Improving the Diagnosability of Business Process Management Systems Using Test Points

D. Borrego, M. T. Gómez-López, R. M. Gasca, and R. Ceballos

Dept. de Lenguajes y Sistemas Informáticos, Universidad de Sevilla, Sevilla, Spain,
`{dianabn,maytegomez,gasca,ceball}@us.es`

Abstract. The management and automation of business processes have become an essential task within IT organizations. Diagnosis enables fault isolation in a business process. The diagnosis process uses a set of test points (observations) and a model in order to explain a wrong behavior. In this work, an algorithms to allocate test points is presented. The key idea is to improve the diagnosability, improving the computational complexity for isolating faults in the activities of business processes. The methodology is based on constraint programming.

Key words: Process tracing and monitoring, constraint programming, fault diagnosis, fault isolation, test points

1 Introduction

A business process (BP) is composed of a set of activities which are logically related to achieve a defined goal. Currently BPs can be composed of different subprocesses and a large number of activities that interact by means of a choreography with the same process or another.

BP management includes concepts, methods, and techniques to support the design, administration, configuration, enactment, and analysis of BPs [1]. A BP instance represents a concrete case in the operational process for a model. Each BP model acts as a blueprint for a set of BP instances. When a BP is monitored, some errors can be detected. The diagnosis process is used to detect which task or tasks are responsible of the incorrect behavior of the process for any instance.

Fault diagnosis permits to determine why a BP correctly designed does not work as it is expected. Its aim is to detect and to identify the reason of an unexpected behavior, or in other words, to identify the parts which fail in a BP. The computation is based on observations, which provide information about the current behavior. These observations come from the public information existing in the BP to diagnose, measured either by the direct observation of the user or by means of test points allocated in certain places of the BP. There exists a paper related to the conflicts detection (CDM - Conflict Detecting Mechanism), where the specification of the services described in XML is used to design metaprocesses [2] for the detection of inconsistences between the activities. Another work [3] performs the diagnosis of BPs according to the topology of the activities and their relations with the public information monitored.

The appropriate allocation of test points makes possible the isolation of faults between the activities of a BP. Test points are control points where it is possible to know data that are available at a moment of the execution. The aim of this work is to apply techniques of allocation of test points, in order to improve the computational complexity of isolating faults, and make the BP diagnosable.

In order to allocate the test points, constraint programming can be used since a BP can be modelled as a constraint satisfaction problem (CSP), where its topology is transformed into a graph, in such a way that the nodes and edges can be modelled using constraints.

CSPs are problems where one must find states or objects that satisfy a number of constraints or criteria. They count on a large expressive power, existing many programs able to perform efficient searches of solutions. A large number of problems in artificial intelligence and other areas of computer science can be viewed as special cases of constraint satisfaction problems. Some examples are belief maintenance, scheduling, temporal reasoning, graph problems, etc.

A number of different approaches have been developed for solving CSPs. Some of them use constraint propagation to simplify the original problem. Others use backtracking to directly search for solutions. Some are a combination of these two techniques. In general, a CSP is formed by a set of variables, a finite and discrete domain for each variable, and a set of constraints. Each constraint is defined over some subset of the original set of variables and limits the combinations of values that the variables in this subset can take. The goal is to find one assignment to the variables such that all the constraints are satisfied [4] [5].

The paper is structured as follows: Section 2 explains how it is possible to solve the problem of the allocation of test points in BPs, including three different objectives to achieve. Section 3 shows some experimental results. And finally, conclusions and future work are presented.

2 Allocation of Test Points in a Business Process

The aim of this paper is to apply techniques for the allocation of test points in BPs. This paper is centered in the BPs that fail in any instance, or its behavior does not correspond with the expected.

Definition 1. *Being the private information the non-observable information exchanged between activities, a set of activities T is a Cluster, (i) if it does not exist common private information of any activity of the cluster with any activity outside the cluster, and (ii) if for all $Q \subset T$ then Q is not a cluster of activities.*

The connections between different clusters are through public information. Therefore, each cluster is completely monitored, since all the connections that the cluster has with the rest of the activities in the BP are public. This monitoring makes possible that the detection of faults inside a cluster can be done independently of the rest of the clusters.

The test points make possible the separation of the activities of the BP into different clusters, reducing the computational complexity in the diagnosis process and making easier the detection of the incorrect activity. This is possible since the diagnosis of the whole BP can be performed based on the diagnosis of each cluster separately. As it is explained in [9], being A the set of activities of a BP, with n activities, let us divide that set into two clusters C_1 and C_2, with n - m and m activities respectively, such as $C_1 \cup C_2 = A$. When it comes to detect the conflicts, the computational complexity in both clusters separately is lower than in the whole BP A, since the number of possible diagnoses of C_1 and C_2 is $(2^{n-m}) + (2^m)$ - 2 $\leq 2^{n-m} \cdot 2^m$ - 2, which is less than 2^n - 1, which is the computational complexity for the whole set of activities A.

Regarding the diagnosability, it is improved through the allocation of test points, according to the next definition.

Definition 2. *The Diagnosability level is the quotient of the number of the (classes of) faults which can be distinguished each other, and the number of all the possible faults. Being nAct the number of activities in a BP, the size of the possible faults is initially* 2^{nAct} *- 1.*

This is, without allocating any test points, all the information within the BP is private and only one fault can be distinguished: the *nAct* activities fail or not. When some test points are allocated and m clusters are obtained, the number of faults that can be distinguished according to Definition 2 are 2^m - 1. Therefore, the number of clusters obtained improves the diagnosability exponentially.

In order to illustrate the allocation of test points, Fig. 1 presents an example of BP that has been used to obtain clusters of activities.

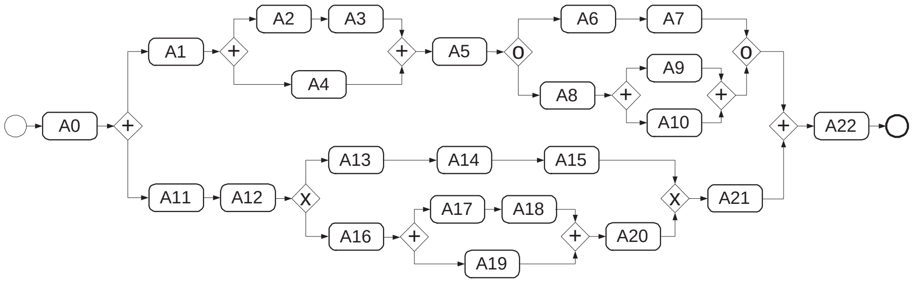

Fig. 1. Example of business process.

When it comes to allocate test points in a BP, the algorithm proposed can be configured to achieve different possible objectives. In the case of this paper, it can be configured to solve the problem with three different objectives, depending on the user necessity or the problem requirements:

- Given a number of test points to allocate, maximize the number of clusters.
- Given a number of clusters to obtain, minimize the test points to allocate.
- Given a maximum number of activities per cluster, minimize the number of test points to allocate.

Since the BPs are going to be modelled as CSPs, how the graph is transformed and the implemented algorithm are detailed in the following.

2.1 Improving the Diagnosability Using Constraint Programming

Constraint programming is based on the resolution of CSPs. A CSP consists of $\langle X, D, C\rangle$ where X is a set of n variables $x_1, x_2, ..., x_n$ whose values are taken from finite, discrete domains $D_1, D_2, ..., D_n$ respectively, and C is a set of constraints on their values. The constraint c_k $(x_{k1}, \ldots, x_{kn})$ is a predicate that is defined on the Cartesian product $D_{k1} \ldots D_{kj}$. This predicate is true iff the value assignment of these variables satisfies the constraint c_k.

Initially, the BP is considered as a directed graph, where the activities are the nodes and the connections between them are the edges. Those nodes and edges give rise to variables in a CSP:

- *nAct*: this constant-variable represents the number of activities in the BP.
- *nCon*: this constant-variable holds the number of edges between activities.
- *clusterOfAct$_i$*: this set of $nAct$ variables represents the cluster where each activity i is contained.
- *testPoint$_j$*: this set of $nCon$ variables holds the possible new test points in the BP. There are as many variables as connections in the BP, and the possible values are boolean: $true$ implies that there must be a test point in a determined connection, and $false$ means the opposite.
- *nTestPoints*: this variable holds the number of allocated test points.
- *nClusters*: this variable holds the number of obtained clusters.

Table 1 shows these variables with their corresponding domains (D).

Table 1. Variables of the CSP.

$clusterOfAct_i$, $D: \{0, \ldots, nAct-1\}$
$testPoint_j$, $D: \{true, false\}$
$nTestPoints$, $D: \{0, \ldots, nCon\}$
$nClusters$, $D: \{1, \ldots, nAct\}$

Once the BP is transformed into a graph, each edge gives rise to a constraint within the CSP, which establishes that if there is not a test point in a link between two activities, both activities are necessarily in the same cluster.

As an example, let us suppose that the connection between the activities $A14$ and $A15$ in Fig. 1 is considered the n-th possible test point of the set $testPoint_j$. Therefore, the constraint added to the CSP would be:

$\texttt{if(}testPoint_n = \texttt{false)} \Rightarrow clusterOfAct_{A14} = clusterOfAct_{A15}$

That constraint means that if the connection between $A14$ and $A15$ does not count on a test point, $A14$ and $A15$ are necessarily in the same cluster. It is not possible to assert the opposite statement, since the existing of a test point in the connection between $A14$ and $A15$ cannot imply that $A14$ and $A15$ are in different clusters because it is possible that these two activities are connected through another path in the graph. Those added constraints model the structure of the BP as a CSP. They are used in the three different objectives that the solution proposed in this paper achieves. But each objective needs different new constraints and goals to be completely modelled. Therefore, the three objectives and their corresponding configuration are detailed in the following subsections.

2.2 Objective 1: to Maximize the Number of Clusters with a Fixed Number of Test Points to Allocate

In order to achieve this objective, new information must be added to the CSP:

- The number of test points must be limited to a value t. This is, the number of *true* values in the set $testPoint_j$ must be equal to t.
- The goal is included in the CSP: taking all the combination of pairs of values in the set $clusterOfAct_i$, the number of different pairs of values must be maximized. What is being obtained is that the maximum number of activities are placed in different clusters, maximizing the number of clusters obtained.

Being $pairs_{i,j}$ a variable that indicates if each pair of activities i and j are in a different cluster (value 1) or in the same one (value 0), Table 2 shows the new information added to the CSP for Objective 1.

Table 2. New information for the CSP of objective 1.

Variable and domain: $pairs_{i,j}$, $D : \{0, 1\}$
Constraints: $nTestPoints = t$, $t \in \{0, \ldots, nCon\}$
$\forall i, j \in \{1, \ldots, nAct\}$ $(clusterOfAct_i \neq clusterOfAct_j) \leftrightarrow pairs_{i,j} = 1$
Goal: $maximize(\sum_{i=1}^{nAct} \sum_{j=1}^{nAct} pairs_{i,j})$

The computational complexity of the modelled CSP is exponential for the number of connections. When the BP has a large number of activities, the time needed to solve the CSP makes this solution inappropriate. In order to sort out this problem, the greedy algorithm presented in [9] is used. In short, that algorithm assigns a weight to each edge in the graph, and applies the Floyd's algorithm to find the minimal path between each pair of nodes. That minimal paths decide through a voting mechanism which are the bottlenecks of the BP. This is, which are the most important connections to allocate test points.

The obtained result from the greedy algorithm is a set with the different connections and the votes received for each one of them. This set is used to select the collection of variables in $testPoint_j$ that correspond to the connections where will be better to allocate test points. Those variables will be the only ones taken into account in the solution of the CSP, improving the computational complexity. If n variables are chosen, the number of possible solutions of the CSP is lower than 2^n. On the other hand, the optimal solution is not guaranteed.

2.3 Objective 2: to Allocate the Minimum Number of Test Points in Order to Obtain a Fixed Number of Clusters

In order to model this solution, it is necessary to include some constraints on the CSP. Those constraints establish the number of clusters in a value $numClusters$ determined by the user or the specification of the problem. For this solution, the configured goal is to minimize the number of values equal to *true* in the set $testPoint_j$. This CSP does not present computational problems, so that it is not necessary to include any previous method to improve the execution time.

Table 3 includes the new constraint and goal.

Table 3. New information for the CSP of objective 2.

Variable: $numClusters$, $D : \{1, \ldots, nAct\}$
Constraint: $nClusters = numClusters$, $numClusters \in \{1, \ldots, nAct\}$
Goal: $minimize(nTestPoints)$

2.4 Objective 3: to Minimize the Number of Test Points to Allocate in Order to Obtain Clusters with a Maximum Number of Activities

It is necessary to add more information to the initial CSP:

- Constraints to limit the number of activities that belong to each cluster to the value $maxNumAct$, configured by the user or the problem specification.
- Constraints to keep the CSP solver from finding out equivalent solutions. For example, if the BP counts on three activities $A0$, $A1$, $A2$, if the solver allocates these activities in the clusters {1, 1, 2} respectively, that is the same solution than allocating those activities in the clusters {2, 2, 1}, {2, 2, 3} and so on. All those solutions are the same solution, because in all of them the activities $A0$ and $A1$ are in the same cluster and $A2$ is allocated in a different one. The inclusion of these constraints in the model of the CSP reduces the computational complexity, since it gets a huge search space reduction.
- The goal to achieve: an objective function to establish the minimization of the number of test points to allocate.

Table 4 shows these new constraints and goal.

Table 4. New information for the CSP of objective 3.

Constraints:
$\forall i \in \{0, \ldots, nClusters - 1\}\ occurrences(i, clusterOfAct) \leq maxNumAct$
$clusterOfAct_1 = 0$
$\forall i \in \{0, \ldots, nAct - 1\} clusterOfAct_i \leq max(clusterOfAct_j) + 1,\ j \in \{1, \ldots, i - 1\}$
Goal: $minimize(nTestPoints)$

Now the CSP solver can be executed to find the optimal solution. But the computational complexity is exponential, so that it is necessary to add some kind of bound to reduce the search space of the variables in the CSP.

In order to get a bound, a new greedy method is used. It allocates test points in the BP in a linear time. The solution provided by this greedy algorithm may not be the optimal solution, but it provides a very useful bound for the number of test points that reduces drastically the domain of the variables $clusterOfAct_i$.

Applying the algorithm without any bound to the BP in Fig. 1 to get clusters with a maximum size of 5 activities, it allocates 6 test points. The greedy algorithm allocates 9 test points, that obviously is not an optimal solution. But when they are used as a bound, the exhaustive algorithm can find the optimal solution (6 test points) needing only a tenth of the time spent without the bound.

The greedy algorithm is based on the topology of the BPs, taking advantage of the knowledge about the different control flow patterns that are used to model a BP. Since frequently there are topologic structures where a set of branches that form a split are synchronized by means of a join, it is possible to analyze the processes in a deep way. The splits and joins that appear in a BP will enable to divide it in different levels. This is, when a single thread of execution splits into two or more branches, and those branches later converge in a join, the activities in those branches are in an inferior level than the activities in the main thread.

Figure 2 shows an example where the BP counts on nine activities. The splits and joins make that the BP has three levels:

- Level 1: is the main level, composed of all the activities $A0$, $A1$, $A2$, $A3$, $A4$, $A5$, $A6$, $A7$ and $A8$.
- Level 2: composed of the activities $A1$, $A2$, $A3$, $A4$, $A5$, $A6$ and $A7$, which are the activities within the outer split and join.
- Level 3: where the activities $A5$ and $A6$ are included. They are the activities within the second split and join (the inner one).

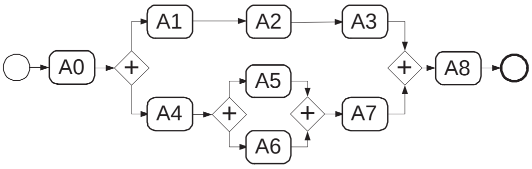

Fig. 2. Business process with three levels.

Based on this idea of levels within the BP, the greedy algorithm is made up of several steps:

1. **Transformation of the BP into a graph.** In this step, the BP is transformed into a labelled directed graph. Each edge's label is used to indicate if it allocates a test point or not. Likewise, the nodes will count on two kinds of information: the name of the activity which the node is representing (information already known in this step), and a label that indicates the levels where the node is within the BP (information obtained in the next step).
2. **Labelling the nodes of the graph.** The labelling of the nodes takes place. This task is performed traversing the graph from the initial state of the BP, following the edges up to the final state. During this walk of the graph, the nodes are labelled depending on the levels where they are situated: the splits are matched to their corresponding joins, and the labels are assigned from upper to lower levels. The label of the main level (which includes the whole BP) is the string "1". The label assigned to the nodes in the rest of levels is formed by the label of its upper level, concatenating a number to represent the new level. The different numbers that appear in a label indicate all the levels where the node is located. Following with the example in Fig. 2, the different labels assigned to its nodes are shown in Fig. 3.
   The three levels in this BP are labelled as "1", "12" and "123". At the same time that these labels are assigned, a tree with the hierarchy of levels is built. Each node of this tree stores the label of a level and the nodes of the graph which are previous and subsequent to that level.

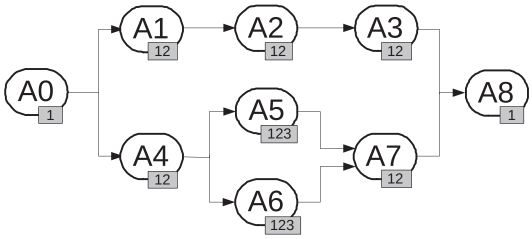

Fig. 3. Graph of the business process with labels in the nodes.

The tree of levels for Fig. 3 is shown in Fig. 4. In each node, the level and the previous and subsequent node for that level appear. Level ”1” does not have previous and subsequent node, since that level represents the whole BP.

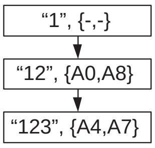

Fig. 4. Tree of levels.

3. **Allocating the test points.** Using the tree of levels, this task performs a recursive process over the levels to allocate the test points. Algorithm 1 shows that recursive process. The algorithm presents some sentences marked with numbers (1, 2, 3) that will be explain in detail in the following.
   - (1) Allocate test points in the input and outputs of a level: the idea is to isolate the activities of a level from the rest of the activities in the BP. Therefore, this sentence entails the fact of allocating test points after the previous node and before the subsequent node of the level. For example, let us suppose that some test points must be allocated in Fig. 3 and the maximum number of activities per cluster is four. The recursive algorithm begins in level ”1” that contains nine activities (the whole BP). Since this level is two large and it is not a leaf in the tree, the algorithm moves forward to the only child in the tree: level ”12”. The number of activities in this level (seven) is also bigger than the maximum per cluster, so that this level will be studied independently of the rest of the activities in the BP and must be isolated: a test point in allocated in the input of the level (output of $A0$) and two in the outputs of the level (outputs of $A3$ and $A7$).
   - (2) Allocate test points in a level: either because the level is a leaf or because it has already been isolated, this sentence entails the moment of allocating test points in the activities of a level using the exhaustive CSP explained at the beginning of this Subsection (2.4).
   - (3) Reduction of the graph: once the test points have been allocated in a level, this level must be considered as a *black box* in upper levels. Therefore the graph must be reduced in order to allocate the test points in the whole BP without taking into account the activities of that level.

Algorithm 1 Recursive algorithm to allocate test points

```
//beginning in the level in the root of the tree of levels
if there are more activities in this level than the maximum activities per cluster then
  if the level is a leaf of the tree of levels then
    (1) allocate test points in the input and outputs of the level in the graph
    (2) allocate test points in the activities of this level in the graph
  else
    for all child c in the tree do
      recursive call: run this algorithm over activities in level c
      if any test point was allocated in level c then
        (3) reduce the graph
      else
        //nothing, the activities of c will be taken into account in the upper level
      end if
    end for
    (1) allocate test points in the input and outputs of the level
    (2) allocate test points in the activities of this level
  end if
else if the number of activities in this level is equal to the maximum activities per cluster then
  (1) allocate test points in the input and outputs of the level
else
  //nothing, the activities of this level will be taken into account in the upper level
end if
```

Let us suppose that in the graph in Fig. 3 the level "123" has been isolated and the test points have already been allocated on it. This level must be replaced by a *black box* delimited by test points, as it is shown in Fig. 5.

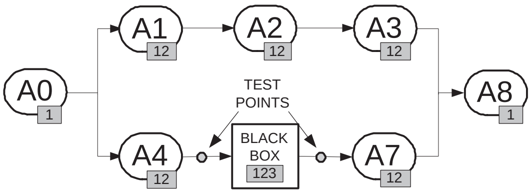

Fig. 5. Reduced graph.

3 Experimental Results

Let us show the results when it comes to apply the presented algorithm to the example in Fig. 1. It is detailed in the following:

- Results of Objective 1 and 2 are shown in Table 5 and 6 respectively.
- Objective 3: Minimize the number of test points to allocate in order to obtain clusters with a maximum number of activities. This objective requires a more complex process, like the labelling of the nodes in the graph (shown in Fig.

Table 5. Results for objective 1.

Results with nTestPoints = 5
test points in outputs of: $A0, A5, A7, A12, A20$
number of obtained clusters: 7
Clusters: $\{A0\}$, $\{A1, A2, A3, A4, A5\}$, $\{A6, A7\}$, $\{A8, A9, A10, A21, A22\}$, $\{A11, A12\}$, $\{A13, A14, A15\}$, $\{A16, A17, A18, A19, A20\}$

Table 6. Results for objective 2.

Results with nClusters = 5
number of test points needed: 4
test points in outputs of: $A0, A5, A12, A21$
Clusters: $\{A0\}$, $\{A1, A2, A3, A4, A5\}$, $\{A6, A7, A8, A9, A10, A22\}$, $\{A11, A12\}$, $\{A13, A14, A15, A16, A17, A18, A19, A20, A21\}$

6), or the obtention of the associated tree (Fig. 7). The final results of the process are shown in Table 7.

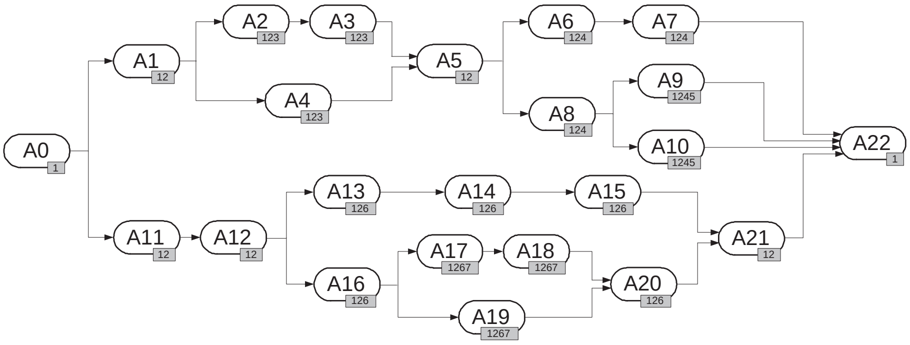

Fig. 6. Graph with labels in the nodes.

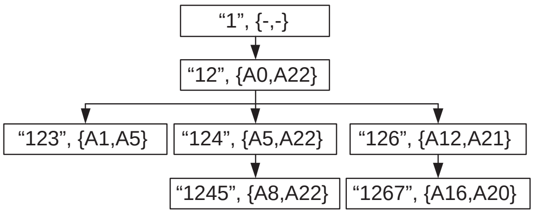

Fig. 7. Tree of labels.

3.1 Execution Time

In this subsection, the computational complexity of the exhaustive and greedy methods in Objectives 1 and 3 are compared. We present the execution time of

Table 7. Results for objective 3.

Results with a maximum of 5 activities per cluster
number of test points needed: 6
test points in outputs of: $A1, A5, A8, A12, A20, A21$
number of clusters obtained: 6
Clusters: $\{A0, A1, A11, A12\}, \{A2, A3, A4, A5\}$, $\{A8\}$, $\{A6, A7, A9, A10, A22\}$, $\{A13, A14, A15, A21\}, \{A16, A17, A18, A19, A20\}$

the explained algorithms applied to some BPs with different number of activities (from 5 to 50 activities per BP). Those BPs are benchmarks that have been generated to check the three different objectives to achieve.

Figure 8 shows the execution time for the Objective 1. In the chart, the execution time of the exhaustive algorithm and the algorithm that uses the greedy method explained can be compared. It is possible to see that the exhaustive method presents an exponential execution time, whereas the computational complexity of the greedy method is linear.

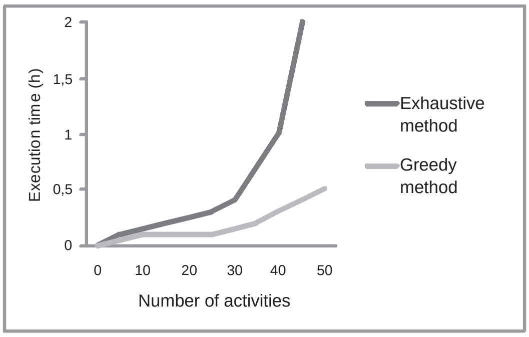

Fig. 8. Execution time for objective 1.

Likewise, Fig. 9 shows the difference between the execution time spent by the exhaustive and the greedy method explained to solve Objective 3. It is possible to see the difference between the exponential execution time for the exhaustive method and the linear complexity when the greedy algorithm is used to establish a bound in the number of test points.

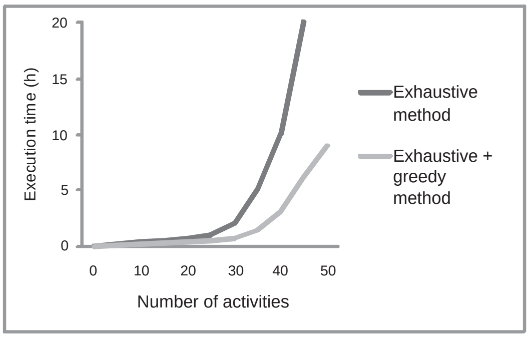

Fig. 9. Execution time for objective 3.

4 Conclusions and Future Work

This work presents an algorithm to allocate test points in BPs. The aim is to apply techniques of allocation of test points in order to improve the computational complexity of isolating faults in the diagnosis process, and make the BPs diagnosable. Three different objectives have been achieved, depending on the user necessity or the problem specification.

As future work, it is interesting to perform the diagnosis of the BPs once the test points have been allocated. They give us additional information that is useful to achieve a more efficient and precise process to find out the minimal diagnosis. Likewise, it can help us with the problem of the scalability of some BPs, making easier the diagnosis process over a BP with many activities.

Acknowledgements

This work has been partially funded by Junta de Andalucía by means la Consejería de Innovación, Ciencia y Empresa (P08-TIC-04095) and by the Ministry of Science and Technology of Spain (TIN2009-13714) and the European Regional Development Fund (ERDF/FEDER).

References

1. Weske, M.: Business Process Management. Concepts, Languages, Architectures. Springer, Berlin (2007)
2. Huang, S., Chu, Y., Li, S., Yen, D.C.: Enhancing conflict detecting mechanism for Web Services composition: A business process flow model transformation approach. Inf. Softw. Technol. vol. 50, num. 11, pp. 1069–1087. ISSN 0950-5849. Butterworth-Heinemann, Newton, MA, USA (2008)
3. Borrego, D, Gasca, R. M., Gómez-López, M. T., Barba, I: Choreography Analysis for Diagnosing Faulty Activities in Business-to-Business Collaboration. 20th International Workshop on Principles of Diagnosis (DX-09). pp. 171–178. Stockholm, Sweden (2009).
4. Rossi, F., van Beek, P., Walsh, T.: Handbook of Constraint Programming. Elsevier. ISBN 978-0-444-52726-4 (2006)
5. Dechter, R: Constraint Processing. Morgan Kaufmann Publisher. ISBN 1-55860-890-7 (2003)
6. Reiter, R.: A theory of diagnosis from first principles. Artificial Intelligence. Elsevier Science Publishers Ltd.vol. 32. num. 1. pp. 57–95. Essex, UK (April, 1987)
7. de Kleer, J., Mackworth, A. K., Reiter, R.: Characterizing diagnoses and systems. Artif. Intell.vol. 56. num. 2-3. pp. 197–222. ISSN 0004-3702. Elsevier Science Publishers Ltd. Essex, UK (1992)
8. Dressler, O., Struss, P.: A Toolbox Integrating Model-based Diagnosability Analysis and Automated Generation of Diagnostics. The 14th International Workshop on Principles of Diagnosis (DX03). pp. 99-104. Washington, D.C., USA (2003)
9. Ceballos, R., Cejudo, V., Gasca, R.M., Del Valle, C.: A Topological-Based Method for Allocating Sensors by Using CSP Techniques. CAEPIA. pp. 62–68. (2005)