An algorithm and a prototype for the dynamic discovery of e-services.

Mohand-Said Hacid, Alain Léger, Farouk Toumani, Christophe Rey

An Algorithm and a Prototype for the Dynamic Discovery of E-Services

Mohand Said Hacid[1] Alain Leger[2] Farouk Toumani[3] Christophe Rey[4]

Research Report LIMOS/RR03-05
Revisited - July 2003

July 8, 2003

[1] LISI, mshacid@lisi.fr
[2] France Telecom R&D, alain.leger@rd.francetelecom.com
[3] LIMOS, ftoumani@isima.fr
[4] LIMOS, rey@isima.fr

Abstract.

In [9] and [10], we have shown that the dynamic discovery of e-services (or web services) can be reduced to the computational problem of finding all minimal transversals with a minimal cost of a hypergraph. Here, we propose an algorithm called *computeBCov* to achieve this task: it is in fact the classical algorithm to generate minimal transversals of a hypergraph (given in [13]) which we have optimized and adapted. We give and theoretically study worst cases for *computeBCov*. We describe the D^2CP prototype in which *computeBCov* has been implemented, and, through many tests, show how it behaves on worst cases and on generated sets of e-services.

Keywords: e-services, web services, discovery, prototype, hypergraphs, transversals

Résumé

Dans [9] et [10], nous avons montré que la découverte dynamique de e-services (ou services web) pouvait être vue comme la recherche des transversaux minimaux de coût minimal dans un hypergraphe. Dans ce rapport, nous proposons un algorithme appelé *computeBCov* pour faire ce calcul : c'est une optimisation et une adaptation de l'algorithme classique de calcul des transversaux minimaux d'un hypergraphe (donné notamment dans [13]). Nous donnons et étudions de très mauvais cas pour *computeBCov* afin d'étudier sa complexité au pire. Nous décrivons le système D^2CP qui implémente *computeBCov*. Au travers de nombreux tests, nous montrons comment ce prototype se comporte sur les mauvais cas ainsi que sur des cas générés aléatoirement par D^2CP.

Mots clés : e-services, services web, découverte, prototype, hypergraphes, transversaux minimaux

Acknowledgement / Remerciements

This study is closely linked to the european project MKBEEM: MKBEEM stands for Multilingual Knowledge Based European Electronic Marketplace (IST-1999-10589, 1st Feb. 2000 - 1st Dec. 2002). Its aim is to provide electronic marketplaces with intelligent, knowledge-based multilingual services.

1 Introduction

The internet are revolutionizing the way companies interact with their suppliers, partners and clients. In the last decade, the number and type of on-line services increased considerably and leads to a new form of automation, namely B2B and B2C e-commerce. A recent industrial initiative envisions a new paradigm for electronic commerce in which applications are wrapped and presented as integrated electronic services (E-services) [1, 2]. An e-service can be defined as an application made available via the Internet by a service provider, and accessible by clients [5]. Examples of e-services currently available range from on-line travel reservation or banking services to entire business functions of an organization. What makes such a vision attractive is that e-services promise to be capable of intelligent interaction by being able to discover and negotiate with each other, compose themselves into more complex services, etc [4, 1].

In this paper, we present an algorithm to achieve the dynamic discovery of e-services: given a user query and a bunch of e-services expressed in the same logical language, this algorithm finds the best subsets (or combinations) of e-services that match the query, according to 2 criteria : the information present in the query but not in the e-services must be minimized, and the information present in the e-services and not in the query must also be minimized.

The detailed formalization of this problem is given in [9] and [10]. It shows that it amounts to compute the minimal transversals with a minimal cost of a hypergraph, in which the vertices are the e-services and the edges are the different parts of the query. This is what our algorithm do by proposing an adaptation and an optimization of the classical algorithm to compute minimal transversals of a hypergraph [3, 6, 13].

The work presented in this paper is part of the project called MKBEEM[5] which aim is to provide electronic marketplaces with intelligent, knowledge-based multilingual services. In this project, e-services are used to describe the offers delivered by the MKBEEM platform independently from specific providers. The algorithm described in this paper allow clients to dynamically discover the available e-services that best meet their needs, to examine their properties and capabilities and possibly to complete missing information.

The rest of this paper is organized as follows. Section 2 summarizes the problem of the dynamic discovery of e-services (detailed in [9] and [10]) to set up the corresponding algorithmic problem of the computation of minimal transversals with a minimal cost of a hypergraph. In Section 3, we explain the algorithm we propose. Section 4 focuses on the implementation of this algorithm. Then in section 5, we discuss some experiments we have done. And eventually, we conclude in Section 6.

[5] MKBEEM stands for Multilingual Knowledge Based European Electronic Marketplace (IST-1999-10589, 1st Feb. 2000 - 1st Dec. 2002).

2 Preliminaries

Let us consider an ontology[6] that contains the following e-services:
- $ToTravel$ allowing to consult a list of trips given a departure place, an arrival place, an arrival date and an arrival time,
- $FromTravel$ allowing to consult a list of trips given a departure place, an arrival place, a departure date and a departure time,
- $Hotel$ allowing to consult a list of hotels given a destination place, the check-in date, the check-out date, the number of adults and the number of children.

Now, assume we have the following query "I want to go from Paris to Madrid on Friday 21st of June, look for an accommodation there for one week (from 21st of June to 28th of June) and rent a car". Our goal is to answer Q with the closest combination of e-services E. Considering our ontology of e-services, the possibly interesting combinations are: $E_1 = \{Hotel, ToTravel\}$ and $E_2 = \{Hotel, FromTravel\}$. To determine the closest to Q, we have to get and measure the two types of extra information, implied by each combination wrt Q, and given in Figure 1. For each combination, these two kinds of *"extra information"* are:

- the information which is contained in the query Q and not contained in any combination (cf. Table 1, column Rest), and
- the information contained in a combination and not contained in the query Q (cf. Table 1, column Missing information).

Solution	Rest	Missing information
E_1	car rental, departure date	arrival date, arrival time, number of adults, number of children
E_2	car rental	departure time, number of adults, number of children

Fig. 1. Example of extra information.

Continuing with the example, the best combinations are discovered by searching the ones that bring the least possible of extra information with respect to the query. It is clear that to better meet the user needs, it is more interesting to try to minimize, in first, the first kind of extra information (i.e., the column Rest). Here, the extra information of $ToTravel$ is "bigger" than the extra information of $FromTravel$. So, the best combination for the query is $\{Hotel, FromTravel\}$. If the Rest of each combination had the same size, then the better combination would be the one with the least Miss (cf. the column Missing Information).

As detailed in [9] and [10], we can use Description Logics, a family of logical languages well known in the Knowledge Representation and Reasoning community, to describe Q and the e-services (see figure 2).

[6] This ontology describes some e-services extracted from the French railways company (SNCF) web site (http://www.sncf.com).

ToTravel	$\doteq$	($\geq$ 1 departurePlace) $\sqcap$ ($\forall$ departurePlace.Location) $\sqcap$ ($\geq$ 1 arrivalPlace) $\sqcap$ ($\forall$ arrivalPlace.Location) $\sqcap$ ($\geq$ 1 arrivalDate) $\sqcap$ ($\forall$ arrivalDate.Date) $\sqcap$ ($\geq$ 1 arrivalTime) $\sqcap$ ($\forall$ arrivalTime.Time)
FromTravel	$\doteq$	($\geq$ 1 departurePlace) $\sqcap$ ($\forall$ departurePlace.Location) $\sqcap$ ($\geq$ 1 arrivalPlace) $\sqcap$ ($\forall$ arrivalPlace.Location) $\sqcap$ ($\geq$ 1 departureDate) $\sqcap$ ($\forall$ departureDate.Date) $\sqcap$ ($\geq$ 1 departureTime) $\sqcap$ ($\forall$ departureTime.Time)
Hotel	$\doteq$	Accommodation $\sqcap$ ($\geq$ 1 destinationPlace) $\sqcap$ ($\forall$ destinationPlace.Location) $\sqcap$ ($\geq$ 1 checkIn) $\sqcap$ ($\forall$ checkIn.Date) $\sqcap$ ($\geq$ 1 checkOut) $\sqcap$ ($\forall$ checkOut.Date) $\sqcap$ ($\geq$ 1 nbAdults) $\sqcap$ ($\forall$ nbAdults.Integer) $\sqcap$ ($\geq$ 1 nbChildren) $\sqcap$ ($\forall$ nbChildren.Integer)
Q	$\doteq$	($\geq$ 1 departurePlace) $\sqcap$ ($\forall$ departurePlace.Location) $\sqcap$ ($\geq$ 1 arrivalPlace) $\sqcap$ ($\forall$ arrivalPlace.Location) $\sqcap$ ($\geq$ 1 departureDate) $\sqcap$ ($\forall$ departureDate.Date) $\sqcap$ Accommodation $\sqcap$ ($\geq$ 1 destinationPlace) $\sqcap$ ($\forall$ destinationPlace.Location) $\sqcap$ ($\geq$ 1 checkIn) $\sqcap$ ($\forall$ checkIn.Date) $\sqcap$ ($\geq$ 1 checkOut) $\sqcap$ ($\forall$ checkOut.Date) $\sqcap$ carRental

Fig. 2. *Logical representation of the e-services and the query Q.*

From the logical representation of the e-services and the query, a hypergraph can be constructed (see figure 3) where the vertices are derived from the e-services, and the edges are derived from the atomic parts of Q.

Then, as it is shown in [9] and [10], finding the closest combinations of e-services amounts to finding the minimal transversals with a minimal cost of the generated hypergraph.

3 Algorithm

In this section we give an algorithm, called *computeBCov*, for computing the minimal transversals with minimal cost of the hypergraph $\widehat{\mathcal{H}}_{\mathcal{T}Q}$ which is the hypergraph built as shown in previous section from the ontology (or "terminology") $\mathcal{T}$ and the query Q. The problem of computing minimal transversals of an hypergraph is central in various fields of computer science [6]. A few algorithms have been designed to solve this problem (see section 3.4). In our case, the problem is slightly different: we want to compute the minimal transversals *with a minimal cost*. Thus we propose an adaptation of the classical algorithm to compute minimal transversals based on a combinatorial optimization technique called branch-and-bound. In addition, we propose a new way to generate the minimal transversal candidates which is an optimization w.r.t. the classical algorithm.

So, in section 3.1, we first explain the classical algorithm to compute minimal transversals of a hypergraph. In section 3.2, we present the optimization in the generation of the minimal transversal candidates. And finally, in section 3.5, we

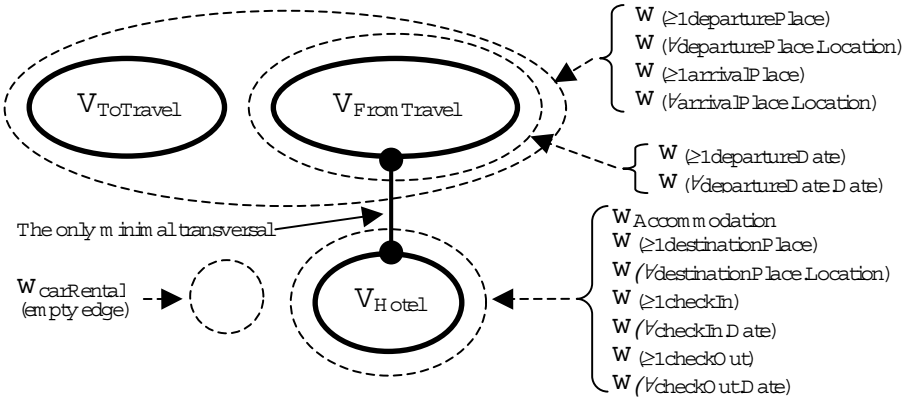


Fig. 3. *Hypergraph $\widehat{\mathcal{H}}_{TQ}$ built from the logical representation of the e-services (the ontology $\mathcal{T}$) and Q.*

present the Branch and Bound adaptation of the classical algorithm, which leads directly to our algorithm *computeBCov*.

3.1 Classical incremental approach[3, 13, 6]

Let us first recall some useful definitions regarding hypergraphs.

Definition 1. ***hypergraph and transversals*** *[6]*
A hypergraph $\mathcal{H}$ is a pair (Σ, Γ) of a finite set $\Sigma = \{v_1, \ldots, v_n\}$ and a set $\Gamma = \{e_1, \ldots, e_m\}$ of subsets of Σ. The elements of Σ are called vertices, and the elements of Γ are called edges.
A set $T \subseteq \Sigma$ is a transversal of $\mathcal{H}$ if for each $\varepsilon \in \Gamma$, $T \cap \varepsilon \neq \emptyset$. A transversal T is minimal if no proper subset T' of T is a transversal. The set of the minimal transversals of an hypergraph $\mathcal{H}$ is noted $Tr(\mathcal{H})$.

To compute minimal transversals of a hypergraph H, the classical way is to start from the minimal transversals set of H', where H' is equal to H minus one of its edges. Then the computation is clearly incremental: to compute minimal transversals of a hypergraph H with one more edge E than another hypergraph H' for which minimal transversals are known, there are two steps (see algorithm 1 below):

- step 1 (line 3 of algorithm 1): the generation of a set of candidates by computation of all possible unions between one transversal X of H' and one vertex e of E (we recall that both X and E are vertices sets),
- step 2 (line 4 of algorithm 1): the pruning of non minimal candidates, ie the candidates in which at least one other candidate is included.

Algorithm 1 *Classical algorithm to compute minimal transversals of a hypergraph (see [13])*

Require: A simple hypergraph H on a set R of vertices
Ensure: The edges of the hypergraph Tr(H) which are the minimal transversals of H
1: Tr:= {∅};
2: **for all** edge E ∈ H **do**
3: Tr := {X∪{e}|X ∈Tr and e∈E};
4: Tr := {X∈ Tr| there is no set Y ∈ Tr with Y⊂X};
5: **end for**

So, at each iteration, many candidates are generated, and then only a few (the minimal ones) are kept at the end of the iteration. Thus, the idea of the optimization presented in the next section is to find a way, at each iteration, to only generate minimal candidates so that the pruning phase become useless.

3.2 Optimization in the candidates generation step

In this section we bring a theorem (theorem 1) which gives a necessary and sufficient condition that characterizes transversal candidates that are not minimal. So it allows us to derive algorithm 2 from algorithm 1: algorithm 2 is an optimization of the classical algorithm in that there is no pruning step anymore and that only minimal transversals are generated at each iteration. Of course the generation step is then more complicated.

Theorem 1. *Let $\mathcal{H}$ be an hypergraph and its associated set of minimal transversals $Tr(\mathcal{H}) = \{X_i \mid i \in \{1,..,m\}\}$. Let $e = \{c_j \mid j \in \{1,..,n\}\}$ be an extra edge of $\mathcal{H}$. Let $\mathcal{H}' = \mathcal{H} \cup e$.*
$\forall i \in \{1,..,m\}$ it holds:

a) if $X_i \cap e \neq \emptyset$:
$\forall j \in \{1,..,n\}$:
- *$(c_j \notin X_i \cap e) \rightarrow (X_i \cup \{c_j\}$ is a transversal of $\mathcal{H}'$ that is not minimal)*
- *$(c_j \in X_i \cap e) \rightarrow (X_i \cup \{c_j\} = X_i$ is a minimal transversal of $\mathcal{H}'$)*

b) if $X_i \cap e = \emptyset$:
$\forall j \in \{1,..,n\}$:
$(X_i \cup \{c_j\}$ is a transversal of $\mathcal{H}'$ that is not minimal$) \leftrightarrow (\exists X_k \in Tr(\mathcal{H}) \mid X_k \cap e = \{c_j\}$ and $X_k \setminus \{c_j\} \subset X_i)$

Proof. a): straightforward.
b): we recall (1) $X_i \cap e = \emptyset$, (2) $X_i \in Tr(\mathcal{H})$, (3) $\mathcal{H}' = \mathcal{H} \cup e$ and (4) $c_j \in e$.
Let $(*) = X_i \cup \{c_j\}$ is a non minimal transversal of $\mathcal{H}'$.
Let $(**) = \exists X_k \in Tr(\mathcal{H}) \mid X_k \cap e = \{c_j\}$ and $X_k \setminus \{c_j\} \subset X_i$. Let's note $tr(\mathcal{H})$ the set of all transversals of $\mathcal{H}$ (recall that $Tr(\mathcal{H})$ is the set of all *minimal* transversals of $\mathcal{H}$).

$$
\begin{aligned}
(*) \overset{(1,2,3)}{\Leftrightarrow} \quad & \exists Y \mid Y \in Tr(\mathcal{H}') \text{ and} \\
& \qquad Y \subset X_i \cup \{c_j\} \\
\overset{(1,2,3,4)}{\Leftrightarrow} \quad & \exists Y \mid Y \in Tr(\mathcal{H}') \text{ and} \\
& \qquad c_j \in Y \text{ and} \\
& \qquad Y \setminus \{c_j\} \subset X_i \\
\Leftrightarrow \quad & \exists Y \mid \forall e' \in \mathcal{H}',\ Y \cap e' \neq \emptyset \text{ and} \\
& \qquad \forall c_y \in Y,\ Y \setminus \{c_y\} \notin tr(\mathcal{H}') \text{ and} \\
& \qquad c_j \in Y \text{ and} \\
& \qquad Y \setminus \{c_j\} \subset X_i \\
\Leftrightarrow \quad & \exists Y \mid \forall e' \in \mathcal{H}',\ Y \cap e' \neq \emptyset \text{ and} \\
& \qquad Y \setminus \{c_j\} \notin tr(\mathcal{H}') \text{ and} \\
& \qquad \forall c_y \in Y,\ c_y \neq c_j,\ Y \setminus \{c_y\} \notin tr(\mathcal{H}') \text{ and} \\
& \qquad c_j \in Y \text{ and} \\
& \qquad Y \setminus \{c_j\} \subset X_i \\
\overset{(2,3)}{\Leftrightarrow} \quad & \exists Y \mid \forall e' \in \mathcal{H}',\ Y \cap e' \neq \emptyset \text{ and} \\
& \qquad Y \setminus \{c_j\} \notin tr(\mathcal{H}) \text{ and} \\
& \qquad \forall c_y \in Y,\ c_y \neq c_j,\ Y \setminus \{c_y\} \notin tr(\mathcal{H}') \text{ and} \\
& \qquad c_j \in Y \text{ and} \\
& \qquad Y \setminus \{c_j\} \subset X_i \\
\overset{(3,4)}{\Leftrightarrow} \quad & \exists Y \mid \forall e' \in \mathcal{H}',\ Y \cap e' \neq \emptyset \text{ and} \\
& \qquad Y \setminus \{c_j\} \notin tr(\mathcal{H}) \text{ and} \\
& \qquad \forall c_y \in Y,\ c_y \neq c_j,\ Y \setminus \{c_y\} \notin tr(\mathcal{H}) \text{ and} \\
& \qquad c_j \in Y \text{ and} \\
& \qquad Y \setminus \{c_j\} \subset X_i \\
\overset{(3,4)}{\Leftrightarrow} \quad & \exists Y \mid \forall e' \in \mathcal{H},\ Y \cap e' \neq \emptyset \text{ and} \\
& \qquad Y \setminus \{c_j\} \notin tr(\mathcal{H}) \text{ and} \\
& \qquad \forall c_y \in Y,\ c_y \neq c_j,\ Y \setminus \{c_y\} \notin tr(\mathcal{H}) \text{ and} \\
& \qquad c_j \in Y \text{ and} \\
& \qquad Y \setminus \{c_j\} \subset X_i \\
\Leftrightarrow \quad & \exists Y \mid Y \in Tr(\mathcal{H}) \text{ and} \\
& \qquad c_j \in Y \text{ and} \\
& \qquad Y \setminus \{c_j\} \subset X_i \\
\overset{(1,4)}{\Leftrightarrow} \quad & \exists Y \mid Y \in Tr(\mathcal{H}) \text{ and} \\
& \qquad Y \cap e = c_j \text{ and} \\
& \qquad Y \setminus \{c_j\} \subset X_i \\
\Leftrightarrow \quad & (**)
\end{aligned}
$$

Proposed algorithm for the minimal transversals generation

The algorithm 2 presented below is an improvement of the classical algorithm 1 based on theorem 1.

Another trivial improvement is, at the beginning of the algorithm, to put in Tr all vertices that are in 1-vertex edges, since these vertices belong to all transversals.

Algorithm 2 *Improvement of the classical algorithm to compute minimal transversals of a hypergraph*

Require: A simple hypergraph H on a set R of vertices
Ensure: The edges of the hypergraph $Tr(H)$ which are the minimal transversals of H

$Tr := \{E|\ E$ is a 1-vertex edge of $H\}$;
for all edge $E \in H|\ |E| \geq 2$ **do**
  $Tr_{np} := \emptyset$;
  $Tr_{1p} := \emptyset$;
  $Tr_{notp} := \emptyset$;
  for all $X \in Tr$ **do**
    $I := E \cap X$;
    if $|I| \geq 2$ **then**
      $Tr_{np} := Tr_{np} \cup \{X\}$;
    else if $|I| = 1$ **then**
      $Tr_{1p} := Tr_{1p} \cup \{(X, I)\}$;
    else
      $Tr_{notp} := Tr_{notp} \cup \{(X, E)\}$;
    end if
  end for
  for all $(X, I) \in Tr_{1p}$ **do**
    for all $(Y, E) \in Tr_{notp} | X \backslash_I \subset Y$ **do**
      $E := E \backslash_I$;
    end for
  end for
  $Tr := Tr_{np}$;
  for all $(X, I) \in Tr_{1p}$ **do**
    $Tr := Tr \cup \{X\}$;
  end for
  for all $(Y, E) \in Tr_{notp}$ **do**
    for all vertex $s \in E$ **do**
      $Tr := Tr \cup \{Y \cup \{s\}\}$;
    end for
  end for
end for

Algorithms 1 and 2 allow to compute minimal transversals of a hypergraph. Our problem is to compute minimal transversals with a minimal cost of a hypergraph. So in section 3.5, we will see that we can adapt algorithms 1 and 2

to this problem via the adding of a combinatorial optimization technique called Branch and Bound: we have called the obtained algorithm *computeBCov*.

But before, in the next section, we will discuss about the complexity of algorithm 2, and compare it with algorithm 1 in worst cases.

3.3 Complexity of algorithm 2

Number of minimal transversals In this section, we want to evaluate the optimization we have proposed in the last section for the computation of minimal transversals w.r.t. the classical algorithm. The theoretical complexity of this problem is not yet known. Of course, it is well known that there can be an exponential (with respect to the input size) number of minimal transversals in a hypergraph: for example, if there are m edges of 2 vertices in a hypergraph H, knowing that each edge has 2 vertices that are only in this edge (so there are $n = 2 * m$ vertices), then the number of minimal transversals will be 2^m. So the real theoretical problem is whether there exists an output-polynomial algorithm (that is polynomial with respect to the combined size of the hypergraph and the number of minimal transversals. Up to our knowledge, the best theoretical time bound is given in [14], where it is shown that the generation of the transversal hypergraph can be done in incremental subexponential time $k^{O(logk)}$, where k is the combined size of the input and the output.

With theorem 1, we know a way to generate at each iteration only the minimal transversals of the hypergraph built with the edges already examined. So it is natural to think that theorem 1 could give us a way to evaluate the number of minimal transversals generated at each iteration and so the global number of minimal transversals. That would be very useful to determine the complexity of algorithm 2 with respect to the output size.

Unfortunately, it seems that theorem 1 is not sufficient to derive such information about the complexity of algorithm 2. Indeed, the study of many examples allows us to do the following remarks:

- A 1-persisting can make non-minimal 0,1 or 2 candidates built from a non-persisting (we don't know if there is an upper bound).
- It can happen that all candidates generated by a non-persisting may be non-minimal. That means that the number of minimal transversals generated at iteration i can be smaller than the number of thoses generated at iteration $i-1$. This implies that it is difficult to evaluate the number of minimal transversals that will be generated at i knowing the number of those generated at $i-1$.

However, through these examples, it seems (but NOTHING is proven) that two different 1-persistings cannot make non-minimal a same candidate built from a non-persisting. If it were true, it could be a good start point towards an evaluation of the number of minimal transversal generated at each iteration.

Worst cases for algorithms 1 and 2

We present now a way to construct worst cases for both algorithms 1 and 2, based on an estimation of the numbers of inclusion tests and transversal generations at each iteration. We give examples of hypergraphs that are worst cases for both algorithms.

Suppose we have an hypergraph H with m edges and n vertices. We will examine two sorts of operations:

- The inclusion test between two sets of vertices: as the maximum number of vertices there can be in a set of vertices is n, then testing an inclusion between two vertices sets is $\mathcal{O}(n^2)$ if sets are not supposed to be ordered ($\mathcal{O}(n)$ if they are ordered). As the computation of an intersection between two sets of vertices has the same complexity as an inclusion test between the same two sets, then we will count together inclusion tests and intersection computations.
- The generation of a new transversal as the union between an existing transversal (generated at iteration $i-1$) and a vertex of the edge e_i examinated at iteration i: this operation is $\mathcal{O}(n)$ if we suppose that we have to test that the vertex that will be added is not already in the existing transversal ($\mathcal{O}(1)$, that is constant else).

To cope with our implementation in which inclusion tests are $\mathcal{O}(n^2)$ and generations are $\mathcal{O}(n)$, we can say that there is a factor n between an inclusion test and a transversal generation.

Now, let x_{i-1} be the number of minimal transversals at the end of iteration $i-1$. At iteration i, the edge e_i is examinated.

At each iteration, algorithm 1 works in two steps:

- the generation step: the number of transversals that are generated is always $x_{i-1} * |e_i|$,
- the pruning step: to identify the minimal transversals, it is mandatory to test whether each generated transversal is included in each other generated transversal. So the number of inclusion tests is $\mathcal{O}((x_{i-1} * |e_i|)^2)$, with $|e_i| = \mathcal{O}(n)$.

So it is clear that the worst case is the one where each edge of the hypergraph has vertices that are only in this edge. It is the previously evoked case where there is an exponential number of minimal tranversals. This number is obviously $\mathcal{O}(n^m)$.

Example 1. An example, with 13 edges and 2 vertices by edge (so 26 vertices), we can construct the following hypergraph:

$$H_1 = \{\{1,2\},\{3,4\},\{5,6\},\{7,8\},\{9,10\},\{11,12\},\{13,14\},\{15,16\},\{17,18\}$$
$$\{19,20\},\{21,22\},\{23,24\},\{25,26\}\}$$

and then the set $Tr(H_1)$ of all the minimal transversals of H is built with $2^{13} = 8192$ minimal transversals. For algorithm 1, the number of inclusion tests

is:

$$\sum_{i=1}^{13} 2^i * (2^i - 1) = 89462102$$

The way algorithm 2 works is a little bit more complicated. At iteration i, there are three steps:

- Step 1: the intersection between the x_{i-1} minimal transversals generated at iteration $i-1$ and the $|e_i|$ vertices of edge e_i is computed in order to determine for each minimal transversal generated at iteration $i-1$ if it is a 1-, n- or not-persisting at iteration i. Let respectively r_i, s_i and t_i be the number of 1-, n-, and not-persisting at iteration i.
- Step 2: according to theorem 1, each vertex of e_i that makes a transversal 1-persisting has to be removed from that 1-persisting in order to test the strict inclusion between this 1-persisting minus this vertex and each not-persisting. So this is a step of inclusion tests.
- Step 3: each not-persisting generates some new minimal transversals with a maximum of $|e_i|$ new minimal transversals by not-persisting.

In order to build a worst case, as intersection computations (that is step 1) are mandatory, the only way is to maximize both the number of inclusion tests in step 2 and transversal generations in step 3. As we have seen before, an inclusion test is more expensive than a generation, but in order to have the most inclusion tests possible during the execution, we need to have a lot of transversals at each iteration, so their number must also be maximized.

First it seems obvious to say that to maximize both numbers (inclusion tests and transversal generations), the number s_i of n-persisting at each iteration has to be zero. Indeed if $s_i = 0$, $r_i + t_i$ will be maximum, and so the number of inclusion tests and transversal generations will be greater.

If we try to maximize the only number of inclusion tests, we have to remark that the number of inclusion tests at iteration i is always $r_i * t_i$. As $s_i = 0$ and $r_i + t_i = x_{i-1}$, then it is easy to show that the number of inclusion tests will be the greatest if and only if:

$$\text{if } x_{i-1} \text{ is even, then } r_i = t_i = x_{i-1}/2$$

$$\text{if } x_{i-1} \text{ is odd, then } r_i = \lfloor x_{i-1}/2 \rfloor \text{ and } t_i = \lceil x_{i-1}/2 \rceil$$

$$\text{or } t_i = \lfloor x_{i-1}/2 \rfloor \text{ and } r_i = \lceil x_{i-1}/2 \rceil$$

So we can say that the maximal number of inclusion tests, for iteration i, is $\mathcal{O}(x_{i-1}^2)$.

If we try to maximize the only number of transversal generations, then it is the same remarks as those for algorithm 1: the worst case is the one where the total number of minimal transversals is exponential according to the input size ($\mathcal{O}(n^m)$).

So, in order to construct a worst case for algorithm 2, we have to build a hypergraph that has an exponential number of transversals and such that, at each iteration i of algorithm 2, there is no n-persisting, and there are as many 1-persisting as not-persisting.

Example 2. As we can see in figure 4, the following hypergraph follows near exactly these recommandations:

$$H_2 = \{\{1,2,3,4\},\{3,4,5,6\},\{5,6,7,8\},\{7,8,9,10\},\{9,10,11,12\},\{11,12,13,14\},$$
$$\{13,14,15,16\},\{15,16,17,18\},\{17,18,19,20\},\{19,20,21,22\},\{21,22,23,24\},$$
$$\{23,24,25,26\},\{25,26,27,28\}\}$$

The numbers of 1-, n- and not-persisting and the number of transversals evolves as shown in figure 4. As we can see, H_2 is nearly a worst case for algorithm

Iteration	1	2	3	4	5	6	7	8	9	10	11	12	13
Number of 1-persisting	0	2	4	4	16	24	48	112	192	416	832	1600	3328
Number of n-persisting	0	0	0	0	0	0	0	0	0	0	0	0	0
Number of not-persisting	1	2	2	8	12	24	56	96	208	416	800	1664	3264
Number of min transversals	4	6	12	28	48	104	208	400	832	1632	3264	6592	13056

Fig. 4. Evolution of algorithm 2 on H_2.

2 because the total number of minimal transversals is exponential according to the input size and at each iteration there are nearly as many 1-persisting as not-persisting.

Example 3. Now, figure 5 compare number of inclusions tests in the executions of algorithms 1 and 2 for H_1 and H_2.

Figure 6 sums up the different results discussed in this section for the worst cases of algorithms 1 and 2.

Before seeing how to using algorithm 2, we first references a list of papers that focus on the problem of minimal transversals.

3.4 Minimal transversals bibliography

The problem of computing minimal transversals of an hypergraph is central in various fields of computer science [6, 7]. The precise complexity of this problem is still an open problem. In [14], it is shown that the generation of the transversal hypergraph (i.e. the set of all minimal transversals) can be done in incremental subexponential time $k^{O(log(k))}$, where k is the combined size of the input and the output. To our knowledge, this is the best theoretical time bound for the problem

Iteration	1	2	3	4	5	6	7	8	9	10	11	12	13
# of minimal transversals for H_1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192
# of inclusion tests for algo 1 on H_1	2	12	56	240	992	4032	16256	65280	261632	1047552	4192256	16773120	67100672
# of inclusion tests for algo 2 on H_1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192
# of minimal transversals for H_2	4	6	12	28	48	104	208	400	832	1632	3264	6592	13056
# of inclusion tests for algo 1 on H_2	12	30	132	756	2256	10712	43056	159600	691392	2661792	10650432	43447872	170446080
# of inclusion tests for algo 2 on H_2	4	10	20	60	240	680	2896	11152	40768	174688	668864	2668992	10875648

Fig. 5. Number of inclusion tests.

	# candidates generations	# tests inclusions and intersection computations	# of minimal transversals at the end of the iteration
Algorithm 1	$x_{i-1} * \|e_i\|$	$(x_{i-1} * \|e_i\|)^2$	$x_i = x_{i-1} * \|e_i\|$
Algorithm 2	$\frac{x_{i-1}}{2} * \|e_i\|$	$x_{i-1} + \frac{x_{i-1}^2}{4}$	$x_i = \frac{x_{i-1}}{2} * (\|e_i\| + 1)$

Fig. 6. Critical values at each iteration in worst cases for algorithms 1 and 2.

of the generation of the transversal hypergraph. Other important complexity results can be found in [8].

In an algorithmic point of view, the first and classical algorithm (algorithm 1 taken from [13]) is derived from a property on hypergraph transversals given in [3] and in [6]. Our proposition, algorithm 2, is an optimization of this classical algorithm. Another optimization of the classical algorithm is proposed in [11]: in that paper, a property called "appropriate node" is given that allows to generate only minimal candidates. Thus, appropriate nodes and our theorem 1 about the persistings have the same role. In addition, appropriate nodes provide the interesting feature of allowing the generation of transversals in a depthfirst manner, thus saving a lot of space used during the execution of the algorithm. The counterpart is that the computation of appropriate nodes seems to be time-consuming.

Two other approaches to compute minimal transversals have been proposed which have no link with the classical algorithm. In [12], a levelwise strategy is used to compute minimal transversals. This technique is efficient if the average cardinality of the minimal transversals is reduced. For other cases, it is very time-consuming. In [15] a depth-first enumeration of the vertices, based on the lectic order, is achieved: this heuristic seems to give better results than the previous algorithm, due to a limited space search and so a limited memory usage.

3.5 The *computeBCov* algorithm

In this section, we present the algorithm *computeBCov*. *computeBCov* computes the transversals with a minimal cost of the hypergraph $\widehat{\mathcal{H}}_{\mathcal{T}Q}$.

Knowing algorithms 1 and 2, the *naive* approach to compute minimal transversals with a minimal cost of $\widehat{\mathcal{H}}_{\mathcal{T}Q}$ would consist in computing all minimal transversals of $\widehat{\mathcal{H}}_{\mathcal{T}Q}$, evaluating their costs and then picking those which have the lowest cost.

Instead, *computeBCov* uses the iterative structure of algorithm 1 and 2. Indeed, at each iteration, once the minimal transversals have been computed, an additional pruning criteria is used to reduce the number of these minimal transversals at this iteration, while ensuring that the minimal transversals with a minimal cost are still computable from those remaining minimal transversals.

The main idea behind *computeBCov* is to use a Branch-and-Bound like enumeration of transversals to prune amongst minimal transversals those which will not generate transversals with a minimal cost. At the beginning of *computeBCov* (line 3), a simple heuristic is used to efficiently compute a cost of an apriori *good* transversal (i.e., a transversal expected to have a small cost). This is carried out by adding, for each edge of the hypergraph, the cost of the vertex that has the minimal cost. The resulting cost is stored in the variable $CostEval$. As we have, for any set of vertices $X = \{V_{S_i}\}$:

$$cost(X) = |Miss_{E_X}(Q)| \leq \sum_i |Miss_{S_i}(Q)| = \sum_{V_{S_i} \in X} cost(\{V_{S_i}\})$$

the evaluation is an upper bound of the cost of a feasible transversal. Then, at each iteration, after the generation of the minimal transversals (line 5), we can eliminate from Tr any minimal transversal that has a greater cost than $CostEval$ (line 8 and line 9), since that candidate cannot lead to a transversal that is better than what we already know. Then, from each candidate transversal that remains in Tr, we compute a new evaluation for $CostEval$ by considering only remaining edges (line 11). If one of these new evaluations is strictly lower than the current evaluation, we replace $CostEval$ by this new evaluation (lines 12 and 13).

At the end of the algorithm, each computed minimal transversal $X \in Tr$ is translated into a concept E_X which constitutes an element of the solution to the $\mathcal{BCOV}(\mathcal{T}, Q)$ problem.

3.6 The *CostEval* computation policy

The Branch and Bound optimization technique implies that a new evaluation of an a priori good transversal has to be computed at each iteration of algorithm 3 (cf line 11). The principle is the same as the one explained in section 3.5 to compute the first evaluation of an a priori good transversal: to the current cost of the current transversal at iteration i (stored in the variable $RealCost$), we add the cost of the vertices that have the lowest cost for all edges that have not been taken into account yet. This is the $CostEval$ computation policy 1. In

Algorithm 3 ***computeBCov***

Require: An instance $\mathcal{BCOV}(\mathcal{T}, Q)$ of the best covering problem.
Ensure: The set of the best covers of Q using $\mathcal{T}$.

Build the associated hypergraph $\widehat{\mathcal{H}}_{\mathcal{T}Q} = (\Sigma, \Gamma')$.
$Tr \leftarrow \emptyset$.

$$CostEval \leftarrow \sum_{e \in \Gamma'} \min_{V_{S_i} \in e} (|Miss_{S_i}(Q)|);$$

for all edge $e \in \Gamma'$ **do**
  $Tr \leftarrow$ {minimal transversals generated as proposed in algorithm 2}
  for all minimal transversal $X \in Tr$ **do**
    $RealCost \leftarrow |Miss_{E_X}(Q)|$;
    if $RealCost > CostEval$ **then**
      $Tr \leftarrow Tr \setminus X$;
    else if $RealCost < CostEval$ **then**

$$Eval \leftarrow RealCost + \sum_{f \in \Gamma' |\ f \cap X = \emptyset} \min_{V_{S_i} \in f} (|Miss_{S_i}(Q)|);$$

      if $Eval < CostEval$ **then**
        $CostEval \leftarrow Eval$;
      end if
    end if
  end for
end for
for all $X \in Tr$ such that $|Miss_{E_X}(Q)| = CostEval$ **do**
  return the concept E_X.
end for

fact in algorithm 3, there is a modified version of policy 1: instead of adding the cost of the vertices that have the lowest cost for all edges that have not been taken into account yet, we add these costs only for edges that have not been taken into account yet **and** that have an empty intersection with the current transversal, because other edges are obviously covered by this transversal. This is policy 2. It is obvious that policy 2 implies that the values of $CostEval$ will be at least as good as in policy 1, and often better (lower). So with policy 2, the Branch and Bound technique will be more efficient. However, policy 2 needs some intersection computations. As it is not a priori clear which policy will give the better overall results, we have implemented both. Results will be given in section 5. In fact, we will see that policy 2 is generally better.

4 Prototype

In this section we present the Java prototype in which *computeBCov* has been implemented. We have called this system D^2CP, for "Dynamic Discovery of Concepts Prototype". After the presentation of D^2CP capabilities (section 4.1), we will see how D^2CP helps the user in generating ontologies (section 4.2), in running different algorithms to discover e-services (section 4.3) and in displaying results (section 4.4).

4.1 Overview of D^2CP

The purpose of this system is to be a platform for testing the *computeBCov* algorithm. The main functionalities of D^2CP are the following:

- It allows the user to generate random acyclic terminologies of e-services. These are generated from DTD files and from parameters given by the user via D^2CP's interface. These generated terminologies are then stored as XML files. See section 4.2.
- It allows the user to search the best covers of a query Q using:
  - its own terminologies,
  - terminologies generated by D^2CP (in order to test the computation on large sets of e-services).

  See section 5.
- It allows the user to run different versions of the best covers computation algorithm, which are of course *computeBCov* but also any algorithm that can be derived from algorithm 1 and 2. See section 4.3[7].
- It allows the user to run these algorithms:
  - with a precise tree-like trace, which is very convenient to verify the execution, and very didactic in order to understand how they work,
  - with time measures in order to compare the different algorithms.

  See section 5.

Figure 7 sums up the functionalities of the system and figure 8 shows the graphical user interface of D^2CP.

[7] From now the name *computeBCov* will stand for the algorithm that is detailed in section 3.5, but also for any variant of it.

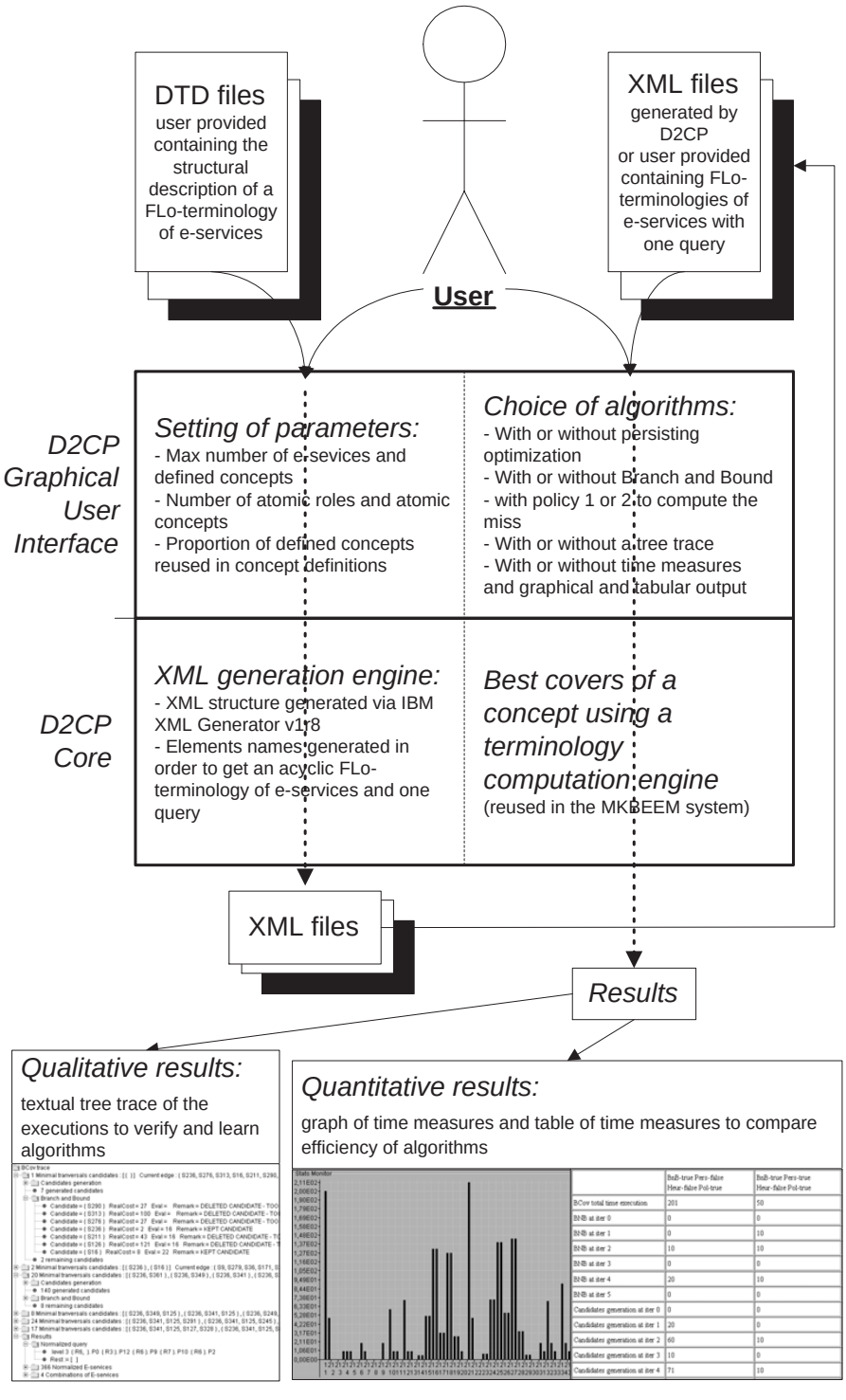


Fig. 7. Overview of the D^2CP functionalities.

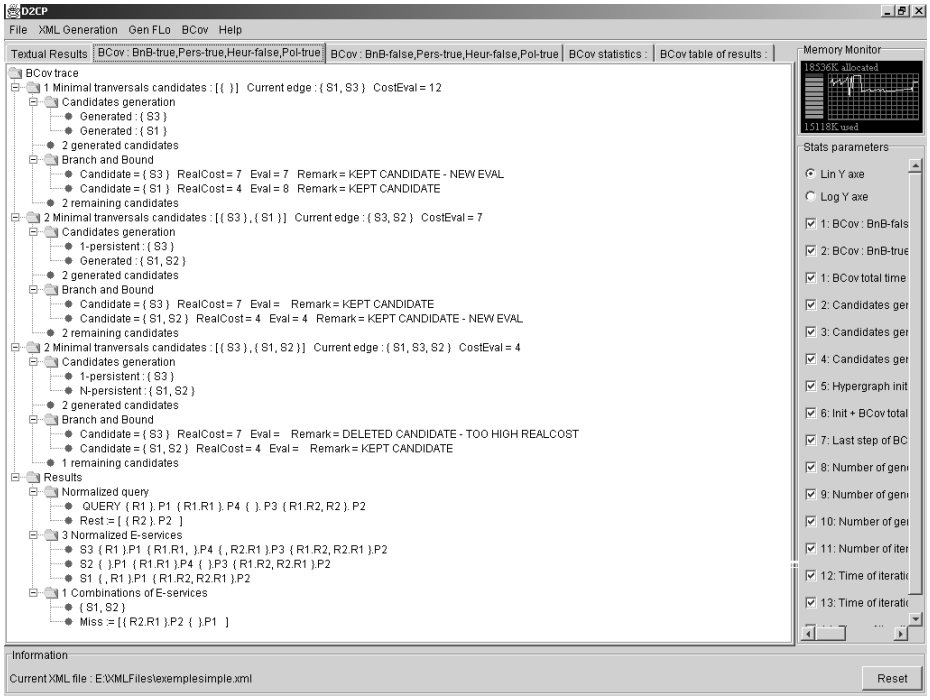


Fig. 8. The D^2CP graphical user interface.

4.2 Generation of terminologies

The ontology generation part of D^2CP allows to generate ontologies as XML files according to a DTD file that describe the basic structure of the language used. This generation is divided into two parts:

- first, the structure of the XML ontology is created via the use of IBM XML Generator v1r8[8],
- second, some element names and values are generated in order to obtain an acyclic ontology valid wrt the dynamic discovery of e-services process.

As the dynamic discovery of e-services is implemented for the description logic $\mathcal{FL}_0$, the DTD file from which XML $\mathcal{FL}_0$-ontologies files are generated contains the structural definition of the $\mathcal{FL}_0$ description logic. Figure 9 shows this DTD and figure 10 shows the 2 steps in the generation of an $\mathcal{FL}_0$-ontology.

```
<!-- X-FLo DTD 1.0 -->

<!ELEMENT FLo-Ontology  (Name , Description , Author , Creation-Date , Version , Conceptualization )>

<!ELEMENT Name  (#PCDATA )>
<!ELEMENT Description  (#PCDATA )>
<!ELEMENT Author  (#PCDATA )>
<!ELEMENT Creation-Date  (#PCDATA )>
<!ELEMENT Version  (#PCDATA )>

<!-Conceptualization : 50%  of e-services and 50% of concepts -->
<!ELEMENT Conceptualization  (Concept-Definition | Eservice-Definition)+>

<!-- Eservice Definitions -->
<!ELEMENT Eservice-Definition ( Concept-Name, Concept-Description )>

<!-- Concept Definitions -->
<!ELEMENT Concept-Definition ( Concept-Name, Concept-Description )>

<!ELEMENT Concept-Name (#PCDATA )>

<!ELEMENT Concept-Description (AND) >

<!-Conjunction : Between 1 and 6 conjuncts -->
<!ELEMENT AND ( (Concept-Description-Bis), (Concept-Description-Bis)?, (Concept-Description-Bis)?,
(Concept-Description-Bis)?, (Concept-Description-Bis)?, (Concept-Description-Bis)?) >

<!ELEMENT Concept-Description-Bis  ( Atomic-Concept | FORALL ) >

<!ELEMENT Atomic-Concept (#PCDATA )>

<!ELEMENT Atomic-Role (#PCDATA )>

<!ELEMENT FORALL  (Atomic-Role, Concept-Name )>
```

Fig. 9. The DTD file containing the definition of the $\mathcal{FL}_0$ description logic.

Before the generation, the D^2CP user can adjust 7 parameters:

[8] See http://www.alphaworks.ibm.com/tech/xmlgenerator.

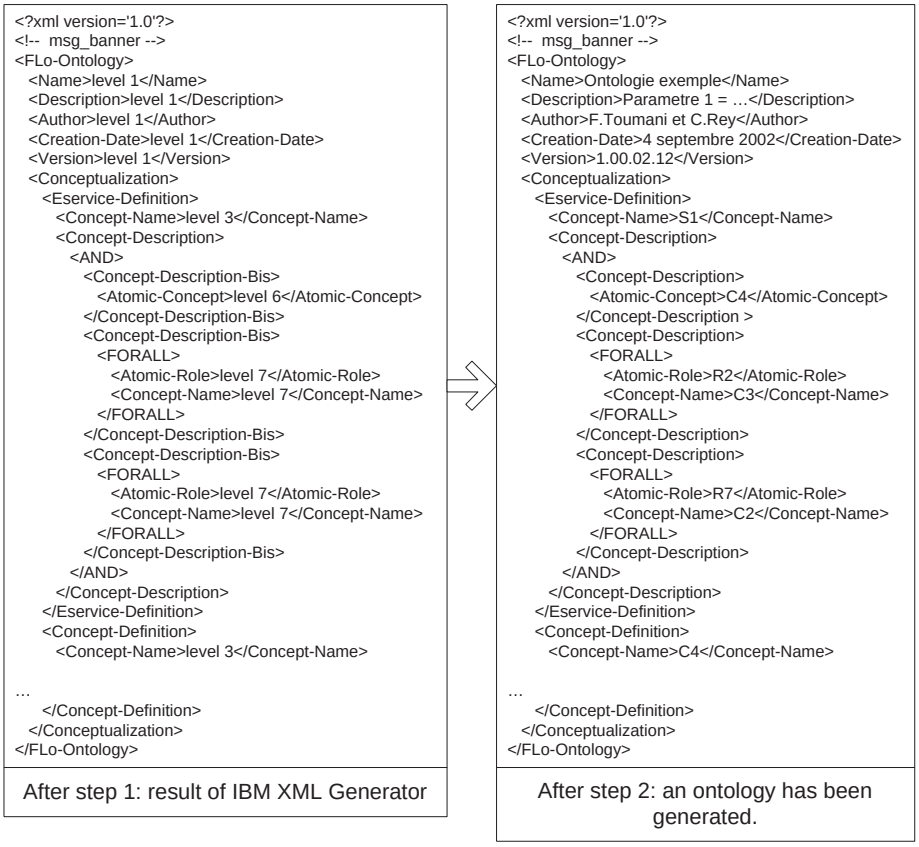


Fig. 10. A generated XML ontology containing the e-service $S_1 \doteq C_4 \sqcap \forall R_2.C_3 \sqcap \forall R_7.C_2$.

- the maximal number of both concept definitions and e-services definitions in the ontology: this parameter is allowed by the IBM XML Generator API used in D^2CP and can be set by the user via a dialog box in D^2CP (see figure 11).
- the maximal number of child elements of the element *AND* in concept descriptions: this can be done by modifying the DTD file by hand, that is by adding or removing *(Concept-Description-Bis)?* as a child of the element *AND*. For example,
  <!ELEMENT AND ((Concept-Description-Bis), (Concept-Description-Bis)?, (Concept-Description-Bis)?, (Concept-Description-Bis)?, (Concept-Description-Bis)?, (Concept-Description-Bis)?) >
  says that *AND* has at least one child and at most 6.
- the proportion of concept definitions wrt e-services definitions: this can be done by modifying the DTD file. For example,
  <!ELEMENT Conceptualization (Concept-Definition | Concept-Definition | Eservice-Definition)+>
  says that there will be 66% of concept definitions and 33% of e-services definitions in the XML ontology.
- the proportion of the *Atomic-Concept* elements wrt the *FORALL* elements: this can be done by modifying the DTD file. For example,
  <!ELEMENT Concept-Description-Bis (Atomic-Concept | FORALL | FORALL | FORALL) >
  says that the child of the element *Concept-Description-Bis* will be an "Atomic-Concept" for 25% of all *Concept-Description-Bis*, and *FORALL* for 75%.
- the number of atomic concepts used in the ontology: this can be set by the user via a dialog box in D^2CP (see figure 11).
- the number of atomic roles used in the ontology: this can be set by the user via a dialog box in D^2CP (see figure 11).
- the proportion of defined concepts that are reused in the definition of other concepts or e-services: this can be set by the user via a dialog box in D^2CP (see figure 11).

4.3 *computeBCov* and the other algorithms

D^2CP allows the user to test at most 6 versions of *computeBCov* to dynamically discover e-services from an XML ontology, that is to compute minimal transversals with a minimal cost of a hypergraph. Indeed, via the dialog box shown in figure 12, the user can choose between three couples of parameters that will decide which algorithm will be applied to achieve the dynamic discovery of e-services.

In figure 12, we can see that the user can choose whether he wants to run an execution of *computeBCov*

- with or without the Branch and Bound optimization

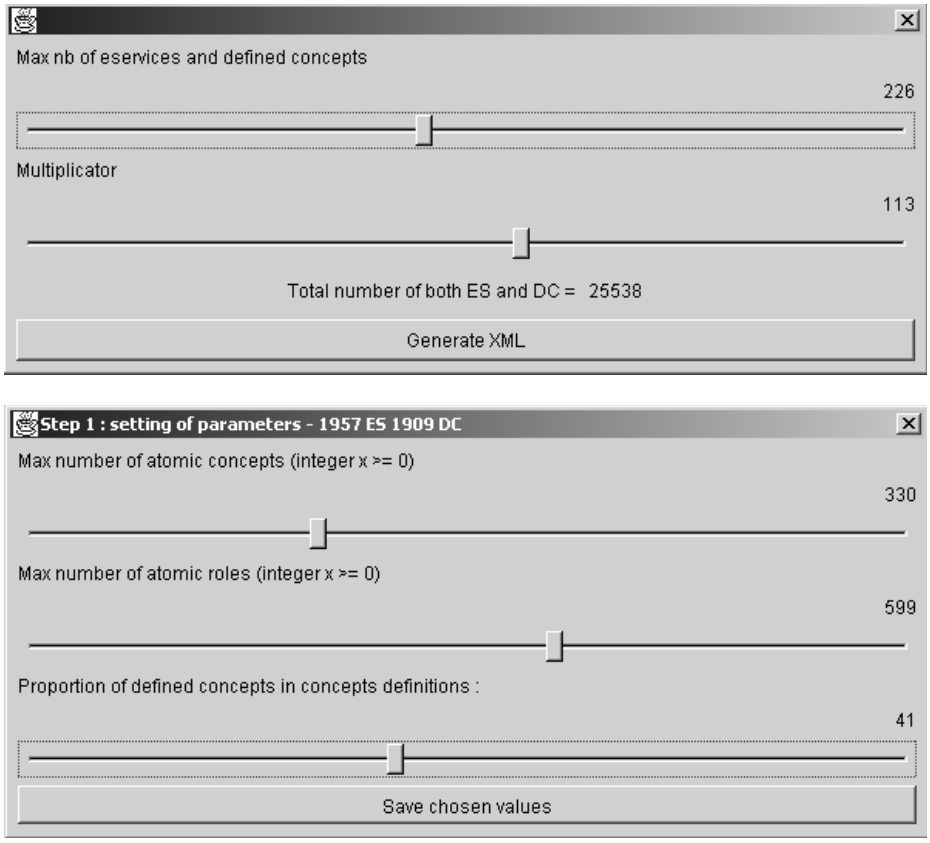


Fig. 11. Dialog boxes of D^2CP that allows to set parameters of XML generation.

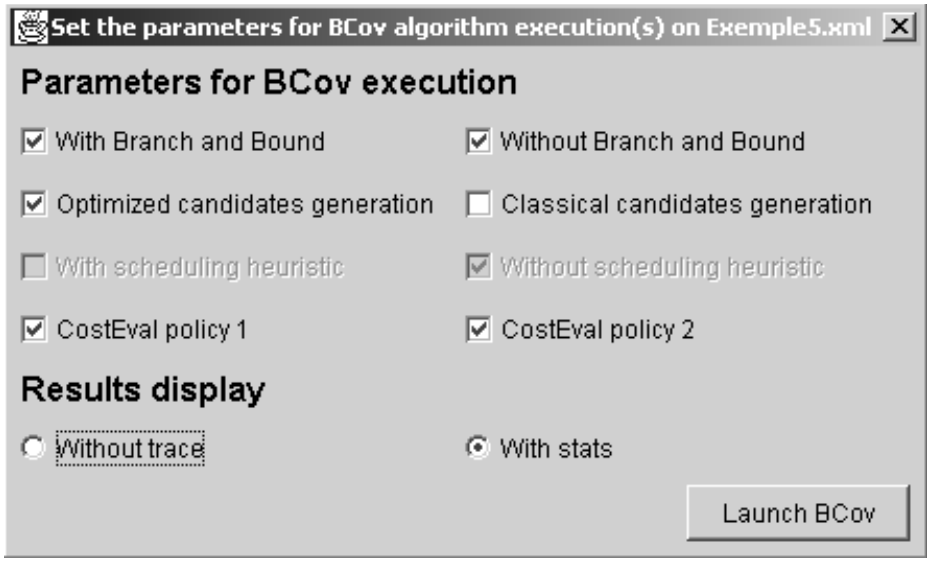


Fig. 12. Dialog box of D^2CP that allows to set parameters of the algorithms that will be run to achieve the dynamic discovery of e-services.

- with the optimized candidates generation (that is with algorithm 2 to generate minimal transversals) or with the classical candidates generation (with algorithm 1)
- and with policy 1 or policy 2 to compute *CostEval* (see section 3.6) if the Branch and Bound has been chosen (if it is not chosen, there is no difference between policy 1 and policy 2, since they are evaluation functions used in the Branch and Bound).

According to the user choices[9], D^2CP will run one algorithm for each possibility. In the case of figure 12, D^2CP will run 3 different algorithms:

- one with BnB, optimized generation and policy 1,
- one with BnB, optimized generation and policy 2 and
- one without BnB, optimized generation (policy 1 or policy 2 is useless since the BnB is not chosen).

All *computeBCov*variants are summarized in figure 13 where they are ordered according to their theoretical efficiency.

4.4 Results display

In figure 12, we can see that the user can choose the type of output he wants D^2CP to display:

[9] The gray line about a "scheduling heuristic" stands for an heuristic that has not been implemented yet in D^2CP.

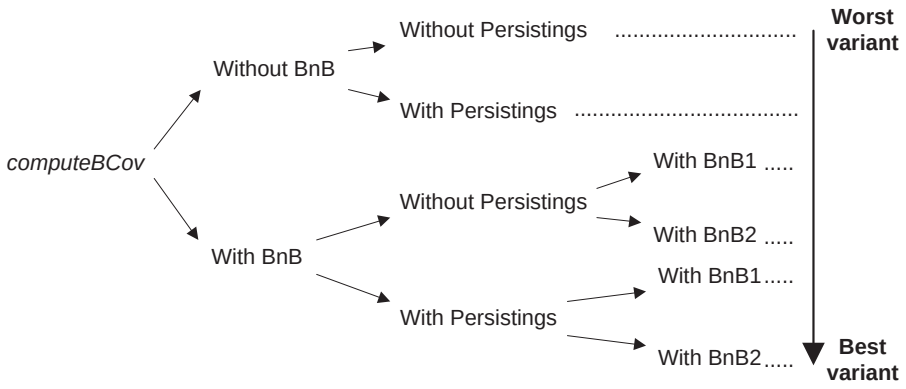


Fig. 13. All 6 variants of *computeBCov* ranked according to their a priori theoretical performances.

- without trace and without stats: textual results only,
- textual results plus a trace of each execution (see figure 8 for an example of a trace),
- textual results plus statistics about each execution: statistics are output under two ways:
  - as an interactive histogram, with a linear or logarithmic scale, where the user can choose the values he wants to compare by checking the check-boxes on the right of the GUI (see figure 14)
  - and as a table of values that is moreover saved as an HTML file, so that the user can import these data into a spreadsheet program (see figure 15).
- or textual results, plus histogram, plus statistics.

When the user choose both the trace and the stats display, then D^2CP make two different executions for each variant of *computeBCov* it has to run: indeed the display of the trace may slower the execution because of the display of each candidate for all iterations. So to have significant statistical results, there must be two different executions.

5 Tests

In this section, we first study in details one execution of *computeBCov* on a small example. This allows us to explain *computeBCov* on an example and to verify that *computeBCov* is well implemented into D^2CP. Then we study the two examples detailed in section 3.3 and representing worst cases for *computeBCov*À t least we will study the effectiveness of the different versions of *computeBCov* on two bigger examples generated by D^2CP.

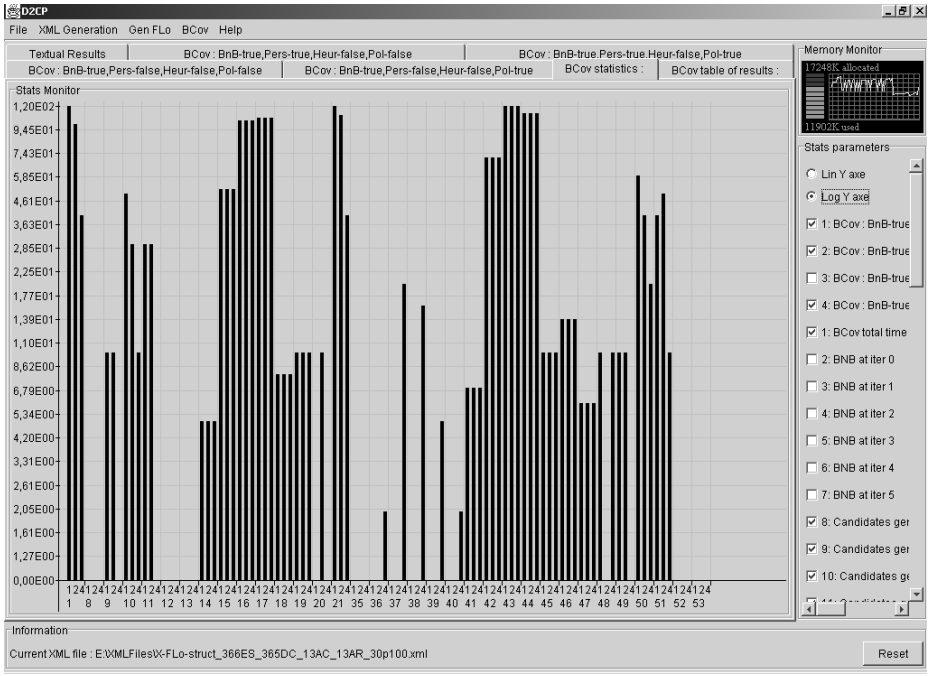
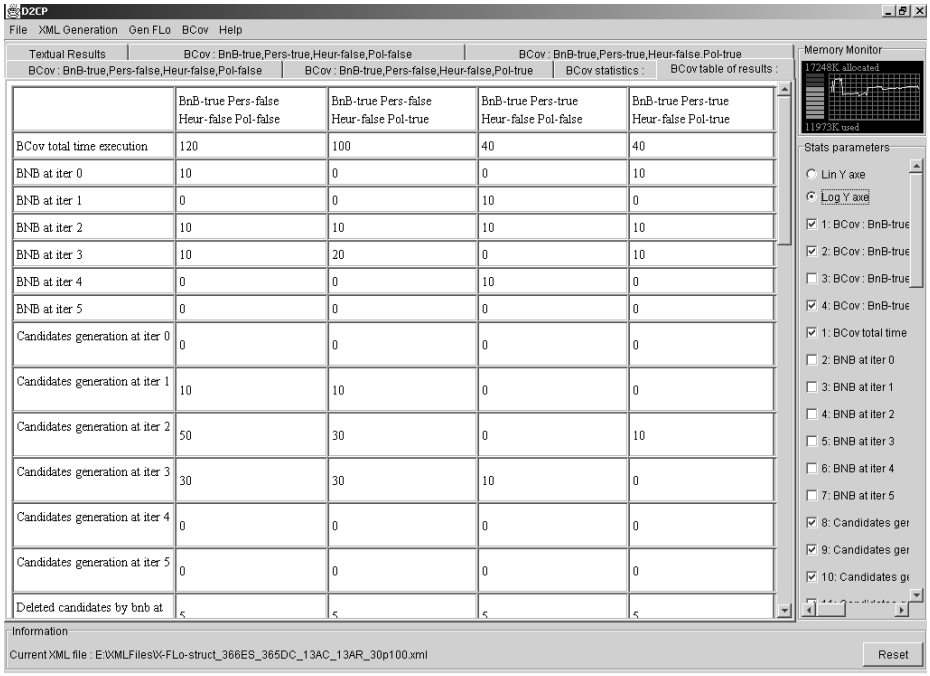


Fig. 14. The graphical representation of statistics in D^2CP.



	BnB-true Pers-false Heur-false Pol-false	BnB-true Pers-false Heur-false Pol-true	BnB-true Pers-true Heur-false Pol-false	BnB-true Pers-true Heur-false Pol-true
BCov total time execution	120	100	40	40
BNB at iter 0	10	0	0	10
BNB at iter 1	0	0	10	0
BNB at iter 2	10	10	10	10
BNB at iter 3	10	20	0	10
BNB at iter 4	0	0	10	0
BNB at iter 5	0	0	0	0
Candidates generation at iter 0	0	0	0	0
Candidates generation at iter 1	10	10	0	0
Candidates generation at iter 2	50	30	0	10
Candidates generation at iter 3	30	30	10	0
Candidates generation at iter 4	0	0	0	0
Candidates generation at iter 5	0	0	0	0
Deleted candidates by bnb at	5	5	5	5

Fig. 15. The tabular representation of statistics in D^2CP.

5.1 Functional test

The example presented below is the example which execution trace is shown in figure 8. Before explaining *computeBCov* on it, let's see its main components.

The terminology $\mathcal{T}$ of defined concepts C_i is the following:

- $C_1 \doteq \forall R_1.P_1$
- $C_2 \doteq \forall R_1 R_2.P_2$
- $C_3 \doteq \forall R_2.P_2$
- $C_4 \doteq \forall \epsilon.P_3$
- $C_5 \doteq \forall R_1 R_1.P_4$

The e-services expressed with the C_i are the following:

- $S_1 \doteq C_1 \sqcap C_2 \sqcap \forall \epsilon.P_1 \sqcap \forall R_2 R_1.P_2$
- $S_2 \doteq C_2 \sqcap C_4 \sqcap C_5 \sqcap \forall \epsilon.P_1 \sqcap \forall R_2 R_1.P_2$
- $S_3 \doteq C_1 \sqcap C_2 \sqcap C_4 \sqcap C_5 \sqcap \forall R_2 R_1.P_2 \sqcap \forall R_2 R_1.P_3 \sqcap \forall \epsilon.P_4$

The query Q expressed with the C_i is:
$Q \doteq C_1 \sqcap \forall R_1.C_3 \sqcap R_2.P_2 \sqcap C_4 \sqcap C_5$

So the normalized e-services and query are:

- $S_1 \doteq \forall \epsilon.P_1 \sqcap \forall R_1.P_1 \sqcap \forall R_1 R_2.P_2 \sqcap \forall R_2 R_1.P_2$

- $S_2 \doteq \forall\epsilon.P_1 \sqcap \forall R_1R_2.P_2 \sqcap \forall R_2R_1.P_2 \sqcap \forall\epsilon.P_3 \sqcap \forall R_1R_1.P_4$
- $S_3 \doteq \forall R_1.P_1 \sqcap \forall R_2R_1.P_2 \sqcap \forall R_1R_2.P_2 \sqcap \forall R_2R_1.P_3 \sqcap \forall\epsilon.P_3 \sqcap \forall R_1R_1.P_4 \sqcap \forall\epsilon.P_4$
- $Q \doteq \forall R_1.P_1 \sqcap \forall R_1R_2.P_2 \sqcap \forall R_2.P_2 \sqcap \forall\epsilon.P_3 \sqcap \forall R_1R_1.P_4$

From this knowledge, an hypergraph $H_{\mathcal{T}Q}$ is built (see [9, 10]). This hypergraph is shown in figure 16. *computeBCov* computes the minimal transversals with a minimal cost of this kind of hypergraphs.

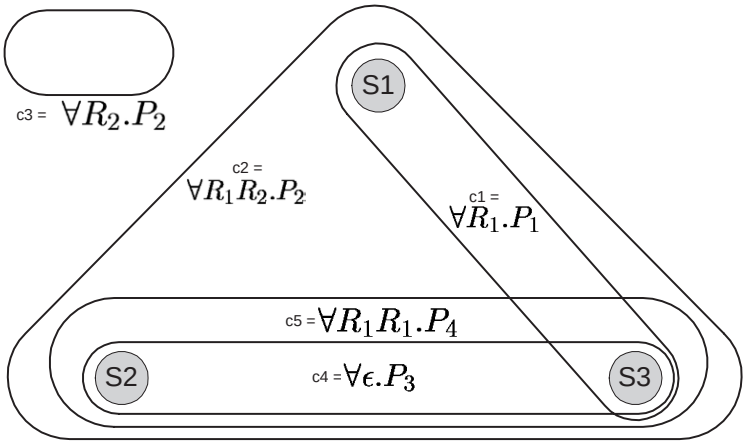


Fig. 16. *Hypergraph $H_{\mathcal{T}Q}$: edges are the atomic parts of Q and vertices are the e-services S_i.*

Let us now explain the execution of *computeBCov* on $H_{\mathcal{T}Q}$, which is shown in figure 8. The version of *computeBCov* which trace is shown in figure 8 uses algorithm 2 to computes transversal candidates at each iteration, Branch and Bound optimization, and policy 2 to compute $CostEval$. As there are only 3 different edges in $H_{\mathcal{T}Q}$, there are only 3 iterations.

- Iteration 1:
  - The set of transversals computed during last iteration is obviously empty.
  - The first evaluation of a feasible transversal for $H_{\mathcal{T}Q}$ has a cost of 12 (sum of the costs of the vertex that have the lowest cost for all edges of $H_{\mathcal{T}Q}$).
  - The current edge is $\{S_1, S_3\}$.
  - So there are neither 1-, n-, nor not-persisting, only 2 generated transversals: $\{S_1\}$ and $\{S_3\}$.
  - Branch an Bound:
    - The cost of $\{S_3\}$ is 7, and from $\{S_3\}$ a transversal of $H_{\mathcal{T}Q}$ can be built with a cost of 7, so the evaluation changes from 12 to 7.

    * The cost of $\{S_1\}$ is 4, and from $\{S_1\}$ a transversal of $H_{\mathcal{T}Q}$ can be built with a cost of 8, so the evaluation stays to 7.
- Iteration 2:
  - The set of transversals computed during last iteration is $\{\{S_1\}, \{S_3\}\}$.
  - The current evaluation is 7.
  - The current edge is $\{S_3, S_2\}$.
  - $\{S_3\}$ is 1-persisting (because $|\{S_3\} \cap \{S_3, S_2\}| = 1$), and $\{S_1\}$ is not-persisting (because $|\{S_1\} \cap \{S_3, S_2\}| = 0$). As $\{S_3\} \setminus (\{S_3\} \cap \{S_3, S_2\}) = \emptyset \subset \{S_1\}$, then we know, according theorem 1, that $\{S_1\} \cup (\{S_3\} \cap \{S_3, S_2\}) = \{S_1\} \cup \{S_3\}$ will not be minimal. So the only generation at this iteration will be $\{S_1\} \cup \{S_2\}$.
  - Branch an Bound:
    * The cost of $\{S_3\}$ is still 7, so, as the current evaluation is also 7, we know that it is useless to try to find from $\{S_3\}$ a new evaluation that strictly lower than 7. But $\{S_3\}$ can still be a minimal transversal with a minimal cost of $H_{\mathcal{T}Q}$, that's why it is kept.
    * The cost of $\{S_1, S_2\}$ is 4, and from $\{S_1, S_2\}$ a transversal of $H_{\mathcal{T}Q}$ can be built with a cost of 4, so the evaluation changes from 7 to 4.
- Iteration 3:
  - The set of transversals computed during last iteration is $\{\{S_3\}, \{S_1, S_2\}\}$.
  - The current evaluation is 4.
  - The current edge is $\{S_1, S_3, S_2\}$.
  - $\{S_3\}$ is 1-persisting (because $|\{S_3\} \cap \{S_1, S_3, S_2\}| = 1$), and $\{S_1, S_2\}$ is n-persisting (because $|\{S_1, S_2\} \cap \{S_1, S_3, S_2\}| = 2 > 1$). As there is no other transversals from last iteration, then the transversals of this iteration are still $\{S_3\}$ and $\{S_1, S_2\}\}$.
  - Branch an Bound:
    * The cost of $\{S_3\}$ is still 7, so, as the current evaluation is 4 and stricty lower, we know that $\{S_3\}$ is has not a minimal cost. So $\{S_3\}$ is deleted.
    * The cost of $\{S_1, S_2\}$ is 4. As the current evaluation is also 4, nothing has to be done.
- End: there is no edge left, so the last step consists in keeping among the transversals that remain those which have the minimal cost. Here, only $\{S_1, S_2\}$ remains, so $\{S_1, S_2\}$ is the only combination of e-services that best answers the query Q. What is called the "rest" is the part of Q that was in any e-service, and what is called the "miss" is the part of $\{S_1, S_2\}$ that was not in Q.

5.2 Effectiveness tests for *computeBCov* worst cases

In this section, we use D^2CP to see how the *computeBCov* algorithm behaves with hypergraphs H_1 and H_2, which are theoretical very bad cases.

In figures 17 and 18, we compare the execution of *computeBCov* on H_1 and H_2 with

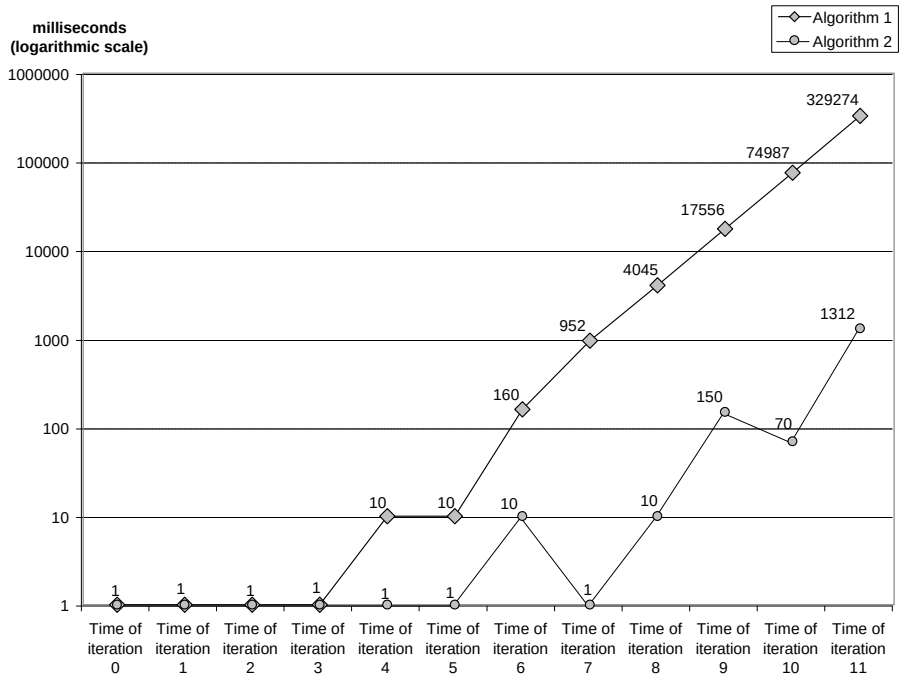


Fig. 17. *Execution of both algorithms 1 and 2 on H_1 by D^2CP.*

- "algorithm 1" : the variant of *computeBCov* without Branch and Bound and with the classical candidate generation (with algorithm 1) and
- "algorithm 2" : the variant of *computeBCov* without Branch and Bound and with the optimized candidate generation (with algorithm 2).

Figure 19 sums up the overall time execution of D^2CP for each execution of both algorithms on both cases.

Then, in figure 20, we study the behaviour of the best theoretical variant of *computeBCov* on increasing size hypergraphs instances built as H_2 is. As H_2 is a very bad case for all *computeBCov* variant, then this evolution represents an upper bound in computation times of same size inputs for *computeBCov*.

Figure 20 confirms what is theoretically true: as there can be an exponential number of solutions, *computeBCov* is, in the worst case, computable in exponential time. But it is interesting to remark the following points:

- in the very bad case of H_2, it takes about 1 second to solve an instance of the problem which has until 3264 solutions (best combinations of services)

- and it takes between 1 and 20 seconds to solve an instance with until 13056 solutions.

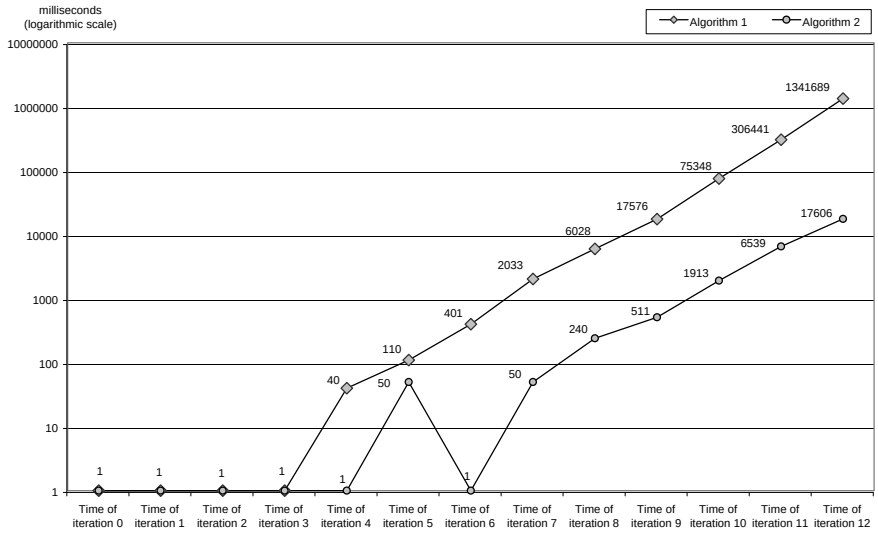


Fig. 18. *Execution of both algorithms 1 and 2 on H_2 by D^2CP.*

5.3 Effectiveness tests for generated cases

In this section we study three cases which have been generated by D^2CP. After presenting them, we show and discuss the time results of the executions of the variants of *computeBCov* on these three cases.

Presentation of three study cases These cases are presented in figure 21.

Let's now study these examples in details and try to discover if they can be considered realistic or not. For us, a realistic ontology (called Mark ontology in figure 21) and a realistic query, wrt the characteristics of figure 21, have:

- at least 3000 concepts (both defined and e-services),
- at least 5 times more defined concepts than of e-services,
- at least 20 conjuncts in the query,
- at least 1 atomic concept for 20 concepts (defined or e-services),
- at least 1 atomic role for 30 concepts (defined or e-services)
- and at least 30% of defined concepts reused in concept definitions.

These values take into account our experience in the domain of e-services modeling. However, this is a very first set of values that may greatly vary in the future. Anyway, knowing these give us a mark to evaluate each case.

Case 1 is a small ontology, especially concerning the number of defined concept (only 365 defined concepts and 366 e-services). Its query is also very little (6 conjuncts and about 10 e-services per conjunct). But only 13 atomic concepts

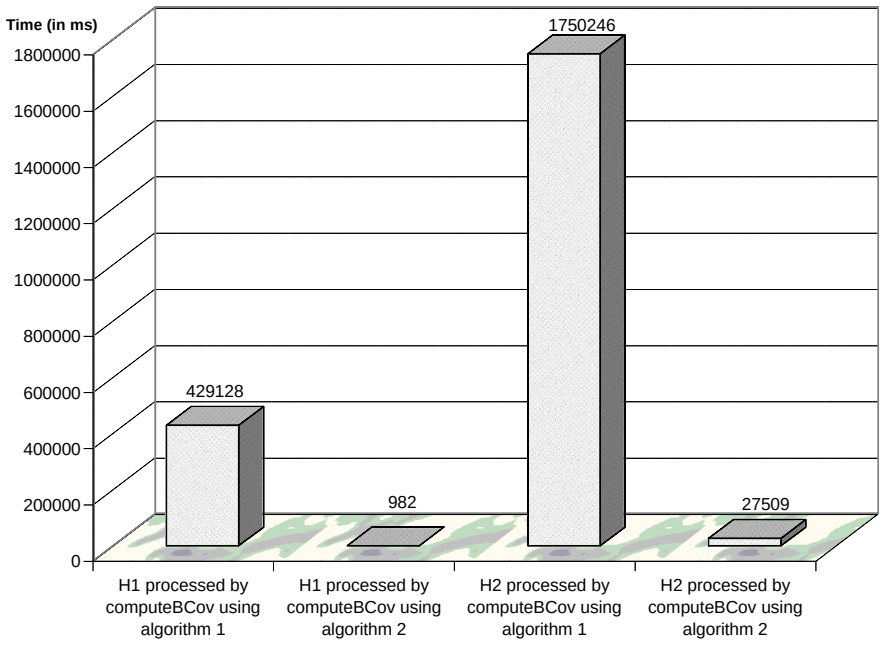


Fig. 19. *Overall time execution of computeBCov on H_1 and H_2 by D^2CP.*

and roles make this case more difficult than one can imagine for a real ontology. Indeed these two values imply that each atomic concept and role is often used in concepts description (of defined concepts and e-services). So the corresponding problem of finding minimal transversals with a minimal cost will obviously be more difficult. At last, the proportion of 30% of defined concepts reused in concept definitions appears to be quite realistic (even if a real ontology might have a greater proportion). To sum up, case 1 represents a small ontology with a small query, both quite realistic concerning their structure, but that may be a bad case for *computeBCov*.

Case 2 is a greater ontology (but still quite small): 660 e-services and 1334 defined concepts. With 33 conjuncts and an average of about 20 e-services by conjunct, its query can be considered to be quite realistic. The number of atomic roles and concepts stay quite low comparing to what one can imagine for a realistic ontology having this size, implying that this case may be a bad case for *computeBCov*. The proportion of 20% of defined concepts reused in concepts definitions is perhaps a little low but not totally unrealistic. To sum up, case 2 represents a small ontology with a realistic query, that may be a bad case for *computeBCov*.

Case 3 is an ontology which size is quite realistic: about 4000 concepts (defined and e-services) and six times more defined concepts (3405) than of e-

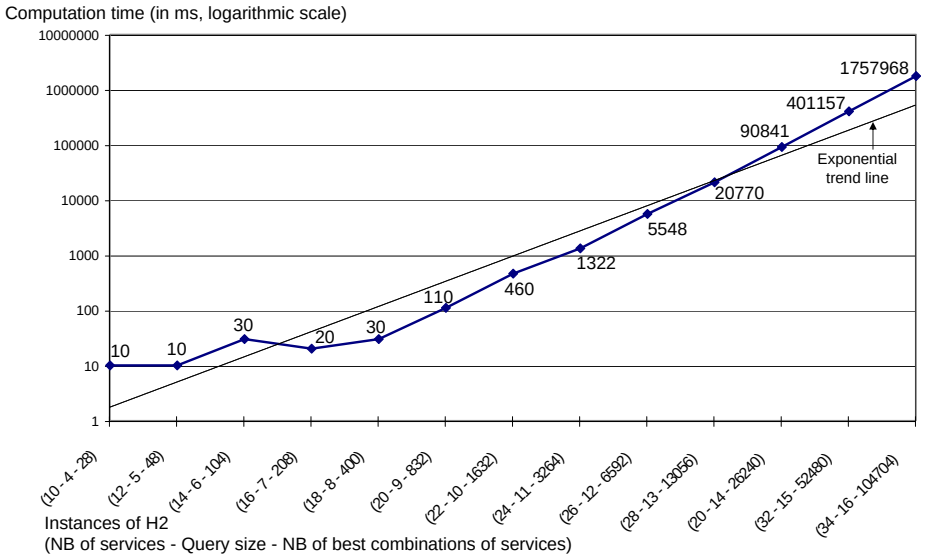


Fig. 20. Overall time computation of the best variant of *computeBCov* applied on different size varying instances of H_2.

Characteristic	Mark ontology	Case 1	Case 2	Case 3
Number of defined concepts	2500	365	1334	3405
Number of e-services (ie. of vertices in the hypergraph)	500	366	660	570
Number of conjuncts in the query (ie. of edges in the hypergraph)	20	6	33	12
Average number of e-services for one conjunct	20	10.83	20.84	30.75
Number of atomic concepts	150	13	30	12
Number of atomic roles	100	13	30	12
Proportion of defined concepts used in concept definitions	30%	30%	20%	33%

Fig. 21. Main characteristics of the 3 cases generated by D^2CP.

services (570). With 12 conjuncts and about 30 e-services by conjunct, the query is small but still realistic. Only 12 atomic roles and atomic concepts imply that it will be a bad case for *computeBCov*. 33% of defined concepts in concept definitions indicate that the way concept descriptions are built is quite realistic. To sum up, case 3 represents a realistic ontology that is a bad case for *computeBCov* with a quite small query.

Overall time results of *computeBCov* executions We have run each variant of *computeBCov* (8 variants, see section 4.3) on the three cases. The overall time results are given in figure 22.

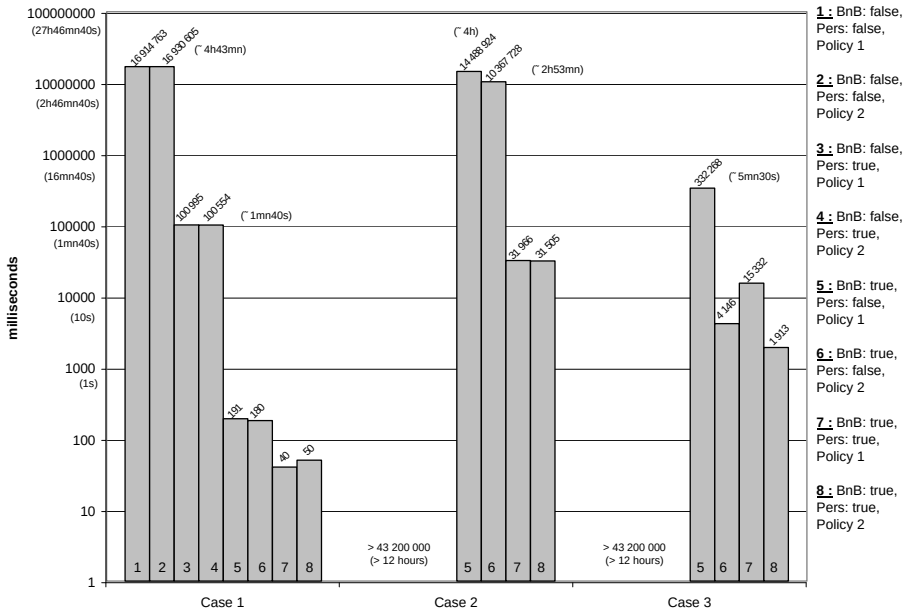


Fig. 22. *Overall time results for the execution of each variant of computeBCov on the three study cases by D^2CP.*

From figure 22, we can say that:

- The Branch and Bound optimization technique is very interesting:
  - for case 1, even without the persisting, it allows a real-time solution
  - and for cases 2 and 3, it allows the study of the impact of the persistings and the two policies, and coupled with the persisting, it nearly allows a real-time solution.

  Clearly, the impact of the BnB is to limit the combinatorial explosion of the problem.

– The persistings are also very interesting, but always associated with the BnB. They also limit the combinatorial explosion and even allow to get close to a real-time solution, especially for case 2.
– Policy 2 is better than policy 1: for cases 2 and 3, it helps the BnB to give better results.

So the couple BnB with policy2) and persistings appears to be efficient to solve the dynamic discovery of e-services on these three generated examples. As we have seen before, cases 2 and 3 are close to real cases about their size but are worse cases than real ones because of their little number of atomic concepts and roles. So we can imagine that performances would be at least as good for real normal cases.

Detailed study of cases 1, 2 and 3 In this section we will study in details the results of the executions of D^2CP for cases 1, 2 and 3. We discuss each case on the basis of the graphical representations of the following results:

– the execution time of each iteration (figure 23 for case 1, figure 26 for case 2 and figure 30 for case 3).
– the number of generated minimal transversal candidates at each iteration (figure 24 for case 1, figure 27 for case 2 and figure 31 for case 3).
– the average time for the generation of one minimal transversal candidate at each iteration (figure 25 for case 1, figure 28 for case 2 and figure 32 for case 3).

Moreover, for cases 2 and 3, we provide two graphs (figure 29 and 33) which help to see the effect of the candidates generation and BnB steps on each other. After explaining each case, we sum up the main results (see figure 34).

Case 1 (figures 23, 24 and 25) From 23 we can distinguish 3 groups of variants of *computeBCov*

– the very slow variants: without BnB and without theorem 1 (for minimal transversals generation),
– the slow variants: without BnB and with theorem 1
– and the quick variants: with BnB (time of each iteration lower than 100ms).

There are three closely linked arguments that explain this distribution:

– First, it is clear, from figure 24, that the BnB technique limits (or even avoids in this case) the exponential explosion of the number of generated minimal transversal candidates that occurs for variants without BnB. For variants with BnB, the number of candidates is more or less constant during the 6 iterations.
– Second, the generation step (with or without theorem 1) is such that the average time to compute one candidate increase linerarly with the number of candidates (see figures 24 and 25). It is easy to understand: in this generation step, the more there are candidates, the more there will be possibilities to

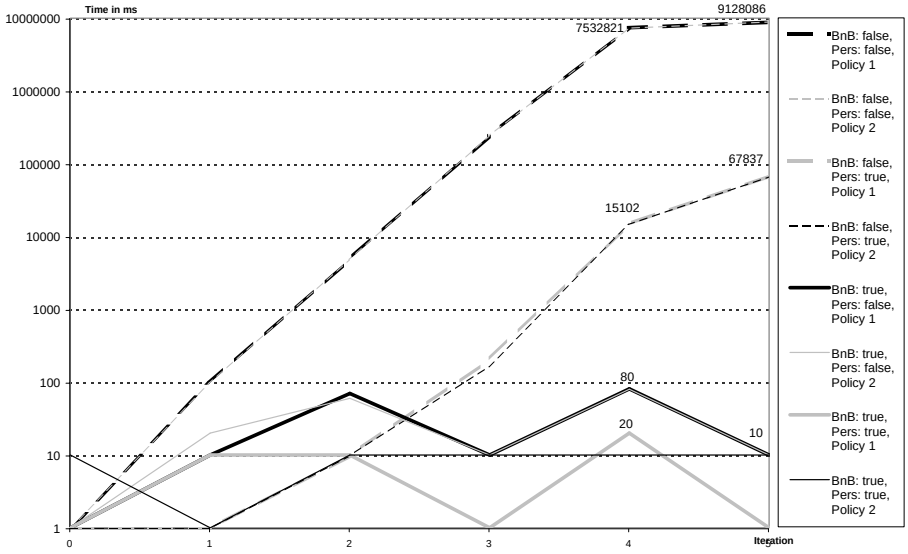


Fig. 23. *Execution time of each iteration during processing of case 1 by D^2CP.*

generate other candidates, and the longer it will take to test them with each other in order to only keep the minimal ones. So, when the number of candidates increases exponentially, so do the average time to compute a candidate, which implies a longer iteration time. And so, when the number of candidates is more little, then the average time to generate them is shorter.

– The addition of theorem 1 in the candidates generation step brings a great improvement in the average generation time. We can see, from figure 25, that the average time to compute one candidate evolves as before as the number of candidates, but at very lower level: the trend is the same (as without theorem 1), but the times are far more shorter. So theorem 1 greatly improves the candidates generation step. And, associated with BnB (i.e. when the number of candidates doesn't explode), theorem 1 keeps on greatly improving this average generation time (in fact, this is then so effective for this little case 1 than we cannot really analyze what it happens for this variants, because time measures are to low).

To sum up:

– The BnB technique limits the exponential explosion of the number of candidates. This implies a lower average candidate generation time.
– Theorem 1 also greatly lowers the average candidate generation time, whatever there is BnB or not.

Case 2 (figures 26, 27, 28 and 29) The study of case 2 is limited to variants of *computeBCov* with BnB, because executions without BnB made with D^2CP were stopped, due to a too long time (more than 12 hours).

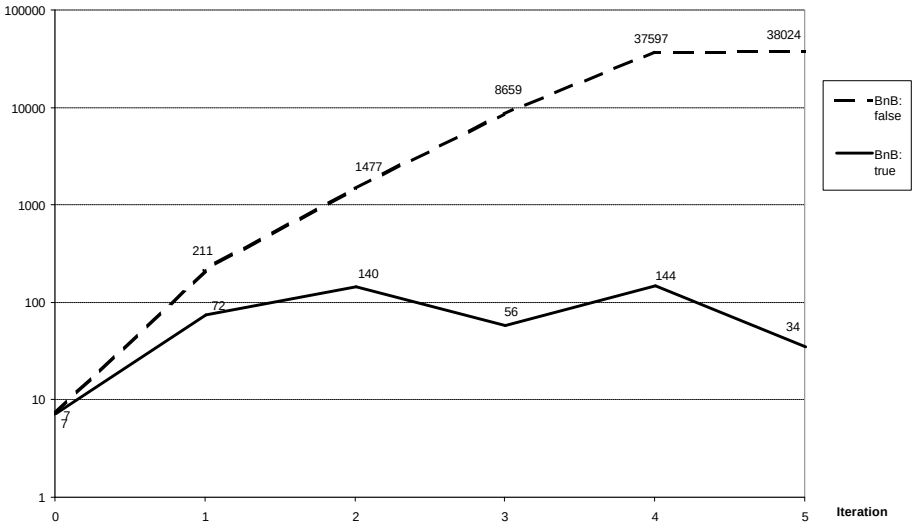


Fig. 24. *Number of generated minimal transversal candidates at each iteration during processing of case 1 by D^2CP.*

From figure 26, we can see 2 points.

The first point is that there are two types of algorithm behaviour: those with the candidates generation step based on theorem 1 (i.e. "Pers = true"), and those with the classical generation step. As it was mentioned for case 1, these two types of *computeBCov* variants evolves on the whole the same way, but the use of theorem 1 importantly lowers the time values. This is explained in figure 29 where we can see that theorem 1 implies a drastic fall in the time of the candidates generation step (knowing from figure 27 that the number of candidated is quite the same for the 4 variants of *computeBCov* studied here). This drastic fall is itself due to the average time for the generation of one candidate: from figure 28, we see that theorem 1 implies that this time is drastically lowered wrt to the classical generation.
Besides, we can remark from figure 29 that the times of the BnB steps are almost the same for each variant of *computeBCov*: as the number of generated candidates are also the same, this means that the BnB step is not influenced by theorem 1.

The second point is that, during the whole process, we can distinguish 3 phases:

- Phase 1: from iteration 0 to 4, *computeBCov* is beginning, and the time measures for these 5 iterations are not very significant.
- Phase 2: from iteration 5 to 23, the time of each iteration is bounded: with theorem 1, this time is nearly constant, and without theorem 1, there are oscillations between a lower and an upper bound.

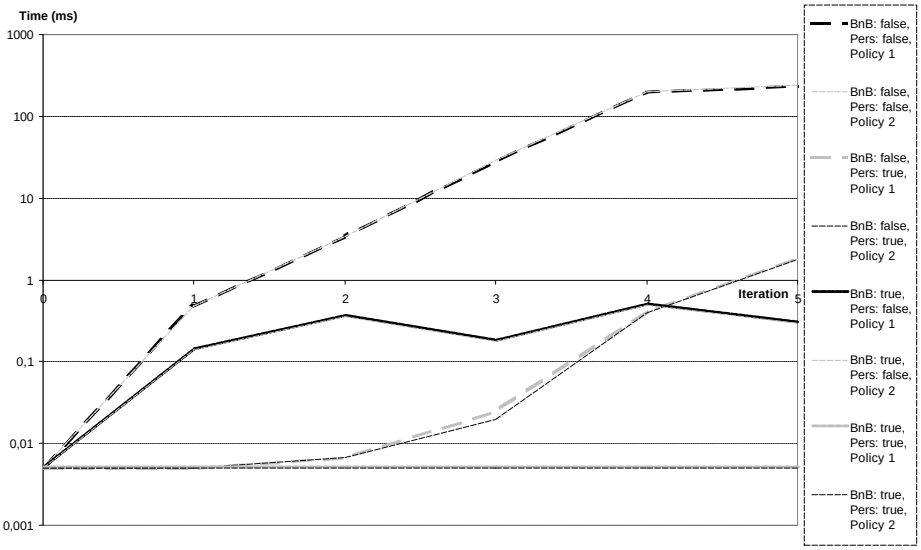


Fig. 25. *Average time to generate one minimal transversal candidate at each iteration during processing of case 1 by* D^2CP.

– Phase 3: from iteration 24 to 32, there is an explosion in the time values before a fall which indicates the end of *computeBCov*.

Let's explain phases 2 and 3.

During phase 2, without theorem 1, the iteration time oscillates between two bounded values, while, with theorem 1, this time is almost constant. Logically, the same thing happens in figure 28 for the average time of a candidate generation. As for case 1, this is due to the fact that during some iterations, many non minimal candidates are generated if theorem 1 is not used: their generation is time-consuming, so as their deletion. In section 3.3, we have theoretically shown that theorem 1 avoids generating such useless candidates. Here we show this optimization is really time-saving in a real execution.

During phase 3, without theorem 1, we can see that iterations 27 and 29 are very long. It is explained by the fact that, at iteration 27, there are not so many candidates (less than 4000 from figure 27), but the average time to generate them is very high (figure 28), and at iteration 29, the average generation time is not very high, but there are a huge number of candidates (24773). For variants with theorem 1, as the average time to generate a candidate is always very low, the iteration time mainly depends on the BnB time and so depends almost exclusively on the number of generated candidates.

To sum up:

– Theorem 1 implies a very time-saving candidates generation step. It follows that the iteration time is very close to the BnB time.

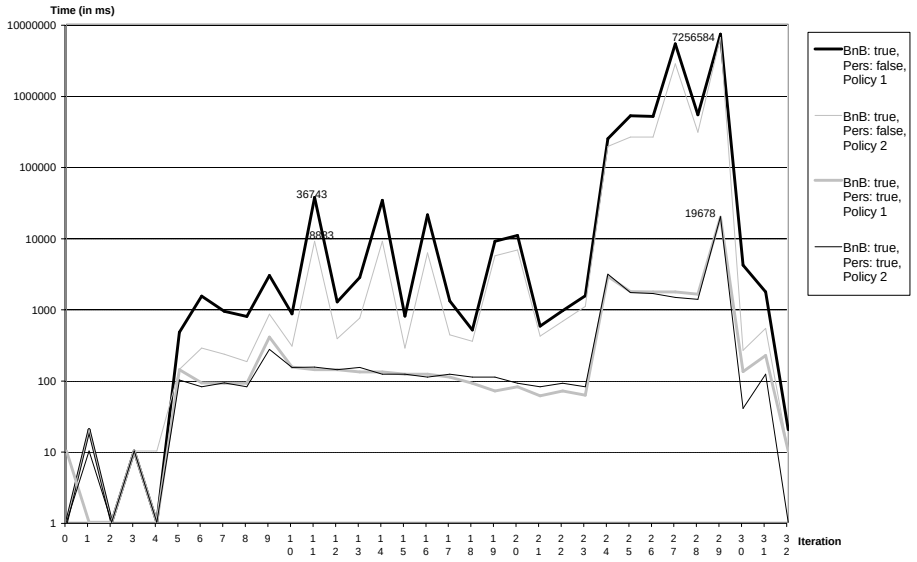


Fig. 26. *Execution time of each iteration during processing of case 2 by D^2CP.*

– The BnB is not influenced by the candidates generation step, but mainly by the number of generated candidates.

Case 3 (figures 30, 31, 32 and 33) From figure 30, we can see the same difference between variants with and variants without theorem 1. Moreover, we can see in this case the influence of policy 2: in figure 31, we can see that policy 2 implies a fall in the number of generated candidates from iteration 6 to the end. The effect is then the same as for case 1 and 2: the less candidates, the shorter the average time to generate one. We can see that in figure 32 also.

Figure 33 provides a good summary of what happens in case 3:

– The BnB is optimized by policy 2, but, as in cases 1 and 2, not influenced by theorem 1.
– Theorem 1 implies a large gain in efficiency.
– The variant of *computeBCov* with BnB, theorem 1 and policy 2 is very quick (a few seconds).

To conclude this section, figure 34 shows how the BnB, theorem 1 and policy 2 have an influence on the total execution time of *computeBCov*.

6 Conclusion

References

1. Data Engineering Bulletin: Special Issue on Infrastructure for Advanced E-Services. 24(1), IEEE Computer Society, 2001.

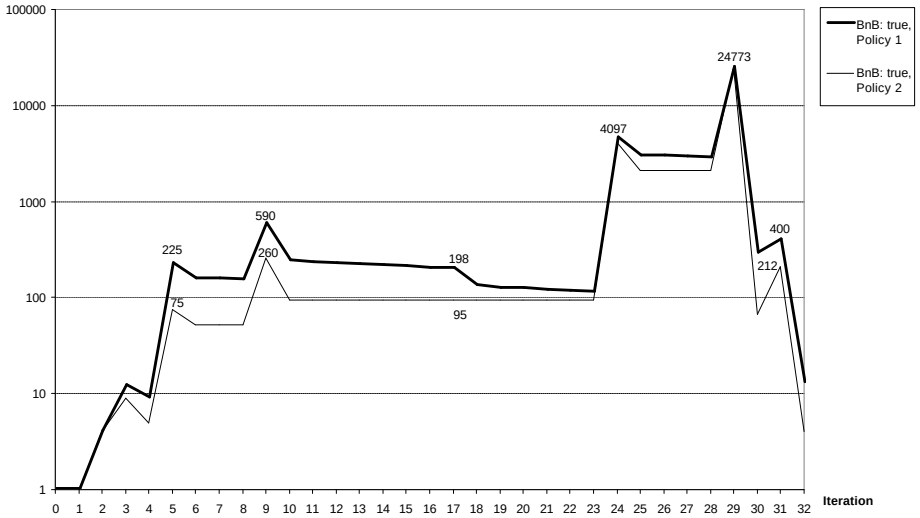


Fig. 27. *Number of generated minimal transversal candidates at each iteration during processing of case 2 by D^2CP.*

2. The VLDB Journal: Special Issue on E-Services. 10(1), Springer-Verlag Berlin Heidelberg, 2001.
3. C. Berge. *Hypergraphs*, volume 45 of *North Holland Mathematical Library*. Elsevier Science Publishers B.V. (North-Holland), 1989.
4. Fabio Casati and Ming-Chien Shan. Dynamic and adaptive composition of e-services. *Information Systems*, 26(3):143–163, May 2001.
5. Fabio Casati and Ming-Chien Shan. Models and Languages for Describing and Discovering E-Services. In *Proceedings of SIGMOD 2001, Santa Barbara, USA*, May 2001.
6. T. Eiter and G. Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278–1304, 1995.
7. Thomas Eiter and Georg Gottlob. Hypergraph transversal computation and related problems in logic and ai. In Sergio Flesca, Sergio Greco, Nicola Leone, and Giovambattista Ianni, editors, *Proceeding of the Logics in Artificial Intelligence, European Conference, JELIA, Cosenza, Italy, September, 23-26*, volume 2424 of *Lecture Notes in Computer Science*. Springer, 2002.
8. Thomas Eiter, Georg Gottlob, and Kazuhisa Makino. New results on monotone dualization and generating hypergraph transversals. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC-02)*, pages 14–22, New York, May 19–21 2002. ACM Press.
9. M.S. Hacid, A. Léger, C. Rey, and F. Toumani. Computing concept covers: a preliminary report. In *Proceedings of the International Workshop on Description Logics (DL'02), April 19 to April 21, 2002. Toulouse. France*, April 2002.
10. M.S. Hacid, A. Léger, C. Rey, and F. Toumani. Dynamic discovery of e-services in a knowledge representation and reasoning context. In *Proceedings of the 18th French conference on advanced databases (BDA), Paris, France*, October 2002.

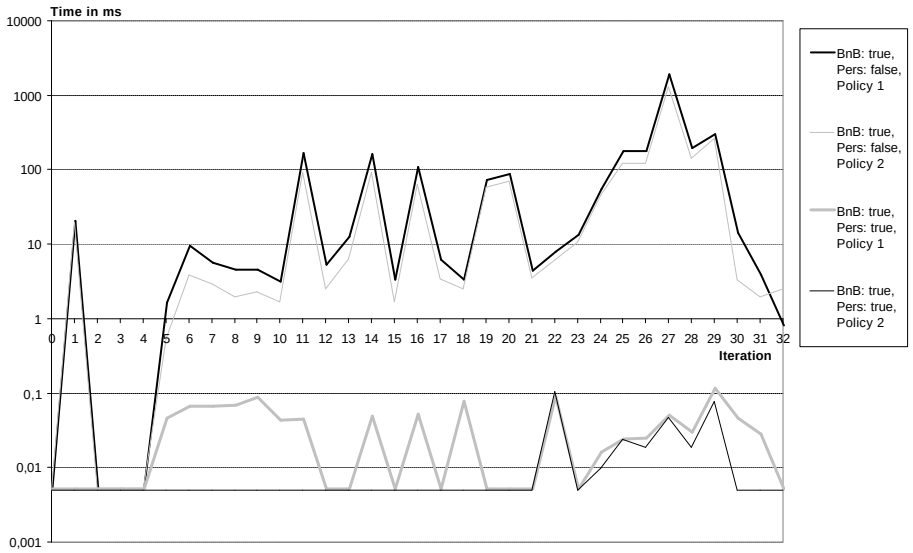


Fig. 28. *Average time to generate one minimal transversal candidate at each iteration during processing of case 2 by* D^2CP.

11. Dimitris J. Kavvadias and Elias C. Stavropoulos. Evaluation of an algorithm for the transversal hypergraph problem. *Lecture Notes in Computer Science*, 1668:72–85, 1999.
12. Stéphane Lopes, Jean-Marc Petit, and Lotfi Lakhal. Efficient discovery of functional dependencies and armstrong relations. In Carlo Zaniolo, Peter C. Lockemann, Marc H. Scholl, and Torsten Grust, editors, *Proceedings of the 7th International Conference on Extending Database Technology (EDBT 2000), Konstanz, Germany*, volume 1777 of *Lecture Notes in Computer Science*, pages 350–364. Springer, 2000.
13. H. Mannila and K-J Räihä. *The Design of Relational Databases*. Addison-Wesley, Wokingham, England, 1994.
14. Leonid Khachiyan Michael L. Fredman. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21(3):618–628, November 1996.
15. Catharine Wyss, Chris Giannella, and Edward Robertson. Fastfds: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances. In Yahiko Kambayashi, Werner Winiwarter, and Masatoshi Arikawa, editors, *Proceedings of the Data Warehousing and Knowledge Discovery, Third International Conference, DaWaK 2001, Munich, Germany, September 5-7*, volume 2114 of *Lecture Notes in Computer Science*. Springer, 2001.

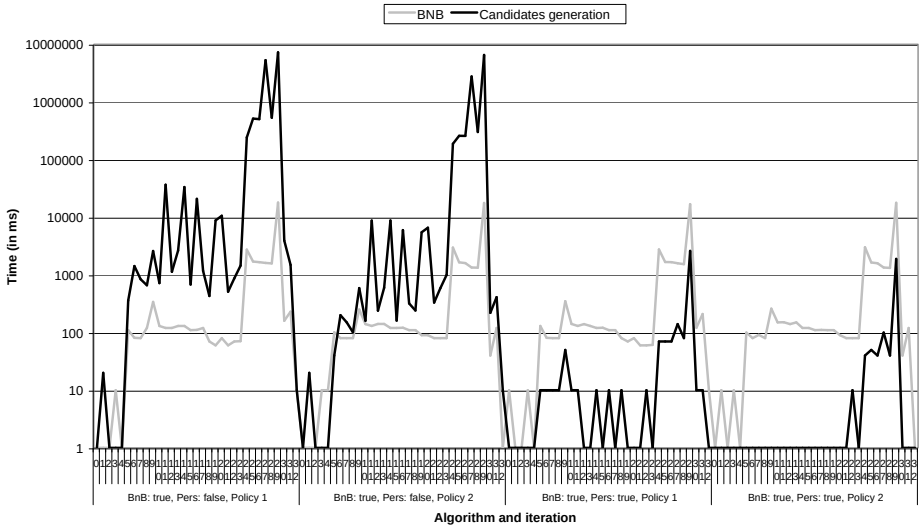


Fig. 29. *Executions times of BnB and candidates generation steps at each iteration and for each variant of computeBCov during processing of case 2 by* D^2CP.

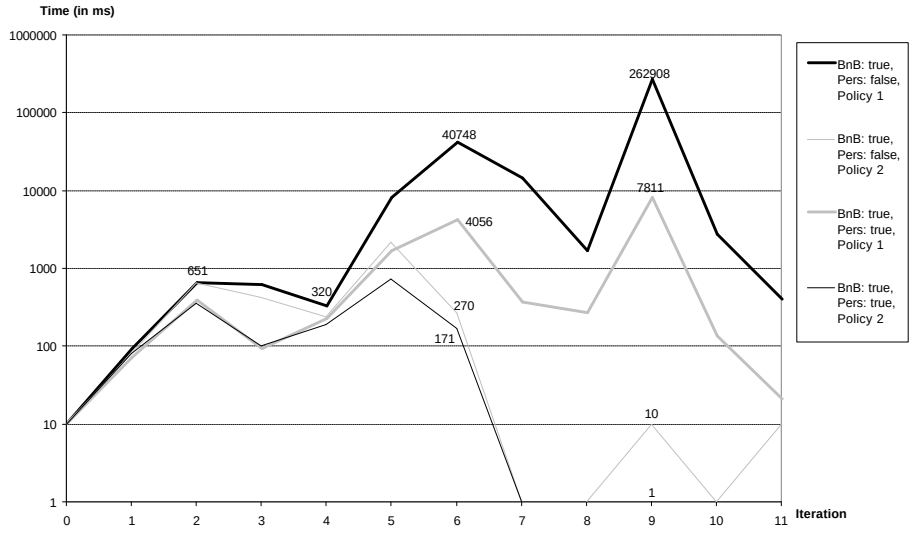


Fig. 30. *Execution time of each iteration during processing of case 3 by* D^2CP.

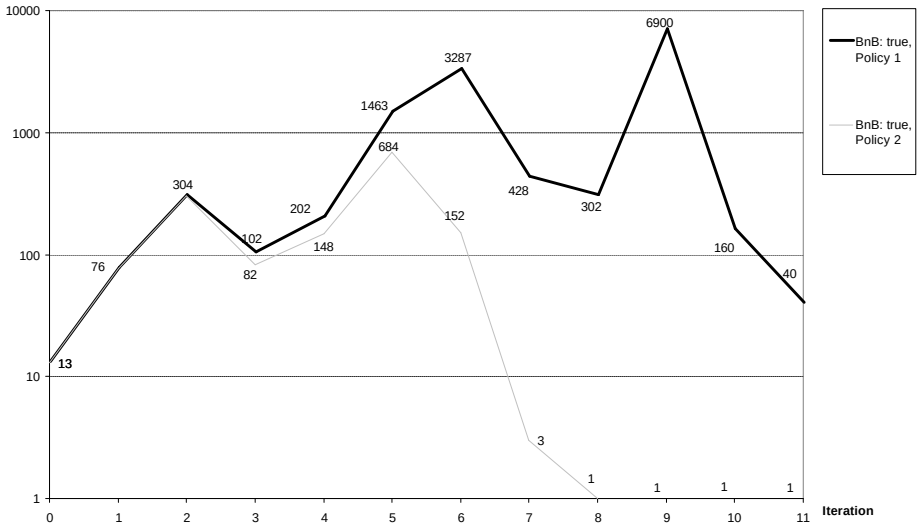


Fig. 31. *Number of generated minimal transversal candidates at each iteration during processing of case 3 by* D^2CP.

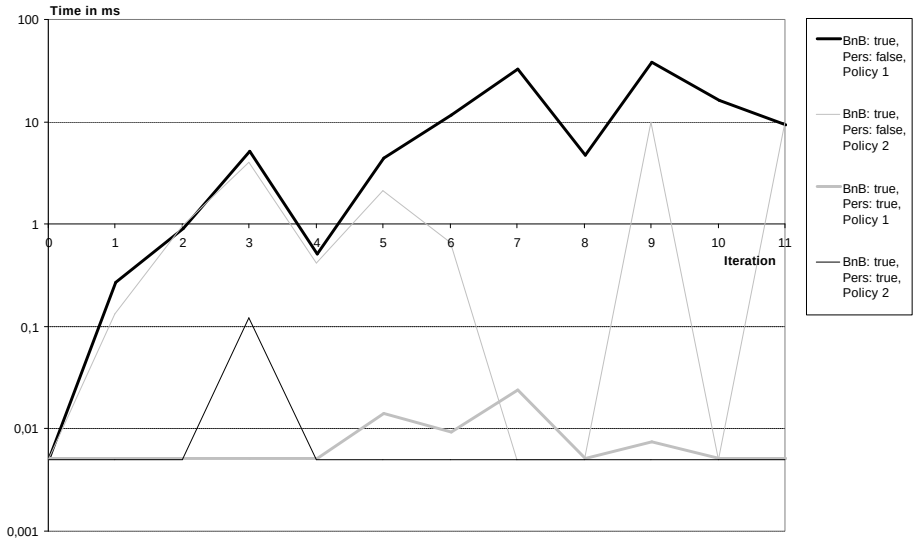


Fig. 32. *Average time to generate one minimal transversal candidate at each iteration during processing of case 3 by* D^2CP.

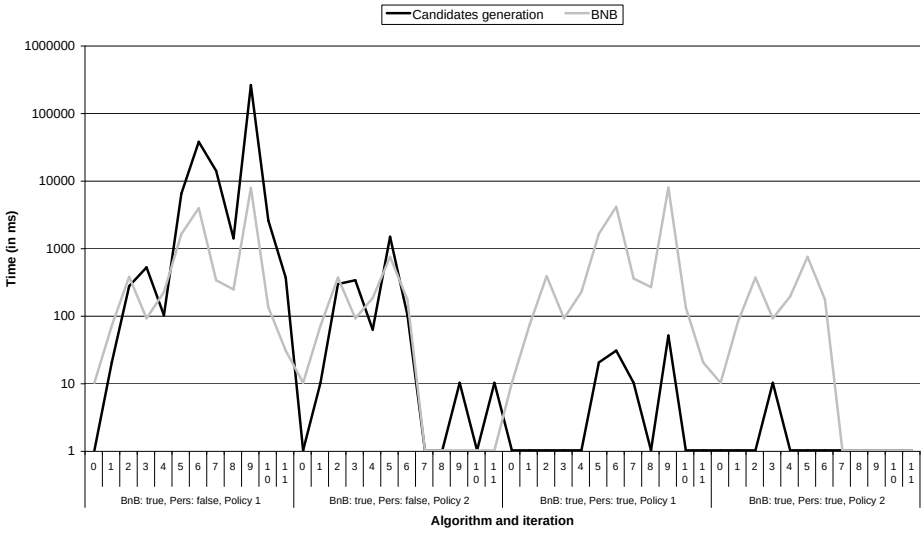


Fig. 33. *Executions times of BnB and candidates generation steps at each iteration and for each variant of computeBCov during processing of case 3 by* D^2CP.

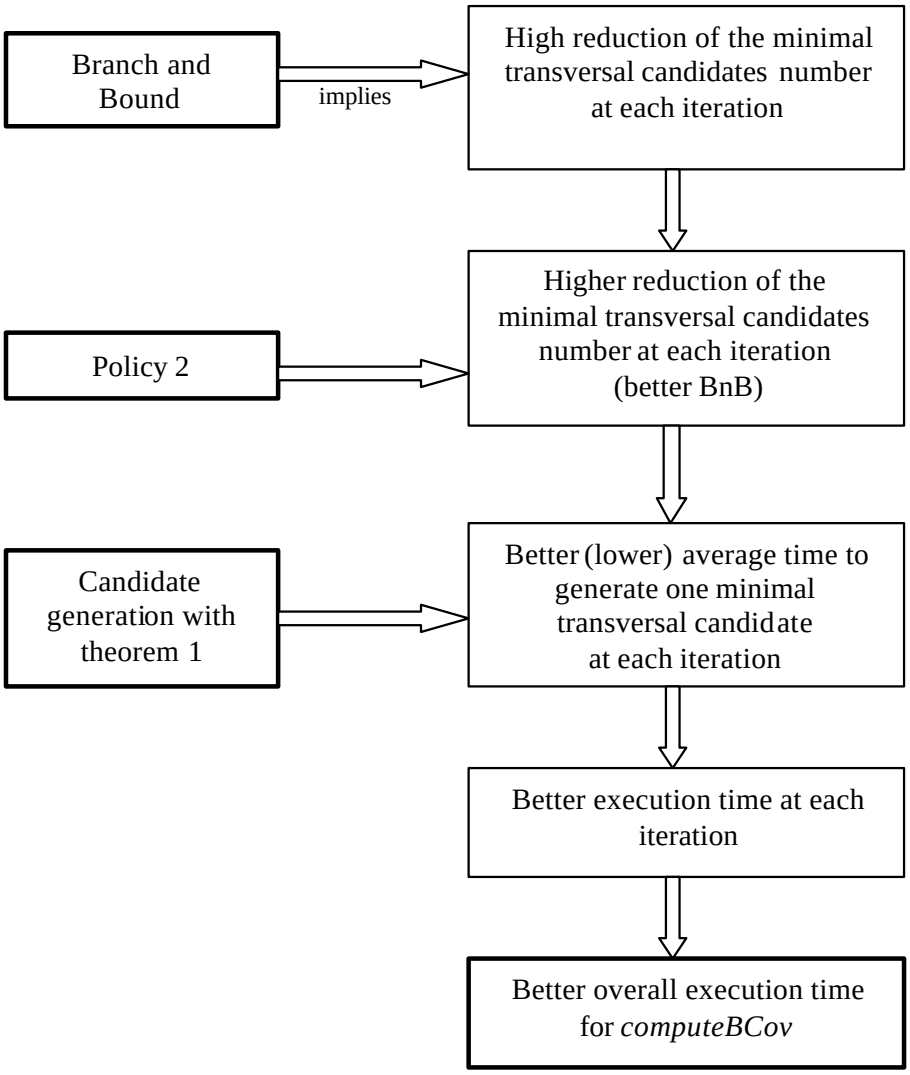


Fig. 34. *Summary of the main results of the study of cases 1, 2 and 3 concerning computeBCov.*