Scatter search for real-time scheduling with timing, precedence and exclusion constraints

Article *in* International Journal of Advanced Operations Management · October 2010

DOI: 10.1109/ICMWI.2010.5648002

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

Scatter search for Real-Time Scheduling with Timing, Precedence and Exclusion Constraints

Adel Bouridah
Mila University center
Mila, Algeria
a.bouridah@centre-univ-mila.dz

Habiba Drias
Research Laboratory in Artificial Intelligence (LRIA), USTHB
Algiers, Algeria
hdrias@usthb.dz

Yacine Laalaoui
National School of computer science
Algiers, Algeria
yacine.laalaoui@gmail.com

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 previous works.**

Keywords: real-time, pre-run-time scheduling, meta-heuristic, Scatter search.

I. 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. 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.

To produce a feasible schedule, one can use blind search methods like Best-First-Search and Depth-First-Search, or Branch-Bound-First methods [9] [7] [1]. It is well known that all these methods have an exponential time complexity, because the general scheduling problem is NP-Hard [4]. Consequently, results can not be computed in a reasonable time and search space. One can also use meta-heuristic like Cooperative Ants [6]. 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.

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 Optimization Problem (COP) wherein feasible solutions are feasible schedules. Solutions are modeled 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 minimization, jitter minimization [3], combining off-line and priority based scheduling methods [2] [8].

II. REAL-TIME SCHEDULING

A. *THE TASK MODEL*

The task model we use is the same presented in [6]. A periodic task τi is characterized 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 which is called the deadline is the amount of time given to the task to complete its execution.

B. *MODELLING A REAL TIME APPLICATION*

As mentioned in [6] we consider only periodic tasks in our study for the task model. Our approach includes the same steps presented in [6]. Also, the synchronization constraints (Exclusion and precedence constraints) are the same of those presented in [6].

a) Optimization 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 Si as Lateness(Si) = d(Si)-End(Si).

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 minimize F(X) knowing that a solution is feasible when F(X) = 0.

III. THE SCHEDULING ALGORITHM

A. *OVERVIEW*

The proposed scheduling algorithm is an adaptation of Scatter Search defined in [5] to the problem of scheduling hard real-time tasks with no pre-emption for mono-processor environment. Let S1, S2, …, Sn denote the set of all segments that compose our real-time task system. Our algorithm searches the feasible solution between the couple of permutation (Si, Start execution time of Si). The start execution time of any segment is calculated automatically from the position of the segment 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(Si) = Max(EndTime(Si-1), ri) thus solutions are simplified as the permutations of segment. Any segment must appear only once in a solution.

B. SCATTER SEARCH ALGORITHM

The general framework of 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.
```

1) GENERATING THE INITIAL POPULATION

In the initial phase of the algorithm we need to construct the 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 *sbegin* that meets all exclusion and precedence constraints.
2. Apply the diversification generator on sbegin to generate all trial solutions of the initial population.

2) DESIGN OF SCATTER SEARCH COMPONENTS FOR THE SCHEDULING PROBLEM

a) 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 *Si* belonging to the Candidate List, this last segment *Si* 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 with respect to all exclusion and precedence constraints. We propose three different diversification generators:

Random diversification generator:

This generator generates randomly diverse trail solutions but these solutions must meet the synchronization constraints.

Diversification generator maximizing distances:

This generator is inspired from the one described in [5] 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 = (S1, S2, ..., Sn). Define the subsequence C(h:k), where k is a positive integer between 1 and h, to be given by C(h:k) = (Sk, Sk+h, Sk+2h, ..., Sk+rh), where r is the largest non negative integer such that k+rh   n. Then define the permutation C(h), for h   n, to be C(h) = (C(h:h), C(h:h-1), ..., C(h:1)).

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 synchronization constraints.

b) 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 synchronization 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(sk,si) mechanism tries to insert the segment si between sk and sk-1
   Example: let c=s1, s2, s3, s4, s5, s6, s7, s8, if Push(s3,s7) succeeds then
   c= s1, s2, s7, s3, s4, s5, s6, s8.
2. The interchange(sk,si) mechanism tries to exchange positions between si and sk.
   Example: let c=s1, s2, s3, s4, s5, s6, s7, s8, if Interchange(S3,S7) succeeds then
   c= s1, s2, s7, s4, s5, s6, s3,s8.

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={Sviol1,Sviol2,….,Sviolk} ;
5. for each segment Si of ViolSet do
   Assume that SprecSi is the last segment of c that has
   precedence relation with Si
      Sk← Successeur (SprecSi);
      While (Sk != Si and (Not Push(Sk,Si) and Not
            Interchange(Sk,Si)))
              Sk← Successeur (Sk) ;
      EndWhile
   End For each
End
```

c) 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 (optimization criteria) and the set is updated with the goal of increasing the quality, decreasing F(X) because we have defined the problem as a minimization 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 dmin(X)> dmin(Xb). We note that dmin(X) is the distance between X and RefSet1 and not 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 Ci et Cj are two solutions and D(Ci,Cj) is the distance between these solutions. Both solutions have the same segments S1,S2…Sk...,Sn but in two different orders. We note S_k^i the position of Sk in Ci, $\left|S_k^i - S_k^j\right|$ is the number of positions that separate the position of Sk in Ci from these in Cj then:

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

And

$$D(X, \mathrm{Re}fSet) = D(X, \mathrm{Re}fSet1) = \min_{C^i \in \mathrm{Re}fset1} D(X, C^i)$$

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 generated subsets and the updated reference set.

d) Subsets generation method

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

e) Combination method

Three variants for the combination method have been developed. All variants are based on voting procedure. The proposed combination method operates on several seed solutions but really it operates only on two seed solutions regarding to our subsets generation method. Therefore, the combination method produces at most one result solution which is the centre of gravity for the seed solutions. In some situation, the combination method does not compute any solution because this combined solution violates synchronization constraints of the real time application. In order to check the synchronization 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 x1,x2,…,xk are seed solutions and xc is the combined solution which we want to generate. All solutions are composed of n segments;
Initialisation : - initialise the candidate and admissible lists; Size(xc)=0 because no segments are yet in xc;
While ( Size(xc)≤n and not Stop )
1.  Each solution xi(s1,s2,..,sn) votes for its first segment not yet in xc only if this segment belongs to the admissible list of the actual position of xc.
2.  If there is no solution xi voted for then stop=true, the centre of gravity does not meet synchronization constraints;
    Else
    - Select the segment to be inserted in xc by applying one of the combination variants criteria;
    - Put this segment at the end of xc;
    - Update candidate and admissible lists for the new free position of xc that we need to fill in the next step;
  End While
  if (not Stop) then xc is the centre of gravity else there is no combined solution for the seed solutions x1,x2,…,xk.
End.
```

The selection of the next segment to be inserted in xc 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 x1,…,xk are the reference solutions which will be combined and xc is the combined solution to construct. Assume also that x1,…,xk have respectively f1,…,fk as objective functions such that each solution xi will cooperate in xc with $\frac{v^i}{\sum_{j=1}^{k} v^j} * n$ segments where vi=n-fi.

At the beginning each solution Xi has $\frac{f^i}{\sum_{j=1}^{k} f^j}$ as score.

This score is decremented by one when a segment voted by xi is assigned to xc.

Combination according to the segments deadlines

From the several segments voted, the segments with the lowest deadline will be affected to xc.

IV. Experimental results

In order to test the performance of our approach, we have implemented the algorithm we have developed. We have integrated our implementation in the Framework HeuristicLab (www.Heuristiclab.com) as Plug-ins. HeuristicLab is a very efficient framework for developing and testing optimization 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 1.8GHZ and 512Mb 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 4 tasks instead of 6 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 I. Pi 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 I. Empirical instances timing parameters

Tasks\parameters	ri	Ci	Di	Pi
τ1	0	10	20	Random≥20
τ2	0	15	50	Random≥50
τ3	0	1	1000	Random≥1000
τ4	0	25	500	Random≥500

A. Diversification generators comparison

We have tested the three alternatives proposed for the diversification generator on our 4 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 change the population size and the size of RefSet1 (high quality solutions) and RefSet2 (diverse solutions) as shown in Table II then the search time (second column) and the number of generated solution are evaluated.

TABLE II. 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 3199	24.50 s 3771	01:24.29 s 14218
Diversification generator maximizing distances	Stagnation	Stagnation	Stagnation
Diversification generator using mutation	Stagnation	Stagnation	Stagnation

We conclude 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 problem modeling because we manipulate during the search of the feasible solution only the solutions that meet synchronization constraints.

We have made the same tests to compare the combination methods and we have concluded that the performances of the three variants are approximately equal. We have concluded also that the algorithm settings such as population size and high quality reference set size affect the performance of the algorithm.

B. Comparison with other algorithms

After the integration of the problem under study and the scatter search algorithm in the Heuristiclab environment, we have taken advantage of the presence of other algorithms in this environment to solve the real time scheduling problem, without any effort of modeling or programming. It concerns specifically genetic algorithms and Random search.

Here we present the test of comparison between these two approaches with ours. We reintroduce the same settings for scatter search defined in the precedent test (Table III). The parameters for the Genetic algorithms are tuned as follows: 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 represents 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 III. SCATTER SEARCH SETTINGS

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

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 V (ST for search time, SQ for solution quality):

TABLE IV. 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	1851	0	5 705.20	0	94.24	49
95	2837	0	6 442.99	0	177.42	56

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.

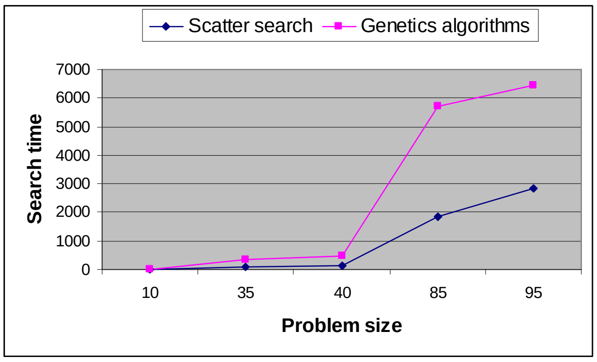

Figure 1. Comparison between scatter search and genetic algorithms

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 1.

V. CONCLUSION

We have presented in this paper a pre-run-time scheduling algorithm for real time tasks with timing, precedence and exclusion constraints. It is based on the scatter search meta-heuristic. We have implemented and tested the algorithm on different instance sizes. With the support of the Heuristic-Lab environment, a GA algorithm and a random search have been developed. Then we have 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 retrieve the feasible solution while scatter search response time is faster.

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

REFERENCES

[1] Cavalcante, S.V, *"A Hardware-Software Co- Design System for Embedded Real-Time Applications,"* PhD Thesis University of Newcastle upon Tone, 1997.

[2] Dobrin, R, "*Combining Off-line Schedule and Fixed Priority Scheduling in Real-Time Computer Systems,*" PhD Thesis Mälardalen University, Sweden, 2005.

[3] Dinatale, M., Stankovic, J.A, "Applicability of simulated annealing methods to real-time scheduling and jitter control," *Proceeding of IEEE Real-Time Systems Symposium*, Pisa, Italy, pp. 190-199, 1995.

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

[5] Glover, F., "A Template for Scatter Search and Path Relinking," *In Artificial Evolution, Lecture Notes in Computer Science* 1363, J.-K. Hao, E. Lutton, E. Ronald , M. Schoenauer and D. Snyers (Eds.), Springer-Verlag, pp. 13-54, 1998.

[6] Laalaoui, Y., Drias, H., Bouridah, A., Badlishab, A., "Ant Colony System with Stagnation Avoidance for the Scheduling of Real-time Tasks," *IEEE SSCI CI-Sched 2009*, Nashville, Tennessee, USA, 2009.

[7] Shepard, T., Gagne, M., "A pre-time schudling algorithme for hardreal time system," *IEEE Transactions on Software Engineering*, Vol.17, No.7, pp. 669-677, 1991.

[8] Xu, J., Lam, K.-Y., "Integrating RunTime Scheduling and Preruntime Scheduling of Real-Time Processes," *Proc 23rd IFAC/ IFIP Workshop Real-Time Programming*, 1998.

[9] Xu, J., Parnas, D., "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, 1992.