Available online at www.sciencedirect.com

ScienceDirect

IFAC-PapersOnLine 48-3 (2015) 507–512

# Safety-Level Aware Bin-Packing Approach for Control Functions Assignment

**Mohamed BENAZOUZ** **Jean Marc FAURE**

*mohamed.benazouz@ens-cachan.fr* *jean-marc.faure@ens-cachan.fr*
*LURPA, ENS Cachan, France,*

**Abstract:** The assignment of functions to controllers is a crucial step when building an operational control system architecture. We identified this problem as a Multiple Choice Vector Bin-Packing with Conflicts that is a generalization of the one-dimensional Bin-Packing problem. Such problems are known to be strongly NP-Hard and exact techniques to solve them are too time and/or space consuming because of the combinatorial explosion. Therefore, in this paper, we propose a fast First-Fit Decreasing based heuristic that derives an optimized number of controllers in polynomial time. The objective is to minimize the global cost of controllers while satisfying safety constraints. We show through experiments that the adopted approach allows us to obtain values that are in average very close to the optimum.

*Keywords:* Control Function, Bin-Packing Problem, Multiple Choice, Conflicts, NP-hard Problem.

## 1. INTRODUCTION AND RELATED WORKS

The operational control architecture of a production system is composed of a set of physical devices (e.g. Programmable Logic Controllers, industrial computers) that performs the necessary tasks (or functions) to control the plant.

In order to build this architecture, control functions must first be assigned to the physical controllers while satisfying several architectural and safety constraints. Afterwards, the resulting architecture must go through a validation process in which performances, such as response time (*see.* Meunier et al. (2007)) and availability are checked. In fact, the behavior of the control system is highly impacted by assignment decisions.

According to risks incurred in case of failure, functions are classified in different safety-level classes. Controllers of different criticality factor are then provided. The higher the safety-level of the function the greater the criticality factor of the controller hosting it must be. Then, functions of each class can only be assigned to some safety compatible controllers.

When assigning functions, we should also consider that controllers have limited capacity in terms of resources such as cpu, memory, and inputs/outputs... Besides, some functions are associated with separation constraints. A separation constraint between a pair of funtions indicates that they must not be assigned to the same controller. This is usually due to safety reasons and fault-tolerance considerations. For instance, in order to increase the availability of some critical processes, their associated functions are made redundant. Therefore, these redundant functions must not be assigned to the same controller as otherwise, if the controller hosting them fails, the task they perform will no longer be available which may have disastrous consequences.

The high complexity of operational architecture design leads designers to have recourse to an exploration process. The architecture designer iterates over several variations by changing functions assignment decisions until a satisfactory solution that meets all requirements is found. Due to their potentially excessive run-times, exact techniques are discarded in such an exploration process. Fast and scalable heuristics can be of great help in this decision process as they provide a sufficiently fast evaluation tools to allow going through several designs.

In this paper, we seek to minimize the global cost of the architecture such that previously mentioned constraints are satisfied. Our assignment problem generalizes more than a variant of the Bin-Packing problem (in short BP). This combinatorial problem consists of packing items of different sizes into the minimum possible number of identical bins with a given capacity. In its multi-dimensional variant known as Vector Bin-Packing (in short VBP), items and bins are multi-dimensional. Dimensions may correspond in our case to cpu and memory consumption, number of inputs/outputs.

When considering separation constraints, the problem becomes a VBP with conflicts. This problem combines the VBP and the Vertex Coloring problem.We are then given a conflict graph in which each vertex corresponds to a conflicting item. A pair of conflicting items that cannot be assigned to the same bin are represented by an edge in the conflict graph between the two corresponding vertices.

The conflict-free version of our problem, *i.e.* without considering separation constraints, can be seen as an instance of a Multiple-Choice VBP (MVBP) (*see.* Patt-Shamir

⋆ This work was supported by Cluster CONNEXION program (COntrôle commande Nucléaire Numérique pour l'EXport et la rénovatION).

10.1016/j.ifacol.2015.06.132

and Rawitz (2012)). In this variant of the Bin-Packing problem, different types of bins with differents costs and characteristics are provided, and items have different incarnations depending on these types. Due to safety constraints, we address a special case of the MVBP in which a function would have no incarnation for incompatible types of controllers.

Both the Bin-Packing and the Vertex Coloring problems are strongly NP-hard (*see.* Garey and Johnson (1979)). Thus, as a generalization of these two problems, the control functions assignement problem we address in this paper is also strongly NP-hard. Real instances we are aiming at may have around ten thousand functions and need hundreds of controllers with a number of functions per controller that may exceed 40. Such instances are beyond the solving capacity of exact algorithms for which we can already find instances of two hundred items that are still open, *i.e.* for which no optimal solution is known (*see.* Brandão and Pedroso (2013a)). On the other hand, for such instances with high number of items per bin, heuristics provide good solutions because the waste in bins tends to be smaller.

***Contribution*** The principal contribution of this paper is a safety-level aware vector bin-packing approach that mimimizes the cost of an operational control architecture by assigning functions to safety compatible controllers while satisfying resource and separation constraints. This is accomplished as follows. First, a main strategy, that indicates an order into which functions must be treated, is detailed. This order is specified according to functions safety-level classes and to the costs of their compatible controllers.

At each step of the order, a bin-packing algorithm is called to assign functions. We propose a heuristic that is an improved version of the First-Fit Decreasing algorithm (in short FFD). The FFD, that is a greedy algorithm, has been proven to be a powerful fast heuristic to solve the one-dimentional bin-packing problem. The version proposed was developed specifically to manage the multi-dimensional and the conflicts aspects in accordance with our main strategy. The FFD operates by first sorting functions which may greatly impact the results. We propose three different sortings that alternate between giving priority to resource constraints satisfaction and conflicts resolving.

Finally, we run several experiments in order to prove the accuracy of our ordering strategy and our adapted version of the FFD.

***Related works*** In Lemattre et al. (2011), authors propose a technique based on the verification of a reachability property on a network of communicating automata to model the assignment problem. However, this technique is meant more to find a feasible solution that satisfies constraints than to obtain an optimized one. Besides, controllers are considered of the same cost no matter their integrity level.

As mentioned before, the problem we address in this paper is one of several variants of bin-packing problem. Despite the extremely rich literature in this field, we were not able to find works that consider jointly both the multiple-choice of bins and the conflicting items when solving a vector bin-packing problem. In Brandão and Pedroso (2013a) and Brandão and Pedroso (2013b), authors propose an exact technique based on an Arc-Flow formulation that represents all the feasible packing solutions of a problem in a compact graph. Different arc-flow models are then built using this formulation for the VBP, the one-dimensional BP with conflicts and the MVBP. Though it seems possible to combine these models in order to build an arc-flow model for our specific problem and for general MVBP with conflicts, we will be quickly limited by the instances that can be treated using such formulation. In fact, in addition to the size of our real instances that may exclude the use of any exact methods, this arc-flow technique is very sensitive to the number of items per bin (10 items per object is a maximum limit).

In Patt-Shamir and Rawitz (2012), a polynomial-time approximation algorithm for the MVBP is proposed whose approximation ratio depends on the number of dimensions. On the other hand, VBP with conflicts are hard to approximate for general graphs of conflicts. Then, in most of the existing works this variant is considered on restricted graphs of conflicts. Authors in Epstein et al. (2008) propose approximations for the 2-dimensional bin-packing with perfect and bi-partite conflicts graphs. In our instances specification, separation constraints do not follow a specified pattern; their conflict graph should then be considered as general.

The paper is organized as follows: our notations and problem constraints are presented in the next Section 2. A Mixed Integer Linear Program (in short MILP) that models our problem is also proposed. Then, our safety-level FFD based approach is detailed in Section 3. Section 4 is dedicated to experiment results. We conclude in Section 5.

## 2. PROBLEM FORMULATION

For the sake of comprehension, and without impacting the generality, we will limit our formulation to the 2-dimensional VBP case.

Let us consider a set of functions $\mathbb{F}$ and a set of controllers $\mathbb{C}$. Each function $f_i \in \mathbb{F}$ is characterized by its

- $inout_i$ : the number of its inputs and outputs,
- $cpu_i$ : its cpu consumption, and
- $sl_i$ : its safety level with $sl_i \in \{A, B, C, NC\}$.

A function $f_i$ with $sl_i = A$ (*resp.* $B$,$C$,$NC$) is said to be of class $A$ (*resp.* $B$,$C$,$NC$).

On the other hand, each controller $C_j \in \mathbb{C}$ is characterized by its

- $INOUT_j$ : its capacity in terms of inputs/outputs.
- $CPU_j$ : its processing capacity,
- $CF_j$ : its criticality factor with $CF_j \in \{0, 1, 2\}$,
- $COST_j$ : its cost.

A controller $C_j$ with $CF_j = 0$ (*resp.* 1,2) is said to be of category 0 (*resp.* 1,2). Controllers with smaller criticality factor identifier are safer and more reliable. The cost of a controller is proportional to its characteristics. Since in our case, all controllers are considered to have the same processing capacity $CPU$, *i.e.*

$$\forall C_j \in \mathbb{C}, CPU_j = CPU,$$

and the same capacity of inputs/outputs $INOUT$, *i.e.*

$$\forall C_j \in \mathbb{C}, INOUT_j = INOUT,$$

the cost of a controller depends only on its criticality factor. The safer it is, the more expensive it is.

For safety reasons, functions must be assigned to controllers according to their safety levels. A controller of

- category 0 can only host A-class and B-class functions,
- category 1 can only host B-class and C-class functions,
- category 2 can only host C-class and NC-class functions.

A loose upper bound on the number of controllers of each category can then be derived. Let us define by $\mathbb{F}_A$ (*resp.* $\mathbb{F}_B$, $\mathbb{F}_C$,$\mathbb{F}_{NC}$) the set of A-class (*resp.* $B$,$C$,$NC$) functions such that $\mathbb{F} = \mathbb{F}_A \cup \mathbb{F}_B \cup \mathbb{F}_C \cup \mathbb{F}_{NC}$. Then, as we seek to minimize the cost of controllers, we need at most $|\mathbb{F}_A|$ (*resp.* $|\mathbb{F}_B|$, $|\mathbb{F}_C|+|\mathbb{F}_{NC}|$) controllers of category 0 (*resp.* 1, 2). In fact, if we consider the solution that consists in each controller hosting only one function, functions of $\mathbb{F}_A$ would be packed in controllers of category 0, those of $\mathbb{F}_B$ would be packed into controllers of category 1, and functions of $\mathbb{F}_C$ and $\mathbb{F}_{NC}$ would be packed into controllers of category 2.

Using this loose upper bound and following safety constraints, we can construct $\mathbb{S}$, the set of couples $(f_i, C_j) \in \mathbb{F} \times \mathbb{C}$ such that the class of $f_i$ is incompatible with the category of $C_j$.

Besides, as an input of our problem we are given a list $\mathbb{L}$ of pairs of functions $(f_i, f_{i'})$ such that $f_i$ is in conflict with $f_{i'}$.

Let us consider also the following Boolean variables

- $x_{ij}$ that should take the value 1 iff the function $f_i$ is assigned to controller $C_j$, and
- $U_j$ that should take the value 1 iff the controller $C_j$ is used, *i.e.* hosting at least one function.

Then, our assignment problem can be modelled by the following Mixed Integer Linear Program (in short MILP):

$$minimize \sum_{C_j \in \mathbb{C}} COST_j \cdot U_j \quad subject\ to$$

$$\begin{cases}
\forall f_i \in \mathbb{F}, & \sum_{C_j \in \mathbb{C}} x_{ij} & = 1 & (1) \\
\forall C_j \in \mathbb{C}, & \sum_{f_i \in \mathbb{F}} inout_i \cdot x_{ij} & \leq INOUT & (2) \\
\forall C_j \in \mathbb{C}, & \sum_{f_i \in \mathbb{F}} cpu_i \cdot x_{ij} & \leq CPU & (3) \\
\forall (f_i, C_j) \in \mathbb{F} \times \mathbb{C}, & x_{ij} & \leq U_j & (4) \\
\forall C_j \in \mathbb{C}, \forall (f_i, f_{i'}) \in \mathbb{L}, & x_{ij} + x_{i'j} & \leq 1 & (5) \\
\forall (f_i, C_j) \in \mathbb{S}, & x_{ij} & = 0 & (6) \\
\forall (f_i, C_j) \in \mathbb{F} \times \mathbb{C}, & x_{ij} & \in \{0,1\} & (7) \\
\forall C_j \in \mathbb{C}, & U_j & \in \{0,1\} & (8)
\end{cases}$$

This MILP minimizes the global cost of used controllers. Constraint (1) models the fact that a function must be assigned to one and only one controller. Constraints (2) and (3) satisfy the limited capacity of controllers in terms of CPU and INOUT resources. Constraint (4) indicates whether a controller is used or not. Constraint (5) ensures the separation of conflicting functions, *i.e.* their assignment to different controllers. Constraint (6) ensures the compatibility of the classes of functions with the categories of controllers to which they are assigned.

For solvers that may be more efficient on MILPs without Boolean constraints, the domain of variables $x_{ij}$ in (7) and $U_j$ in (8) can be relaxed to $\mathbb{N}$. In fact, the minimization objective and the set of contraints (1) and (4) bind their values to 0 or 1.

## 3. OUR APPROACH

At first, we present our assignment strategy to manage safety compatibilities between functions and controllers in order to reduce the cost of the global architecture. Then, we detail our packing algorithm.

### *3.1 Safety-Level Management*

To manage safety constraints, we adopted the following strategy that specifies an order in which functions are assigned to compatible controllers.

**Algorithm 1** Main Strategy Algorithm

```
1: Pack functions of class A into controllers of category 0
2: Pack functions of class NC into controllers of category 2
3: Pack functions of class B into remaining space of already used controllers of category 0, otherwise use new controllers of category 1
4: Pack functions of class C into remaining space of already used controllers of category 1 or 2, otherwise use new controllers of category 2
```

This packing order is driven by the cost minimization objective. In fact, since functions of class A can only be packed into controllers of category 0, we proceed by packing them first. This has for result to set definitely the number of controllers of category 0 to its minimum possible as, first, they are the most expensive controllers and, second, all the remaining functions can be assigned to cheaper ones. Then, we pack functions of class NC to controllers of category 2. The obtained number of controllers is a lower bound on the number of controllers of this category. Now that these two classes are packed, we resume by packing functions of class B. We use first the remaining resources in already created controllers of category 0 at step 1, and if it is not sufficient, we create new controllers of category 1 as they are cheaper. Finally, we pack functions of class C into the available space provided by already created controllers at step 2 and 3 and by creating new controllers of category 2 if necessary.

Note that steps (1) and (2) involve different classes of functions and different categories of controllers. Therefore, these two first steps can be interchanged and even parallelized. As for the last two steps, it is important to proceed as detailed in the specified order to follow the objective of minimizing the cost of controllers. Indeed, if we pack functions of class C first, it would be difficult to determine whether new created controllers should be of category 2 or 1 in anticipation of the possible assignment of functions of class B into these controllers.

We should mention that at this level, the proposed order of packing does not specify the underlying algorithm used to assign functions to controllers, that is detailed in the next subsection.

### *3.2 First-Fit Decreasing Assignment Approach*

The first-fit packing algorithm (*see.* Algorithm 2) consists of going through the list of functions (line 1) and for each function to assign it to the first encountered controller in which it can fit in (lines 2 to 7). If there is no already created controller in which the function can fit in, then we create a new controller and we assign the function to it (lines 8 to 11).

**Algorithm 2** First-Fit Algorithm

```
1: for All functions $f_i/i \in \{1, 2, ..., n\}$ do
2:    for All controllers $C_j/j \in \{1, 2, ...\}$ do
3:       if $f_i$ fits in $C_j$ then
4:          Assign $f_i$ to $C_j$
5:          Break the inner loop to assign the next function
6:       end if
7:    end for
8:    if $f_i$ does not fit in any available controller then
9:       Call Create New Controller ($f_i$)
10:      Assign $f_i$ to the new controller
11:   end if
12: end for
```

Note that each time a function is assigned to a controller, we need to update the remaining resources of this controller (lines 4 and 10).

In the decreasing variant of this algorithm (*see.* Algorithm 3), the functions are first sorted in a non-increasing order following their characteristics (line 1) before they are packed (line 2), which corresponds to the simple policy that consists of packing big items first in the original algorithm.

**Algorithm 3** First-Fit Decreasing Algorithm (FFD)

```
1: Sort functions in a non-increasing order
2: Call First-Fit Algorithm
```

Now, in our case, this leaves several questions to be answered. We need to define the fitting in conditions, which are detailed in the next subsection. Besides, we need to specify the policy of controllers creation (See Subsection 3.4). Finally, since we are in a conflicting and multidimensional context, we have to define how functions are sorted (See Subsection 3.5).

### *3.3 Fitting in conditions*

A function $f_i$ fits in a controller $C_j$ if and only if the following conditions are satisfied

- the class $sl_i$ is compatible with the category $CL_j$.
- the controller $C_j$ has enough remaining cpu and inouts resources to perform $f_i$.
- $f_i$ is not in conflict with any already assigned function $f_{i'}$ to the controller $C_j$.

Several ways exist to check quickly the last condition. One way is to keep for each function a list of controllers to which a conflicting function has already been assigned. These lists are updated as we advance in the assignment process. Then, if a controller satisfies the two first fitting in conditions but belong to this list, it is excluded.

### *3.4 Controllers Creation Policy*

The policy to set the category of a new created controller is defined in Algorithm 4.

**Algorithm 4** Create New Controller ($f_i$)

```
1: if $f_i$ is of Class A then
2:    Create a new controller of category 0
3: else if $f_i$ is of Class B then
4:    Create a new controller of category 1
5: else if $f_i$ is of Class C then
6:    Create a new controller of category 2
7: else if $f_i$ is of Class NC then
8:    Create a new controller of category 2
9: end if
```

The consistency of this policy with our cost minimization objective is ensured by the order in which functions of different classes are packed as presented in Subsection 3.1. If another order is specified this policy is no more relevant.

### *3.5 Sorting Criteria*

In the original version of the bin-packing problem, items are sorted based on their unique dimension, their size. Given the additional set of constraints that we treat, several criteria can be adopted to sort functions. The generic idea in that case is to aggregate subsets of characteristics of an object to get only one characteristic that represents them. In one sense, this will have for effect to reduce the multidimensional problem into a mono-dimensional one for which we can apply the original version of the algorithm.

We adopted two agregators whereby functions are sorted according to:

- *maxResources* : the relative maximum of their number of inouts and their cpu consumption $\max(\frac{cpu_i}{CPU}, \frac{inout_i}{INOUT})$.
- *meanResources* : the relative mean of their number of inouts and their cpu consumption $\frac{cpu_i}{CPU} + \frac{inout_i}{INOUT}$.

It is necessary to use relative values instead of absolute ones as in the general case dimensions are not of the same nature. Note also that these aggregators present the advantage of being easily extensible to multiple dimensions. Nonetheless, they present in our case the disadvantage of giving priority to resolving resources constraints and ignore the conflicting aspect of our problem, which may lead to an overestimation of necessary number of controllers. This can be highlighted by the following example.

Let us suppose that at the end of an assignment process two last functions $f_i$, $f_{i'}$ remain such that $f_i > f_{i'}$ according to both previously defined sorting criteria. We suppose also that there exists only two controllers $C_j$ and $C_{j'}$ with $j > j'$ that offer enough resources to host $f_i$ and $f_{i'}$ but cannot host both of them at the same time.

Then, according to Algorithm 3, $f_i$ is assigned first to $C_j$. Now, if we suppose that a conflicting function with $f_{i'}$ is already assigned to $C_{j'}$, then we will need to create a new controller to assign $f_{i'}$ while a better solution would have been to assign $f_{i'}$ to $C_j$ and $f_i$ to $C_{j'}$. In order to overcome such a case we considered another aggregator that sorts functions according to

- *nbrConflicts* : their number of conflicts $+(\frac{cpu_i}{CPU} + \frac{inout_i}{INOUT})/2$.

Following this criterion, the more a function has conflicts with other functions the more it is preferable to assign it first. Functions with no conflicts will be assigned at last.

The impact of these different criteria will be more discussed through experiments in the next section.

## 4. EXPERIMENT RESULTS

In this section, we study the impact of our decisions on obtained results. Usually, when solving NP-hard combinatorial problems, we are very concerned with algorithms run-times. Nonetheless, since run-times and scalability are not a problem for a greedy algorithm such as the FFD, we are more concerned about the quality of solutions.

Apart from the overestimation that may result from the underlying packing algorithm used (FFD in our case), our strategy to manage safety-level constraints described in Subsection 3.1 may result in an overstimation of the required number of controllers of category 1 and 2. Category 0 is discarded as explained before.

In order to evaluate the accuracy of our results we decided to compare them with optimal solutions. Since exact techniques are unable to solve the problem at full scale (~10000 functions and ~600 controllers), the comparison was made on the basis of samples of reduced size.

One thousand samples of 200 functions each were generated. Those samples were obtained according to characteristics provided in a specification of anonymized data that specifies by mean of intervals and their proportions: cpu loads, number of inouts, number of functions by class, number of conflicting functions and for each conflicting function its number of conflicts. However, due to the reduced size of samples, the number of conflicting functions as well as their number of conflicts were reduced with regard to the factor 10000 to 200. Otherwise, all the functions would have been in conflict with the others and assignment solutions would be obvious. We expect the generator to deliver representative samples for our analysis so as to allow us to derive estimations on real instances.

We considered that a controller of a certain category is twice more expensive than a controller of the below category. As for optimal solutions, they were obtained by solving the MILP proposed in Section 2 using the *CPLEX* solver IBM-ILOG (2014).

### *4.1 First-Fit Decreasing accuracy*

First, we decided to check our choice of the FFD heuristic as an underlying packing technique to our global safety-level aware strategy presented in Subsection 3.1. In order to isolate the effect of our global strategy to integrate the safety aspect, the only solution was to check the FFD on the number of controllers of category 0 obtained. In fact, the computation of the number of controllers for the other categories involves at least three steps of the main algorithm 1.

In 97.3% of samples, the FFD algorithm using the three criteria described in Subsection 3.5 allowed us to compute the exact number of controllers of category 0. This performance is closely related to the large number of functions per controller in our instances which allows reducing the waste in controllers.

The criterion that prioritizes conflicts resolution allowed by itself obtaining 96.8% while the other two criteria do not exceed 55% each. However, at this stage, it was to early to decide definitely whether to prune or not those criteria that give priority to resources over conflicts resolution. Therefore, we decided to deep our analysis on the impact of sorting criteria and their interaction with our global strategy.

### *4.2 Impact of Sorting Criterion*

In this subsection, the impact of sorting criteria on the obtained architecture cost is evaluated. In a first experiment, the same criterion was kept through all the steps of Algorithm 1.

We adopted three indicators for the comparison of these criteria

- NbrOptimal : The number of optimal solutions obtained.
- AvgRelInc : The average relative increase in cost.
- MaxRelInc : The maximum relative increase in cost.

The second indicator is an average value over the thousand samples whereas the third indicator highlights the worst case performance.

Table 1. Comparison of sorting criteria (values in %)

| Criterion | NbrOptimal | AvgRelInc | MaxRelInc |
|---|---|---|---|
| *maxResources* | 18.8 | 5.87 | 26.66 |
| *meanResources* | 20.7 | 5.53 | 24.14 |
| *nbrConflicts* | 56.2 | 1.92 | 14.28 |
| Best solution | 59.8 | 1.63 | 13.79 |

Contrary to our expectations for a good lead of meanResources criterion over maxResources criterion, results reported in Table 1 show that sorting functions according to their relative mean of resources performs slightly better than the relative maximum of resources. However, all the indicators confirm that sorting according to the number of conflicts far outperforms maxResources and meanResources sortings. Nonetheless, these last should not be discarded as it is shown by the Best solution row that keeps for each sample the best solution obtained amongst the three. In fact, some additional exact solutions (less than 4%) are provided exclusively by those criteria. In other terms, for some instances, it is better to give priority to resources over conflicts when sorting.

Seeking to improve these results, we allowed combining sorting criteria in a second experiment, *i.e.* to change the sorting criterion between the subsequent steps of the main algorithm 1. This would result in $3^4 = 81$ possible combination when the three criteria are used. Nevertheless, due to the reduced running time of the FFD heuristic (less than 10 milliseconds per combination), we could afford it. Results are reported in Table 2.

Table 2. Combining criteria (values in %)

| Combined criteria | NbrOptimal | AvgRelInc | MaxRelInc |
|---|---|---|---|
| maxRes. & meanRes. | 31.2 | 4.28 | 21.43 |
| maxRes. & nbrConfl. | 65.8 | 1.38 | 13.33 |
| meanRes. & nbrConfl. | 65.9 | 1.34 | 13.33 |
| All three criteria | 67.9 | 1.24 | 13.33 |

All indicators have improved which proves that within the same sample, different classes of functions may need different sortings. For some samples, different sortings result in the same number of controllers until a certain step of Algorithm 1 but may result in different numbers of controllers in the subsequent steps.

Combining maxResources with meanResources enhances their results, but they stay far behind the nbrConflicts criterion. However, the addition of any of these resource based criteria to the nbrConflicts criterion allows obtaining 66% of exact solutions (rows 2 and 3). That is an improvement of 17% in comparison with nbrConflicts row of Table 1. The combination of all three criteria allows us to get 2% more of exact solutions.

The AvgRelInc indicator does know an improvement of around 24% (*see.* Best solution row of Table 1). However, considering that it was already excellent (below 2% of relative increase), this improvement is not that noticeable. As for the MaxRelInc indicator, it has barely improved and got stuck at 13.33%. The reason will be more discussed in next subsection.

At this stage, we can affirm that the combination of at least one resource based sorting (preferably meanResources) with the conflicting based sorting is mandatory for more accurate results.

### 4.3 What would be expected for real scaled instances?

At full scale samples, we expect all the indicators to get better and our safety-level aware technique to reduce the gap to the optimum in most cases. Particularly, the maximum relative increase is more likely to decrease.

In order to derive such conclusion, we analyzed the overestimation that results from our heuristic in terms of number of controllers instead of global cost. We realized that the registered maximum relative increase of 13.33% is principally due to the reduced number of controllers needed for our small samples. In most of cases, this increase corresponded to only one additional controller of category 0. Then, in such a case, it is related to the behaviour of the FFD algorithm. When the number of functions increases, the needed number of controllers increases. However, the gap in number of controllers between the heuristic and the optimum value is more likely to increase at lower pace. Hence, the maximum relative increase decreases.

This can also be explained by examining the performance of the original FFD heuristic. In the one-dimensional bin-packing problem, the FFD has been shown to use at most $(\frac{11}{9} \cdot Optimal + 1)$ bins, which in the worst case corresponds to a maximum relative increase of $\frac{(\frac{11}{9} \cdot Optimal+1)-Optimal}{Optimal} = \frac{2}{9} + \frac{1}{Optimal}$. Then, the more the instance needs bins, the more the FFD gets better. We expect the same behaviour in our case too.

Just to give an order of values, the average optimal value for the reduced size samples is around 5 controllers per category. Then, if we consider the same formula as before, the second part of the maximum relative increase, *i.e.* $\frac{1}{Optimal}$, accounts for around 48% of the overestimation. When the optimal number of controllers attains hundreds this amount declines significantly improving by the way the quality of results.

## 5. CONCLUSION

In this paper we adopted a simple but effective strategy to assign functions to safety compatible controllers. Combined with an adapted version of the FFD, it allowed us to achieve good results while maintaining the run-time as low as possible. Nonetheless, at this stage, we have not integrated all the aspects of the operational control architecture design. The work presented in this paper constitutes just one step towards the automation of this complex process.

## REFERENCES

Brandão, F. and Pedroso, J.P. (2013a). Bin Packing and Related Problems: General Arc-flow Formulation with Graph Compression. Technical Report DCC-2013-08, Faculdade de Ciências da Universidade do Porto, Portugal.

Brandão, F. and Pedroso, J.P. (2013b). Multiple-choice Vector Bin Packing: Arc-flow Formulation with Graph Compression. Technical Report DCC-2013-13, Faculdade de Ciências da Universidade do Porto, Portugal.

Epstein, L., Levin, A., and van Stee, R. (2008). Two-dimensional packing with conflicts. *Acta Inf.*, 45(3), 155–175.

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

IBM-ILOG (2014). CPLEX Optimizer.

Lemattre, T., Denis, B., Faure, J.M., Pétin, J.F., and Salaün, P. (2011). Designing operational control architectures of critical systems by reachability analysis. In *CASE*, 12–18. IEEE.

Meunier, P., Denis, B., and Lesage, J.J. (2007). Temporal performance evaluation of control architecture in automation systems. *Proceedings of 6th EUROSIM Congress on Modelling and Simulation.*

Patt-Shamir, B. and Rawitz, D. (2012). Vector bin packing with multiple-choice. *Discrete Appl. Math.*, 160(10-11).