Managing Temporal Constraints in Control Systems

Kristian Sandström and Christer Norström

Mälardalen Real-Time Research Centre, Department of Computer Engineering,
Mälardalen University, Västerås, Sweden

Abstract

Design and implementation of motion control applications include the mapping of control design to real-time system implementation. Important parameters from control design include deviation from nominal period time of an activity, end-to-end timing constraints, temporal correlation between different sampling tasks, and constraints on temporal variations in output. These parameters should also be considered in the real-time systems design, since translating them to simple deadlines may lead to sub-optimal solutions. Many real-time systems in industry today are based on pre-emptive priority based run-time systems, and hence, it is highly desirable to fulfill the temporal requirements by correctly assigning attributes such as priorities and offsets to the tasks executing in such systems. However, this is a non-trivial mapping, which should be supported by appropriate methods and tools. In this paper we propose a method, which by assigning priorities and offsets to tasks provides guarantees that complex timing constraints are met. The method handles periodic and sporadic tasks, shared resources, and varying execution times of tasks. We present the method, which uses a genetic algorithm, together with simulation results, showing that the proposed method is capable to efficiently handle complex constraints on task sets of realistic sizes covering most embedded control systems.

1 Introduction

To successfully design and implement motion control applications, such as robots, vehicle/trucks, and mobile machinery, in distributed computer systems there is a need to make a smooth and predictable transition from the design of a control system to its implementation in the computer system. One important prerequisite to accomplish this for real-time systems is to appropriately derive and model application timing requirements [1]. Moreover, these requirements must be translated into timing constraints that are suitable for implementation, thereby providing means for interaction between control and computer engineers. The timing constraints in the control design cannot be directly mapped to attributes of a real-time system, such as priorities, period times, deadlines and offsets of tasks. Assigning the attributes of the tasks so that the complex timing constraints derived from the control design are fulfilled is a non-trivial problem. Typical complex timing constraints are tolerances on sampling periods, end-to-end timing constraints, temporal correlation between different sampling tasks, and constraints on temporal variations in output.

The aim of this paper is to show how these complex timing constraints can be mapped to attributes of periodic tasks running on standard pre-emptive priority based multitasking real-time operating systems, as for example WxWorks provided by Windriver, in such a way that the timing constraints are fulfilled. In order to guarantee the behaviour of a control system subject to complex timing constraints, one must also consider that execution times of activities in most cases vary. Varying execution times will directly affect e.g., constraints on maximum deviation from a nominal period time.

Bate and Burns [2], propose a related method for assigning offsets and priorities to a fixed priority pre-emptive task set. They define a specification model that allows for expressing similar constraints as defined in Section 2 of this paper. However, their method does not consider the use of shared resources between sporadic and periodic tasks. Furthermore, attribute assignment for dealing with constraints on period time variation is managed using a heuristic algorithm with local optimization that, according to the authors, can lead to attribute assignments causing unschedulable systems when a feasible solution exists. For this reason it is difficult to extend the method to incorporate the additional constraints we would like to consider. Several researchers have attacked the same problem by generating off-line schedules [3][4][5]. A major disadvantage of such a solution is that it cannot be handled by a standard priority-based RTOS. Furthermore, they do not support pre-emption, sporadic activities, and varying task execution times. In [6][7] a method is presented for translating off-line schedules to task attributes for fixed priority systems (FPS). This method could be combined with methods in [3][4][5] and would enable the use of priority-based RTOS for those methods. However, the combined approached would still inherit the limitations of not supporting pre-emption, sporadic activities, and varying task execution times when searching for a solution to the complex constraints. The method allows for using on-line acceptance test for sporadic and aperiodic tasks to use spare capacity from the tasks translated from the off-line schedule to FPS, while the methods presented in this paper add no run-time overhead for managing sporadic tasks. Moreover, when translating an off-line schedule the method in [6][7] in some cases have to represent an off-line scheduled task by more than one FPS task. This can result in an FPS system with artificial tasks and thereby a greater number of tasks compared to our method. However, if more instances are used it is possible to find solutions that cannot be found otherwise. It is possible using the method presented in this paper to include more than one instance for some or all tasks.

In [8] the authors present a design methodology for real-time systems with end-to-end timing constraints, temporal correlation between different sampling tasks and constraints on temporal variations in output. The methodology derives period times, deadlines, and offsets for the tasks. However, the task model does not agree with a standard priority-based RTOS and constraints on period time variation cannot be expressed. Furthermore, the method assumes that task execution times are static. The work presented in [9] uses genetic algorithms for minimizing jitter in communication using field busses. The problem solved is quite different from the one presented in this paper in that only jitter is minimized and messages do not have interrelated temporal constraints

The motivation for the work presented in this paper mainly originates from our participation in a real industrial project where we used a specification model with support for periodic tasks, deadlines, precedence relationships, mutual exclusions, and offsets [10]. By using this model we can express all timing constraints required by the application. However, the designer has to manually translate the timing constraints into attributes of the used model. This is possible for simple systems, but in systems with many such requirements it becomes very difficult to assign these attributes manually. Even if the designer succeeds in finding a feasible mapping, we get a maintenance problem [10].

In this work we use an enhanced specification model that supports temporal dependencies between tasks. We will show that we can solve the problem of mapping a system described by this specification model to a run-time system model in an efficient way by using a genetic algorithm (GA). There are several reasons for using the GA approach. GA is a general optimisation method that has been used successfully for solving a wide variety of complex problems including scheduling, e.g., in [11][12][13][14]. It can also easily be extended to optimise on other attributes such as minimising the response time of handling an event. One of the most important properties of the GA is its ability to deliver a result that fulfils a subset

of the timing constraints in cases where it is impossible to fulfil all constraints. This information is important since the designer then can get an indication of which constraints that can not be fulfilled and thereby simplify the re-modelling of the application. Also, even if not all timing constraints are fulfilled, the application requirements may in some cases still be fulfilled since the robustness of the control design can tolerate deviations from the specification. However, this has to be verified by control analysis. Simulation results show that our algorithm performs well compared to the algorithm presented in [2] and that it finds solutions to a high degree when the considered systems are schedulable.
Thus, the contributions of this paper are:

- A specification model for describing systems with complex timing constraints.
- A synthesis algorithm that assigns priorities and offsets to tasks to fulfil the timing constraints given our specification model.
- Simulation results showing the efficiency of the suggested method.

The rest of this paper is organised as follows. Section 2 describes the used system model. The method for attribute assignment is covered in Section 3. In Section 4 simulation results are presented, followed by the conclusion of the paper in Section 5. An extensive example of the method can be found in Appendix A.

2 System model

The system model is divided into two parts. The first part specifies the required behaviour of the run-time system and the second part is a definition of the specification model used to express the constraints of the task set.

2.1 Run-time system model

The basic model for the run-time system is a priority based, pre-emptive run-time system with shared resources protected by semaphores conforming to the priority ceiling protocol. Furthermore, the run-time system should provide a mechanism to enforce phasing between tasks i.e., offsets, and the ability to periodically release tasks with some predefined resolution, e.g., the operating system tick. These required features exist in many RTOS and if not, it is quite easy to construct these mechanisms from existing RTOS primitives.

The run-time system may also support prioritised sporadic activities.

2.2 Specification model

The specification model defines the information that has to be specified for each periodic and sporadic task, as well as the constraints that can be expressed on a task set. A periodic task is defined by its worst-case execution time (WCET), best-case execution time (BCET), and nominal period time. The nominal period time is the desired rate at which the task should be executed. Below, the terms start time, completion time, and release are used. The start time of a task is the actual time when the first instruction is executed on the processor, as opposed to release, which is the time when the task becomes ready to execute. The completion time of a task is the time when the last instruction has been executed.
The following constraints can be expressed for and between periodic tasks:

- Deadline – deadline relative to the release of the task.
- Precedence – constraint specifying the execution order between two tasks.
- Separation – constraint of a minimum distance between the completion of one task and start of another task.
- Jitter – the maximum allowed deviation from the nominal period time. Jitter constraints are used to control the deviation from nominal period for e.g., sampling and actuation activities.

    - Start Jitter – maximum allowed deviation of a task's start time from its nominal period time. The constraint is specified by an upper (S_h) and lower bound (S_l) on the time between two consecutive executions of a task.
    - Completion Jitter – maximum allowed deviation of a task's completion time from its nominal period time. The constraint is specified by an upper (C_h) and lower bound (C_l) on the time between two consecutive executions of a task.
- Latency – constraint specifying a maximum allowed distance between the start of one task and the completion of another task.
- Correlation – constraint on the maximum time between executions of two or more tasks executing in parallel. This constraint is used to correlate concurrent sampling or actuation activities in time.
- Shared resources – specification of the tasks that use semaphores and times for the tasks critical sections.

Periodic tasks may have a varying execution time and phasing relative to each other and hence the start time and completion time for a task can vary. This must be considered when finding an attribute assignment meeting the constraints for a task set. Therefore the analysis of a task set is performed for all instances over the least common multiple (*lcm)* of tasks period times. This is necessary since calculation of the earliest- and latest start time and completion time without considering all instances would be too pessimistic. As an example, consider the tasks τ_1 and τ_2 depicted in Figure 1. The tasks have a period time of 10 and only one constraint, a precedence constraint between τ_1 and τ_2. Assume a completion time of τ_1 equal to 3, and a start time of τ_2 equal to 4 in one period and a completion time of τ_1 equal to 2, and a start time of τ_2 equal to 2 in the next period. The latest completion of t_1 relative to the period is 3 and the earliest start relative to the period for τ_2 is 2, i.e., the precedence is violated considering only the task timing while if the separate instances are considered one can see that precedence is achieved between τ_1 and τ_2.

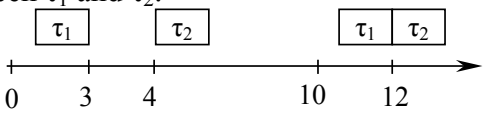


Figure 1. The execution of the two tasks τ_1 and τ_2.

Phasing of tasks and the start- and completion time variations are incorporated into the model by describing, for each instance of a task during the *lcm*, the earliest start time (*est*), the latest start time (*lst*), the earliest completion time (*ect*), and the latest completion time (*lct*). The constraints and notation are defined below, where τ_i represents task *i* and τ_i^n represents instance *n* of task *i*.

est(τ_i^n) - earliest start time of τ_i^n.

lst(τ_i^n) - latest start time of τ_i^n.

ect(τ_i^n) - earliest completion time of τ_i^n.

lct(τ_i^n) - latest completion time of τ_i^n.

Deadline <*deadline*, τ_i> holds iff

$$lct(\tau_i^n) - (periodTime(\tau_i) * n + offset(\tau_i)) \leq deadline$$

Precedence <τ_i, τ_j> holds iff

$$lct(\tau_i^n) \leq est(\tau_j^n)$$

Separation <*separation*, τ_i, τ_j> holds iff

$$est(\tau_j^n) - lct(\tau_i^n) \geq separation$$

Start Jitter <S_h, S_l, τ_i> holds iff

$$lst(\tau_i^{n+1}) - est(\tau_i^n) \le S_h \wedge est(\tau_i^{n+1}) - lst(\tau_i^n) \ge S_l$$

Completion Jitter $<C_h, C_l, \tau_i>$ holds iff

$$lct(\tau_i^{n+1}) - ect(\tau_i^n) \le C_h \wedge ect(\tau_i^{n+1}) - lct(\tau_i^n) \ge C_l$$

Latency $<latency, \tau_i, \tau_j>$ holds iff

$$T_i = T_j \rightarrow \forall n(lct(\tau_i^n) \le est(\tau_j^n) \wedge lct(\tau_j^n) - est(\tau_i^n) \le latency) \wedge$$

$$T_i > T_j \rightarrow \forall n \exists m : lct(\tau_i^n) \le est(\tau_j^m) \wedge lct(\tau_j^m) - est(\tau_i^n) \le latency \wedge$$

$$T_i < T_j \rightarrow \forall n \exists m : lct(\tau_i^m) \le est(\tau_j^n) \wedge lct(\tau_j^n) - est(\tau_i^m) \le latency$$

Correlation $<Correlation, \tau_i, \tau_{i+1}, \ldots, \tau_{i+m}>$ holds iff

$$\forall j,k \in i..(i+m) : \left| lst(\tau_j^n) - est(\tau_k^n) \right| \le Correlation$$

A sporadic task is specified by a worst-case execution time, a minimum inter-arrival time, and a deadline. Here, the best-case execution time is not considered, since the best case considering the entire task set is that the sporadic task is not activated at all at a given instance. The minimum inter-arrival time specifies the shortest possible time between two consecutive activations of the task. The deadline is relative to the release of the task. It is also possible for sporadic tasks to use semaphores that are shared with both sporadic and periodic tasks. Note that an interrupt should be modelled as a sporadic task.

3 Attribute Assignment

This section describes the algorithm for assigning priorities and offsets to the periodic tasks and priorities to sporadic tasks in order to meet the constraints specified for a task set. It is assumed that constraints are specified according to the model defined in the previous section. The heart of the attribute assignment is a genetic algorithm that assigns offsets and priorities, evaluates the assignments, and incrementally finds new assignments, thereby gradually achieving the required system behaviour. The general idea of a GA is to let individuals in a population gradually improve by the mechanisms of natural selection. In this case the individuals consists of attribute assignments for a tasks set and the environment to master is the constraints put on that task set. An overview of the structure and operation of the genetic algorithm used is given below.

1. Initial *Population* – The algorithm initially makes a number of guesses about the assignment of priorities and offsets for the complete task set. A complete assignment for the entire task set is referred to as a *genome*.
2. Apply *Objective function* – The objective function calculates a goodness value for each genome, given how far the genome is from meeting the requirements. If the objective is reached, the algorithm has found a solution and is terminated.
3. *Crossover* – In this step parts of different genomes are combined to produce an offspring, i.e., a new genome built from two other genomes.
4. *Mutation* – Randomly alters a genome by e.g., by reassigning a priority in the genome by a random number.
5. Repeat from step 2, each iteration is referred to as a *generation.*

An assignment of offsets and priorities for a task set is represented by a set of offset priority pairs for the periodic tasks and a priority for each sporadic task, e.g., a task set with periodic tasks t_1 to t_i and sporadic tasks st_1 to st_j is represented by the set g: $\{<priority_1, offset_1>,\ldots,<priority_i, offset_i>,<priority_1>,\ldots,<priority_j>\}$. The population of the genetic algorithm then consists of a number of such priority-offset sets $G = \{g_1, \ldots, g_n\}$.

The objective function calculates start times and completion times for the task set and derives a single value used for sorting different genomes by their closeness to the optimum, where the representation of optimum is defined by the genetic algorithm, e.g., the lower value

the closer to optimal. The deviations from the requirements for a task set, using the offsets and priorities of a given genome, are calculated by rearranging the formulas earlier described in Section 2. For example, deviation from the distance constraint is calculated by reformulating $lct(\tau_j^n) - est(\tau_i^n) \geq dist$ as $dist - (lct(\tau_j^n) - est(\tau_i^n))$. The objective value is then expressed as a percentage of the allowed deviation, e.g., $dist - (lct(\tau_j^n) - est(\tau_i^n)) \ / \ dist$. This value is divided by the number of instances, during an *lcm*, of the task. The division by *dist* and the number of instances is done in order to normalise the value against other constraints so that not too strong emphasis is put on some constraints. The objective value for a genome is the sum of the normalised values calculated for each constraint. The objective function is at the end of this section.

The analysis performed to calculate the earliest and latest start times and completion times for the instances of the task set can be divided into two cases: 1) The earliest start time and completion time are calculated disregarding the sporadic tasks, using the best-case execution times, and assuming that no tasks are blocked when using shared resources. 2) The latest start time and completion time is calculated considering interference from sporadic tasks, using the worst-case execution times and assuming maximal blocking.

In the objective function given below, task instances are assumed to be enumerated starting with zero for the first instance. The function numInst() returns the number of instances for a task during the *lcm* of the complete task set. The objective function is executed for each of the attribute assignments contained in the GA population. For each assignment an objective value is returned, and that value is then used to rank the different attribute assignments for a given task set. Pessimism in the objective function can be reduced if the mechanism of the used run-time system is considered. For example, in pre-emptive priority based systems, tasks with the same offsets are not influenced by sporadic activities independently of each other. The goal of this work has not been to provide an optimal objective function, the goal has been focused on the overall success of the method, which is indicated by the simulations in section 4.

Objective function

objective = 0

for each task τ_i "Deadline"

for each instance *n* of task τ_i

if $lct(\tau_i^n) > deadline(\tau_i) + n \cdot periodTime(\tau_i) + offset(\tau_i)$

$deviation = lct(\tau_i^n) - deadline(\tau_i) - n \cdot periodTime(\tau_i) - offset(\tau_i)$

$objective = objective + deviation \ / \ deadline(\tau_i) \ /$ numInst(τ_i)

for each *precedence constraint* $<\tau_i , \tau_j>$

for each instance *n* of task τ_i and τ_j

if $lct(\tau_i^n) > est(\tau_j^n)$

$deviation = 1$

$objective = objective + deviation \ /$ numInst(τ_i)

for each *separation constraint* $<separation, \tau_i , \tau_j>$

for each instance *n* of task τ_i and τ_j

if $est(\tau_j^n) - lct(\tau_i^n) < separation$

$deviation = separation - (est(\tau_j^n) - lct(\tau_i^n))$

$objective = objective + deviation \ / \ separation \ /$ numInst(τ_i)

for each *start jitter constraint* $<S_h, S_l \ \tau_i>$

for each instance *n* of task τ_i

if $lst(\tau_i^{n+1}) - est(\tau_i^n) > S_h$ *"(lcm + $lst(\tau_i^0)$) – $est(\tau_i^n)$ when n=numInst$_i$-1"*

$deviation = (lst(\tau_i^{n+1}) - est(\tau_i^n)) - S_h$

$objective = objective + deviation / S_h / 2 /$ numInst(τ_i)

if $est(\tau_i^{n+1}) - lst(\tau_i^n) < S_l$ *"(lcm + $est(\tau_i^0)$) – $lst(\tau_i^n)$ when n=numInst$_i$-1"*

$deviation = S_l - (est(\tau_i^{n+1}) - lst(\tau_i^n))$

$objective = objective + deviation / S_l / 2 /$ numInst(τ_i)

Completion Jitter calculated as start jitter above.

for each *latency constraint* <*latency*, τ_i , τ_j >

if $periodTime(\tau_i) = periodTime(\tau_j)$

for each instance *n* of task τ_i and τ_j

if $lct(\tau_i^n) \leq est(\tau_j^n)$

If $lct(\tau_j^n) - est(\tau_i^n) > latency$

$deviation = (lct(\tau_j^n) - est(\tau_i^n)) - latency$

$objective = objective + deviation / latency /$ numInst(τ_i)

else

$objective = objective + 1 /$ numInst(τ_i)

if $periodTime(\tau_i) > periodTime(\tau_j)$

for each instance *n* of task τ_I

find the instance τ_j^m *with earliest lct, where* $lct(\tau_i^n) \leq est(\tau_j^m)$

if an instance τ_j^m *is found*

if $lct(\tau_j^m) - est(\tau_i^n) > latency$

$deviation = (lct(\tau_j^m) - est(\tau_i^n)) - latency$

$objective = objective + deviation / latency /$ numInst(τ_i)

else

$objective = objective + 1 /$ numInst(τ_i)

if $periodTime(\tau_i) < periodTime(\tau_j)$

Analogous to $periodTime(\tau_i) > periodTime(\tau_j)$

for each *correlation constraint* <*correlation*, τ_i, …, τ_m >

for each instance *n* of task τ_i to τ_m

find the maximum difference between $lst(\tau_k^n) - est(\tau_l^n)$ for
any k and l

in [i..m] where $k \neq l$.

if $lst(\tau_k^n) - est(\tau_l^n) > correlation$

$deviation = lst(\tau_k^n) - est(\tau_l^n) - correlation$

$objective$ = objective + $deviation / correlation / numInst(\tau_i)$

End Objective function

4 Results

A series of simulations have been carried out to evaluate the performance of the proposed method. The first set of simulations shows the success ratio, for the GA, at assigning priorities and offsets to task sets so that the constraints for the task sets are met. For this set of simulations all the constraints presented in this paper are used by the generated task sets. The second set of simulations compares the method in this paper to the algorithm presented in [2] by Bate and Burns. To be able to compare the two approaches the constraints not supported by the algorithm in [2] have been removed.

4.1 Simulation set up

Periodic and sporadic tasks are randomly generated until the specified utilization is reached. The periodic utilization is randomly generated in [0, U], and the sporadic utilization equaling U – periodic utilization, where U is the desired system utilization. The period time of the periodic tasks are randomly selected from a number of predefined period times and the WCET is randomly generated as a percentage of the period time, the percentage is specified as a range, e.g., 2%-4% of the period time. The BCET for a periodic task is defined as a percentage of the WCET of the task. Table 2 displays the numbers used for the periodic tasks in the simulations presented in this paper. The minimum inter-arrival time for sporadic tasks are randomly generated from a predefined set of ranges and the WCET is generated in the same way as for the periodic tasks. The parameters for the sporadic tasks in the simulations are given in table 1.

Min. Inter-arrival time	Distribution %
[0,1000]	20
[1000, 5000]	70
[5000,20000]	10
WCET in percentage of min. inter-arrival time	
[0,1]	30
[1,2]	40
[2,5]	30

Table 1. Data for generating sporadic tasks.

Period time	Distribution %
10000	20
25000	20
50000	40
100000	20
WCET in percentage of period time	
[0,2]	45
[2,4]	50
[4,8]	5
BCET in percentage of WCET	
[0,70]	10
[70,80]	30
[80,90]	30
[90,97]	30

Table 2. Data for generating periodic tasks.

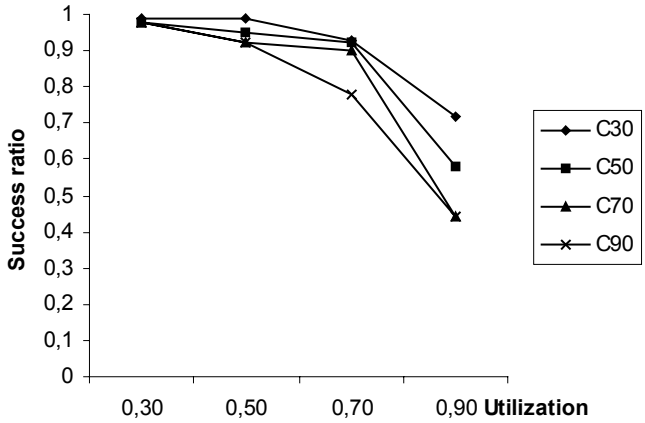
When the tasks with period times, WCETs, and BCETs have been generated, offsets and priorities are randomly generated. The temporal behavior of the task set is then analyzed and the constraints are generated based on the temporal information so that the task set fulfils the constraints. The number of constraints generated are based on a predefined percentage of the number of periodic tasks, e.g., if 70% is specified for the amount of constraints and the number of periodic tasks are 20, then there will be 14 tasks involved in the constraints. Deadlines are not included in this number since a deadline is always randomly generated for each task.

The simulations are performed for four different utilization levels, 30%, 50%, 70%, and 90%. For each utilization level simulations are carried out for different amounts of constraints, 30%, 50%, 70%, and 90%, where the constraints are generated as presented above. There are 100 task sets generated for each utilization and constraints level. The simulations were run on a 550 MHz Pentium III processor with 128 MB RAM. The software is implemented using C/C++. For the basic data structures and operations of the genetic algorithm we used the GAlib genetic algorithm package, written by Matthew Wall at the Massachusetts Institute of Technology. The GA will terminate on success, or when reaching 2000 generations, or when there is no improvement in the objective value for 100 generations.

4.2 Success ratio for the GA

For this simulation all the constraints presented in this paper are used by the generated task sets as well as shared resources between periodic and sporadic tasks (and other combinations). There is an even distribution between the numbers of tasks assigned to the different constraints in order to get a good coverage of all the constraints. Start and completion jitter is

not treated as two separate constraints when the amount of constraints is calculated. If a jitter constraint is generated, both start and completion jitter is generated for the task and it is viewed as one task with one constraint. The graph shows the percentage of task sets with attributes assigned that fully meet the constraints. The average number of sporadic and periodic tasks, as well as the total number of tasks is given in Table 3. Finally, Table 4 shows the average computation time for processing the task sets, including computation times for both correct and incorrect assignments.



Graph 1. The success ratio of the GA algorithm for different load and number of constraints.

Constraints / Utilization	30%	50%	70%	90%
30%	11.7/5.8/5.9	19.3/9.3/10.0	26.9/12.6/14.2	34.6/16.3/18.3
50%	11.8/6.1/5.8	19.3/9.4/9.8	27.0/12.7/14.34	34.4/16.5/17.9
70%	11.6/5.5/6.2	19.7/9.8/9.9	26.2/14.1/12.3	34.4/16.8/17.6
90%	11.6/5.6/6.0	19.0/10.2/8.8	27.2/13.2/14.0	33.8/16.3/17.5

Table 3. The average number of tasks for the simulation, displayed as total/periodic/sporadic

Constraints / Utilization	30%	50%	70%	90%
30%	1	1	1	1
50%	10	12	18	17
70%	98	128	124	132
90%	699	792	882	774

Table 4. The average computation time in seconds.

The simulations indicate that the method solves the attribute assignment to a high degree and that the success ratio decreases as the utilization and the number of constraints increases. The number of tasks given by the second graph shows that the results are valid for fairly large embedded systems, while the computation time given in the last graph indicates that for significantly larger task sets, in terms of number of tasks, with high utilization the computation times may be too long to be practical. The computation time of the algorithms is

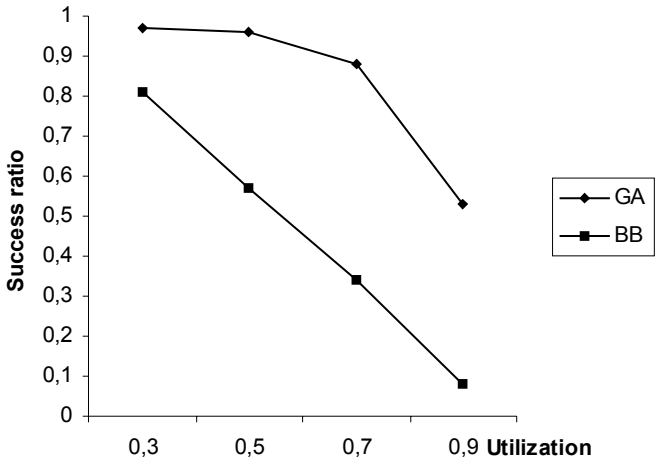
mainly related to the number of tasks and not so strongly to the number of constraints because the analysis is the computationally most demanding part. Although, a large number of constraints may increase the computation time by requiring more generations of the GA, and thereby more analysis, to find a solution. The correlation between the number of constraints and the computation time is also dependent on the termination criteria of the GA. Since the GA terminates on a given number of iterations and when improvement of the algorithm is to slow, large task set with many constraints will to a higher degree be stopped by the termination criteria compared to a large task set with few constraints. Thus, the computation time is kept down at the cost of a lower success ratio. Relaxing the termination criteria would increase computation times but also most likely the success ratio.

4.3 Comparison with Bate and Burns' method

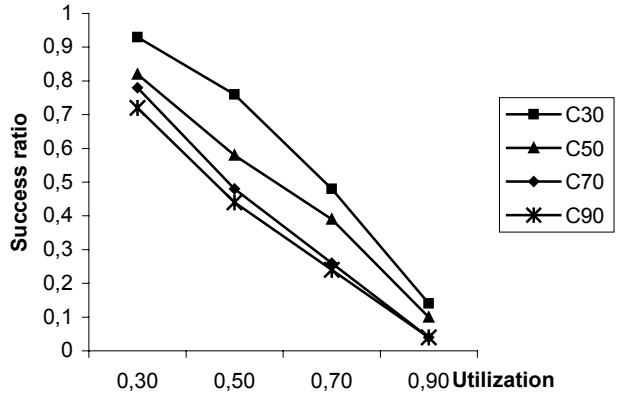
In this section we compare the performance of our method with that of the method presented by Bate and Burns' [2]. Since their method handles less general constraints, we will here use a restricted simulation set-up, in that we have to exclude start jitter, correlation, and shared resources from the task model.

There is an even distribution between the number of tasks assigned to jitter constraints and the number of tasks assigned to separation and latency constraints. The ratio between tasks with latency compared to separation constraints will be 4/1. The ratio is set to reflect an assumed higher number of latency constraints than separation constraints in real systems; this assumption is based on many years industrial experience gained by the authors.

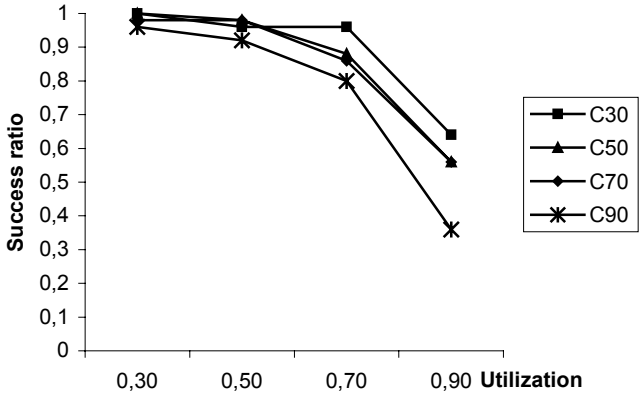
The first graph for this set of simulations (graph 2) shows, for both algorithms, the percentage of task sets with attributes assigned that fully meet the constraints. The success ratio depicted is the average ratio for all constraint levels. The following two graphs (graphs 3 and 4) give the success ratio for each constraint level. In the graphs below, the GA algorithm is denoted GA and the algorithm in [2] is denoted BB.



Graph 2. The average success ratio.



Graph 3. Success ratio for the BB algorithm.



Graph 4. Success ratios for the GA algorithm.

The average numbers of tasks are close to the numbers for the first simulation, given in Table 3. The computation times for the GA algorithm are in the same magnitude as those given in Table 4, while the algorithm in [2] has computation times of at most a few milliseconds.
Graph 2, displaying the success ratio for the two algorithms, shows that the algorithm presented in this paper performs well compared to the algorithm in [2], especially for task sets with high utilization. The difference in success ratio depends on that the GA performs global optimization of the attributes, distributing the tasks execution, using offsets, as needed for meeting the constraints, while the algorithm in [2] does not globally optimize the attribute assignment, but assign attributes based on the properties of each constraint individually. As the utilization increases the effect of using global as opposed to local optimization becomes more apparent.

5 Conclusion

The problem of assigning priorities and offsets to tasks from a specification model supporting complex timing constraints is an important part in the implementation of real-time control systems that consist of a number of periodic control activities executing with different frequencies while exchanging data. Such control systems are for instance common in motion control algorithms. Sampled data control applications, in general, are real-time systems that are sensitive to deviations from nominal deterministic timing, i.e. the timing that normally is assumed in control design. Since an implementation of a computer control system inevitably introduces time-delays and time-variations, it is important to investigate the sensitivity of a control system to such “timing disturbances” during the control engineering phase. Moreover, timing tolerances together with other timing requirements must be clearly communicated from

the design phase (presumably carried out by control engineers) to the implementation phase (presumably carried out by computer engineers). Further, many motion control applications are used in safety critical contexts, and/or environments where high reliability and availability are required. This emphasises the need for analysis of the correctness of the computer control system prior to implementation.

In this paper we propose a method for fulfilling complex temporal requirements by assigning priorities and offsets to tasks, running on a standard commercial RTOS. The method uses a genetic algorithm to search for an acceptable solution, i.e., a solution satisfying all constraints. However, even in cases when an acceptable solution cannot be found, the genetic algorithm will provide a near optimal solution indicating which constraints that are difficult (or impossible) to satisfy). This is important from an engineering perspective, since the result can be used as input for remodelling of the application.

Results from simulation shows that the algorithm presented in this paper has a high success ratio in assigning attributes that make schedulable task sets meeting their constraints. Moreover, in comparison to the algorithm in [2] the GA algorithm performs well, with noticeable higher success ratio. The computation times for the GA algorithm is considerable much longer than for the algorithm in [2], which handles a substantially simpler task model. However, the simulation results show that the proposed method efficiently assigns attributes for task sets of a size that covers most embedded control systems, in reasonable time for an off-line tool.

The method proposed in this paper supports specification of jitter constraints for both task start and completion times, making it possible to more precisely control the variations in period time of a task. More over, separation constraints, and latency constraints are supported, as are correlation constraints between tasks executing in parallel. The varying execution times of tasks are supported as well as shared resources between sporadic and period tasks.

Finally, future work includes adding optimisation on other criteria, e.g., minimisation of the number of used offset and priority levels, and general minimisation of e.g., jitter, in order to have as low jitter as possible in the system. Due to the architecture of the GA it is easy to add new optimisations as the ones listed above, and since it only requires additional functionality in the objective function it is cheap in terms of computation time.

6 References

[1] Törngren M. Fundamentals of implementing real-time control applications in Distributed computer systems. J. Of Real-Time Systems, 14, 219-250, Kluwer Academic Publishers, 1998.

[2] Bate I. and Burns A. An Approach to Task Attribute Assignment for Uniprocessor Systems. In Proc. 11th Euromicro Conference on Real-Time Systems (ECRTS99), York, England, June 9-11, 1999, IEEE Computer Society.

[3] Mok A. K., Tsou D., and De Rooij R. C. M. The MSP.RTL Real-Time Scheduler Synthesis Tool. In Proc. 17th IEEE Real-Time Systems Symposium, pp. 118-128. IEEE Computer Society.

[4] Würtz J. and Schild K. Scheduling of Time-Triggered Real-Time Systems, In Constraints, pp. 335-357, Oct, 2000. Kluwer Academic Publishers.

[5] Cheng S. T. and Agrawala A. K. Allocation and Scheduling of Real-Time Periodic Tasks with Relative Timing Constraints. Second International Workshop on Real-Time Computing Systems and Applications (RTCSA'95) October 25-27, 1995.

[6] Radu Dobrin, Gerhard Fohler, Peter Puschner. Translating Off-line Schedules into Task Attributes for Fixed Priority Scheduling. In Proceedings of *Real-Time Systems Symposium* London, UK, December 2001.

[7] Radu Dobrin, Yusuf Özdemir, Gerhard Fohler. Task Attribute Assignment of Fixed Priority Scheduled Tasks to Reenact Off-Line Schedules. In *Proceedings of RTCSA 2000* Korea , December 2000.

[8] Gerber R., Saksena M, and Hong S. Guaranteeing Real-Time Requirements with Resource-Based Calibration of Periodic Processes. IEEE Transactions on Software Engineering, 21(7), July 1995.

[9] F. Coutinho, J.A. Fonseca, J. Barreiros, E. Costa. Jitter Minimisation with Genetic Algorithms, Proceedings 3rd IEEE International Workshop on Factory Communication Systems, Portugal, September 2000.

[10] Norström C., Gustafsson M, Sandström K., Mäki-Turja J, Bånkestad N. Experiences from Introducing State-of-the-art Real-Time Techniques in the Automotive Industry, In Proc. *Eigth IEEE International Conference and Workshop on the Engineering of Compute-Based Systems* Washington, US , April 2001. IEEE Computer Society

[11] Zomaya A. Y., Ward C., and Macey B. Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues. In IEEE Transaction on Parallel and Distributed Systems, VOL. 10, NO 8, August 1999.

[12] Corrêa R. C., Ferreira A., and Rebreyend P. Scheduling Multiprocessor Tasks with Genetic Algorithms. In IEEE Transactions on Parallel and Distributed systems, VOL 10, NO. 8, August 1999.

[13] Grajcar M. Genetic List Scheduling Algorithm for Scheduling and Allocation on a Loosely Coupled Heterogeneous Multiprocessor System. In Proc. 36th Design Automation Conference (DAC), p. 280-285, New Orleans, 1999. ACM Press.

[14] Faucou S., Déplanche A., and Beauvais J. Heuristic Techniques for Allocating and Scheduling Communicating Periodic Tasks in Distributed Real-Time Systems. In 3rd IEEE International Workshop on Factory Communication Systems, September 6, 2000. IEEE Industrial Electronics Society.

[15] Törngren M. Modelling and Design of Distributed Real-time Control Applications. PhD thesis, Dept. of Machine Design, The Royal Institute of Technology, Stockholm, Sweden, 1995.

Appendix A: Example

In this example we assume an application for which constraints needed for implementation have been derived from a control design and expressed according to the specification model described in Section 2. The example system consists of 4 periodic tasks and 1 sporadic task. In Table 5 the periodic tasks are listed.

Task	Wcet	Bcet	Period time
A	2	2	20
B	3	3	20
C	2	2	20
D	3	3	20

Table 5: The example task set.

In addition there is one sporadic task, *SP*, in the system. The sporadic task *SP* has a *Wcet* of 2 and a *minimum inter-arrival time* of 9. Furthermore, task *SP* has a *deadline* constraint of 6. The constraints that apply to the periodic task set are given below.

Start Jitter, <21, 19, A>
Start Jitter, <21, 19, C>
Latency, <9, A, B >
Separation <4, C, D >

Given the specification above, the GA tries to find an attribute assignment in the run-time model such that the execution of the task set fulfils the constraints. The termination criterion of the GA is that the objective function evaluates to zero, i.e., one genome meets all the constraints.

The offset and priority for a periodic task τ_i is represented by the tuple op_i = <offset, priority> and the priority for a sporadic task is represented by p_j = <priority>. The complete representation for the task set of the example is op_A, op_B, op_C, op_D, p_{SP}. A high value represent a high priority.

The size of the population of the genetic algorithm is 3 and the number of offspring generated in each generation is 2. The initial population is given in table 6.

Genom	op_A	op_B	op_C	op_D	p_{SP}
1	<0,2	<13,1	<0,5>	<1,4>	<3>
2	<2,4	<6,2>	<9,3>	<15,1	<5>
3	<1,4	<9,5>	<0,4>	<14,1	<3>

Table 6. The initial population.

According to the operation of the GA the next step is to apply the objective function to the genomes of the population. Included in this is to make analysis of each genome.

In Figure 2 the worst and best case scenarios' for the task set are displayed, assuming offset and priorities according to genome 1.

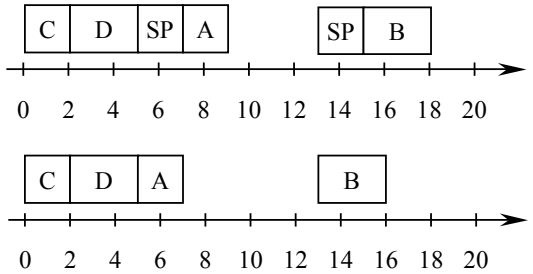


Figure 2. The worst and best case execution scenario.

Table 7 shows the start times and completion times for the tasks.

Task	*lst*	*lct*	*est*	*ect*
A	7	9	5	7
B	15	18	13	16
C	0	2	0	2
D	2	5	2	5
SP	-	7	-	-

Table 7. Start and completion of the tasks.

The result of the objective function, presented in Section 3, for genome 1 can then be calculated as:

Start Jitter, <21, 19, A>

$deviation = (lst(\tau_i^{n+1}) - est(\tau_i^n)) - S_h = (20+7) - 5 - 21 = 1$

$objective = objective + deviation / S_h / 2 / numInst(\tau_i) =$

$= 0 + 1/21/2/1 = 0{,}024$

$deviation = S_l - (est(\tau_i^{n+1}) - lst(\tau_i^n)) = 19 - (25-7) = 1$

$objective = objective + deviation / S_l / 2 / numInst(\tau_i) =$

$= 0{,}024 + 1/19/2/1 = 0{,}05$

Start Jitter, <21, 19, C>

$(lst(\tau_i^{n+1}) - est(\tau_i^n)) < S_h$ *"**Constraint met**"*

$S_l < (est(\tau_i^{n+1}) - lst(\tau_i^n))$ *"**Constraint met**"*

Latency, <9, A, B >

$deviation = (lct(\tau_j^n) - est(\tau_i^n)) - latency = 18 - 5 - 9 = 4$

$objective = 0{,}05 + 4 / 9 = 0{,}49$

Separation <4, C, D >

$deviation = separation - (est(\tau_j^n) - lct(\tau_i^n)) = 4 - (2-2) = 4$

$objective = 0{,}49 + 4 / 4 = 1{,}49$

Deadline <6, SP>

$deviation = lct(\tau_i^n) - deadline = 7 - 6 = 1$

$objective = 1{,}49 + 1 / 7 = 1{,}63$

Objective: 1,63

In Figure 3 the worst and best case scenarios' for the task set are displayed, assuming offset and priorities according to genome 2.

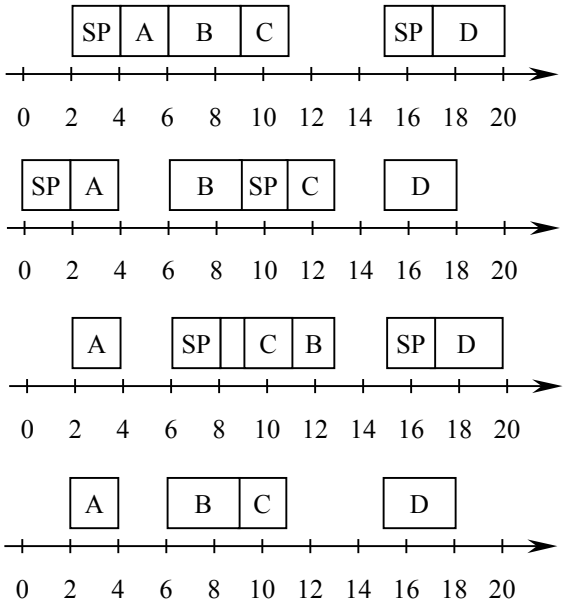


Figure 3. The worst and best case execution scenario.

Table 8 shows the start times and completion times for the tasks.

Task	*lst*	*lct*	*est*	*ect*
A	4	6	2	4
B	8	13	6	9
C	11	13	9	11
D	17	20	15	18
SP	-	2	-	-

Table 8. Start and completion of the tasks.

The result of the objective function for genome 2 can then be calculated as:

Start Jitter, <21, 19, A>

$deviation = (lst(\tau_i^{n+1}) - est(\tau_i^n)) - S_h = 24 - 2 - 21 = 1$

$objective = 0 + 1/21/2 = 0{,}024$

$deviation = S_l - (est(\tau_i^{n+1}) - lst(\tau_i^n)) = 19 - (22\text{-}4) = 1$

$objective = 0{,}024 + 1/19/2 = 0{,}05$

Start Jitter, <21, 19, C>

$deviation = (lst(\tau_i^{n+1}) - est(\tau_i^n)) - S_h = 31 - 9 - 21 = 1$

$objective = 0{,}05 + 1/21/2 = 0{,}074$

$deviation = S_l - (est(\tau_i^{n+1}) - lst(\tau_i^n)) = 19 - (22\text{-}4) = 1$

$objective = 0{,}074 + 1/19/2 = 0{,}1$

Latency, <9, A, B >

$deviation = (lct(\tau_j^n) - est(\tau_i^n)) - latency = 13 - 2 - 9 = 2$

$objective = 0{,}1 + 2 / 9 = 0{,}32$

Separation <4, C, D >

$deviation = separation - (est(\tau_j^n) - lct(\tau_i^n)) = 4 - (15\text{-}11) = 0$

$objective = 0{,}32 + 0 / 4 = 0{,}32$

Deadline <6, SP>

$lct(\tau_i^n) <= deadline$ *"**Constraint met**"*

Objective: 0,32

In Figure 4 the worst and best case scenarios' for the task set are displayed, assuming offset and priorities according to genome 3.

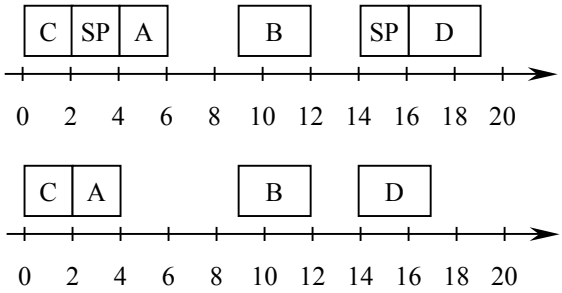


Figure 4. The worst and best case execution scenario.

Table 9 shows the start times and completion times for the tasks.

Task	*lst*	*lct*	*est*	*ect*
A	4	6	2	4
B	9	12	9	12
C	0	2	0	2
D	16	19	14	17
SP	-	5	-	-

Table 9. Start and completion of the tasks.

The result of the objective function for genome 3 can then be calculated as:

Start Jitter, <21, 19, A>

$deviation = (lst(\tau_i^{n+1}) - est(\tau_i^n)) - S_h = 24 - 2 - 21 = 1$

$objective = 0 + 1/21/2 = 0{,}024$

$deviation = S_l - (est(\tau_i^{n+1}) - lst(\tau_i^n)) = 19 - (22\text{-}4) = 1$

$objective = 0{,}024 + 1/19/2 = 0{,}05$

Start Jitter, <21, 19, C>

$(lst(\tau_i^{n+1}) - est(\tau_i^n)) < S_h$ *"**Constraint met**"*

$S_l < (est(\tau_i^{n+1}) - lst(\tau_i^n))$ *"**Constraint met**"*

Latency, <9, A, B >

$deviation = (lct(\tau_j^n) - est(\tau_i^n)) - latency = 12 - 2 - 9 = 1$

$objective = 0{,}05 + 1 / 9 = 0{,}16$

Separation <4, C, D >

$separation < (est(\tau_j^n) - lct(\tau_i^n))$ *"**Constraint met**"*

Deadline <6, SP>

$lct(\tau_i^n) < deadline$ *"**Constraint met**"*

Objective: 0,16

Since no genome meets the termination criteria the GA proceeds with the next step, i.e., to generate offspring from the population. In this example we use one point crossover. In the crossover operation a new genome is formed by selecting a position in the genome, taking the information to the left of that position from one genome, and combine it with the information to the right of that position from another genome. In the example the crossover results in two new genomes. By combining op_A and op_B from genome 2 with op_C, op_D, and p_{SP} from genome 1 the first genome is produced. The second genome is produced by combining op_A and op_B from genome 2 with op_C, op_D, and p_{SP} from genome 3. Table 10 display the new genomes.

Genom	op_A	op_B	op_C	op_D	p_{SP}
4	<2,4	<6,2>	<0,5>	<1,4>	<3>
5	<2,4	<6,2>	<0,4>	<14,1	<3>

Table 10. The new genomes.

The next step is to apply a random mutation to the new genomes with some probability. In the example op_A of genome 4 is changed from <2,4> to <4,4> by mutation.
In Figure 5 the worst and best case scenarios' for the task set are displayed, assuming offset and priorities according to genome 4.

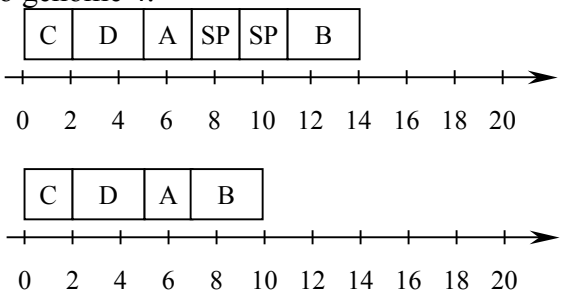


Figure 5. The worst and best case execution scenario.

Table 11 shows the start times and completion times for the tasks.

Task	*lst*	*lct*	*est*	*ect*
A	5	7	5	7
B	11	14	7	10
C	0	2	0	2
D	2	5	2	5
SP	-	9	-	-

Table 11. Start and completion of the tasks.

The result of the objective function for genome 4 can then be calculated as:

Start Jitter, <21, 19, A>

$(lst(\tau_i^{n+1}) - est(\tau_i^n)) < S_h$ *"**Constraint met**"*

$S_l < (est(\tau_i^{n+1}) - lst(\tau_i^n))$ *"**Constraint met**"*

Start Jitter, <21, 19, C>

$(lst(\tau_i^{n+1}) - est(\tau_i^n)) < S_h$ *"**Constraint met**"*

$S_l < (est(\tau_i^{n+1}) - lst(\tau_i^n))$ *"**Constraint met**"*

Latency, <9, A, B >

$deviation = (lct(\tau_j^n) - est(\tau_i^n)) - latency = 14 - 5 - 9 = 0$

objective = 0 + 0 / 9 = 0

Separation <4, C, D >

deviation = separation – (est(τ_j^n) – lct(τ_i^n)) = 4 – (2-2) = 4

objective = 0 + 4 / 4 = 1

Deadline <6, SP>

deviation = lct(τ_i^n) - deadline = 9 – 6 = 3

objective = 1 + 3 / 6 = 1,5

Objective: 1,5

In Figure 6 the worst and best case scenarios' for the task set are displayed, assuming offset and priorities according to genome 5.

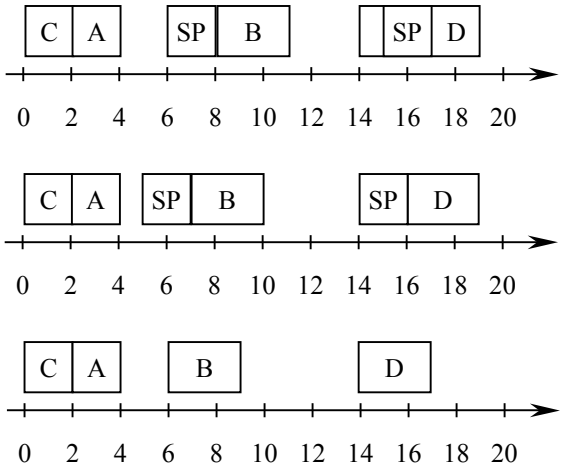


Figure 6. The worst and best case execution scenario.

Table 12 shows the start times and completion times for the tasks.

Task	*lst*	*lct*	*est*	*ect*
A	2	4	2	4
B	8	11	6	9
C	0	2	0	2
D	16	19	14	17
SP	-	6	-	-

Table 12. Start and completion of the tasks.

The result of the objective function for genome 5 can then be calculated as:

Start Jitter, <21, 19, A>

(lst(τ_i^{n+1}) – est(τ_i^n)) < S_h *"**Constraint met**"*

S_l < (est(τ_i^{n+1}) – lst(τ_i^n)) *"**Constraint met**"*

Start Jitter, <21, 19, C>

(lst(τ_i^{n+1}) – est(τ_i^n)) < S_h *"**Constraint met**"*

S_l < (est(τ_i^{n+1}) – lst(τ_i^n)) *"**Constraint met**"*

Latency, <9, A, B >

deviation = (lct(τ_j^n) - est(τ_i^n)) – latency = 11 – 2 – 9 = 0

objective = 0 + 0 / 9 = 0
Separation <4, C, D >
*separation < (est(τ_j^n) – lct(τ_i^n)) "**Constraint met**"*

Deadline <6, SP>

*lct(τ_i^n) <= deadline "**Constraint met**"*

Objective: 0

The best genomes are selected and the resulting population with their respective objective value is given in Table 13.

Genom	op_A	op_B	op_C	op_D	p_{SP}	*value*
5	<0,2	<13,1	<0,5>	<1,4>	<3>	0
2	<2,4	<6,2>	<9,3>	<15,1	<5>	0,16
3	<1,4	<9,5>	<0,4>	<14,1	<3>	0,49

Table 13. The resulting population.

As the best genome meets the termination criterion the GA is terminated. We have found an offset and priority assignment that fulfil all the temporal constraints.