Iterative approaches for a dynamic memory allocation problem in embedded systems

María Soto, André Rossi, Marc Sevaux*

Université de Bretagne-Sud
Lab-STICC, CNRS, Centre de recherche
BP 92116, F-56321 Lorient, France

Abstract

Memory allocation has a significant impact on energy consumption in embedded systems. In this paper, we are interested in dynamic memory allocation for embedded systems with a special emphasis on time performance. We propose two mid-term iterative approaches which are compared with existing long-term and short-term approaches, and with an ILP formulation as well. These approaches rely on solving a static version of the allocation problem and they take advantage of previous works for addressing the static problem. A statistic analysis is carried out for showing that the mid-term approach is the best one in terms of solution quality.

Keywords: Memory allocation, Embedded systems, Metaheuristics, VNS.

1. Introduction

Technology offers more and more features allowing embedded systems (such as smart phones) to surf the Web or to process HD pictures. As a consequence, the design of embedded systems becomes more and more complex. There exist some CAD tools such as Gaut [6] to generate the architecture of a circuit from its specifications. However, the designs produced by a CAD software are generally not energy aware, which is of course a major drawback. An interesting work on buffer minimization for designing embedded system has been conducted in [14], where the objective is minimizing the

*Corresponding author
Email address: marc.sevaux@univ-ubs.fr (María Soto, André Rossi, Marc Sevaux)

total surface of the buffers used for communicating data between two tasks of the application.

Designers want to minimize power consumption [1], and to some extent, electronics practitioners consider that minimizing power consumption is equivalent to minimizing the running time of the application to be executed by the embedded system [4]. Moreover, the power consumption of a given application can be estimated using an empirical model as in [9], and parallelization of data access is viewed as the main action point for minimizing execution time, and consequently power consumption.

Memory allocation and data allocation are among the main challenges in the design of embedded systems. This paper is focused on memory allocation in embedded systems because this point has a significant impact on power consumption as shown by Wuytack *et al.* in [25]. One has to know that the process of memory allocation is done off-line, during the design of the embedded system. Once set, the final embedded system is implemented in a specific architecture and will not be modified on-line. Thus, it is possible to spend some time to optimize memory allocation without consequences on the running time of the final application.

In 2012, Google has suggested a machine reassignment problem as the topic for the ROADeF/EURO Challenge [16]. This problem is a general memory allocation problem and may be considered close to the problem addressed in our study. The machine reassignment problem consists in assigning each process to exactly one machine considering the resource capacity constraints and hard constraints that make some allocations impossible. The objective is to improve the usage of a set of machines by minimizing the total cost, which is the sum of a load cost, a balance cost and several move costs. Thus, the set of machines with a limited resources capacity can be seem as the set of memory banks, and processes are the data structures to be allocated. The conflicts between the processes of a services are represented by conflicts between data structures. However, there are three major differences between the problem addressed in this paper and the machine reassignment problem proposed by ROADEF/EURO in 2012. The first one is that in the machine reassignment problem, the processes of a service have to be executed on distinct machines whereas no such limitations exist in our problem. The second one is the cost computation, especially the balance cost, that is very specific to machine reassignment problem, and that differs significantly from the cost computation for the problem addressed in this paper. Finally, the more significant difference is that the problem addressed in this paper

is dynamic. As the application requirements vary over time, a solution is a series of allocations, where each allocation is computed by taking present and future requirements into account.

A relevant problem of data allocation is addressed in [2, 12], it consists in finding the optimal allocation of relations to multidisk database such that the expected query cost is minimized. Each query involves two relations and has a probability of occurrence. The register allocation problem [10] is close to the simple version of the memory allocation problem tackled in [21]. This problem consists in assigning variables in a computer program to hardware registers in a processor. The objective is to minimize the number of used registers considering that some variables cannot be assigned to the same register. In these problems, the relations and variables can be modeled as the data structures. Disks and registers can be represented as the memory banks. However, the main differences between these problems and our problem are the external memory available and the objective functions. Also, these problems relate to static memory allocation, whereas this paper addresses the dynamic memory allocation.

This work is an extension of the general work conducted by María Soto during her PhD thesis [17]. Related problems are covered in [19, 20, 21, 23].

We have addressed various simpler versions of the memory allocation problem: in Soto *et al.* [22], we have proposed a mixed integer linear programming formulation for the general memory allocation problem and a variable neighborhood search (VNS) algorithm [13]. We have also studied an even more simplified version of this problem in Soto *et al.* [21].

We have dealt with the dynamic memory allocation problem in Soto *et al.* [23] for which an integer linear programming formulation and two iterative approaches have been devised to address this dynamic problem. In this paper, we propose a new iterative approach which combines these two iterative approaches. Similar studies with iterative approaches but in a completely different domain have also been conducted on another problem in [24].

This paper is organized as follows. Section 2 provides a short introduction to dynamic memory allocation problem. Section 3 briefly recalls the two previous iterative approaches and introduces the new method. The proposed approach is compared to previous works in Section 4, and computational results are also discussed. Finally, Section 5 presents conclusions and future work on this problem.

2. Dynamic memory allocation problem

This section presents the dynamic memory allocation problem. Readers interested in more details about this problem, about the ILP formulation and about the $\mathcal{NP}$-hard status of this problem are referred to Soto *et al.* [23]. The considered memory architecture is similar to the one of a TI C6201 device [9]. It is composed of m memory banks whose capacity is c_j kilo Bytes (kB) for all $j \in \{1, \ldots, m\}$ and an external memory denoted by $m+1$, which does not have a practical capacity limit. The processor needs q milliseconds for accessing data structures located in a memory bank, and it spends an access time which is p times longer when data structures are in the external memory.

Time is split into T time intervals whose durations may be different. The application to be implemented is assumed to be given as C source code, whose n data structures (*i.e.* variables, arrays, structures) have to be loaded in memory banks or external memory. The size of data structure s_i for $i \in \{1, \ldots, n\}$ is expressed in kB. During each time interval I_t, the application requires accessing a given subset A_t of its data structures. Thus $e_{i,t}$ denotes the number of times that data structure $s_i \in A_t$ is accessed during time interval I_t, for all $t \in \{1, \ldots, T\}$. The processor can access all its memory bank simultaneously, which allows for parallel data loading. Thus, two data structures can be loaded in parallel provided that they are allocated to two different memory banks. Two data structures are said to be *conflicting* whenever they are required at the same time by the processor. Each conflict has a cost, which is equal to the number of times the conflict appears in the current time interval. This cost might be non-integer if the application source code has been analyzed by a statistic-based code-profiling software [8, 11]. Thus, the number of conflicts in I_t is denoted by o_t, and $d_{k,t}$ is the cost of conflict $(k,t) = (k_1, k_2)$ during time interval I_t for all k in $\{1, \ldots, o_t\}$, k_1 and k_2 in A_t, and t in $\{1, \ldots, T\}$.

A conflict between two data structures is said to be *closed* if both data structures are allocated to two different memory banks. In any other case, the conflict is said to be *open* and its cost has to be paid.

Initially, all data structures are in the external memory and memory banks are empty. The time required for moving a data structure from the external memory to a memory bank (and vice-versa) is v ms/kB. The time required for moving a data structure from a memory bank to another is ℓ ms/kB. We suppose that $v \geq \ell$ and $v < p$ because a DMA (Direct Memory

Access) controller is part of the memory architecture, which allows for a direct access to data structures.

The cost of an allocation is expressed in milliseconds and is the sum of two terms. The first term is the access cost for executing operations in the application which also includes the cost of conflicts cost of all the data structures that are in a memory bank, plus the access cost of all the data structures allocated to the external memory minus the closed conflict cost. The second term is the cost of changing the allocation of structures from a time interval to the next one.

The dynamic memory allocation problem consists in allocating a memory bank or the external memory to any data structure of the application for each time interval, so as to minimize the time spent accessing and moving data structures while satisfying the memory bank capacity.

3. Iterative Approaches

In Soto *et al.* [23], we have introduced two heuristic iterative approaches which iteratively build a solution. They rely on addressing static memory allocation problems. The first approach called *Long-term* addresses the *MemExplorer* problem (ME) [22] which consists in finding a memory allocation for data structures such that the time spent accessing these data is minimized, for a given number of capacitated memory banks and an external memory. The second approach named *Short-term* addresses the *MemExplorer-Prime* problem (ME')[23] which is similar to MemExplorer but that considers an initial memory allocation for data structures. Thus, in addition to minimizing the time spent accessing data, the cost of changing the allocation of these data is also minimized. Table 1 points out the differences between these two problems in terms of inputs, outputs, constraints and objective function.

In [23] and [22], these two static problems have been tackled using a Variable Neighborhood Search-based approach hybridized with a Tabu Search. The VNS starts with greedy and random solutions and explores two different neighborhoods. The first neighborhood swaps data structures in memory banks provided that memory banks capacity are not exceeded, whereas the second one performs new allocations which may lead to exceed the memory bank capacity (in that case the allocation is repaired afterward). The Tabu Search procedure uses the same neighborhoods, with a dynamic tabu list whose tenure is smartly adapted during the search.

Table 1: *MemExplorer* and *MemExplorer-Prime* summary

Problem	Inputs	Constraints	Objective function	Outputs
ME	n, m, q, p, s_i, e_i, d_k, list of conflicts	memory bank capacity	access cost	single allocation of all data structures to memory banks
ME'	*same as above* + v, ℓ + initial allocation	*same as above*	access cost + transitional cost	*same as above*

For each time interval t, *Long-term* builds a preliminary allocation P_t called the *seed* allocation. The allocation for the time interval t, denoted by X_t is built upon that *seed* allocation. Sections 3.1 and 3.2 present two ways of building a solution based on the *seed* allocations at each iteration. The present paragraph focuses on the construction of the *seed* solution P_t at time interval t.

The *seed* allocation is selected among two candidate allocations. The first one is the *seed* allocation for the previous time interval P_{t-1}. The second candidate is the allocation found by solving MemExplorer on an instance built by gathering all the data structures, number of accesses to data, conflicts and cost involved from the current interval t to the last one. The total cost of both candidate allocations is then computed. This cost is the sum of two sub-costs. The first sub-cost is the cost that we would have if the candidate allocation was applied from the current time interval to the last one. The second sub-cost is the cost that should be paid for changing the memory mapping from the allocation of the previous time interval to the considered candidate allocation. The candidate allocation associated with the minimum total cost is selected as the *seed* allocation, and referred to as P_t.

Short-term builds an allocation for a time interval by solving MemExplorer-Prime on an instance built by gathering all the data structures, number of accesses to data, conflicts and costs involved only in the current time interval, and also by considering the allocation in the previous time interval.

Long-term takes into account the application's requirements (data structures, number of accesses to data, conflicts and costs) for the current and future time intervals.

We now explain why *Short-term* and *Long-term* produce solutions that tend to have specific features. When the capacity of memory banks is large,

Long-term tends to avoid moving data structures from a time interval to the next one. Indeed, for all t, a good *seed* allocation found at the first time intervals has a high probability to reach a total cost lesser than the other candidate allocation, and to be selected as a *seed* allocation for the next intervals. This is partly due to the fact that the changing cost of such an allocation is zero, as memory bank capacity allows for keeping the same allocation for all time intervals.

By contrast, *Short-term* minimizes the cost incurred by accessing data structures and the cost for changing the current allocation of data structures from a time interval to the next one. In general, the cost for moving data structures involved in a conflict is much less than the cost to be paid if the conflict is open. Thus, *Short-term* is prone to change the allocation of data structures from a time interval to the next one, but often at the expense of global solution quality.

Since *Long-term* has the advantage of stability over time, and *Short-term* has the advantage of flexibility, we are interested in a mid-term approach that combines the benefits of both approaches. We propose two mid-term approaches, the first one uses a rolling horizon and the other one weights the time interval requirements. These approaches are described below.

3.1. Mid-term approach with a rolling horizon

Long-term solves MemExplorer for all time intervals and *Short-term* addresses MemExplorer-Prime only once. Therefore, the mid-term approach should solve MemExplorer and MemExplorer-Prime for an intermediate number of times intervals. Based on this idea, the first version of mid-term approach uses a rolling horizon of length h. Algorithm 1 presents *Mid-term with a rolling horizon* which proceeds as follows. An allocation of data structures to memory banks has to be set for each time interval sequentially from the first time interval to the last one. As in *Long term*, the allocation at time interval t is based on a *seed* allocation (P_t) which is the best one out of three allocations: the *seed* allocation of the previous time interval (P_{t-1}), the allocation of MemExplorer for the following h time intervals (M_t), and the allocation of MemExplorer-Prime for the same h time intervals with the allocation of the previous interval as initial allocation (M'_t).

A complete solution is represented by a matrix X, which is composed of $T+1$ vectors X_t for all $t \in \{0, \ldots, T\}$. Vector X_t is the memory allocation at time interval t. Thus element x_{ti} of vector X_t represents the allocation of data structure i at time interval t. Element p_{ti} of vector P_t represents

Algorithm 1: Mid-term with a rolling horizon

1 **Input**: length of rolling horizon h
2 `// Initializations`
3
4 Initialize X_0: $x_{0i} = m + 1$, for all $i \in \{1, \ldots, n\}$
5 $P_0 \leftarrow X_0$, $p_{0i} = x_{0i}$
6 **for** $t \leftarrow 1$ **to** T **do**
7 `// Generate the three candidate allocations`
8 $P_{t-1} \leftarrow$ previous *seed* allocation
9 $M_t \leftarrow$ MemExplorer for h time intervals
10 $M'_t \leftarrow$ MemExplorer-Prime for h time intervals (initial alloc. X_{t-1})
11 `// Compute the cost and select the seed allocation`
12 Compute the cost of candidate allocations P_{t-1}, M_t and M'_t
13 Select the minimum cost allocation as the *seed* allocation P_t
14 `// Build the solution for current time interval`
15 $x_{ti} = p_{ti}$ for all $i \in \cup_{j=0}^{j=t} A_t$ and $x_{ti} = m + 1$ for the remaining data structures
16 **end**

the seed allocation of data structure i at time interval t. Initially, all data structures are allocated to the external memory, hence $x_{0i} = m + 1$ for all $i \in \{1, \ldots n\}$ (Algo 1 - line 3). As mentioned before, the *seed* allocation in the time interval t is denoted by P_t. The initial *seed* allocation P_0 is the initial memory allocation X_0 (line 4).

For each time interval t, this algorithm first generates three candidate allocations. The first one is the *seed* allocation P_{t-1} used to build the memory allocation of the previous time interval X_{t-1} (line 7). The second one, M_t is the allocation obtained by solving MemExplorer for data structures, number of accesses to data, conflicts and access costs involved from the time interval t to time interval $t+h-1$ (line 8). The last candidate allocation, M'_t is generated by MemExplorer-Prime for h time intervals using the memory mapping of the previous time interval X_{t-1}, as the initial allocation. Allocation M'_t takes into account the transitional cost for changing the allocation of data structures from the allocation of the previous time interval and the access cost (line 9).

For each of the three candidate allocations, the algorithm computes the

total cost which is the sum of two sub-costs. The first one is the access cost that we would have if the candidate allocation was applied from the current time interval t to the last one. The second sub-cost is the transitional cost for moving data structures from time interval $t-1$ to t that should be paid if the candidate allocation would be applied (line 11).

The selected allocation is the one whose total cost is minimum (line 12). The allocation X_t for time interval t is initialized to the *seed* allocation P_t (line 13); then, the data structures that are not required until the current time interval are allocated to the external memory.

3.2. Mid-term approach with weighted future requirements

Long-term is not bound to change data structures allocation, because future and present requirements are considered equally important. In particular, a very high cost conflict occurring in a quite distant future will have a significant impact on all the allocations to be computed before its actual occurrence. On the contrary, *Short-term* is prone to move data structures in order to find an allocation that only minimizes the cost of moving data structures and access cost incurred by the current time interval. This is of course done at the expense of anticipation of future requirements that are totally ignored, and it can be observed that this approach produces too many moves.

To overcome these drawbacks, we propose another mid-term approach which attempts to balance the number of moves between two successive time intervals. We have then weighted the requirements of each time interval. Future requirements are less and less weighted as they are far away from the current time interval. This mid-term approach considers future requirements but giving more importance to the one appearing in the nearest future. Thus, we avoid the drawbacks of *Long-term* while taking advantage of the flexible nature of *Short-term*. Algorithm 2 shows *Mid-term with weighted future requirements*.

First, the decay rate $\alpha \in [0, 1]$ has to be set. At time interval t, the cost of all the conflicts that arise from time interval $t+1$ are multiplied by α, those arising from time interval $t+2$ are multiplied by α^2, those arising from time interval $t+k$ are multiplied by α^k, until the end of the horizon. *Long-term* corresponds to $\alpha = 1$ whereas *Short-term* can be computed with $\alpha = 0$.

Numerical experiments suggest that the best trade off between these two extreme strategies can be found based on the ratio between memory bank

Algorithm 2: Mid-term with weighted future requirements

Input: Decay rate α
`// Initialization`
Initialize X_0: $x_{0i} = m + 1$, for all $i \in \{1, \dots n\}$
$P_0 \leftarrow X_0$
for $t \leftarrow 1$ **to** T **do**
    `// Generate candidate allocations`
    $P_{t-1} \leftarrow$ previous *seed* allocation
    $M_t \leftarrow$ MemExplorer (α-weighted requirements from interval t to T)
    $M'_t \leftarrow$ MemExplorer-Prime (α-weighted requirements from interval t to T and initial allocation X_{t-1})
    `// Compute the cost and select the seed allocation`
    Compute the cost of candidate allocations P_{t-1}, M_t and M'_t
    Select the minimum cost allocation as the *seed* allocation P_t
    `// Build final allocation for current time interval`
    $x_{ti} = p_{ti}$ for all $i \in \cup_{j=0}^{j=t} A_t$ and $x_{ti} = m + 1$ for the remaining data structures
end

capacity and the size of data structures, but it also depends on p, ℓ and v. The decay rate is then empirically defined by:

$$\alpha = \begin{cases} 1 \quad \text{if } \dfrac{\sum_{j=1}^{m} c_j}{\sum_{i=1}^{n} s_i} \geq 1 \\ \min\left(1, \dfrac{v}{p \cdot \ell} \cdot \dfrac{\sum_{j=1}^{m} c_j}{\sum_{i=1}^{n} s_i}\right) \text{ otherwise} \end{cases} \tag{1}$$

Whenever the memory bank capacity is large enough to accommodate all data structures, *Long-term* tends to be very efficient as very few modifications from a time interval to the next one may be needed. Then, if the ratio $\frac{\sum_{j=1}^{m} c_j}{\sum_{i=1}^{n} s_i}$ increases, then a large α is suitable. In addition, a high cost for moving data structures also leads to favor steady assignments for data structures. Consequently, a large $\frac{v}{\ell}$ leads to high values for α. However, a large penalty cost p makes it more and more costly to access data structures that are in the external memory. Then, if memory bank capacity is modest, it might be

profitable to move the data structures that are required at each time interval in the memory banks, hence favoring flexibility. So, if p is large, then a smaller decay rate α should be desirable to reach high-quality solutions.

Initialization is the same as in the Mid-term approach with a rolling horizon (Algo 2 -line 3-4), and a time interval allocation is generated as in Algorithm 1.

At each time interval t, Algorithm 2 generates three candidate allocations (line 7-9). The first one is the *seed* allocation P_{t-1} used to build the memory allocation of the previous time interval X_{t-1} (line 7). The second one, M_t is the allocation obtained by solving MemExplorer by weighting the requirements for the current time interval t to the last one as explained above (line 8). The cost of conflicts occurring at time interval t are multiplied by α^{r-t} for all $r \in \{t, \ldots, T\}$. Hence, the future requirements are less and less weighted as they are far away from the current time interval. The last candidate allocation, M'_t is the allocation found by MemExplorer-Prime weighting the requirements and using the memory mapping of the previous time interval X_{t-1} as the initial allocation (line 9). The rest of the algorithm is similar to Algorithm 1.

4. Computational Results

4.1. Implementation

From our experience, it appears that the solution quality of *Long-term* and *Short-term* is very sensitive to the ratio of the memory bank capacity over the total size of data structures. We have used two groups of instances for testing our algorithms. Group A is composed of real and artificial instances for which memory bank capacity is too small for accommodating all the data structures (hence the external memory is necessary). The instances in Group B are the same, but the memory bank capacity is large enough for accommodating all the data structures.

Real instances originate from electronic design problems addressed in the Lab-STICC laboratory. Artificial instances are composed of the last eleven instances of Table 2, and originate from DIMACS [15]. They have been enriched by randomly generating conflict costs, number and capacity of memory banks, sizes and number of accesses to data structures. For each instance, we have divided the conflicts and data structures into different time intervals. All instances are available online for download [18].

We have tested our approaches with artificially large instances to assess their practical use for forthcoming needs, as technology tends to integrate more and more functionalities in embedded systems.

All algorithms have been implemented in C++ and compiled with `gcc 4.11` in Linux `OS 10.04` using an Intel Pentium IV processor system at 3 GHz with 1 GByte RAM. The time q spent by the processor to access data structures to memory banks is set to 1 ms. We have set factor p to access data structures to external memory to 16, as with TI C6201. The cost v for moving a data structure from the external memory to memory banks and vice versa is set to 4 ms/kB and cost for moving data structures between memory banks is equal to 1 ms/kB.

From our experiments for the mid-term approach with rolling horizon, we empirically set h to 2 for the instances with less than 8 conflicts ($o < 8$), and for the remaining instances h is set to 3. The decay rate α is calculated using Equation (1).

4.2. Results

In Tables 2 and 3, we report the results of the ILP formulation [23]. The instances are sorted by non decreasing number of conflicts.

The first three columns show the main characteristics of instances such as instance's name, number of data structures n, conflicts o, memory bank m and time intervals T. The next columns present the cost reached by the ILP formulation solved with Xpress-MP. The stopping criterion is set to one hour after the last integer value is reached. The best solution found so far (if any) is then returned by the solver. For the ILP we display the lower bound and the best solution. The solver cannot find any result for some large instances due to memory issues.

Tables 4 and 5 present the main results concerning the iterative approaches. For each instance, we have run each iterative approach twelve times. Table 4 displays the objective function value of the best solution found and its CPU time. Additionally, we have computed the standard deviation (σ) and the coefficient of variance (CV) for each approach. The coefficient of variance is the ratio of the standard deviation to the mean cost. CV allows us to compare the variance of approaches using instances which have widely different means. The coefficient of variance shows that iterative approaches do not have high variability building solutions. Since all iterative approaches have very similar values for CV (around 0.01), they are not reported in the tables.

At the bottom of these tables, we report the number of optimal solutions and the number of best solutions returned by each approach. We also report the average CPU time and the gap to optimality. A star has been added to the instance's name to locate the one for which the optimal value is known. Mid-term approach with weighted future needs always finds the optimal solution when it is known. For the other cases, it reaches the best cost except for the instance `mulsol_i4dyA` in group A and `mpeg2enc2dy2B` in group B. Mid-term approach with rolling horizon returns the best solution for the instance `mulsol_i4dyA`. For the instance `mpeg2enc2dy2A` *Long-term* reaches the best objective function value. The reason why mid-term does not return the best solution for `mpeg2enc2dy2A` is that the decay rate α is not one (see the long-term approach).

In group A, the number of best solutions reported by the mid-term approach with weighted needs is 35, compared to 12 with the mid-term approach with rolling horizon, 15 with *Short-term*, 11 with *Long-term* and 24 with the ILP model. In group B, the mid-term approach with weighted needs reaches the best solution for 36 instances, while *Long-term* does it for 25 instances, the mid-term approach with rolling horizon does it for 22 instances and *Short-term* reaches the best solution for only 14 instances.

In group B, the mid-term approach with weighted needs works with α equals to one, *i.e*, the future requirements are as important as the present ones, as in *Long-term*. However, this mid-term approach reports better results than *Long-term*, because it considers three candidate allocations at each time interval.

In general, *Long-term* reaches the best results when the capacity of memory banks is sufficiently large, because it requires less allocation changes from a time interval to the next one. When the cost of moving data structures is small enough, then *Short-term* reaches the best results.

The minimum average gap is reached by the mid-term approach with weighted needs, which shows the competitiveness of this approach over the other ones. The highest CPU time is obtained by the ILP formulation and the lowest one is reached by the *Short-term* approach, which is not a surprising result.

In order to assess the benefit of comparing three candidate allocations in the mid-term approach with weighted future needs, we have tested this approach using only two candidate allocations. The impact of using three candidate allocations is significant for large instances only. For this reason, Table 6 displays the objective value returned by using the approach without

Table 2: ILP results for Group A

Instances			ILP			
Name	$n\backslash o\backslash m$	T	cost	lower bound	Optimal	(s)
gsm_newdyA	6\5\2	2	339.00	339.00	yes	0.05
gsm_newdy2A	6\5\2	2	7,808.00	7,808.00	yes	0.03
compressdy2A	6\6\2	3	4,260,352.00	4,260,352.00	yes	0.08
incompressdy2A	10\6\2	3	4,305,152.00	4,305,152.00	yes	0.08
lmsbdy2A	8\7\2	3	38,805,539.00	38,805,539.00	yes	0.13
cjpegdy2A	11\7\2	4	28,099,184.00	28,099,184.00	yes	0.17
incjpegdy2A	11\7\2	4	18,883,184.00	18,883,184.00	yes	0.17
lmsbvdyA	8\8\2	3	4,226,438.00	4,226,438.00	yes	0.14
lmsbvdy2A	8\8\2	3	20,051,919.00	20,051,919.00	yes	0.19
lmsbvdyexpdyA	8\8\2	4	67,250,834.00	67,250,834.00	yes	0.14
lmsbv01dy2A	8\8\2	4	67,250,834.00	67,250,834.00	yes	0.16
spectraldyA	9\8\2	3	29,512.00	29,512.00	yes	0.11
adpcmdy2A	10\8\2	3	44,192.00	44,192.00	yes	0.14
inadpcmdy2A	10\8\2	3	46,420.00	46,420.00	yes	0.17
inspectraldyA	10\8\2	3	41,232.00	41,232.00	yes	0.08
incompressdyA	10\10\2	3	6,714,272.00	6,714,272.00	yes	0.05
inexample10A	10\10\2	3	133.36	133.36	yes	0.13
inlmsbdy2A	16\14\2	4	48,341,034.00	48,341,034.00	yes	0.16
ingsm_newdy2A	18\15\2	3	2,266.24	2,266.24	yes	0.05
ingsmdyA	19\18\2	5	7,237.00	7,237.00	yes	0.27
involterradyA	24\18\2	3	411.00	411.00	yes	0.33
inexample50A	50\50\2	5	1,022.47	1,022.47	yes	0.42
treillisdy2A	33\61\2	6	1,805.56	1,805.56	yes	0.83
inturbocodedyA	36\66\3	3	23,052.00	23,052.00	yes	0.22
mpeg2enc2dy2A	130\239\2	12	9,904.09	8,890.37	no	3,615.50
alidy2A	192\960\6	48	108,699.00	79,754.40	no	3,731.87
myciel7dyA	191\2360\4	24	486,449.00	232,212.00	no	3,675.82
zeroin_i3dyA	206\3540\15	35	750,128.00	375,064.00	no	4,642.19
zeroin_i2dyA	211\3541\15	35	-	-	no	3,600.00
r125.5dyA	125\3838\18	38	-	-	no	3,600.00
mulsol_i2dyA	188\3885\16	39	764,693.00	281,743.00	no	4,459.82
mulsol_i1dyA	197\3925\25	39	-	-	no	3,600.00
mulsol_i4dyA	185\3946\16	39	-	-	no	3,600.00
mulsol_i5dyA	186\3973\16	40	748,781.00	264,240.00	no	4,130.75
zeroin_i1dyA	211\4100\25	41	-	-	no	3,600.00
r125.1cdyA	125\7501\23	75	-	-	no	3,600.00
fpsol2i3dyA	425\8688\15	87	-	-	no	3,600.00
Number of optimal solutions			24			
Average CPU (s)						1,374.60

Table 3: ILP results for Group B

Instances			ILP			
Name	$n\backslash o\backslash m$	T	cost	lower bound	Optimal	(s)
gsm_newdyB	6\5\2	2	336.00	336.00	yes	0.016
gsm_newdy2B	6\5\2	2	7,808.00	7,808.00	yes	0.016
compressdy2B	6\6\2	3	334,400.00	334,400.00	yes	0.016
incompressdy2B	10\6\2	3	356,608.00	356,608.00	yes	0.109
lmsbdy2B	8\7\2	3	7,393,285.00	7,393,285.00	yes	0.032
cjpegdy2B	11\7\2	4	4,466,800.00	4,466,800.00	yes	0.093
incjpegdy2B	11\7\2	4	4,466,800.00	4,466,800.00	yes	0.156
lmsbvdyB	8\8\2	3	4,196,158.00	4,196,158.00	yes	0.031
lmsbvdy2B	8\8\2	3	4,323,294.00	4,323,294.00	yes	0.031
lmsbvdyexpdyB	8\8\2	4	4,334,256.00	4,334,256.00	yes	0.078
lmsbv01dy2B	8\8\2	4	4,334,256.00	4,334,256.00	yes	0.093
spectraldyB	9\8\2	3	7,604.00	7,604.00	yes	0.031
adpcmdy2B	10\8\2	3	44,192.00	44,192.00	yes	0.055
inadpcmdy2B	10\8\2	3	46,420.00	46,420.00	yes	0.078
inspectraldyB	10\8\2	3	9,488.00	9,488.00	yes	0.145
incompressdyB	10\10\2	3	695,476.00	695,476.00	yes	0.218
inexample10B	10\10\2	3	109.36	109.36	yes	0.093
inlmsbdy2B	16\14\2	4	16,879,628.00	16,879,628.00	yes	0.465
ingsm_newdy2B	18\15\2	3	2,266.24	2,266.24	yes	0.07
ingsmdyB	19\18\2	5	5,067.00	5,067.00	yes	0.805
involterradyB	24\18\2	3	411.00	411.00	yes	0.175
inexample50B	50\50\2	5	511.59	255.80	no	3,647.01
treillisdy2B	33\61\2	6	1,805.56	1,805.56	yes	0.515
inturbocodedyB	36\66\3	3	6,402.00	6,402.00	yes	1.15
mpeg2enc2dy2B	130\239\2	12	9,830.44	4,915.22	no	3,637.47
alidy2B	192\960\6	48	93,330.00	46,665.00	no	3,767.53
myciel7dyB	191\2360\4	24	300,127.00	147,560.00	no	3,610.73
zeroin_i3dyB	206\3540\15	35	-	-	no	3,600.00
zeroin_i2dyB	211\3541\15	35	-	-	no	3,600.00
r125.5dyB	125\3838\18	38	-	-	no	3,600.00
mulsol_i2dyB	188\3885\16	39	440,887.00	215,011.00	no	3,704.72
mulsol_i1dyB	197\3925\25	39	-	-	no	3,600.00
mulsol_i4dyB	185\3946\16	39	-	-	no	3,600.00
mulsol_i5dyB	186\3973\16	40	430,690.00	220,528.00	no	3,693.96
zeroin_i1dyB	211\4100\25	41	-	-	no	3,600.00
r125.1cdyB	125\7501\23	75	-	-	no	3,600.00
fpsol2i3dyB	425\8688\15	87	-	-	no	3,600.00
Number of optimal solutions			23			
Average CPU (s)						1,418.00

Table 4: Iterative approaches results for Group A

Instance	Long-term		Short-term		Mid-term rolling horizon		Mid-term weighted needs		
Name	cost	(s)	cost	(s)	cost	(s)	cost	(s)	α
*gsm_newdyA	339.00	0.21	339.00	0.05	339.00	0.21	339.00	0.21	0.17
*gsm_newdy2A	7,808.00	0.02	7,808.00	0.03	7,808.00	0.04	7,808.00	0.04	0.17
*compressdy2A	4,262,400.00	0.43	4,260,352.00	0.43	4,260,352.00	0.64	4,260,352.00	0.69	0.14
*incompressdy2A	4,305,152.00	0.44	4,305,152.00	0.22	4,305,152.00	0.59	4,305,152.00	0.63	0.05
*lmsbdy2A	54,510,627.00	0.51	38,823,971.00	0.27	38,807,587.00	0.74	38,805,539.00	0.77	0.14
*cjpegdy2A	50,049,648.00	0.73	28,099,184.00	0.31	50,049,648.00	0.99	28,099,184.00	1.20	0.07
*incjpegdy2A	40,850,032.00	0.68	18,883,184.00	0.26	40,833,648.00	0.92	18,883,184.00	1.21	0.07
*lmsbvdyA	4,226,438.00	0.55	4,226,795.00	0.31	4,226,438.00	0.78	4,226,438.00	0.89	0.12
*lmsbvdy2A	20,074,461.00	0.39	20,051,919.00	0.16	20,051,919.00	0.55	20,051,919.00	0.55	0.14
*lmsbvdyexpdyA	67,300,041.00	0.82	67,285,680.00	0.52	67,300,041.00	1.32	67,250,834.00	1.43	0.10
*lmsbv01dy2A	67,300,041.00	0.81	67,285,680.00	0.55	67,300,041.00	1.32	67,250,834.00	1.42	0.10
*spectraldyA	43,560.00	0.58	29,551.00	0.31	41,254.00	0.75	29,512.00	0.79	0.10
*adpcmdy2A	44,192.00	0.06	49,120.00	0.08	44,192.00	0.10	44,192.00	0.13	0.11
*inadpcmdy2A	46,420.00	0.06	46,420.00	0.06	46,420.00	0.10	46,420.00	0.11	0.11
*inspectraldyA	73,088.00	0.64	41,232.00	0.39	73,088.00	0.92	41,232.00	1.18	0.05
*incompressdyA	6,714,272.00	0.61	6,714,272.00	0.39	6,714,272.00	0.92	6,714,272.00	1.07	0.05
*inexample10A	525.66	0.78	133.36	0.38	390.31	1.20	133.36	1.20	0.10
*inlmsbdy2A	205,299,844.00	1.02	48,341,034.00	0.41	172,857,445.00	1.44	48,341,034.00	1.82	0.06
*ingsm_newdy2A	2,266.24	0.07	2,266.24	0.07	2,266.24	0.12	2,266.24	0.12	0.06
*ingsmdyA	34,344.00	1.28	7,237.00	0.37	23,752.00	1.59	7,237.00	2.01	0.05
*involterradyA	411.00	0.12	411.00	0.13	411.00	0.20	411.00	0.18	0.06
*inexample50A	3,502.88	3.64	1,025.21	1.43	2,649.09	4.84	1,022.47	6.46	0.04
*treillisdy2A	1,805.56	0.50	1,867.00	0.57	1,810.19	0.76	1,805.56	0.96	0.15
*inturbocodedyA	51,864.00	2.02	23,070.00	0.87	48,360.00	2.79	23,052.00	3.03	0.06
mpeg2enc2dy2A	9,812.31	4.20	10,264.03	3.01	10,190.31	5.44	10,190.56	5.53	0.05
alidy2A	1,116,698.00	160.48	111,382.00	29.91	305,380.00	124.62	108,693.00	64.08	0.03
myciel7dyA	800,622.00	377.08	555,610.00	41.07	547,066.00	265.90	472,091.00	179.89	0.10
zeroin_i3dyA	997,617.00	1,463.72	903,824.00	107.05	706,531.00	688.82	703,390.00	483.15	0.10
zeroin_i2dyA	935,763.00	1,186.34	817,312.00	114.28	641,684.95	708.22	634,366.00	457.12	0.10
r125.5dyA	1,521,866.00	1,028.86	689,553.00	78.90	672,984.00	471.11	621,773.00	332.45	0.13
mulsol_i2dyA	1,271,960.00	1,286.69	1,120,751.00	129.82	746,079.80	790.90	743,182.00	541.44	0.11
mulsol_i1dyA	1,276,746.00	1,396.13	1,124,113.00	153.04	740,119.00	768.76	709,795.00	649.39	0.08
mulsol_i4dyA	1,175,568.00	1,657.35	1,091,936.00	137.00	661,234.00	734.38	695,340.00	596.88	0.11
mulsol_i5dyA	1,270,156.00	1,480.78	1,160,151.00	143.35	703,946.00	781.72	685,759.00	617.17	0.11
zeroin_i1dyA	860,190.00	1,691.16	881,192.00	165.62	648,437.00	806.58	647,422.00	620.25	0.08
r125.1cdyA	2,339,105.00	950.50	954,699.00	152.10	931,199.00	726.73	919,103.00	627.80	0.16
fpsol2i3dyA	4,137,879.00	1,062.37	3,565,114.00	479.62	3,640,869.00	1,453.46	3,150,378.00	1,784.11	0.05
Optimal sol.	10		15		11		24		
Best sol.	11		15		12		35		
Aver. CPU		371.96		47.12		225.69		188.85	
Aver. gap (%)	96		5		49		1		

Table 5: Iterative approaches results for Group B

Instance	Long-term		Short-term		Mid-term rolling horizon		Mid-term weighted needs		
Name	cost	(s)	cost	(s)	cost	(s)	cost	(s)	α
*gsm_newdyB	336.00	0.03	338.00	0.04	336.00	0.05	336.00	0.05	1.00
*gsm_newdy2B	7,808.00	0.03	7,808.00	0.03	7,808.00	0.04	7,808.00	0.04	1.00
*compressdy2B	334,400.00	0.05	334,912.00	0.06	334,400.00	0.08	334,400.00	0.09	1.00
*incompressdy2B	356,608.00	0.08	357,120.00	0.07	356,608.00	0.11	356,608.00	0.12	1.00
*lmsbdy2B	7,393,285.00	0.06	7,395,333.00	0.08	7,393,285.00	0.12	7,393,285.00	0.12	1.00
*cjpegdy2B	4,466,800.00	0.10	4,466,800.00	0.10	4,466,800.00	0.16	4,466,800.00	0.18	1.00
*incjpegdy2B	4,466,800.00	0.09	4,466,800.00	0.11	4,466,800.00	0.17	4,466,800.00	0.18	1.00
*lmsbvdyB	4,196,158.00	0.07	4,196,158.00	0.07	4,196,158.00	0.11	4,196,158.00	0.12	1.00
*lmsbvdy2B	4,323,294.00	0.06	4,323,294.00	0.07	4,323,294.00	0.10	4,323,294.00	0.10	1.00
*lmsbvdyexpdyB	4,334,256.00	0.09	4,334,256.00	0.10	4,334,256.00	0.16	4,334,256.00	0.16	1.00
*lmsbv01dy2B	4,334,256.00	0.09	4,334,256.00	0.10	4,334,256.00	0.16	4,334,256.00	0.16	1.00
*spectraldyB	7,604.00	0.07	7,604.00	0.07	7,604.00	0.12	7,604.00	0.12	1.00
*adpcmdy2B	44,192.00	0.06	49,120.00	0.07	44,192.00	0.10	44,192.00	0.11	1.00
*inadpcmdy2B	46,420.00	0.06	46,420.00	0.06	46,420.00	0.10	46,420.00	0.10	1.00
*inspectraldyB	9,488.00	0.92	9,608.00	0.30	9,488.00	1.30	9,488.00	1.27	1.00
*incompressdyB	695,476.00	0.96	695,536.00	0.49	695,476.00	1.38	695,476.00	1.39	1.00
*inexample10B	109.36	1.42	109.36	0.47	109.36	1.91	109.36	1.94	1.00
*inlmsbdy2B	16,879,628.00	0.62	16,879,628.00	0.29	16,879,628.00	0.73	16,879,628.00	1.10	1.00
*ingsm_newdy2B	2,266.24	0.07	2,266.24	0.08	2,266.24	0.11	2,266.24	0.12	1.00
*ingsmdyB	5,067.00	0.88	5,067.00	0.34	5,067.00	0.91	5,067.00	1,65	1.00
*involterradyB	411.00	0.11	411.00	0.13	411.00	0.20	411.00	0,18	1.00
inexample50B	516.04	9.77	511.59	1.63	510.93	7.97	510.59	10.40	0.80
*treillisdy2B	1,805.56	0.52	1,867.00	0.57	1,810.19	0.72	1,805.56	0.90	1.00
*inturbocodedyB	6,402.00	0.26	6,501.00	0.25	6,402.00	0.42	6,402.00	0.41	1.00
mpeg2enc2dy2B	9,812.31	4.14	10,264.03	3.05	10,190.56	5.64	10,190.31	5.43	0.93
alidy2B	60,573.00	352.48	60,602.00	18.41	60,571.00	41.28	59648.00	38.97	0.51
myciel7dyB	162,011.00	41.86	166,429.00	9.85	166,209.00	23.81	161,684.00	62.45	1.00
zeroin_i3dyB	211,585.00	470.51	219,654.00	22.73	215,089.00	46.81	211,581.00	505.04	1.00
zeroin_i2dyB	210,077.00	332.42	216,326.00	23.83	212,504.00	47.43	210,077.00	437.17	1.00
r125.5dyB	219,977.00	331.59	221,419.00	11.35	220,735.00	25.66	219,910.00	481.68	1.00
mulsol_i2dyB	231,671.00	408.25	238,770.00	18.72	235,755.00	38.79	231,650.00	496.25	1.00
mulsol_i1dyB	222,293.00	420.03	230,076.00	25.38	225,071.00	50.87	222,291.00	574.03	1.00
mulsol_i4dyB	231,178.00	387.62	239,978.00	17.69	234,676.00	38.54	231,154.00	452.40	1.00
mulsol_i5dyB	235,038.00	353.40	242,198.00	18.71	239,640.00	40.15	235,011.00	439.18	1.00
zeroin_i1dyB	231,057.00	567.14	237,775.00	28.38	233,821.00	57.82	231,036.00	645.84	1.00
r125.1cdyB	403,032.00	828.94	404,687.00	26.31	403,410.00	58.85	402,724.00	1,269.46	1.00
fpsol2i3dyB	515,555.00	1,374.28	527,995.00	111.67	525,609.00	237.13	515,404.00	1,999.16	1.00
Optimal sol.	23		14		22		23		
Best sol.	25		14		22		36		
Aver. CPU		159.17		9.23		19.73		200.76	
Aver. gap (%)	6		7		6		6		

the previous *seed* allocation P_{t-1}, without MemExplorer M_t and without MemExplorer-Prime for the largest instances of Group A.

The gap in Table 6 measures the impact of not using one of the candidate allocation in the mid-term approach. If the approach does not consider the MemExplorer allocation as a candidate allocation, then the quality of mid-term solution decreases on average by 28%. If MemExplorer-Prime is not used, the solution quality drops by an average of 10%. Finally, if the previous *seed* allocation is turned off, the quality solution decreases by an average of 2%.

Table 6: Assessing mid-term approach with weighted future needs

Instance	Mid weighted	without P_{t-1}	gap	without M_t	gap	without M'_t	gap
*treillisdy2A	1,805.56	1,810.19	0.00	1,810.25	0.00	1,986.56	0.10
mpeg2enc2dy2A	10,190.56	10,190.56	0.00	10,190.56	0.00	12,046.34	0.18
alidy2A	108,693.00	108,860.00	0.00	111,849.00	0.03	109,475.00	0.01
myciel7dyA	472,091.00	480,514.00	0.02	571,190.00	0.21	540,903.00	0.15
zeroin_i3dyA	703,390.00	700,728.00	0.00	939,445.00	0.34	806,031.00	0.15
zeroin_i2dyA	634,366.00	642,026.00	0.01	832,730.00	0.31	685,407.00	0.08
r125.5dyA	621,773.00	624,884.00	0.01	687,131.00	0.11	647,384.00	0.04
mulsol_i2dyA	743,182.00	746,763.00	0.00	1,146,375.00	0.54	830,451.00	0.12
mulsol_i1dyA	709,795.00	729,262.00	0.03	1,092,377.00	0.54	786,813.00	0.11
mulsol_i4dyA	695,340.00	732,603.00	0.05	1,131,758.00	0.63	746,599.00	0.07
mulsol_i5dyA	685,759.00	732,451.00	0.07	1,150,778.00	0.68	769,228.00	0.12
zeroin_i1dyA	647,422.00	670,385.00	0.04	855,477.00	0.32	713,735.00	0.10
r125.1cdyA	919,103.00	914,700.00	0.00	944,898.00	0.03	1,018,998.00	0.11
fpsol2i3dyA	3,150,378.00	3,185,194.00	0.01	3,594,043.00	0.14	3,215,902.00	0.02
Average gap (%)			1.77		27.66		9.69

4.3. Statistic analysis

Cost analysis. In order to achieve a fair comparison, we use the non parametric Friedman test [7]. We use this test to detect differences in the performance of iterative approaches and ILP formulation using the results presented by the above tables. We can use this test because the result over instances are mutually independent. Thus we apply the Friedman test for costs comparing (univariate model [3]) the performance in terms of solution quality.

The null hypothesis assumes that for each instance, the ranking of the approaches is equally likely. The null hypothesis is rejected at the level of significance 0.95 if the p-value is less than 0.05. Friedman test is 6.9488 for the cost and its p-value is 1.364×10^{-11}. We can reject the null hypothesis

for the cost at the level of significance 0.95. Then, there exists at least one approach whose performance is different from at least one of the other ones.

As the null hypothesis of Friedman test was rejected, we can use a *Post-Hoc* paired comparison to know if two metaheuristics are different [5]. Table 7 summarizes p-values for the paired comparisons for the cost. The bold values means the approaches are different in terms of cost.

Table 7: p-value for Post-Hoc paired comparison for the cost

	Long-term	Short-term	Mid rolling	Mid weighted
ILP	8.54×10^{-1}	5.28×10^{-1}	9.20×10^{-1}	$\mathbf{3.41 \times 10^{-7}}$
Long-term	-	9.90×10^{-1}	3.52×10^{-1}	$\mathbf{4.60 \times 10^{-10}}$
Short-term	-	-	1.43×10^{-1}	$\mathbf{1.38 \times 10^{-11}}$
Mid rolling	-	-	-	$\mathbf{3.01 \times 10^{-5}}$

For these two groups of instances, we conclude that mid-term approach with weighted future needs has the best performance in terms of cost. There does not exist a significant difference between the performance of the ILP formulation, *Long-term*, *Short-term* and mid-term with rolling horizon approaches.

Impact of the seed solution. We also apply Friedman test to complete the assessment of the mid-term approach with weighted future requirements using the data in Table 6. The value for this test is 5.3358 whose p-value is 6.546×10^{-7}. Then we can conclude that there exists at least one candidate solution whose impact in the mid-term approach is different from at least one of the other ones. Table 8 presents the Post-Hoc comparison for evaluating the mid-term approach with weighted approach.

Table 8: p-value for Post-Hoc for assessing mid-term with weighted needs

	P_{t-1}	M_t	M'_t
Mid weighted	4.28×10^{-1}	$\mathbf{2.42 \times 10^{-6}}$	$\mathbf{4.79 \times 10^{-7}}$
P_{t-1}	-	$\mathbf{2.79 \times 10^{-3}}$	$\mathbf{7.47 \times 10^{-4}}$
M_t	-	-	9.90×10^{-1}

From Tables 6 and 8, we can deduce that the candidate solutions produced by MemExplorer and MemExplorer-Prime have the same impact on the mid-term approach with weighted needs. The statistic analysis confirms that the

previous seed solution has quite a small impact on solution quality. Indeed, Table 6 shows that the improvement due to this candidate is on average 2%, which is worth keeping.

5. Conclusion

We have presented two mid-term approaches for iteratively building a solution for the dynamic memory allocation problem that arises in the design of embedded systems. These approaches are based on two static subproblems for addressing memory allocation that have been devised earlier for addressing simplified problem versions. The first approach builds a solution for a time interval by taking into account the requirements involved during the rolling horizon. The other mid-term approach integrates the future needs through a decay rate. The statistic tests show that the mid-term approach with a rolling horizon has not a significant performance in terms of cost and running time. Mid-term approach with weighted future requirements reached the best performance in terms of cost but not in terms of CPU time. Indeed, the main drawback of these approaches is that they do not consider the complete problem in all time intervals. Consequently, future work should concentrate on designing a global approach that builds a solution for all time intervals such as a Greedy Randomized Adaptive Search Procedure or a Variable Neighborhood Search.

References

[1] D. Atienza, S. Mamagkakis, F. Poletti, J. Mendias, F. Catthoor, L. Benini, and D. Soudris. Efficient system-level prototyping of power-aware dynamic memory managers for embedded systems. *Integration, the VLSI Journal*, 39(2):113–130, 2006.

[2] C-T. Chang. Optimization approach for data allocation in multidisk database. *European Journal of Operational Research*, 143(1):210–217, 2002.

[3] M. Chiarandini, L. Paquete, M. Preuss, and E. Ridge. Experiments on metaheuristics: Methodological overview and open issues. Technical Report DMF-2007-03-003, The Danish Mathematical Society, Denmark, 2007.

[4] A. Chimientia, L. Fanucci, R. Locatellic, and S. Saponarac. VLSI architecture for a low-power video codec system. *Microelectronics Journal*, 33(5):417–427, 2002.

[5] W.J. Conover. *Practical nonparametric statistic*. Wiley, New York, USA, third edition, 1999.

[6] P. Coussy, E. Casseau, P. Bomel, A. Baganne, and E. Martin. A formal method for hardware IP design and integration under I/O and timing constraints. *ACM Transactions on Embedded Computing System*, 5(1):29–53, 2006.

[7] M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32:675–701, 1937.

[8] M. Iverson, F. Ozguner, and L. Potter. Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment. *IEEE Transactions on Computers*, 48(12):1374–1379, 1999.

[9] N. Julien, J. Laurent, E. Senn, and E. Martin. Power consumption modeling and characterization of the TI C6201. *IEEE Micro*, 23(5):40–49, 2003.

[10] P.K. Krause. The complexity of register allocation. *Discrete Applied Mathematics*, 2013.

[11] W. Lee and M. Chang. A study of dynamic memory management in C++ programs. *Comp. Languages Systems and Structures*, 28(3):237–272, 2002.

[12] M-H. Lin. An optimal workload-based data allocation approach for multidisk databases. *Data and Knowledge Engineering*, 68(5):499 – 508, 2009.

[13] N. Mladenović and P. Hansen. Variable neighbourhood decomposition search. *Computers and Operations Research*, 24(11):1097–1100, 1997.

[14] A. Munier-Kordon and J-B. Note. A buffer minimization problem for the design of embedded systems. *European Journal of Operational Research*, 164(3):669–679, 2005.

[15] D. Porumbel. DIMACS graphs: Benchmark instances and best upper bound, 2009.

[16] Roadef/EURO. Challenge 2012: Machine reassignment.

[17] M. Soto. *Optimization methods for the memory allocation problems in embedded systems*. PhD thesis, Université de Bretagne-Sud, Lorient, France, 2011.

[18] M. Soto, A. Rossi, and M. Sevaux. Instances for the dynamic memory allocation problem. Available for download.

[19] M. Soto, A. Rossi, and M. Sevaux. Two upper bounds on the chromatic number. In *Proc. CTW09 8th Cologne-Twente Workshop on Graphs and Combinatorial Optimization*, pages 191–194, Paris, France, 2009.

[20] M. Soto, A. Rossi, and M. Sevaux. Exact and metaheuristic approaches for a memory allocation problem. In *Proc. EU/MEeting, workshop on the metaheuristics community*, pages 25–29, Lorient, France, 2010.

[21] M. Soto, A. Rossi, and M. Sevaux. Métaheuristiques pour l'allocation de mémoire dans les systèmes embarqués. In *Proc. ROADEF 11e congrès de la société Française de Recherche Opérationelle est d'Aide à la Décision*, pages 35–43, Toulouse, France, 2010.

[22] M. Soto, A. Rossi, and M. Sevaux. A mathematical model and a metaheuristic approach for a memory allocation problem. *Journal of Heuristics*, 18(1):149–167, mar 2011.

[23] M. Soto, A. Rossi, and M. Sevaux. Two iterative metaheuristic approaches to dynamic memory allocation for embedded systems. In P. Merz and J-K. Hao, editors, *Evolutionary Computation in Combinatorial Optimization - 11th European Conference, EvoCOP 2011, Torino, Italy, April 27-29, 2011. Proceedings*, volume 6622 of *Lecture Notes in Computer Science*, pages 250–261. Springer, 2011.

[24] C. Wilbaut, S. Salhi, and S. Hanafi. An iterative variable-based fixation heuristic for the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, 199(2):339–348, 2009.

[25] S. Wuytack, F. Catthoor, L. Nachtergaele, and H. De Man. Power exploration for data dominated video application. In *Proc. IEEE International Symposium on Low Power Electronics and Design*, pages 359–364, Monterey, CA, USA, 1996.