Danmarks
Tekniske
Universitet

DTU

System Optimization Course Project

Group 20

Shuoqiang Zeng - s210357
Jiaxuan Wu - s212625
Peichen Liu - s222475
Yingli Duan - s222462
Yuansheng Zhao - s222425

December 18, 2023

Contents

1 Introduction **2**
1.1 Problem Introduction . . . 2
1.2 Report structure . . . 2

2 Problem Analysis **3**
2.1 Problem Formulation . . . 3
2.2 Mathematical Model . . . 3

3 Simulation Optimization **6**
3.1 Top-down overview . . . 6
3.2 Optimization with GA . . . 8
3.2.1 Time complexity and feasibility . . . 8
3.2.2 Generic algorithm . . . 8
3.3 Simulation Model . . . 9
3.3.1 Event trigger tasks schedule simulation using Explicit Deadline Periodic(EDP) . . . 9
3.3.2 EDF simulation for determining the schedule of TT tasks . . . 11
3.4 Correctness Test . . . 12
3.4.1 Implement . . . 12
3.4.2 Result . . . 14

4 Experiment **16**
4.1 Solution to the original problem . . . 16
4.2 Speedup by using Processor Demand Criterion . . . 17
4.2.1 Implementation . . . 17
4.2.2 Result . . . 19
4.3 Optimize WCRT by re-assigning priorities of ET tasks . . . 20
4.4 Speed up with External Point Method . . . 22
4.4.1 Introduction . . . 22
4.4.2 Implement . . . 23
4.4.3 Discussion . . . 24
4.5 Take criticality tasks into consideration . . . 24
4.5.1 Introduction . . . 24
4.5.2 Implementation . . . 24
4.5.3 Result . . . 25

5 Conclusion **27**

A Appendix **29**
A.1 Work division and contribution . . . 29

1 Introduction

1.1 Problem Introduction

The scheduling of system tasks in the current embedded or on-board system development is usually divided into two types: time-triggered(TT) tasks and event-triggered(ET) tasks. For systems that are linked to human safety, such as Advanced Driver Assistance Systems(ADAS) or aircraft flight control systems, it is critical to ensure each task completed within the specified time (which called the WCRT of the tasks).
The goal of our optimization project is to make an optimization algorithm that can schedule all tasks to fire and meet the processing time they needed within their specified WCRT time. Our project is based on MATLAB, which is a commercial software produced by The MathWorks Co., provides cross-platform developing environment and meets our requirements. We used the documents and datasets of tasks provided by this course(02229) as a reference for our project, and used the additional optimization method provided in the paper(which is labeled as extensions) to optimize our project so that we can find out the best solutions. In addition, we shall appreciate all the helps from our professor Paul Pop and teaching assistants. We also use open-source optimization method which called GA provided in Matlab to look for further optimal solutions. We also take a huge amount of attempts of tests to adjust the parameters so that we can achieve the targets.

1.2 Report structure

We start by defining the problem as an optimization model in Section 2 followed by the description and analysis of our optimization algorithm in Section 3. Then we introduce the experiments and evaluation towards the original problem and its extension problems in Section 4 followed by the conclusion in Section 5. In Appendix A, we list detailed work division and contributions of each group member.

2 Problem Analysis

2.1 Problem Formulation

The combinatorial optimization problem can be formulated as follows. As an input you are given: (1) An application model consisting of a set of TT tasks and a set of ET tasks. TT tasks are periodic. *Periodic tasks* consist of an infinite sequence of identical activities, called instances or *jobs*, that are regularly activated with a constant period. For each periodic TT task we know its period, worst-case execution time (WCET) and deadline. ET tasks are *sporadic*, i.e., consecutive jobs are separated by a minimum inter-arrival time, which can be seen as a "minimum period". For each ET task we know the minimum inter-arrival time, WCET and deadline, as well as its priority. (2) An architecture model consisting of one core that schedules TT tasks with *timeline scheduling* and ET tasks with *polling servers*, which are themselves TT tasks. All tasks are considered *preemptable*, i.e., the schedule can be constructed such that they are stopped to allowed another task to run, and they are restarted afterwards to complete.

Targets: Design and implement an optimization algorithm that determines an optimized solution which consists of the following: (i) The number of polling servers, which then become extra TT tasks. (ii) For each polling server (task), the period, budget, and deadline, as well as (iii) which sub-sets of ET tasks are handled within the respective polling servers. Moreover, the TT tasks also need to be schedulable, i.e., given the found polling tasks (and their parameters), it shall be possible to find (iv) a TT schedule such that also the TT tasks are schedulable. (v) Optimization objective: the average *worst-case response times* (WCRT) of all tasks (TT and ET) is minimized. The WCRT is defined as the time it takes for a task to complete once it has been "released" (i.e., also known as its "arrival time", when it is ready for execution), considering potential interruptions by other tasks.

Requirements: The solution should be optimized such that:(a) both the TT and ET tasks are schedulable, i.e., they complete before their deadlines; (b) the ET task separation constraints are satisfied

2.2 Mathematical Model

When we consider the aforementioned problem and build an optimization model, we need to identify its decision variables, constraints and objective function.

First, the decision variables of this problem are composite, because we need to decide the number of polling servers n, attributes of each polling server A_i, and which the server each TT task is placed S_i. The attributes of each polling server include duration, deadline and period.

$$A_i = C_i, D_i, T_i \tag{1}$$

Where C_i, D_i and T_i are the computation duration, deadline and period of i^{th} polling server respectively. In the following paper, we denote these 3 elements as a set of the alternatives of a polling server, because we need 3 elements to define a polling server together.

Considering a case with n polling servers and m TT tasks, we organize all needed variables as a solution vector as follows.

$$\mathbf{x} = [n, \{C_1, D_1, T_1\}, ..., \{C_i, D_i, T_i\}, ...\{C_n, D_n, T_n\}, S_1, S_2, ..., S_j, ..., S_m] \tag{2}$$

Where n is the number of polling servers, and S_j is the label of polling server where j^{th} ET task is placed in this solution. There are n sets of alternatives in the solution vector to define all polling servers. Similarly, there are m elements to assign all ET tasks in polling servers. Note that n is redundant to this mathematical model. Thus, this decision vector is complete to be interpreted as a defined solution to the problem.

The objective function of this problem is to minimize the average of worst-case running time of all time-trigger tasks and event-trigger tasks. However, it is hard to directly map a solution vector to the corresponding worst-case running time with a closed formula due to the nonlinearity of the problem. Here, we use an abstract function *f(**x**)* to represent the average worst-case running time corresponding to solution ***x***, and we will introduce in detail the simulation models for solution validation in section 2.

If a solution vector is feasible to this problem, it should meet some constraints. These constraints can be both from our definition of decision variables and from the domain knowledge of this problem.

$$\textbf{Min.} f(\boldsymbol{x})$$

$$m \geq n \geq 1 \tag{3a}$$

$$T_i \geq D_i \geq C_i \tag{3b}$$

$$T_i \geq 2 \tag{3c}$$

$$n \geq S_i \geq 1 \tag{3d}$$

$$n, D_i, T_i, C_i, S_i \in N^+ \tag{3e}$$

The constraint 3a limits the search range of n, the number of polling servers; here we set m as the upper bound of n because empty polling servers cannot make the solution better. Constraints 3b and 3c are the basic requirements of polling servers schedulability. Constraints 3d sets that all ET tasks should be deployed in assigned polling servers. Constraints 3e are the integer constraints. The integer constraints also apply to alternatives of polling servers because one time-unit in simulation is pretty short and the simulation time can always be treated as an integer number of time units.
We also need to concern the schedulability of ET tasks, TT tasks and polling servers. However, it is hard to give closed formulas here. We use penalization to avoid that infeasible solution become the best solution. To be specific, when the simulations, which we use to

validate solutions, finds a solution is not feasible, we penalize the objective value, and the penalty value is big enough, so the infeasible solution cannot be the optimal.

3 Simulation Optimization

3.1 Top-down overview

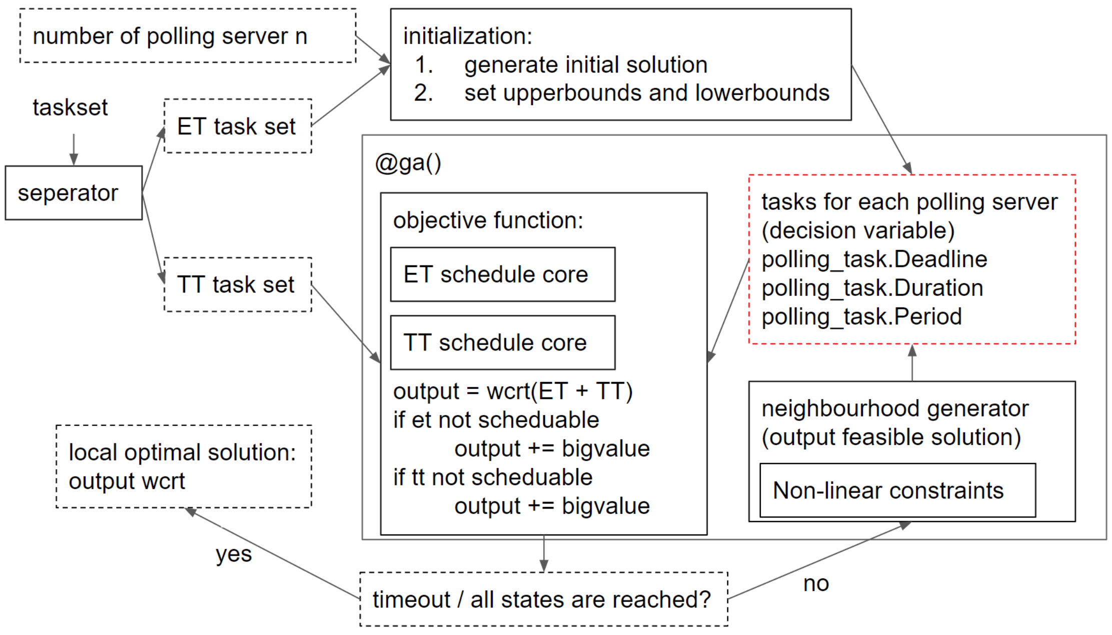


Figure 1: Top-down overview of our optimization algorithm

The structure of our optimization algorithm is shown in Fig 1. We design our optimization algorithm in this way mainly because we want to use the **ga** build-in function in Global Optimization Toolbox of Matlab [1].

First, an input taskset is separated into **ET taskset** and **TT taskset** by our **separator**. Then we generate initial solution according to input polling server number **n** and **ET taskset**. Next, we set upperbounds and lowerbounds of the mutation for possible solutions.

Then the initial solution is given to the **objective function** within **ga()** built-in function. **Objective function** will execute **ET schedule core**(Algorithm 3) and **TT schedule core**(Algorithm 4) sequentially and return the WCRT of all ET and TT tasks, which is the **objective value**. If ET or TT tasks are not scheduable, we will give add a big penalty to the returned **objective value**. The output objective value will be given to the **neighbourhood generator** in **ga()**, and generate feasible neighbourhoods that meet our **non-linear constraints**[3]. Finally, this process will loop until the generation cap and output found optimal **WCRT**.

Concrete pseudo code of the **main function** (top-down overview) and **objective function** is shown in Algorithm 1 and 2 respectively:

Algorithm 1: Pseudo code of main function

```
Data: folder $F$, number of polling server $N_p$, Max Generation $Max\_Gen$, population size
      $Pop\_Size$
Result: Found optimal solution $x$
/* Read the first data set under folder F */
1  $TT, ET \leftarrow convertor\{separator\{F\}, 1\}$;
/* Compute lcm and factors */
2  $T \leftarrow lcm\{T_i \mid \forall \tau_i \in \mathcal{T}^{ET}\}$;
3  $factors \leftarrow sort\{all\ factors\ of\ T \cup [1, 20]\}$;
/* Initialize GA */
4  Set random seed ;
5  $length\ of\ solution \leftarrow 1 + number\ of\ TT\ tasks + 3 * N_p$;
6  Set upper bound and lower bound for solutions;
/* Execute GA */
7  $x \leftarrow ga\{@objective\ function,\ length\ of\ solution,\ upper\ bound,\ lower\ bound,$
8          non-linear constraints, Max_Gen, Pop_Size$\}$
```

Algorithm 2: Pseudo code of objective function

```
Data: solution vector $sol$, penalty for infeasible solution $penalty$
Result: objective value $val$
/* get data from solution vector */
1  $N_p \leftarrow$get number of polling server from the length of $sol$;
2  $PS \leftarrow$ get polling server attributes from $sol$;
3  $val \leftarrow 0$;
4  $infeasible \leftarrow 0$;
/* get wcrt of ET tasks */
5  for $i = 1$ to $N_p$ do
6      if et tasks is not empty then
7          $[WCRT_{ET},\ result] \leftarrow ET\_schedule\_core\{PS_i,\ all\ et\ tasks\ in\ PS_i\}$;
8          $val \leftarrow val + WCRT_{ET}$;
9          if result is infeasible then
10             $infeasible \leftarrow 1$;
/* get wcrt of TT tasks */
11 $TT \leftarrow TT \cup PS$;
12 $[WCRT_{TT},\ result] \leftarrow TT\_schedule\_core\{TT\}$;
13 if result is infeasible then
14     infeasible $\leftarrow$ 1;
15 $val \leftarrow val + sum\{WCRT_{TT}\}$;
16 $val \leftarrow \frac{val}{number\ of\ ET\ and\ TT\ tasks} + infeasible * penalty$;
```

3.2 Optimization with GA

3.2.1 Time complexity and feasibility

Configuring in ADAS is an recurring topic[2]. As an optimization problem, we focus 2 aspects: (1) validation of solutions; (2) finding the optimal solution.

In this problem, pending tasks consists event trigger tasks and time trigger tasks, and we need to configure both kinds. Almeida Pedreiras [3] utilizes a simulation model to validate the running schedule of event trigger tasks in CPU. As for time trigger tasks, Sinnen[4] and Buttazzo[5] introduces an intuitive but efficient scheduling algorithm, named early deadline first (EDF) simulation. In this study, we refer these two algorithms to schedule tasks and validate the performance of schedule tables.

In the second methodological aspect is finding the optimal solution. Since we already define the optimization problem in the previous subsection and the search space of solution vectors, we can calculate the complexity of the enumeration algorithm of our optimization problem, which is

$$O(n_p, n_{ET}, D_T) = n_p^{nET+1} * D_T^{3*n_p}$$

where n_p is the polling server, n_{ET} is the number of input et tasks, and D_T is the feasible domain of periods.

According to [2], the scheduling problem for modern ADAS platforms cannot be solved efficiently by a polynomial-time algorithm and for such intractable problems, researchers have proposed the use of problem-specific heuristics and metaheuristics [6], as an alternative to exact optimization methods which have exponential running times. Thus we consider the problem as an NP-hard problem.

3.2.2 Generic algorithm

In metaheuristics, genetic algorithm (GA) has its solid advantages for large scale optimization problem. First, when the population is big, GA is robust and guarantees a good enough solution. Second, when it is given a good crossover, GA is efficient. For large scale problem, GA also suits for prallelization, shortening the runtime. Due to these advantages, we think GA is suitable to our problem.

When we use GA to solve this problem, We need to set some parameters, helping the algorithm to work effectively and get a good result.

Matlab provides build-in functions to help us setup the Genetic Algorithm, the parameter what we are using is in the following:
CrossoverFcn:{'crossoverscattered'}, which refers to using the default function in GA
MutationFcn:{'mutationgaussian'}, which refers to use GA without constraints

SelectionFcn:{'selectionstochunif'}, which refers to use default setting building in GA
Population:2000, which is an important number which we chooses due to numbers of tests we have done.
MaxGenerations:{100*n},(n is nature numbers), which refers the standard setting of GA, the multi-generation is not changing and shows the best solution we founded remains in the end.

What's more, we hope our results to be reproducible. Thus we utilize rng (Random Seed), a random seed generator in Matlab, to specify the stochastic in the algorithm.

We also need to set the initial solution of GA. We hope the initial solution is feasible to this problem, although we can validate but refuse (penalize) the infeasible solution. Three attributes of the first polling server, including computation, period and deadline, is set to 100, 200, 200 respectively, and all ET tasks are assigned in the first polling server. As for solutions with several polling servers, we set their attributes to 1,10000,10000, so they will have only slight influence on the EDF scheduling simulation; Note that no ET tasks are assigned in these polling servers, so ET tasks are always schedulable.

The penalty value to infeasible solutions is 10000. The value is to penalize the unschedulable solutions. 10,000 is big enough, considering the scale of objective values, so that these solutions cannot be the optimal. Note that a feasible solution should meet several constraints as mentioned in section 2.2. However, only unschedulable solutions are penalized, while other constraints are limited by linear or nonlinear closed expression (inequality) constraints, and our algorithm will not search and validate solutions breaking these closed expression. The main advantages of this approach is minimizing the search space. In the limited search space, GA is more likely to generate feasible solutions, and its efficiency is improved.

A solution of this model can consist of several polling servers. However, predictably, the search space scale will dramatically increase with the increasing of polling server number. Thus, we use the climbing algorithm to find the optimal number of polling servers. The climbing algorithm will start from 1 polling server, and we consider the original number to be optimal once it is found that adding an additional polling server, the optimal value found by GA is worse. In the case of different numbers of polling servers, we run GA separately and compare the performance of optimal solutions consisting of different number of GA.

3.3 Simulation Model

3.3.1 Event trigger tasks schedule simulation using Explicit Deadline Periodic(EDP)

Introduction The ET(event trigger) tasks are handled with polling servers[7]. There can be multiple polling servers and several ET tasks assigned to a polling server. So, it is necessary to judge whether the ET tasks can be scheduled in each polling server and

calculate their worst-case response times. We choose the Explicit Deadline Periodic(EDP)[8] to accomplish this mission.
The $lslbf(t)$ is the linear supply lower bound function with $\alpha = \frac{C_p}{T_p}$ and $\Delta = T_p + D_p - 2 \cdot C_p$, as

$$lslbf(t) = max\{0, (t - \Delta)\} \tag{4}$$

The $H_i(t)$ means that the maximum load of level-i. It is defined as

$$H_i(t) = \sum_{\forall \tau_j \in \mathcal{T}^{ET}, p_j \geq p_i} \left\lceil \frac{t}{T_j} \right\rceil \cdot C_j. \tag{5}$$

The schedulability condition for an ET task $\tau_i \in \mathcal{T}^{ET}$ is defined in Lemma 1 of [?]. The worst-case response time R_i for task $\tau_i \in \mathcal{T}^{ET}$ can be calculated by determining $H_i(t)$ intersects the $lslbf(t)$, as follows:

$$R_i = \text{earliest } t : t = \Delta + H_i(t)/\alpha. \tag{6}$$

Implementation The input of this algorithm is the table of the ET tasks in a polling server and the attributes of this polling server including the duration, period, and deadline. And the output is the worst-case response time of each task and whether these ET tasks can be scheduled by the polling server with these attributes. And Algorithm 3 will show the pseudo-code of this method.

Algorithm 3: Schedulability of ET tasks under a given polling task

```
Data: polling task budget $C_p$, polling task period $T_p$, polling task deadline $D_p$, subset of
      ET tasks to check $\mathcal{T}^{ET}$
Result: $\{true, false\}, responseTime$
/* Compute $\alpha$ and $\Delta$ according to [10] */
1  $\Delta \leftarrow T_p + D_p - 2 \cdot C_p$;
2  $\alpha \leftarrow \frac{C_p}{T_p}$;
   /* The hyperperiod is the least common multiple of all task periods in $\mathcal{T}^{ET}$
      */
3  $T \leftarrow lcm\{T_i \mid \forall \tau_i \in \mathcal{T}^{ET}\}$;
4  for $\tau_i \in \mathcal{T}^{ET}$ do
5      $t \leftarrow 0$;
       /* Initialize the response time of $\tau_i$ to a value exceeding the deadline */
6      $responseTime \leftarrow D_i + 1$;
       /* Remember, we are dealing with constrained deadline tasks, hence, in
          the worst case arrival pattern, the intersection must lie within the
          hyperperiod if the task set is schedulable. */
7      while $t \leq T$ do
           /* The supply at time $t$ (c.f. [11]) */
8          $supply \leftarrow \max(0, \alpha \cdot (t - \Delta))$;
           /* Compute the maximum demand at time $t$ according to Eq. 5 */
9          $demand \leftarrow 0$ ;
10         for $\tau_j \in \mathcal{T}^{ET}$ with $p_j \geq p_i$ do
11             $demand \leftarrow demand + \left\lceil \frac{t}{T_j} \right\rceil \cdot C_j$ ;
           /* According to Lemma 1 of [11], we are searching for the earliest
              time, when the supply exceeds the demand */
12         if $supply \geq demand$ then
13             $responseTime \leftarrow t$;
14             break;
15         $t \leftarrow t + 1$;
       /* If for any task the intersection of the demand and supply are larger
          than the task deadline, the task set $\mathcal{T}^{ET}$ is not schedulable using the
          given polling task parameters. */
16     if $responseTime > D_i$ then
17         return false, responseTime;
18 return true, responseTime;
```

3.3.2 EDF simulation for determining the schedule of TT tasks

Besides the event trigger (ET) tasks, we also need to consider the time trigger (TT) tasks in this problem. Early deadline first (EDF)[9] scheduling is an intuitive and effective method. Besides TT tasks, polling servers, mentioned in EDP, are also scheduled together, and they also equivalent to the tasks in the actual scheduling. In the EDF simulation, we also hold an assumption that the simulation time-unit is small enough, so all time, e.g. computation time, deadline, period, are the integer multiple of a time unit.

The idea of EDF is inspecting the dealines of all unfinished tasks (including polling servers), and executing tasks with the closest deadline first. When a task is executed, the pending calculation time subtracts the corresponding execution time. When the pending calculation time is 0, the task is marked as finished.

The tasks have different periods. To ensure the the completeness of execution, the simulation time is set to the least common multiple of all period data. Thus, all TT tasks and polling servers are released and executed several times. Tasks are released periodically according to their period. When a task is released, the simulation model will refresh the pending calculation time and deadline; all tasks are released at $t = 0$ as initialization. When a task is finished before its deadline, The program counts how long it takes to finish after this release. If this time is longer than the previously recorded WCRT, then update the WCRT of this tasks. In contrast, if the simulation model detects that a task has not completed after its deadline, the model will report that the set of tasks cannot be properly scheduled.

The advantage of the aforementioned set of simulation time is that it can guarantee the schedulability. However, in practice, the runtime of simulation model and storage space of schedule table is an issue that impact the feasibility of the model. To be specific, the polling servers' periods are a segment of decision variables, and their searching spaces are from 2 to the least common multiple of TT tasks' periods. The polling server periods, therefore, are likely to be large and mutually exclusive with the period of TT tasks. Thus, the simulation time can be very long, especially when the solution exploits several polling servers. We solve this issue by limiting the search space of polling server's periods, and the limited search space includes integer from 2 to 20 and the factors of least common multiple of TT tasks' period. This solution has its solid advantages, including complete task execution, runtime of simulation, and the same extreme values as the original search space. The pseudo-code is shown in Algorithm 4.

3.4 Correctness Test

3.4.1 Implement

To ensure the correctness of the simulation model, we use a function to check whether the result of schedules all satisfies the constraints of our project. If the simulation model is true, it must meet three constraints in our project:

1. All the tasks must be finished before the deadline;
2. The real computation time of each task must equal the required computation time in every period.

Algorithm 4: Scheduling TT tasks via EDF simulation

```
Data: TT task set $\mathcal{T}^{TT}$ including polling server tasks $\mathcal{T}^{poll}$
Result: TT schedule table ($\sigma$) and WCRTs of TT tasks ($WCRT_i$)
1  $T \leftarrow lcm\{T_i \mid \forall \tau_i \in \mathcal{T}^{TT} \cup \mathcal{T}^{poll}\}$ ;
2  $\forall\ \tau_i \in \mathcal{T}^{TT} \cup \mathcal{T}^{poll}$: $c_i \leftarrow C_i$; $d_i \leftarrow D_i$;$r_i \leftarrow 0$;$WCRT_i \leftarrow 0$;
3  $t \leftarrow 0$;
   /* We go through each slot in the schedule table until $T$ */
4  while $t < T$ do
5      for $\tau_i \in \mathcal{T}^{TT} \cup \mathcal{T}^{poll}$ do
6          if $c_i > 0 \wedge d_i \leqslant t$ then
7              return $\emptyset$; /* Deadline miss! */
8          if $t\%T_i == 0$ then
               /* Task release at time $t$ */
9              $r_i \leftarrow t$;
10             $c_i \leftarrow C_i$;
11             $d_i \leftarrow t + D_i$;
       /* Check if there is any tasks with computation left */
12     if $\left[c_i = 0, \forall i \in \mathcal{T}^{TT} \cup \mathcal{T}^{poll}\right]$ then
           /* If no task has computation left, schedule idle slot */
13         $\sigma$[t] $\leftarrow$ idle;
14     else
           /* If there are tasks with computation left, schedule the one with the
              earliest deadline */
15         $\sigma$[t] $\leftarrow \tau_i = EDF(t, \mathcal{T}^{TT} \cup \mathcal{T}^{poll})$;
16         $c_i \leftarrow c_i - 1$;
17         if $c_i == 0 \wedge d_i \geqslant t$ then
               /* Check if the current WCRT is larger than the current maximum.
                  */
18             if $t - r_i \geqslant WCRT_i$ then
19                 $WCRT_i \leftarrow t - r_i$;
20
21     $t \leftarrow t + 1$;
22 if $\left[c_i > 0, \forall i \in \mathcal{T}^{TT} \cup \mathcal{T}^{poll}\right]$ then
       /* Schedule is infeasible if any TT task has $c_i > 0$ at this point */
23     return $\emptyset$;
24 returns $\sigma, WCRT_i$;
```

3. No overlap of the execution time in each moment.

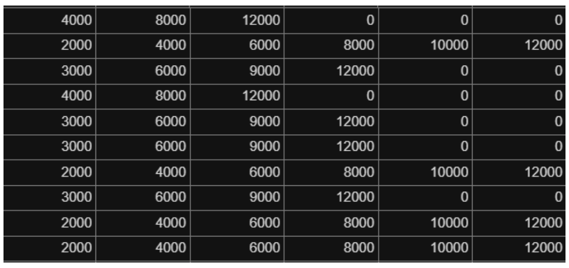
In our output schedule table, we list all the moments, where each of them can be only followed by one task. In this case, the last constrain can be guaranteed. So we only need to check the first two constraints.
The pseudo-code is displayed in Algorithm 5.

Algorithm 5: correctness test

```
Data: task$_table, result, schedule\_table, statistic\_result, expect\_static\_result$
Result: true,false
1  if $result$ is empty then
2  |   pass
3  else
   |   /* Calculate hyperperiod                                      */
4  |   T ← $task\_table.Period(1)$; for
5  |   times ← $T./task\_table.period$
6  |   [task_name,~] ← $unique(schedule\_table(:,2),'last')$;
7  |   expect_relative_ddl ← $task_table.Deadline$
8  |   expect_absolute_ddl ← $zeros(size(task_name,1),max(times))$
9  |   expect_absolute$_ddl(:,1)$ ← $task\_table.Deadline$
10 |   real_execution_time ← $zeros(size(task\_name,1),max(times)+1)$
11 |   for $i \in length(length(task\_table))$ do
12 |   |   for $j \in times(i)+1$ & $j$ start at 3 do
13 |   |   |   expect_absolute_ddl(i,j)
   |   |   |   ← $expect\_absolute\_ddl(i,j-1) + task\_table.period(i)$
14 |   for $i \in length(length(task\_table))$ do
15 |   |   for $j \in times(i)$ & $j$ start at 2 do                     height 0.4pt dep
   |   |   |   /* Only need to check whether the computation time is equal to
   |   |   |      expectation absolute deadline                          */
16 |   |   |   $real\_execution\_time \leftarrow sum(schedule_table(expect\_absolute\_ddl(i,j-1), expect\_absolute\_ddl(i,j)) == i)$
17 |   return sum(sum(real_execution_time==task_table.Duration))==sum(times)
```

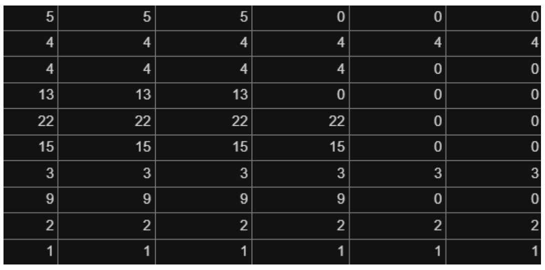
3.4.2 Result

Illustrated here are the results for the first 10 tasks. Figure 2(a) is the absolute deadline. In each deadline, the computation time must be the same with what we expected. Figure 2(b) is the computation time, while the first column in figure 2(c) is the expected one. It is obvious that all the tasks satisfy the constraints.
According the the results, we may draw a conclusion that our simulation system can schedule the tasks of the ADAS correctly. In the following thesis, we use this simulation model to validate the solutions.



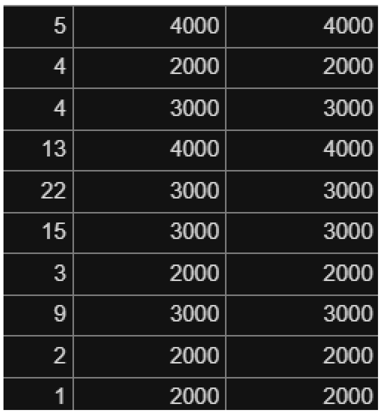
4000	8000	12000	0	0	0
2000	4000	6000	8000	10000	12000
3000	6000	9000	12000	0	0
4000	8000	12000	0	0	0
3000	6000	9000	12000	0	0
3000	6000	9000	12000	0	0
2000	4000	6000	8000	10000	12000
3000	6000	9000	12000	0	0
2000	4000	6000	8000	10000	12000
2000	4000	6000	8000	10000	12000

(a) expect absolute deadline



5	5	5	0	0	0
4	4	4	4	4	4
4	4	4	4	0	0
13	13	13	0	0	0
22	22	22	22	0	0
15	15	15	15	0	0
3	3	3	3	3	3
9	9	9	9	0	0
2	2	2	2	2	2
1	1	1	1	1	1

(b) Real execution time



5	4000	4000
4	2000	2000
4	3000	3000
13	4000	4000
22	3000	3000
15	3000	3000
3	2000	2000
9	3000	3000
2	2000	2000
1	2000	2000

(c) task table(duration, period, deadline)

Figure 2: Correctness test

4 Experiment

4.1 Solution to the original problem

For a given test case and polling server number, our original version can find the sub-set of ET tasks, period, budget and deadline for each polling server so that the output WCRT to be the lowest. We set number of polling server to be 1, penalty to be 10000, population to be 1000 and max generation to be 100. The simulation result after running for several hours is shown in Fig 3.

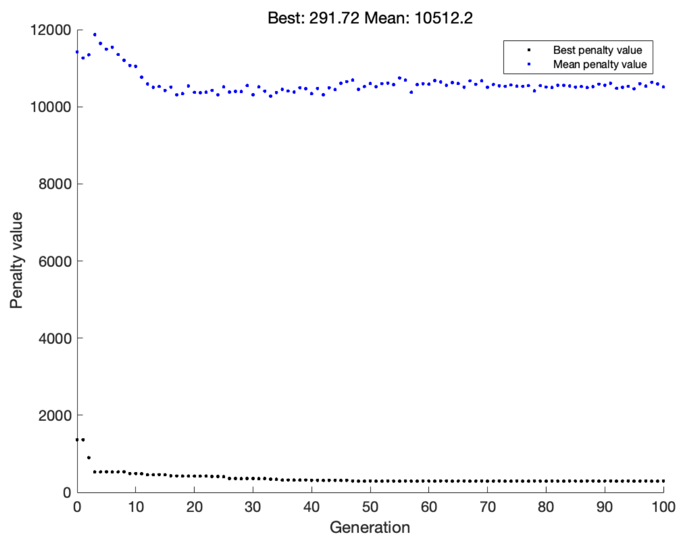


Figure 3: Simulation result of our solution to the original problem with 1 polling server

Then we search the optimal solution when 2 polling servers are utilized. The simulation result after running is shown in Fig 4.

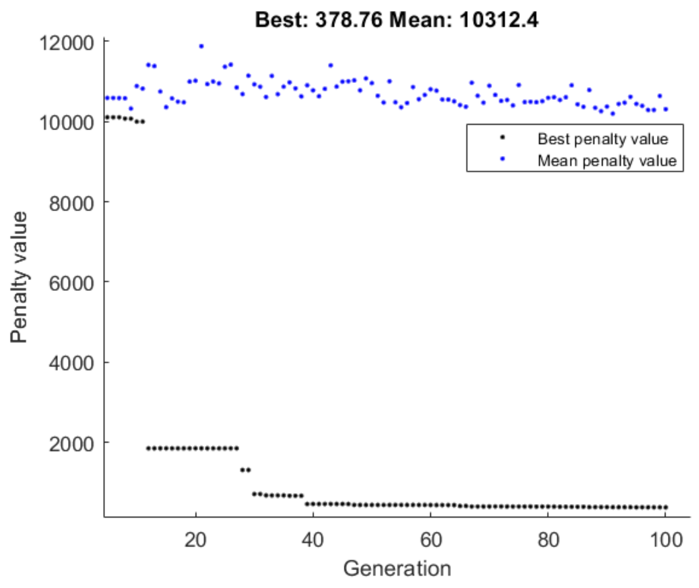


Figure 4: Simulation result of our solution to the original problem with 2 polling servers

According to the climbing algorithm for the number of polling server, we found the result become worse, and we can infer that WCRT of system reach the best when number of polling server is 1.

4.2 Speedup by using Processor Demand Criterion

4.2.1 Implementation

In this extension problem, we check the schedulability of TT tasks by using *processor demand criterion*[9]. This book provides us with two theorems. The first one based on the condition that the relative deadlines of a set of tasks equal to periods. They are schedulable if and only if:

$$L > 0 \quad \sum_{i=1}^{n} \lfloor L/T_i \rfloor C_i \leq L \tag{7}$$

However, in our project, the relative deadlines not equal to periods. So this formula is not suitable for our project.
And the second theorem is based on the relative deadlines equal to periods. As can be seen in the test case, all the relative deadlines of TT tasks are equal to their periods. However, in terms of the polling server, which is regarded as a TT task in our project, the relative deadline is less than the period in some cases. As a consequence, the second theorem is tailored to our project. The basic principle of this theorem is shown as followed.

In time [0,L], the processor is:

$$g(0, L) = \sum_{i_1}^{n} \eta_i(t_1, t_2)C_i \tag{8}$$

We assume that the relative deadlines(D_i) are shorter than periods(T_i), and the periodic tasks are activated at time $t = 0$. In this case, the number of instances of task τ_i that contribute the demand in $[0, L]$ is:

$$\tau_i(0, L) = \lfloor (L + T_i - D_i)_i \rfloor \tag{9}$$

Thus the processor demand in $[0, L]$ is:

$$g(0, L) = \sum_{i=1}^{n} \lfloor (L + T_i - D_i)/T_i * C_i \rfloor \tag{10}$$

Function $g(0, L)$ is the *Demand Bound Functions*:

$$dbf(t) = \sum_{i=1}^{n} (\lfloor (t + T_i - D_i)/Ti \rfloor)C_i \tag{11}$$

where T_i is the period of each task, D_i is the relative deadline and C_i is the computation time.
According to theorem of Jeffay and Stone[**?**], a set of periodic tasks is schedulable if and only if:

$$t \in \mathscr{D} \quad dbf(t)t \tag{12}$$

Besides, we can observe that:

1) The activated time is all $t = 0$, we can get the hyperperiod H, so we need to verify the value of L is not larger than H.

2) Step function $g(0, L)$ only increase while L crosses a deadline d_k. As a result, L only needs to get the values of all deadlines.

Finally, we can judge whether a set of TT tasks are schedulable or not by using the following two conditions:

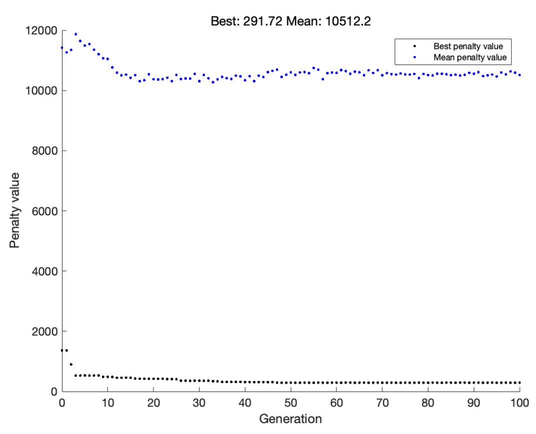
$$\begin{cases} U < 1 \\ t \in \mathscr{D} \quad dbf(t) \leq t, \end{cases} \tag{13}$$

where

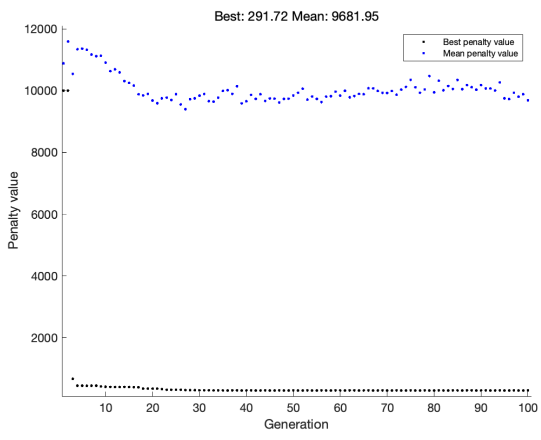
$$\mathscr{D} = \{d_k | d_k \leq min[H, max(D_{max}, L^*)]\} \tag{14}$$

and

$$L^* = (\sum_{i=1}^{n} (T_i - D_i)U_i)/(1 - U) \tag{15}$$



(a) result of our original version



(b) result of our improved version with PDC method

Figure 5: Results of the original version and extension one version

and

$$U = \sum_{i=1}^{n} C_i/T_i \tag{16}$$

4.2.2 Result

To prove the correctness and robustness of our programme with extension one, 34 test cases were tested by us. The result shows that this programme can perfectly well in all the test cases. And we take one test case as an example.
Figure 5 demonstrates the search progress of the original version and the improved one with processor demand criterion of test case "inf_10_10" in testcases_seperation_VFINAL folder.
As can be seen in these two pictures, we get the same result between two versions of our optimization algorithms under the same data set, which is 291.72. The table 1 below illustrates the final attributes of the polling server. Besides, we compare the running time between two versions of our our optimization algorithms. In our test, the average running time of 100 generation of original version is 974s, while the second one only needs 783s. We reduce the running time by 19.6% through deploying Processor Demand Criterion method.

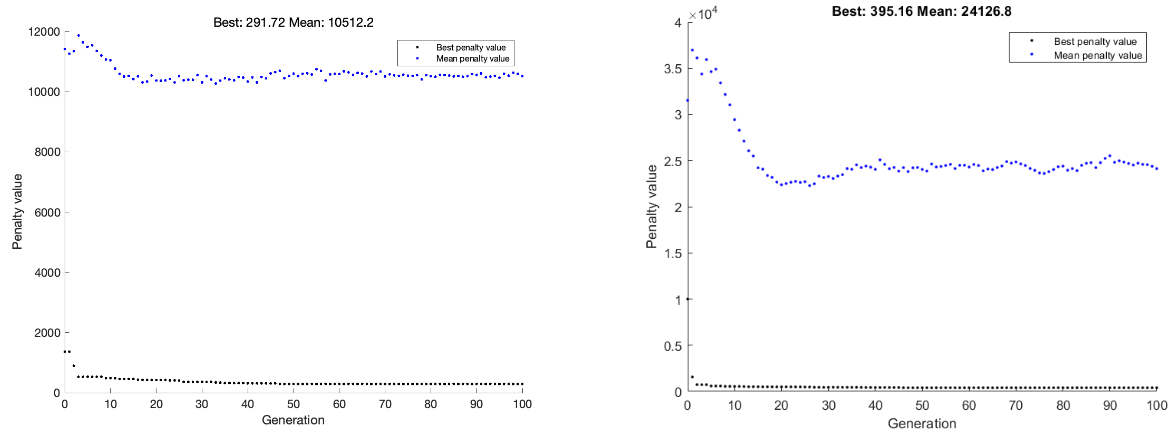
	Computation time	Period Time	Deadline
original version	1	2	1
processor demand criterion	1	2	1

Table 1: final solution vector comparison between two versions

4.3 Optimize WCRT by re-assigning priorities of ET tasks

We already get the inference from experiment 1 that if we increase the number of polling servers, output WCRT will become worse. Thus we want to verify a hypothesis that when having multiple polling tasks handling different subsets of ET tasks, it may be beneficial to re-assign priorities within each individual ET task sub-set.

But before that we verified another hypothesis that **tasks with lower deadline should be given higher priority in order to get better WCRT**. As shown in Fig 6, we compare normal test cases with test cases whose ET tasks priorities are all 0, and get verified our hypothesis. With this inference, we can greatly narrow down the search space.



(a) result with original test case (b) result with all-equal priority test case

Figure 6: Results comparison with original and all-equal priority test cases

Then we have to modify the template of chromosomes and upper and lower bound for each slot. Since we already verify that ET tasks with lower deadline should be given higher priority, we want to use k-means to group ET tasks with nearby deadlines into one category.

$$\mathbf{x} = [n, \{C_1, D_1, T_1\}, ..., \{C_i, D_i, T_i\}, ...\{C_n, D_n, T_n\}, \{S_1, S_2, ..., S_j, ..., S_m\}, \{P_1, P_2, ..., P_j, ..., P_m\}] \tag{17}$$

The first element represents the number of polling servers. The second part represents [computing,period,deadline], 3 elements for each polling server. The 3rd part represents which polling server each ET task is put in. The added fourth part represents priority of each ET task.

For example, if we have 3 cluster centers for 6 priorities, first we need to sort cluster centers from small to big. Then for the tasks in cluster 1, the upper bound of them is 6 and lower bound is 4. For the tasks in cluster 2, the upper bound of them is 4 and lower bound is 2. For the tasks in cluster 3, the upper bound of them is 2 and lower bound is 0.

Then We set number of polling server to be 1, penalty to be 10000, population to be 1000 and max generation to be 100. From Figure 7, our method get the similar result with the best test cases.

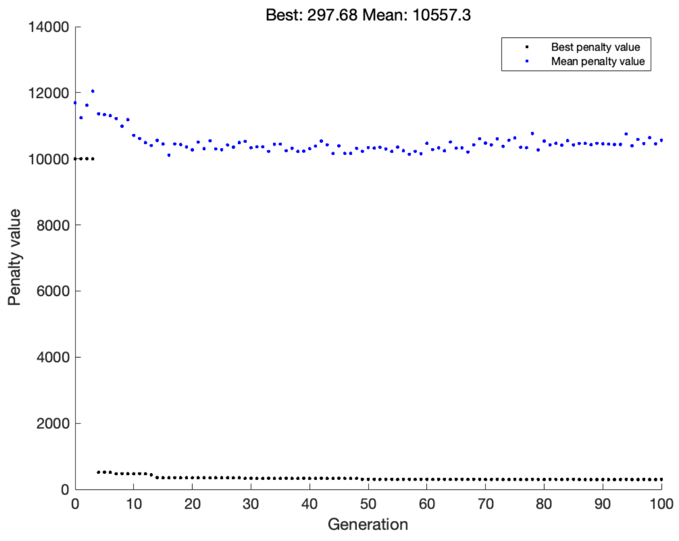


Figure 7: Simulation result of third version with priority re-assignment

Task number	Real-priority	Configured-priority
task 1	2	0
task 2	0	0
task 3	2	1
task 4	4	2
task 5	3	2
task 6	2	2
task 7	2	2
task 8	4	2
task 9	4	2
task 10	6	2
task 11	4	3
task 12	4	3
task 13	6	5
task 14	6	5
task 15	6	5
task 16	4	5
task 17	6	5
task 18	4	6
task 19	6	6
task 20	6	6

Table 2: Solution vector comparison

Compared to the result with prearranged priority, the result shown in the above figure(2) is slightly worse. On the one hand, it validates the correctness of our insights about the proper priority of ET tasks. On the other hand, the gap of performance difference are because first, the rearranged priority of ET tasks is already relative new. The second reason can be the metaheuristic's solid feature, that global optimal solution is not guaranteed.

4.4 Speed up with External Point Method

4.4.1 Introduction

In section 3.3.1, we check whether the deadline is missed for any ET tasks in the given input set. However, there is an optimization that can be employed by checking the schedulability of selected ET tasks using the so-called external points method from [11]. The process of the external points method is to use another function to replace the *lslbf(t)* from Eq.(4) to remove some points which can not be scheduled to optimize the time of this algorithm.

4.4.2 Implement

In this section, we try to implement it but we can not reach the ideal situation. We built a function to verify the feasibility of this algorithm. We modify the $lslbf(t)$ from Eq(4). We use the two farthest points to construct this line. And the specific pseudo-code is shown in Algorithm 6.

Algorithm 6: Schedulability of ET tasks under a given polling task

```
Data: polling task budget $C_p$, polling task period $T_p$, polling task deadline $D_p$, subset of
      ET tasks to check $\mathcal{T}^{ET}$
Result: $\{true, false\}, responseTime$
1  $\Delta \leftarrow T_p + D_p - 2 \cdot C_p$;
2  $\alpha \leftarrow \frac{C_p}{T_p}$;
3  $T \leftarrow lcm\{T_i \mid \forall \tau_i \in \mathcal{T}^{ET}\}$;
4  for $\tau_i \in \mathcal{T}^{ET}$ do
5      for $\tau_j \in \mathcal{T}^{ET}$ with $p_j \geq p_i$ do
6          $demand_max \leftarrow demand_max + \left\lceil \frac{t}{T_j} \right\rceil \cdot C_j$ ;
   /* $\alpha_{ex4}$ and $\delta_{ex4}$ means the slope and intercept of the $lslbf(t)$. $T_{min}$ and $T_{max}$
      mean the minimum and maximum value of the period of the polling server.
      $C_{min}$ means the minimum value of the polling task budget of the polling
      server                                                                   */
7  $\alpha_{ex4} \leftarrow (demand_{max}$ - C$_{min})/(T_{max} - T_{min})$;
8  $\delta_{ex4} \leftarrow$ -(C$_{min} - \alpha_{ex4}$ * T$_{min})/(\alpha_{ex4})$;
9  for $\tau_i \in \mathcal{T}^{ET}$ do
10     $t \leftarrow 0$;
11     $responseTime \leftarrow D_i + 1$;
12     while $t \leq T$ do
13         $supply \leftarrow \alpha_{ex4} * (t - \delta)$;
14         $demand \leftarrow 0$ ;
15         for $\tau_j \in \mathcal{T}^{ET}$ with $p_j \geq p_i$ do
16             $demand \leftarrow demand + \left\lceil \frac{t}{T_j} \right\rceil \cdot C_j$ ;
17         if $supply \geq demand$ then
18             $responseTime \leftarrow t$;
19             break;
20         $t \leftarrow t + 1$;
21     if $responseTime > D_i$ then
22         return false, responseTime;
23 return true, responseTime;
```

4.4.3 Discussion

The method introduced in the paper [11] assumes $T_p = D_p$, while in our test cases usually $T_p > D_p$. So, it can not get the ideal effect when we try to achieve the algorithm. We can not find a feasible solution because every solution is penalized. The result will be shown as follows.

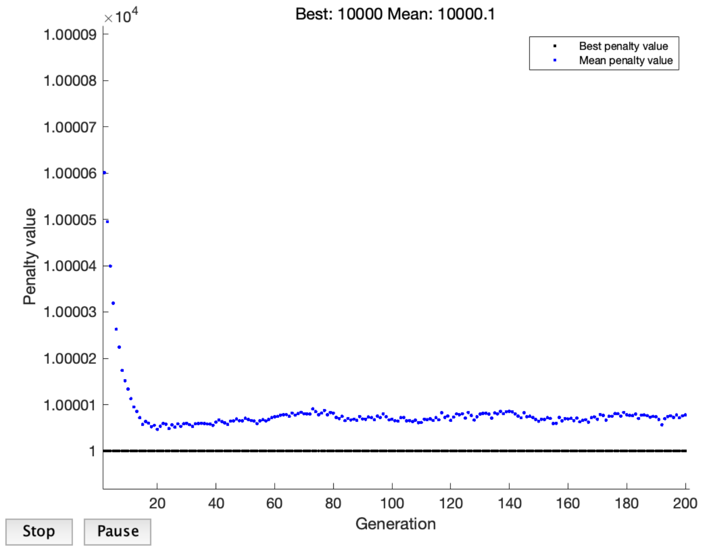


Figure 8: Simulation result with external point method

4.5 Take criticality tasks into consideration

4.5.1 Introduction

In the original algorithm of ET tasks using the EDP algorithm, we just distribute the ET tasks into different polling servers randomly. But ET tasks may have the "separation requirements" which means the ET tasks have a definite polling server. For example, different ET tasks may belong to applications of different critical. We want to use this method to protect the higher criticality tasks from lower criticality tasks. So, in extension 5, several ET tasks will be distributed polling server numbers before doing the optimization.

4.5.2 Implementation

To make sure the polling sever is fixed in these tasks, we make a judge before set the range of the polling servers' number. In this part, we will separate the tasks having a definite polling server or not. The range of the tasks which have a definite polling server number is the number of the polling server. And the tasks which do not have a definite polling server will have a range from zero to the max of the polling server number.

4.5.3 Result

After distributing the polling server numbers to these ET tasks, we can figure out the object value is worse than the initial version by testing the separation test cases. We take one test case of the separation test case as follows. The number of polling servers is 2. And the parameter of these polling servers is shown in table(3). The result of the mission is shown in table(4).

	Computation time	Period Time	Deadline
polling server 1	32	120	73
polling server 2	22	96	22

Table 3: Attributes of polling servers in final solution vector

Task number	Output polling server assignment	Input test case
task 1	2	2
task 2	1	1
task 3	2	0
task 4	1	0
task 5	1	0
task 6	1	0
task 7	1	1
task 8	2	0
task 9	2	0
task 10	2	0
task 11	2	0
task 12	2	0
task 13	1	0
task 14	1	0
task 15	2	0
task 16	2	0
task 17	2	0
task 18	2	0
task 19	2	0
task 20	2	0

Table 4: Solution vector comparison

The figure of the simulation result is shown in the Fig(9).

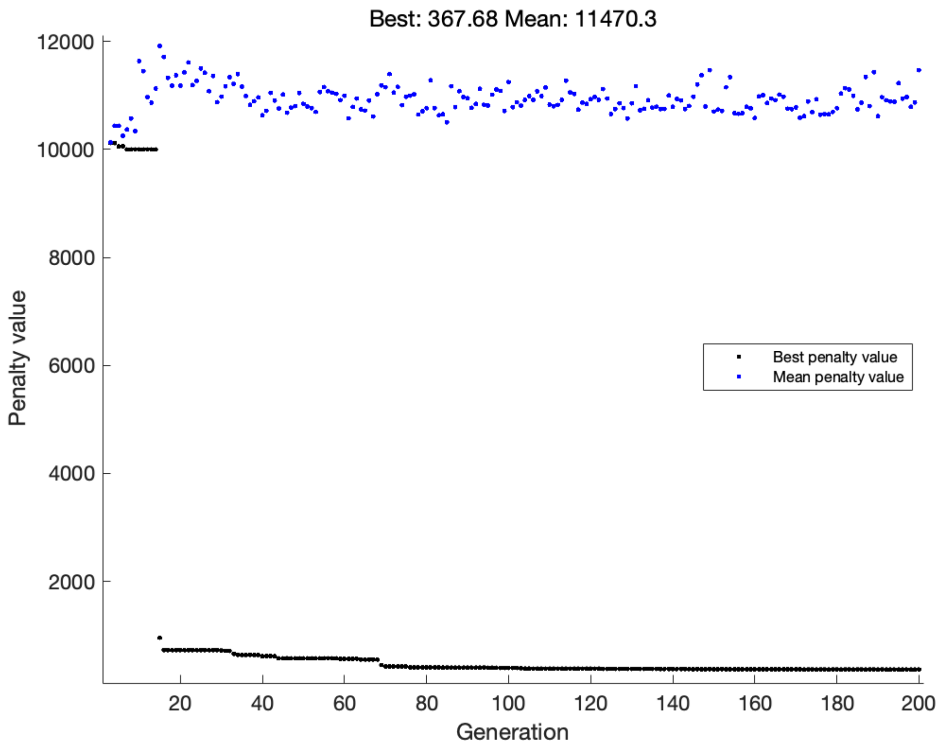


Figure 9: result of the test case with pre-assigned tasks

5 Conclusion

In this project, we developed a simulation model by which TT and ET tasks are scheduled according to a certain sorting principle, build a mathematical model about the optimization problem, and utilize the generic algorithm to find optimal solution in feasible time, successfully solving the combinatorial optimization problem in ADAS. Our study allows an ADAS to optimally decide the number of polling servers, their attributes (e.g. computation time, period and deadline), and the allocation of ET tasks among these polling servers. What's more, we also implement some beneficial expansion and exploration.

In Section 4.2, **we speed up the running time of our optimization algorithm by 19.6% using Processor Demand Criterion method.** We achieved this by checking if the set of TT tasks is schedulable by using the Processor Demand Criterion (PDC) before creating a schedule table using EDF. If the PDC detects the TT tasks are not schedulable, we can skip executing the EDF but just penalize the objective function to achieve the speedup of our optimization algorithm.

In Section 4.3, **we tried to optimize WCRT by re-assigning priorities of ET tasks** The test cases contain the priorities of the ET tasks as inputs. We also implement a good insight about prioritization of ET tasks. Although the performance of optimal solution with re-assigning priority is slightly worse than optimal solution with the re-assigned priority, this attempt makes it possible to optimally assign the priorities of ET tasks and shows the potential for future improvement.

In Section 4.4, **we tried to speed up our optimization algorithm with External Point Method, but did not achieve the expected result**. We guess the reason is that the EDM method [11] assumes $T_p = D_p$, while in our test cases $T_p > D_p$. So, it can not get the ideal effect and every solution is penalized. Thus we conclude that External Point Method is not suitable for our project.

In Section 4.5, **we take criticality tasks into consideration**, which means tasks assigned with higher criticality and tasks assigned with lower criticality cannot exist in the same polling server. We achieved this by first judge the proper number of polling servers and then set the upper bound and lower bound to specific et task slots. After getting the optimal solution, we compare its objective value with that under original unconstrained test case and the results turned out to be worse without surprise.

References

[1] Official document of Genetic Algorithm, `https://se.mathworks.com/help/gads/genetic-algorithm.html`

[2] Shane D McLean, Emil Alexander Juul Hansen, Paul Pop, and Silviu S Craciunas. Configuring adas platforms for automotive applications using metaheuristics. Frontiers in Robotics and AI, 8, 2021.

[3] Luis Almeida and Paulo Pedreiras. Scheduling within temporal partitions: Response-time analysis and server design. In Proc. EMSOFT. ACM, 2004. `doi:10.1145/1017753.10`

[4] Oliver Sinnen. Task scheduling for parallel systems. WileySons, 2007.

[5] Giorgio C Buttazzo. Hard real-time computing systems: predictable scheduling algorithms and applications, volume 24. Springer Science Business.

[6] Burke, Edmund K. and Graham Kendall. "Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques." (2013).

[7] Insik Shin and Insup Lee. Compositional real-time scheduling framework with periodic model. ACM Trans. Embed. Comput. Syst., 7(3):30:1-30:39, 2008. `doi:10.1145/1347375`

[8] Giuseppe Lipari and Enrico Bini. Resource partitioning among real-time applications. In Proc. ECRTS, 2003. `doi:10.1109/EMRTS.2003.1212738.`

[9] Giorgio C Buttazzo. Hard real-time computing systems: predictable scheduling algorithms and applications, volume 24. Springer Science & Business Media, 2011.

[10] Amir Aminifar, Enrico Bini, Petru Eles, and Zebo Peng. Designing bandwidth-efficient stabilizing control servers. In Proc. RTSS. IEEE, 2013. `doi:10.1109/RTSS`

[11] Luis Almeida and Paulo Pedreiras. Scheduling within temporal partitions: Response-time analysis and server design. In Proc. EMSOFT. ACM, 2004. `doi:10.1145/1017753.10`

A Appendix

A.1 Work division and contribution

Jiaxuan Wu(s212625): In algorithm design, I participated in designing the architecture of our optimization algorithm and also the interface of its components. In algorithm implementation, I participated in implementation of original version, improved version with Processor Demand Criterion (PDC) method and the improved version that can re-assign priorities of ET tasks. In report writing, I'm responsible for structure of the overleaf project, section 3.1 (Top-down overview), section 4.1 (Solution to the original problem), section 4.3 (Optimize WCRT by re-assigning priorities of ET tasks), and final full text checking.

Yingli Duan(s222462): In algorithm design, I participated in designing the architecture of the optimization algorithm. In algorithm implementation, I am responsible for checking the schedulability of TT tasks, and the correctness test, and participate in checking the schedulability of ET tasks. In the report part, I am responsible for section4.2.1(Speedup by using Processor Demand Criterion) and section3.4(Correctness Test) and take part in writing section3.2(Optimization with GA). What is more, I also responsible for the function to generate the time-computation time images, which finally are not demonstrated in our report.

Peichen Liu(s222475): In the simulator design, I finished the Event trigger tasks schedule simulation using Explicit Deadline Periodic(EDP) section and participated in the EDF algorithm design. In algorithm implementation, I participate in the Speed up with External Point Method and Take criticality tasks into consideration section. In the report part, I am responsible for section 3.3.1(Event trigger tasks schedule simulation using Explicit Deadline Periodic), section 4.4(Speed up with External Point Method), and section 4.5(Take criticality tasks into consideration).

Yuansheng Zhao(s222425): In simulator design, I finished the file input part and also the data convert part makes the data in multiple .csv files can be loaded to our simulator simply, also cooperates with other team members with the data implementation. In the testing part, I spent more than 6 hours to get the best solutions and worked out for more than 25 groups of test raw data. In the report part, I am responsible for the section 3.2.2(Genetic algorithm's parameters) and 1.1(Problem Introduction). What's more, I tried to implement extension 4 but after several failed attempts, I found it may not fit for our simulator.

Shuoqiang Zeng(s210357): I am responsible to the theoretical part, including the mathematical representation of the optimization model and the definition of constraints, decision variables and objective function (section 2.2). I and Wu design the structure of solution to this simulation optimization problem (section 3.1). As for the optimization part (section 3.2), I use GA, programme constraints, apply the penalty method for unschedulable solutions, and code the main body of objective function. Among these works, Zhao helps me to determine the GA's parameters (section 3.2.2), and the objective function call functions about simulation. In the simulation model, I implement the EDF simulation of TT tasks (3.3.2), and Liu helps debug and decide the data structure. I worked with Wu on section 4.3, responding the extension 2, but the documentation of this work is finished by Wu individually.