# A scatter search algorithm for real-time scheduling with timing, precedence and exclusion constraints

**Article** *in* International Journal of Advanced Operations Management · January 2013

DOI: 10.1504/IJAOM.2013.053531

CITATIONS
0

READS
154

**3 authors**, including:

Adel Bouridah
Centre universitaire de Mila
**6** PUBLICATIONS **15** CITATIONS
SEE PROFILE

Habiba Drias
University of Science and Technology Houari Boumediene
**235** PUBLICATIONS **1,676** CITATIONS
SEE PROFILE

# A scatter search algorithm for real-time scheduling with timing, precedence and exclusion constraints

Adel Bouridah*

Mila University Center,
Mila, 43000, Algeria
E-mail: a.bouridah@centre-univ-mila.dz
*Corresponding author

Habiba Drias

LRIA Laboratory,
Faculty of Computer Science,
USTHB University,
16111 El-Alia, Bab-Ezzouar, Algiers, Algeria
E-mail: h_drias@hotmail.fr

Yacine Laalaoui

Department of Postgraduate Studies,
National Computer Science School,
16000 Oued-Smar, Algiers, Algeria
E-mail: yacine.laalaoui@gmail.com
E-mail: y_laalaoui@esi.dz

**Abstract:** Tasks scheduling is one of the most important challenges in embedded hard-real-time systems. The problem is known to be NP-hard and exhaustive search algorithms have no significant benefit in large-scale context. This paper proposes a scatter search-based approach for mono-processor systems with timing, precedence and exclusion constraints with no pre-emption. An empirical study is undertaken and comparison is done with results of other algorithms.

**Keywords:** real-time; pre-run-time scheduling; meta-heuristic; scatter search.

**Reference** to this paper should be made as follows: Bouridah, A., Drias, H. and Laalaoui, Y. (2013) 'A scatter search algorithm for real-time scheduling with timing, precedence and exclusion constraints', *Int. J. Advanced Operations Management*, Vol. 5, No. 2, pp.181–197.

**Biographical notes:** Adel Bouridah has received his BSc and MSc from the National High School of Computer Science (Ex. INI) of Algeria in 2005 and 2008 respectively. Currently, he is a PhD student at the same school. His main research interests span artificial intelligence field with a focus on meta-heuristics, scheduling under timing constraints and intrusion detection problem.

Habiba Drias earned his Master in Computer Science from Case Western Reserve University, Cleveland, USA in 1984 and doctorate prepared at Paris 6 University from Algiers USTHB University in 1993. She has directed the Computer Science Institute of USTHB, the Laboratory of Research in Artificial Intelligence (LRIA) and the National High School of Computer Science (Ex. INI) for several years. She has more than 50 published papers in the domains of AI, e-commerce, computational complexity and the satisfiability problem.

Yacine Laalaoui received his BSc, MSc and PhD from National High School of Computer Science (Ex. INI) of Algeria in 2002, 2005 and 2010 respectively. His main research interests span artificial intelligence field with a focus on meta-heuristics, constraints satisfaction problems (CSP), machine learning and scheduling under timing constraints.

This paper is a revised and expanded version of a paper entitled 'Scatter search for real-time scheduling with timing, precedence and exclusion constraints' presented at International Conference on Machine and Web Intelligence ICMWI 2010, Algiers, 3–5 October 2010.

## 1 Introduction

The technology of embedding hardware and software components is becoming ubiquitous. The presence of this technology varies from simple domestic devices to complex and critical applications. Usually, the latter systems are under stringent timing constraints (known also as real-time systems). Failure to satisfy specified timing constraints can lead to disastrous damage. In typical real-time system, there is a computing unit of one or more processors which is connected to several devices. A real-time program (known also as real-time task system) runs on the computing unit. It acquires sensors data to get information about the state of the process, computes these inputs, and then produces outputs to be sent to the devices throughout actuators. In the simplest case, devices produce inputs at regular intervals of time and outputs to be sent are also periodic. The real-time task system has to meet timing constraints in order to maintain the process in an acceptable state. The problem here is to find a schedule of tasks on one or more processors architecture such that all timing constraints will be met. Such a schedule is named a feasible schedule.

There are two classes of scheduling methods, online methods and offline methods. The validation of online methods is done using analytic conditions. Online methods are easy to implement but these methods are limited for simple real time systems with no shared resources and precedence constraints. Offline methods are applied in case of complex hard real-time systems with a great number of shared resources and precedence constraints. In offline methods, the system characteristics must be known in advance in order to produce a feasible schedule and the result will be saved in a data structure and consulted at run time by a dispatcher.

To produce a feasible schedule, one can use blind search methods like best-first-search and depth-first-search, or branch-bound-first methods (Xu and Parnas, 1992; Shepard and Gagne, 1991; Cavalcante, 1997). It is well known that all these methods have an exponential time complexity, because the general scheduling problem is NP-hard (Garey and Johnson, 1979). Consequently, results cannot be computed in a

reasonable time and search space. One can also use a learning-based algorithm to address the deadline scheduling problem (Laalaoui and Drias, 2009) or meta-heuristic like cooperative ants (Laalaoui et al., 2009). The use of such meta-heuristic was shown very helpful to handle the problem of time and search space and reduce those costs to reasonable values.

Scatter search is a population-based approach that has recently been shown to yield promising outcomes for solving combinatorial and non-linear optimisation problems. Based on formulations originally proposed in the 1960s for combining decision rules and problem constraints, scatter search uses strategies for combining solution vectors that have been proven effective in a variety of problem settings. Scatter search has given interesting results in solving many combinatorial optimisation problems like the Max-Sat problem (Drias, 2001), hard Max-Sat problem (Drias and Khabzaoui, 2001), DNA sequencing (Blazewicz et al., 2004), project scheduling (Debels et al., 2006) and other problems.

This paper addresses the problem of scheduling tasks with timing, precedence and exclusion constraints on single processor architecture with no pre-emption. The scheduling under study is considered as a combinatorial optimisation problem (COP) wherein feasible solutions are feasible schedules. Solutions are modelled as permutations of tasks (real permutation of segments) and the search space is composed of these permutations. During the search process, the scatter search uses only solutions that meet exclusion and precedence constraints to find the feasible solution in which all timing constraints are fulfilled.

The work presented in this paper aims at enhancing researches in pre-run-time scheduling methods in order to deal with more imposed constraints on complex real-time embedded systems like context switching minimisation, jitter minimisation (Dinatale and Stankovic, 1995), combining offline and priority-based scheduling methods (Dobrin, 2005; Xu and Lam, 1998).

## 2 Related works

Only few meta-heuristics including simulated annealing, genetic algorithms and ant colony optimisation have been devoted to real-time scheduling. The former was used by Tindell et al. (1992) to treat the problem of scheduling in distributed environment with a special communication protocol. It was also used by Dinatale and Stankovic (1995) to handle the problem in multiprocessor architecture with jitter minimisation and non-pre-emptive of tasks. Genetic algorithms were used by Nossal (1998) to tackle the problem of pre-emptive scheduling of inter-related tasks with precedence and exclusion constraints in extensible multiprocessor architecture. Navet and Migge (2003) have used genetic algorithms to solve a non-standard problem of real-time scheduling which is the assignment of policies (round Robin or FIFO scheduling) and priorities to tasks in POSIX1003.1b compliant systems. Also Laalaoui et al. (2009) have used ant colony system for mono-processor scheduling of hard real-time tasks with timing, precedence and exclusion constraints including pre-emption. To our knowledge no work has been done yet using scatter search to address the problem of scheduling hard real-time tasks in single processor architecture with timing precedence and exclusion relations, which is the main focus of the present paper. Scatter search is known to be more complicated to design than other meta-heuristics because it handles both the quality of solutions and

their dispersion in the search space. However, when well developed; scatter search can yield very interesting outcomes.

## 3 Real-time scheduling

### *3.1 The task model*

The task model we use is inspired from the task model of Liu and Layland (1973). A periodic task $\tau i$ is characterised by the tuple < *ri*, *Ci*, *Di*, *Pi* >, where *ri* is the first release time, *Ci* is the worst computation time, *Pi* is the period of activation and *Di*, called the deadline, is the amount of time given to the task to complete its execution. We note $di = ri + Di$, the moment when the task should be completed. It is assumed that $Di \geq Ci$, otherwise no feasible schedule exists. A sporadic task is not released at regular intervals of time, but the minimum duration between two requests is known in advance (Mok, 1983). In this paper, no sporadic tasks are considered because there are several methods to make an automatic transformation of sporadic tasks to periodic tasks (Xu, 2003). We assume that *ri*, *Ci*, *Di* and *Pi* as well as any other parameters expressed in time have integer values representing the number of time units. Processor resources are measured in terms of processor time units.

Because the periods of tasks are not necessary equal, more instances of tasks should be added in the schedule length period in order to study the scheduling feasibility over a relevant finite time interval. In our case, the schedule length period is the least common multiple of all periods of specified tasks (Xu and Parnas, 1993).

Since tasks can share resources and in order to identify time slices of using resources, some parameters must be added to the previous tuple as follows: $\tau i = < ri, Ci, Di, Pi, \{R1 < \alpha 1, \beta 1, \gamma 1 >, R2 < \alpha 2, \beta 2, \gamma 2 >, \ldots.\} >$,where *Rk* is the $k^{th}$ resource used in the task $\tau i$, $\alpha k$ is the time duration before the use of the resource *Rk*, $\beta k$ is the time duration of using the resource *Rk* and $\gamma k$ is the time interval after the use of the resource *Rk*. We note that for each *Rk*: $\alpha k + \beta k + \gamma k = Ci$. Before dealing with the problem of scheduling tasks, we need to introduce the following definitions and notations:

- Let *P* denote the set of all tasks.
- Each task $\tau i$ consists of a finite set of segments: *S*0, *S*1…*Sn*, where *Si* denotes its $i^{th}$ segment ($0 \leq i \leq n$).
- Some segments can together encapsulate a shared resource, and then constitute a critical section. Let *X* denotes the set of all critical sections.
- For each critical section $x \in X$, we note *x*0 and *xn* their first and last segments respectively.
- Let *Start*(*S*) be the start time of the segment *S*. It represents the moment when a segment *S* starts its execution.
- Let *End*(*S*) be the finished time of the segment *S* yielding the processor to another segment.
- Let *Lateness*(*S*) = *d*(*S*) – *End*(*S*), where *d*(*S*) is the deadline of the segment *S*.

### *3.2 Modelling a real time application*

As mentioned above, we consider only periodic tasks in our study. Our approach includes the following steps:

1 Identification of all critical sections of the real time application caused by sharing resources or precedence relations.

2 Divide each task of the real time application into several segments to simplify the definition of synchronisation constraints between critical sections.

3 For each task $\tau i$ that has $ri$ as its release time, $Di$ as its deadline time and constituted from the set of segments : $S_{i,1}, S_{i,2},\ldots,S_{i,n}$ where $C(S_{i,j})$ is the worst computation time of the $j^{th}$ segment of this set. The release time and the deadline time of each segment is defined as follows:

$$r\left(S_{i,j}\right) = r_i + \sum_{k=1}^{j-1} C\left(S_{i,k}\right) \text{ and } D\left(S_{i,j}\right) = Di - \sum_{k=j+1}^{n} C\left(S_{i,k}\right)$$

4 Redefining synchronisation constraints between segments or a set of segments to prevent shared resources from simultaneous access of concurrent tasks and to insure the correct order between dependent segments of the real time application.

5 Each solution of the search space is an ordered list of segments $(S_k, S_i,\ldots,S_n)$ which is called a permutation. Each segment of the real time application must appear in the permutation only once. A solution is feasible when all end executions time of segments are lower then their deadlines and all synchronisation constraints of the system are fulfilled.

#### *3.2.1 The synchronisation constraint*

We need to define exclusion constraints between segments or a set of segments to prevent shared resources from simultaneously accessing concurrent tasks and precedence constraints to insure the correct order between dependent segments of the real time application. The set of all constraints is denoted Ω.

##### *3.2.1.1 The exclusion constraints*

Exclusion constraints, denoted '⊗', are defined between critical sections of different tasks in order to specify mutual exclusion. For instance a critical section can contain one or more segments that share resources in read/write. If $x$, $y$ are two critical sections and the relationship: $x \otimes y$ is specified, we have to forbid any execution of any segment belonging to $y$ during the execution of any segment of $x$ and vice versa. Exclusion constraints are:

- Commutative: $x \otimes y = y \otimes x$.
- Distributed: if $x \otimes y$ and $x$ is composed of the segments $S_{x,1}, S_{x,2},\ldots, S_{x,i}$ and $y$ is composed of the segments $S_{y,1}, S_{y,2},\ldots,S_{y,j}$. So, $\{S_{x,1} \otimes S_{y,1}; S_{x,1} \otimes S_{y,2};\ldots; S_{x,i} \otimes S_{y,j}\}$ are all exclusion constraints for the system.

- We note that if $x \otimes y$ is specified, and $x$, $y$ are two critical section compounds with only one unit segment, so there is no need to specify the exclusion constraint because no pre-emption can occur during one processor time unit.

The set of all exclusion constraints is denoted EXCLUDES.

#### *3.2.1.2 The precedence constraints*

Precedence constraints, denoted '$\mapsto$', are defined between segments in order to insure the correct order of segments belonging to the same task and to force the producer/consumer paradigm between segments belonging to different tasks[1]. The key property of precedence constraints is the transitivity, i.e., if $S_1 \mapsto S_2$ and $S_2 \mapsto S_3$, then $S_1 \mapsto S_3$. The set of all precedence constraints is denoted PRECEDES.

#### *3.2.1.3 Optimisation criteria*

The objective of our algorithm is to find a feasible solution in which each segment must have a positive lateness. We define lateness of the segment $S_i$ as $Lateness(S_i) = d(S_i) - End(S_i)$.

The solution quality depends on the number of segments that have negative lateness. If $X$ is a solution in which $k$ segments do not respect their deadline and $F(X)$ the quality of $X$ then $F(X) = k$. The goal is to minimise $F(X)$ knowing that a solution is feasible when $F(X) = 0$.

## 4 The scheduling algorithm

### *4.1 Overview*

The proposed scheduling algorithm is an adaptation of scatter search defined in Glover (1998) to the problem of scheduling hard real-time tasks with no pre-emption for mono-processor environment. Let $S_1$, $S_2$,…,$S_n$ denote the set of all segments that compose our real-time task system. Really, our algorithm searches a feasible solution which is one of the permutations of the couple ($S_i$, start execution time of $S_i$). However, The start execution time of any segment is calculated automatically from the position of the segment in the solution as the maximal value between the end execution time of the segment that it precedes in the solution and its release time, i.e., $StartTime(S_i) = Max(EndTime(S_i - 1), r_i)$. Thus, permutations (solutions) are simplified as the permutations of segment. Any segment must appear only once in a solution.

The special issue of our algorithm is that the search space that it will really explore is not composed of all permutations possibilities (if we have $n$ segments so there will be n! possibilities) but only by permutations (solutions) that meet the exclusion and precedence constraints of the real time system. Therefore during the search, our algorithm never manipulates any solution that violates exclusion or precedence constraints.

### *4.2 Scatter search algorithm*

The general framework of our scatter search algorithm is defined as follows:

**Algorithm**

**Begin**

I ***Initialization phase***

1 Use the diversification generator to generate the initial population.

2 Deduct the reference set refset from the initial population.

II ***Evolution phase***

***While*** ((number of evaluated solutions < MaxSol) and (number of iterations < MaxIter)) ***do***

1 Generate subset from the reference set using the subset generating method.

2 Apply the combination method and put the result solutions in the pool of the combined solutions.

3 Apply the improvement method to the pool of the combined solution and put the result solutions in the pool of the improved solutions.

4 Update Refset using the improved solutions of this iteration.

***EndWhile***

**End.**

### *4.2.1 Generating the initial population*

In the initial phase of the algorithm, we need to construct a set of trial solutions that we call initial population. From these trial solutions, we deduce the reference set RefSet that will be used by the algorithm in the evolution phase. The initial population is constructed as follows:

1 generate a random trial solution called $s_{begin}$ that meets all exclusion and precedence constrains

2 apply the diversification generator on $s_{begin}$ to generate all trial solutions of the initial population.

### *4.2.2 Design of scatter search components for the scheduling problem*

#### *4.2.2.1 Diversification generator*

The diversification generator uses one seed solution to produce $k$ diverse solutions. All solutions generated with this generator must meet exclusion and precedence constraints so that the generator can check the eligibility of any segment before its insertion in any position of the solution which it tries to construct. When the diversification generator tries to build any trial solution, it may use the two lists named candidate list and admissible list in order to check the eligibility of segments. These two lists are defined as follows:

- *Candidate list:* This list is used to insure the satisfaction of precedence constraints. Initially, it contains the set of segments with no predecessors. At the end of execution of each segment $S_i$ belonging to the candidate list, this last segment $S_i$ is removed and all their successors are added to this list.

- *Admissible list:* This list is a subset of the candidate list. It contains only admissible segments which respect to all exclusion constraints besides the respect of precedence constraints. We propose three different diversification generators:
  1 *Random diversification generator:* This generator generates randomly diverse trail solutions but these solutions must meet the synchronisation constraints.
  2 *Diversification generator maximising distances:* This generator is inspired from the one described in Glover (1998) for the permutation problems. Assume that a given trial solution $C$ used as a seed is represented by indexing its segments such that they can appear in consecutive order, to yield $C = (S_1, S_2,\ldots,S_n)$. Define the subsequence $C_{(h:k)}$, where $k$ is a positive integer between 1 and $h$, to be given by $C_{(h:k)} = (S_k, S_{k+h}, S_{k+2h},\ldots,S_{k+rh})$, where $r$ is the largest non-negative integer such that $k + rh \leq n$. Then define the permutation $C(h)$, for $h \leq n$, to be $C(h) = (C_{(h:h)}, C_{(h:h-1)},\ldots,C_{(h:1)})$.
  3 *Diversification generator using mutation:* This generator makes a mutation between two positions, $i$ and $j$, drawn randomly in the seed solution but the result solution must meet the synchronisation constraints.

#### *4.2.2.2 Improvement method*

The improvement method of scatter search enables local search to improve the quality of the seed solution. For this purpose this method tries to reduce the number of segments which do not meet their deadline by shifting them to the left of the solution. While the improvement method makes shifting, it may not falsify the synchronisation constraints of the solutions and also the segments which meet their deadline.

The improvement method uses two mechanisms of shifting defined as follows:

1 The *Push*($S_k$, $S_i$) mechanism tries to insert the segment $S_i$ between $S_k$ and $S_{k-1}$.

Example: $C = S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8$.

If *Push*($S_3$, $S_7$) succeeds then $C = S_1, S_2, S_7, S_3, S_4, S_5, S_6, S_8$

2 The *Interchange*($S_k$, $S_i$) mechanism tries to exchange positions between $S_i$ and $S_k$.

Example: $C = S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8$.

If *Interchange*($S_3$, $S_7$) succeeds then $C = S_1, S_2, S_7, S_4, S_5, S_6, S_3, S_8$

The algorithm of the improvement method is the following:

---

**Algorithm**

**Begin**

Assume that C is the seed solution to improve.

3 **C* ← C;**

4 **SORT ALL SEGMENTS WHICH VIOLATE THEIR DEADLINE:**
ViolSet = {$S_{viol1}$, $S_{viol2}$,…,$S_{violk}$};

5 FOR **EACH SEGMENT $S_I$ OF VIOLSET** DO
Assume that $S_{precSi}$ is the last segment of C that has precedence relation with Si

$S_k \leftarrow$ Successeur($S_{precSi}$);
**While** ($S_k$! = $S_i$ and (Not Push($S_k$, $S_i$) **and** Not Interchange($S_k$, $S_i$)))
$S_k \leftarrow$ Successeur($S_k$);
**EndWhile**
**End For each**
**End**

### *4.2.2.3 Reference set update method*

RefSet is composed of two subsets, the first one, namely RefSet1 consists of $b1$ high quality solutions and the second called RefSet2 consists of $b2$ diverse solutions.

The first subset is referred to as the 'high quality' subset and the second is referred to as the 'diverse subset'. The solutions in RefSet1 are ordered according to their objective function value (optimisation criteria) and the set is updated with the goal of increasing the quality, that is to say decreasing $F(X)$ because we have defined the problem as a minimisation problem. That is, a new solution $X$ replaces a reference solution $Xb1$ if $F(X) < F(Xb1)$. The solutions in RefSet2 are ordered according to their diversity value and the update has the goal of increasing diversity. Therefore, a new solution $X$ replaces reference solution $Xb$ if $d\min(X) > d\min(Xb)$. Note that $d\min(X)$ is the distance between $X$ and RefSet1, and not the one between $X$ and RefSet2.

The distance between two solutions is the number of positions that we must change for the first solution to obtain the second one. Assume that $C_i$ and $C_j$ are two solutions and $D(C_i, C_j)$ is the distance between these solutions. Both solutions have the same segments $S_1, S_2 \ldots S_k \ldots, S_n$ but in two different orders. We note by $S_k^i$, the position of $S_k$ in $C_i$, and by $|S_k^i - S_k^j|$, the number of positions that separate the position of $S_k$ in $C_i$ from these in $C_j$ then:

$$D(C_i, C_j) = \sum_{k=1,n} \left| S_k^i - S_k^j \right|$$

and

$$D(X, RefSet) = D(X, RefSet1) = \underset{C_i \in RefSet1}{Min} D(X, C_i).$$

The reference set updating method uses a static mechanism to update RefSet so that the reference set is updated when all improved solutions of the iteration are generated. This method is simple to implement because there is no interaction between the order of subsets generation and the reference set updating.

### *4.2.2.4 Subsets generation method*

We limit our subsets generation method to yield only subsets of all pair-wise combinations of the solutions in the RefSet.

### *4.2.2.5 Combination method*

Three variants for the combination method have been developed. All variants are based on voting procedure. The proposed combination method can operate on several seed solutions but regarding to our subsets generation method (described above) it will operate really on only two seed solutions (solutions of the subsets). Therefore, the combination method produces at most one result solution which is the centre of gravity for the seed solutions. In some case, the combination method does not compute any solution because this combined solution violates synchronisation constraints of the real time application. In order to check the synchronisation constraints, we use the two lists of candidate and admissible segments defined above.

The skeleton of the algorithm for the three variants is the following:

---

**Algorithm**

**Begin**

Assume that $X_1, X_2,\ldots,X_k$ are seed solutions and $X_C$ is the combined solution which we want to generate. All solutions are composed of n segments;

***Initialisation:*** - initialise the candidate and admissible lists; Size($X_C$) = 0 because no segments are yet in $X_C$;

**While** (Size($X_C$) ≤ n and not Stop)

1 Each solution $x_i(s_1, s_2,\ldots,s_n)$ votes for its first segment not yet in $x_c$ only if this segment belongs to the admissible list of the actual position of $x_c$.

2 If there is no solution $x_i$ voted for then stop = true, the centre of gravity does not meet synchronization constraints;

**Else**

- Select the segment to be inserted in $X_C$ by applying one of the combination variants criteria;
- Put this segment at the end of $X_C$;
- Update candidate and admissible lists for the new free position of $X_C$ that we need to fill in the next step;

**End While**

**if** (not Stop) then $X_C$ is the centre of gravity else there is no combined solution for the seed solutions $X_1, X_2,\ldots,X_k$.

**End.**

---

The selection of the next segment to be inserted in $X_C$ is done according to one of the following variants:

- *Combination according to the segments positions:* The segment priority depends on the position of this segment in the reference solution which votes for it. Therefore in this variant after the voting procedure, we choose the segment with the lowest position.
- *Combination according to solutions qualities:* Each solution cooperates in the combined solution with a percentage depending on its quality as follows:

  Assume that $X_1,\ldots,X_k$ are the reference solutions which will be combined and $X_C$ is the combined solution to construct. Assume also that $X_1,\ldots,X_k$ have respectively

$f_1,\ldots,f_k$ as objective functions such that each solution $X_i$ will cooperate in $X_C$ with $\frac{v_i}{\sum_{j=1}^{k} v_j} * n$ segments where $v_i = n - f_i$.

At the beginning each solution $X_i$ has $\frac{v_i}{\sum_{j=1}^{k} v_j} * n$ as its score.

This score is decremented by one when a segment voted by $X_i$ is assigned to $X_C$.

- *Combination according to the segments deadlines:* From the several segments voted, the segments with the lowest deadline will be affected to $X_C$.

## 5 Experimental results

In order to test the performance of our approach, we have implemented the proposed algorithm. We have integrated our implementation in the Framework HeuristicLab (http://www.Heuristiclab.com) as a plug-ins. HeuristicLab is a very efficient framework for developing and testing optimisation methods, parameters and applying all these ingredients on a multitude of problems. We have used C# as programming language with the framework.net 1.1, Windows XP as the operating system on laptop machine with AMD Athlon processor 1.8 GHZ and 512 Mb of RAM.

The algorithm is tested with several instances generated for different problem sizes. These problem instances are inspired from a real instance called Mine-Pump reduced to four tasks instead of six tasks of the original system.

- *Reduced Mine-Pump and our instance:* The Mine-Pump system describes a system of pumping water in mine environment. It is composed of a set of four periodic tasks in the instance. The timing parameters of each task are shown in Table 1. Tasks decomposition and critical sections definition with precedence and exclusion constraints are explained in the Appendix. $P_i$ is changed for each test (random value) to obtain different problem sizes. The problem size is the number of segments in the system after the transformation to the periodic case of this instance.

**Table 1** Empirical instances timing parameters

| *Tasks\parameters* | $r_i$ | $C_i$ | $D_i$ | $P_i$ |
|---|---|---|---|---|
| $\tau_1$ | 0 | 10 | 20 | Random ≥ 20 |
| $\tau_2$ | 0 | 15 | 50 | Random ≥ 50 |
| $\tau_3$ | 0 | 1 | 1,000 | Random ≥ 1,000 |
| $\tau_4$ | 0 | 25 | 500 | Random ≥ 500 |

### *5.1 Diversification generators comparison*

We have tested the three alternatives proposed for the diversification generator on our four task instances. We have fixed the following parameters for the algorithm:

- stop criterion: max number of steps (iterations): 50;
  max number of evaluations: 50,000.

We have changed the population size and sizes of RefSet1 (height quality solutions) and RefSet2 (diverse solutions) as shown in Table 2 then the search time (second) and the number of generated solution are evaluated.

The conclusion is that according to these results, only the random diversification generator avoids stagnation, the two other generators found in the literature stagnate before reaching the result. This is affected by our modelling of the problem because we manipulate during the search of the feasible solution only the solutions that meet synchronisation constraints.

**Table 2** Comparison of the alternatives of the diversification generator

| *Alternative\parameters* | *Population size = 10. RefSet1 size = 3. RefSet2 size= 2.* | *Population size = 20. RefSet1 size = 3. RefSet2 size = 2.* | *Population size = 40. RefSet1 size = 5. RefSet2 size = 3.* |
|---|---|---|---|
| Random diversification generator | 17.31 s<br>3,199 | 24.50 s<br>3,771 | 01:24.29 s<br>14,218 |
| Diversification generator maximising distances | Stagnation | Stagnation | Stagnation |
| Diversification generator using mutation | Stagnation | Stagnation | Stagnation |

## *5.2 Combination methods comparison*

The same parameters fixed for diversification generators comparison are used. The results are shown in Table 3.

**Table 3** Comparison of the alternatives of the combination generator

| *Alternative\parameters* | *Population size = 10. RefSet1 size = 3. RefSet2 size= 2.* | *Population size = 20. RefSet1 size = 3. RefSet2 size = 2.* | *Population size = 40. RefSet1 size = 5. RefSet2 size = 3.* |
|---|---|---|---|
| Combination according to the segments positions | 17.31 s<br>3,199 | 24.50 s<br>3,771 | 01:24.29 s<br>14,218 |
| Combination according to the solutions qualities | 17.93 s<br>3,235 | 25.4 s<br>3,947 | 01:24.90 s<br>14,184 |
| Combination according to the segments deadlines | 17.67 s<br>3,207 | 26.34 s<br>21,770 | 45.00 s<br>31,780 |

From these tests results, we conclude that the performances of the three variants are approximately equal. We conclude also that the algorithm settings such as population size and height quality reference set size affect the performance of the algorithm.

## *5.3 Comparison between search time and average search time*

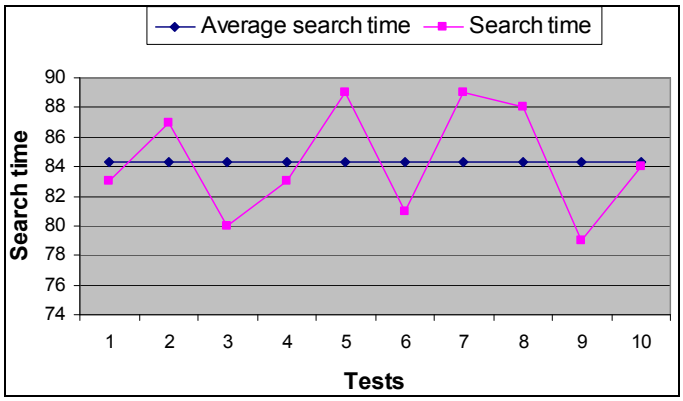
Here we repeat the same test ten times and evaluate the variation between the search time of every test and the average search time of these tests. The algorithm settings for this test are shown in Table 4.

**Table 4** Algorithm settings for tests 3 and 4

| | |
|---|---|
| Population size | 40 |
| RefSet1 size | 5 |
| RefSet2 size | 3 |
| Diversification generator | Random diversification generator |
| Combination method | Combination according to the segments positions |
| Max number of steps | 50 |
| Max number of evaluations | 50,000 |

According to the curve of Figure 1, the variation between search time and average search time is very small (about one per ten). This result is very interesting for the search algorithm because the search time depends strongly on the problem instance and algorithm settings.

**Figure 1** Comparison between search time and average search time (see online version for colours)




## *5.4 Comparison with other algorithms*

After the integration of the problem under study and the scatter search algorithm in the Heuristiclab environment, we have taken into advantage of the presence of other algorithms in this environment that can be used to solve the real time scheduling problem, without any effort of modelling or programming. The approaches of interest here are genetic algorithms and random search. In the case of the genetic algorithms, our scatter search methods will be used as operators.

In what follows, we present comparison results between our approach and these two approaches. We reintroduce the same settings for scatter search defined in the precedent test (Table 4). The settings fixed for the genetic algorithms are the following: population size = 40, mutation rate = 0.05. The replacement strategy named ‘elitism’ is proper to Heuristiclab as well as the selection operator named ‘roulette’. The crossover operator is our combination method according to the segments positions and the mutation operator is our improvement operator. For the random search, the only parameter that we need to set is the maximal rounds. It was set to 1,000.

**Table 5** Comparison with other algorithms

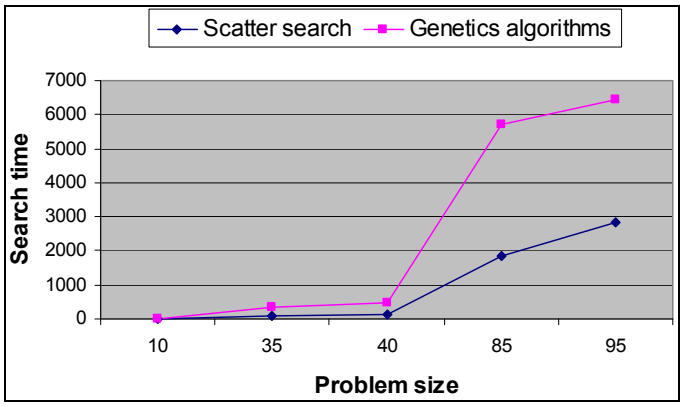
| *Problem size\ algorithm* | *Scatter search* | | *Genetic algorithms* | | *Random search* | |
|---|---|---|---|---|---|---|
| | *ST* | *SQ* | *ST* | *SQ* | *ST* | *SQ* |
| 10 | 3.55 | 0 | 15.21 | 0 | 00.42 | 0 |
| 35 | 94 | 0 | 331.95 | 0 | 07.55 | 17 |
| 40 | 149 | 0 | 451.55 | 0 | 12.38 | 20 |
| 85 | 1,851 | 0 | 5,705.20 | 0 | 94.24 | 49 |
| 95 | 2,837 | 0 | 6,442.99 | 0 | 177.42 | 56 |

Considering that the solution quality is the segments number that do not meet their deadlines, the result of search time and solution quality returned by each algorithm are shown in Table 5 (ST for search time, SQ for solution quality):

According to this set of tests, we conclude that the random search has the best search time but with the worst solution quality when the problem size increases. Therefore this algorithm, which is a local optimum search, is not important for the real time scheduling problem. According to this conclusion the random search algorithm will not be compared with genetic algorithms and scatter search.

Both genetic algorithms and scatter search retrieve the feasible solution and scatter search outperforms genetic algorithms for search time. Therefore, the difference between the search time of both methods increases when the problem size increases, this is clarified by Figure 2.

**Figure 2** Comparison between scatter search and genetic algorithms (see online version for colours)




## 6 Conclusions

We have presented in this paper a pre-run-time scheduling algorithm for real time tasks with timing, precedence and exclusion constraints. The algorithm is based on the scatter search meta-heuristic. We have implemented and tested the algorithm on different instance sizes. Using the heuristic-lab environment, a GA algorithm and a random search have been implemented. We then compared the three approaches: the scatter search, the

genetic algorithm and the random search between them. In terms of solution quality (finding feasible solution) both scatter search and genetic algorithm found a feasible solution while scatter search response time was faster.

In the future, we intend to consider several other aspects of scheduling and we plan to handle the pre-emption issue more specifically. Another perspective is to extend scatter search for real time systems on multi-processor architecture.

## Acknowledgments

The authors would like to thank anonymous reviewers for their valuable comments to improve the presentation of this work.

## References

Blazewicz, J., Glover, F. and Kasprzak, M. (2004) 'DNA sequencing – tabu and scatter search combined', *INFORMS Journal on Computing*, Vol. 16, No. 3, pp.232–240.

Cavalcante, S.V. (1997) 'A hardware-software co-design system for embedded real-time applications', PhD thesis, University of Newcastle upon Tone.

Debels, D., De Reyck, B., Leus, R. and Vanhoucke, M. (2006) 'A hybridscatter search/electromagnetism meta-heuristic for project scheduling', *European Journal of Operational Research*, Vol. 169, No. 2, pp.638–653.

Dinatale, M. and Stankovic, J.A. (1995) 'Applicability of simulated annealing methods to real-time scheduling and jitter control', *Proceeding of IEEE Real-time Systems Symposium*, Pisa, Italy, pp.190–199.

Dobrin, R. (2005) 'Combining off-line schedule and fixed priority scheduling in real-time computer systems', PhD thesis, Mälardalen University, Sweden.

Drias, H. (2001) 'Scatter search for Max-Sat problem', *Proceedings of IEEE SSST*, USA.

Drias, H. and Khabzaoui, M. (2001) 'Scatter search with random walk strategy for solving hard Max-W-Sat problems', *LNAI 2070*, pp.35–44.

Garey, M.R. and Johnson, D.S. (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, USA.

Glover, F. (1998) 'A template for scatter search and path relinking', in Hao, J-K., Lutton, E., Ronald, E., Schoenauer, M. and Snyers, D. (Eds.): *Artificial Evolution*, Lecture Notes in Computer Science 1363, pp.13–54.

Laalaoui, Y. and Drias, H. (2009) 'Learning-based approach for multiprocessor scheduling under timing constraints and N-Queens problems', *International Journal of Advanced Operations Management (IJAOM)*, Vol. 1, No. 4, pp.290–311.

Laalaoui, Y., Drias, H., Bouridah, A. and Badlishab, A. (2009) 'Ant colony system with stagnation avoidance for the scheduling of real-time tasks', *IEEE SSCI CI-Sched 2009*, Nashville, Tennessee, USA.

Liu, C.L. and Layland, J.W. (1973) 'Scheduling algorithms for multiprogramming in a hard real-time environment', *Journal of the ACM*, Vol. 20, No. 1, pp.46–61.

Mok, A.K. (1983) 'Fundamental design for the hard real-time environments', PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Boston, MA, USA.

Navet, N. and Migge, J. (2003) 'Fine tuning the scheduling of tasks through a genetic algorithm: application to Posix1003.1b compliant systems', *IEE Proc-Software*, Vol. 150, No. 1, pp.13–24.

Nossal, R. (1998) 'An evolutionary approach to multiprocessor scheduling of dependant tasks', *Future Generation Comp. Syst.*, Vol. 14, Nos. 5–6, pp.383–392.

Shepard, T. and Gagne, M. (1991) 'A pre-time scheduling algorithm for hard-real time system', *IEEE Transactions on Software Engineering*, Vol. 17, No. 7, pp.669–677.

Tindell, K., Burns, A. and Wellings, A.J. (1992) 'Allocating hard-real-time tasks: an NP-hard problem made easy', *Journal of Real-Time Systems*, Vol. 4, No. 2, pp.145–165.

Xu, J. (2003) 'On inspection and verification of software with timing requirements', *IEEE Transactions on Software Engineering*, Vol. 29, No. 8, pp.705–720.

Xu, J. and Lam, K-Y. (1998) 'Integrating runtime scheduling and pre-runtime scheduling of real-time processes,' *Proc. 23rd IFAC/IFIP Workshop Real-time Programming*.

Xu, J. and Parnas, D. (1992) 'Pre-run-time scheduling of processes with exclusion relations on nested or overlapping critical sections', *Proc. Eleventh Annual IEEE International Phoenix Conference on Computers and Communications (IPCCC-92)*, Scottsdale, Arizona, pp.774–782.

Xu, J. and Parnas, D. (1993) 'On satisfying timing constraints in hard-real-time systems', *IEEE Transactions on Software Engineering*, Vol. 19, No. 1, pp.70–84.

## Notes

1 It is known for the real time community that precedence constraints are specified between segments of tasks only when tasks have the same periods.

## Appendix

### *Test problem description*

The system is a set of four real time periodic tasks. We note that we have used this problem as a model in our tests and we have changed the period duration of each task to obtain different problem sizes. Timing, precedence and exclusion constraints are defined in the following lines.

---

Task T_1 < 0, 10, 20, 80 >

Segments:

T_11 <0, 8, 18>

T_12 <8, 2, 20>

Critical sections:

T_1_SC1(T_11, T_12)

Task T_2 < 0, 15, 50, 500 >

Segments:

T_21 < 0, 12, 47 >

T_22 < 12, 3, 50 >

Critical sections :

T_2_SC1(T_21, T_22)

Task T_3 < 0, 1, 1,000, 1,000 >

Segments:
T_31 < 0, 1, 1,000>
Critical sections:
T_3_SC1(T_31)
TaskF T_4 < 0, 25, 500, 500 >
Segments:
T_41 < 0, 25, 500 >

Critical sections:
T_4_SC1(T_41)
Constraints:
EXCLUSION:
T_1_SC1 EXCLUDE T_2_SC1
PRECEDENCE:
T_21 PRECED T_41

Precedence constraints between segments belonging to the same tasks are naturally added. The problem size is calculated after the transformation of the real time task system to the periodic case using the schedule length period.