multiple of periods for all nodes in $\mathcal{G}$, is denoted as $H$. Within a hyper-period, the k-th instance of node $v_i$ starts execution at time $s_{ki}$ *non-preemptively* (required by the industry challenge, but may be relaxed in our method), and finishes at $f_{ki} = s_{ki} + c_i$. For a node $v_i$ with precedence constraints, we denote all its source tasks as $pre(v_i)$, and all its successor as $suc(v_i)$.

A path in DAG is described by a node sequence $\lambda_{be} = \{v_b, ..., v_e\}$, which starts at node $v_b$ and ends at node $v_e$, and is connected in sequence, i.e., $(v_i, v_{i+1}) \in E$.

Any questions about this paper can be sent to Sen Wang: swang666@vt.edu, Haibo Zeng: hbzeng@vt.edu

There are multiple requirements posed to the scheduling system:

- **Schedulability**. All the nodes should be schedulable, i.e.,

$$r_i \leq d_i \tag{1}$$

  In this paper, we consider implicit deadline for simplicity, i.e., $d_i = T_i$.
- **Computation resource bounds**. All the computation resources $R_i$ (e.g., CPU, GPU) are not overloaded for any time interval from $t_i$ to $t_j$. The mathematical description is given by demand bound function (DBF) [2] as follows

$$\forall R_i, \text{DBF}(t_i, t_j) \leq t_j - t_i \tag{2}$$

  We will give a more detailed description for this requirement in the methodology section.
- **Sensor bound**. If we use $v_i$ to denote a node under consideration, and $\{v_l | v_l \in pre(v_i)\}$ represents all its predecessor nodes, then for any instance $v_{ik}$ with start time at $s_{ik}$, the time difference of all the source data tokens $v_{lj}$ must be smaller than $\Theta_s$.

$$\forall s_{ik}, \forall s_{lj} \in \{0 \leq l \leq \frac{H}{T_l} | s_{lj} \leq s_{ik} < s_{l(j+1)}\}: \tag{3}$$

$$\max_l s_{lj} - \min_l s_{lj} \leq \Theta_s \tag{4}$$

- **Event chain**. For all the event chains $\lambda_{be}$, the response time from its start at $s_{bi}$ to its end at $s_{ej}$ should be bounded:

$$\forall \in \lambda_{be}, s_{ej} + c_e - s_{bi} \leq \Theta_e \tag{5}$$

  where the instances of $v_{ik} \in \lambda_{be}$ are matched by the following constraints:

$$\forall b \leq i \leq e,\ s_{(i-1)l} \leq s_{ik} \leq s_{(i-1)(l+1)} \tag{6}$$

- **Period constraints**. All the instances of each task cannot start earlier than the beginning of their periods.

$$\forall i, k,\ s_{ik} \geq T_i(k-1) \tag{7}$$

- **DAG dependency**. Each node cannot start until all its dependency tasks have finished.

$$\forall v_i, \forall l \in pre(v_i), s_{l0} + c_l \leq s_{i0} \tag{8}$$

Unlike traditional DAG systems where all the nodes have same frequency, in our model where different nodes have different frequency, over-sampling or under-sampling [3] would be unavoidable.

## II. Methodology

We model the scheduling problem as an optimization problem, and propose to use gradient-based methods to solve it for efficiency. The main framework is shown in Fig. 1.

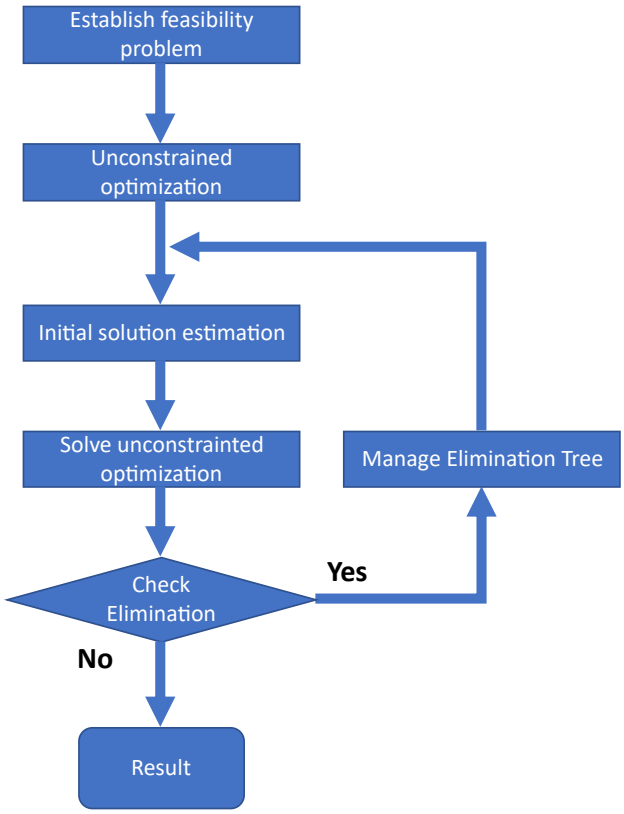


Figure 1: Main optimization framework

### A. Schedulability analysis

We propose a new method to analyze schedulability for non-preemptive situations based on the ILP formulation proposed by Baruah [4]. The proposed algorithm is proved to yield the same results as [4], but with worst computation cost of only $O(N^2)$ (in average $O(N\log(N))$) as opposed to $O(N^3)$ in [4]. Formal description and proof are skipped because of word limits, but the basic ideas are described as follows:

In non-preemptive situations, an equivalent schedulability analysis can be obtained by calculating "interval" overlapping, where an "interval" is defined as a task instance's execution interval. For example, job $v_{ik}$'s execution interval starts at $s_{ik}$, and ends at $s_{ik} + c_i$. If all the intervals are not overlapping with each other, then the system is schedulable.

### B. Optimization problem formulation

In this paper, we decide to formulate the scheduling problem as an optimization problem. Inspired from operation research [4], the variables are the start time of all the node instances in the DAG graph $\mathcal{G}$ within a hyper-period. Such modeling method is very flexible and can easily describe all the constraints without pessimistic assumptions. The constraints are described in the previous section, as specified by Equation eqs. (1) to (8). The optimization problem that we are considering is general, in the sense that different constraints can be added or removed freely if only they can be expressed mathematically.

The objective of the optimization is to find a set of start variables for each task instance such that all the constraints are satisfied.

### C. Approximated unconstrained optimization

Depending on task periods, the optimization problem proposed could have a large number of variables and constraints. As such, integer linear programming proposed in Baruah's formulation [4] is not appropriate in our case. However, efficient algorithms exist by noticing that most constraints as described above are only concerned with a few variables. Such sparsity is well exploited in many robotics and optimization problems such as simulated localization and mapping (SLAM) [5], [6], motion planning [7], where thousands or even millions of variables are optimized together with fast speed.

To exploit such sparsity with nonlinear optimizer, we transform the feasibility problem above into an unconstrained optimization problem with barrier function [8]. Given a constraint such as

$$f(x) \leq 0 \tag{9}$$

It is transformed into an objective function

$$\text{Barrier}(f(x)) = \begin{cases} 0, & f(x) \leq 0 \\ g(x), & \text{otherwise} \end{cases} \tag{10}$$

where the $g(x) > 0$ is a punishment function for violated constraints.

Since most constraints are linear with respect to their variables, the punishment function is usually straightforward to design. For example, the schedulability constraints

$$r_i(\mathbf{s}) - d_i \leq 0 \tag{11}$$

is transformed to

$$\text{Barrier}(r_i(\mathbf{s}) - d_i) = \begin{cases} 0, & r_i(\mathbf{s}) - d_i \leq 0 \\ d_i - r_i(\mathbf{s}), & \text{otherwise} \end{cases} \tag{12}$$

After transforming constraints into objective function with the barrier method, we formulate a least-square minimization problem as follows:

$$\min_{\mathbf{s}} \sum_{m=1}^{M} \text{Barrier}^2(f_i(\mathbf{s})) \tag{13}$$

Since the Barrier function always gives a positive error if some constraints are violated, objective function 13 establishes a

*necessary and sufficient* condition for schedulability analysis: **s** is schedulable if and only if:

$$\sum_{m=1}^{M} \text{Barrier}^2(f_i(\mathbf{s})) = \mathbf{0} \tag{14}$$

### D. Gradient-based optimization method

After formulating an optimization problem, we use gradient-based trust-region methods [7], [9] to solve it.

*1) Numerical Jacobian evaluation:* Since many constraints are not differentiable, numerical methods are used to estimate Jacobian matrix in these situations:

$$\frac{\partial f}{\partial x_i} = \frac{f(x_1, ..., x_i + h, ..., x_N) - f(x_1, ..., x_i - h, ..., x_N)}{2h} \tag{15}$$

The $h$ parameter above should be *reasonably* small to properly estimate Jacobian matrix. Although numerical Jacobian is simple, analytic Jacobian should always be preferred whenever possible because it would save lots of computation cost.

*2) Vanishing gradient problem:* During optimization, some points have 0 gradient with non-zero error, as shown in Fig. 2. To handle this issue, a simple idea would be increasing the granularity, i.e., $h$ in Equation 15, until the gradient is not zero if the interval overlap error is not 0. More effective ideas such as random walk will be exploited in the future.

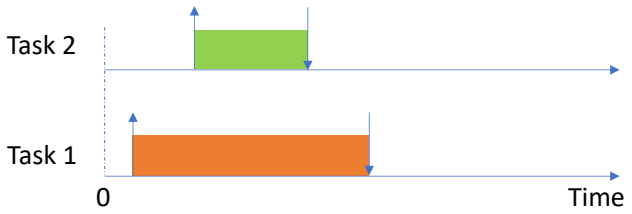


Figure 2: When task 1's *interval*(see section II-A) fully covers task 2's interval, gradient for both the start time of task 1 and task 2 would become 0, even though the system is not schedulable.

### E. Managing elimination forest

In this section, we propose a new algorithm named *elimination forest* to bring more efficient and effective optimization. Let's consider an unconstrained optimization problem:

$$\min_x f(x) \tag{16}$$

where $f(x) : R^n \to R^m$ is a discrete function. Furthermore, we assume this optimization problem is solved by gradient-based methods as mentioned above. Two definitions are given there for the ease of presentation.

**Definition II.1** (Local optimal point)**.** *A point $x_0$ is a local optimal point if*

$$\forall \Delta \in \{R^n | \ |\Delta| = 1\}, \ \delta \to 0 :$$

$$f(x_0) \leq f(x_0 + \Delta\delta) \tag{17}$$

**Definition II.2** (Pseudo-local optimal point (PSOP))**.** *A point $x_0$ for an objective function $f(x)$ is called pseudo-local optimal point if*

- *It is judged as a local optimal point from Jacobian $J$, Hessian $H$ or their variants:*

$$\delta \to 0 : f(x_0) < f(x_0 + \Delta(J, H)) \tag{18}$$

  *where $\Delta$ is given by the common gradient based methods such as steepest descent, Gauss-Newton, Levenberg-Marquardt (LM) [10], [11], Dogleg(DL) [12], etc.*
- *It is not a local optimal point, i.e.,*

$$\exists \Delta \in \{R^n | \ |\Delta| = 1\}, \ \delta \to 0 :$$

$$f(x_0 + \Delta\delta) \leq f(x_0) \tag{19}$$

*1) Leaving PSOC by elimination:* In our experience, it is very common that the optimization algorithm is stuck at a local optimal or a pseudo-local optimal point. Two possible reasons are summarized:

- Numerical Jacobian at a discrete point is not estimated appropriately such that the Jacobian matrix becomes very large towards one direction. As a result, optimizer tends to go along the opposite direction, which may be misleading.
- Some constraints are very tight such that slightly adjusting some variables to optimize the objective function in one direction will cause other constraints to be violated.

Both two situations suggest that we may be able to leave this situation if we can leave some dimension of objective function out of consideration. For example, if the optimization algorithm terminates at some point $\mathbf{x}_0$ and $\mathbf{f}(\mathbf{x}_0) > 0$, we would go through each dimension $\mathbf{f}_i(\mathbf{x})_0$, if we find that

$$f_i(\mathbf{x}_0) = \theta \tag{20}$$

where $\theta \geq 0$ is smaller than a very small pre-defined threshold, then we stop optimizing toward this criteria, and eliminate this dimension from objective function. To keep variables remain at $f_i(\mathbf{x}_0)$ in the future, we transform it into a constraint added to the variables in such a form

$$x_k = g(\mathbf{x}_0, \theta) \tag{21}$$

In other words, $x_k$ depends on other variables and can be derived from Equation 21. In the following optimization loops, $x_k$ will always be replaced with its dependency variables based on Equation 21.

*2) Building elimination forest:* As the iteration loop continues, more and more variables may be eliminated and could bring confliction. Inspired from Bayes tree proposed by Kaess *et al.* [13], we propose elimination forest, an efficient algorithm for managing elimination.

Building an elimination forest is simple. First create a node (root for a tree) for each variables. Each time a variable is eliminated, add dependency edges from the eliminated variables to the dependency variables. Since all the nodes in a tree have fixed relative value (given by equation 21), adding a

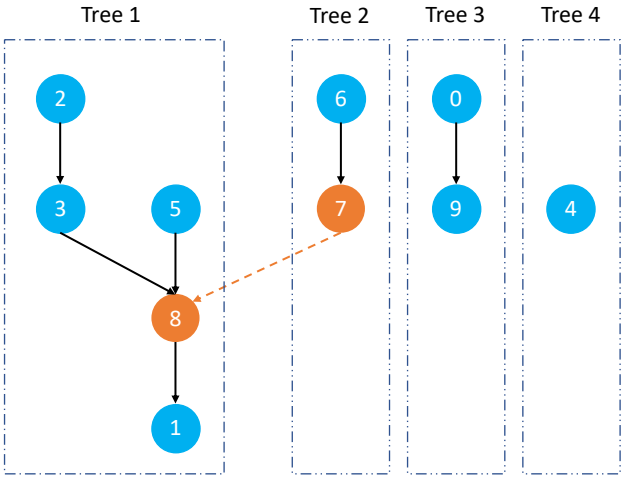


Figure 3: Managing elimination forest: After adding an edge from node 7 to node 8, all the nodes in tree 1 and 2 have fixed relative value. Careful attention must be paid to prevent related nodes, such as node 6 and node 2, violate some constraints.

new dependency may indirectly make some nodes in those two trees violate some constraints. In that case, confliction check must be performed to all the nodes in the two related trees. An example is given in Fig. 3.

Building elimination forest is also helpful in improving algorithm efficiency because all the nodes inside an elimination tree are already checked to be confliction-free. When two nodes are considered to be added together, we only need to check confliction between the two trees. It will save more time and effort when performing optimization with respect to a large number of variables.

### *F. Initial solution estimation*

Flexible initialization policy makes NLP compatible with most available scheduling algorithms to achieve further performance boost. If only one scheduling algorithm could satisfy parts of requirements, then it can be used as initialization and the proposed algorithm is expected to work better. For example, Rate Monotonic is used in our experiments.

## III. Preliminary Results

Because of time limits, we only finished partial, slow implementation of the proposed ideas. The algorithm is tested in 500 random task sets, whose periods are limited to a small range $\{100, 200, 300, 400, 500\}$ for fast evaluation. The results are summarized in Table tables I to V. The first row reports initialization method's performance, second row reports baseline optimization method simulated annealing (SA), third row are the proposed NLP algorithm. All the experiments within one table have the same initialization, and all the experiments are evaluated based on the same task sets.

Several preliminary conclusions can be drawn from these experiments:

**Algorithm 1:** Managing elimination forest inside one iteration

**Input:** variables **x** after performing unconstrained optimization, elimination graph $G$
**Output:** elimination graph $G$

**if** $G$ *is empty* **then**
  Add **x** as independent nodes to $G$
**else**
**end**
// iterate over objective function
**for** $(i = 0;\ i < m;\ i = i + 1)$ **do**
  **if** $f_i(\boldsymbol{x}) \leq \theta$ **then**
    $x_0 =$ Eliminated variable
    $X =$ Dependency variables
    Trees = {}
    **for** $(x_j \in \{x_0, X\})$ **do**
      Trees.add( *ExtractSubGraph*($G$, $x_j$))
    **end**
    **if** *CheckConfliction*(*Trees*) **then**
      $G$. *AddEdge*$(x_0, X)$
    **end**
  **end**
**end**
**return** G

- The proposed algorithm improves around $50\%$ to $1000\%$ acceptance rate in the experiments, which proves the validity and effectiveness of the algorithm.
- The proposed algorithm can easily handle systems with different constraints and requirements.
- Although relying on a reasonable initialization method, the proposed algorithm does not require a very good initialization, which is critical in practice.
- It takes a longer time than the baseline methods, which we believe is mostly an implementation issue and can be improved.

In the near future, we will perform more extensive experimental evaluation on larger random task sets. Another major goal is to look at ways to improve the runtime efficiency of the proposed methods.

## References

[1] PerceptIn, "2021 rtss industry challenge." http://2021.rtss.org/industry-session/, 2021.
[2] S. Baruah, M. Bertogna, and G. Buttazzo, "Multiprocessor scheduling for real-time systems," in *Embedded Systems*, 2015.
[3] J. Abdullah, G. Dai, and W. Yi, "Worst-case cause-effect reaction latency in systems with non-blocking communication," *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1625–1630, 2019.
[4] S. Baruah, "Scheduling dags when processor assignments are specified," *Proceedings of the 28th International Conference on Real-Time Networks and Systems*, 2020.
[5] F. Dellaert and M. Kaess, "Factor graphs for robot perception," *Found. Trends Robotics*, vol. 6, pp. 1–139, 2017.
[6] F. Dellaert and M. Kaess, "Square root sam: Simultaneous localization and mapping via square root information smoothing," *The International Journal of Robotics Research*, vol. 25, pp. 1181 – 1203, 2006.

Table I: Random task sets subject to DBF, DAG constraints, single processor

| Algorithm | Accept rate (Error<1) | Accept rate (Error<0.1) | Average Initial Error (time units) | Average Optimized Error (time units) | Average time (seconds) |
|---|---|---|---|---|---|
| RM&DAG_Initialize | 54.4% | 54.4% | 54.87 | - | 2e-4 |
| SA_RM_Optimize | 54.4% | 54.4% | 54.87 | 54.87 | 0.446 |
| **NLP_RM_Optimize** | 72% | 66.6% | 54.87 | 3.43 | 0.165 |

Table II: Random task sets subject to DBF, DAG, SensorBound constraints, single processor

| Algorithm | Accept rate (Error<1) | Accept rate (Error<0.1) | Average Initial Error (time units) | Average Optimized Error (time units) | Average time (seconds) |
|---|---|---|---|---|---|
| RM&DAG_Initialize | 31.4% | 31.4% | 158.29 | - | 2e-4 |
| SA_RM_Optimize | 31.4% | 31.4% | 158.29 | 158.29 | 0.443 |
| **NLP_RM_Optimize** | 58% | 53.6% | 158.29 | 31.12 | 1.58 |

Table III: Random task sets subject to DBF, DAG, SensorBound, two processors

| Algorithm | Accept rate (Error<1) | Accept rate (Error<0.1) | Average Initial Error (time units) | Average Optimized Error (time units) | Average time (seconds) |
|---|---|---|---|---|---|
| RM&DAG_Initialize | 40.2% | 38.4% | 115.04 | - | 2e-4 |
| SA_RM_Optimize | 40.2% | 38.4% | 115.04 | 115.04 | 0.471 |
| **NLP_RM_Optimize** | 66.2% | 63.4% | 115.04 | 22.28 | 1.65 |

Table IV: Random task sets subject to DBF, DAG constraints, two processors

| Algorithm | Accept rate (Error<1) | Accept rate (Error<0.1) | Average Initial Error (time units) | Average Optimized Error (time units) | Average time (seconds) |
|---|---|---|---|---|---|
| RM_Initialize | 7.8% | 7.8% | 91.45 | - | 3e-4 |
| SA_RM_Optimize | 7.8% | 7.8% | 91.45 | 91.45 | 0.453 |
| **NLP_RM_Optimize** | **81.6%** | 78.8% | 91.45 | 2.63 | 0.22 |

Table V: Random task sets subject to DBF, DAG, SensorBound, two processors

| Algorithm | Accept rate (Error<1) | Accept rate (Error<0.1) | Average Initial Error (time units) | Average Optimized Error (time units) | Average time (seconds) |
|---|---|---|---|---|---|
| RM_Initialize | 6.8% | 6.8% | 168.94 | - | 3e-4 |
| SA_RM_Optimize | 6.8% | 6.8% | 168.94 | 168.94 | 0.451 |
| **NLP_RM_Optimize** | **66.2%** | 64.8% | 168.94 | 24.50 | 2.13 |


[7] M. Mukadam, J. Dong, X. Yan, F. Dellaert, and B. Boots, "Continuous-time gaussian process motion planning via probabilistic inference," *The International Journal of Robotics Research*, vol. 37, pp. 1319 – 1340, 2018.

[8] S. P. Boyd and L. Vandenberghe, "Convex optimization," *IEEE Transactions on Automatic Control*, vol. 51, pp. 1859–1859, 2006.

[9] S. Wang, J. Chen, X. Deng, S. Hutchinson, and F. Dellaert, "Robot calligraphy using pseudospectral optimal control in conjunction with a novel dynamic brush model," *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6696–6703, 2020.

[10] K. Levenberg, "A method for the solution of certain non – linear problems in least squares," *Quarterly of Applied Mathematics*, vol. 2, pp. 164–168, 1944.

[11] D. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of The Society for Industrial and Applied Mathematics*, vol. 11, pp. 431–441, 1963.

[12] M. Powell, "A new algorithm for unconstrained optimization," 1970.

[13] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert, "isam2: Incremental smoothing and mapping using the bayes tree," *The International Journal of Robotics Research*, vol. 31, pp. 216 – 235, 2012.