A Constraint-based Job-Shop Scheduling Model for Software Development Planning

Irene Barba, Carmelo Del Valle, and Diana Borrego

Dpto. Lenguajes y Sistemas Informáticos,
Universidad de Sevilla, Spain
{irenebr,carmelo,dianabn}@us.es

Abstract. This paper proposes a constraint-based model for the Job Shop Scheduling Problem to be solved using local search techniques. The model can be used to represent a multiple software process planning problem when the different (activities of) projects compete for limited staff. The main aspects of the model are: the use of integer variables which represent the relative order of the operations to be scheduled, and two global constraints, *alldifferent* and *increasing*, for ensuring feasibility. An interesting property of the model is that cycle detection in the schedules is implicit in the satisfaction of the constraints. In order to test the proposed model, a parameterized local search algorithm has been used, with a neighborhood similar to the Nowicki and Smutnicki one, which has been adapted in order to be suitable for the proposed model.

Key words: job shop scheduling, local search, constraint satisfaction problems, software development processes

1 Introduction

Software development has been modelled using a wide range of approaches. They vary according to the focus of the analysis and they address successfully the whole development process depending on how it is carried out. Many of the software management tools use temporal information and ignore in some ways the resources to be used, considering them unlimited, since they are based on PERT and CPM analysis. These may not be adequate in different situations, for example when smaller multiple projects are developed and project compete for limited staff [10]. A job shop approach, traditional in manufacturing, may represent an important aid since it can manage the interactions between projects and resources in a natural way and enables to consider minimizing different goals, as development time (makespan) and cost, while satisfying all the temporal and resource constraints.

With this aim, a job shop scheduling model is presented in this paper, so that it can represent a multiple software project to be planned. The equivalence of terms used from both areas is in such a way that jobs correspond to single software projects, and resources can represent each person or software development team working in the projects.

Constraint Programming (CP) has evolved in the last decade to a mature field due to, among others, the use of different generic and interchangeable procedures for inference and search, which can be used for solving different types of problems [2, 12]. Although a separation of models and algorithms is desirable for reusability issues, there is an influence between them that must be taken into account when a good behavior of the whole resulting method is pursued. Most models that have been used in CP have been tested using complete algorithms, and they are not equally suitable for other algorithmic approaches such as local search [16].

This paper proposes a Constraint Satisfaction Problem (CSP) model for the Job Shop Scheduling Problem (JSSP) to be solved using local search techniques, that is, it defines the variables which determine a solution, the related constraints of the problem involving those variables, and some possible neighborhoods. The problem has been solved by different authors using local search [8, 11, 15], but the novelty consist of the proposed model, based on including the ordering of the operations directly in the variables and constraints of the CSP, so that further definitions and developments of the main components of local search algorithms would take advantage of this representation.

For such techniques, a very important issue is the defined neighborhood, that is, the set of candidates to which the walk may continue from the current solution. For JSSP, one of the best methods was proposed by Nowicki and Smutnicki [11], whose neighborhood was more constrained than other previous approaches. An adaptation and an extension of this neighborhood are proposed in this work, in order to be suitable for the defined CSP model.

The rest of the paper is organized as follows. Section 2 presents a formulation of the JSSP. Section 3 includes the main ideas of local search algorithms. Section 4 describes the proposed model. Next, experimental results are shown and analyzed. Finally, Section 6 presents some conclusions and future work.

2 Problem Definition

The Job Shop Scheduling Problem [1, 8] may be formulated as follows. We are given a set of n jobs $J_1, ..., J_n$ and a set of m machines $M_1, ..., M_m$. Each job J_i consists of a sequence of n_i operations $op_{i1}, ..., op_{i,n_i}$, which must be processed in this order. Each operation op_{ij} must be processed for p_{ij} time units, without preemption, on machine $\mu_{ij} \in \{M_1, ..., M_m\}$. Each machine can only process one operation at a time. So, two types of constraints are defined, the precedence constraints among the operations of each job, and the resource constraints which force to select a permutation order of the operations that use each machine. These last constraints are the source of the NP-hard complexity of JSSP [4].

The typical objective, used in this work, is to find a feasible solution, minimizing the makespan, $C_{max} = max_{i=1..n}\{C_i\}$, where C_i is the completion time of job J_i, i.e. the completion time of op_{i,n_i}.

Figure 1 shows the disjunctive graph representation for a simple example of the problem, with $n = 3$ and $n_i = 3$, $\forall i$. In a disjunctive graph $G = (V, C, D)$, we have a set V of nodes which correspond to the operations of the job-shop,

a set C of directed arcs corresponding to the precedence constraints, and a set D of undirected arcs which connect the operations that use the same machine. A solution to the problem consists of fixing a direction for the undirected arcs, being feasible if there are no cycles.

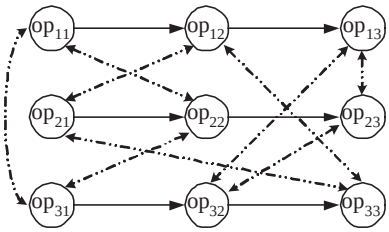


Fig. 1. A disjunctive graph for a job shop problem.

3 Constraint-Based Local Search

Most solving algorithms for CSPs proposed in the CP area are complete, and lastly local search is being considered as promising for solving large instances of complex problems [16], where complete algorithms fail. The constraints from the CSP model may be used for guarantying feasibility of the solutions explored, or even using their possible (degree of) violation as a guide for the search. Most of ideas associated to local search algorithms in other areas can be used for solving CSPs, or in our case, a Constraint Optimization Problem (COP).

Local search algorithms move iteratively through the set of feasible solutions. For those movements, a neighborhood for the current solution is determined in each iteration as a set of the solutions that can be selected as the next solution, and that can be obtained from the current solution with small changes. Depending on the method of choosing the next solution from neighborhood and the criteria for stopping the iterative sequence of movements, different algorithms can be defined [6]. In order to test the proposed model, we have used a basic tabu search algorithm [5] containing the main components that have been proved useful in local search, as described in Section 4.4.

4 Our Proposal

4.1 The CSP Model

A CSP is defined by a set of variables V, a set of domains of values for each variable D and a set of constraints that involve the variables C. Typical CSP models for the JSSP state the start times st_{ij} of the operations op_{ij} as the variables of the CSP [3], and the constraints are divided in two groups, precedence constraints

($st_{ij} + p_{ij} \leq st_{i,j+1}$) and resource constraints ($st_{ij} + p_{ij} \leq st_{kl} \vee st_{kl} + p_{kl} \leq st_{ij}$, op_{ij} and op_{kl} using the same machine). Our proposed CSP model is based on using the CSP variables to establish the execution orden of the operations of the JSSP, resulting in a simple model.

Let Π_J be a JSSP with a set J of n jobs, a set M of m machines, and a set O of $\#ops$ operations. The proposed model has the following components:

- Each operation op_{ij} is represented as an integer variable of the CSP v_{ij}, therefore the set of variables is $V = \{v_{ij}, 1 \leq i \leq n, 1 \leq j \leq n_i\}$.
- The domain of each variable v_{ij} is $D(v_{ij}) = [1..\#ops], \forall v_{ij} \in V$.
- The set C of constraints contains two types of items:
  1. **Precedence Constraints**: The value of each variable v_{ij} has to be less than the value of all the variables corresponding to the following operations in the same job: $v_{ij} < v_{ik}, \forall v_{ij}, v_{ik}$ such that $j < k$. In order to improve the efficiency and to obtain a clearer model, a new constraint (*increasing*) has been used between the operations of each job. It is defined on a sequence of variables $\{v_1, v_2, \ldots, v_n\}$ and it is equivalent to the satisfaction of the conditions $v_1 < v_2 < \ldots < v_n$.
  2. **Resource Constraints**: In order to satisfy that each machine can process only one operation at the same time, all the variable values are forced to be different from the others (*alldifferent* constraint is used), i.e., each solution is a permutation of the set $\{1, 2, \ldots, \#ops\}$.

An interesting property of the model, using the *increasing* and *alldifferent* constraints, is that cycle detection in the disjunctive graph is implicit in the satisfaction of the constraints, so no solution of the CSP will contain cycles.

A solution for the constrained problem, in which a value for each CSP variable is given, is a permutation of $1..\#ops$ variables and can be represented by an ordered sequence of operations S. With this sequence we associate an "earliest start schedule" by planning the operations in the order induced by the sequence, resulting in a JSSP solution. We denote $S(m)$ as the ordered sequence of operations that are executed on the machine m in the order fixed by the solution represented by S. Figure 2 shows a solution for the problem of Fig. 1. First, the value for each variable is shown and below is the corresponding solution S, where the position a in the sequence represents the value of the variable $S[a]$ ($v_{ij} = a \equiv S[a] = op_{ij}$). Also, the ordered sequences corresponding to each machine are shown. Finally, Fig. 2 shows a JSSP solution where all the arcs in the graph are directed according to the fixed order in S. Notice that there can be several solutions of the CSP problem that lead to the same schedule, for example the solution $S = \{op_{21}, op_{31}, op_{32}, op_{11}, op_{12}, op_{13}, op_{22}, op_{33}, op_{23}\}$ for the problem of Fig. 1 leads to the same schedule that the solution shown in Fig. 2.

From now, we will use $PM(v)$ and $SM(v)$ to refer to the predecessor and successor variables of v on its machine, and similarly $PJ(v)$ and $SJ(v)$ on its job. $PM(PM(v))$ is denoted by $PM_2(v)$ (the same for $SM(v)$) and so on. Moreover, we denote $m(v)$ as the machine in which the operation corresponding to the variable v has to be executed.

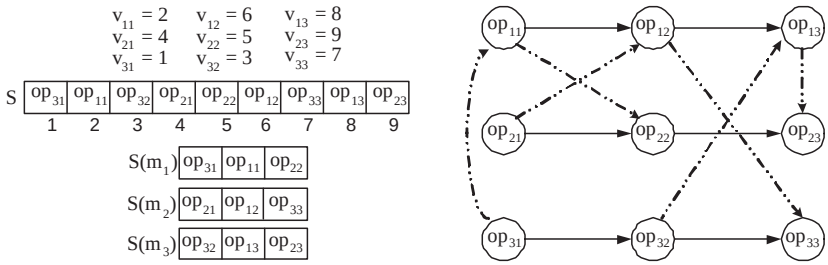


Fig. 2. Example of a feasible solution

4.2 Cycle Detection

A solution for the problem consists of establishing directions for the undirected arcs in the disjunctive graph (Section 2), being feasible if there not exists any cycle. A cycle for a solution in the disjunctive graph is a closed directed (simple) path, with no repeated vertices other than the starting and ending vertices.

We can see a cycle as a sequence of operations that contains two types of edges:

- Precedence edges: are fixed by the problem.
- Resource edges: are given by the decisions made to solve the problem.

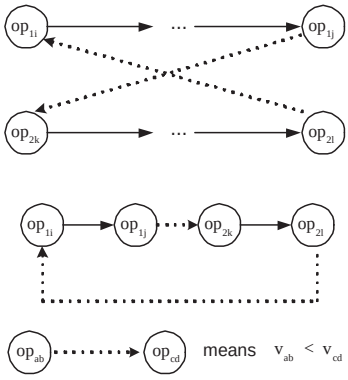


Fig. 3. A cycle in a disjunctive graph

All the possible cycles that can be formed in the graph involve, at least, two machines and four operations, two belonging to one job, and two belonging to another job, such as it is shown in the figure 3. In this figure it is possible to see a cycle formed by four operations, two belonging to J_1 (op_{1i} and op_{1j}) and two belonging to J_2 (op_{2k} and op_{2l}). In the sequence of operations appears, at least,

two precedences edges, that connect operations using different machines. All the operations that appear in the figure can be executed on different machines, so the machines involved in this cycle can be between 2 and 4. It is important to clarify that $op_{1,j}$ and $op_{2,k}$ do not have to be executed in the same machine (the same for $op_{1,i}$ and $op_{2,l}$).

It can be proven that any solution of the CSP, with the proposed model, will contain no cycles.

4.3 Neighborhoods

For JSSP, most of the successful approaches use neighborhood based on reversing critical operations (increasing their durations imply a larger makespan) that must be processed on the same machine. One of the best methods was proposed by Nowicki and Smutnicki [11], whose neighborhood was more constrained than other previous approaches. The movements allowed were to reverse two adjacent critical operations belonging to the same critical block (a sequence of critical operations on the same machine) so that one of them is not an internal operation in the block, excluding the swap between the first two operations of the first block when the second one is internal, and the swap between the last two operations of the last block when the first is internal.

We define a family of neighborhoods for the proposed model in which the basic idea is to make a swap between the values of two variables corresponding to operations of the same machine, i.e., between the relative order of those operations in a solution, trying to change the order of operations belonging to a critical path of a solution S ($CP(S)$ from now), based on the Nowicki and Smutnicki (NS from now) neighborhood.

For a variable v, $\sigma(v)$ is defined as the set of the variables w satisfying the following condition: **the swap between** v **and** w **in** S (denoted as $swap(v, w, S)$) **causes a swap between** v **and** $PM(v)$ **on** $m(v)$ **and this is the only swap caused on** $m(v)$. The variables w that meet this condition are those between $PM_2(v)$ (not included) and $PM(v)$ (included) in S. We can see that the swaps between v and variables that appear before $PJ(v)$ in S lead to unsatisfiable solutions. Then, $\sigma(v)$, when v is not the first in its job and has, at least, two predecessors on its machine, is defined as:

$$\sigma(v) = \{w \in V \mid max(PJ(v), PM_2(v)) < w \leq PM(v)\}$$

If $PJ(v)$ and $PM_2(v)$ do not exist, the outer lower bound is 0. On the other hand, if only one of them exists, the outer lower bound is established by it. Lastly, all the variables which have the smallest value on their machine (i.e., which are executed first) do not have any possible swaps ($\sigma = \emptyset$).

In Fig. 4 different cases of possible swaps are shown. In Fig. 4.a, $PJ(v)$ appears before $PM_2(v)$, then the outer lower bound of the range of possibilities is established by $PM_2(v)$. In Fig. 4.b, $PM_2(v)$ is before $PJ(v)$, then it is given by $PJ(v)$. In Fig. 4.c, $PJ(v)$ is after $PM(v)$, so no swap for v can be realized.

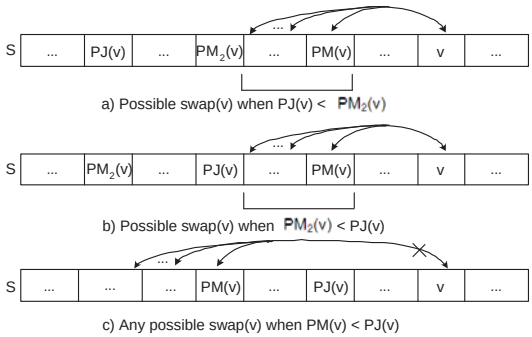


Fig. 4. Possible swaps for a variable v

In order to reduce and set a maximum number of neighbors for a solution, we define a parameter δ, as the maximum number of possible swaps for a variable v, from $PM(v)$ toward variables appearing before it in S. It must be noticed that δ has to be greater than 1 so that the algorithm can reach any possible solution, taking into account the proposed model. According to this parameter, the set of considered swaps for a variable, is defined as:

$$\sigma_\delta(v) = \{w \in V \mid max(PM(v) - \delta, PJ(v), PM_2(v)) < w \ \wedge \ w \leq PM(v)\}$$

A family of neighborhoods, $N^\delta_{1\lambda}$, depending on the possible variables to swap, has been defined. For $\lambda = 0$, the idea is to swap variables that are at the beginning or at the end of a critical block (CB from now, $CB(v)$ for the CB of a variable v), except the beginning of the first CB or the end of the last CB, similar to NS neighborhood. These variables are given by the set $V_0(S)$:

$$V_0(S) = \{v \in CP(S) \mid v = SM(first(CB(v))) \vee v = last(CB(v))\}$$

where $first(CB(v))$ and $last(CB(v))$ are the first and the last operations of $CB(v)$, respectively. $V_0(S)$ contains the possible variables to be swapped in N^δ_{10} ($N^\delta_{10} = \{swap(v, w, S) \mid v \in V_0(S) \wedge w \in \sigma_\delta(v)\}$).

Due to the proposed model and the tabu search, it is possible to reach a solution which an empty neighborhood. In order to overcome this problem and get more diversification during the search, other more general neighborhoods $N^\delta_{1\lambda}$, different from NS proposal, have been defined depending on a parameter λ. For $\lambda > 0$, it is allowed to swap internal variables of CBs, more internal as λ is increasing. The set of possible variables to swap, is now given by:

$$V_\lambda(S) = \{v \in CP(S) \mid (v = SM_{\lambda+1}(first(CB(v))) \vee v = PM_\lambda(last(CB(v))))$$
$$\wedge \ \lambda \leq \#CB(v)/2\}$$

Then, the neighborhood $N^\delta_{1\lambda}$ is defined as $N^\delta_{1\lambda} = \{swap(v, w, S) \mid v \in V_\lambda(S) \wedge w \in \sigma_\delta(v)\}$. In order to allow swaps between all the non-critical operations (belonging or not to CP), another neighborhood has been defined: $N^\delta_2 = \{swap(v, w, S) \mid v \in V \wedge w \in \sigma_\delta(v)\}$.

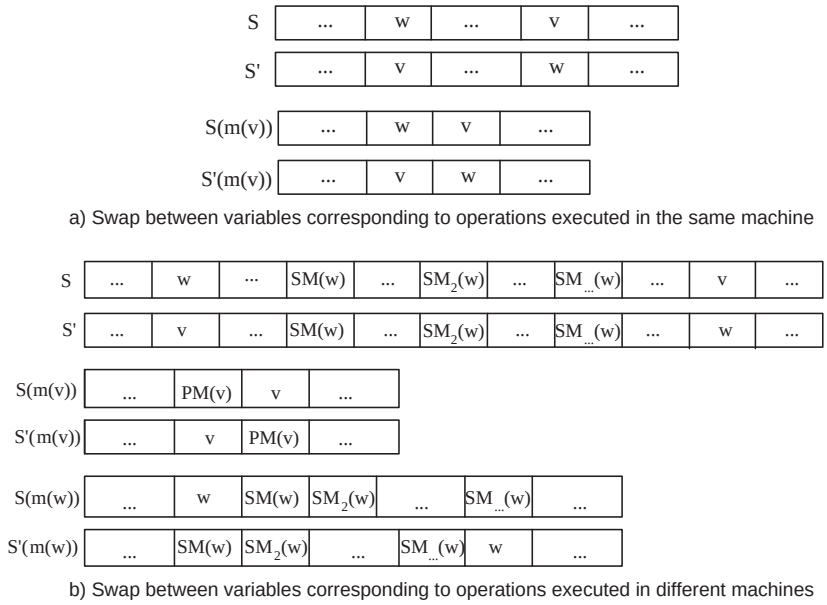


Fig. 5. Swap between variables

The swap between v and w has the following consequences:

1. Swap between the execution order of v and $PM(v)$, executed on the same machine, **which does not depend on** w (change in $m(v)$).
2. If $w \neq PM(v)$, i.e. $m(w) \neq m(v)$, other changes will be given. If $SM(w) < v$, then the execution orders of all the operations w' satisfying $m(w') = m(w)$ and $w < w' < v$ will be changed. Specifically, the relative order of all these operations are moved forward on their machine (change in $m(w)$).

According to this, two types of movements can be given. First, the swap between variables corresponding to operations executed on the same machine, only one swap in $S(m(v))$ is given (Fig. 5.a). Secondly, the swap between variables corresponding to operations executed on different machines, that leads to a swap in $S(m(v))$ and several swaps in $S(m(w))$, one for each direct or indirect successor on the machine of w that is between w and v in S (Fig. 5.b). In Fig. 5 the neighbor for S is referred as S'.

4.4 The Parameterized Algorithm

Considering the defined neighborhoods, a local search algorithm has been developed (Alg. 1). Although any initial solution can be used, the choice of better initial solutions usually allows to obtain better results, as it is found for the NS method [9]. In this way, for the experiments of the next section, we have

Algorithm 1: The parameterized local search algorithm

```
begin
    determine an initial solution S randomly
    best := S;
    for it := 1 to NIterations do
        if solution has not improved in last K iterations then
            choose a neighbor S' of best in N_2^δ randomly;
        else
            λ := 0;
            repeat
                determine set N_1λ^δ of non-tabu neighbors of S;
                if N_1λ^δ is not empty then
                    choose a best solution S' in N_1λ^δ;
                else
                    λ := λ + 1;
            until S' has been selected or λ > maxBlockSize(S)/2 ;
            if S' has not been selected (all of N_1λ^δ are empty) then
                choose a neighbor S' of S in N_2^δ randomly;
        S := S';
        if S improves best then
            best := S;
end
```

used the INSA algorithm [11]. As indicated in Subsection 4.1, a schedule can be represented by different solutions of the model. Thereby, for selecting the actual initial solution a random procedure is used from the schedule obtained by the INSA algorithm.

According to the evolution of the search, different neighborhoods are used in order to select the next movement, which will correspond to a feasible solution. In each iteration, a movement to the best neighbor of $N^{\delta}_{1,0}$ is attempted ($\lambda = 0$), but, if the neighborhood is empty or all their members are in the tabu list, a more extended neighborhood is searched, by increasing λ. If λ reaches the allowed maximum value without finding a suitable next solution to visit, the more general neighborhood N^{δ}_2 is used, and now the neighbor is selected randomly. N^{δ}_2 is also used when the algorithm has not found a better solution for a number K of iterations. In this case, the algorithm returns to the best solution found so far.

Besides that, most of the computational cost of local search algorithms are due to the evaluation of neighbors. In order to reduce its amount, several approaches have been proposed, such as that of Taillard [14], which evaluates the neighbors using a lower bound estimation of the makespan in constant time, instead of calculating it exactly. In the proposed algorithm, the selection of candidates is made in two steps. First, the best swap between two critical operations is selected using the Taillard estimation of the expected makespan . After that,

a variable is selected from the δ possibilities, choosing the one with the greatest improvement in its slack because of the change.

5 Experimental Results

ILOG JSolver [7] has been used for implementing the Algorithm 1, and for managing the constraints of the problem. As stated before, the algorithm has several parameters, δ, K (maximum number of iterations without improving the solution), and the tabu list size (TLS), that may affect its behavior, and its tuning represents a non-trivial problem. Since the main interest of this work is not the competitiveness of the algorithm proposed, but the CSP model which is defined, a scenario for some comparative results was chosen, in which the algorithm would be executed for a fixed number of 10000 iterations and , which were selected randomly from the results of the INSA algorithm. For such situation, the value of K was chosen to be 1000. For selecting δ and TLS, the algorithm was run on a reduced set of instances for δ from 2 to 5 and from TLS from 5 to 10. The best results on the minimal and average makespan of the best solution after 10000 iterations were found for $\delta = 2$ and $TLS = 6$. The best results for $\delta = 2$ can be explained by the fact that for higher values of δ, there is more probability for finding a variable w such that the swap between v and w will be feasible, which would enforce the diversification strategy too much.

Table 1 shows the results of the algorithm for a larger set of JSSP benchmarks, taken from the OR-library, and some harder instances from Taillard [13]. For each JSSP instance, the table shows some statistics about the algorithm used in this work: the relative error of the best solution from the 100 restarts ($BRE\%$) with respect to the best known solution (UB, which is not proved optimal for the values indicated by *), the mean relative error ($MRE\%$) and the standard deviation of relative error ($SDRE\%$). Also, the mean computational time for running the algorithm is given (RT). As reference, the results obtained by the NS algorithm is shown in two situations: in the original form, that takes into account several factors, and after 10000 iterations. As expected, the algorithm is not fully competitive (as well as it has been developed in Java, many of its components are not optimized) with that of Nowicki and Smutnicki, considered as one of the best methods for solving the JSSP. Instead, the results shown must be taken as a reference for further improvements of the algorithm or for different approaches that can use the model.

6 Conclusions and Future Work

This paper proposes a CSP model for the Job Shop Scheduling Problem to be solved using local search techniques. The model can be used to represent a multiple software process planning problem when the different (activities of) projects compete for limited staff. The main aspects of the model are the use of integer variables which represent the relative order of the operations to be scheduled and two types of global constraints for ensuring feasibility. Also, a

Table 1. Results on a set of JSS instances

				Proposed Model				NS			
Instance	n	m	UB	$BRE\%$	$MRE\%$	$SDRE\%$	RT	$BRE\%$	RT	$BRE\%_{10^4}$	RT_{10^4}
FT10	10	10	930	2.25	3.95	0.96	8.72	0	0.68	0	0.25
ABZ7	20	15	656	9.90	13.98	2.16	64.40	2.28	4.62	3.20	0.84
LA02	10	5	655	0.45	3.80	1.83	3.02	0	0.10	0	0.11
LA19	10	10	842	2.49	6.25	1.82	10.52	0.11	0.83	0.11	0.35
LA21	15	10	1046	5.16	8.23	1.05	18.76	0.86	0.86	0.86	0.42
LA24	15	10	935	3.85	6.56	1.13	18.72	1.39	1.33	1.50	0.45
LA25	15	10	977	7.26	11.24	1.84	20.65	1.12	1.39	2.04	0.45
LA27	20	10	1235	6.96	12.07	2.03	30.73	1.94	1.27	1.94	0.51
LA29	20	10	1152	8.42	12.57	2.22	33.24	3.13	3.40	4.51	0.48
LA36	15	15	1268	7.09	11.42	1.25	38.61	0.79	3.66	2.76	0.62
LA37	15	15	1397	9.09	14.59	2.30	41.80	1.50	2.74	3.29	0.78
LA38	15	15	1196	5.85	8.09	0.99	41.72	1.84	2.75	2.59	0.65
LA39	15	15	1233	7.94	10.44	0.90	41.04	0.89	3.50	1.62	0.79
LA40	15	15	1222	7.03	10.28	1.02	36.52	1.64	2.40	2.13	0.62
TA02	15	15	1244	6.35	10.49	1.64	36.47	2.73	2.83	2.73	0.70
TA18	20	15	1396*	12.60	15.25	1.32	57.82	3.65	4.64	5.73	0.97
TA26	20	20	1645*	9.36	13.14	1.34	93.66	3.10	10.64	3.28	1.58
TA32	30	15	1795*	14.20	17.84	1.30	116.93	3.12	18.36	6.85	1.44

neighborhood for this model has been defined based on an adaptation of Nowicki and Smutnicki one. The main focus is not on the competitiveness of the algorithm which is proposed, but in the definition of the CSP model.

As future work, the algorithm and neighborhood should be improved for solving more efficiently the JSSP. Also, we think that the proposed model can be adapted for other similar sequencing problems in a direct way.

On the other hand, it is intended to extend the software development process to other (more generic or specific) models and to adapt the corresponding (planning and/or) scheduling models and solving algorithms.

Acknowledgments. This work has been partially supported by the Spanish Ministerio de Educación y Ciencia through a coordinated research project(grant DIP2006-15476-C02-01) and Feder (ERDF).

References

1. P. Brucker and S. Hnust, *Complex Scheduling*, Springer, 2006.
2. R. Dechter, *Constraint Processing*, Morgan Kaufmann Publishers, 2003.
3. R. Dechter, I. Meiri, and J. Pearl, 'Temporal constraint networks', *Artificial Intelligence*, **49**, 61–95, (1991).

4. M. R. Garey, D. S. Johnson, and R. Sethi, 'The complexity of flowshop and jobshop scheduling', *Math. Oper. Res.*, **1(2)**, 117–129, (1976).
5. F. Glover and M. Laguna, *Tabu Search*, Blackwell Scientific Publishing, Oxford, England, 1993.
6. H. H. Hoos and T. Stutzle, *Stochastic Local Search. Foundations and Applications*, Morgan Kaufmann, 2005.
7. ILOG, 'Ilog JSolver', (2003).
8. A. Jain and S. Meeran, 'Deterministic job-shop scheduling: Past, present, and future', *European Journal of Operational Research*, **113 (2)**, 390–434, (1999).
9. A. Jain, B. Rangaswamy, and S. Meeran, 'New and Stronger Job-Shop Neighbourhoods: A Focus on the Method of Nowicki and Smutnicki (1996)', *Journal of Heuristics*, **6**, 457-480, (2000).
10. C. A. Leonhard, and J. S. Davis, 'Job-Shop Development Model: A Case Study', *IEEE Software*, **12 (2)**, 86-92, (1995).
11. E. Nowicki and C. Smutnicki, 'A fast taboo search algorithm for the job-shop problem', *Management Science*, **42(6)**, 797813, (1996).
12. F. Rossi, P. van Beek, and T. Walsh, *Handbook of Constraint Programming*, Elsevier, 2006.
13. E. Taillard, 'Benchmarks for basic scheduling problems', *European Journal of Operational Research*, **64**, 278-285, (1993).
14. E. Taillard, 'Parallel Taboo Search Techniques for the Job-Shop Scheduling Problem', *ORSA Journal on Computing*, **16(2)**, 108-117, (1994).
15. R. Vaessens, E. Aarts, and J. Lenstra, 'Job-shop scheduling by local search', *INFORMS Journal on Computing*, **8**, 302–317, (1994).
16. P. Van Hentenryck and L. Michel, *Constraint-Based Local Search*, The MIT Press, 2005.

Generación de casos de prueba para composiciones de servicios web utilizando Búsqueda Dispersa

Raquel Blanco, José García-Fanjul, Javier Tuya

Departamento de Informática, Universidad de Oviedo
Campus de Viesques s/n, Gijón, Asturias, 33204, España
{rblanco, jgfanjul, tuya}@uniovi.es

Resumen. Una parte que conlleva una labor intensiva dentro de la prueba del software es la generación de casos de prueba. El coste de esta tarea puede ser reducido mediante el uso de técnicas que permitan su automatización. En este trabajo se presenta un enfoque basado en la técnica metaheurística Búsqueda Dispersa para la generación automática de casos de prueba de procesos de negocio especificados en BPEL. Como criterio de suficiente se emplea el criterio de cobertura de transiciones.

Palabras Clave: Prueba software, generación automática de casos de prueba, composiciones de servicios web, Búsqueda Dispersa, cobertura de transiciones.

1 Introducción

Una parte que conlleva una labor intensiva dentro de la prueba del software es la generación de casos de prueba. El coste de esta tarea puede ser reducido mediante el uso de técnicas que permitan su automatización. En arquitecturas orientadas a servicios el despliegue del software como una serie de servicios tiene como objetivo, a corto o medio plazo, que dichos servicios sean invocados desde otros programas software o servicios. Por ello, el uso de técnicas de prueba automáticas y bien establecidas es esencial para asegurar la calidad del servicio desarrollado y también para facilitar las pruebas de regresión.

La búsqueda de una solución óptima en el problema de la generación de casos de prueba tiene un gran coste computacional, y por este motivo las técnicas que permiten llevar a cabo la automatización tratan de obtener una solución próxima a la óptima. Como consecuencia, estas técnicas han atraído un creciente interés por parte de muchos investigadores en los últimos años. Por otro lado, la naturaleza de los problemas de Ingeniería del Software es ideal para la aplicación de técnicas metaheurísticas, como se muestra en el trabajo de Harman y Jones [17]. Una de esos problemas es la prueba del software, la cual es tratada como un problema de búsqueda u optimización, como se muestra en varios trabajos que llevan a cabo una revisión de la literatura [22, 23]. Además las técnicas de búsqueda metaheurísticas han obtenidos buenos resultados en la generación de casos de prueba [22].

La técnica metaheurística más ampliamente utilizada en este campo son los Algoritmos Genéticos [1, 2, 3, 9, 10, 15, 20, 25, 30, 31, 32]. Otras técnicas que también

han sido empleadas son el Recocido Simulado [21, 29, 32], Programación Genética [28], Búsqueda Tabú [11, 12], Repulsión Simulada [7], Algoritmos Evolutivos [8, 24], *Hill Climbing* [20], Estrategias Evolutivas [2], Algoritmos de Estimación de Distribuciones [27] o Búsqueda Dispersa [4, 5, 27].

Respecto a la generación de casos de prueba para el proceso de negocio de las composiciones de servicios web, se han empleado comprobadores de modelos [14, 18] o Redes de Petri [13]. La aplicación de las técnicas de búsqueda metaheurísticas es muy reciente y el único trabajo que emplea una de estas técnicas, llamada Búsqueda Dispersa, a la generación de casos de prueba para procesos de negocio especificados en BPEL es [6].

En este trabajo se propone el uso de la técnica metaheurística Búsqueda Dispersa [16, 19] para la generación automática de casos de prueba para composiciones de servicios especificadas en BPEL. El enfoque presentado en este trabajo es una evolución del algoritmo TCSS-LS descrito en [5], el cual genera casos de prueba para el criterio de cobertura de ramas para programas escritos en lenguaje C, y se basa en el presentado en [6].

El resto del artículo se organiza de la siguiente forma. La próxima sección describe brevemente la especificación de composiciones de servicios empleando BPEL. La sección 3 detalla el enfoque basado en Búsqueda Dispersa para la generación automática de casos de prueba. En la sección 4 se presentan los resultados preliminares y por último en la sección 5 se presentan las conclusiones de este trabajo.

2 Especificación de composiciones de servicios web utilizando BPEL

Las especificaciones BPEL representan el comportamiento del proceso de negocio basado en composiciones de servicios web. Estas especificaciones son documentos XML formados por dos partes principales: declaraciones y la especificación del propio proceso de negocio.

En la parte de declaraciones se identifican los interfaces entre el proceso BPEL y los participantes en el proceso de negocio, es decir, los servicios y las agrupaciones de operaciones que se pueden invocar dentro de dichos interfaces. Otros elementos incluidos en esta parte son las variables, donde se puede llevar a cabo un almacenamiento intermedio de valores.

La especificación del proceso de negocio consiste en un conjunto de actividades que pueden ser ejecutadas. Estas actividades pueden ser básicas o estructuradas. Entre las actividades básicas se encuentran aquellas que permiten al proceso de negocio invocar servicios web (actividad *invoke*), recibir invocaciones de los mismos (actividad *receive*) o actualizar el valor de las variables (actividad *assign*). Las actividades estructuradas indican el orden y bajo qué condiciones se ejecutan una serie de actividades. Por ejemplo: una actividad *sequence* establece un orden secuencial y una actividad *while* fuerza la repetición de la ejecución de un conjunto de actividades hasta que se deje de cumplir una determinada condición. Una actividad estructurada que no es tan común en otros lenguajes es la actividad *flow*, la cual indica que un conjunto de

actividades se ejecutan de forma concurrentes, y por tanto un *flow* se completa cuando todas sus actividades han finalizado.

3 Generación de casos de prueba para procesos de negocio BPEL utilizando Búsqueda Dispersa

En esta sección se explica la adaptación de la técnica Búsqueda Dispersa a la generación de casos de prueba para procesos de negocios especificados en BPEL, utilizando el criterio de cobertura de transiciones. En la sección 3.1 se presentan los aspectos generales del enfoque basado en Búsqueda Dispersa, denominado TCSS-LS. Las secciones 3.2 y 3.3 muestran el proceso de búsqueda de nuevos casos de prueba.

3.1 Planteamiento del problema

El objetivo de este trabajo es probar composiciones de servicios web especificadas en BPEL. Las variables de entrada del proceso de negocio son las variables recibidas por parte del proceso de negocio desde los servicios web (llamados participantes en BPEL) que interactúan en la especificación BPEL. La definición de un caso de prueba incluye los valores de las variables de entrada y las transiciones ejecutadas dentro del proceso de negocio.

La especificación BPEL no incluye directamente información sobre el comportamiento de los servicios web que participan en el proceso de negocio, por lo tanto se debe construir un modelo para cada participante, basado en su interface con el proceso de negocio.

El proceso de negocio especificado en BPEL se representa por medio de un grafo de estados, donde los nodos representan los estados del proceso de negocio y los arcos representan el cambio de estado desde un nodo i hasta un nodo j cuando la decisión asociada a dicho arco es cierta, es decir, un arco representa una transición en el proceso de negocio. Mediante el uso del grafo de estados es posible determinar las transiciones que han sido cubiertas por los casos de prueba generados, ya que el proceso de negocio ha sido instrumentado para conocer el camino seguido.

En la Fig. 1 se muestra el grafo de estados que representa el proceso de negocio de la composición “loan approval”, publicado en el estándar [26], cuyo objetivo es concluir si una petición para obtener un préstamo será aprobado o no. Para ello recibe una petición desde un servicio participante llamado “customer” e invoca a otros dos servicios participantes: “assesor” y “approver”. El servicio “assesor” mide el riesgo asociado a las peticiones de pequeña cantidad, mientras que el servicio “approver” evalúa las peticiones que o han sido hechas por una gran cantidad de dinero o han sido evaluadas como arriesgadas por el “assesor”. Cada transición del proceso de negocio se numera como Tk. Nótese que algunas transiciones del grafo no son numeradas, puesto que son cubiertas por los casos de prueba que cubren otras transiciones o conjuntos de transiciones. Por ejemplo, si un caso de prueba cubre las transiciones T2 y T4, entonces la transición desde el estado “invoke approver” hasta el estado final también será cubierta.

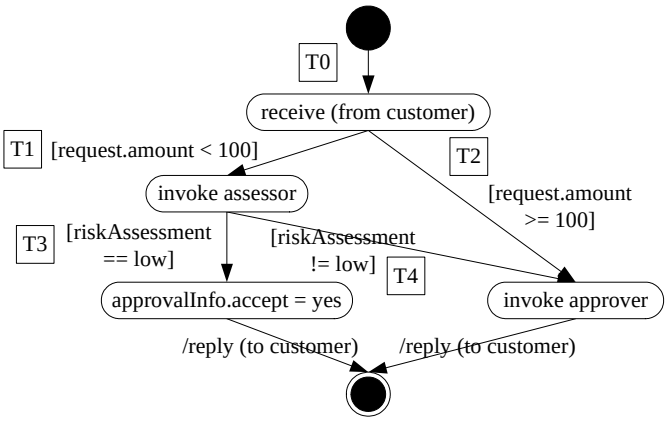


Fig. 1. Grafo de estado del servicio "loan approval"

El objetivo de TCSS-LS es generar casos de prueba que permitan cubrir todas las transiciones del proceso de negocio. Este objetivo general se puede dividir en subobjetivos, consistiendo cada uno de ellos en encontrar casos de prueba que alcancen un determinado arco (transición) Tk del grafo de estados.

Para alcanzar dichos subobjetivos, las transiciones del grafo de estados almacenan determinada información durante el proceso de generación de casos de prueba, la cual permite conocer las transiciones que han sido cubiertas y es utilizada para avanzar en el proceso de búsqueda. Cada transición almacena esta información en su propio conjunto de soluciones, llamado Conjunto de Referencia. A diferencia del algoritmo original de Búsqueda Dispersa, el enfoque presentado en este trabajo maneja varios Conjuntos de Referencia. Cada Conjunto de Referencia se denomina S_k, donde k es el número de la transición, y está constituido por B_k elementos $T_k^c = < \overline{x}_k^c, p_k^c, fb_k^c, fc_k^c >$, $c \in \{1..B_k\}$, donde:

- $\overline{x}_k^c$ es una solución, es decir, un caso que prueba que alcanza la transición Tk. Cada solución $\overline{x}_k^c$ consiste en un conjunto de valores para las variables de entrada ($\overline{x}1$, $\overline{x}2$,..., $\overline{x}n$) del proceso de negocio bajo prueba que satisfagan las condiciones de las transiciones previas a la transición Tk en el camino seguido. Cada variable de entrada se representa por un vector, ya que cada servicio web puede ser invocado varias veces y cada invocación proporciona un valor independiente.
- p_k^c es el camino cubierto por la solución (caso de prueba), es decir, la secuencia de transiciones alcanzadas del grafo de estados.
- fb_k^c es la distancia al nodo hermano. Esta distancia mide lo cerca que se encuentra la solución de cubrir la transición hermana.
- fc_k^c es la distancia a la siguiente transición que no ha sido cubierta por la solución. Esta distancia indica lo cerca que se encuentra la solución de cubrir dicha transición.

El procedimiento seguido para calcular el tamaño máximo B_k de de los conjuntos de soluciones de cada transición Tk (S_k) y el cálculo de las distancias se puede consultar en [5].

TCSS-LS tratará de hacer los conjuntos de soluciones lo más diversos posibles utilizando una función de diversidad. De este modo trata de explorar un amplio espacio de búsqueda con el fin de encontrar soluciones que puedan cubrir diferentes transiciones de la composición. La diversidad de una solución de un conjunto S_k es una medida relacionada con el camino cubierto por todas las soluciones del conjunto.

3.2 Proceso de búsqueda

El objetivo de TCSS-LS es obtener la máxima cobertura de transiciones, es decir, encontrar soluciones que permitan cubrir todas las transiciones del grafo de estados. Dado que estas soluciones están almacenadas en las transiciones, el objetivo es por tanto que todas las transiciones tengan al menos un elemento en su conjunto S_k. Sin embargo, este objetivo puede no ser alcanzado cuando la composición bajo prueba tiene transiciones inalcanzables. Por ello, TCSS-LS también finaliza su ejecución cuando se genera un número máximo de casos de prueba. Inicialmente los conjuntos S_k están vacíos y se irán rellenando en las sucesivas iteraciones.

La Fig. 2 muestra el esquema del proceso de búsqueda seguido por TCSS-LS. Este proceso comienza generando soluciones aleatorias que serán almacenadas en el Conjunto de Referencia de la transición T0 (S_0). El modelo de la composición de servicios se ejecuta con cada solución, lo que provoca la actualización de los conjuntos S_k de las transiciones alcanzadas. A continuación comienzan las iteraciones del proceso de búsqueda, donde TCSS-LS selecciona una transición (transición en evaluación) para crear los subconjuntos de soluciones a partir de su Conjunto de Referencia, los cuales son utilizados por las reglas de combinación para generar las nuevas soluciones. Estas soluciones, que pueden ser mejoradas, también son ejecutadas en el modelo de la composición para actualizar los conjuntos S_k de las transiciones alcanzadas, cerrando así el ciclo de ejecución.

Cada vez que el modelo de la composición de servicios va a ser ejecutado, los participantes deben ser configurados con los valores de las variables que retornan al proceso de negocio cuando son invocados, es decir, se configuran con la solución a ejecutar dentro del modelo. Esta solución puede que no tenga suficientes valores para una determinada variable, puesto que la invocación al participante que la retorna puede encontrarse dentro de un bucle que se ejecuta, a priori, un número desconocido de veces. Cuando un participante se encuentra en esta situación debe solicitar nuevos valores para la variable a TCSS-LS.

Por otro lado, si la transición seleccionada por TCSS-LS no tiene al menos dos soluciones para realizar las combinaciones se debe llevar a cabo un proceso de backtracking, el cual combina la técnica Búsqueda Dispersa con un método de búsqueda local.

El proceso de backtracking, las reglas de combinación y los procedimientos seguidos en la selección de una transición (similar a la selección de un nodo en el grafo de control de flujo), la creación de los subconjuntos de soluciones, la mejora de las soluciones y la actualización de los conjuntos S_k utilizando la propiedad de diversidad se pueden consultar en [5].

El proceso de búsqueda finaliza cuando todas las transiciones han sido alcanzadas o se ha generado el número máximo de casos de prueba.

La solución final de TCSS-LS está formada por los casos de prueba que cubren las transiciones del grafo de estados, que están almacenados en los conjuntos S_k, el porcentaje de cobertura de transiciones alcanzado y el tiempo consumido en el proceso de búsqueda. Por ejemplo, un caso de prueba para el servicio "loan approval" definido en la Fig. 1 tiene como entradas request.amount = 50, risk.assessment = low y cubre el camino formado por las transiciones T0, T1, T3.

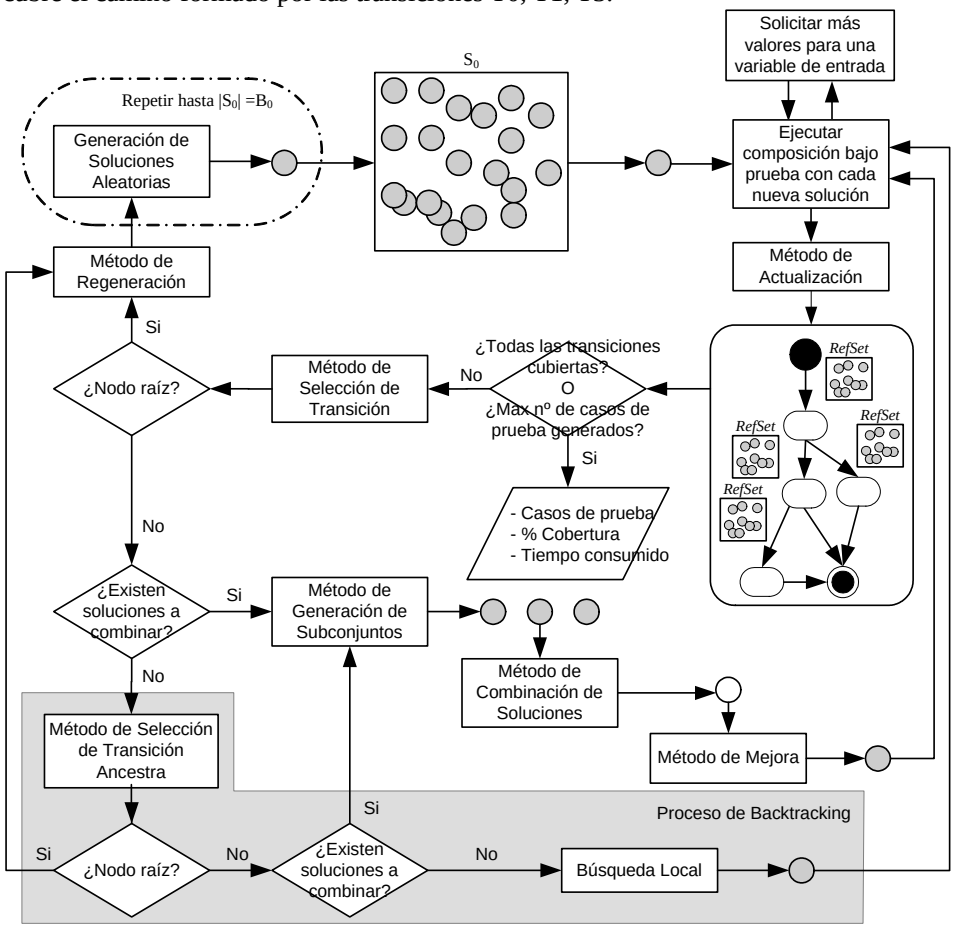


Fig. 2. Esquema de TCSS-LS (mejorado)

3.3 Tratamiento del número variable de valores de las variables de entrada

Un proceso de negocio especificado en BPEL puede contener invocaciones a servicios web dentro de un bucle, lo que ocasiona que la variable retornada por el servicio (el participante) tiene un valor diferente en cada iteración del bucle. Cada uno de esos valores puede cubrir diferentes transiciones del proceso de negocio y por lo tanto

todos ellos deben ser recordados para generar los casos de prueba. Como el número de iteraciones de un bucle no es conocido en la mayoría de los casos, una variable de entrada puede tomar un número desconocido de valores en la ejecución de la composición de servicios y el vector $\bar{x}j$ que la representa en la solución $\bar{x}_k{}^c$ puede tener diferente tamaño en dos soluciones determinadas.

Por ello el algoritmo TCSS-LS descrito en [5] ha sido ampliado para incluir un nuevo método que maneja el número variables de valores de una variable de entrada.

Cuando TCSS-LS crea una solución de forma aleatoria se genera un vector para cada variable de entrada situada dentro de un bucle con un número predeterminado de valores. Posteriormente los participantes en el proceso de negocio se configuran con los vectores de las variables que deben manejar y el proceso de negocio es ejecutado. Cada participante consume y retorna un valor del vector de la variable en cada invocación y cuando agota dichos valores debe solicitar otros nuevos a TCSS-LS.

TCSS-LS busca los nuevos valores para la variable de entrada entre las soluciones del conjunto S_k de la transición en evaluación Tk, tratando de encontrar aquellos que sean más diversos. Para ello se define la función "diversidad de una variable", la cual se calcula sobre el conjunto $S_{k'}=\{T_{k'}{}^1,\ldots,T_{k'}{}^q\} \subseteq S_k$, $T_{k'}{}^c=<\bar{x}_{k'}{}^c;p_{k'};fb_{k'}{}^c;fc_{k'}{}^c>$, que representa la soluciones almacenadas en la transición Tk que no han sido utilizadas previamente para suministrar nuevos valores a los participantes. El valor de diversidad de una variable $\bar{x}j$ se calcula según la siguiente función:

$$div\_var(<\bar{x}j_{k'}^{m}>;S_{k'}) = \sum_{c=1..q}\left(\sum_{z=1..r}\left\{\begin{array}{ll} \left|\bar{x}j_{k'}^{m_z}\right| & \text{si tamaño de } \bar{x}j_{k'}^{c} < z \\ \left|\bar{x}j_{k'}^{c_z}\right| & \text{si tamaño de } \bar{x}j_{k'}^{m} < z \\ \left|\bar{x}j_{k'}^{m_z} - \bar{x}j_{k'}^{c_z}\right| & \text{en caso contrario} \end{array}\right\}\right)$$

donde el índice c=1..q recorre las soluciones del conjunto $S_{k'}$ y el índice z=1..r recorre los valores de la variable de entrada $\bar{x}j$.

TCSS-LS entrega al participante los valores de la variable $\bar{x}j$ de la solución con mayor valor para la función *div_var()*, dado que dicha solución es la menos similar al resto de soluciones, de acuerdo a los valores de la variable $\bar{x}j$. De este modo se incrementa el tamaño del vector de valores de la variable de una solución que se está ejecutando en el proceso de negocio.

Después de que el proceso de negocio haya finalizado su ejecución, TCSS-LS analiza la solución para eliminar los valores de las variables que no han sido empleados, disminuyendo así el tamaño de los vectores. Posteriormente se lleva a cabo el proceso de actualización de los conjuntos S_k, los cuales almacenarán soluciones que presentan tamaños diferentes para los vectores de valores de una determinada variable.

Por otro lado, la generación de nuevas soluciones mediante las reglas de combinación y la función de diversidad presentada en [5] han sido adaptadas para manejar el número variable de valores de las variables de entrada de las soluciones almacenadas en los conjuntos S_k, como se muestra en [6].

Cuando TCSS-LS genera una solución en el proceso de combinación, crea el vector de valores de cada variable de entrada teniendo en cuenta el número de valores que dicha variable presenta en las soluciones a combinar. Posteriormente la nueva solución es ejecutada en la composición bajo prueba y si los participantes consumen

todos los valores de los vectores de una variable, éstos solicitarán otros nuevos a TCSS-LS como se ha indicado previamente.

La creación de soluciones por parte del método de búsqueda local utilizado en el backtracking también ha sido adaptada, para considerar que una nueva solución se genera mediante la modificación de todas las posiciones del vector de una variable determinada, de acuerdo a los procedimientos descritos en [5].

4 Casos de estudio

El algoritmo TCSS-LS presentado en este trabajo ha sido aplicado a dos especificaciones BPEL: “loan approval” y “shipping service”. Ambas especificaciones fueron originalmente publicadas con el estándar BPEL4WS y han sido ampliamente referenciadas en la literatura sobre pruebas de servicios web. La composición “shipping service” describe un servicio básico de gestión de envío de artículos. Los envíos se pueden realizar de dos formas diferentes: un único envío con el total de los artículos solicitados o diferentes envíos parciales hasta que todos los artículos solicitados inicialmente hayan sido enviados. Para comprobar los métodos diseñados en el enfoque presentado en este trabajo se ha modificado la composición “shipping service” como se muestra en la Fig. 3. Se han incluido las transiciones T5 y T6 para incrementar la complejidad de la composición con una condición de igualdad.

Los resultados obtenidos por TCSS-LS se comparan con los obtenidos por un generador aleatorio. En todos los experimentos las condiciones de parada utilizadas para ambos generadores fueron alcanzar el 100% de cobertura de transiciones o generar 200000 casos de prueba, las variables de entrada son de tipo entero y su rango de entrada utiliza 16 bits. Todas las ejecuciones fueron realizadas en una máquina con un procesador Pentium 4 2.80GHz con 512 MB de memoria RAM.

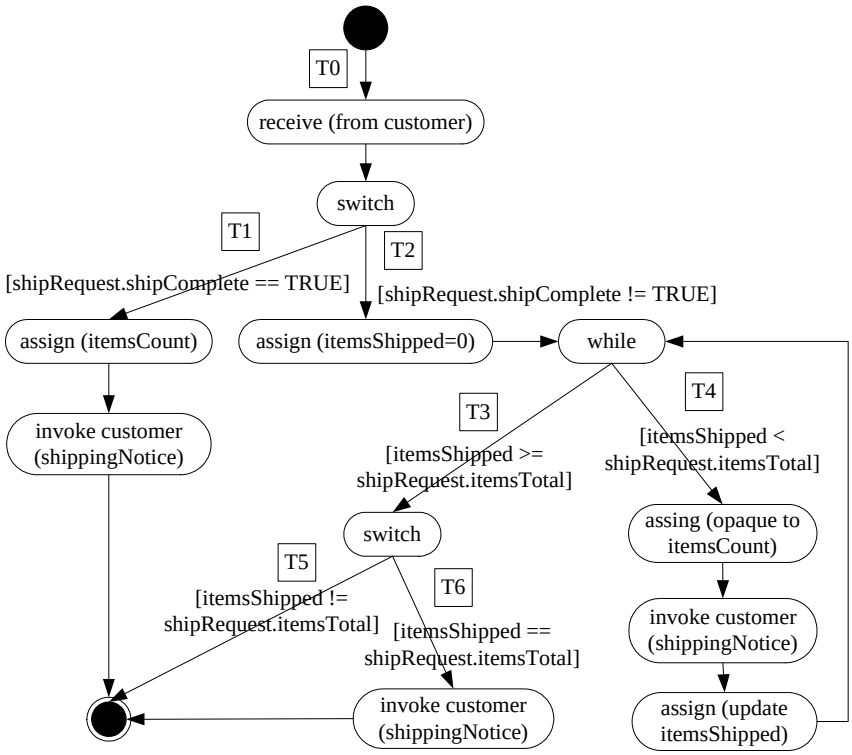


Fig. 3. Grafo de estados de la composición "shipping service"

La Tabla 1 muestra los resultados obtenidos por ambos generadores: porcentaje de cobertura de transiciones alcanzada, el número de casos de prueba que cada generador crea para obtener dicha cobertura y el tiempo consumido (en segundos). TCSS-LS genera menos casos de prueba y consume menos tiempo que el generador aleatorio en ambas composiciones. Además el generador aleatorio no alcanza el 100% de cobertura, mientras que TCSS-LS siempre obtiene la cobertura total.

Tabla 1. Resultados de las composiciones "loan approval" y "shipping service"

	Loan Approval			**Shipping Service**		
	% Cobertura	Casos de Prueba	Tiempo (s)	% Cobertura	Casos de Prueba	Tiempo (s)
TCSS-LS	100	290	0,19	100	144	0,23
Aleatorio	99	54940	1,31	75	36436	0,69

La parte izquierda de la Fig. 4 muestra la evolución del número de casos de prueba creados por ambos generadores para la composición "shipping service", mientras que la parte derecha muestra la evolución del tiempo consumido. El eje x representa el número de casos de prueba generados (parte izquierda de la figura) o el tiempo consumido (parte derecha de la figura) para alcanzar el porcentaje acumulado de cobertura de transiciones representado en el eje y. En estos gráficos se puede observar que

TCSS-LS genera menos casos de prueba y consume menos tiempo que el generador aleatorio para alcanzar cada porcentaje de cobertura de transiciones.

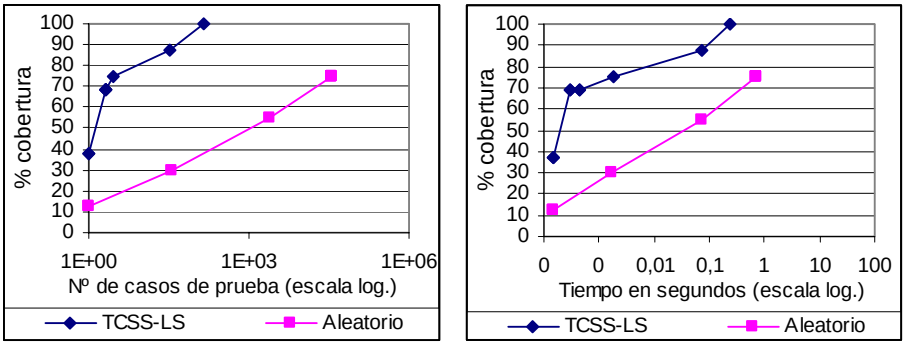


Fig. 4. Evolución del número de casos de prueba generados y el tiempo consumido para la composición "shipping service"

5 Conclusiones y trabajo futuro

Este trabajo presenta nuestro primer enfoque basado en la técnica metaheurística Búsqueda Dispersa para la generación automática de casos de prueba para procesos de negocio especificados en BPEL que satisfagan el criterio de cobertura de transiciones. Este enfoque, llamado TCSS-LS, es una evolución de un trabajo previo.

El proceso de negocio es modelado y representado mediante un grafo de estados. TCSS-LS maneja un conjunto de soluciones en cada transición del grafo, facilitando así la división del objetivo general en subobjetivos, y proporcionando mecanismos para manejar el número variable de valores de las variables de entrada.

Los resultados obtenidos muestran que TCSS-LS puede ser aplicado a la generación de casos de prueba para procesos de negocio especificados en BPEL y que mejora al generador aleatorio.

Líneas de trabajo futuro son el uso de otro criterio de suficiencia como cobertura de pares de transiciones, la mejora de TCSS-LS para manejar la ejecución concurrente de actividades en composiciones de servicios especificados en BPEL, y la experimentación con especificaciones de composiciones reales.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia e Innovación (España) dentro del Programa Nacional para I+D+I, proyectos Test4SOA (TIN2007-67843-C06-01) y RePRIS (TIN2007-30391-E).

Referencias

1. Ahmed, M.A., Hermadi, I.: GA-based multiple paths test data generator. Computers and Operations Research 35(10) (2008) 3107-3124.
2. Alba, E., Chicano, F.: Observations in using parallel and sequential evolutionary algorithms for automatic software testing. Computers and Operations Research 35(10) (2008) 3161-3183.
3. Alshraideh, M., Bottaci, L.: Search-based software test data generation for string data using program-specific search operators. Software Testing Verification and Reliability 16(3) (2006) 175-203.
4. Blanco, R., Tuya, J., Díaz, E., Adenso-Díaz, B.: A scatter search approach for automated branch coverage in software testing. Engineering Intelligent Systems 15(3) (2007) 135-142.
5. Blanco, R., Tuya, J., Adenso-Díaz, B.: Automated test data generation using a scatter search approach, Information and Software Technology 51 (2009) 108-720.
6. Blanco, R., García-Fanjul, J., Tuya, J.: A first approach to test case generation for BPEL compositions of web services using Scatter Search. In: Proceedings of the IEEE International Conference on Software Testing, Verification, and Validation Workshops (2009) 131-140.
7. Bueno, P.M.S., Wong, W.E., Jino, M.: Improving random test sets using the diversity oriented test data generation. In: Proceedings of the Second International Workshop on Random Testing (2007) 10-17.
8. Bühler, O., Wegener, J.: Evolutionary functional testing. Computers and Operational Research 35(10) (2008) 3144-3160.
9. Del Grosso, C., Antoniol, G., Merlo, E., Galinier, P.: Detecting buffer overflow via automatic test input data generation. Computers and Operational Research 35(10) (2008) 3125-3143.
10. Di Penta, M., Canfora, G., Esposito, G., Mazza, V., Bruno, M.: Search-based testing of service level agreements. In: Proceedings of the 9th conference on Genetic and Evolutionary Computation (2007) 1090-1097.
11. Díaz, E.: Generación automática de pruebas estructurales de software mediante Búsqueda Tabú. PhD Thesis Department of Computer Science, University of Oviedo, 2004.
12. Díaz, E., Tuya, J., Blanco, R., Dolado, J.J.: A tabu search algorithm for Software Testing. Computers and Operational Research 35(10) (2008) 3052-3072.
13. Dong, W.L., Yu, H., Zhang, Y.B.: Testing BPEL-based Web Service Composition Using High-level Petri Nets. In: Proceedings of the 10th IEEE Int. EDOC Conf. Hong Kong (2006) 441-444.
14. García-Fanjul, J., Tuya, J, de la Riva, C.: Generating test cases specifications for BPEL compositions of web services using SPIN. In Proceedings of the Int. Workshop on Web Services – Modeling and Testing (2006) 83-94.
15. Girgis, M.R.: Automatic test data generation for data flow testing using a genetic algorithm. Journal of Universal Computer Science 11(6) (2005) 898-915.
16. Glover, F., Laguna, M., Martí, R.: Fundamentals of Scatter Search and Path Relinking. Control and Cybernetics 39(3) (2000) 653-684.
17. Harman, M., Jones, B.F.: Search-based software engineering. Information and Software Technology 43(14) (2001) 833-839.
18. Huang, H., Tsai, W-T, Paul R., Chen, Y.: Automated Model Checking and Testing for Composite Web Services. In: Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (2005) 300-307.
19. Laguna, M., Martí, R.: Scatter Search: Methodology and Implementations in C. Kluwer Academic Publishers, Boston, MA, USA (2002).

20. Li, Z., Harman, M., Hierons, R.M.: Search algorithms for regression test case prioritization. IEEE Transactions on Software Engineering 33(4) (2007) 225-237.
21. Mansour, N., Salame, M.: Data generation for path testing. Software Quality Journal 12, (2004) 121-136.
22. Mantere,T., Alander, J.T.: Evolutionary software engineering, a review. Applied Soft Computing 5(3) (2005) 315-331.
23. McMinn, P.: Search-based software test data generation: a survey. Software Testing Verification and Reliability 14(2) (2004) 105-156.
24. McMinn, P., Holcombe, M.: Evolutionary testing using an extended chaining approach. Evolutionary Computation 14(1) (2006) 41-64.
25. Miller, J., Reformat, M., Zhang, H.: Automatic test data generation using genetic algorithm and program dependence graphs. Information and Software Technology 48 (2006) 586-605.
26. Organization for the Advancement of Structured Information Standards (OASIS), Web Services Business Process Execution Language (WSBPEL), URL: http://www.oasis-open.org.
27. Sagarna, R., Lozano, J.A.: Scatter Search in software testing, comparison and collaboration with Estimation of Distribution Algorithms, European Journal of Operational Research 169, (2006) 392-412.
28. Vergilio, S.R., Pozo, A.: A grammar-guided genetic programming framework configured for data mining and software testing. International Journal of Software Engineering and Knowledge Engineering 16(2) (2006) 245-267.
29. Waeselynck, H., Thévenod-Fosse,P., Abdellatif-Kaddour, O.: Simulated annealing applied to test generation: landscape characterization and stopping criteria. Empirical Software Engineering 12(1) (2007) 35-63.
30. Watkins, A., Hufnagel, E.M., Berndt, D., Johnson, L.: Using genetic algorithms and decision tree induction to classify software failures. International Journal of Software Engineering and Knowledge Engineering 16(2) (2006) 269-291.
31. Wegener, J., Baresel, A., Sthamer, H.: Evolutionary test environment for automatic structural testing, Information & Software Technology 43(14) (2001) 841-854.
32. Xiao, M., El-Attar, M., Reformat, M., Miller, J.: Empirical evaluation of optimization algorithms when used in goal-oriented automated test data generation techniques. Empirical Software Engineering 12(2) (2007) 183-239.

Ant Colony Optimization in Model Checking

Francisco Chicano and Enrique Alba

University of Málaga, Spain,
{chicano, eat}@lcc.uma.es

Abstract. Most of model checkers found in the literature use exact deterministic algorithms to check the properties. The memory required for the verification with these algorithms usually grows in an exponential way with the size of the system to verify. When the search for errors with a low amount of computational resources (memory and time) is a priority (for example, in the first stages of the implementation of a program), non-exhaustive algorithms using heuristic information can be used. In this work we summarize our observations after the application of Ant Colony Optimization to find property violations in concurrent systems using a explicit state model checker. The experimental studies show that ACO finds optimal or near optimal error trails in faulty concurrent systems with a reduced amount of resources, outperforming in most cases the results of algorithms that are widely used in model checking, like Nested Depth First Search. This fact makes ACO suitable for checking properties in large faulty concurrent programs, in which traditional techniques fail to find counterexamples because of the model size.

1 Introduction

Model checking [7] is a fully automatic technique that allows to check if a given concurrent system satisfies a property like, for example, the absence of deadlocks, the absence of starvation, the fulfilment of an invariant, etc. The use of this technique is a must when developing software that controls critical systems, such as an airplane or a spacecraft. However, the memory required for the verification usually grows in an exponential way with the size of the system to verify. This fact is known as the *state explosion problem* and limits the size of the system that a model checker can verify.

When the search for errors with a low amount of computational resources (memory and time) is a priority (e.g., in the first stages of the development), non-exhaustive algorithms using heuristic information can be used. A well-known class of non-exhaustive algorithms for solving complex problems is the class of metaheuristic algorithms [3]. They are search algorithms used in optimization problems that can find good quality solutions in a reasonable time. In this work we summarize the approaches used for the application of one metaheuristic, Ant Colony Optimization (ACO), to the problem of finding property violations in concurrent systems. We also show the results of some experimental studies analyzing the performance of the different approaches.

The paper is organized as follows. The next section presents the background information. Section 3 describes our algorithmic proposals. In Section 4 we present some experimental studies to analyze the performance of our proposals. We also compare our proposals against the most popular algorithms utilized in model checking. Finally, Section 5 outlines the conclusions and future work.

2 Background

In this section we give some details on the way in which properties are checked in explicit state model checking. In particular, we will focus on the model checker HSF-SPIN [9], an experimental model checker by Edelkamp, Lluch-Lafuente and Leue based on the popular model checker SPIN [12]. First, we formally define the concept of *property* of a concurrent system and we detail how the properties are checked. Then, we define the concepts of *strongly connected components* (SCC), *partial order reduction* (POR) and the use of heuristic information.

2.1 Properties and Checking

Let S be the set of possible *states* of a program (concurrent system), S^ω the set of infinite sequences of program states, and S^* the set of finite sequences of program states. The elements of S^ω are called *executions* and the elements of S^* are *partial executions*. However, (partial) executions are not necessarily real (partial) executions of the program. The set of real executions of the program, denoted by M, is a subset of S^ω, that is, $M \subseteq S^\omega$. A *property* P is also a set of executions, $P \subseteq S^\omega$. We say that an execution $\sigma \in S^\omega$ *satisfies* the property P if $\sigma \in P$, and σ *violates* the property if $\sigma \notin P$. In the former case we use the notation $\sigma \vdash P$, and the latter case is denoted with $\sigma \nvdash P$. A property P is a *safety property* if for all executions σ that violate the property there exists a prefix σ_i (partial execution) such that all the extensions of σ_i violate the property. Formally,

$$\forall \sigma \in S^\omega : \sigma \nvdash \mathcal{P} \rightarrow (\exists i \geq 0 : \forall \beta \in S^\omega : \sigma_i \beta \nvdash \mathcal{P}) \ , \tag{1}$$

where σ_i is the partial execution composed of the first i states of σ. Some examples of safety properties are the absence of deadlocks and the fulfilment of invariants. On the other hand, a property P is a *liveness property* if for all the partial executions α there exists at least one extension that satisfies the property, that is,

$$\forall \alpha \in S^* : \exists \beta \in S^\omega, \alpha\beta \vdash \mathcal{P} \ . \tag{2}$$

One example of liveness property is the absence of starvation. The only property that is a safety and liveness property at the same time is the trivial property $P = S^\omega$. It can be proved that any given property can be expressed as an intersection of a safety and a liveness property [2].

In explicit state model checking the concurrent system M and the property P are represented by finite state ω-automata, $\mathcal{A}(M)$ and $\mathcal{A}(P)$ respectively, that accept those executions they contain. In HSF-SPIN (and SPIN) the automaton $\overline{\mathcal{A}(P)}$, which captures the violations of the property, is called *never claim*. In

order to find a violation of a given property, HSF-SPIN explores the intersection (or synchronous product) of the concurrent model and the *never claim*, $\mathcal{A}(M) \cap \overline{\mathcal{A}(P)}$, also called Büchi automaton. HSF-SPIN searches in the Büchi automaton for an execution $\sigma = \alpha\beta^{\omega}$ composed of a partial execution $\alpha \in S^*$ and a cycle of states $\beta \in S^*$ containing an accepting state. If such an execution is found it violates the liveness component of the property and, thus, the whole property. During the search, it is also possible to find a state in which the end state of the *never claim* is reached (if any). This means that an execution has been found that violates the safety component of the property and the partial execution $\alpha \in S^*$ that leads the model to that state violates the property.

Nested Depth First Search algorithm (NDFS) [11] is the most popular algorithm for performing the search. However, if the property is a safety one (the liveness component is *true*) the problem of finding a property violation is reduced to find a partial execution $\alpha \in S^*$, i.e., it is not required to find an additional cycle containing the accepting state. In this case, classical graph exploration algorithms such as Breadth First Search (BFS), or Depth First Search (DFS) can be used for finding property violations.

2.2 Strongly Connected Components

In order to improve the search for property violations it is possible to take into account the structure of the *never claim*. The idea is based on the fact that a cycle of states in the Büchi automaton entails a cycle in the *never claim* (and in the concurrent system). For improving the search first we need to compute the *strongly connected components* (SCCs) of the *never claim*. Then, we classify the SCCs into three categories depending on the accepting cycles they include. By an N-SCC, we denote an SCC in which no cycle is accepting. A P-SCC is an SCC in which there exists at least one accepting cycle and at least one non-accepting cycle. Finally, a F-SCC is an SCC in which all the cycles are accepting [10].

All the cycles found in the Büchi automaton have an associated cycle in the *never claim*, and, according to the definition of SCC, this cycle is included in one SCC of the *never claim*. Furthermore, if the cycle is accepting (which is the objective of the search) this SCC is necessarily a P-SCC or an F-SCC. The classification of the SCCs of the *never claim* can be used to improve the search for property violations. In particular, the accepting states in an N-SCC can be ignored, and the cycles found inside an F-SCC can be considered as accepting.

2.3 Partial Order Reduction

Partial order reduction (POR) is a method that exploits the commutativity of asynchronous systems in order to reduce the size of the state space. The interleaving model in concurrent systems imposes an arbitrary ordering between concurrent events. When the automaton of the concurrent system is built, the events are interleaved in all possible ways. The ordering between independent concurrent instructions is meaningless. Hence, we can consider just one ordering for checking one given property since the other orderings are equivalent. This fact can be used to construct a reduced state graph hopefully much easier to explore compared to the full state graph (original automaton).

We use here a POR proposal based on *ample sets*. The main idea of ample sets is to explore only a subset of the enabled transitions of each state such that the reduced state space is equivalent to the full state space. This reduction of the state space is performed on-the-fly while the graph is generated.

2.4 Using Heuristic Information

In order to guide the search to the accepting states, a heuristic value is associated to each state of the transition graph of the model. Different kinds of heuristic functions have been defined in the past to better guide exhaustive algorithms. Formula-based heuristics, for example, are based on the expression of the LTL formula checked [9]. Using the logic expression that must be false in an accepting state, these heuristics estimate the number of transitions required to get such an accepting state from the current one. Given a logic formula φ, the heuristic function for that formula H_φ is defined using its subformulae. In this work we use a formula-based heuristic that is defined in [9].

There is another group of heuristic functions called state-based heuristics that can be used when the objective state is known. From this group we can highlight the distance of finite state machines H_{fsm}, in which the heuristic value is computed as the sum of the minimum number of transitions required to reach the objective state from the current one in the local automaton of each process.

3 Algorithmic proposals

In order to find property violations in concurrent systems we proposed in the past an algorithm that we call ACOhg, a new variant of ACO [1]. This algorithm can be used when the property to check is a safety property. In the case of liveness properties we use a different algorithm, called ACOhg-live, that contains ACOhg as a component. We describe ACOhg in the next section and ACOhg-live in Section 3.2.

3.1 ACOhg algorithm

The objective of ACOhg is to find a path from the initial node to one objective node from a set O in a very large exploration graph. We denote with f a function that maps the paths of the graph into real numbers. This function must be designed to reach minimum values when the shortest path to an objective node is found. ACOhg minimizes this objective function. In Algorithm 1 we show the pseudocode of ACOhg.

The algorithm works as follows. At the beginning, the variables are initialized (lines 1-5). All the pheromone trails are initialized with the same value: a random number between τ_0^{min} and τ_0^{max}. In the `init` set (initial nodes for the ants construction), a starting path with only the initial node is inserted (line 1). This way, all the ants of the first stage begin the construction of their path at the initial node.

After the initialization, the algorithm enters in a loop that is executed until a given maximum number of steps (*msteps*) set by the user is performed (line 6). In a loop, each ant builds a path starting in the final node of a previous path

Algorithm 1 ACOhg

```
1: init = {initial_node};
2: next_init = ∅;
3: τ = initializePheromone();
4: step = 1;
5: stage = 1;
6: while step ≤ msteps do
7:    for k=1 to colsize do {Ant operations}
8:       a^k = ∅;
9:       a^k_1 = selectInitNodeRandomly (init);
10:      while |a^k| < λ_ant ∧ T(a^k_*) − a^k ≠ ∅ ∧ a^k_* ∉ O do
11:         node = selectSuccessor (a^k_*, T(a^k_*), τ,η);
12:         a^k = a^k + node;
13:         τ = localPheromoneUpdate(τ,ξ,node);
14:      end while
15:      next_init = selectBestPaths(init, next_init, a^k);
16:      if f(a^k) < f(a^best) then
17:         a^best = a^k;
18:      end if
19:   end for
20:   τ = pheromoneEvaporation(τ, ρ);
21:   τ = pheromoneUpdate(τ, a^best);
22:   if step ≡ 0 mod σ_s then
23:      init = next_init;
24:      next_init = ∅;
25:      stage = stage+1;
26:      τ = pheromoneReset();
27:   end if
28:   step = step + 1;
29: end while
```

(line 9). This path is randomly selected from the `init` set. For the construction of the path, the ants enter a loop (lines 10-14) in which each ant k stochastically selects the next node according to the pheromone (τ_{ij}) and the heuristic value (η_{ij}) associated to each arc (a^k_*, j) with $j \in T(a^k_*)$ (line 11). The expression used is the standard random proportional rule used in ACOs [8].

After the movement of an ant from a node to the next one the pheromone trail associated to the arc traversed is updated as in Ant Colony Systems (ACS) [8] using the expression $\tau_{ij} \leftarrow (1-\xi)\tau_{ij}$ (line 13) where ξ, with $0 < \xi < 1$, controls the evaporation of the pheromone during the construction phase. This mechanism increases the exploration of the algorithm, since it reduces the probability that an ant follows the path of a previous ant in the same step. The construction process is iterated until the ant reaches the maximum length λ_{ant}, it finds an objective node, or all the successors of the last node of the current path, $T(a^k_*)$, have been visited by the ant during the construction phase. This last condition prevents the ants from constructing cycles in their paths.

After the construction phase, the ant is used to update the `next_init` set (line 15), which will be the `init` set in the next stage. In `next_init`, only starting paths are allowed and all the paths must have different last nodes. This rule is ensured by `selectBestPaths`. The cardinality of `next_init` is bounded by a given parameter ι. When this limit is reached and a new path must be included in the set, the starting path with higher objective value is removed from the set.

When all the ants have built their paths, a pheromone update phase is performed. First, all the pheromone trails are reduced according to the expression $\tau_{ij} \leftarrow (1-\rho)\tau_{ij}$ (line 20), where ρ is the *pheromone evaporation rate* and it holds that $0 < \rho \leq 1$. Then, the pheromone trails associated to the arcs traversed by the best-so-far ant (a^{best}) are increased using the expression $\tau_{ij} \leftarrow \tau_{ij} + 1/f(a^{best})$, $\forall (i,j) \in a^{best}$ (line 21). This way, the best path found is awarded with an extra amount of pheromone and the ants will follow that path with higher probability in the next step. We use here the mechanism introduced in Max-Min Ant Systems ($\mathcal{MM}$AS) [8] for keeping the value of pheromone trails in a given interval $[\tau_{min}, \tau_{max}]$ in order to maintain the probability of selecting one node above a given threshold. The values of the trail limits are $\tau_{max} = 1/\rho f(a^{best})$ and $\tau_{min} = \tau_{max}/a$ where the parameter a controls the size of the interval.

Finally, with a frequency of σ_s steps, a new stage starts. The `init` set is replaced by `next_init` and all the pheromone trails are removed from memory (lines 22-27). In addition to the pheromone trails, the arcs to which the removed pheromone trails are associated are also discarded (unless they also belong to a path in *next_init*). This removing step allows the algorithm to reduce the amount of memory required to a minimum value. This minimum amount of memory is the one utilized for storing the best paths found in one stage (the *next_init* set).

3.2 ACOhg-live

In this section we present ACOhg-live, an algorithm based on ACOhg for searching for general property violations in concurrent systems. In Algorithm 2 we show a high level object oriented pseudocode of ACOhg-live. We assume that `acohg1` and `acohg2` are two instances of a class implementing ACOhg.

The search that ACOhg-live performs is composed of two different phases. In the first one, ACOhg is used for finding accepting states in the Büchi automaton (line 2 in Algorithm 2). In this phase, the search of ACOhg starts in the initial node of the graph q and the set of objective nodes O is empty. That is, although the algorithm searches for accepting states, there is no preference on a specific set of them. If the algorithm finds accepting states, in a second phase a new search is performed using ACOhg again for each accepting state discovered (lines 3 to 8). In this second search the objective is to find a cycle involving the accepting state. The search starts in one accepting state and the algorithm searches for the same state in order to find a cycle. That is, the initial node of the search and the only objective node are the same: the accepting state. If a cycle is found ACOhg-live returns the complete accepting path (line 6). If no cycle is found for any of the accepting states ACOhg-live runs again the first phase after including the accepting states in a tabu list (line 9). This tabu list prevents the algorithm from searching again cycles containing the just explored accepting states. If one of the accepting states in the tabu list is reached it will not be included in the list of accepting states to be explored in the second phase. ACOhg-live alternates between the two phases until no accepting state is found in the first one (line 10).

The algorithm can also stop its search due to another reason: an end state has been found. That is, when an end state is found either in the first or the second phase of the search the algorithm stops and returns the path from the initial

Algorithm 2 ACOhg-live

```
1: repeat
2:    accpt = acohg1.findAcceptingStates(); {First phase}
3:    for node in accpt do
4:       acohg2.findCycle(node); {Second phase}
5:       if acohg2.cycleFound() then
6:          return acohg2.acceptingPath();
7:       end if
8:    end for
9:    acohg1.insertTabu(accpt);
10: until empty(accpt)
11: return null;
```

state to that end state. If this happens, an execution of the concurrent system has been found that violates the safety component of the checked property.

When the property to check is the absence of deadlocks only the first phase of the search is required. In this case, ACOhg-live searches for deadlock states (states with no successors) instead of accepting states. When a deadlock state is found the algorithm stops returning the path from the initial state to that deadlock state. The second phase of the search, the objective of which is to find an accepting cycle, is never run in this situation.

Now we are going to give the details of the ACOhg algorithms used inside ACOhg-live. First of all, we use a node-based pheromone model, that is, the pheromone trails are associated to the nodes instead of the arcs. This means that all the values τ_{xj} associated to the arcs which head is node j are in fact the same value and is associated to node j. The heuristic values η_{ij} are defined after the heuristic function H using the expression $\eta_{ij} = 1/(1 + H(j))$. This way, η_{ij} increases when $H(j)$ decreases (high preference to explore node j).

Finally, the objective function f to be minimized is defined as

$$f(a^k) = \begin{cases} |\pi + a^k| \text{ if } a^k_* \in O \\ |\pi + a^k| + H(a^k_*) + p_p + p_c \frac{\lambda_{ant} - |a^k|}{\lambda_{ant} - 1} \text{ if } a^k_* \notin O \end{cases}, \tag{3}$$

where π is the starting path in `init` whose last node is the first one of a^k, p_p, and p_c are penalty values that are added when the ant does not end in an objective node and when a^k contains a cycle, respectively. The last term in the second row of Eq. (3) makes the penalty higher in shorter cycles (see [4] for more details).

4 Experimental studies

In this section we present some experimental studies aimed at analyzing the performance of our ACO proposals for the problem of finding property violations in concurrent systems. In the following section we present the Promela models used in the experimentation. Then we show the results of four different analyses: two of them related to the violation of safety properties and the other two related to liveness properties. In all the cases, 100 independent runs of the ACOhg algorithms are performed and the average and the standard deviation are shown.

4.1 Models

In the empirical studies we used nine Promela models, some of them scalable. In Table 1 we present the models with some information about them. They can be found in `oplink.lcc.uma.es` together with the HSF-SPIN and ACOhg source code. In the table we also show the safety and liveness properties that we check in the models.

Table 1. Promela models used in the experiments

Model	LoC	Processes	Safety property	Liveness property
phi$j$	57	$j+1$	deadlock	$\Box(p \rightarrow \Diamond q)$
giopi, j	740	$i+3(j+1)$	deadlock	$\Box(p \rightarrow \Diamond q)$
marriers$j$	142	$j+1$	deadlock	
leader$j$	178	$j+1$	assertion	
needham	260	4	LTL formula	
pots	453	8	deadlock	
alter	64	2		$\Box(p \rightarrow \Diamond q) \wedge \Box(r \rightarrow \Diamond s)$
elev$j$	191	$j+3$		$\Box(p \rightarrow \Diamond q)$
sgc	1001	20		$\Diamond p$

4.2 Safety properties

In this section we compare the results obtained with ACOhg for safety properties against the ones obtained with exact algorithms previously found in the literature. These algorithms are Breadth First Search (BFS), Depth First Search (DFS), A*, and Best First Search (BF). BFS and DFS do not use heuristic information while the other two do. In order to make a fair comparison we use two different ACOhg algorithms: one not using heuristic information (ACOhg-b) and another one using it (ACOhg-h). We show the results of all the algorithms in Table 2. In the table we can see the hit rate (number of executions that got an error trail), the length of the error trails found (number of states), the memory required (in Kilobytes), and the CPU time used (in milliseconds) by each algorithm. We highlight with a grey background the best results (maximum values for hit rate and minimum values for the rest of the measures). For ACOhg-b and ACOhg-h we omit here the standard deviation due to room problems. The parameters used in the ACOhg algorithms are the ones of [1].

In general terms, we can state that ACOhg-b is a robust algorithm that is able to find errors in all the proposed models with a low amount of memory. In addition, it combines the two good features of BFS and DFS: it obtains short error trails, like BFS, while at the same time requires a reduced CPU time, like DFS. Regarding the algorithms using heuristic information, we can state that ACOhg-h is the best trade-off between solution quality and memory required: it obtains almost optimal solutions with a reduced amount of memory.

4.3 Influence of POR

In this section we are going to analyze how the combination of partial order reduction plus ACOhg can help in the search for safety property violations in concurrent models. In Table 3 we present the results of applying ACOhg and ACOhg^{POR} to nine models: three instances of `giop`, `marriers`, and `leader`. The hit rate is always 100 %, and for this reason we omit it. In order to clarify that

Table 2. Results of ACOhg-b and ACOhg-h against the exhaustive algorithms.

Model	Measure	BFS	DFS	ACOhg-b	A*	BF	ACOhg-h
giop2,2	Hit rate	0/1	1/1	100/100	1/1	1/1	100/100
	Length	-	112.00	45.80	44.00	44.00	44.20
	Mem. (KB)	-	3945.00	4814.12	417792.00	2873.00	4482.12
	Time (ms)	-	30.00	113.60	46440.00	10.00	112.40
marriers4	Hit rate	0/1	0/1	57/100	0/1	1/1	84/100
	Length	-	-	92.18	-	108.00	86.65
	Mem. (KB)	-	-	5917.91	-	41980.00	5811.43
	Time (ms)	-	-	257.19	-	190.00	233.33
needham	Hit rate	1/1	1/1	100/100	1/1	1/1	100/100
	Length	5.00	11.00	6.39	5.00	10.00	6.12
	Mem. (KB)	23552.00	62464.00	5026.36	19456.00	4149.00	4865.40
	Time (ms)	1110.00	18880.00	262.00	810.00	20.00	229.50
phi16	Hit rate	0/1	0/1	100/100	1/1	1/1	100/100
	Length	-	-	31.44	17.00	81.00	23.08
	Mem. (KB)	-	-	10905.60	2881.00	10240.00	10680.32
	Time (ms)	-	-	289.40	10.00	40.00	243.80
pots	Hit rate	1/1	1/1	49/100	1/1	1/1	99/100
	Length	5.00	14.00	5.73	5.00	7.00	5.44
	Mem. (KB)	57344.00	12288.00	9304.67	57344.00	6389.00	6974.56
	Time (ms)	4190.00	140.00	441.63	6640.00	50.00	319.49

the reduced amount of memory required by the ACOhg algorithms is not due to the use of the heuristic information, we also show the results obtained with A* for all the models using the same heuristic functions as the ACOhg algorithms. This clearly states that memory reduction is a very appealing attribute of ACOhg itself.

Table 3. Comparison among ACOhg, ACOhg^{POR} and A*.

Model	Measure	ACOhg		ACOhg^{POR}		A*
giop2,1	Length	42.30	1.71	42.10	0.99	42.00
	Mem. (KB)	3428.44	134.95	2979.48	98.33	27648.00
	Time (ms)	202.00	9.06	162.50	5.55	1000.00
giop4,1	Length	70.21	7.56	59.76	5.79	-
	Mem. (KB)	9523.67	331.76	7420.08	422.94	-
	Time (ms)	354.50	42.39	264.90	40.46	-
giop6,1	Length	67.59	13.43	61.74	3.16	-
	Mem. (KB)	11970.56	473.59	11591.68	477.67	-
	Time (ms)	440.60	71.02	391.70	43.86	-
leader6	Length	50.90	4.52	56.36	3.04	37.00
	Mem. (KB)	16005.12	494.39	3710.64	410.29	132096.00
	Time (ms)	494.00	21.12	98.80	8.16	1250.00
leader8	Length	60.83	4.66	74.11	4.51	-
	Mem. (KB)	24381.44	515.98	4831.40	114.10	-
	Time (ms)	1061.20	211.47	198.90	4.67	-
leader10	Length	73.84	4.79	80.86	6.36	-
	Mem. (KB)	30167.04	586.82	7178.05	2225.78	-
	Time (ms)	1910.70	45.02	294.90	66.96	-
marriers10	Length	307.11	34.87	233.19	21.91	-
	Mem. (KB)	34170.88	494.39	18319.36	804.93	-
	Time (ms)	8847.00	634.06	1306.60	126.56	-
marriers15	Length	540.41	60.88	395.10	40.07	-
	Mem. (KB)	51148.80	223.18	26050.56	1256.81	-
	Time (ms)	19740.50	1935.54	3595.00	316.59	-
marriers20	Length	793.62	80.45	569.99	54.63	-
	Mem. (KB)	68003.84	503.64	33351.68	1442.75	-
	Time (ms)	49446.30	7557.40	8174.00	707.71	-

From the results in the table we conclude that the memory required by ACOhg^{POR} is always smaller than the one required by ACOhg. The length of the error paths is smaller for ACOhg^{POR} in six out of the nine models. Finally, the CPU time required by ACOhg^{POR} is up to 6.8 times lower (in `marriers10`) than the time required by ACOhg. Although it is not our objective to optimize the length of the error paths in this work, we can say that, in six out of the nine models, the length of the error paths obtained by ACOhg^{POR} is shorter than the one obtained by ACOhg. We finally also remind that other popular algorithm like A* cannot even be applied to most of these instances (only `giop2,1` and `leader6` can be tackled with A*), and thus we are investigating in a new frontier of high dimension models usually not found in literature.

4.4 Liveness Results

In the next experiment we compare the results obtained with ACOhg-live against the classical algorithm utilized for finding liveness errors in concurrent systems: Nested-DFS. This last algorithm is deterministic and for this reason we only perform one single run. In Table 4 we show the results of both algorithms. We also show the results of a statistical test (with level of significance $\alpha = 0.05$) in order to check if there exist statistically significant differences (last column). A plus sign means that the difference is significant and a minus sign means that it is not. For more details on the experiments see [5].

Table 4. Comparison between ACOhg-live and Nested-DFS

Model	Measure	ACOhg-live		Nested-DFS	Test
`alter`	Hit rate	100/100		1/1	-
	Length	30.68	10.72	64.00	+
	Mem. (KB)	1925.00	0.00	1873.00	+
	Time (ms)	90.00	13.86	0.00	+
`giop2,2`	Hit rate	100/100		1/1	-
	Length	43.76	5.82	298.00	+
	Mem. (KB)	2953.76	327.48	7865.00	+
	Time (ms)	747.50	408.09	240.00	+
`giop6,2`	Hit rate	100/100		0/1	+
	Length	58.77	7.21	•	•
	Mem. (KB)	5588.04	631.36	•	•
	Time (ms)	8733.50	3304.90	•	•
`giop10,2`	Hit rate	86/100		0/1	+
	Length	62.85	7.03	•	•
	Mem. (KB)	9316.67	700.44	•	•
	Time (ms)	43059.07	21417.74	•	•
`phi8`	Hit rate	100/100		1/1	-
	Length	51.36	6.95	3405.00	+
	Mem. (KB)	2014.32	18.87	4005.00	+
	Time (ms)	2126.10	479.64	40.00	+
`phi14`	Hit rate	99/100		1/1	-
	Length	76.05	9.35	10001.00	+
	Mem. (KB)	2496.07	41.81	59392.00	+
	Time (ms)	8070.30	1530.12	2300.00	+
`phi20`	Hit rate	98/100		1/1	-
	Length	97.39	10.14	10001.00	+
	Mem. (KB)	3244.67	91.33	392192.00	+
	Time (ms)	18064.90	5538.30	17460.00	-

The first observation concerning the hit rate is that ACOhg-live is the only one that is able to find error paths in all the models. Nested-DFS is not able

to find error paths in `giop6,2` and `giop10,2` because it requires more than the memory available in the machine used for the experiments (512 MB). With respect to the length of the error paths we observe that ACOhg-live obtains shorter error executions than Nested-DFS in all the models (with statistical significance). If we focus on the computational resources we observe that ACOhg-live requires less memory than Nested-DFS to find the error paths with the only exception of `alter`. The biggest differences are those of `giop6,2` and `giop10,2` in which Nested-DFS requires more than 512 MB of memory while ACOhg-live obtains error paths with 38 MB at most. With respect to the time required for the search, Nested-DFS is faster than ACOhg-live. The mechanisms included in ACOhg-live in order to be able to find short error paths with high hit rate and low amount of memory extend the time required for the search. Anyway, the maximum difference with respect to the time is around six seconds (in `phi14`), which is not too much if we take into account that the error path obtained is much shorter.

4.5 Influence of the SCC improvement

In this final study we compare two versions of the ACOhg-live algorithm: one of them using the SCC improvement (called ACOhg-live$^+$ in the following) and the other one without that improvement (called ACOhg-live$^-$). With this experiment we want to analyze the influence on the results of the SCC improvement. All the properties checked in the experiments have at least one F-SCC in the never claim; none of them has a P-SCC; and all except `sgc` have exactly one N-SCC. In Table 5 we show the results. For more details on the experiments see [6].

Table 5. Influence of the SCC improvement

Model	Measure	ACohg-live$^-$		ACOhg-live$^+$		T	Model	ACohg-live$^-$		ACOhg-live$^+$		T
giop10,2	Hit rate		84/100		89/100	-	elev10		100/100		100/100	-
	Length	68.57	5.29	67.60	6.09	-		126.56	18.32	127.76	16.89	-
	Mem. (KB)	6375.90	542.50	5098.75	1580.90	+		2617.60	7.93	2617.04	9.72	-
	Time (ms)	7816.55	4779.41	935.84	1009.74	+		2577.30	2258.38	2372.90	1963.04	-
giop15,2	Hit rate		46/100		57/100	-	elev15		100/100		100/100	-
	Length	81.26	3.64	78.30	6.49	+		182.02	9.75	180.04	16.83	-
	Mem. (KB)	9001.17	483.22	8538.54	1610.63	-		3163.56	10.98	3164.64	13.44	-
	Time (ms)	11725.65	7307.22	2016.84	1254.88	+		2683.00	3274.20	2812.10	3540.73	-
giop20,2	Hit rate		14/100		30/100	+	elev20		100/100		100/100	-
	Length	93.29	2.08	88.47	4.72	+		233.00	0.00	231.62	13.73	-
	Mem. (KB)	11132.71	894.26	10403.17	1920.50	-		3716.44	13.15	3716.92	11.29	-
	Time (ms)	11360.00	4564.72	2575.33	1103.46	+		3900.60	7141.02	3034.00	4709.07	-
phi20	Hit rate		98/100		97/100	-	alter		100/100		100/100	-
	Length	88.29	6.91	108.73	10.08	+		10.00	0.00	15.82	6.74	+
	Mem. (KB)	3398.63	34.05	3385.04	63.41	-		1929.00	0.00	1929.00	0.00	-
	Time (ms)	5162.04	645.64	851.75	1462.71	+		241.80	59.35	10.40	3.98	+
phi30	Hit rate		94/100		95/100	-	sgc		32/100		100/100	+
	Length	122.60	9.58	139.15	9.06	+		24.00	0.00	24.00	0.00	-
	Mem. (KB)	5146.62	44.70	5148.12	57.48	-		2699.00	23.13	2285.00	0.00	+
	Time (ms)	10980.64	2156.73	2701.79	3876.34	+		575191.88	62021.86	710.20	48.58	+
phi40	Hit rate		77/100		81/100	-						
	Length	154.74	9.74	166.83	9.44	+						
	Mem. (KB)	7573.68	66.50	7545.35	81.04	+						
	Time (ms)	20422.60	5795.93	5807.41	7588.17	+						

From the results in Table 5, we conclude that the use of the SCC improvement increases the hit rate and decreases the computational resources required for the search. The length of error paths could be slightly increased depending on the particular model.

5 Conclusions and Future Work

In this paper we summarize our observations using ACO algorithms for the problem of searching for property violations in concurrent systems. The numerical results shown here are an excerpt from the research work performed during the last two years on this topic. From the results we conclude that ACO algorithms are promising for the model checking domain. They can find short error trails using a low amount of computational resources.

At present, we are investigating how other metaheuristic algorithms perform on this problem. We are also working on new heuristic functions that can guide the search in a better way. As future work, we plan to design and develop new models of parallel ACO algorithms in order to profit from the computational power of a cluster or a grid of computers.

6 Acknowledgements

This work has been partially funded by the Spanish Ministry of Science and Innovation and FEDER under contract TIN2008-06491-C04-01 (the M* project). It has also been partially funded by the Andalusian Government under contract P07-TIC-03044 (DIRICOM project).

References

1. Enrique Alba and Francisco Chicano. Finding safety errors with ACO. In *Proc. of GECCO*, pages 1066–1073, 2007.
2. Bowen Alpern and Fred B. Schneider. Defining liveness. *Inform. Proc. Letters*, 21:181–185, 1985.
3. C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.
4. Francisco Chicano and Enrique Alba. Ant colony optimization with partial order reduction for discovering safety property violations in concurrent models. *Information Processing Letters*, 106(6):221–231, June 2008.
5. Francisco Chicano and Enrique Alba. Finding liveness errors with ACO. In *Proceedings of the World Conference on Computational Intelligence*, pages 3002–3009, Hong Kong, China, 2008.
6. Francisco Chicano and Enrique Alba. Searching for liveness property violations in concurrent systems with ACO. In *Proceedings of Genetic and Evolutionary Computation Conference*, pages 1727–1734, Atlanta, USA, 2008.
7. Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, January 2000.
8. Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. The MIT Press, 2004.
9. Stefan Edelkamp, Alberto Lluch Lafuente, and Stefan Leue. Protocol Verification with Heuristic Search. In *AAAI-Spring Symposium on Model-based Validation Intelligence*, pages 75–83, 2001.
10. Stefan Edelkamp, Stefan Leue, and Alberto Lluch-Lafuente. Directed explicit-state model checking in the validation of communication protocols. *Intl. Jnl. of Soft. Tools for Tech. Transfer*, 5:247–267, 2004.
11. G. J. Holzmann, D. Peled, and M. Yannakakis. On nested depth first search. In *Proc. Second SPIN Workshop*, pages 23–32, 1996.
12. Gerald J. Holzmann. *The SPIN Model Checker*. Addison-Wesley, 2004.

Combinación de distribuciones de probabilidad con AHP

Joseba Esteban López , José Javier Dolado
Departamento de Lenguajes y Sistemas
Universidad del País Vasco U.P.V./E.H.U.
jose.esteban@ehu.es, dolado@si.ehu.es

No Institute Given

Resumen Dado el creciente aumento del uso de las redes Bayesianas en el área de la ingeniaría del software y que dichas redes trabajan con distribuciones de probabilidad tanto discretas como continuas, creemos interesante conocer como combinar varias de estas distribuciones. En este artículo se presenta el método matemático *combinación lineal de opiniones* para aunar distintas distribuciones de probabilidad. El único inconveniente que presenta este sencillo método, consiste en establecer los pesos de cada variable aleatoria. Combinando dicho método con el método de ayuda a toma de decisiones *Analytic Hierarchy Process*, podemos establecer los pesos de cada variable. En este artículo se exponen dos planteamientos en la combinación de distribuciones de probabilidad: combinar varias estimaciones de expertos para obtener una estimación más fiable, y estimar una variable global a partir de la estimación de las partes que lo componen.

1. Introducción

Las redes Bayesianas son cada vez más populares dentro de la ingeniería, la inteligencia artificial y la estadística. En la ingeniería del software se han utilizado en diferentes áreas como la estimación del esfuerzo y la calidad o en pruebas de software. Las redes Bayesianas son modelos gráficos probabilísticos utilizados en la toma de decisiones [1]. Dichas redes trabajan con variables aleatorias. Una variable aleatoria es una variable que puede tomar un valor numérico determinado por el resultado del experimento aleatorio. Es decir, sólo puede tomar un conjunto de valores concretos. Las variables aleatorias se representan mediante distribuciones de probabilidad. La distribución de probabilidad de una variable aleatoria establece el rango y la probabilidad de los valores que puede tomar la variable. Estas distribuciones de probabilidad pueden ser continuas o discretas determinando, de esta forma, el tipo de variable aleatoria. Se denomina variable continua a aquella que puede tomar cualquiera de los infinitos valores existentes dentro de un intervalo. En el caso de variable continua la distribución de probabilidad es la integral de la función de densidad. Por el contrario, se denomina distribución de variable discreta a aquella cuya función de probabilidad sólo toma valores positivos en un conjunto de valores finito o infinito numerable y se denomina función de masa de probabilidad.

Las redes Bayesianas se componen de dos partes. Una parte cualitativa que consiste en una estructura gráfica formada por nodos (variables aleatorias) y dependencias entre nodos. Y otra parte cuantitativa correspondiente a las tablas de probabilidad de los nodos. Estas tablas de probabilidad se pueden obtener a partir de bases de datos (en el caso de la ingeniería del software, de proyectos ya finalizados) o de una manera subjetiva, utilizando creencias de expertos en el domino [2] **Referencia Dolado 2007**.

En el caso de basarnos en el juicio experto para obtener las distribuciones de probabilidad de cada variable, es aconsejable apoyarse en varios expertos con el fin de minimizar posibles errores. Consultar a varios expertos puede verse como una versión subjetiva de aumentar la muestra en un experimento o de incrementar la información de base, es decir, hacerlo más fiable. Esto implica que los expertos deben resolver sus diferencias en cuanto a la definición de cada variable y ponerse de acuerdo en qué es lo que se quiere estimar. Conseguir llegar a un consenso entre los diferentes expertos puede suponer un gran esfuerzo, por lo que es recomendable que cada experto se familiarice con el el domino del problema con el fin de que todos tengan una idea similar del problema [3].

Una vez están de acuerdo, cada experto debe proporcionar su estimación en forma de distribución de probabilidad. Incluso estando conformes con las definiciones de las variables, es posible que los expertos estén en desacuerdo con las probabilidades de dichas variables. Estas desavenencias pueden deberse al empleo de diferentes métodos de obtención de información, las variaciones de los conjuntos de información que utilizan los expertos o los diferentes enfoques filosóficos. En cualquier caso, si los expertos nunca estuvieran en desacuerdo, no tendría cabida consultar a más de un experto. Una vez tenemos las diferentes opiniones de los expertos en forma de distribuciones de probabilidad, hay que combinarlas [3].

La combinación de distribuciones de probabilidad no es un ámbito exclusivo de unificar estimaciones de diferentes expertos sobre la misma variable. También puede darse la necesidad de aunar distintas distribuciones de probabilidad para obtener una distribución que globalice el resto de distribuciones. Por ejemplo, se puede establecer el esfuerzo de desarrollo de un software como la suma del esfuerzo de las diferentes funcionalidades que lo componen. En este caso consistiría en combinar la estimación de esfuerzo de cada funcionalidad que compone un software, para obtener una estimación del esfuerzo total del mismo. Las variables que se quieren combinar deben ser del mismo tipo (esfuerzo en nuestro ejemplo).

El enfoque axiomático, que comentamos en el apartado 2, permite combinar las distribuciones de probabilidad. Para poder utilizar este enfoque hay que establecer pesos a cada estimación. Con este objetivo, nosotros proponemos el método de ayuda a la toma de decisiones Analytic Hierarchy Process (A.H.P.) que planteamos en el tercer y cuarto apartado de este artículo. Por último exponemos las conclusiones y trabajo futuro.

2. Método matemático para la combinación de distribuciones de probabilidad: Enfoque axiomático

Los métodos de agregación matemáticos se componen de procesos analíticos que operan sobre varias distribuciones de probabilidad individuales para obtener una única distribución de probabilidad combinada. Estos métodos matemáticos van desde simples cálculos de sumas, como la media aritmética o geométrica de las probabilidades, hasta procedimientos basados en enfoques axiomáticos [3]. En esta sección exponemos una técnica axiomática para la combinación de distribuciones de probabilidad: Combinación lineal de opiniones (*linear opinion pool*) en el apartado 2.1.

2.1. Combinación lineal de opiniones (*linear opinion pool*)

Este método sencillo de combinación de probabilidades se remonta a la época de Laplace y consiste en aplicar la formula 1, donde n es el número de expertos, $p_i(\Theta)$ representa la distribución del experto i-ésimo para la variable Θ, $p(\Theta)$ representa la distribución de probabilidad combinada, y w_i representa los pesos. Estos pesos han de sumar uno. Por simplificar, p representa la función de masa de en el caso de una distribución de probabilidad conjunta, y la función de densidad en el caso continuo [3].

$$p(\Theta) = \sum_{i=1}^{n} w_i p_i(\Theta) \tag{1}$$

En el caso de querer unificar distribuciones de probabilidad para obtener las distribución de probabilidad de una variable global, n sería el número de variables no globales a combinar, $p_i(\Theta)$ representaría la distribución de la variable i-ésima del mismo tipo de variables a combinar, y $p(\Theta)$ representa la distribución de probabilidad combinada.

Esta técnica es claramente una combinación lineal ponderada de las probabilidades de los expertos. Este método de combinación es fácil de calcular y de entender, además de satisfacer varios axiomas. Por ejemplo, satisface la propiedad de unanimidad (*unanimity*) que establece que si todos los expertos están de acuerdo en una probabilidad, entonces también estarán de acuerdo en la probabilidad combinada. Este método de combinación es el único que cumple la propiedad de marginación: supongamos de Θ es un vector de probabilidades y que nos interesa sólo un elemento de dicho vector, Θ_j, la propiedad de marginación establece que las probabilidades combinadas son las mismas tanto si combinamos las distribuciones marginales de Θ_j, como si combinamos las distribuciones de probabilidad conjunta Θ y después calculamos la distribución marginal de Θ_j [3].

2.2. Establecimiento de pesos en la combinación lineal de opiniones

Los pesos w_i pueden ser utilizados para representar, de algún modo, la calidad de los diferentes expertos o bien el grado de influencia de cada variable

a combinar sobre la variable global. En el caso de querer combinar diferentes estimaciones de distintos expertos sobre una misma variable, los pesos se establecerán de acuerdo a la calidad del experto en el domino como estimador. En el caso de necesitar aunar varias estimaciones para obtener una estimación global, los pesos determinan la influencia relativa de cada variable en la variable global. En el caso del ejemplo anterior de combinar el esfuerzo de cada funcionalidad, no tiene el mismo peso sobre el esfuerzo total una funcionalidad que gestione el acceso de los usuarios a la aplicación, que una funcionalidad que englobe la lógica de negocio de, por ejemplo, una aplicación de alquiler de vehículos vía web, la cual resultará bastante más costosa en términos de esfuerzo.

Si se considera a todos los expertos por igual o que todas las variables influyen de igual manera en una variable global, los pesos tendrán todos el mismo valor $1/n$, siendo n el número de expertos o la cantidad de variables respectivamente. En este caso, este método de combinación de distribuciones de probabilidad se convierte en una media aritmética. Se puede entender que un experto es "mejor"que otro (por ejemplo debido a que dispone de mejor información), esto implicará que el peso asociado a dicho experto tendrá un valor superior al del resto. La determinación de los pesos es una cuestión subjetiva y se pueden dar múltiples interpretaciones a los pesos.

Quizá la mayor complejidad que presenta el método de combinación lineal de opiniones sea el establecimiento de los pesos. Con este fin proponemos la utilización del método de ayuda a la toma de decisiones Analytic Hierarchy Process (A.H.P.) que exponemos en los siguientes apartados.

3. Analytic Hierarchy Process (A.H.P.)

Analytic Hierarchy Process (A.H.P.) es un método de estimación de ayuda a la toma de decisiones basado en múltiples criterios de decisión. AHP fue propuesto por Thomas L. Saaty en la década de los 80 [4]. Desde entonces se ha convertido en una de las técnicas más utilizadas para la toma de decisiones multiatributo. AHP se basa en juicios subjetivos realizados por los expertos. Los expertos aportan su conocimiento subjetivo, consistente en comparaciones entre las principales tareas que constituyen un proyecto software. Los expertos estiman, más que un valor exacto, una medida relativa. Basándose en esta idea, el experto, evaluando la proporción entre cada par de tareas definidas en la aplicación software, consigue una mayor exactitud en sus evaluaciones.

3.1. Algoritmo AHP

El algoritmo del método de estimación Analytic Hierarchy Process consta de cinco pasos que exponen a lo largo de este apartado.

En primer lugar se define el problema. Para esto hay que dividirlo en tres partes: objetivo, criterios y alternativas. El objetivo es la decisión que se ha de tomar. Los criterios representan los factores que afectan a la preferencia o deseabilidad de una alternativa. Pueden estar compuestos por otros criterios o

Definición	Explicación	Valor Relativo	Valor Recíproco
Mismo tamaño	Las dos entidades tienen aproximadamente el mismo tamaño	1	1.00
Ligeramente mayor (menor)	La experiencia o el juicio reconoce una entidad como algo más grande (más pequeño)	3	0.33
Mayor (menor)	La experiencia o el juicio reconoce una entidad como definitivamente más grande (más pequeño)	5	0.20
Mucho mayor (menor)	El dominio de una entidad sobre otra es evidente; un diferencia muy fuerte de tamaño	7	0.14
Extremadamente mayor (menor)	La diferencia entre las entidades comparadas es de un orden de magnitud	9	0.11
Valores intermedios entre puntos adyacentes de la escala	Cuando el compromiso es necesario	2, 4, 6, 8	0.5, 0.25, 0.16, 0.12

Cuadro 1. Escala verbal propuesta por Thomas L. Saaty

subcriterios. Las alternativas son las posibles opciones o acciones de las que se dispone y de las cuales se intenta elegir una. Una alternativa puede ser cualquier entidad relevante en un grupo de interés, como casos de uso, módulos software, objetos etc., es decir, cualquier entidad de la que se pueda conocer las magnitudes que se necesitan a la hora de tomar una decisión. Una vez hecho esto, se debe construir la jerarquía, de la que AHP toma el nombre.

Aunque no se trate de una parte esencial de la metodología de AHP, establecer una escala verbal o *verbal scale* agiliza el proceso de estimación y no hace peligrar la exactitud de la estimación. La escala verbal ayuda a entender cómo de menor es el término "menor que"ó cómo de mayor es el término "mayor que". Se compone de cuatro atributos: definición o etiqueta, explicación, valor relativo y valor recíproco. La escala verbal establece un consenso que evita que los expertos o los participantes en la estimación pierdan tiempo discutiendo sobre el grado de diferencia entre las alternativas comparadas. Thomas L. Saaty [5]nos propone una escala compuesta por nueve valores y sus recíprocos, como se puede ver en 1.

Con la jerarquía y la escala verbal ya bien definidas, se pasa a obtener la matriz de juicios o *judgement matrix*. Es en esta etapa donde entra en juego el juicio experto. El experto, basándose en la escala verbal, debe hacer comparaciones por parejas en cada nivel de la jerarquía, e ir anotándolos en la matriz. La matriz de juicios es de tamaño nxn, siendo n el número de alternativas de las que se dispone. Cada celda de la matriz de juicios contiene un valor a_{ij}, que representa el tamaño relativo de la entidad i respecto del de la entidad j. Los elementos de la matriz se definen como se muestra en (3.1). Si la entidad i es a_{ij} veces mayor (o menor) que la entidad j, entonces la entidad j es $1/a_{ij}$ veces menor (o mayor) que la entidad i. Teniendo en cuenta esta premisa y que la diagonal de la matriz sólo tiene como valor la unidad, no haría falta calcular todos los valores de la matriz. Sólo es necesario calcular una mitad de la matriz, ya sea la parte superior a la diagonal o la inferior.

$$A^{n\times n} = \begin{cases} a_{ij} = \frac{s_i}{s_j} & \text{Cómo de grande o pequeña es la entidad} i \text{respecto de la entidad} j \\ a_{ij} = 1 & \text{Las entidades} i y j \text{son de la misma proporción} \\ a_{ij} = \frac{1}{a_{ji}} & \text{Inversamente proporcionales} \end{cases} \tag{2}$$

A la hora de hacer las comparaciones por parejas, es necesaria la colaboración del experto y al menos una entidad de referencia o *reference task* de la que se conozca su magnitud real. Las proporciones de la entidad de referencia son las primeras que se han de situar en la matriz. Es importante que la proporción de esta entidad no ocupe los valores extremos de la escala verbal, sino que se sitúe más o menos hacia la mitad de la escala. De esta manera se minimizan los posibles prejuicios introducidos en la matriz de juicios. Otra posibilidad, con el mismo objetivo, radica en introducir más de una entidad de referencia repartidas uniformemente en la escala verbal.

Se debe elaborar una matriz de juicios por cada criterio a tener en cuenta en la toma de decisión. A continuación se calcula la escala de proporción o *ratio scale*. La escala de proporción es un vector r en el que cada posición del vector contiene un valor proporcional a la entidad i en relación al criterio elegido. Des esta forma tendremos un vector de proporción por cada criterio elegido. Asimismo, se debe componer una matriz de juicios comparando los criterios elegidos en la toma de decisión y, a continuación, calcular la escala de proporción entre los criterios. Este vector nos permite ponderar cada criterio y así conocer qué criterios afectan más a la decisión. También se calcula un índice de inconsistencia o *inconsistency index* por cada matriz. Este índice nos proporciona una medida de cómo de lejos está nuestra estimación de la consistencia perfecta. Una matriz de juicios perfectamente consistente es aquella en la que todos sus elementos satisfacen $a_{ij} \times a_{jk} = a_{ik}\ \forall i, j, k$.

Como procedimiento para calcular la escala de proporción y el índice de inconsistencia, proponemos la utilización del modelo propuesto por Eduardo Miranda en [5], por sencillez y buenos resultados. Hay que calcular la media geométrica v_i de cada fila de la matriz de juicios definida para un determinado criterio. El valor de v_i viene dado por (3). La escala de proporción, denominada vector de valores propios o *eigenvalue*, r, consiste en un vector en el que cada valor se calcula aplicando (4).

$$v_i = \sqrt[n]{\prod_{j=1}^{n} a_{ij}} \tag{3}$$

$$r = [r_1,\ r_2,\ \ldots\ , r_n]\ con\ r_i = \frac{v_i}{\sum_{j=1}^{n} v_j} \tag{4}$$

El índice de inconsistencia se puede calcular de la siguiente forma (5).

$$CI = \frac{\sqrt{\sum_{i=1}^{n} \sum_{j>i}^{n} \left(ln\ a_{ij} \ - \ ln\ \frac{v_i}{v_j} \right)^2}}{\frac{(n-1)\times(n-2)}{2}} \tag{5}$$

A partir del vector de valores propios o *eigenvalue* y el valor real de la entidad de referencia, se puede calcular el valor absoluto de cada entidad. Para ello se debe aplicar la expresión (6). Aplicando estos cálculos con el resto de criterios, previamente ordenados según el porcentaje de contribución sobre el proyecto en su totalidad, se puede calcular el valor de cada alternativa.

$$Valor_i = \frac{r_i}{r_{refrencia}} \times Valor_{refrencia} \tag{6}$$

En el siguiente apartado, se muestran las características más relevantes de AHP . También se indican los principales problemas que presenta este método de estimación.

3.2. Características de AHP

Analytic Hierarchy Process es uno de los métodos de estimación más sencillos cuya mayor dificultad radica en identificar los atributos y su contribución relativa. Además, proporciona una visión del proyecto software jerarquizada, estructurada y sistemática. Asimismo, AHP es poco propenso a errores, permitiendo estimaciones precisas con hasta un 40 % de comparaciones erróneas. A pesar de este dato no se puede afirmar que AHP sea mejor que la estimación experta, ya que está basado, precisamente, en comparaciones hechas por expertos entre pares de tareas.

AHP aporta una notable ventaja para los expertos, ya que resulta más sencillo hacer comparaciones por parejas (entre los pares de tareas) que estimar cada tarea de una en una. Por otro lado, el número de comparaciones que deben realizar puede suponer un problema. Esto se debe a que la relación de las comparaciones a realizar y el número de tareas es de orden cuadrático. Si nuestro proyecto se compusiera de n tareas, el número de comparaciones que se deberían realizar sería de $n(n-1)/2$. En el supuesto de manejar 30 tareas, se deberán realizar 435 comparaciones. Para evitar esto, aún a riesgo de aumentar la homogeneidad de la matriz de juicios, se pueden agrupar las tareas similares en grupos más reducidos.

En las fases iniciales del desarrollo de un proyecto software suele darse una carencia de datos de referencia. El método de estimación AHP resulta muy útil en estas etapas iniciales, ya que como mínimo necesita un único dato de referencia: la tarea de referencia. Esta característica nos posibilita hacer estimaciones bastante precisas en fases tempranas del desarrollo de un proyecto software. Es importante que el porcentaje de contribución de la tarea de referencia al proyecto global sea lo más preciso posible. Esto aumentará la exactitud de las estimaciones del resto de las tareas. En caso contrario, podrá derivar en errores. La exactitud de

las predicciones también se verá mejorada con el uso de más de una tarea de referencia.

Con el método de estimación AHP puede darse el fenómeno del Rank Reversal o alteración del rango. Este fenómeno consiste en un cambio en el ranking relativo de las tareas, al introducir una nueva tarea o eliminar una de ellas. Los autores Ying-Ming Wang y Taha M.S. Elhag en [6], exponen la existencia de varias propuestas enfocadas a evitar el fenómeno de la alteración en el rango. Estas propuestas tienen en cuenta si la alternativa introducida o eliminada del conjunto de alternativas seleccionadas agrega o no información. En ese mismo artículo se expone la propuesta de los autores, en la que se preserva el ranking de las alternativas sin necesidad de variar los pesos de las alternativas ni el número de criterios. Para esto, los autores proponen la normalización del vector de valores propios o *eigenvalue*.

Un aspecto a tener en cuenta utilizando el método de estimación AHP radica en la escala elegida, tanto por su proporción como por el número de puntos que la constituyen. En AHP el éxito en las estimaciones reside en la exactitud de las comparaciones entre las tareas. Estas comparaciones, son consecuencia directa de la escala de evaluación elegida, así como del número de puntos de la escala. La escala propuesta por Thomas L. Saaty en [4], propone el uso de una escala que varía entre 1/9 y 9, lo que supone un total de 17 puntos en la escala. En [7] los autores aconsejan el uso de una escala con un número de etiquetas más bajo, ya que esto nos facilitará el manejo de la escala. Eduardo Miranda en [5] también propone una escala diferente. Ésta consta de menos puntos que la propuesta por Saaty, y según el autor, es más fácil de utilizar por los expertos.

4. AHP como ponderador

El modo en el que AHP realiza las comparaciones entre las diferentes tareas nos ofrece un buen recurso para establecer los pesos. El procedimiento es más sencillo que el propio algoritmo de AHP. Básicamente es suficiente con construir la matriz de comparaciones y a continuación calcular el vector de valores propios. Se trata de comparar las variables o los expertos unos con otros utilizando la matriz de comparaciones. Estas comparaciones nos sirven para capturar el juicio experto, como ya se ha comentado. El experto deberá responder a una pregunta que permita establecer las diferentes influencias de las variables o de los expertos. La pregunta sería de la forma *¿cuánta más (menos) importancia tiene la variable v_i comparado con la variable v_j sobre la variable global?* o *¿cuánto más (menos) relevante es la opinión del experto p_i comparado con el experto p_j en este dominio?*.

Las preguntas están formuladas de manera que se compara la influencia de cada variable respecto a la variable global. Una vez realizada las comparaciones con ayuda de la escala verbal, obtenemos el vector de valores propios o de proporción. Este vector establece una proporción entre las diferentes variables comparadas acorde al criterio establecido en la pregunta formulada. La influencia de una variable sobre otra global se puede interpretar como el peso de dicha

variable sobre la variable global. De esta forma, el vector de valores propios se puede interpretar del mismo modo como un vector de pesos. Por tanto, el vector de valores propios nos proporciona directamente los pesos de cada variable.

A continuación presentamos dos ejemplos donde podemos ver la aplicación del método.Veremos un ejemplo de cómo combinar diferentes opiniones de distintos expertos, y otro para mostrar cómo aunar estimaciones de diferentes variables en una más global.

4.1. Combinar diferentes opiniones de expertos

En esta sección aplicamos el método de combinación lineal de opiniones de varios expertos sobre una misma variable. El objetivo es obtener una estimación más confiable que si esta fuera llevada a cabo por un sólo experto. En este ejemplo contamos con tres expertos. Para poder aplicar el método de combinación necesitamos establecer el peso de cada experto. Con este fin, en nuestro ejemplo, se tendrá en cuenta la experiencia de cada miembro y la información de que disponga, tanto en términos de cantidad como de calidad:

- E_1: 10 años de experiencia, dispone se mucha información fiable
- E_2: 5 años de experiencia, dispone se poca información y es poco fiable
- E_3: 15 años de experiencia, dispone se mucha información poco fiable

A los expertos (E_i) se les asocia un peso utilizando el método A.H.P. como se ha explicado anteriormente. Las comparaciones se realizan acorde a la escala verbal propuesta por Saaty que podemos ver en la tabla 1. Las comparaciones entre los diferentes expertos las debe realizar otro experto con el fin de que sea lo más objetiva posible (ver figura 1).

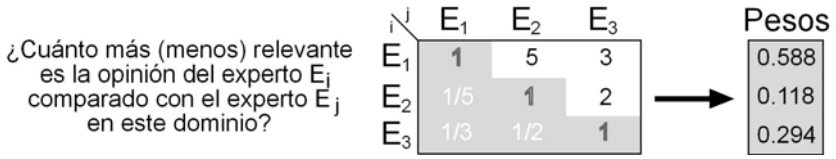


Figura 1. Matriz de juicios y vector de pesos

Los expertos serán los encargados de realizar las estimaciones de una variable aleatoria. En este ejemplo la variable aleatoria que utilizaremos será un factor comúnmente utilizado en el área de estimación de esfuerzo software: la *complejidad del software*. Lo primero que tienen que hacer los expertos es ponerse de acuerdo en lo que se entiende por *complejidad de software* y los valores que puede tomar. Suponemos que en nuestro ejemplo ya se han puesto de acuerdo y han establecido que la variable será discreta y contará con tres estados: *Baja*, *Media* y *Alta*. En la figura 2 podemos ver las distribuciones de probabilidad

de la variable *complejidad* realizada por los tres expertos con los que cuenta el ejemplo.

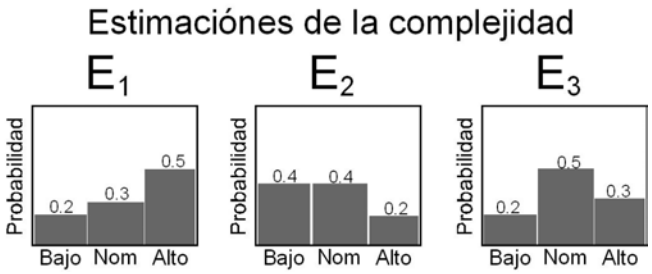


Figura 2. Estimaciones de los expertos sobre la variable *Complejidad*

A continuación lo único que nos queda por hacer es aplicar el método de combinación lineal de opiniones (ver formula 7) y obtener la distribución de probabilidad combinada que podemos ver en la figura 3.

$$p(\Theta) = \sum_{i=1}^{n} w_i p_i(\Theta)$$

$$p(Complejidad_{Baja}) = 0{,}588 * o{,}2 + 0{,}118 * 0{,}4 + 0{,}294 * 0{,}2 = 0{,}2236$$
$$p(Complejidad_{Media}) = 0{,}588 * o{,}3 + 0{,}118 * 0{,}4 + 0{,}294 * 0{,}5 = 0{,}3706$$
$$p(Complejidad_{Alta}) = 0{,}588 * o{,}5 + 0{,}118 * 0{,}2 + 0{,}294 * 0{,}3 = 0{,}4058 \quad (7)$$

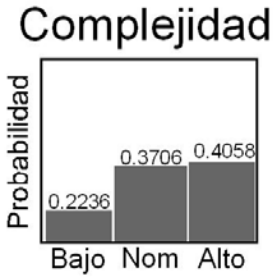


Figura 3. Distribución de probabilidad combinada de la *Complejidad*

4.2. Combinar diferentes variables para establecer una variable globalizadora

En este caso nos planteamos estimar una variable aleatoria a partir de la estimación de cada una de sus partes. Como ejemplo utilizaremos el esfuerzo de

desarrollo de un proyecto software y supondremos que el esfuerzo de desarrollo de un software se establece a partir del esfuerzo de desarrollo de cada una de sus partes (esto no es del todo cierto ya que depende de más factores que afectan al esfuerzo total, pero nos sirve como ejemplo). Pongamos como ejemplo un proyecto software que se compone de tres funcionalidades:

- F_1: Núcleo básico - módulos de seguridad, backup y gestión de usuarios
- F_2: Docu - módulo encargado de gestionar toda la documentación
- F_3: Reservas - módulo encargado de la gestión de reservas de automóviles

Hay que tener en cuenta que, como en este ejemplo, las distribuciones de probabilidad que queremos combinar deben ser del mismo tipo, en nuestro caso el esfuerzo necesario para desarrollar cada una de las funcionalidades. Lo primero que debemos hacer es establecer los pesos. En nuestro ejemplo los pesos nos indican la influencia relativa de cada funcionalidad en el esfuerzo total. Para esto crearemos la matriz de comparaciones. Las filas y las columnas de la matriz de comparaciones estarán formadas por las diferentes alternativas, en nuestro caso las funcionalidades F_1, F_2 y F_3. A continuación, con ayuda de un experto en el dominio, realizaremos las comparaciones entre cada una de las funcionalidades respondiendo a la pregunta: *¿Cuánto más (menos) influencia tiene la funcionalidad F_i sobre el esfuerzo comparándolo con la funcionalidad F_j?*. Ver figura 4.

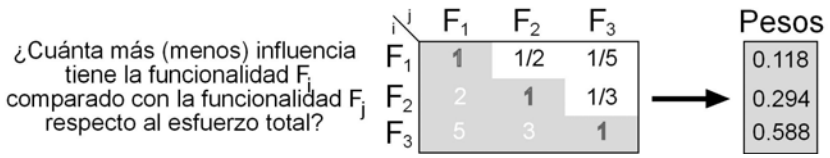


Figura 4. Matriz de juicios y vector de pesos

Para responder a estas comparaciones utilizaremos la escala verbal propuesta por Saaty que aparece en el cuadro 1. Podemos ver en la figura 4 las comparaciones realizadas por un experto así como el resultado del siguiente paso: calcular el vector de proporciones. Dado que este vector responde a la pregunta antes formulada que compara la influencia de cada funcionalidad sobre el esfuerzo total y sus valores suman uno, lo podemos interpretar como el peso de cada funcionalidad respecto al esfuerzo.

Una vez calculados los pesos procedemos a combinar las distribuciones de probabilidad de cada funcionalidad. Estas distribuciones representan una estimación del esfuerzo de desarrollo de cada funcionalidad. En nuestro ejemplo se trata de distribuciones de probabilidad discretas en tres estados: *Bajo*, *Nominal* y *Alto*. En la figura 5 podemos ver las distribuciones de probabilidad obtenidas a partir de un experto.

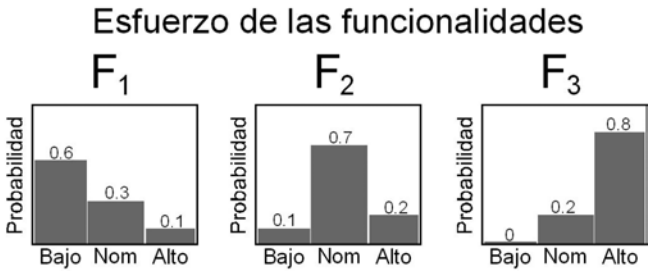


Figura 5. Distribuciones de probabilidad de las funcionalidades y el esfuerzo total

Los cálculos necesarios para establecer la distribución de probabilidad del esfuerzo total se muestran en 8. Se calcula por estados: se multiplica el peso de cada variable por la probabilidad del estado de dicha variable. Se suman y ya tenemos la probabilidad del estado de las variable global.

$$p(\Theta) = \sum_{i=1}^{n} w_i p_i(\Theta)$$

$$p(\Theta_{Bajo}) = 0{,}118 * o{,}6 + 0{,}294 * 0{,}1 + 0{,}588 * 0 = 0{,}1002$$
$$p(\Theta_{Nominal}) = 0{,}118 * o{,}3 + 0{,}294 * 0{,}7 + 0{,}588 * 0{,}2 = 0{,}3588$$
$$p(\Theta_{Alto}) = 0{,}118 * o{,}1 + 0{,}294 * 0{,}2 + 0{,}588 * 0{,}8 = 0{,}541 \quad (8)$$

Se trata de un método muy sencillo de calcular una vez tenemos los pesos de las variables. En la figura 6 podemos ver la distribución de probabilidad del esfuerzo total.

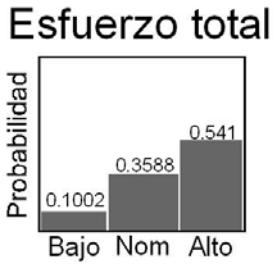


Figura 6. Distribuciones de probabilidad del esfuerzo total

5. Conclusiones y trabajo futuro

En este artículo se ha presentado una técnica de combinación de distribuciones de probabilidad. El método de combinación lineal de opiniones resulta ser

un método muy sencillo de aplicar y de entender, donde la mayor dificultad consiste en establecer los pesos de cada variable o experto. Para el establecimiento de dichos pesos consideramos que el método de ayuda a la toma de decisiones Analytic Hierarchy Process (A.H.P.)resulta apropiado para este fin, además de sencillo y fiable.

Se ha expuesto la combinación de ambos métodos desde dos puntos de vista bastante útiles dentro de la ingeniería del software. Por un lado se puede utilizar la combinación lineal de opiniones para combinar diferentes estimaciones de distintos expertos. Pero también se puede aplicar el mismo enfoque para establecer la estimación de una variable a partir de la combinación de la estimación de sus partes, por ejemplo entendiendo la estimación del esfuerzo de desarrollo de un software como la combinación de las estimaciones las funcionalidades que componen dicho software.

Agradecimientos: Este trabajo se ha desarrollado gracias a la financiación del proyecto TIN2004-06689-C03-01.

Referencias

1. Enrique Castillo, Jose M. Gutierrez, and Ali S. Hadi. *Expert Systems and Probabiistic Network Models*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.
2. Esperanza Manso and Jose Javier Dolado. *Técnicas cuantitativas para la gestión en la Ingeniería del software*. Netbiblio, 2007.
3. Clemen R.T. and Winkler R.L. Combining probability distributions from experts in risk analysis. *Risk Analysis*, 19:187–203(17), April 1999.
4. T.L. Saaty. How to make a decision - the analytic hierarchy process. *INTERFACES*, 24(6):19–43, NOV-DEC 1994.
5. Eduardo Miranda. Improving subjective estimates using paired comparisons. *IEEE Software*, 18(1):87–91, feb 2001.
6. Ying-Ming Wang and Taha M. S. Elhag. An approach to avoiding rank reversal in ahp. *Decis. Support Syst.*, 42(3):1474–1480, 2006.
7. Sahrman Barker Martin Shepperd and Martin Aylett. The analytic hierarchy process and data-less prediction. *Empirical Software Engineering Research Group ESERG: TR98-04*, 1998.

On the Correlation between Static Measures and Code Coverage using Evolutionary Test Case Generation

Javier Ferrer, Francisco Chicano, y Enrique Alba

Departamento de Lenguajes y Ciencias de la Computación
Universidad de Málaga, Spain
{ferrer,chicano,eat}@lcc.uma.es

Resumen. Evolutionary testing is a very popular domain in the field of search based software engineering that consists in automatically generating test cases for a given piece of code using evolutionary algorithms. One of the most important measures used to evaluate the quality of the generated test suites is code coverage. In this paper we want to analyze if there exists a correlation between some static measures computed on the test program and the code coverage when an evolutionary test case generator is used. In particular, we use Evolutionary Strategies (ES) as search engine of the test case generator. We have also developed a program generator that is able to create Java programs with the desired values of the static measures. The experimental study includes a benchmark of 3600 programs automatically generated to find correlations between the measures. The results of this study can be used in future work for the development of a tool that decides the test case generation method according to the static measures computed on a given program.

Palabras clave: Evolutionary testing, branch coverage, evolutionary algorithms, evolutionary strategy

1 Introduction

Automatic software testing is one of the most studied topics in the field of Search-Based Software Engineering (SBSE) [8]. From the first works [10] to nowadays many approaches have been proposed for solving the automatic test case generation problem. This great effort in building computer aided software testing tools is motivated by the cost and importance of the testing phase in the software development cycle. It is estimated that half the time spent on software project development, and more than half its cost, is devoted to testing the product [4]. This explains why the Software Industry and Academia are interested in automatic tools for testing.

Evolutionary algorithms (EAs) have been the most popular search algorithms for generating test cases [8]. In fact, the term *evolutionary testing* was coined to refer to this approach. In the paradigm of *structural testing* a lot of research has been performed using EAs and, in particular, different elements of the structure of a program have been studied in detail. Some examples are the presence of flags in conditions [2], the coverage of loops [5], the existence of internal states [19] and the presence of possible exceptions [16].

The objective of an automatic test case generator used for structural testing is to find a test case suite that is able to cover all the software elements. These elements can be instructions, branches, atomic conditions, and so on. The performance of an automatic test case generator is usually measured as the percentage of elements that the generated test suite is able to cover in the test program. This measure is called *coverage*. The coverage obtained depends not only on the test case generator, but also on the program being tested. Then, we can ask the following research questions:

- *RQ1*: Is there any static measure of the test program having a clear correlation with the coverage percentage?
- *RQ2*: Which are these measures and how they correlate with coverage?

As we said before, coverage depends also on the test case generator. Then, in order to completely answer the questions we should use all the possible automatic test case generators or, at least, a large number of them. We can also focus on one test case generator and answer to the previous questions on this generator. This is what we do in this paper. In particular, we study the influence on the coverage of a set of static software measures when we use an evolutionary test case generator. This study can be used to predict the value of a dynamic measure, coverage, from static measures. This way, it is possible to develop tools taking into account the static measures to decide which automatic test case generation method is more suitable for a given program.

The rest of the paper is organized as follows. In the next section we present the measures that we use in our study. Then, we detail the evolutionary test case generator used in Section 3. After that, Section 4 describes the experiments performed and discusses the results obtained. Finally, in Section 5 some conclusions and future work are outlined.

2 Measures

The measures used in this study are six: number of sentences, number of atomic conditions per condition, total number of conditions, nesting degree, coverage, and McCabe's cyclomatic complexity. The three first measures are easy to understand. The nesting degree is the maximum number of conditional statements that are nested one inside another. In the following paragraphs we describe in more detail the coverage and the McCabe's cyclomatic complexity.

In order to define a coverage measure, we first need to determine which kind of element is going to be "covered". Different coverage measures can be defined depending on the kind of element to cover. *Statement coverage*, for example, is defined as the percentage of statements that are executed. In this work we use *branch coverage*, which is the percentage of branches exercised in a program. This coverage measure is used in most of the related papers in the literature.

Cyclomatic complexity is a complexity measure of code related to the number of ways there are to traverse a piece of code. This determines the minimum number of inputs needed to test all the ways to execute the program. Cyclomatic complexity is computed using the control flow graph of the program: the nodes of the graph correspond to indivisible groups of sentences of a program, and a directed edge connects two nodes if the second sentence might be executed immediately after the first sentence. Cyclomatic complexity may also be applied to individual functions, modules, methods or classes within a program and is formally defined as follows:

$$v(G) = E - N + 2P; \tag{1}$$

where E is the number of edges of the graph, N is the number of nodes of the graph and P is the number of connected components.

In Figure 1, we show an example of control flow graph (G). It is assumed that each node can be reached by the entry node and each node can reach the exit node. The maximum number of linearly independent circuits in G is 9-6+2=5, and this is the cyclomatic complexity.

The correlation between the cyclomatic complexity and the number of software faults has been studied in some research articles [3, 7]. Most such studies find a strong positive correlation between the cyclomatic complexity and the defects: the higher the complexity

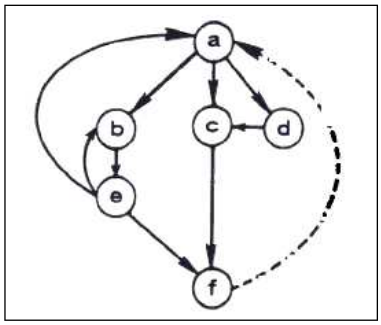


Fig. 1. The original graph of the McCabe's article

the larger the number of faults. For example, a 2008 study by metric-monitoring software supplier Energy [6], analyzed classes of open-source Java applications and divided them into two sets based on how commonly faults were found in them. They found strong correlation between cyclomatic complexity and their faultiness, with classes with a combined complexity of 11 having a probability of being fault-prone of just 0.28, rising to 0.98 for classes with a complexity of 74.

In addition to this correlation between complexity and errors, a connection has been found between complexity and difficulty to understand software. Nowadays, the subjective reliability of software is expressed in statements such as "I understand this program well enough to know that the tests I have executed are adequate to provide my desired level of confidence in the software". For that reason, we make a hard link between complexity and difficulty of discovering errors.

Since McCabe proposed the cyclomatic complexity, it has received several criticisms. Weyuker [18] concluded that one of the obvious intuitive weaknesses of the cyclomatic complexity is that it makes no provision for distinguishing between programs which perform very little computation and those which perform massive amounts of computation, provided that they have the same decision structure. Piwowarski [12] noticed that cyclomatic complexity is the same for N nested `if` statements and N sequential `if` statements.

In connection with our research questions, Weyuker's critic is not relevant, since coverage does not take into account the amount of computation made by a block of statements. However, Piworarski's critic is important in our research because the nesting degree of a program is inverse correlated with the branch coverage as we will show in the experimental section.

3 Test Case Generator

Our test case generator breaks down the global objective (to cover all the branches) into several partial objectives consisting of dealing with only one branch of the program. Then, each partial objective can be treated as a separate optimization problem in which the function to be minimized is a distance between the current test case and one satisfying the partial objective. In order to solve such minimization problem EAs are used. The main loop of the test data generator is shown in Fig. 2.

In a loop, the test case generator selects a partial objective (a branch) and uses the optimization algorithm to search for test cases exercising that branch. When a test case covers a branch, the test case is stored in a set associated to that branch. The structure composed of the sets associated to all the branches is called *coverage table*. After the optimization algorithm stops, the main loop starts again and the test case generator selects a different

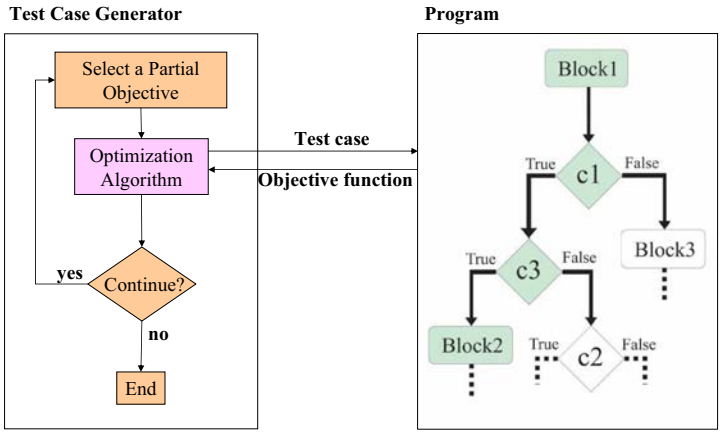


Fig. 2. The test case generation process

branch. This scheme is repeated until total branch coverage is obtained or a maximum number of consecutive failures of the optimization algorithm is reached. When this happens the test data generator exits the main loop and returns the sets of test cases associated to all the branches. In the rest of this section we describe two important issues related to the test case generator: the objective function to minimize and the optimization algorithm used.

3.1 Objective Function

Following on from the discussion in the previous section, we have to solve several minimization problems: one for each branch. Now we need to define an objective function (for each branch) to be minimized. This function will be used for evaluating each test case, and its definition depends on the desired branch and whether the program flow reaches the branching condition associated to the target branch or not. If the condition is reached we can define the objective function on the basis of the logical expression of the branching condition and the values of the program variables when the condition is reached. The resulting expression is called *branch distance* and can be defined recursively on the structure of the logical expression. That is, for an expression composed of other expressions joined by logical operators the branch distance is computed as an aggregation of the branch distance applied to the component logical expressions. For the Java logical operators `&` and `|` we define the branch distance as[1]:

$$bd(a\&b) = bd(a) + bd(b) \tag{2}$$

$$bd(a|b) = \min(bd(a), bd(b)) \tag{3}$$

where a and b are logical expressions.

In order to completely specify the branch distance we need to define its value in the base case of the recursion, that is, for atomic conditions. The particular expression used for the branch distance in this case depends on the operator of the atomic condition. The operands of the condition appear in the expression. A lot of research has been devoted in the past to the study of appropriate branch distances in software testing. An accurate branch distance taking into account the value of each atomic condition and the value of its operands can better guide the search. In procedural software testing these accurate functions

[1] These operators are the Java *and*, *or* logical operators without shortcut evaluation. For the sake of clarity we omit here the definition of the branch distance for other operators.

are well-known and popular in the literature. They are based on distance measures defined for relational operators like $<$, $>$, and so on [9]. We use these distance measures used in the literature.

When a test case does not reach the branching condition of the target branch we cannot use the branch distance as objective function. In this case, we identify the branching condition c whose value must first change in order to cover the target branch (critical branching condition) and we define the objective function as the branch distance of this branching condition plus the *approximation level*. The approximation level, denoted here with $ap(c, b)$, is defined as the number of branching nodes lying between the critical one (c) and the target branch (b) [17].

In this paper we also add a real valued penalty in the objective function to those test cases that do not reach the branching condition of the target branch. With this penalty, denoted by p, the objective value of any test case that does not reach the target branching condition is higher than any test case that reaches the target branching condition. The exact value of the penalty depends on the target branching condition and it is always an upper bound of the target branch distance. Finally, the expression for the objective function is as follows:

$$f_b(x) = \begin{cases} bd_b(x) & \text{if } b \text{ is reached by } x \\ bd_c(x) + ap(c,b) + p & \text{otherwise} \end{cases} \quad (4)$$

where c is the critical branching condition, and bd_b, bd_c are the branch distances of branching conditions b and c.

Nested branches pose a great challenge for the search. For example, if the condition associated to a branch is nested within three conditional statements, all the conditions of these statements must be true in order for the program flow to proceed onto the next one. Therefore, for the purposes of computing the objective function, it is not possible to compute the branch distance for the second and third nested conditions until the first one is true. This gradual release of information might cause efficiency problems for the search (what McMinn calls the *nesting problem* [11]), which forces us to concentrate on satisfying each predicate sequentially.

In order to alleviate the nesting problem, the test case generator selects as objective in each loop one branch whose associated condition has been previously reached by other test cases stored in the coverage table. Some of these test cases are inserted in the initial population of the EA used for solving the optimization problem. The percentage of individuals introduced in this way in the population is called the *replacement factor* and is denoted by Rf. At the beginning of the generation process some random test cases are generated in order to reach some branching conditions.

3.2 Optimization Algorithm

EAs [1] are metaheuristic search techniques loosely based on the principles of natural evolution, namely, adaptation and survival of the fittest. These techniques have been shown to be very effective in solving hard optimization tasks. They are based on a set of tentative solutions (individuals) called *population*. The problem knowledge is usually enclosed in an objective function, the so-called *fitness function*, which assigns a quality value to the individuals. In Fig. 3 we show the main loop of an EA.

Initially, the algorithm creates a population of μ individuals randomly or by using a seeding algorithm. At each step, the algorithm applies stochastic operators such as selection, recombination, and mutation (we call them variation operators in Fig. 3) in order to compute a set of λ descendant individuals P'(t). The objective of the selection operator is to select some individuals from the population to which the other operators will be applied. The

```
t := 0;
P(t) = Generate ();
Evaluate (P(t));
while not StopCriterion do
   P'(t) := VariationOps (P(t));
   Evaluate (P'(t));
   P(t+1) := Replace (P'(t),P(t));
   t := t+1;
endwhile;
```

Fig. 3. Pseudocode of an EA

recombination operator generates a new individual from several ones by combining their solution components. This operator is able to put together good solution components that are scattered in the population. On the other hand, the mutation operator modifies one single individual and is the source of new different solution components in the population. The individuals created are evaluated according to the fitness function. The last step of the loop is a replacement operation in which the individuals for the new population $P(t+1)$ are selected from the offspring $P'(t)$ and the old one $P(t)$. This process is repeated until a stop criterion is fulfilled, such as reaching a pre-programmed number of iterations of the algorithm or finding an individual with a preset target quality. In the following we focus on the details of the specific EAs used in this work to perform the test case generation.

We used an Evolutionary Strategy (ES) for the search of test cases for a given branch. In an ES [14] each individual is composed of a vector of real numbers representing the problem variables ($\mathbf{x}$), a vector of standard deviations (σ) and, optionally, a vector of angles (ω). These two last vectors are used as parameters for the main operator of this technique: the Gaussian mutation. They are evolved together with the problem variables themselves, thus allowing the algorithm to self-adapt the search to the landscape. For the recombination operator of an ES there are many alternatives: each of the three real vectors of an individual can be recombined in a different way. However, this operator is less important than the mutation. The mutation operator is governed by the three following equations:

$$\sigma_i' = \sigma_i \exp(\tau N(0,1) + \eta N_i(0,1)) \ , \tag{5}$$

$$\omega_i' = \omega_i + \varphi N_i(0,1) \ , \tag{6}$$

$$\mathbf{x}' = \mathbf{x} + \mathbf{N}(\mathbf{0}, C(\sigma', \omega')) \ , \tag{7}$$

where $C(\sigma', \omega')$ is the covariance matrix associated to σ' and ω', $N(0,1)$ is the standard univariate normal distribution, and $\mathbf{N}(\mathbf{0}, C)$ is the multivariate normal distribution with mean $\mathbf{0}$ and covariance matrix C. The subindex i in the standard normal distribution indicates that a new random number is generated anew for each component of the vector. The notation $N(0,1)$ is used for indicating that the same random number is used for all the components. The parameters τ, η, and φ are set to $(2n)^{-1/2}$, $(4n)^{-1/4}$, and $5\pi/180$, respectively, as suggested in [15]. With respect to the replacement operator, there is a special notation to indicate wether the old population is taken into account or not to form the new population. When only the new individuals are used, we have a (μ, λ)-ES; otherwise, we have a $(\mu + \lambda)$-ES.

Regarding the representation, each component of the vector solution $\mathbf{x}$ is rounded to the nearest integer and used as actual parameter of the method under test. There is no limit in the input domain, thus allowing the ES to explore the whole solution space. This contrasts with other techniques such as genetic algorithm with binary representation that can only explore a limited region of the search space.

In the experimental section we have used two ESs: a (1+5) ES without crossover, that we call (1+5)-ES, and a (25+5)-ES with uniform crossover, called (25+5)-ES$_c$. In the latter algorithm, random selection was used.

4 Experimental Section

In order to study the correlations between the static measures and the coverage, we first need a large number of test programs. For the study to be well-founded, we require a lot of programs having the same value for the static measures as well as programs having different values for the measures. It is not easy to find such a variety of programs in the related literature. Thus, we decided to automatically generate the programs. This way, it is possible to randomly generate programs with the desired values for the static measures and, most important, we can generate different programs with the same values for the static measures.

The automatic program generation raises a non-trivial question: are the generated programs realistic? That is, could them be found in real-world? Using automatic program generation it is not likely to find programs that are similar to the ones who a programmer would make. This is especially true if the program generation is not driven by a specification. However, this is not a drawback in our study, since we want to analyze the correlations between some static measures of the programs and code coverage. In this situation, "realistic programs" means programs that have similar values for the considered static measures as the ones found in real-world; and we can easily fulfil this requirement.

Our program generator takes into account the desired values for the number of atomic conditions per condition, the nesting degree, the number of sentences and the number of variables. With these parameters and other (less important) ones, the program generator creates a program with a defined control flow graph containing several conditions. The main features of the generated programs are:

- They deal with integer input parameters.
- Their conditions are joined by whichever logical operator.
- They are randomly generated.

Due to the randomness of the generation, the static measures could take values that are different from the ones specified in the configuration file of the program generator. For this reason, in a later phase, we used the free tool CyVis to measure the actual values for the static measures. CyVis [13] is a free software tool for metrics collection, analysis and visualization of Java based programs.

The methodology applied for the program generation is the following. First, we analyzed a set of Java source files from the JDK 1.5, in particular, the package `java.util`; and we computed the static measures on these files. In Table 1 we show a summary of this analysis. Next, we used the ranges of the most interesting values obtained in this previous analysis as a guide to generate Java source files having values in the same range for the static measures. This way, we generate programs that are realistic with respect to the static measures, making the following study meaningful. Finally, we generated a total of 3600 Java programs using our program generator and we applied our test case generator with (1+5)-ES and (25+5)-ES$_c$ to all of them 5 times. We need 5 independent runs of each algorithm because they are stochastic algorithms: one single run is not meaningful; instead, several runs are required and the average and the standard deviation are used for comparison purposes. The experimental study requires a total of $3600 \cdot 5 \cdot 2 = 36000$ independent runs of the test case generator.

4.1 Results

After the execution of all the independent runs for the two algorithms in the 3600 programs, in this section we analyze the linear correlation between the static measures and the coverage.

Tabla 1. Range of values obtained in java.util library

Parameter	Minimum	Maximum
Number of Sentences	10	294
Nesting Degree	1	7
McCabe Cyclomatic Complexity	1	80

We use the Pearson correlation coefficient to study the degree of linear correlation between two variables. This coefficient is usually represented by r, and is computed with the following formula:

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{(n-1)S_x S_y}$$

where x_i and y_i are the values of the samples, n is the number of cases, S_x and S_y are the standard deviations of each variable.

First, we study the correlation between the number of sentences and the branch coverage. We obtain a correlation of 13.4% for these two variables using the (25+5)-ES$_c$ algorithm and 18.8% with the (1+5)-ES algorithm[2]. In Figure 4 we plot the average coverage against the number of sentences for ES and all the programs. It can be observed that the number of sentences is not a significant parameter and it has no influence in the coverage measure. The results obtained with ES$_c$ are similar and we omit the corresponding graph.

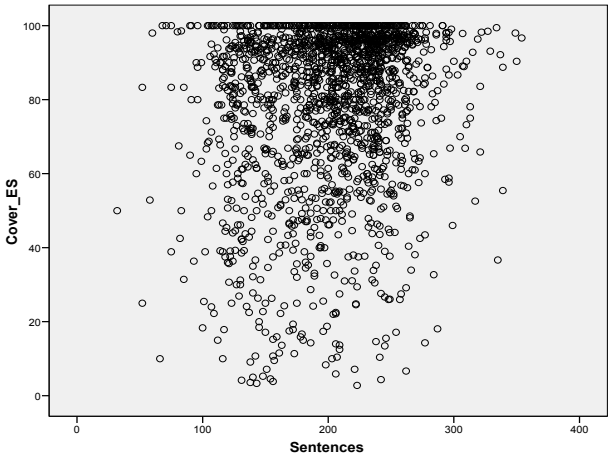


Fig. 4. Average branch coverage against the number of sentences for ES in all the programs

In second place, we study the correlation between the number of atomic conditions per condition and coverage. In Table 2 we show the average coverage obtained for all the programs with the same number of atomic conditions per condition when ES and ES$_c$ are used. From the results we conclude that there is no linear correlation between these two variables. The minimum values for coverage are reached with 1 and 7 atomic conditions per condition. This could seem counterintuitive, but a large condition with a sequence of

[2] In order to save room, in the following we use ES$_c$ and ES to refer to (25+5)-ES$_c$ and (1+5)-ES, respectively

logical operators, can be easily satisfied due to OR operators. Otherwise, a short condition composed of AND operators can be more difficult to satisfy.

Tabla 2. Correlation between the number of atomic conditions per condition and average coverage for both algorithms

At. Conds.	ES_c	ES
1	$83.59\%_{21.69}$	$77.74\%_{23.49}$
2	$82.85\%_{20.33}$	$78.54\%_{21.82}$
3	$85.15\%_{19.82}$	$81.39\%_{21.53}$
4	$\mathbf{88.04}\%_{16.42}$	$\mathbf{83.23}\%_{19.87}$
5	$85.06\%_{19.19}$	$80.50\%_{21.53}$
6	$84.16\%_{19.58}$	$79.12\%_{23.09}$
7	$81.24\%_{21.18}$	$76.26\%_{23.74}$
r	-0.50 %	0.30 %

Now we analyze the influence on coverage of total number of conditions of a program. In Figure 5, we can observe that programs with a small number of conditions reach a higher coverage than the programs with a large number of conditions. This could be interpreted as large programs with many conditions are more difficult to test. If there are a lot of conditions, this generates a lot of different paths in the control flow graph, this fact is also weighted in the cyclomatic complexity. The correlation coefficient is -16.4% for ES and -21.6% for ES_c.

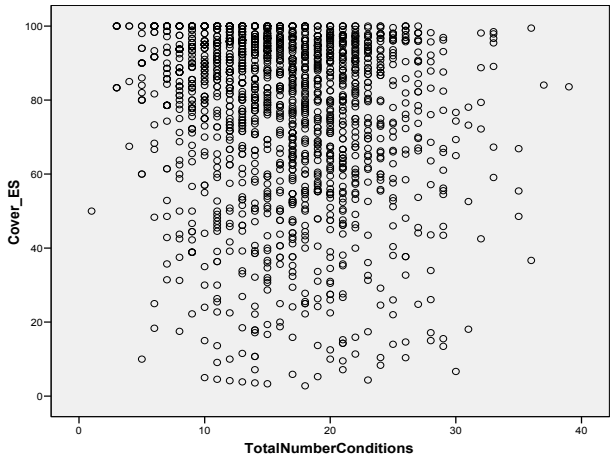


Fig. 5. Average branch coverage against the total number of conditions for ES in all the programs

Let us analyze the nesting degree. In Table 3, we summarize the coverage obtained with different nesting degree. If the nesting degree is increased, the branch coverage decreases and vice versa. It is clear that there is an inverse correlation between these variables. The correlation coefficients are -45.7% and -47% for ES and ES_c respectively, what confirms the observations. As we said in Section 3.1, nested branches pose a great challenge for the search.

Tabla 3. Correlation between nesting degree and average coverage for both algorithms

Nesting	ES_c	ES
1	**96.03**$\%_{5.74}$	**93.56**$\%_{8.15}$
2	94.07$\%_{8.85}$	89.97$\%_{11.29}$
3	90.02$\%_{13.67}$	85.29$\%_{16.71}$
4	85.06$\%_{16.43}$	80.09$\%_{19.18}$
5	77.83$\%_{22.53}$	72.02$\%_{23.90}$
6	73.02$\%_{24.80}$	67.47$\%_{24.66}$
7	64.45$\%_{25.96}$	58.41$\%_{27.16}$
r	-47,00 %	-45,70 %

Finally, we study the correlation between the McCabe cyclomatic complexity and coverage. In Figures 6 and 7, we plot the average coverage against the cyclomatic complexity for ES and ES_c in all the programs. In general we can observe that there is no clear correlation between both parameters. The correlation coefficients are -4.8% and -8.6% for ES and ES_c, respectively. These values are very low, and confirms the observations: McCabe's cyclomatic complexity and branch coverage are not correlated. This is somewhat surprising, we would expect a positive correlation between the complexity of a program and the difficulty to get an adequate test suite. However, this is not true: McCabe's cyclomatic complexity cannot be used as a measure of the difficulty to get an adequate test suite. We can go one step forward and try to explain this unexpected behaviour. Cyclomatic complexity is not correlated with branch coverage because complexity does not take into account the nesting degree, which is the parameter with a higher influence on branch coverage.

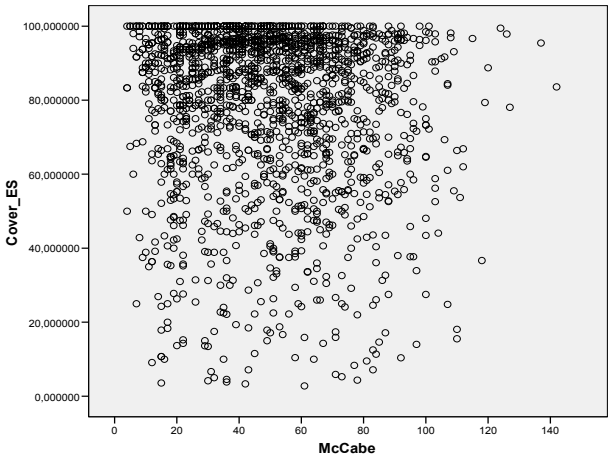


Fig. 6. Average branch coverage against the McCabe's cyclomatic complexity for ES in all the programs

5 Conclusions

In this work we have analyzed the correlation between the branch coverage obtained using automatically generated test suites and five static measures: number of sentences, number

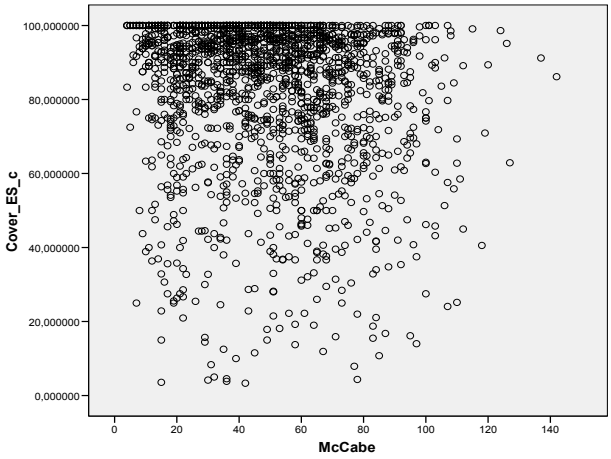


Fig. 7. Average branch coverage against the McCabe's cyclomatic complexity for ES_c in all the programs

of atomic conditions per condition, number of total conditions, nesting degree and McCabe's cyclomatic complexity. The results show that there is a small correlation between the branch coverage and the number of sentences, the number of conditions per conditions and the number of total conditions. On the other hand, the nesting degree is the measure that is more (inverse) correlated to the branch coverage: the higher the nesting degree the lower the coverage. Finally, there is no correlation between McCabe's cyclomatic complexity and branch coverage, that is, cyclomatic complexity does not reflect the difficulty of the automatic generation of test suites.

As future work we plan to advance in the analysis of static and dynamic measures of a program in order to propose a reliable measure of the difficulty of generating adequate test suites. In addition, we want to make an exhaustive study of the static measures in object-oriented programs, using a larger number of static measures. Finally, we would like to design new static measures able to reflect the real difficulty of automatic testing for specific test case generators.

Acknowledgements

This work has been partially funded by the Spanish Ministry of Science and Innovation and FEDER under contract TIN2008-06491-C04-01 (the M* project). It has also been partially funded by the Andalusian Government under contract P07-TIC-03044 (DIRICOM project).

References

1. T. Bäck, D. B. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation.* Oxford University Press, New York NY, 1997.
2. André Baresel, David Wendell Binkley, Mark Harman, and Bogdan Korel. Evolutionary testing in the presence of loop–assigned flags: A testability transformation approach. In *International Symposium on Software Testing and Analysis (ISSTA 2004)*, pages 108–118, 2004.
3. Victor Basili and Barry Perricone. Software errors and complexity: an empirical investigation. *ACM commun*, 27(1):42–52, 1984.
4. Boris Beizer. *Software testing techniques.* Van Nostrand Reinhold Co., New York, NY, USA, 2nd edition, 1990.

5. Eugenia Díaz, Raquel Blanco, and Javier Tuya. Tabu search for automated loop coverage in software testing. In *Proceedings of the International Conference on Knowledge Engineering and Decision Support (ICKEDS)*, pages 229–234, Porto, 2006.
6. Mark Dixon. An objective measure of code quality. *Technical Report*, February 2008.
7. T.M. Khoshgoftaar and J.C. Munson. Predicting software development errors using software complexity metrics. *IEEE Journal on Selected Areas in Communications*, 1990.
8. Phil McMinn. Search-based software test data generation: a survey. *Software Testing, Verification and Reliability*, 14(2):105–156, June 2004.
9. Christoph C. Michael, Gary McGraw, and Michael A. Schatz. Generating software test data by evolution. *IEEE Transactions on Software Engineering*, 27(12):1085–1110, December 2001.
10. Webb Miller and David L. Spooner. Automatic generation of floating-point test data. *IEEE Trans. Software Eng.*, 2(3):223–226, 1976.
11. David Binkley Phil McMinn and Mark Harman. Testability transformation for efficient automated test data search in the presence of nesting. *Proceedings of the Third UK Software Testing Workshop 2005*, pages 165–182, 2005.
12. Paul Piwarski. A nesting level complexity measure. *SIGPLAN*, 17(9):44–50, 1982.
13. Vinay Iyer Pradeep Selvaraj. Cyvis. Sourceforge, 2005.
14. I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Fromman-Holzboog Verlag, Stuttgart, 1973.
15. G. Rudolph. *Evolutionary Computation 1. Basic Algorithms and Operators*, volume 1, chapter 9, Evolution Strategies, pages 81–88. IOP Publishing Lt, 2000.
16. Nigel Tracey, John Clark, Keith Mander, and John McDermid. Automated test-data generation for exception conditions. *Software Practice and Experience*, 30(1):61–79, 2000.
17. Joachim Wegener, Andre Baresel, and Harmen Sthamer. Evolutionary test environment for automatic structural testing. *Information and Software Technology*, 43(14):841–854, December 2001.
18. E.J. Weyuker. Evaluating software complexity measures. *IEEE Trans. Software Eng.*, 14(9):1357–1365, 1988.
19. Yuan Zhan and John A. Clark. The state problem for test generation in simulink. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1941–1948. ACM Press, 2006.

Una propuesta usando Restricciones para la toma de decisiones en la tolerancia a fallos en procesos de negocio

M. T Gómez-López and Rafael Martínez Gasca and Diana Borrego Núñez

Departamento de Lenguajes y Sistemas Informáticos, Escuela Técnica Superior de Ingeniería Informática, Universidad de Sevilla, España,
{maytegomez, gasca, dianabn}@us.es

Resumen Cuando un proceso de negocio no obtiene el objetivo que se propone será necesario realizar una diagnosis y detección de errores, descubriendo qué servicios están funcionando de forma incorrecta para su sustitución. El objetivo de este artículo es describir los pasos necesarios para buscar otro servicio o actividad que pueda sustituir al que está fallando de una forma eficiente. Para automatizar la búsqueda y sustitución de servicios, es posible describir la funcionalidad de dichos procesos mediante restricciones, lo cual facilitaría la identificación de sus posibles sustitutos. También se analiza en este artículo cómo adaptar el traspaso de información que se realiza entre los servicios mediante mensajes en XML, a la descripción del comportamiento de dichos servicios mediante restricciones.

1. Introduction

Los procesos de negocio son un conjunto de actividades o servicios relacionados mediante un flujo de trabajo y/o datos para obtener un objetivo definido. Cuando un proceso de negocio se somete a una monitorización y diagnosis, pueden ser detectados errores en algunas de sus actividades. En este trabajo nos centraremos en los procesos de negocio que están formados por un conjunto de servicios web, donde uno de ellos falla para cualquier instancia de ejecución, o su funcionamiento es defectuoso.

La diagnosis de fallos ha sido una cuestión analizada en otros trabajos [1], ya que la conversión de las técnicas de inteligencia artificial [2][3] al modelado mediante procesos de negocio no son automáticas. Uno de los mayores problemas de la diagnosis de procesos viene derivada de que el modelo está distribuido y no se tiene una visión ni conocimiento global de su comportamiento.

Más relacionado con la diagnosis de procesos de negocio existen trabajos relacionados con la detección de conflictos (CDM - Conflict Detecting Mechanism), donde la especificación de los servicios descrita en XML se utiliza para diseñar meta-procesos [4] para la detección de inconsistencias en las actividades. En el ámbito de CDM, los conflictos se definen como la causa que viola y modifica el comportamiento normal o esperado en un determinado estado de la ejecución

del proceso de negocio. Para la detección de estos errores, el área de procesos de negocio también se ayuda de la monitorización de procesos (BPMod-Business Process Monitoring), que es un lenguaje de consulta para la monitorización propuesto en [5].

Cuando los procesos de negocios están formados por servicios web, es muy utilizado el estándar BPEL (Business Process Execution Language [6]), que provee de un lenguaje basado en XML, el cual describe tanto la interfaz del proceso como sus operaciones lógicas y su línea de ejecución. La utilización de este lenguaje también ha sido llevaba al campo de la tolerancia a fallos [7].

Una vez detectado el servicio que ha fallado, será necesario decidir qué tareas se tendrán que llevar a cabo dependiendo de qué servicio está fallando. Para sustituir un servicio, se tendrá que encontrar otro que incluya su especificación y preferiblemente en el menor tiempo posible. Para automatizar este proceso, es necesario conocer y tener especificado de una manera formal su comportamiento, junto a un repositorio de posibles servicios sustitutos. Una forma de describir dicho funcionamiento es mediante restricciones, que permiten la formalización de contratos y existen técnicas para optimizar la búsqueda de servicios consistentes con otros. La utilización de restricciones facilitaría la toma de decisiones ante la búsqueda de un sustituto, ya que existen técnicas desarrolladas en este ámbito [8][9] para optimizar la localización de restricciones mediante consultas a una Base de Datos de Restricciones.

La búsqueda de un nuevo servicio o un conjunto de ellos en una base de datos conlleva tres problemas fundamentales a solucionar:

- Representación de los servicios mediante un lenguaje de especificación cercano a las restricciones que pueda describir su comportamiento.
- Almacenamiento de dicha descripción en una base de datos para hacer la información persistente y facilitar una búsqueda eficiente.
- Búsqueda eficiente del mejor servicio, o combinación de ellos, para sustituir el detectado como incorrecto.

A lo largo de este artículo, vamos a analizar estas tres tareas y cómo mejorarlas para que la combinación de ellas ayuden a la toma de decisiones en la orquestación de los procesos ante la detección de errores.

2. Representación de servicios web mediante restricciones

El estándar BPEL (Business Process Execution Language [6]) permite describir tanto la interfaz del proceso como sus operaciones lógica y su línea de ejecución. La notación que describe BPEL se basa en la descripción del comportamiento específico de procesos basados en servicios web, que se puede encontrar en la bibliografía como BPEL4WS. Los procesos en BPEL4WS exportan e importan funcionalidad usando exclusivamente las interfaces de los servicios web. Los procesos de negocio se pueden describir de dos formas:

- Modelos de procesos de negocio ejecutables con un determinado comportamiento.

- Protocolos de negocio, usando descripción de negocios que especifican el intercambio mutuo de mensajes sin revelar su comportamiento interno. La descripción de los protocolos de negocio se denomina descripción de *procesos abstractos*.

El lenguaje BPEL4WS se puede usar para modelar tanto los procesos ejecutables como los abstractos, definiendo un lenguaje tanto para la especificación formal como para el intercambio de mensajes para un protocolo determinado.

Las siguientes especificaciones describen el espacio de los Servicios Web: *SOAP*, *Web Services Description Language (WSDL)*, y *Universal Description, Discovery, and Integration (UDDI)*. SOAP define un protocolo de mensajería en XML para la interacción entre servicios básicos. WSDL introduce una gramática para la descripción de servicios. UDDI provee de la infraestructura necesaria para publicar y descubrir servicios de una forma sistemática. Estas tres especificaciones combinadas permiten una funcionalidad completa en los servicios web con un modelo independiente de la plataforma.

Cuando en este artículo se hace referencia al término *Restricción*, nos referimos a un conjunto de variables definidas sobre un dominio y relacionadas entre sí. Donde dicha relación entre variables se puede describir de una manera compacta mediante una combinación con operadores lógicos de ecuaciones e inecuaciones.

La implementación M de un servicio en un determinado proceso tiene que satisfacer un Contrato C descrito mediante una precondición y una postcondición, lo que se puede representar mediante restricciones que se deben cumplir al terminar el servicio. También hay que tener en cuenta que dicho servicio tendrá un conjunto de variables de conexión que serán públicas y estarán relacionadas con otros servicios mediante puertos, mientras que otro conjunto de variables serán privadas. Si M satisface el contrato C y están definidas sobre los mismos puertos, se dirá que $M \subseteq C$. Para sustituir M por otro servicio, será necesario encontrar otra restricción M' que está definida sobre los mismos puertos y que también satisfaga el aserto $M' \subseteq C$.

Para mostrar un ejemplo de representación de servicios web con restricciones utilizaremos el mostrado en la figura 1. Este ejemplo utiliza un conjunto de servicios que representan la obtención de ganancias haciendo distintos tipos de inversiones. Partiendo de una cantidad determinada se utilizará un servicio (*División Capital en distintas Estrategias*) que desglosará la cantidad disponible en distintas partidas. Las distintas estrategias son: *Inversión en Bolsa*, *Inversión Inmobiliaria*, e *Inversión en empresas de I+D+I*. Cada una de estos servicios obtendrá unos beneficios en función de la cantidad invertida y un margen de inversión a dicha cantidad. Esto significa que cada servicio recibirá un par de valores <inversión, margen>, de forma que la inversión total en cada uno de los servicios será de inversión±margen. Como variable de salida del servicio se obtendrá una gananciaTotal. Si por ejemplo tras un proceso de diagnosis se detecta que el servicio *Inversión en Bolsa* no funciona de manera correcta, será necesario buscar un nuevo servicio que haga las veces de éste. Para esto es necesario tener la descripción de dicho servicio, de forma que sea posible la sustitución por otro. Un ejemplo de la descripción de la tarea *Inversión en Bolsa* es la sigu-

iente, donde se describen el precio de cada una de las acciones disponible en este servicio PA_1, ..., PA_n, la cantidad disponibles de cada una de ellas C_1, ..., C_n, y el porcentaje de ganancia tras un determinado tiempo en cada una de ellas $PG_1 \ldots PG_n$. Toda esta información será obtenida por el servicio de forma independiente. Para una cantidad de inversión dada y un determinado margen, se comprarán un conjunto de acciones, tantas de cada tipo como se representan con las variables N_1, ..., N_n. La descripción del comportamiento del servicio será:

$$N_1{*}PG_1 + N_2{*}PG_2 + \ldots N_n{*}PG_n = \mathsf{gananciaTotal}$$
$$\wedge$$
$$PA_1{*}N_1 + PA_2{*}N_2 + \ldots PA_n{*}N_n \leq \mathsf{inversión} + \mathsf{margen}$$
$$\wedge$$
$$PA_1{*}N_1 + PA_2{*}N_2 + \ldots PA_n{*}N_n \geq \mathsf{inversión} - \mathsf{margen}$$
$$\wedge N_1 \leq C_1 \wedge N_2 \leq C_2 \wedge \ldots \wedge N_n \leq C_n$$

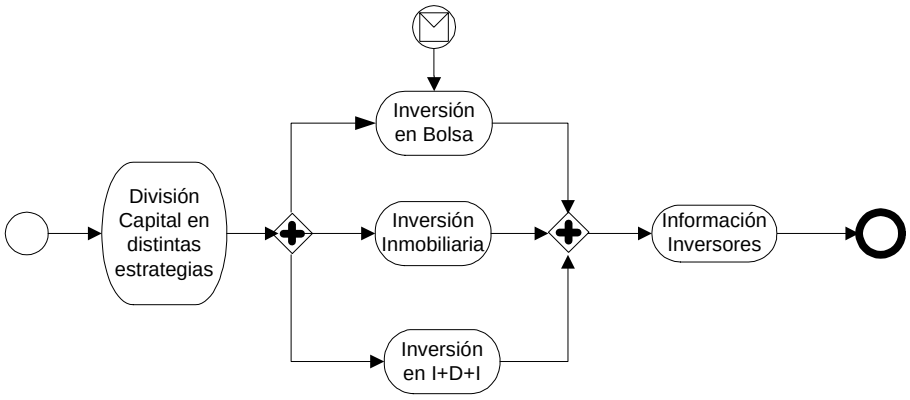


Figura 1. Ejemplo de Proceso de negocio

La restricción que describe el funcionamiento del servicio *Inversión en Bolsa* define la postcondición de dicho servicio, mientras que la precondición también deberá ser representada mediante una restricción.

Como el intercambio de información entre los servicios web se realiza mediante un protocolo de mensajería, la información que se intercambia en dichos mensajes marcará qué variables son conocidas entre los diferentes procesos y cuáles son privadas. En el ejemplo de la inversión en bolsa, existen dos tipos de mensajes, uno de recepción de datos y otro de envío de resultados. En este caso son las variables inversión, margen y gananciaTotal, y los mensajes serán de la forma:

```
<message name="DatosInversionBolsa">
  <part name="inversion"type="xsd:integer"/ >
  <part name="margen"type="xsd:integer"/ >
< /message>

<message name="SalidaInversionBolsa">
  <part name="gananciaTotal"type="xsd:integer"/ >
< /message>
```

3. Almacenamiento de especificaciones de servicios web en bases de datos

Cuando un sistema combina diferentes servicios en un proceso para obtener un objetivo, es común que cada una de las fases de dicha producción trabaje con un conjunto de datos almacenados en diferentes estructuras de almacenamiento y de diferente naturaleza. Como se ha analizado en la sección anterior, el comportamiento de los servicios puede ser mapeado a restricciones, por lo que la forma natural de almacenarlas será en una Base de Datos de Restricciones (CDB - Constraint Database).

Las Bases de Datos de Restricciones tuvieron su origen a principios de 1990 con el artículo de Kanellakis, Kuper y Revesz [10], ampliándose más tarde en los trabajos [11][12]. Originalmente, el objetivo de las CDBs fue definir una versión de bases de datos a la que se le añadió funcionalidad relacionada con la Programación Lógica con Restricciones (Constraint Logic Programming - CLP).

Las CDBs fueron definidas en base a la lógica y modelo de primer orden, tanto para la descripción de restricciones como para la evaluación de las consultas. El fundamento de las CDBs se apoya en la dualidad de representar un conjunto de datos mediante una restricción (fórmula) sobre un conjunto de variables libres x_1, ..., x_m, junto a la utilización de atributos clásicos del álgebra relacional a_1, ..., a_n que sólo pueden tomar un valor en cada tupla. Esta ampliación de las bases de datos de restricciones, desarrollada más ampliamente en [9], permite almacenar y consultar las restricciones como un dato cualquiera como los enteros, cadenas...

- Una *k-tupla restricción* con las variables $x_1, \ldots, x_k$ es una conjunción finita de la forma $\varphi_1 \wedge \ldots \wedge \varphi_N$ donde cada φ_i, para $1 \leq i \leq N$, es un *Atributo Clásico o Univaluado* o un *Atributo Restricción*. Un atributo univaluado es de la forma $\{x_j$=*Constante*$\}$ donde $x_j \in \{x_1, \ldots, x_k\}$. Un atributo restricción es una relación representada mediante una restricción entre las variables $x_1, \ldots, x_k$ no pertenecientes a un atributo univaluado.
- Una *relación restrictiva* está definida como un conjunto de *Atributos Univaluados* y un conjunto de *Atributos Restricción*. Una *relación restrictiva de aridad k*, es un conjunto finito $r = \{\psi_1, \ldots, \psi_M\}$, donde cada ψ_j para $1 \leq j \leq M$ es una k-tupla restricción sobre las variables $x_1, \ldots, x_k$ tal como se muestra en la figura 2. La fórmula correspondiente es la disjunción de

la forma $\psi_1 \vee \ldots \vee \psi_M$, tal que $\psi_j = \varphi_1 \wedge \ldots \wedge \varphi_N$ donde cada φ_i para $1 \leq i \leq N$ es una *k-tupla restricción*. Si existe en cada $\psi_j \in r$ una φ_i de la forma $\{x{=}Constante\}$, donde x sea la misma variable en todos los φ_i pertenecientes a distintos ψ_j y x no aparece en el resto de los φ_i de una misma ψ_j, se dirá que x es un **atributo univaluado**, mientas que el resto de variables formarán parte en uno o varios **Atributos Restricción**.

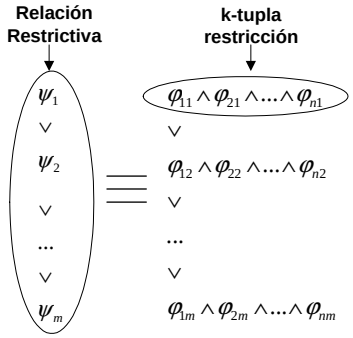


Figura 2. Representación de k-tuplas restricción y relaciones restrictivas

- Por lo tanto, una *Base de Datos de Restricciones* es una colección finita de relaciones restrictivas formadas por **atributos univaluados** y **atributos restricción**.

La idea es que se tendrían todos los servicios definidos en una CDB mediante sus correspondientes contratos, precondición y postcondición, y cuando un servicio es detectado como erróneo, se buscará en la o las bases de datos de restricciones pertinentes qué nuevo servicio podría sustituirlo porque tenga la misma especificación.

Esta dualidad permite almacenar los datos que describen el comportamiento de los servicios web como restricciones, y sus variables públicas y puertos como atributos clásicos. Un ejemplo de representación de los servicios descritos en la figura 1 es el mostrado en la figura 3. En este ejemplo aparece dos atributo clásico que serán el identificador del servicio web (IDSW) y la descripción textual (Descripción). Por otra parte también existen dos atributos restricción que definen el contrato de cada servicio, la precondición y la postcondición.

Para conectar los distintos servicios, se utilizarán las variables definidas como públicas, al representar la información mediante restricciones es posible realizar operaciones de proyección sobre ellas para modificar sus variables. Esto haría posible que un mismo servicio mostrara diferentes precondiciones y postcondiciones en función de las variables que hiciera públicas en cada momento, o dependiendo del servicio con el que se relacionara.

IDSW	Precondición	Postcondición	Descripción
2354	margen≥0∧inversión≥0	*N1*PG1 + N2*PG2 +...Nn*PGn* = gananciaTotal ∧...	Inversión en bonos
3487	1.000.000≥inversión∧inversión≥120.000	Inversión*1,8-2.500=ganaciaTotal	Inversión Inmobiliaria
5561	18.000≥inversión∧inversión≥0 ∧700≥gastoFijo≥500	Inversion*1,4-gastoFijo=gananciaTotal	Inversión en I+D+I
...	...	...	...

Figura 3. Ejemplo de tuplas con Servicios Web

Para que las consultas a la CDB sean más eficientes, cuando se crea una Base de Datos de Restricciones se crean también tres tablas (*Restricciones*, *Variables* y *Restricciones/Variables*) mostradas en la figura 4, que mantienen las relaciones entre las variables y las restricciones. Estas tablas disminuyen la complejidad computacional de las consultas ya que con una sola consulta se puede conocer qué restricciones tienen qué variables y viceversa. Estas tablas permiten identificar cada restricción (tabla *Restricciones*), cada variable (tabla *Variables*) y establecer la relación entre ambas (tabla *Restricciones/Variables*), lo que ayuda a detectar las restricciones relacionadas con una consulta sin necesidad de tratarlas todas. Estas tablas no son visibles ni accesibles por el usuario de una forma directa, pero sí de una manera implícita cuando se añaden, modifican, borran o consultan restricciones de la base de datos. La tabla *Restricciones* almacena el *idRestricción* que corresponde al identificador del objeto (OID) generado por el sistema, y la *Etiqueta* acorde con el tipo de restricción que representa (Polinómica, lineal, de igualdad o inecuación). La tabla *Variables* almacena, para cada variable, su identificador, su nombre y el dominio (Natural, Entero o Flotante) al que pertenece. La tabla *Restricciones/Variables* almacena la relación entre las variables y las restricciones, y sobre qué rango (*Rango_Inicio*, *Rango_Fin*) están definidas cada una de las variables para cada una de las restricciones en las que participan.

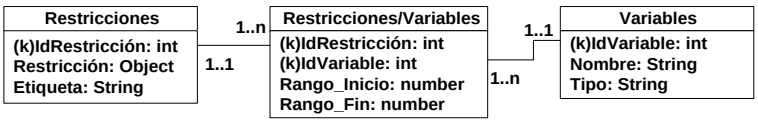


Figura 4. Indexación entre variables y restricciones

4. Búsqueda eficiente de Servicios Web sustitutos

En la bibliografía hay trabajos que se han centrado en las consultas relacionadas con servicios web [13], [14]. Más concretamente, La propuesta de este trabajo se centra en buscar de forma eficiente servicios que puedan sustituir a

uno dado, para ello debemos tener en cuenta como ya dijimos en la sección 2 que una implementación M de un servicio de un proceso satisface un contrato C si satisface la postcondición, sujeto a una precondición dada.

Sean $\mathtt{S}_W\mathtt{(Pre)}$ y $\mathtt{S}_W\mathtt{(Post)}$ la representación de la precondición y la postcondición del contrato correspondiente a un servicio $\mathtt{S}_W$ que implementa una actividad de un proceso de negocio. Para poder obtener otro servicio S_b que pueda sustituir el que trabaja de forma incorrecta S_a, será necesario encontrar otro servicio que cumpla las siguientes asertos:

- $S_b\mathtt{(Pre)} \supseteq S_a\mathtt{(Pre)}$. Lo que significa que todos los posibles valores de entrada del servicio S_a están recogidos en el nuevo servicio S_b.
- $S_b\mathtt{(Post)} \subseteq S_a\mathtt{(Post)}$. Lo que significa que los valores de salida que devuelva el servicio S_b son equivalentes o están incluidos en los valores de salida que devolvía el servicio S_a.

Nuestra propuesta define la implementación de esta comprobación mediante la construcción de Problemas de Satisfacción de Restricciones, más concretamente con el operador de Selección de las CBDs.

Un problema de satisfacción de restricciones (Constraint Satisfaction Problem - CSP) es una representación de un conjunto de variables, dominios y restricciones, ligado a uno o varios esquema de razonamiento para resolverlo. Formalmente, está definido por la tripleta <X, D, C> donde, X = $\{x_1, x_2, \ldots, x_n\}$ es un conjunto finito de variables, D = $\{d(x_1), d(x_2), \ldots, d(x_n)\}$ es el conjunto de dominios de cada una de las variables, y C = $\{C_1, C_2, \ldots, C_m\}$ un conjunto de restricciones. Cada restricción C_i está definida como una relación R de un conjunto de variables V = $\{x_i, x_j, \ldots, x_k\}$ a lo que se denomina *ámbito de la restricción*.

Para hacer más eficiente la búsqueda de servicios, la implementación de CDBs que se propone utilizar es la descrita en [15]. Esta propuesta busca la eficiencia, por lo que intenta evitar la construcción y resolución de CSPs en la medida de lo posible. Para ello hacemos un análisis de los rangos de las variables, que se almacenarán tal como se ha descrito en la sección anterior. Existen casos donde el análisis de dichos rangos evita la construcción de CSPs. Analizando el mínimo y máximo valor que puede tomar una variable, es posible inferir si las restricciones involucradas en la comparación cumplen o no una condición. Por ejemplo si se comparan dos restricciones C_x y C_y, ambas definidas sobre las variables v_1 e v_2, donde los rangos son $C_x(v_1{:}[5..15], v_2{:}[20..30])$ y $C_y(v_1{:}[20..25], v_2{:}[40..55])$. En este caso, el predicado fuera $C_x \subseteq C_y$ seguro que no se cumple, y el servicio no serviría como sustituto. Volviendo al ejemplo anterior, si los dominios de las variables fueran $C_x(v_1{:}[5..15], v_2{:}[20..30])$ y $C_y(v_1{:}[2..25], v_2{:}[10..55])$, y el predicado $C_x \subseteq C_y$, pese a que los dominios de las variables en C_x están incluidas en los dominio de las variables en C_y, no se puede asegurar que se cumple el predicado para las restricciones. Un ejemplo donde los rangos de las variables están incluidos dentro de otra, pero no sus soluciones, es el mostrado en la figura 5.

Para cada una de las variables de las restricciones de los servicios que se quieren analizar, existen diferentes posibilidades entre los rangos de las variables,

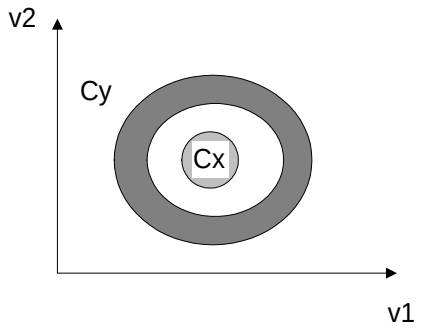


Figura 5. Ejemplo de $C_x \nsubseteq C_y$

mostradas en la figura 6. En función de cada una de las relaciones entre los rangos y la resolución de CSP, se puede concluir que:

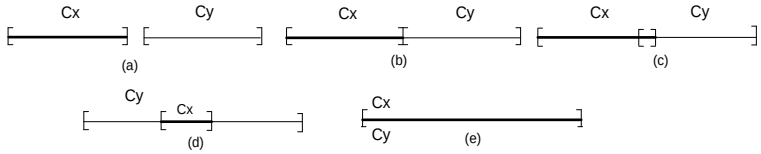


Figura 6. Tipos de relaciones entre los rangos de las variables de dos restricciones

- (a): $C_x \subseteq C_y$ es falso
- (b): $C_x \subseteq C_y$ es falso
- (c): $C_x \subseteq C_y$ es falso
- (d): $C_x \subseteq C_y$ es necesario crear un CSP para saber si es cierto o falso
- (a): $C_x \subseteq C_y$ es necesario crear un CSP para saber si es cierto o falso

La creación del CSP se basa en que $C_x \subseteq C_y$ será cierto si todas las soluciones de C_x lo son también de C_y. Para la inclusión, es necesario que las restricciones estén definidas sobre las mismas variables, de forma que sean C_x y C_y dos restricciones donde $X = \{x_1, x_2, \ldots, x_n\}$ son las variables de C_x y C_y, $C_x \subseteq C_y$ es igual que la implicación $(C_x \rightarrow C_y)$ [16]. Esta operación determina si todas las soluciones de C_x son también soluciones de C_y, aunque es posible que C_y tenga soluciones que no pertenezcan a C_x.

Para evitar analizar todas las soluciones de C_x, comprobando que éstas también lo son de C_y, se buscará una solución donde esto no ocurra, de forma que la evaluación de la condición $C_x \subseteq C_y$ corresponde con la fórmula:

$$\neg(\exists_{x_i \in X}(C_x \ \wedge \ \neg C_y))$$

Y el CSP que se creará es:

$$\boxed{\begin{array}{l} X = \{x_1, \ x_2, \ \ldots, \ x_n\} \\ C_x \ \wedge \ \neg C_y \end{array}}$$

Si se encuentra alguna solución para el CSP devolverá *falso*, y *cierto* si no encuentra ninguna.

Cuando no se encuentra un único servicio, será necesario buscar una combinación de varios que puedan definir un nuevo servicio. Para la composición de servicios existen trabajos previos relacionados [17], [18], [19].

5. Conclusions and Future Work

Este artículo presenta los pasos necesarios para la búsqueda eficiente de servicios que implementen las actividades de un proceso de negocio que permita su sustitución cuando éstas fallen. El uso de CDBs permite crear indexación entre las restricciones y las variables, para que la evaluación de la selección sea lo más eficiente posible.

Como trabajo futuro se quieren analizar el resto de operaciones de servicios que se describen en el lenguaje BPEL, tanto las básicas como las estructurales, dando un análisis completo de todas las posibilidades.

También se plantea como trabajo futuro la búsqueda de restricciones para combinar distintos servicios web y que se pueda obtener servicios más eficientes. Esto implicará cómo saber si pueden sustituir un servicio determinado, cuál es el mejor y cuál es la opción más eficiente en caso de que existieran varios servicios o combinación de ellos para sustituir el erróneo. Esto puede ser analizado con el operador de proyección de las Bases de Datos de Restricciones.

6. Agradecimientos

Este trabajo ha sido parcialmente subvencionado por la Junta de Andalucía, mediante la Conserjería de Innovación, Ciencia y Empresas (P08-TIC-04095) y por el Ministerio de Ciencia y Tecnología de España (DPI2006-15476-C02-01) y las ayudas Europeas al desarrollo (Fondo FEDER).

Referencias

1. Stefano Bocconi, Claudia Picardi, Xavier Pucel, Daniele Theseider Dupré, and Louise Travé-Massuyès. Model-based diagnosability analysis for web services. In *AI*IA '07: Proceedings of the 10th Congress of the Italian Association for Artificial Intelligence on AI*IA 2007*, pages 24–35, Berlin, Heidelberg, 2007. Springer-Verlag.

2. R. Reiter. A theory of diagnosis from first principles. volume 1, pages 57–96, 1987.
3. R. Greiner, B. A. Smith, and R. W. Wilkerson. A correction to the algorithm in reiter's theory of diagnosis. volume 41, pages 79–88. Elsevier Science Publishers Ltd, 1989.
4. Shi-Ming Huang, Yuang-Te Chu, Shing-Han Li, and David C. Yen. Enhancing conflict detecting mechanism for web services composition: A business process flow model transformation approach. *Inf. Softw. Technol.*, 50(11):1069–1087, 2008.
5. Catriel Beeri, Anat Eyal, Tova Milo, and Alon Pilberg. Monitoring business processes with queries. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 603–614. VLDB Endowment, 2007.
6. Business Process Execution Language for Web Services. http://www.ibm.com/developerworks/library/ws-bpel/, 2003.
7. Glen Dobson. Using ws-bpel to implement software fault tolerance for web services. In *EUROMICRO '06: Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 126–133, Washington, DC, USA, 2006. IEEE Computer Society.
8. María Teresa Gómez-López, Rafael M. Gasca, Carmelo Del Valle, and F. Fernando de la Rosa T. Querying a polynomial constraint object-relational database in model-based diagnosis. In *International Conference on Database and Expert Systems Applications-DEXA*, volume 3588, pages 848–857. Lecture Notes in Computer Science, Springer, 2005.
9. María Teresa Gómez López, Rafael Ceballos, Rafael M. Gasca, and Carmelo Del Valle. Developing a labelled object-relational constraint database architecture for the projection operator. *Data Knowl. Eng.*, 68(1):146–172, 2009.
10. G. M. Kuper P. C. Kanellakis and P. Z. Revesz. Constraint query languages. *Symposium on Principles of Database Systems*, pages 299–313, 1990.
11. G. Kuper, L. Libkin, and J. Paredaes. *"Introduction", Constraint Databases, G. Kuper and L. Libkin and J. Paredaes.* Springer, 2000.
12. P. Revesz. Datalog and Constraints. *Constraint Databases*, pages 155–170, 2000.
13. D. Martin S. Narayanan R. Waldinger G. Denker, J. Hobbs. Querying and accessing information on the semantic web. In *Proc. Semantic Web Workshop, in conjunction with 10th international Worldwide Web Conference*, 2001.
14. Catriel Beeri, Anat Eyal, Simon Kamenkovich, and Tova Milo. Querying business processes. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 343–354. VLDB Endowment, 2006.
15. María Teresa Gómez López. Lorcdb: Gestor de bases de datos objeto-relacionales de restricciones. Ph D thesis, 2007.
16. Kim Marriott and Peter J. Stuckey. *"Programming with Constraints. An introduction", Simplification, Optimization and Implication.* The Mitt Press, 1998.
17. Singh M.P. Cheng, Z. and M.A. Vouk. Composition constraints for semantic web services. In *Proceedings of WWW2002 Workshop on Real World RDF and Semantic Web Application*, 2003.
18. Srini Narayanan and Sheila A. McIlraith. Simulation, verification and automated composition of web services. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 77–88, New York, NY, USA, 2002. ACM.
19. Rajesh Thiagarajan and Markus Stumptner. Service composition with consistency-based matchmaking: A csp-based approach. In *ECOWS '07: Proceedings of the Fifth European Conference on Web Services*, pages 23–32, Washington, DC, USA, 2007. IEEE Computer Society.

SMOTE-I: mejora del algoritmo SMOTE para balanceo de clases minoritarias

J Moreno[1], D Rodríguez [1], MA Sicilia, JC Riquelme[2] y R Ruiz[3]
[1] Departamento de ciencias de la Computación
Campus Externo. Universidad de Alcalá, 287
Ctra. Barcelona km. 33.6, - 28871 Alcalá de Henares (Madrid)
[2]Escuela Técnica Superior de Ingeniería Informática
Avda. Reina Mercedes, s/n, 41012 – Sevilla, España
[3] Escuela Politécnica Superior
Universidad Pablo de Olavide
Ctra. Utrera Km 1, 41013 Sevilla
{daniel.rodriguezg,msicilia}@uah.es,robertoruiz@upo.es, riquelme@us.es

Resumen. Las técnicas de minería de datos están encaminadas a desarrollar algoritmos que sean capaces de tratar y analizar datos de forma automática con objeto de extraer de cualquier tipo de información subyacente en dichos datos. El problema del desbalanceo de los datos consiste en la predominancia de ciertos valores en los datos y la escasez o ausencia de otros que dificulta o impide la extracción de información. En este trabajo se presenta un nuevo algoritmo basado en SMOTE y llamado SMOTE-I que mejora al original con la la base de datos analizada.

Palabras clave: SMOTE, ENN, desbalanceo de datos.

1 Introducción

En un caso ideal todos los datos pertenecientes a cada clase se encuentran agrupados entre ellos y claramente diferenciables del resto de clases. La realidad es bien distinta y con frecuencia los datos presentan diferentes problemas que dificultan la labor de los clasificadores y disminuyen la calidad de la clasificación realizada. Los problemas más destacables son (i) el ruido es un problema derivado de la naturaleza de los datos consistente en el gran parecido que presentan entre sí datos pertenecientes a clases distintas o datos erróneos; (ii) el solapamiento entre clases ocurre cuando datos de clases distintas ocupan un espacio común debido a que algunos de los atributos de dichas clases comparten un mismo rango de valore; y (iii) el desbalanceo de clases ocurre cuando el número de instancias de cada clase es muy diferente

En estas circunstancias los clasificadores presentan una tendencia de clasificación hacia la clase mayoritaria, minimizando de ésta manera el error de clasificación y clasificando correctamente instancias de clase mayoritaria en detrimento de instancias de clase minoritaria. Entre las medidas que podemos tomar para el tratamiento del ruido se encuentran (i) usar algoritmos tolerantes al ruido; (ii) usar algoritmos que

reduzcan el ruido filtrando las instancias ruidosas y (iii) usar algoritmos que corrijan las instancias que generan el ruido. Para el tratamiento del solapamiento podemos utilizar algoritmos de "limpieza" que reduzcan las áreas de solapamiento. Finalmente, hay más opciones para el tratamiento del tratamiento del desbalanceo, entre las que se encuentran:

- *Sampling*: Consiste en balancear la distribución de las clases añadiendo ejemplos de las clase minoritaria. A ésta técnica se le denomina *oversampling*. Algunos de los algoritmos más representativos son SMOTE [1], y *Resampling* [5]. También es posible realizar lo contrario: eliminar ejemplos de la clase mayoritaria. Ésta técnica se conoce como *undersampling*. Un algoritmo bastante representativo es RUS [3]. Ambas técnicas tienen ventajas e inconvenientes. Entre los inconvenientes del *undersampling* está la pérdida de información que se produce al eliminar instancias de la muestra. Sin embargo tiene la ventana de que reduce el tiempo de procesado del conjunto de datos. *Oversampling* tiene la ventaja de no perder información pero puede repetir muestras con ruido además de aumentar el tiempo necesario para procesar el conjunto de datos.
- *Oversampling*:
    - *SMOTE*: Genera nuevas instancias de la clase minoritaria interpolando los valores de las instancias minoritarias más cercanas a una dada.
    - *Resampling*: Duplica al azar instancias de la clase minoritaria
- *Undersampling*:
    - *Random undersampling*: Elimina al azar instancias de la clase mayoritaria
    - *Tomek Links* [4]: Elimina sólo instancias de la clase mayoritaria que sean redundantes o que se encuentren muy cerca de instancias de la clase minoritaria.
    - *Wilson Editing* [6]. También conocido como ENN (*Editing Nearest Neighbor*) elimina aquellas instancias donde la mayoría de sus vecinos pertenecen a otra clase.
- *Boosting* consiste en asociar pesos a cada instancia que se van modificando en cada iteración del clasificador. Inicialmente todas las instancias tienen el mismo peso y después de cada iteración, en función del error cometido en la clasificación se reajustan los pesos con objeto de reducir dicho error:
    - AdaBoost: Implementa el algoritmo de *Boosting* descrito. En cada iteración AdaBoost genera nuevas instancias utilizando *Resampling*
    - SMOTEBoost: Es similar a AdaBoost pero usa SMOTE en lugar del *Resampling* para generar nuevas instancias.
    - RUSBoost: Aplica AdaBoost pero en cada iteración utiliza RUS (*Random Undersampling*) que reducen el tamaño de la muestra de datos y simplifican y aumentan el rendimiento del clasificador.

2 Trabajos relacionados: SMOTE y ENN

Fix y Hodges [2] publicaron el algoritmo de la regla del vecino más cercano. La idea básica del algoritmo es suponer que instancias próximas entre sí tienen mayor probabilidad de pertenecen a la misma clase. Para clasificar una nueva instancia, se realiza un cálculo de la distancia entre cada atributo de la nueva instancia y el resto de instancias del conjunto de datos y se asocia a la clase de la instancia más cercana. El principal inconveniente del algoritmo es el alto coste computacional que tiene.

SMOTE (*Syntetic Minority Over-sampling Technique*) [1] es un algoritmo de *oversampling* que genera instancias "sintéticas" o artificiales para equilibrar la muestra de datos basado en la regla del vecino más cercano. La generación se realiza extrapolando nuevas instancias en lugar de duplicarlas como hace el algoritmo de *Resampling*. Para cada una de las instancias minoritarias se buscan las instancias minoritarias vecinas (más cercanas) y se crean N instancias entre la línea que une la instancia original y cada una de las vecinas. El valor de N depende del tamaño de *oversampling* deseado. Para un caso del 200% por cada instancia de la clase minoritaria deben crearse dos nuevas instancias genéricas.

SMOTE es un algoritmo de sobre-muestreo de ejemplos utilizado para la clase minoritaria:

- Crea ejemplos sintéticos en lugar de hacer un sobre-muestreo con reemplazo.
- Opera en el espacio de atributos *feature space*, en lugar del espacio de datos *data space*.
- Crea un ejemplo sintético a lo largo de los segmentos de línea que unen alguno o todos los k vecinos más cercanos de la clase minoritaria.
- Se eligen algunos de los k vecinos más cercanos de manera aleatoria (no se utilizan todos).
- SMOTE utiliza típicamente $k = 5$.

El algoritmo de SMOTE realiza los siguientes pasos:

- Recibe como parámetro el porcentaje de ejemplos a sobre-muestrear.
- Calcula el número de ejemplos que tiene que generar.
- Calcula los k vecinos más cercanos de los ejemplos de la clase minoritaria.
- Genera los ejemplos siguiendo este proceso:
    - Para cada ejemplo de la clase minoritaria, elige aleatoriamente el vecino a utilizar para crear el nuevo ejemplo.
    - Para cada atributo del ejemplo a sobre-muestrear, calcula la diferencia entre el vector de atributos muestra y el vecino elegido.
    - Multiplica esta diferencia por un número aleatorio entre 0 y 1.
    - Suma este último valor al valor original de la muestra.
    - Devuelve el conjunto de ejemplos sintéticos.

SMOTE es el algoritmo herramienta más utilizada para realizar el sobre muestreo pero presenta los siguientes inconvenientes: (i) puede generar muchos ejemplos artificiales cuyas semillas son ejemplos con ruido; (ii) al generar un nuevo ejemplo, interpola entre dos ejemplos de la clase minoritaria, sin embargo, pueden existir muchos ejemplos cercanos o inclusive entre ellos de la clase mayoritaria, generando modelos incorrectos; (iii) sólo funciona con variables continuas y (iv) no tiene una forma clara de decidir cuántos ejemplos generar. Para minimizar los inconvenientes

citados después de usar SMOTE pueden aplicarse otros algoritmos de limpieza para reducir el ruido y el solapamiento de las clases:

- *Tomek Links*: Elimina las instancias de las clases mayoritaria y minoritaria que estén muy próximas entre sí.
- *Wilson Editing* (ENN): Elimina cualquier instancia cuya etiqueta sea distinta de las clases que la rodean. ENN suele eliminar más instancias que Tomek Link.
- Además en éste trabajo se presenta una versión modificada SMOTE-I que ofrece una solución "automática" para balancear conjuntos de datos con más de una clase minoritaria aportando a la vez una sencilla solución al problema de determinar el número de ejemplos a generar sintéticamente.

El algoritmo de Edición de Wilson, también conocido como la regla del vecino más cercano editado (*Edited Nearest Neighbor*) elimina las instancias mal clasificadas de un conjunto de datos mediante la regla *k-NN*. Para identificar estas instancias el algoritmo elimina aquellas instancias cuya clase no coincide con la clase de la mayoría de sus vecinos. Este algoritmo elimina la superposición entre clases y por tanto parte de posible ruido que pueda haber en la muestra. El algoritmo de ENN realiza los siguientes pasos:

1. Recibe como parámetro el número de vecinos a buscar para cada clase
2. Calcula los k vecinos más cercanos de cada instancia.
3. Si la mayoría de los vecinos son de clase distinta y marca la instancia.
4. Elimina las instancias marcadas.

3 SMOTE-I

La definición original de SMOTE únicamente contempla datos en los que hay una única clase minoritaria. La modificación propuesta en éste trabajo permite *C* clases minoritarias ajustando automáticamente los pesos de cada clase para generar las instancias apropiadas de cada una de ellas. Adicionalmente también se ha añadido la opción de invertir los pesos de las clases minoritarias. Dado un conjunto de datos con un total de *T* instancias pertenecientes a *C* clases distintas, y una probabilidad de máxima para considerar cualquier clase como minoritaria *P*: Una clase C_n con In instancias, siendo $0 \leq n < C$ será minoritaria si su ocurrencia, $P_n = I_n/T$ es menor que *P*. El número de instancias sintéticas a generar para cada clase Cn minoritaria será igual al factor de sobremuestreo *N*. El número total de instancias sintéticas generadas *TS* será:

$$TS = \sum I_n * N$$

Como puede observarse, el número de instancias generadas en caso de haber más de una clase minoritaria será proporcional al número de instancias de cada clase. De ésta forma, se generarán más instancias sintéticas para las clases minoritarias con mayor número de instancias cuando lo deseable es justamente lo contrario: Generar mayor número de instancias sintéticas para las clases con menor número de instancias. Para realizar ésta inversión y teniendo en cuenta las definiciones anteriores: Definiremos la probabilidad invertida de cada instancia PI_n como:

$$PI_n = \frac{1 - P_n}{C - 1}$$

y determinaremos el número de instancias sintéticas a generar para cada clase C_n minoritaria, *SN* de la siguiente manera:

$$SN = \frac{PI_n \cdot T}{I_n}$$

La siguiente tabla muestra un ejemplo de un conjunto de 238 instancias distribuidas en 5 clases siendo $P = 50\%$:

I_n	P_n	$1 - P_n$	PI_n	$PI_n * T$	$(PI_n * T) - I_n$	*SN*	*Nro instancias sintéticas por clase*
12	0.05	0.95	0.24	56.5	44.5	4	48
21	0.09	0.91	0.23	54.25	33.25	2	42
23	0.10	0.90	0.23	53.75	30.75	1	23
59	0.25	0.75	0.19	44.75	-14.25	0	0
Clase mayoritaria:							
123	0.52						
Totales:							
238	1	4	1	238			113

Tabla 1. Ejemplo cálculos SMOTE-I

Si observamos la anterior tabla, la columna PIn*T y su total, PIn*T indica el número total de instancias que deberían generarse de cada clase y puesto que cada clase ya cuenta con In instancias, lo correcto es restar ese valor tal como muestra la columna $(PI_n * T) - I_n$. De ésta manera, redefinimos SN como:

$$SN = \frac{\left(PI_n * T\right) - I_n}{I_n}$$

La última columna de la tabla nos muestra el resultado de la inversión de probabilidades que favorece la creación de instancias sintéticas de las clases con menor número de instancias reales. En el apartado 5. Resultados Experimentales puede observarse la aplicación de éste algoritmo.

Resultados Experimentales

En éste proyecto se han utilizado un conjunto de datos obtenidos del repositorio de PROMISE[1], llamado *usp5* y contienen clases nominales y ha sido utilizados para experimentar con el algoritmo SMOTE-I y comparar los resultados con el SMOTE original y ENN en problemas de clasificación. Más concretamente se ha utilizado

[1] http://promisedata.org/

C4.5 (implementado bajo el nombre de J48) en la herramienta Weka. J48 es un árbol de decisión cada nivel corresponde a una clase, cada nodo a un atributo y las ramas a los valores asociados a dichos atributos.

El conjunto de datos `usp5` de PROMISE contiene 15 atributos que representan información acerca métricas del esfuerzo en proyectos de ingeniería del software. Los atributos son los siguientes:

- *ID*: Identificador de tres dígitos
- *ObjType*: Tipo de objeto (PJ-*project*, FT-*feature*, RQ-*requirement*)
- *Effort*: Esfuerzo actual en horas gastadas en tareas relacionadas con la implementación del objeto por todas las personas participantes.
- *Funct%*: Porcentaje de funcionalidad desarrollada.
- *IntComplx*: Complejidad interna de cálculo.
- *DataFile*: Número de ficheros de datos/tablas accedidos.
- *DataEn*: Número de puntos de entrada de datos.
- *DataOut*: Número de puntos de salida de datos.
- *UFP*: Puntos de Función sin ajustar.
- *Lang*: Lenguaje de desarrollo utilizado.
- *Tools*: Herramientas de desarrollo y plataformas.
- *ToolExpr*: Experiencia de los desarrolladores con el lenguaje y las herramientas de desarrollo.
- *AppExpr*: Nivel de experiencia desarrollando aplicaciones.
- *TeamSize*: Tamaño del equipo de desarrollo.
- *DBMS*: Sistema gestor de bases de datos utilizado
- *Method*: Metodología de desarrollo utilizada.
- *AppType*: Tipo de sistema/arquitectura. Es la clase del *dataset*. Puede contener los siguientes valores: *B/S, C/S, BC/S, Centered, Other*

A continuación en las tablas 2 y 3 se presentan los resultados utilizando validación cruzada de 10 (*10 CV*).

					ENN	
Clase	***J48***	***Resample***	***ENN***	***SM-I***	***Resamp***	***SM-I***
BC/S	0,878	0,973	0,831	0,897	0,973	0,899
B/S	0,853	0,952	0,866	0,831	0,952	0,852
NULL	0,737	0,833	0,778	0,833	0,833	0,857
C/S	0,678	0,818	0,667	0,721	0,818	0,806
C	0,000	0,000	0,000	0,950	0,000	0,927
S	0,000	0,800	0,000	0,977	0,800	0,977
B	0,000	0,000	0,000	0,909	0,000	0,976
3-Tier,_B/S	0,000	0,000	0,000	0,000	0,000	0,000
BS	0,667	0,800	0,667	1,000	0,800	0,952
BC	0,000	0,000	0,000	0,000	0,000	0,000
B/C	0,000	0,000	0,000	0,000	0,000	0,000

Tabla 2. Usp05, F-Measure clasificando con J48

					ENN	
Clase	*J48*	*Resamp*	*ENN*	*SM-I*	*Resamp*	*SM-I*
BC/S	0,936	0,981	0,915	0,955	0,981	0,973
B/S	0,984	0,999	0,979	0,996	0,999	0,997
NULL	0,985	0,926	0,984	0,961	0,926	0,964
C/S	0,872	0,930	0,870	0,885	0,930	0,960
C	0,690	?	0,680	0,972	?	0,974
S	0,808	0,997	0,804	0,998	0,997	0,998
B	0,433	0,995	0,408	0,998	0,995	1,000
3-Tier,_B/S	0,463	?	0,456	0,475	?	0,474
BS	1,000	0,999	0,999	1,000	0,999	0,999
BC	0,463	?	0,459	0,480	?	0,478
B/C	0,443	?	0,424	0,472	?	0,470

Tabla 3. Usp05, A-ROC clasificando con J48

En el conjunto de datos utilizado con más de dos clases, en los resultados obtenidos se observa cómo a pesar de haber balanceado el conjunto de instancias mediante SMOTE el porcentaje de instancias clasificadas correctamente no aumenta significativamente, e incluso es inferior a los resultados obtenidos aplicando directamente el clasificador a los datos. Además, a pesar de obtenerse menor porcentaje de instancias clasificadas correctamente, los valores de *AUC* y *F-Measure* obtenidos después de aplicar SMOTE mejoran algo en relación a los resultados obtenidos sin aplicarlo. También se observa una mejora significativa en éstos valores al aplicar la versión SMOTE-I en conjunción con el clasificador J48.

5 Conclusiones y trabajo futuro

En este trabajo se ha presentado una modificación del conocido algoritmo de SMOTE. Este es uno de los problemas que presenta SMOTE ya que al interpolar entre dos instancias minoritarias es posible que entre ellas existan instancias de la clase mayoritaria dificultando así la labor del clasificador. La variante de SMOTE presentada en este trabajo tiene en cuenta la distribución de las clases y genera las instancias sintéticas acorde a esta distribución.

Los trabajos futuros deben ir encaminados al desarrollo de nuevos algoritmos de balanceo de datos y a mejorar los existentes como por ejemplo el estudio del número óptimo de instancias sintéticas a generar por SMOTE o SMOTE-I.

Agradecimientos

Esta investigación está financiada por el Ministerio de Educación y Ciencia (TIN2007-68084-C02-00) y las universidades de Alcalá, Sevilla y Pablo de Olavide.

Referencias

1. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer W.P. (2002). SMOTE: Syntetic Minority Over-sampling Technique. Journal of Artificial Inteligence Research 16, 321-357. Disponible en: http://www.jair.org/media/953/live-953-2037-jair.pdf
2. Fix, E. Hodges, J. (1952). Discriminatory analysis, nonparametric discrimination: small simple performance. Technical Report 11, US Air Force, School of Aviation Medicine, Randolph Field, TX.
3. Kamei, Y., Otros (2007). The Effects of Over and Under Sampling on Fault-prone Module Detection. Empirical Software Engineering and Measurement, 2007. First International Symposium on Volume, Issue, 20-21 Sept. 2007. Page(s)196-204. IEEE Computer Society Press, Los Alamos
4. Tomek , I. (1976), Two modifications of CNN, IEEE Transactions on Systems, Man and Cybernetics 6. IEEE Computer Society Press, Los Alamos.
5. Wilson , D. L. (1972), Asymptotic properties of nearest neighbor rules using edited data, IEEE Transactions on Systems, Man and Cybernetics . IEEE Computer Society Press, Los Alamos.
6. Witten, I. Frank, E. (2005), Data Mining: Practical machine learning tools and techniques, 2nd ed., Morgan Kaufmann, San Francisco

Aplicación de las Técnicas de Modelado y Simulación en la Gestión de Servicios TI

Elena Orta[1], Mercedes Ruiz[1] y Miguel Toro[2]
[1] Departamento de Lenguajes y Sistemas Informáticos
Escuela Superior de Ingeniería
C/ Chile, 1
11003 – Cádiz, España
[2]Departamento de Lenguajes y Sistemas Informáticos
Escuela Técnica Superior de Ingeniería Informática
Avda. Reina Mercedes, s/n
41012 – Sevilla, España
{elena.orta, mercedes.ruiz}@uca.es, migueltoro@us.es

Resumen. Las técnicas de modelado y simulación ofrecen la posibilidad de experimentar diferentes decisiones y analizar sus resultados en sistemas donde el coste o el riesgo de una experimentación real son prohibitivos. En este trabajo se ofrece una visión global de las posibles aplicaciones de estas técnicas en el ámbito de los procesos de ITIL V3 (*Information Technology Infraestructure Library*) y se presentan modelos dinámicos de simulación aplicados en el ámbito de los procesos *Generación de la Estrategia* y *Gestión de la Capacidad*. La finalidad principal de los modelos propuestos es, por un lado, ayudar a configurar adecuadamente los parámetros TI para garantizar el cumplimiento de los objetivos estratégicos del negocio; por el otro, facilitar la gestión de la capacidad de los servicios que los proveedores asignan a sus clientes con el objeto de asegurar el cumplimiento de los acuerdos de nivel de servicio establecidos en los SLAs.

Palabras clave: Reglas de Negocio, Gestión de servicios TI, Modelado y Simulación.

1 Introducción

En la actualidad, los servicios TI representan una parte sustancial de los procesos de negocio de las empresas y gestionarlos adecuadamente es fundamental para ofrecer servicios de calidad a un coste razonable que satisfagan las expectativas de los clientes y se correspondan con los objetivos del negocio.

La aplicación de las técnicas de modelado y simulación en el ámbito de la gestión de servicios de TI, ofrece la posibilidad de experimentar diferentes decisiones y analizar sus resultados en sistemas donde el coste o el riesgo de una experimentación real son prohibitivos. Por tanto, son herramientas de ayuda para resolver diversos problemas de la gestión de servicios TI o para la mejora de los procesos de gestión.

En este trabajo se ofrece una visión general de las posibles aplicaciones de las técnicas de modelado y simulación en el ámbito de los procesos de gestión de

servicios de ITIL V3 [7]. Asimismo, se presentan modelos dinámicos de simulación que se aplican en el ámbito del proceso *Generación de la Estrategia* del módulo *Estrategia del Servicio* y del proceso *Gestión de la Capacidad* del módulo *Diseño del Servicio*. El objetivo fundamental del modelo propuesto en el ámbito del proceso *Generación de la Estrategia* es ayudar a configurar adecuadamente los parámetros TI con la finalidad de asegurar el cumplimiento de los objetivos estratégicos del negocio. El propósito de los modelos propuestos en el ámbito del proceso *Gestión de la Capacidad* es ayudar a gestionar la capacidad de los servicios que los proveedores asignan a sus clientes con el objeto de garantizar el cumplimiento de los SLAs (*Service Level Agreements*).

El trabajo se estructura de la siguiente manera: el apartado 2 ofrece una visión general de la gestión de servicios TI. En el apartado 3 se explican los conceptos generales de las técnicas de modelado y simulación y se analizan las posibles aplicaciones de estas técnicas en el ámbito de los procesos de ITIL V3. En el apartado 4 se presentan modelos dinámicos de simulación construidos por los autores de este trabajo. Finalmente, el apartado 5 contiene las conclusiones y trabajos futuros.

2 Gestión de servicios TI

La gestión de servicios TI es la disciplina que se centra en la gestión de las personas, procesos y tecnologías que colaboran para asegurar la calidad de los servicios TI. Los principales objetivos de la gestión de servicios TI son alinear los servicios TI con las expectativas actuales y futuras de la empresa y sus clientes, garantizar y mejorar la calidad de los servicios y reducir su coste.

Existen varios modelos y estándares que ofrecen marcos de referencia de buenas prácticas en la gestión de servicios TI, tales como, *COBIT* [12], *CMMI para Servicios* (*CMMI-SVC*) [16], *ITIL V3* [13] y el estándar *ISO/IEC 20000* [5]. *ITIL* es uno de los modelos más utilizados en la actualidad y *COBIT*, *CMMI-SVC* y el estándar *ISO/IEC 20000* lo utilizan como referencia. Tomando como base las fases del ciclo de vida de los servicios, ITIL V3 estructura los procesos de gestión de servicios en cinco módulos: *Estrategia del Servicio*, *Diseño del Servicio*, *Transición del Servicio*, O*peración del Servicio* y *Mejora Continua del Servicio*.

3 Simulación y Gestión de Servicios TI

En este apartado se introducen los conceptos generales de las técnicas de modelado y simulación y se ofrece una visión general de las posibles aplicaciones de estas técnicas en el ámbito de los procesos de ITIL V3.

3.1. Conceptos generales de modelado y simulación

Un modelo de simulación es un modelo matemático que representa de forma simplificada un sistema dinámico y que se resuelve mediante la simulación. Estos

modelos permiten experimentar diferentes decisiones y analizar sus resultados en sistemas donde el coste o el riesgo de una experimentación real es muy elevado. La principal ventaja que presentan los modelos de simulación frente a los modelos analíticos es que permiten capturar y modelar el alto grado de incertidumbre que presentan algunos sistemas, así como representar sistemas que tienen estructuras e interacciones dinámicas complejas y modelar la realimentación de los sistemas.

El análisis de sensibilidad es una valiosa técnica de simulación que permite analizar los efectos que determinados rangos de valores de los parámetros de entrada del modelo tienen en las variables de salida. En consecuencia, ayuda a determinar el rango de resultados esperados cuando hay incertidumbre en los parámetros de entrada y facilita la identificación de los parámetros de entrada que más influyen en los resultados y que, por tanto, deben ser medidos y controlados con mayor precisión.

Existen diferentes enfoques de simulación tales como, modelos de procesos basados en estados, simulación discreta de eventos, sistemas dinámicos o simulación continua, modelos de redes de Petri, modelos de colas, así como enfoques híbridos que integran más de un enfoque de simulación para la obtención del modelo.

3.2. Aplicación de las técnicas de modelado y simulación en los procesos de ITIL

En este apartado se ofrece una visión general de las posibles aplicaciones de las técnicas de modelado y simulación en el ámbito de los procesos de ITIL V3. En la Tabla 1 se indican los principales procesos de ITIL V3 en los que es beneficioso aplicar la simulación, así como las referencias a aquellos trabajos que describen un caso de esta aplicación. Los autores de este trabajo han realizado un estudio de los ámbitos de aplicación de los trabajos referenciados lo que les ha permitido realizar la asociación a los procesos de ITIL que aquí se presenta.

Tabla 1. Aplicación de las técnicas de modelado y simulación en los procesos de ITIL V3

Módulo	Procesos
Estrategia del Servicio	Generación de la Estrategia [3],[10],[15] Gestión Financiera [4],[9]
Diseño del Servicio	Gestión de los Niveles de Servicio [1],[4],[9],[11] Gestión de la Capacidad [9],[11] Gestión de la Continuidad Gestión de la Disponibilidad Gestión de la Seguridad [14]
Transición del Servicio	Gestión de Cambios [15] Planificación y Soporte a la Transición [4]
Operación del Servicio	Cumplimiento de las Peticiones del Servicio [1],[11] Gestión de Incidentes [2],[6] Gestión de Problemas [6]
Mejora continua del Servicio	Mejora de los procesos [2]

- ***Módulo: Estrategia del servicio***

La *Estrategia del Servicio* es la fase de diseño, desarrollo e implementación de la gestión del servicio como un recurso estratégico. El objetivo fundamental de este módulo es definir las políticas y objetivos estratégicos de la organización. Los módulos *Diseño*, *Transición*, *Operación* y *Mejora Continua del Servicio* ponen en práctica la estrategia que resulte definida en este primer módulo a través de ajustes y cambios en la provisión de los servicios. Los modelos de simulación son esenciales en cualquier proceso de gestión estratégica dado que en estos procesos existe un elevado grado de incertidumbre en la predicción del futuro e influyen muchas variables que generalmente tienen una evolución desconocida. Los principales procesos de este módulo en los que es útil aplicar las técnicas de modelado y simulación son *Generación de la Estrategia* y *Gestión Financiera*.

Generación de la Estratégica

Este proceso es el responsable de la evaluación sistemática de la naturaleza de un negocio con la finalidad principal de definir los objetivos del negocio, desarrollar las estrategias necesarias para alcanzar estos objetivos y asignar adecuadamente los recursos para poder llevar a cabo estas estrategias y mejorar frente a los competidores. En el ámbito de este proceso, las técnicas de modelado y simulación ayudan en la determinación de los objetivos estratégicos de cualquier tipo de organización y en la evaluación del grado de cumplimiento de estos objetivos cuando se utilizan diferentes estrategias de negocio. Asimismo, permiten evaluar los costes, el valor y el riesgo asociado a la aplicación de las diferentes estrategias de negocio.

En [3] se presenta un modelo dinámico de simulación que permite preveer qué escenario es el más adecuado a 3 ó 5 años y validar si los objetivos estratégicos definidos en la organización son correctos. El modelo permite simular diferentes estrategias de obtención de los objetivos del negocio y ayuda a determinar la estrategia más adecuada. En [10] se propone un modelo dinámico de simulación que ayuda a determinar una configuración óptima de parámetros TI que garantice el cumplimiento de los objetivos del negocio. Los autores de [15] proponen utilizar el entorno de gestión de arquitecturas y servicios TI Adoit para modelar y simular los procesos de gestión de servicios con el objeto de identificar y medir el nivel de alineamiento existente entre los procesos de negocio y los servicios TI.

Gestión Financiera

El objetivo fundamental de este proceso es administrar de manera eficaz y rentable los servicios y las organizaciones TI. En el ámbito de este proceso, los modelos de simulación ayudan a determinar y analizar los costes reales asociados a la provisión de los servicios, a evaluar el retorno de la inversión y a analizar los riesgos asociados. El análisis de los resultados obtenidos en las simulaciones ayudará a los proveedores de servicios a tomar decisiones en el proceso de planificación de las inversiones con la finalidad de optimizar los costes y minimizar los riesgos asociados a la provisión de los servicios.

En [4] se presenta un modelo de redes de Petri que permite analizar y evaluar el coste total de los servicios considerando diferentes equipos de desarrollo de servicios y diferentes proveedores de servicios externos. El modelo dinámico de simulación propuesto en [9] permite evaluar las penalizaciones que tendrían que asumir los proveedores de servicios por el incumplimiento de los SLAs.

- ***Módulo: Diseño del Servicio***

La finalidad principal de este módulo es diseñar servicios TI apropiados que cumplan los requisitos actuales y futuros de la organización. Proporciona guías para diseñar y desarrollar los servicios y los procesos de gestión de servicios. Los principales procesos de este módulo en los que pueden aplicarse las técnicas de modelado y simulación son los siguientes: *Gestión de los Niveles de Servicio*, *Gestión de la Capacidad*, *Gestión de la Continuidad*, *Gestión de la Disponibilidad* y *Gestión de la Seguridad*.

Gestión de los Niveles de Servicio

El objetivo fundamental de este proceso es definir, negociar y supervisar la calidad de los servicios TI. La aplicación de los modelos de simulación en el ámbito de este proceso permite evaluar los efectos de las acciones de mejora llevadas a cabo en la organización en el cumplimiento de los SLAs. Por otro lado, ayudan a tomar decisiones en las actividades de asignación del personal y de los recursos necesarios para garantizar el cumplimiento de los SLAs y en el ámbito de la renegociación de los SLAs. Asimismo, permiten determinar la asignación de recursos más adecuada y evaluar los efectos de variar los parámetros de los SLAs en la calidad y en el coste de los servicios.

Los modelos dinámicos de simulación propuestos en [9] y [11] ayudan a tomar decisiones en la gestión de la capacidad de los servicios con el objeto de garantizar el cumplimiento de los SLAs. Asimismo, proporcionan información que puede utilizarse en la renegociación de los SLAs. En [1] se presenta un modelo dinámico de simulación que ayuda a decidir cómo asignar los recursos con el objeto de garantizar el cumplimiento de los SLAs. Los autores de [4] utilizan un modelo de redes de Petri para evaluar los efectos de diferentes selecciones de proveedores de los servicios en el cumplimiento de los SLAs.

Gestión de la Capacidad

Este proceso se encarga de proporcionar la capacidad necesaria para suministrar servicios de calidad a un coste razonable. Los modelos de simulación aplicados en el ámbito de este proceso permiten simular diferentes escenarios futuros previsibles con la finalidad de determinar los requisitos de capacidad necesarios para proporcionar, a un coste razonable, los niveles acordados de los servicios.

El modelo dinámico propuesto en [11] permite detectar situaciones en las que la capacidad del servicio contratada por un cliente es mayor que la necesaria para garantizar en cumplimiento de los SLAs. En estos casos, el proveedor del servicio podría asignarle al cliente una capacidad inferior a la contratada y reutilizarla con

otros clientes que demanden el mismo servicio. En [9] se presenta un modelo dinámico que permite estudiar los efectos de diferentes políticas de gestión de la capacidad de los servicios en el rendimiento de los mismos. Ambos modelos se aplican en el ámbito del proceso *Gestión de la Capacidad* de ITIL.

Gestión de la Continuidad

Los objetivos principales de este proceso son garantizar la rápida recuperación de los servicios tras una interrupción y establecer políticas y procedimientos que eviten las interrupciones de los servicios o minimicen sus consecuencias. La aplicación de la simulación en el ámbito de este proceso permite a los proveedores de servicios evaluar los riesgos y el impacto que tiene en el negocio la interrupción de los servicios y ayudan a determinar medidas preventivas que garanticen la continuidad del servicio. Por otro lado, permiten analizar los efectos de diferentes políticas de recuperación de servicios en el cumplimiento de los SLAs y ayudan a tomar decisiones en el proceso de asignación de los recursos necesarios para garantizar la continuidad del servicio. Actualmente, no se han encontrado trabajos que apliquen las técnicas de modelado y simulación en el ámbito de este proceso.

Gestión de la Disponibilidad

Este proceso es el responsable de optimizar y monitorizar los servicios con el objeto de que funcionen ininterrumpidamente y de manera fiable, y se garantice el cumplimiento de los SLAs. Los modelos de simulación aplicados en el ámbito de este proceso permiten analizar los efectos de los fallos de los componentes y de las interrupciones del servicio en el rendimiento del servicio y en el cumplimiento de los parámetros de disponibilidad del SLA (*tiempo máximo de recuperación*, *tiempo medio de fallos* y *tiempo medio entre incidentes del sistema)*. Por otro lado, permiten evaluar los efectos de las medidas de mejora de la infraestructura TI en la disponibilidad de los servicios. No se han encontrado trabajos actuales que apliquen la simulación en el ámbito de este proceso.

Gestión de la Seguridad

La finalidad de este proceso es diseñar una política de seguridad alineada con las necesidades del negocio, asegurar el cumplimiento de los estándares de seguridad y minimizar los riesgos de seguridad. La aplicación de las técnicas de modelado y simulación en el ámbito de este proceso permite evaluar los efectos de diferentes políticas de seguridad y protocolos de acceso a la información en el cumplimiento de los parámetros de seguridad acordados en los SLAs. Por otro lado, permiten preveer los niveles de seguridad que son necesarios establecer ante diferentes tendencias de las peticiones de los servicios, así como analizar los riesgos asociados e identificar vulnerabilidades. El análisis de los resultados obtenidos en las simulaciones ayuda a las organizaciones a tomar decisiones en la elaboración de los planes de seguridad.

En [14] se propone un modelo dinámico de simulación que ayuda a determinar los valores óptimos de los parámetros de gestión de la seguridad de los servicios y las interrelaciones que existen entre ellos.

- ***Módulo: Transición del Servicio***

Los procesos de este módulo son los responsables de desarrollar y mejorar las capacidades para el paso a producción de los nuevos servicios y de los servicios modificados. Los principales procesos en los que pueden aplicarse las técnicas de modelado y simulación son la *Gestión de Cambios* y la *Planificación y Soporte a la Transición*.

Gestión de Cambios

El principal objetivo de este proceso es la evaluación y planificación del proceso de cambio para asegurar que éstos se llevan a cabo de forma eficiente, siguiendo los procedimientos establecidos y asegurando la calidad y la continuidad del servicio. Los modelos de simulación aplicados en el ámbito de este proceso permiten analizar los costes asociados a los cambios realizados en la organización, evaluar el retorno de la inversión realizada y estudiar los efectos de los cambios en la calidad de los servicios. Por otro lado, ayudan a los proveedores de servicios a tomar decisiones en el proceso de planificación de los cambios para que éstos se realicen de forma eficiente y se garantice la calidad y la continuidad del servicio.

En [15] se modela y simula el proceso de *Gestión de Cambios* utilizando el entorno de gestión de arquitecturas y servicios TI Adoit. Los autores indican que la técnica propuesta es aplicable a otros procesos de gestión de servicios TI.

Planificación y Soporte a la Transición

Este proceso se encarga de planificar y coordinar los recursos para asegurar que las estrategias del servicio se llevan a cabo. Asimismo, es el responsable de identificar, gestionar y controlar los riesgos de fallos e interrupciones producidos durante las actividades de transición. La aplicación de la simulación en el ámbito de este proceso ayuda a los proveedores de servicios a decidir cómo planificar los recursos para garantizar la calidad de los servicios y el paso a producción en los plazos previstos y con los costes estimados.

En [4] se propone utilizar modelos de redes de Petri para ayudar a elegir, en tiempo de ejecución, a los proveedores de los servicios que componen otros servicios más complejos. Las simulaciones realizadas permiten evaluar los efectos que diferentes selecciones de proveedores tienen en la calidad y en los costes de los servicios.

- ***Módulo: Operación del Servicio***

La finalidad de los procesos de este módulo es garantizar la efectividad y eficacia en la provisión y soporte de los servicios con la finalidad de generar valor tanto para los clientes como para los proveedores de servicios. Los principales procesos de este módulo es los que se pueden aplicar las técnicas de modelado y simulación son el *Cumplimiento de las Peticiones del Servicio*, la *Gestión de Incidentes* y la *Gestión de Problemas*.

Cumplimiento de las Peticiones del Servicio

Este proceso es el responsable de permitir que los clientes soliciten y reciban servicios, así como de garantizar la provisión de servicios de calidad. Los modelos de simulación aplicados en el ámbito de este proceso permiten analizar el comportamiento de los servicios ante diferentes tendencias de las peticiones del servicio recibidas y tomar decisiones en la asignación de los recursos para garantizar la adecuada provisión de los servicios.

En [1] y [11] se proponen modelos dinámicos de simulación que ayudan a decidir cómo asignar los recursos ante diferentes tendencias de las peticiones del servicio.

Gestión de Incidencias y Gestión de Problemas

El proceso *Gestión de Incidencias* tiene como objetivo solucionar cualquier incidencia que provoque una interrupción en el servicio de la manera más rápida y eficaz posible, sin preocuparse de analizar sus causas. Por otro lado, el proceso *Gestión de Problemas* es el responsable de analizar las causas de las alteraciones en la provisión de los servicios, determinar las posibles soluciones a las mismas y proponer los cambios que sean necesarios para restablecer la calidad de los servicios.

La aplicación de los modelos de simulación en el ámbito de estos procesos permite estudiar el comportamiento de los servicios ante un determinado problema y analizar los efectos de las posibles soluciones del problema en la infraestructura TI, en los costes de la provisión del servicio y en el cumplimiento de los SLAs. Por otro lado, permiten analizar el coste y el beneficio de contratar personal especializado para la resolución de los problemas y ayudan a determinar qué medidas de mejora llevar a cabo para prevenir la aparición de nuevos problemas.

En [2] se presenta SYMIAN, una herramienta de ayuda a la decisión en el ámbito de la gestión de incidencias, basada en la simulación de eventos discretos. En [6] se propone un modelo dinámico de simulación aplicado en el ámbito de los procesos *Gestión de Incidencias* y *Gestión de Problemas* cuyo propósito fundamental es ayudar a gestionar los retrasos que se producen en un proceso sobre cuyas causas existe un cierto grado de incertidumbre.

- ***Módulo: Mejora Continua del Servicio***

Los procesos de este módulo se encargan de medir, evaluar y mejorar tanto la calidad de los servicios como el nivel de madurez de los procesos del ciclo de vida de los servicios. Las técnicas de modelado y simulación son de gran utilidad en el ámbito del proceso *Mejora de Procesos* de este módulo.

Mejora de Procesos

La finalidad principal de este proceso es mejorar la calidad de los servicios y el nivel de madurez de los procesos de gestión de servicios. La simulación facilita la toma de decisiones en el ámbito de la mejora de los procesos de gestión ya que permite predecir el impacto de un cambio en el proceso antes de que éste se realice. La herramienta de simulación de eventos discretos que se presenta en [2] se ha

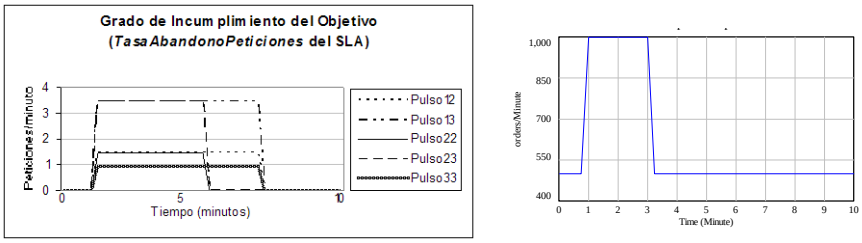
diseñado con la finalidad de evaluar y optimizar el proceso de *Gestión de Incidencias* de las organizaciones.

4 Modelos dinámicos de simulación construidos

En este apartado se presentan los modelos dinámicos de simulación que los autores de este trabajo han realizado en el ámbito de los procesos *Generación de la Estrategia* [10] y *Gestión de la Capacidad* [9],[11].

4.1. Modelos dinámicos de simulación en el proceso *Generación de la Estrategia*

El modelo dinámico propuesto en [10] permite simular las reglas de negocio de una organización y ayuda a decidir cómo configurar los parámetros TI para garantizar el cumplimiento de los objetivos estratégicos del negocio. En el caso de estudio considerado en este trabajo, el modelo permite determinar la menor capacidad del servicio que garantiza el cumplimiento de la siguiente regla de negocio de una empresa proveedora de servicios: "El *número de peticiones del servicio canceladas por sobrepasar el tiempo de respuesta acordado tiene que ser menor o igual al 15% de las peticiones del servicio recibidas*". Los escenarios de simulación se han configurado variando la capacidad del servicio y la tendencia que presentan las peticiones del servicio recibidas por parte de un cliente.



1a **1b**

Fig. 1. *Tendencia de las Peticiones Recibidas* (1a) y *Grado de Incumplimiento del Objetivo del Negocio* (1b)

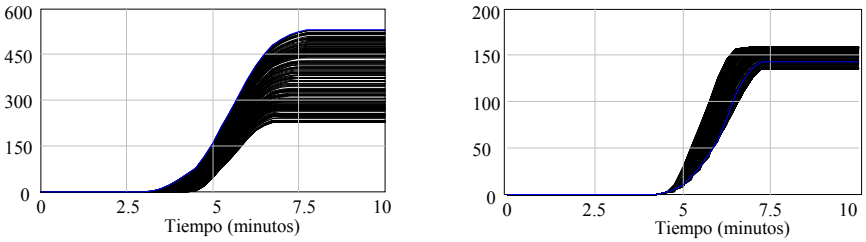
Para ilustrar estas ideas, en la Figura 1a se muestra la variable de salida *Grado de Incumplimiento del Objetivo del Negocio* obtenida en las simulaciones que se han realizado considerando que las peticiones del servicio presentan la tendencia que se muestra en la Figura 1b. El análisis de los resultados obtenidos permite determinar la menor capacidad del servicio que garantiza el cumplimiento de la regla de negocio. Por otro lado, también ayuda a evaluar la desviación respecto del objetivo del negocio cuando se utilizan capacidades del servicio inferiores a ésta que se produce. En el estudio realizado en [10] se concluye que la desviación respecto del objetivo del negocio depende de las características del pulso que modela la tendencia de las peticiones del servicio recibidas.

4.2. Modelos dinámicos de simulación en el proceso *Gestión de la Capacidad*

En [11] se propone un modelo dinámico de simulación que ayuda a gestionar la capacidad de los servicios que los proveedores asignan a sus clientes con la finalidad de garantizar el cumplimiento de los SLAs. La utilidad principal de este modelo es ayudar a los proveedores de servicios a detectar situaciones en las que la capacidad del servicio contratada por un cliente es mayor que la necesaria para garantizar el cumplimiento de los SLAs. En estos casos, el proveedor podría asignarle al cliente una capacidad inferior a la contratada y reutilizarla para otros clientes que demanden el mismo servicio, favoreciéndose así una mejor gestión de sus recursos.

El modelo propuesto en [9] es una ampliación del propuesto en [11] y permite estudiar los efectos de diferentes políticas de gestión de la capacidad de los servicios que los proveedores asignan a sus clientes en el cumplimiento de los SLAs. En concreto, permite analizar si el rendimiento real del servicio es el acordado con el cliente y evaluar la penalización que tendría que asumir el proveedor por el incumplimiento de los tiempos de respuesta establecidos (*tiempo de respuesta esperado* y *tiempo de respuesta máximo*). Los escenarios de simulación se han configurado variando la tendencia que presentan las peticiones del servicio recibidas, los acuerdos de nivel de servicio establecidos, la política de gestión de la capacidad del servicio y los parámetros de gestión de esta capacidad. El análisis de los resultados obtenidos en las simulaciones permite realizar los siguientes estudios:

- Evaluar el cumplimiento de los parámetros del SLA con la capacidad del servicio contratada por el cliente.
- Determinar qué capacidades del servicio y qué parámetros de gestión de esta capacidad garantizan que el rendimiento del servicio es el acordado con el cliente.
- Evaluar la penalización que tendría que asumir el proveedor del servicio por el incumplimiento de los tiempos de respuesta establecidos en los SLAs.
- Comparar los efectos de diferentes políticas de gestión de la capacidad del servicio en el cumplimiento de los SLAs y determinar qué política es más adecuada.
- Evaluar los efectos de variar los parámetros del SLA en el rendimiento real del servicio. Los resultados de las simulaciones pueden servir de base de la renegociación del SLA que el proveedor ha firmado con el cliente.
- Analizar los efectos de las políticas de gestión de la capacidad en el cumplimiento de los SLAs con diferentes tendencias de las peticiones del servicio recibidas.



2a **2b**

Fig. 2. Variable salida *PenalizaciónIncumplimientoTiemposRespuesta Políticas A* (2a) y *B* (2b).

Para ilustrar estas ideas, la Figura 2 muestra la variable de salida *Penalización por Incumplimiento de los Tiempos de Respuesta* obtenida en los análisis de sensibilidad que se han realizado considerando dos políticas diferentes de gestión de la capacidad del servicio (*Políticas A* y *B*). El análisis de los resultados obtenidos indica que la penalización que tendría que asumir el proveedor es mayor con la *Política A* que con la *Política B*. Asimismo, el carácter dinámico del modelo permite determinar que el proveedor es penalizado antes con la *Política A* (2a) que con la *Política B* (2b).

5 Conclusiones y trabajos futuros

En este trabajo se presentan un conjunto de los resultados de una iniciativa de investigación que se centra en la utilización de las técnicas de modelado y simulación en los procesos de gestión de servicios TI. Se ofrece una visión general de las posibles aplicaciones de estas técnicas en el ámbito de los procesos de ITIL V3 y se referencian trabajos actuales que describen un caso de esta aplicación. Finalmente, se presentan modelos dinámicos de simulación que se aplican en el ámbito de los procesos *Generación de la Estrategia* y *Gestión de la Capacidad*.

En el ámbito de la gestión de servicios TI, la simulación ayuda a determinar los objetivos estratégicos de las organizaciones y a evaluar el grado de cumplimiento de estos objetivos. Por otro lado, facilitan la toma de decisiones en el proceso de planificación de las inversiones con la finalidad de optimizar los costes y minimizar los riesgos asociados a la provisión de los servicios. Asimismo, ayudan a tomar decisiones en los procesos de diseño, operación y transición del servicio con el objeto de suministrar servicios de calidad a un coste razonable, de garantizar el cumplimiento de los SLAs y de asegurar la continuidad, la disponibilidad y la seguridad de los servicios. Estas técnicas también facilitan la toma de decisiones en el ámbito de la mejora de los procesos de gestión de servicios ya que permiten predecir el impacto de un cambio en el proceso antes de que éste se realice.

La finalidad principal del modelo dinámico de simulación propuesto en el ámbito del proceso *Generación de la Estrategia* es simular las reglas de negocio de una organización y ayudar a determinar la configuración óptima de parámetros TI que garantiza el cumplimiento de los objetivos estratégicos del negocio. El objetivo de los modelos propuestos en el ámbito del proceso *Gestión de la Capacidad* es ayudar a gestionar la capacidad de los servicios que los proveedores asignan a sus clientes con la finalidad de garantizar el cumplimiento de los SLAs.

Nuestros trabajos fututos se centran principalmente en ampliar estos modelos añadiéndoles otros parámetros y variables que influyen en los procesos *Generación de la Estrategia* y *Gestión de la Capacidad*, así como en desarrollar nuevos modelos que ayuden a la toma de decisiones en diferentes procesos de gestión de servicios TI. Finalmente, es nuestro objetivo aplicar los modelos en organizaciones que proporcionen servicios TI al fin de obtener información adicional que permita la mejora de los modelos, así como su utilización en el ámbito de la toma de decisiones en dichas organizaciones.

Agradecimientos

Esta investigación está parcialmente financiada por el Ministerio de Educación y Ciencia y por los fondos europeos FEDER (proyecto TIN2007-67843-C06-04).

Referencias

1. An, L.,Jeng,J.J.: Web Services Management Using System Dynamics. En Proceedings of the IEEE International Conference on Web Services (ICWS 2005), pp.347-354.
2. Bartolini, C., Stefanelli, C., Tortonesi, M.:SYMIAN: A Simulation Tool for the Optimization of the IT Incident Management Process. En Proceedings of the 19th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management: Managing Large-Scale Service Deployment (2008), pp. 83-94.
3. Folgueras, A., Sáenz F.J., De la Cámara, M.: Técnicas de Modelado de los Costes Variables aplicados al Proceso de Planificación Estratégica con Filosofía Ciclo de Vida de Servicios TI. En línea http://www.uc3m.es/portal/page/portal/congresos_jornadas/congreso_itsmf.
4. Grabarnik, G.,Ludwig,H., Schwartz,L.: Management of Service Process QoS in a Service Provider – Service Supplier Environment. En Proceedings of the 9th IEEE International Conference on E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (CEC-EEE 2007), pp. 543-550.
5. ISO/IEC 20000-1:2005.Information Technology–Service Management, Part 1: Specification, Part 2: Code of Practice. ISO/IEC (2005).
6. Lee J.H., Han, Y.S., Kum C.H.: IT Service Management Case based Simulation Analysis & Design: Systems Dynamics Approach. En Proceedings of the IEEE International Conference on Convergence Information Technology, En Proceedings of the IEEE Computer Society (2007), pp. 1559-1566.
7. Office of Government Commerce: The Official Introduction to the ITIL Service Lifecycle. The Stationary Office (TSO) (2007).
8. Office of Government Commerce: Service Design. The Stationary Office (TSO) (2007).
9. Orta, E., Ruiz, M., Toro, M.: Análisis de los Efectos de las Políticas de Gestión de la Capacidad de los Servicios en el cumplimiento de los SLAs utilizando Simulación. En Actas de las XIV Jornadas en Ingeniería del Software y Bases de Datos (JISBD 2009), San Sebastián, del 8 al 11 de septiembre de 2009. Artículo aceptado, pendiente de publicación.
10. Orta, E., Ruiz, M., Toro, M.: Analyzing Strategic Business Rules through Simulation Modeling. En Proceedings of the IFIP Conference on E-Business, E-Services and E-Society (I3E 2009), Nancy del 23 al 25 de septiembre. Artículo aceptado, pendiente de publicación.
11. Orta, E., Ruiz, M., Toro, M.: Aplicación de las Técnicas de Modelado y Simulación en la Gestión de la Capacidad de los Servicios TI. En Actas de las XIII Jornadas en Ingeniería del Software y Bases de Datos (JISBD 2008), pp 299-310.
12. Página oficial de COBIT: http://www.isaca.org/cobit.
13. Página oficial de ITIL: http://www.tso.co.uk/ITIL.
14. Sarriegi, J.M., Santos, J., Torres, J.M., Imizcoz, D., Plandolit, A.L.: Modeling Security Management of Information Systems: Analysis of a Ongoing Practical Case. En línea http://systemdynamics.org/conferences/2006/proceed/papers/SARRI206.pdf.
15. Silva, E., Chaix, Y.: Business and IT Governance Alignment. Simulation Essay on a Business Process and IT Service Model. En Proceedings of the 41st Hawaii International Conference on System Science (2008), pp. 434-444.
16. Software Engineering Institute (Carnegie Mellon): CMMI for Services Model. http://www.sei.cmu.edu/pub/documents/09.reports/09tr001.pdf

COMPETITIVE INTELLIGENCE BASED ON SOCIAL NETWORKS FOR DECISION MAKING

Fco Fernando de la Rosa Troyano[1], María Teresa Gómez López[1]and Rafael Martínez Gasca[1]

[1] Departamento de Lenguajes y Sistemas Informáticos. Dpto. Lenguajes y sistemas informaticos. University of Seville, Spain
{ffrosat, maytegomez, gasca}@us.es

Abstract. In previous works a framework has been presented to extract from internet the scientific community interested in a specific topic. The process uses search engines query results and e-mails address co-occurrences to obtain the invisible colleges and subtopics of a community. This work presents the use of this technique to implement several competitive intelligence tasks to help in decision making in a research area. In order to show an illustrative purpose, this technique is applied to analyze the social network of participants in several Veille Stratégique Scientifique & Technologique editions.

Keywords: Social networks extraction, competitive intelligence, search engine mining, emails address mining.

1 Introduction

One of the consequences caused by internet expansion is the exponential growth of public information. This information is stored in a set of interlinked heterogeneous sources. Search engines play a key role in internet searching information, but the general approaches to analyze the information systems cannot integrate different sources. The analysis of the stored information in a systematic and automatic way can help to decision making in competitive intelligence. In this context, the co-occurrences analysis becomes imperative to implement intelligence competitive tasks.

In previous works a process to extract automatically the scientific community interested in a specific topic was presented. The process carries out a systematic search engines queries. These queries are specially designed to extract the goal network through a posterior analysis of the search engines requests. Analysing this extracted information is also possible to determine the community subtopics of interest. This paper discusses the use of this technique to implement several competitive intelligence tasks. For example: searching for experts, gathering of information, organizations collaborations analysis, countries collaborations analysis or products impacts analysis.

The paper is divided into the following sections. Section 2 presents the related works and compares them with our proposal. Section 3 presents the Social networks topic-driven extraction algorithm. Section 4 analyzes some Competitive Intelligence Tasks. In order to illustrate the purpose, the defined process is applied to extract the social network of participants in the Veille Stratégique Scientifique & Technologique (VSST). Finally, conclusions and future work are presented.

2 Related Works

Previous works have defined social networks extraction processes for different information web sources: search engines [8][10][12], chats [14], DBLP [5], FOAF archives [12][13], SourceForge [1], mailing lists [2], etc. The review of this paper is restricted to approaches using search engines to extract social networks. One of the systems that uses search engines to extract social networks is REFERRAL WEB [8] which was designed to use Altavista engine, the obtained network is focused on a specific person (egocentric network). For this reason, it only needs to know the name of the ego. By mean of an entity recognition system extracts a list of related people. To measure the significance of the relationship between X and Y (the variable Y contains any of the persons related to X) uses the Jaccard coefficient [7]. The above process may be repeated recursively.

Recently two systems POLYPHONET [10][11] and FLINK [12][13] were launched, these systems obtain the social network using a list of the members of a determined community. The basic algorithm of both systems must do a query for each pair X and Y to create the affinity matrix (X and Y are two different names of the list). Both systems use a threshold to determine when relationships are meaningful. These systems have several disadvantages, for example the calculated affinities are difficult to understand. The terms co-occurrence on the indexed pages may be due to several factors: co-authorship, participation in the same event (i.e., program committee), referenced in the same paper, etc. By using personal names in the query could add ambiguity to the results, since the probability that the name refers to more than one person is high. Sometimes several names reference to the same person (i.e., 'Rafael Martínez Gasca' or 'R.M. Gasca'). Another disadvantage is the high cost of extracting the social network, since in the worst case for each pair of members a query must be performed. This is a major problem given that licenses to use search engines may have limitations. For example, Google does not allow more than 1000 queries per day (to calculate a network of 500 actors would need 125 days). The scalable algorithm implementations [11] have reduced the numbers of necessary queries to complete a whole network (according to the author, for 503 actors 19,852 queries are needed, 20 days).

The reviewed works use lists of names to extract social networks. In contrast there is the possibility of replacing the lists of names by e-mails address. In [9] is described

an algorithm to extracting social relationships of a specific internet domain using e-mails address. This algorithm is used for social engineering attacks in security of computer systems area and has a high cost of extracting the social network, for each pair of e-mail address a query must be performed. This paper proposes a social network extraction algorithm based on e-mails to improve the complexity of reviewed algorithms. The query model that is used makes it more robust to ambiguity, and allows a clear interpretation of the relationships extracted. It does not use learning process. Unlike previous work [5], the proposed algorithm does not need an initial list of e-mails and it uses heuristics for driving the construction of a social network by topic and by importance of the network members.

The practice cases presented in this paper have been developed with TREDAR tool. This tool permits to define business processes in an interactive way. The specific analyzed case is a competitive intelligence process [19] that permits recollect and extract data from internet about VSST community and perform different types of analyses for the decision making to solve different queries. This tool also allows us to spread and visualize the obtained data to decision-making support. In this way, all the necessary stages to provide a vigilance service and competitive intelligence are integrated through web technology.

3 Social networks topic-driven extraction algorithm

In this section the algorithm to social networks topic-driven extractions formalize in [6] is described. The algorithm is divided into three steps:

- To seed e-mails address extraction: To start the extraction process is necessary to have a set of e-mails. Different processes for extracting e-mail addresses from web can be defined. The e-mails address selection process used by de algorithm is described below:
  - To perform queries with the '<topic>' and visit the web documents returned by the search engine
  - To extract the e-mails from the <a> tags of the html pages and
  - For each e-mail address, to check through the query '<email>' '<topic>' the association degree of e-mail to the topic. The process considers a high association degree if the number of documents returned by the previous query exceeds a threshold.
- To expand the e-mail address: The algorithm expands the social network with new relationships extracted from search engine queries result. For each mail address the algorithm do:
  - To create a query using the schema: '<username>' '<domain>' filetype:pdf. For example, the username of the e-mail address ffrosat@ us.es is ffrosat and the domain is us.es, hence the query is: 'ffrosat' 'us.es' filetype:pdf and extracts the result contexts.

  - The contexts were analyzed using regular expressions, to analyze only the contexts in which verify the e-mail appearance. Each new e-mail that appears in the context is added to the social network as a new node and the relationships are added. The nodes and the relationships are associated with a counter which will represent their importance in the network. If any e-mail address appears again at some context, the counter will increase by one unit, also the relationships.
- Social networks topic-driven: This principal process is iterative and stores the e-mails address in a priority queue.
  - Initially, the queue is initialized with the e-mails address seeds.
  - In each iteration an e-mails address is extracted from the queue top, and it is checked the association degree of the e-mail address to the topic. If the association degree exceeds a threshold, then it is expanded with the neighbors of the e-mails address.
  - The iteration is repeated until the queue is empty and other strategies can be implemented, for example limiting the number of web pages visited or the number of emails-address of the social network.
  - After completing the expansion, the new nodes are added to the queue.

Although it is possible that a person can have more than one e-mail address (for example a personal and a institutional e-mail address), it is also true that an e-mail address typically identify a person. It could avoid the ambiguity problems of other methodologies, but not the variety problems. Rearranging the queue several driven strategies can be implemented: maximizing the association degree of e-mail to the topic or other social networks analysis measures such as the degree, pagerank or the betweenness.

4 Competitive Intelligence Tasks

In order to illustrate the **social networks topic-driven extraction** technique, the 'VSST' and 'Interlligence competitive' topic has been used. The goal is the extraction of the social network of the community of the several Veille Stratégique Scientifique & Technologique (VSST) editions. The 327 seeds were extracted from different VSST web pages editions. After running the extraction algorithm, the social network had 2.107 nodes and 7.102 edges. Of these nodes, 432 nodes exceeded the minimum threshold, only these nodes were expanded.

Table 1. Numbers of documents and queries associated to each topic

Query	#Queries	#ndocs>0
<login> <domain> filetype :pdf	932	
<email> 'VSST'	932	377
<email> 'competitive intelligence'	932	367
<email> 'text mining'	932	435
TOTAL	3728 (4 days)	

The social network extracted can be used to find experts in certain topics, [15] uses the number of pages for develop experts ranking. In our case we can use the query <email> "<topic>" to estimate de number of documents associated and develop a topic ranking of experts, as it is shown in Table 1 and Table 2. There are alternatives to the impact measure such as Mindshare. The advantage that we have proposed in the experts finding task is the own of the social network, this allows using the ARS measures to classify the experts, for example: degree, authority, centrality, betweens, etc [16].

Table 2. Numbers of documents associated to each topic

VSST <email> "VSST"		COMPETITIVE INTELLIGENCE <email> "competitive intelligence"		TEXTMINING <email> "text mining"		MIN(VSST+IC,DEGREE)	
Bernard Dousset	86	David Doose	200	M. Boughanem	118	Bernard Dousset	0,525
Y. Bertacchini	41	Eric Andonoff	200	Josiane Mothe	104	Luc Quoniam	0,44
Odile Thiery	34	Luc Quoniam	125	Alessandro Zanasi	50	Bertacchini	0,32
Jacques Ducloy	26	Bernard Dousset	105	Luc Quoniam	43	Marisela Rodriguez-Salvador	0,295
Humbert Lesca	24	Bertacchini	95	Pascal Poncelet	31	Henri Dou	0,25
Amos David	22	Humbert Lesca	59	Stanley Loh	26	Yara Rezende	0,2
Henri Dou	19	Marisela Rodriguez-Salvador	59	Key-Sun Choi	25	Eric Andonoff	0,2
Luc Quoniam	10	Dou Henri	50	Jian-Yun NIE	25	Odile Thiery	0,19
Humbert Lesca	10	Yara Rezende	43	Alessandro Zanasi	24	Carlos Merino	0,16

An appropriate selection of email addresses can be used to model an area of interest and optimize the gatherer of information. For example, using the query <email> filetype:pdf to download pdf documents of experts in the area. This gathering of information can be used to refine the expert ranking. In order to analyze the global topics impact of the social network, it is possible analyzing the impact of products. In [5] this approach is used to analyze two ARS tools, Pajek and Ucinet. It concluded that the global impact of Ucinet on the social network was a 20% larger than Pajek. Probably the difference of impact is due to its usability. The mayor problem of this approach is the words ambiguity.

By using Figure 1 it is possible to analyze the cooperation relations between the countries of the VSST participants. In this case underlines the strategic position of France and his historical connecting with the Maghreb countries. And using Figure 2 is possible to analyze the cooperation between different organizations through their

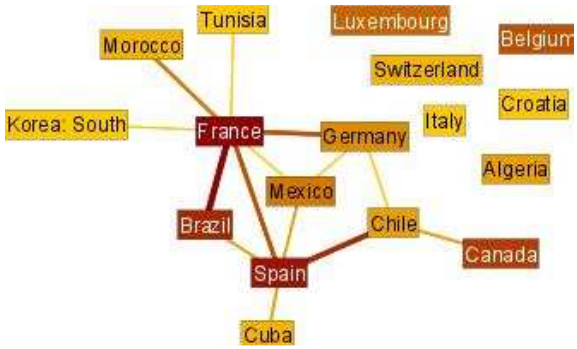


Fig. 1. VSST countries collaborations

domains. For example, analyze the domains we can discover new competitors (IALE, CDE, ISCOPE, IMCSLINE, etc), new clients (LAPOSTE, CEA, EADS, etc) or new investigation centers (IRIT, INIST, etc). And analyzing the relationships we can appreciate the central position of diverse universities in the network. Also using the IP domains is also possible to allocate the organizations [5].

Figure 3 represents in a lexical network the VSST community topics of interest. In order to build this map, two tasks have been executed: (1) the key words of the paper published in VSST events in 2007 have been extracted (2) the concurrency network has been calculated [18]. Using this map, it is possible to determine the interesting centers of the researcher that participate in the events. The follow topics of interest have been extracted in this example: text-mining, natural language processing, creativity and innovation, information retrieval and filtering, co-word analysis, business-intelligence.

The results for the VSST case study are online available in http://www.lsi.us.es/~ffrosat/index.php/Ffrosat/MapaListaVSST2007Es.

5 Conclusions and future work

In this paper some techniques have been combined helping in the making-decision process of an research organization. The topics that have been used are : extraction of information from the web, analysis of social networks and co-occurrences.

The queries that can be solved with these techniques are :

- Which are the experts in an specific area?
- Which are the most influence groups ?
- How the organizations are structured?
- Which are the most influential countries ?
- Which are the goal network?

This work provides a framework for making-decision processes to analyse the distributed data in different and heterogeneous sources, non centered in any database. Figure 4 shows in a visual way the main ideas developed in this work.

As future work we propose an automatic classification of the domains using the available information in the researh center webs. Also it will be interesting to develop techniques to extract information about the entities to be represented in the maps (names, departments, telephone numbers…)

Acknowledgments. This work has been partially funded by Junta de Andalucía (P08-TIC-04095).

References

1. K. Crowston and J. Howison. (2005) The Social Structure of Free and Open Source Software Development, Vol. 10, No. 2, First Monday.
2. A. Culotta, R. Bekkerman, and A. McCallum. (2004) 'Extracting social networks and contact information from e-mail and the web', Proceedings of the Conference on Email and Spam.
3. F. de la Rosa T., S. Pozo and R. M. Gasca. (2005) 'Análisis y visualización de comunidades científicas con información Extraída de la web', IEEE Latin America Transactions, Vol. 3, No. 1, ISSN: 1548-0992.
4. F. de la Rosa T, R. M. Gasca, L. González y F. Velasco (2005). Análisis de Redes Sociales mediante Diagramas Estratégicos y Diagramas Estructurales. Redes: Revista Hispana para el Análisis de Redes Sociales. ISNN: 1579-0185. Vol. 8.
5. F. de la Rosa T., F. and Gasca, R.M. (2007) 'Sistemas de inteligencia web basados en redes sociales', Revista Hispana para el Análisis de Redes Sociales, Vol. 12, ISSN: 1579-0185.
6. F. de la Rosa T. and R. M. Gasca (2008) Automatic extraction of social networks by topics of interest. IJCAT , Vol. 33 , Nr. 4, p. 292-299.
7. P. Jaccard (1901) 'Étude comparative de la distribution florale dans une portion des Alpes et des Jura', Bull Soc Vaudoise Sci Nat, Vol. 37, pp.547–579.
8. H. Kautz, B. Selman and M. Shah (1997) 'The hidden web', AI Magazine, Vol. 18, No. 2, pp.27–35.
9. J. Long (2005) The Google Hacker's Guide, Retrieved, ISBN 1-59749-176-4.
10. Y. Matsuo, H. Tomobe, K. Hasida, and M. Ishizuka. (2003) 'Mining social network of conference participants from the web', Proceedings of the International Conference on Web Intelligence, pp.190–194.
11. Y. Matsuo, J. Mori, M. Hamasaki, H. Takeda, T. Nishimura, K. Hashida and M. Ishizuka. (2006) 'Polyphonet: an advanced social network extraction system, Proceedings of WWW2006.

12. P. Mika. (2004) 'Bootstrapping the FOAF-Web: an experiment in social network mining', Proc. 1st Workshop Friend of a Friend, Social Networking and the Semantic Web.
13. P. Mika. (2005) 'Flink: Semantic web technology for the extraction and analysis of social networks', Journal of Web Semantics, Vol. 3, No. 2.
14. P. Mutton. (2004) Inferring and Visualising Social Networks on Internet Relay Chat, InfoVis, Austin, TX, pp.35–43.
15. Da Silva, A, Manniana, B., Quoniam, L. and Rostaing, H. 'Searching for Experts on the Internet' Competitive Intelligence Rewiew, USA, v. 11, n. 4, 2000.
16. Scott, J. P. (2000) 'Social Network Analysis: A Handbook. Second edition' Sage Publications.
17. X. Canaleta, P. Ros, A. Vallejo, D. Vernet and A. Zaballos. "A system to extract social networks based on the processing of information obtained from Internet", Proceedings of the 11th International Conference of the Catalan Association for Artificial Intelligence(CCIA 2008), IOS Press, ISBN: 978-1-58603-925-7, 2008
18. N. Coulter, I. Monarch and S. Konda. (1998). "Software engineering as seen through its research literature: A study in co-word analysis". Journal of the American Society for Information Science, 49(13), pp. 1206-1223.
19. P. Escorsa P.y R. Maspons (2001), De la Vigilancia Tecnológica a la Inteligencia Competitiva. Madrid. Prentice Hall.
20. V.A. Bucheli y F. González (2007) Herramienta informática para vigilancia VIGTECH-. Avances en Sistemas e Informática, v. 4 n.1, pp 117-126, Junio 2007, Facultad de Minas, Universidad Nacional de Colombia, Medellín

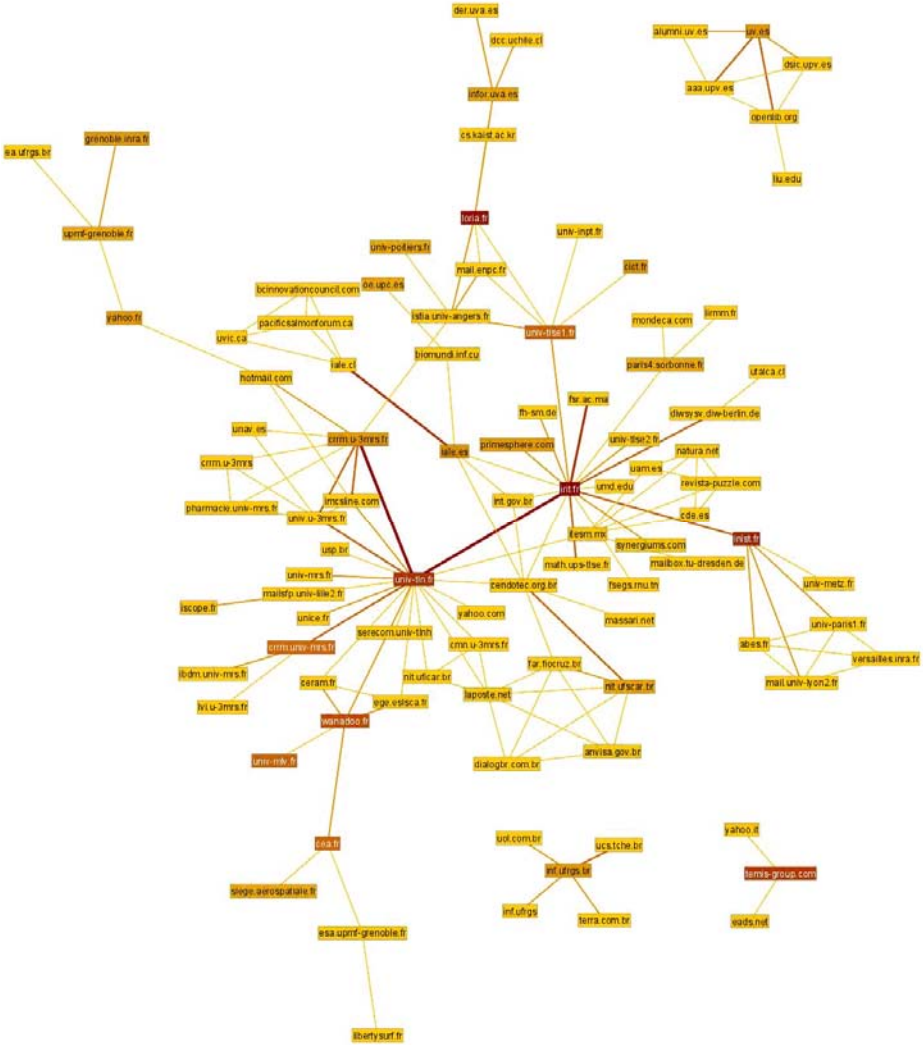


Fig. 2. VSST organizations collaborations

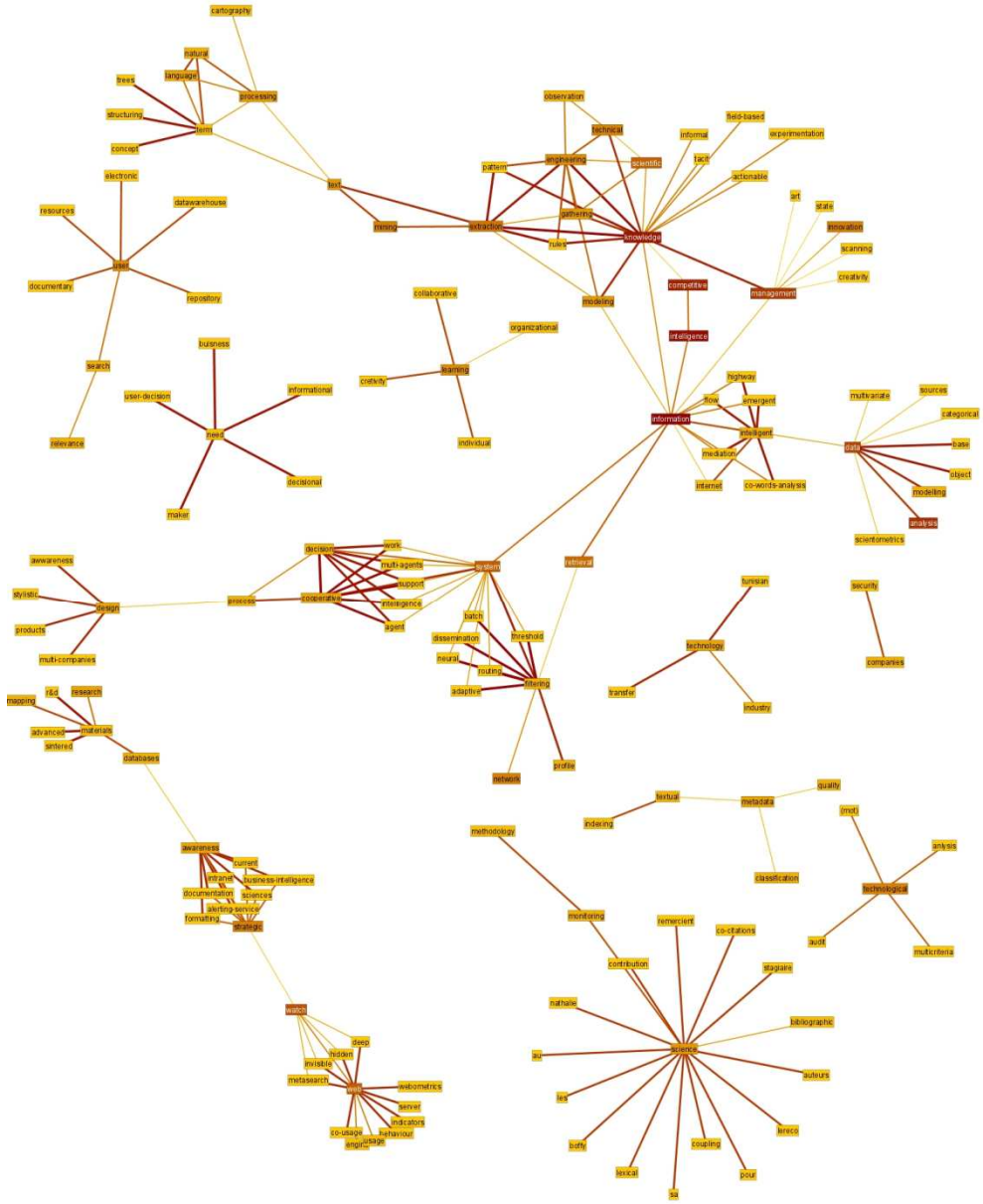


Fig. 3. VSST topics of interest

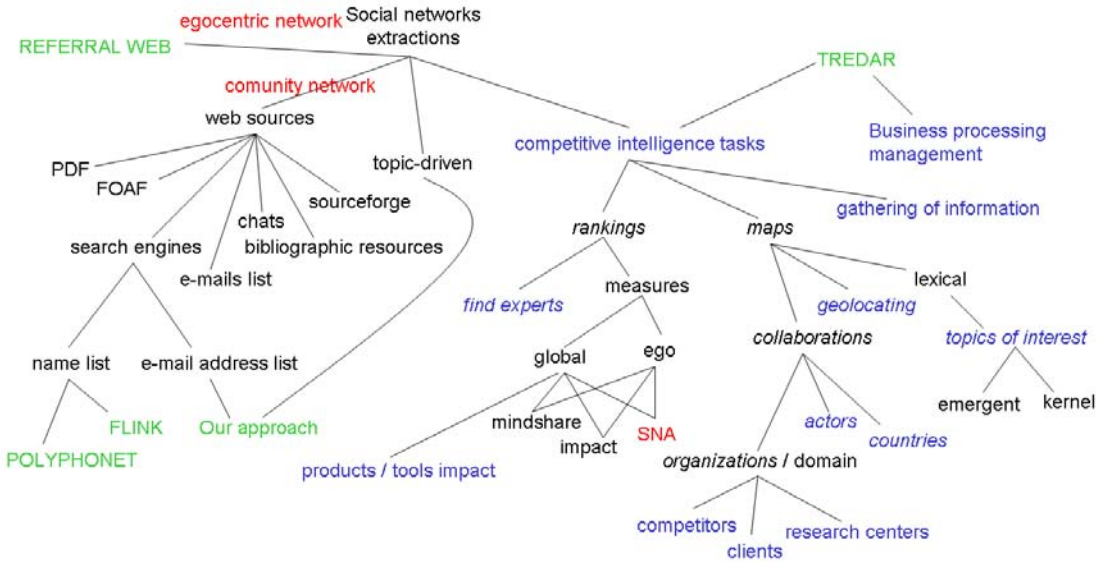


Fig. 4. Work mind map

Oráculos de prueba: Un planteamiento heurístico de apoyo a decisión

Arturo H. Torres[1], María J. Escalona[1], Manuel Mejías[1], Javier J. Gutiérrez[1]

[1] Departamento de Lenguajes y Sistemas Informáticos, Universidad de Sevilla,
Avd. Reina Mercedes sn. 41040 Sevilla, España
arturoh.torres.exts®juntadeandalucia.es
{escalona,risoto,javierj}@lsi.us.es

Resumen. Uno de los mayores desafíos hacia la consecución de las pruebas 100% automáticas está relacionado con la obtención de un eficiente oráculo de pruebas. La cuestión referente a decidir si el resultado de una prueba es aceptable o no, está relacionado estrictamente con la planificación de las pruebas, y específicamente al problema de cómo derivar los casos de prueba. Esto corresponde a lo que es denominado "oráculo", idealmente es un método que provee las salidas esperadas de cada caso de prueba dado. Este trabajo presenta un planteamiento heurístico inicial que apoya a la decisión de la aceptabilidad de una prueba en el ámbito de las aplicaciones Web.

Palabras clave: Oráculos de prueba, pruebas automáticas, pruebas de software.

1 Introducción

La creciente complejidad de las aplicaciones Web ha causado que el campo de la Ingeniería Web, definida como la aplicación sistemática, disciplinada y cuantificable de aproximaciones para el desarrollo y evolución eficiente de aplicaciones de alta calidad en la World Wide Web [8] se haya desarrollado de manera muy acelerada.

Desafortunadamente, dicha complejidad no parece estar acompañada de los mecanismos adecuados que garanticen la calidad de unos sistemas de los que cada día existe mayor dependencia a nivel social, funcional y económico.

Esta carencia de calidad ha venido generando una preocupación creciente entre la comunidad científica y empresarial involucrada en el desarrollo Web. Así pues, en los últimos años surgen varias iniciativas con el objetivo de definir marcos de referencia adecuados a estas nuevas tendencias de creación de software. Dentro de estos marcos de referencia se encuentra las pruebas de software destinados a este tipo de aplicaciones. Debido a la rapidez de crecimiento y desarrollo de las aplicaciones Web, se ha necesario e indispensable un proceso de pruebas eficiente y que tienda a ser lo más automatizado posible, consiguiendo de esta manera un ahorro de tiempo y costos.

Para asegurar la calidad del software, la técnica de pruebas es uno de los métodos más eficaces. De entre los desafíos más importantes de las pruebas de software con el objetivo de automatizar las pruebas, se encuentra el oráculo de pruebas. Es decir, la cuestión referente a decidir si el resultado de una prueba es aceptable o no, la cual está relacionada estrictamente con la planificación de las pruebas, y concretamente al problema de cómo derivar los casos de prueba.

Este desafío, el de obtener un oráculo eficiente, es un tema interesante para ser planteado y discutido. En principio, las redes neuronales son una opción válida de estudio para soportar nuestro oráculo. Es decir, las tareas de decisión que actualmente la mayoría de probadores realiza, en cuanto si el resultado de una prueba es válido o no, serían soportadas por métodos heurísticos. El objetivo de este trabajo es sólo dar un planeamiento inicial heurístico que nos permita abrir un estudio sobre la viabilidad de la utilización de técnicas de decisión heurísticas para la obtención de oráculos de prueba automáticos.

La estructura del trabajo continúa con una exposición sobre los desafíos de las pruebas de software (sección 2), útiles para ubicar la importancia de los oráculos de prueba como premisas para la obtención de pruebas 100% automáticas. Seguidamente, en la sección 3 se presenta de manera más detallada lo que se persigue con las pruebas automáticas. En la sección 4 se presenta los oráculos de prueba y sus características. Luego, en la sección 5 se presenta el planteamiento inicial heurístico de los oráculos de prueba. Finalmente, la sección 6 presenta las conclusiones y trabajos futuros.

2 Desafíos de las pruebas de software

Las pruebas de software son un término bastante amplio y abarcan una extensa gama de actividades muy diferentes, desde las pruebas realizadas por el desarrollador de una pequeña pieza de código (pruebas unitarias), hasta la validación del cliente de un gran sistema de información (pruebas de aceptación).

En todas estas fases, los casos de prueba pueden ser concebidos con objetivos muy variados, tales como validar si existen desviaciones en los requisitos del usuario, evaluar la conformidad de una especificación estándar, evaluar la robustez de las condiciones de carga, o de entradas maliciosas, medir atributos como el desempeño o usabilidad, estimar la confianza operacional, etc. Además, la actividad de las pruebas podrían ser llevadas a cabo por diversos procedimientos formales, tales como la planificación y documentación rigurosa, o como las informales y ad hoc (pruebas de exploración).

Como consecuencia de esta variedad de objetivos y ámbitos, se plantean una multiplicidad de términos para las pruebas de software, lo cual ha generado confusión y muchos problemas en la investigación sobre pruebas de software.

Para aclararlo y organizarlo en una vista unificada, presentamos la propuesta de Bertolino [2] acerca de clasificación de los problemas comunes y de los muchos significados de las pruebas de software. El primer concepto a capturar es el encontrar el denominador común, si existe, entre todas las posibles facetas de las pruebas.

Bertolino propone que el denominador común puede ser una vista muy abstracta. Dada una pieza de software (cualquiera que sea en tipología, tamaño y dominio) las pruebas siempre consisten en observar una muestra de las ejecuciones de las pruebas, y dar un veredicto sobre ellos.

A partir de esta visión general, se pueden concretar diferentes casos, distinguiendo los aspectos específicos que pueden caracterizar la muestra observada.

Una vez distinguidos los aspectos específicos que pueden caracterizar a la muestra observada, es necesario tener unas orientaciones acerca de cuál es el estado actual de las propuestas relacionadas con las pruebas de software y hacia dónde se deberían de dirigir; es decir, precisamos de un *roadmap*.

Un plan de trabajo proporciona la orientación para llegar al destino deseado, a partir del punto "tú estás aquí". El *roadmap* de la investigación de las pruebas de software está organizado de la siguiente forma:

1. El punto "tú estás aquí" consiste de los últimos logros de la investigación (tomando en cuenta que algunos de estos esfuerzos están todavía en curso).
2. El destino deseado se representa en la forma de un conjunto de sueños: se usa este término, para indicar que son metas asintóticas. Son, por definición, inalcanzables y su valor se mantiene exactamente como los polos de atracción para las futuras investigaciones.
3. En el medio están los desafíos de las actuales y futuras investigaciones de pruebas de software. Estos desafíos constituyen las orientaciones a ser seguidas, en el camino hacia los "sueños", y como tales, es la parte más importante del *roadmap*.

El *roadmap* está ilustrado en la Figura 1. Dentro de él, en el centro se sitúan las investigaciones en curso y las investigaciones emergentes, con muchos tópicos maduros (los logros) sobre la izquierda, y sobre la derecha las últimas metas (los sueños). Cuatro tiras horizontales representan las rutas de investigación identificadas hacia los "sueños":

1. Teoría de pruebas universal.
2. Pruebas basadas en modelos.
3. Pruebas 100% automáticas.
4. Ingeniería de pruebas con eficacia maximizada.

Los desafíos horizontales corresponden a las interrogantes *WHY*, *HOW*, *HOW MUCH*, *WHAT*, *WHERE*, y *WHEN* sin un orden específico.

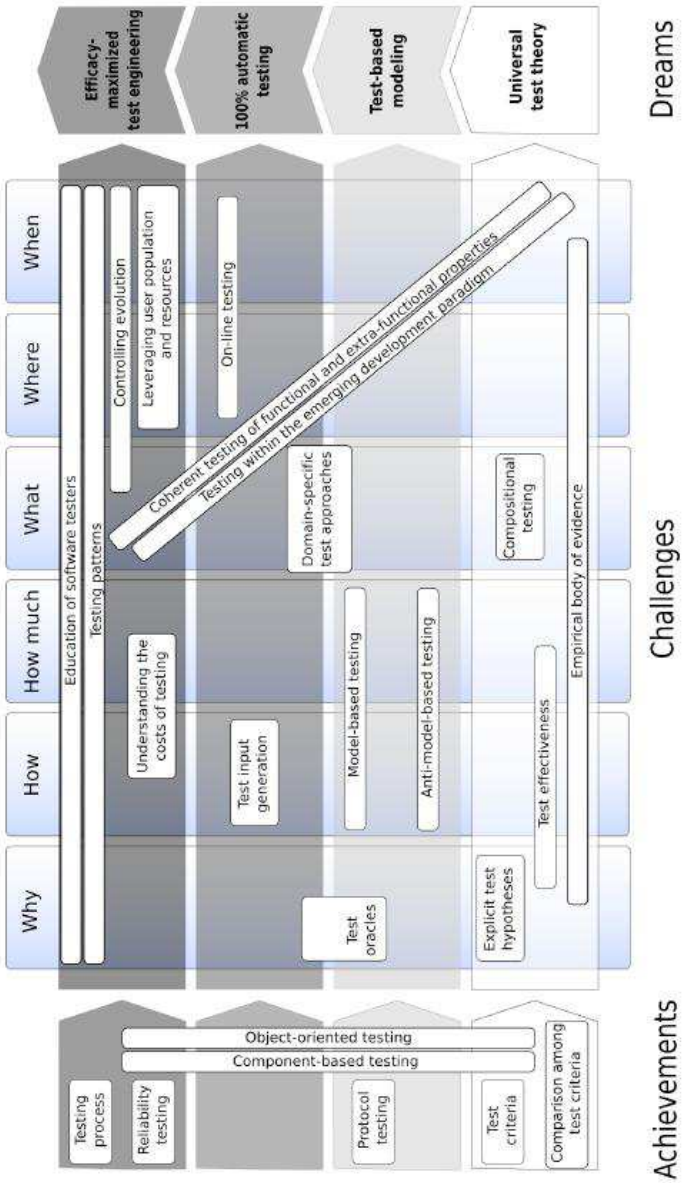


Figura. 1. Roadmap de las pruebas de software [2]

Los desafíos de la investigación en pruebas de software encuentran su lugar en este plan, verticalmente dependiendo si es un largo sueño, y hacia la que tienden principalmente, y horizontalmente de acuerdo a las cuestiones introducidas.

En base al *roadmap* detallado por Bertolino, y según la naturaleza de nuestra propuesta, consideramos oportuno centrarnos en alcanzar dos de los "sueños" mencionados: modelado basado en pruebas y las pruebas 100% automáticas.

Los siguientes apartados abordan estos conceptos, pero relacionados sólo con el sueño de las pruebas 100% automáticas, y para ello el tratamiento del desafío de los oráculos de pruebas automáticos.

3 Pruebas 100% automáticas

La automatización es una de las formas de mantener la calidad del análisis y de las pruebas, en línea con la actual complejidad del software. La investigación en ingeniería del software pone gran énfasis en la automatización de la producción del software, con una mayor parte en las herramientas de desarrollo, generando cada vez más grandes y complejas cantidades de código con menos esfuerzo.

La otra cara de la moneda es el gran peligro que tienen los métodos para evaluar calidad del software producido, en particular, los métodos de prueba que no pueden mantener el ritmo de los métodos de construcción de software.

Una gran parte de la investigación actual sobre las pruebas tiene por objeto mejorar el grado de automatización, ya sea mediante el desarrollo de técnicas avanzadas para la generación de entradas de pruebas, o para encontrar procedimientos de soporte para la automatización del proceso de prueba.

El “sueño”' vendría a ser un potente ambiente integrado de pruebas, que por sí solo, pueda, automáticamente, hacerse cargo de la generación y recuperación del código necesario (drivers, stubs, simuladores), generando los casos de prueba más adecuados, ejecutándolos y finalmente expidiendo un informe de la prueba.

Esta idea ha atraído muchos seguidores, por ejemplo, el método de pruebas continuas de Saff [13], precisamente intentan ejecutar pruebas en *background* sobre la máquina de los desarrolladores mientras ellos programan.

Se han realizado muchos pasos prometedores para las pruebas unitarias, que es ampliamente reconocida como la fase esencial que asegura la calidad del software, porque examinar unidades individuales de forma aislada puede permitir detectar tempranamente aquellos fallos sutiles que difícilmente se encuentran en el nivel de las pruebas de sistema.

Desafortunadamente, las pruebas unitarias son muchas veces mal realizadas o dejadas de lado por completo, al considerarse una actividad cara. Es así, que se necesita enfoques para hacerlo más factible dentro de los procesos de desarrollo de la industria del software.

Uno de los principales componentes que intervienen en el alto costo de las pruebas unitarias es la enorme cantidad de codificación extra, necesaria para simular el ambiente donde la unidad debe ejecutarse y en donde se ejecuta el chequeo funcional necesario para las salidas de la unidad. Para aliviar tales tareas, los *frameworks* de la familia XUnit tienen un gran éxito entre los desarrolladores. Entre ellas la más satisfactoria es JUnit [10], la cual permite la automatización del código de los casos de prueba Java.

Otro ejemplo es el proporcionado por la noción de "agitación de software" [3], una técnica de pruebas unitarias automáticas soportadas por la herramienta comercial Agitator, la cual combina diferentes análisis, tales como la ejecución simbólica, resolución de restricciones, y la generación aleatoria de entradas para la generación de los datos de entrada.

También existe otro método de pruebas unitarias, las parametrizables (PUT) [14]; es decir, las pruebas unitarias codificadas que no son fijas, sino que dependen de algunos parámetros de entrada. PUT puede describir el comportamiento abstracto en una forma concisa. Usando técnicas de ejecución simbólica y resolviendo restricciones, puede encontrar entradas para los PUTs para alcanzar un alto código de cobertura.

Los tres ejemplos citados no son ciertamente exhaustivos. La tendencia común que surge es el esfuerzo por combinar de manera eficiente los diversos tipos de análisis, y esto, junto con el aumento exponencial de los recursos computacionales disponibles, podría ser realmente la dirección hacia el sueño de la automatización 100% de las pruebas.

4 Oráculos de prueba

La cuestión referente a decidir si el resultado de una prueba es aceptable o no, está relacionado estrictamente con la planificación de las pruebas, y específicamente al problema de cómo derivar los casos de prueba. Esto corresponde a lo que es denominado "oráculo", idealmente es un método que provee las salidas esperadas de cada caso de prueba dado; de manera más realista, es una heurística que puede emitir un veredicto de pasa/fallo sobre las salidas de prueba observadas.

Aunque es evidente que una prueba de ejecución para la cual no somos capaces de discriminar entre el éxito y el fracaso, es una prueba inútil, y aunque la importancia de este problema se ha planteado muy temprano en la literatura [16], al problema del oráculo le ha prestado poca atención en la investigación y en la práctica existen pocas soluciones alternativas.

Con el incremento de la complejidad y criticidad de las aplicaciones de software, está destinado a convertirse en un obstáculo que bloquee la fiabilidad de la automatización de las pruebas.

Es más, la precisión y la eficiencia de los oráculos afectan al coste y a la eficacia de las pruebas. No se desea que las pruebas fallidas pasen desapercibidas, pero por otro lado, no queremos notificar muchos falsos positivos, los cuales echan a perder los recursos.
Se necesita encontrar métodos eficientes para la realización y automatización de las pruebas. Baresi y Young [1] proporcionan un estudio crítico de las soluciones de oráculos, concluyendo en las siguientes características:

Comportamiento y estado abstracto vs. concreto: las pruebas basadas en modelos prometen aliviar el problema de los oráculos, ya que el mismo modelo puede actuar como oráculo; sin embargo, para los oráculos basados en las descripciones abstractas del comportamiento del código, el problema sigue siendo el salvar las distancias entre las entidades concretas observadas y las entidades abstractas especificadas.

Parcialidad: convincentemente los oráculos parciales son la única solución viable para la automatización del oráculo. El reto es encontrar la mejor compensación entre precisión y costo.

Cuantificación: Para los oráculos de prueba implementados por lenguajes de especificación ejecutables, se trata de encontrar un compromiso entre la expresividad y la eficiencia. Hasta el momento no hay un claro equilibrio óptimo, ni algún método completamente satisfactorio para adaptar los cuantificadores.

Selección de casos de prueba y oráculos: Idealmente, los oráculos deben ser ortogonales a la selección de los casos de prueba; sin embargo, en las pruebas basadas por modelos, los modelos disponibles son muchas veces usados para derivar clases de pruebas y oráculos de prueba de clases específicas.

5 Planteamiento heurístico

Convencionalmente, el trabajo de las pruebas es usualmente realizado por personal experimentado con pruebas manuales. Es un duro trabajo que comúnmente cuesta más del 50% de la inversión de un proyecto [11] [12]. Hacer que este trabajo se realice sin la interferencia de personas es uno de los desafíos hacia el sueño de las pruebas automáticas. Muchas actividades de pruebas pueden ser automatizadas, generación de casos de prueba, verificación, etc. Sin embargo, los oráculos de prueba son un mecanismo o proceso utilizado para generar los resultados esperados del software bajo prueba y su automatización es un desafío actual. Para direccionar las pruebas automáticas, el método de generación automática de oráculos de prueba es el primero que tiene que tomarse en cuenta.

Ciertamente, si el personal de pruebas juzga la corrección de un programa, ellos deben saber distinguir los resultados que deberían salir. Los oráculos de prueba es ahora una rama importante para las pruebas de software automáticas, pero aún falta un entendimiento completo sobre el tema. Muchas de las investigaciones actuales acerca de la generación automática de oráculos de prueba están principalmente relacionadas con la utilización de métodos formales de verificación. Las especificaciones formales son usadas para la generación de oráculos de prueba [7] [17]. También hay trabajos relacionados con métodos dirigidos por modelos [5] [6] [4] y algoritmos heurísticos [9] [18].

Todos los métodos de prueba de software dependen de la disponibilidad de un oráculo, es decir, de algún método que verifique la exactitud del sistema bajo prueba. Un oráculo ideal proporciona un juicio infalible para la ejecución del programa. Existen varios métodos para oráculos definidos para verificar las pruebas a través de la industria del software. La captura y comparación de los resultados es fundamental para el éxito de las pruebas de software [1].

Las pruebas automáticas requieren de un trabajo más complicado con relación a las pruebas manuales. Los probadores humanos tienen la ventaja de poder decidir de acuerdo a su experiencia personal. No obstante, un probador humano al evaluar el comportamiento de un programa tiene como desventaja la exactitud limitada o el costo que ello significa. Para que sea posible una prueba automática, es condición necesaria que el oráculo de pruebas que proporciona el veredicto de pasa/fallo del caso de prueba sea también automático. La Figura 2 muestra el esquema del planteamiento propuesto, en donde una red neuronal se encargue de ser el verificador. Tanto la topología como la definición de las entradas del oráculo han de ser estudiadas, así como el tipo de entrenamiento que deba tener.

.

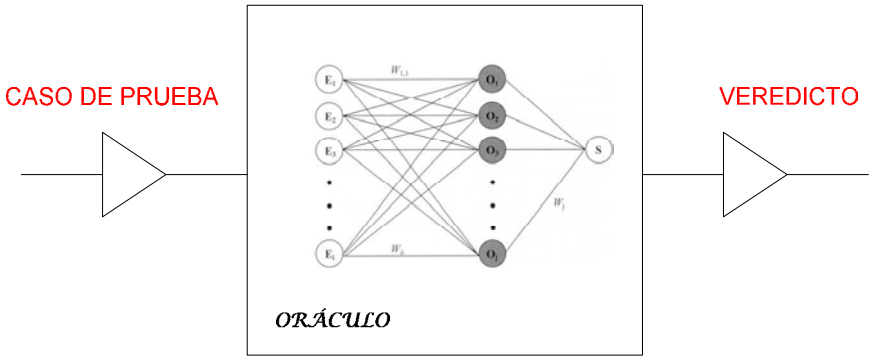


Figura. 2. Planteamiento de oráculo de prueba heurístico [2]

6 Conclusiones y trabajos futuros

A partir del estudio comparativo realizado por nuestro grupo de investigación [15], con el objetivo del estudio de las pruebas automáticas en un contexto MDWE (Model-Driven Web Engineering), en dicho estudio se identificaron diversos desafíos que deben ser abordados: automatización del proceso, enfoque MDA (Model-Driven Architecture), reglas de transformación, proceso basados en navegación y finalmente los oráculos de pruebas eficientes. Este trabajo es un planteamiento inicial sobre el cómo abordar los oráculos de pruebas con métodos heurísticos. A pesar de ser un planteamiento superficial, nuestra intención es obtener el *feedback* y estudiar la viabilidad de la utilización de técnicas de decisión heurísticas para la obtención de oráculos de prueba automáticos.

Agradecimientos. Esta trabajo ha sido apoyado por el proyecto QSimTest (TIN2007-67843-C06 03) y el proyecto RePRIS del Ministerio de Educación y Ciencia (TIN2007-30391-E), España.

Referencias

1. Baresi, L., Youngh, M.: Test oracles. Technical report, Dept. of Comp. and Information Science, Univ. of Oregon (2001).
2. Bertolino, A.: Software testing research: Achievements, challenges, dreams. In FOSE ’07: Future of Software Engineering, pages 85–103, Washington, DC, USA, 2007. IEEE Computer Society (2007).
3. Boshernitsan, M., Doong, R., Savoia, A.: From daikon to agitator: lessons and challenges in building a commercial tool for developer testing. In ISSTA ’06: Proceedings of the 2006 international symposium on Software testing and analysis, pages 169–180, New York, NY, USA. ACM Press (2006).
4. Callahan, J. R., Easterbrook, S. M.: Generating Test Oracles via Model Checking, Technique Report, NASA/WVU Software Research Lab (1998).
5. Clarke, E. M., Grumber, O., Long, D.: Model Checking and Abstraction. In Proceedings of the Nineteenth Annual ACM Symposium on Applications, North-Holland (1993).
6. Clarke, E. M., Grumberg O., Long, D.: Verification tools for finite-state concurrent system. Springer Berlin, Lecture Notes in Computer Science, Volume 803 (1994).
7. Dillon, L. K., Ramakrishna, Y. S.: Generating Oracles from your favorite Temporal Logic Specifications. Proc of the 4th ACM SIGSOFT Symp on the Foundations of Software Engineering, pp 106-117 (1996).
8. Heuser, L.: The real world or web engineering? In Proceedings of the 4th International Conference on Web Engineering. Volume 3140, page 15, Berlin, Springer (2004).
9. Hoffman, D.: Heuristic test oracles. Software Testing and Quality Engineering, Vol. 12, (1999).
10. JUnit.org. http://www.junit.org/index.htm
11. Jungmayr, S.: “Reviewing Software Artifacts for Testability”, Barcelona, Spain, Proceedings of the EuroSTAR’99, (1999).
12. Pressman, R.: Software Engineering: A Practitioner’s Approach, (2004).

13. Saff, D., D. Ernst M.: An experimental evaluation of continuous testing during development. In ISSTA '04: Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis, pages 76–85, New York, NY, USA. ACM (2004).
14. Tillmann, N., Schulte, W.: Unit tests reloaded: parameterized unit testing with symbolic execution. Software, IEEE, 23(4):38–47, (2006).
15. Torres, A. H., Escalona, M. J., Mejías, M, Gutiérrez, J.: A MDA-Based Testing:A comparative study. 4th International Conference on Software and Data Technologies, ICSOFT, Bulgary (2009).
16. Weyuker, E.J.: On testing non-testable programs. The Computer Journal, 25(4):465–470 (1982)
17. Xin, W., Ji, W., Zhi-chang, Q.: An Overiview: Temporal Specification-Based Technologies of Automatic Generation Test Oracle, Vol.28, No. 7, pp127-130 (2006).
18. Zhang, D.: Applying machine learning algorithms in software development. The Proceedings of 2000 Monterey, CA (2000).