A Binary Integer Linear Programming-Based Approach for Solving the Allocation Problem in Multiprocessor Partitioned Scheduling

L. Puente-Maury*, P. Mejía-Alvarez*, L. E. Leyva-del-Foyo†
*Department of Computer Science, CINVESTAV-IPN, Mexico D.F., Mexico
Email: lpuente@computacion.cs.cinvestav.mx; pmalvarez@cs.cinvestav.mx
†Department of Information Technologies, UAM-Cuajimalpa, Mexico D.F., Mexico
Email: lleyva@cua.uam.mx

***Abstract*—Scheduling is a main issue of real-time systems because it involves meeting the deadlines. In this paper, we address the problem of scheduling a set of periodic tasks on m processors under EDF (Earliest Deadline First) using a partitioned scheme. The allocation problem is transformed into a binary integer linear program. Then, it is solved by applying Geoffrion's version of Balas' additive method, optimized for the real-time scheduling problem. In order to assess the feasibility of the approach for a small size practical problem, some experimental results are shown.**

***Keywords*-Balas' additive algorithm, binary integer programming, multiprocessor real-time scheduling, partitioned scheme**

I. Introduction

Real-time scheduling consists of planning the order in which the system's tasks execute, so that their deadlines are met. For uniprocessor systems, the scheduling problem has been widely studied and effective algorithms have been found that take into account many aspects that arise in real-time systems [1].

Nowadays, the use of one processor is not sufficient because of the increasing demands in computing power of modern applications. The commercial availability of multiple cores, their low cost, and the fact that these kind of systems give support to some important issues of real-time such as fault-tolerance, have made the real-time community focus on the study of multiprocessor scheduling in the last decades [2].

The allocation problem, which consists of assigning tasks to processors, has been proved to be NP-hard [3]. This problem has been widely studied in the literature and extended to include processors with different speeds (e.g. heterogeneous) and several software architecture models (e.g. precedence relations). However, the main direction of research has focused on finding out approximated solutions in polynomial time. To our best knowledge, there is no real-time work focusing on applying exact optimization techniques to obtain the best real-time scheduling in multiprocessor systems for the task model that is used in this work.

We address the problem of scheduling a set of n preemptive and periodic tasks with hard deadlines on m homogeneous processors using a partitioned scheme. In this scheme, each task is assigned to a processor and all instances of the task must execute on the same processor. Then, on each processor the assigned tasks are scheduled by EDF. We assume that the timing constraints of the tasks in the system (worst-case execution time, period, etc.) are known *a priori*.

In this context, this work does not pretend to offer the most efficient algorithm for obtaining the optimal solution for the allocation problem in a real-time system, but to investigate how feasible it would be to find that optimal solution. This work uses the same approach as that of the research done in 2008 by Baruah and Bini [6]. Their work models the allocation problem as an integer linear programming (ILP) problem with binary variables. However, they do not apply any exact method to provide a solution.

The contributions of our work are the following:

- An optimal solution of the allocation problem under our task model is provided. Our aim is to find the best possible allocation by applying an implicit enumeration technique, using a well-known algorithm from the operations research area.
- An optimization of the original algorithm is presented, taking into consideration the specificities of the real-time scheduling problem.
- Some experimental results are presented to evaluate the feasibility of applying this method in problems of the presented size.

The paper is organized as follows: section II gives a brief overview of how the allocation problem has been previously studied. In section III, the task model is defined. Later, in section IV, the allocation problem is modeled as a binary linear integer programming problem and our solution strategy is described. Also, the arguments for the improvements to the original algorithm are stated. Section V shows some experimental results that analyze the acceptance ratio and the execution time of the algorithm, to evaluate its feasibility. Finally, section VI gives the conclusions and some future work to be done, so as to improve the performance results.

II. Previous work

The allocation problem has been faced using heuristics or some form of enumerative method. Neural networks, genetic algorithms (GA's) and bin-packing are some of the heuristic approaches. Nevertheless, they all have some disadvantages. For example, for NP-hard problems, GA's do not offer better performance than an algorithm specially designed for a particular problem [4]. For this kind of problems, GA's focus in obtaining approximate solutions in polynomial time. Although many bin-packing heuristics have been proved to have a good performance and GA's have been used in the context of real-time scheduling (see for example [5]), these approaches do not work for our purposes. We intend to obtain an optimal solution of a scheduling problem with reasonably low execution time.

Baruah and Bini modeled the allocation problem with EDF scheduling on each processor as an integer linear problem with binary variables [6]. The exact problem is represented with pseudo-polinomially many restrictions, and then an approximated problem is stated with polinomially many restrictions.

Recently, an off-line non-heuristic approach proposed the construction of lookup tables to partition a set of tasks [7]. By using the table, task partitioning is achieved in polynomial time, but the construction of the table involves choosing certain values of utilizations depending on a fixed value of ϵ and exhaustively trying all possible combinations of distinct utilization values until the capacities of the processors are reached. Having a specific task set, it is enough to look up on the table to obtain the solution. In general, this approach yields high computation time. The smaller the value of ϵ, the greater the accuracy and the computation that must be executed previously.

III. Task Model

In this paper, we are considering the partitioned scheme to schedule a set $T = \{\tau_1, \ldots, \tau_n\}$ of implicit-deadline, periodic tasks without any precedence constraints, executing on m identical processors. Each task τ_i is characterized by its period T_i, its worst-case execution time C_i and its utilization $u_i = \frac{C_i}{T_i}$. After the tasks have been assigned to the processors, they will be scheduled under EDF on each processor. A necessary and sufficient condition for the tasks assigned to each processor to be schedulable by EDF is that the total sum of their utilizations is not greater than the capacity of the processor, which is assumed to be one.

IV. Allocation problem and binary integer linear programs (BILP's)

Here, the optimization problem which models the allocation problem for the task model given above will be described. It is assumed that it is impossible to partition a job, and that there will not be any job-level migration, i.e., all instances of one task must execute on a single processor. Let us denote by:

$$X_{ij} = \begin{cases} 1 \text{ if task } i \text{ is assigned to processor } j, \\ 0 \text{ otherwise.} \end{cases} \tag{4.1}$$

Then, the allocation problem can be stated as the following linear integer problem with $m \times n$ binary variables and $m+n$ constraints:

$$\max \sum_{i=1}^{n} u_i \Big(\sum_{j=1}^{m} X_{ij} \Big) \tag{4.2}$$

$$\text{s.t.} \sum_{i=1}^{n} u_i X_{ij} \leq 1, \quad j = 1, \ldots, m, \tag{4.2a}$$

$$\sum_{j=1}^{m} X_{ij} \leq 1, \quad i = 1, \ldots, n, \tag{4.2b}$$

$$X_{ij} \in \{0, 1\}, \quad i = 1, \ldots, n, \; j = 1, \ldots, m. \tag{4.2c}$$

The objective function is to maximize the processors utilization. The first m constraints correspond to the fact that the utilizations of the tasks allocated to each processor must sum up to 1 (the processor's capacity), at most, i.e., the EDF exact schedulability test. The other n constraints correspond to the fact that each task must be assigned to one processor or not be assigned at all.

A. Solving binary integer linear problems

This BILP problem models the allocation problem exposed in section II. The latter is a multiple knapsack problem where the processors are the bins and the tasks are the objects to be assigned to the bins [13]. As an NP-hard problem, up to now no polynomial time algorithm exists for computing its optimal solution [13].

Integer linear programming problems can be solved by applying any linear programming technique like Gomory's cutting planes, Land and Doig's branch and bound method, or some hybrids of these two, by introducing additional binary restrictions on the variables [8][9][10]. But since these methods were developed to solve ILP's, they do not take any advantage on the special characteristics of BILP's. Some methods have been proposed to solve these binary problems in a more efficient way.

One of them is Balas' method, where all 2^n possible solutions are explicitely or implicitely enumerated [11]. The strategy that makes this method efficient is that only some solutions are selected for explicit enumeration. The branch and bound approach of this method successively assigns values of zeros or ones to some of the variables, so that after treating some of the 2^n combinations, it finds an optimal solution or assures that there is no optimal solution at all.

Based on Balas' work, Arthur Geoffrion gave a reformulated version of the additive algorithm in 1967 [12]. The spatial complexity (storage) used by the original method is

reduced by this version [14]. The aim here is to apply this approach to get the best solution. There exist results showing that it is possible to reach a good tradeoff between time responses and performance of the algorithm [12].

Some of the advantages of this method are:

- Addition is the only arithmetic operation required, thus eliminating roundoff problems. Since no linear systems of equations need to be solved, it is unnecessary to use matrix multiplication, which costs enough.
- The algorithm gives the optimal solution, if one exists.
- If the calculations need to be stopped before termination, usually a feasible solution is in store, even if it is not the optimal.
- At any moment, it is possible to know how many solutions have been implicitly enumerated, thus estimating how much time is needed for termination.
- It is also possible to apply this method to non-linear objective functions.

B. Balas' additive algorithm

Fig. 1 shows a simplified version of Balas' algorithm given in [12] for solving a BILP of the standard form:

$$\min f(X) = C^T X \tag{4.3}$$

$$\text{s.t. } B + AX \geq 0, \tag{4.3a}$$

where X is a n-dimensional vector (X is restricted to be integer), B is a m-dimensional vector, A is a $m \times n$ matrix, $F(X)$ is the objective function, and C is the non-negative n-dimensional vector of cost coefficients.

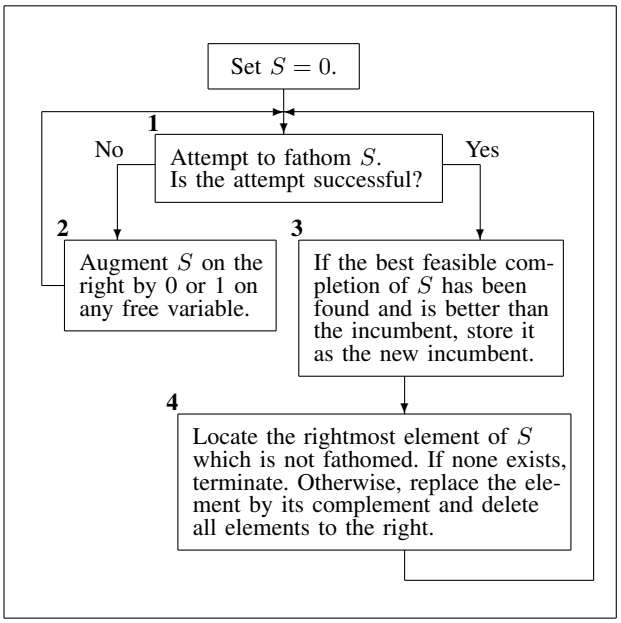

Figure 1. Basic version of Balas' algorithm [12].

Here, some concepts are needed [12]. A **partial solution** S is defined as a set of values assigned to some of the variables of the problem. The variables without value on S are called **free variables**. The **completions** of a partial solution S are all the combinations of the values that could be assigned to the free variables plus the values of the variables in S.

If all possible 2^n combinations of possible solutions to a BILP with n variables are represented as a tree, a partial solution is any node, and the completions are the leafs (see Fig. 2). A partial solution S can be **fathomed** if either:

- It is possible to find the best feasible completion of S.
- There is no feasible completion of S, better than the stored solution.

Geoffrion's version of Balas' algorithm tries to fathom a partial solution on each iteration. Each time a feasible completion is found, it is compared against the incumbent, which is the best feasible solution found up to that moment. If S is fathomed but there is no better solution than the incumbent, then the algorithm goes up until it reaches the level above the variable from which the fathoming was done and takes the other branch, which corresponds to the complement of the value of that variable (see Fig. 2). If S cannot be fathomed, the algorithm goes down one level on the tree (assigns a value to a free variable) and tries to fathom the new partial solution.

To reduce the BILP problem stated on the previous section to the standard form which is needed to apply this

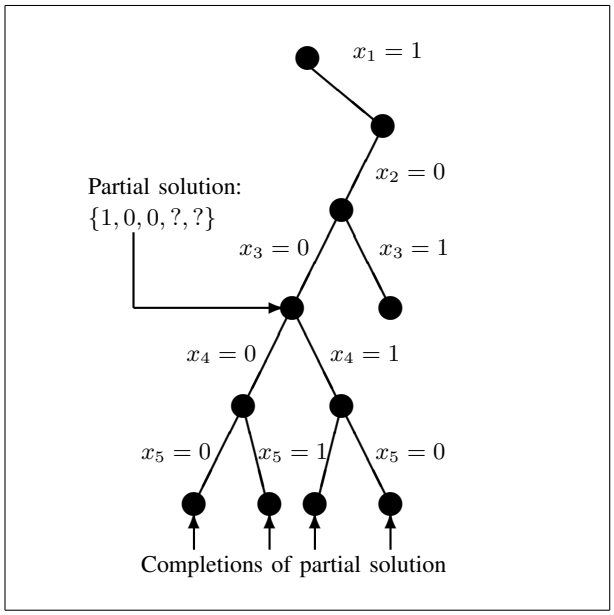

Figure 2. Partial solutions and completions on a Branch & Bound algorithm with 5 variables.

method, it is necessary to change variables:

$$Y_{ij} = 1 - X_{ij},\ \ i = 1, \ldots, n,\ j = 1, \ldots, m. \tag{4.4}$$

This change of variables leads to the problem in the standard form:

$$\min \sum_{j} \sum_{i} (u_i Y_{ij} - u_i) \equiv \min \sum_{i} \sum_{j} u_i Y_{ij} \tag{4.5}$$

$$s.t. (1 - m) + \sum_{j=1}^{m} Y_{ij} \geq 0, \quad i = 1, \ldots, n, \tag{4.5a}$$

$$(1 - \sum_{i=1}^{n} u_i) + \sum_{i=1}^{n} u_i Y_{ij} \geq 0, \quad j = 1, \ldots, m. \tag{4.5b}$$

So that $(1) \times (n \times m)$ vector B^T is:

$$(\underbrace{1 - m, \ldots, 1 - m}_{n \text{ times}}, \underbrace{1 - \sum_{1}^{n} u_i, \ldots, 1 - \sum_{1}^{n} u_i}_{m \text{ times}}). \tag{4.6}$$

And $(n + m) \times (n \times m)$ matrix A has the form:

$$\begin{pmatrix} 1 & 1 & \ldots & 1 & 0 & \ldots & 0 & 0 & \ldots & 0 \\ 0 & 0 & \ldots & 0 & 1 & \ldots & 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 0 & 0 & \ldots & 0 & 0 & \ldots & 0 \\ 0 & 0 & \ldots & 0 & 0 & \ldots & 1 & 1 & \ldots & 1 \\ u_1 & 0 & \ldots & 0 & u_2 & \ldots & u_n & 0 & \ldots & 0 \\ 0 & u_1 & \ldots & 0 & 0 & \ldots & 0 & u_n & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & u_1 & 0 & \ldots & 0 & 0 & \ldots & u_n \end{pmatrix}. \tag{4.7}$$

The algorithm has been modified with the introduction of three optimizations based on the characteristics of the scheduling problem we are dealing with. These are: (1) starting from a known trivial solution; (2) cuttting-off iterations when a scheduling is achieved; (3) reducing number of variables by filtering the task set.

1) Starting from a known trivial solution. Before entering the while loop of Fig. 1, we start with a non-zero solution. Intuitively, the tasks are ordered by decreasing order of their utilizations, and then the first m tasks are assigned to the m processors. In practice, it has been proved that bin-packing heuristics work better on a set of tasks which are ordered by decreasing utilization [15]. So, S is such that the first task with the highest utilization is assigned to the first processor; the second, to the second processor and so on. Our experimental results showed that in general, the best solution found was a completion of this initial solution. This implies a reduction on the number of iterations needed to find the best solution.

2) Cutting-off iterations when a scheduling is achieved. In Step 3 (Fig. 1), the algorithm evaluates all feasible solutions searching a lower bound of the objective function. However, due to the nature of the allocation problem, once all tasks have been allocated, it is not necessary to continue the search for a better solution. Hence, in Step 3, if all tasks have been allocated, the algorithm terminates.

3) Reducing number of variables by filtering the task set. It was noticed that most of the times when there is a task that cannot be allocated at all in any of the processors; the algorithm does a big amount of iterations trying to allocate that task. Based on this, the third improvement is to make a filter of the generated set and take out the tasks that are impossible to allocate due to their big utilizations. For instance, if there are 4 processors and 10 tasks, 5 of which have utilizations greater than 0.5, it will be impossible to allocate one of these tasks. In this example, the problem size would be 40 variables ($40 = 4 \times 10$), which gives a search space of 2^{40}. If a task is eliminated, the problem size is reduced to 36 ($36 = 4 \times 9$), which leads to a search space of 2^{36}. This step has to be done before all steps of Fig. 1 and it could lead to an exponential reduction on the search space.

V. Experimental Results

It is not pretended to contrast the performance of the algorithm proposed here with some other ones. To the best of our knowledge, there are no experimental results regarding the actual execution time performance of exact multiprocessor real-time scheduling algorithms. The purpose of this section is to provide experimental evidences for the suitability of using exact multiprocessor scheduling algorithms (like the one that is proposed) for some practical real-time multiprocessor systems.

Although a comparison of algorithms is beyond this work, it is worth noting that, regarding the evaluation of scheduling algorithms, in operations research area there exist classical benchmarks for basic scheduling problems (see for example Taillard's benchmarks [17]). However, these benchmarks are not suitable to the purposes of this work, due to the differences between the scheduling problem in operations research and the scheduling problem in real-time systems. For example, in standard operations research benchmarks there are no due times nor release times for tasks and usually the objective is to minimize the makespan (the total execution time). In real-time scheduling problem the makespan is not a concern, but the due times of each task. There are no such standard benchmarks for real-time scheduling. Synthetic task sets with suitable statistical characteristics are used instead.

The algorithm proposed in the last section has been tested on task sets randomly generated in order to evaluate its performance. We considered a system with 4 processors and several samples of task sets were generated. The periods of the tasks are uniformly distributed on the interval $[100, 500]$. The sum of their utilizations will be denoted by U_S. For each value of $U_S \in \{2, 2.2, 2.4, \ldots, 3.8, 4\}$, 500 different task sets were tested. This was done to test the performance of the algorithm when the capacity of the system is filled up to 50% ($U_S = 2$), 55%, and so on until it reaches its maximum capacity ($U_S = 4$).

Three experiments were conducted for three different kinds of systems: the first is composed of "*light and heavy*" tasks, the second considers only "*light*" tasks and the third considers only "*heavy*" tasks. The utilizations of the tasks are uniformly distributed on the intervals [0.1;0.7], [0.1;0.4] and [0.4;0.7], for the three systems, respectively.

For a given set of task sets SS, we define the *acceptance ratio* ρ of a multiprocessor scheduling algorithm P as:

$$\rho_{SS} = \frac{\text{number of task sets for which } P \text{ accepted all the tasks}}{\text{number of task sets in } SS} * 100 \qquad (5.1)$$

All experiments were conducted on an Intel Core 2 Duo CPU, 2.93 GHz, 3.0 GB RAM and a Windows 7 32-bits Operating System. The application was written in C and it was compiled with GNU compiler.

Table 1 shows the time in seconds and the number of iterations for all the experiments. As expected, the sample sets composed of "*light*" tasks have the largest average execution time. The reason for this is that there are more variables and the size of the problem is bigger. Also, the problem size is never reduced by the filter due to the size of the tasks.

Table I

TIME (IN SECONDS) AND NUMBER OF ITERATIONS.

	1st Experiment		2nd Experiment		3rd Experiment	
U_s	Time	No. iter.	Time	No. iter.	Time	No. iter.
2	$2 \cdot 10^{-5}$	6	10^{-4}	15	$3 \cdot 10^{-6}$	1.2
2.2	$3 \cdot 10^{-5}$	7	$2 \cdot 10^{-4}$	17	$4 \cdot 10^{-6}$	2
2.4	$4 \cdot 10^{-5}$	9	$2 \cdot 10^{-4}$	19	$7 \cdot 10^{-6}$	3
2.6	$6 \cdot 10^{-5}$	10	$4 \cdot 10^{-3}$	368	$2 \cdot 10^{-5}$	4.3
2.8	$8 \cdot 10^{-5}$	11.5	0.06	6581	$3 \cdot 10^{-5}$	5
3	$5 \cdot 10^{-4}$	69	0.3	$3 \cdot 10^{4}$	$4 \cdot 10^{-5}$	6
3.2	$3 \cdot 10^{-3}$	461	2.8	$2 \cdot 10^{5}$	10^{-3}	215
3.4	0.013	1660	9.8	$8 \cdot 10^{5}$	0.0025	466
3.6	0.038	5010	47.3	$3 \cdot 10^{6}$	0.03	$5 \cdot 10^{3}$
3.8	0.16	$2 \cdot 10^{4}$	208.2	10^{7}	0.09	10^{4}
4	33.7	$3 \cdot 10^{6}$	2250	$8 \cdot 10^{7}$	0.33	$5 \cdot 10^{4}$

Figures 3, 4 and 5 below show the acceptance ratio for the three experiments, respectively.

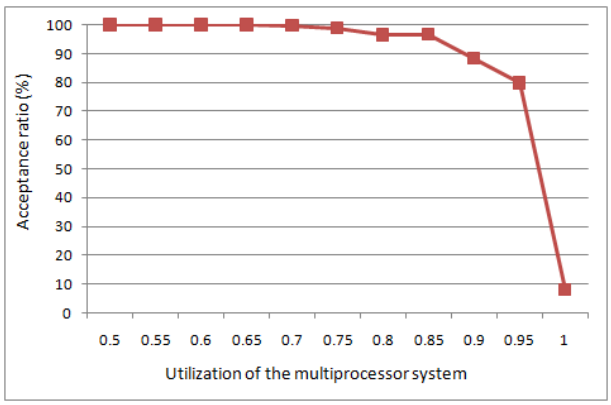

Figure 3. Acceptance ratio as a function of U_s for a heterogeneous system.

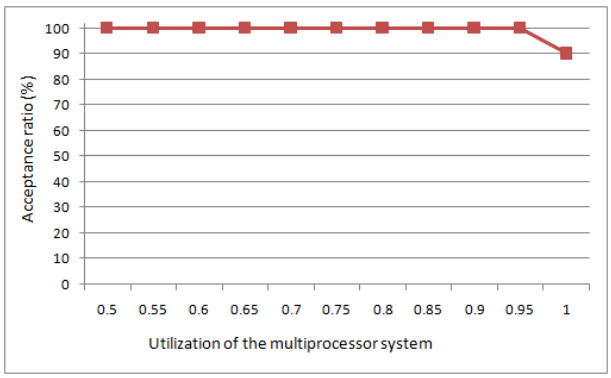

Figure 4. Acceptance ratio as a function of U_s for a "*light*" tasks system.

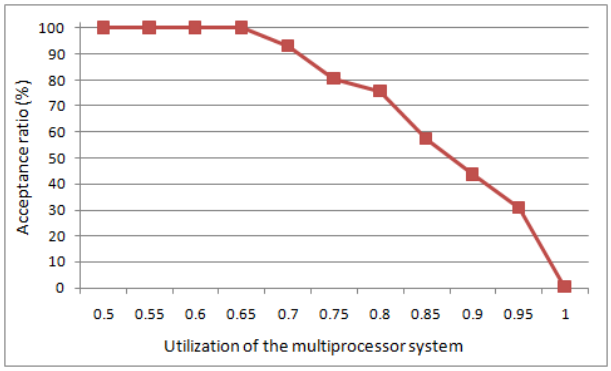

Figure 5. Acceptance ratio as a function of U_s for a "*heavy*" tasks system.

VI. CONCLUSIONS AND FUTURE WORK

By applying Geoffrion's simplified version of Balas' additive algorithm, a better performance is obtained with respect to heuristic algorithms, since this method always gives the optimal scheduling. The only disadvantage of this approach is that it could take a lot of time before termination.

We do not pretend to offer the best algorithm to find out the optimal solution to the allocation problem. To this end, Geoffrion's version of Balas' algorithm with the proposed

optimizations works for our purposes. The experimental results show the feasibility of finding the exact solution for the allocation problem on task sets of the presented size.

The experimental results presented here for 4 processors are valuable. We are dealing with an NP-hard problem, so no efficient algorithms for solving it are known. For common real-time and embedded system applications, 4 processors is a practical limit (due to typical cost, energy and space restrictions in the hardware platform of that kind of systems). These experimental results provide evidence to support the statement that, for some hard real-time applications with processor settings of typical size (up to 4 processors), an exact offline algorithm (as the one that is proposed) could be used instead of an efficient approximation algorithm.

Our future work is based on the improvement proposed by Geoffrion in [16]. In this paper, the introduction of "surrogate constraints" is proposed. These constraints encapsulate all restrictions of the original BILP and cause the algorithm to find a better solution taking into consideration all restrictions at the same time and not just one by one, as Geoffrion's version of Balas' algorithm does. On multidimensional knapsack problems, the use of surrogate constraints on every iteration of the algorithm of Fig.1 leads to solution times that seem to increase as the third power of the number of variables [16].

Also, another optimization can be achieved by changing the data structure to take profit from the sparse matrix which is characteristic of this specific problem.

REFERENCES

[1] J. A. Stankovic, M. Spuri, K. Ramamritham, G. C. Butazzo, *"Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms"*. Boston: Kluwer Academic Publishers, 1998.

[2] R. Davis, A. Burns, "A Survey of Hard Real-Time Scheduling Algorithms and Schedulability Analysis Techniques for Multiprocessor Systems", accepted for publication in *ACM Computing Surveys*, 2011.

[3] J. Y. Leung, J. Whitehead, "On the complexity of fixed-priority scheduling of periodic real-time tasks", in *Performance Evaluation (Netherlands)*, Vol. 2, No. 4, pp.237-250, December 1982.

[4] P. Oliveto, J. He, X. Yao, "Time Complexity of Evolutionary Algorithms for Combinatorial Optimization: A Decade of Results", in *International Journal of Automation and Computing*, Vol. 3, No. 4, pp. 281-293, 2007.

[5] K. Dahal, A. Hossain, B. Varghese, A. Abraham, F. Xhafa, A. Daradoumis, "Scheduling in Multiprocessor System Using Genetic Algorithms", in *Computer Information Systems and Industrial Management Applications*, June 2008.

[6] S. Baruah, E. Bini, "Partitioned scheduling of sporadic task systems: an ILP based approach", in *Proceedings of the International Conference on Design and Architectures for Signal and Image Processing (DASIP 2008)*, Brussels, Belgium. November 2008.

[7] B. Chattopadhay, S. Baruah, "A lookup-table driven approach to partitioned scheduling, in *Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS)*, Chicago, Illinois. April 2011.

[8] R. E. Gomory, "Outline of an algorithm for integer solutions to linear programs", in *Bulletin of the American Mathematical Society*, Vol. 64, No. 5, pp. 275-278, 1958.

[9] A. H. Land, A. G. Doig, "An automatic method of solving discrete programming problems", in *Econometrica*, Vol. 28, No. 3, pp. 497-520, July 1960.

[10] S. S. Rao, "Integer Programming" in *Engineering Optimization: Theory and Practice*, 3^{rd} ed. New York: Wiley, 1996.

[11] E. Balas, "An Additive Algorithm for Solving Linear Programs with Zero-One Variables", in *Operations Research*, Vol. 13, No. 3, pp. 517-549, August 1965.

[12] A. Geoffrion, "Integer Programming by Implicit Enumeration and Balas' Method", in *SIAM Review*, Vol. 9, No. 2 pp. 178-190, April 1967

[13] H. Kellerer, U. Pferschy, D. Pisinger, *Knapsack Problems*. Berlin: Springer, 2004.

[14] C. Petersen, "Computational Experience with Variants of the Balas Algorithm Applied to the Selection of R&D Projects", in *Management Science*, Vol. 13, No. 9, Series A, Sciences, pp. 736-750, May 1967.

[15] O. U. Pereira-Zapata, P. Mejia-Alvarez, "EDF and RM Multiprocessor Scheduling Algorithms: Survey and Performance Evaluation" [Online]. Available: http://delta.cs.cinvestav.mx/~pmejia/multitechreport.pdf, last accessed August 2011.

[16] A. Geoffrion, "An Improved Implicit Enumeration Approach for Integer Programming", in *Operations Research*, Vol. 17, No. 3, pp. 437-454, May-June 1969.

[17] E. Taillard, "Benchmarks for basic scheduling problems", in *European Journal of Operational Research*, Vol. 64, No. 2, pp. 278-285, January 1993.