Available online at www.sciencedirect.com

ScienceDirect

Applied Mathematics and Computation 183 (2006) 491–498

APPLIED MATHEMATICS AND COMPUTATION

www.elsevier.com/locate/amc

Distributed CSPs by graph partitioning

Miguel A. Salido *, Federico Barber

Department of Information Systems and Computation, Technical University of Valencia, Spain

Abstract

Nowadays, many real problems in artificial intelligence can be modelled as constraint satisfaction problems (CSPs). A general CSP is known to be NP-complete. Nevertheless, distributed models may reduce the exponential complexity by partitioning the problem into a set of subproblems. In this paper, we present a preprocess technique to break a single large problem into a set of smaller loosely connected ones. These semi-independent CSPs can be efficiently solved and, furthermore, they can be solved concurrently.
© 2006 Elsevier Inc. All rights reserved.

Keywords: Constraint satisfaction problems; Distributed CSPs; Artificial intelligence

1. Introduction

Many real problems in artificial intelligence (AI) as well as in other areas of computer science and engineering can be efficiently modelled as constraint satisfaction problems (CSPs) and solved using constraint programming techniques. Some examples of such problems include: spatial and temporal planning, qualitative and symbolic reasoning, diagnosis, decision support, scheduling, hardware design and verification, real-time systems and robot planning.

These problems may be soluble or insoluble or they may be hard or easy. How to solve these problems have been the subject of intensive study in recent years.

Most of the work is focused on general methods for solving CSPs. They include backtracking-based search algorithms. While the worst-case complexity of backtracking search is exponential, several heuristics to reduce its average-case complexity have been proposed in [1]. For instance, some algorithms incorporate features such as ordering heuristics: variable ordering [2], value ordering [3], and constraint ordering [4,5].

Agent-based computation has been studied for several years in the field of artificial intelligence and has been widely used in other branches of computer science. Multi-agent systems are computational systems in which several agents interact or work together in order to achieve goals. In the specialized literature, there are many works about distributed CSP. In [6], Yokoo et al. present a formalization and algorithms for solving distributed CSPs.

* Corresponding author.
E-mail address: msalido@dsic.upv.es (M.A. Salido).

0096-3003/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.amc.2006.05.090

Many researchers are also working on graph partitioning [7, 8, 9]. The main objective of graph partitioning is to divide the graph into a set of regions such that each region has roughly the same number of nodes and the sum of all edges connecting different regions is minimized. Graph partitioning can be applied on telephone network design, sparse Gaussian elimination, data mining, clustering, and physical mapping of DNA. Fortunately, many heuristics can solve this problem efficiently. For instance, graphs with over 14,000 nodes and 410,000 edges can be partitioned in under 2 s [10]. Thus, graph partitioning can be useful to distribute the problem into a set of semi-independent sub-CSPs when dealing with large CSPs. In the constraint satisfaction literature, some works integrate partitioning techniques with constraint satisfaction problems mainly in parallel frameworks. They are primarily focused on partitioning the search space. The CSP is split according to the domains of the variables, and each processor solves a part of the complete search space.

In Burg [11], the search space is partitioned into the same number of subspaces as processors are available. In Lin and Yang [12], the search space is divided into d partitions, where d is the domain size of the first variable of the CSP (only one variable is considered for the partitioning).

Our aim is to partition the CSP by means of the variables using a preprocessing technique. To this end, the CSP is divided into a set of smaller loosely connected ones. This division is carried out by a graph partitioning software called Metis [13]. Thus, we can take advantage of the powerful graph partitioning techniques that are available for solving CSPs, mainly in large problems where a centralized solver is unable to manage the problem. This approach of partition is also useful in some application problems, where gathering all information together is not desirable or impossible for security or privacy reasons.

In the following section, we formally define a CSP and summarize some other definitions. In Section 3, we present the graph partitioning heuristic. The general system architecture is presented in Section 4. We evaluate our proposal in Section 5 and, finally, we summarize the conclusions and future work.

2. Definitions

In this section, we review some basic definitions about CSPs.

Definition 1. A constraint satisfaction problem (CSP) consists of:

- a set of variables $V = \{v_1, v_2, \ldots, v_n\}$
- each variable $v_i \in V$ has a set D_{v_i} of possible values (its domain). We denote d_{v_i} the length of domain D_{v_i}
- a finite collection of constraints $C = \{c_1, c_2, \ldots, c_k\}$ that restricts the values that the variables can simultaneously take

Definition 2. A solution to a CSP is an assignment of values to all the variables so that all constraints are satisfied.

Definition 3. A partition of a set C is a set of disjoint subsets of C whose union is C. The subsets are called the blocks of the partition.

Definition 4. A distributed CSP is a CSP in which the variables and constraints are distributed among automated agents [6].

Each agent has some variables and attempts to determine their values. However, there are inter-agent constraints and the value assignment must satisfy these inter-agent constraints. In our model, there are m agents $1, 2, \ldots, m$. Each agent knows a set of constraints and the domains of variables involved in these constraints.

Definition 5. A *block agent* a_j is a virtual entity that essentially has the following properties: autonomy, social ability, reactivity, and pro-activity.

Block agents are autonomous agents. They operate their subproblems without the direct intervention of any other agent or human. *Block agents* interact with each other by sending messages to communicate consistent partial states. They perceive their environment and any changes in it (such as new partial consistent states) and can react with more complete consistent partial states.

3. Graph partitioning heuristic

As we have pointed out in the introduction, many researchers are working on graph partitioning [7, 8, 9] with the aim of dividing the graph into a set of regions such that each region has roughly the same number of nodes and minimizing the sum of all edges connecting different regions. This technique has been widely used in many areas and can be very useful in constraint satisfaction problem, mainly for large CSPs.

Metis [13] is a software package for partitioning large irregular graphs, partitioning large meshes, and computing fill-reducing orderings of sparse matrices. Traditional graph partitioning algorithms compute a partition of a graph by directly operating on the original graph. These algorithms are often too slow and/or produce poor quality partitions. Metis uses novel approaches to successively reduce the size of the graph as well as to further refine the partition during the uncoarsening phase. During uncoarsening, Metis employs algorithms that make it easier to find a high-quality partition in the coarsest graph. During refinement, Metis focuses primarily on the portion of the graph that is close to the partition boundary. These highly tuned algorithms allow Metis to quickly produce high-quality partitions for a large variety of graphs. Metis provides two programs *pmetis* and *kmetis* for partitioning an unstructured graph into k equal size parts. Both of these programs are able to produce high-quality partitions. However, depending on the application, one program may be preferable to the other. In general, *kmetis* is preferred when it is necessary to partition graphs into more than eight partitions. For such cases, *kmetis* is considerably faster than *pmetis*. On the other hand, *pmetis* is preferable for partitioning a graph into a small number of partitions.

Our aim is to divide a CSP into a set of sub-CSPs so that each subproblem has roughly the same number of variables and to minimize the sum of all binary constraints connecting different subproblems. Fig. 1 shows an example of CSP that can be partitioned into two different sub-CSPs. A graph partitioning tool would divide the CSP that contains variables $\{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9\}$ in two Sub-CSPs: $(1) \rightarrow \{v_1, v_2, v_3, v_4, v_5\}$ and $(2) \rightarrow \{(v_5), v_6, v_7, v_8, v_9\}$.

In this way, the sub-CSP1 is composed of a set of variables $V1 = \{v_1, v_2, v_3, v_4, v_5\}$. The set of constraints is composed of the binary constraints that join two variables in $V1$. Sub-CSP2 is composed of a set of variables $V2 = \{(v_5), v_6, v_7, v_8, v_9\}$, where v_5 is an instantiated variable. However, this variable must be included to satisfy the constraint that joins the two sub-CSPs.

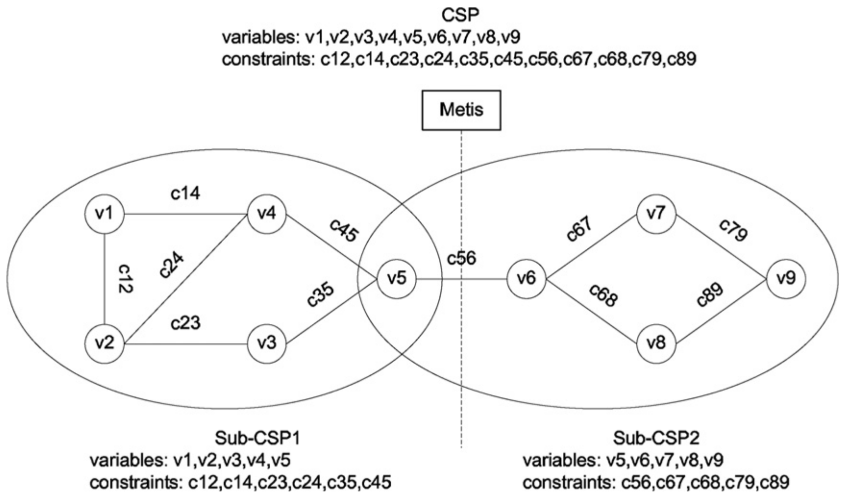
CSP
variables: v1,v2,v3,v4,v5,v6,v7,v8,v9
constraints: c12,c14,c23,c24,c35,c45,c56,c67,c68,c79,c89
Metis
v1
v2
v3
v4
v5
v6
v7
v8
v9
c14
c12
c24
c23
c45
c35
c56
c67
c68
c79
c89
Sub-CSP1
variables: v1,v2,v3,v4,v5
constraints: c12,c14,c23,c24,c35,c45
Sub-CSP2
variables: v5,v6,v7,v8,v9
constraints: c56,c67,c68,c79,c89

Fig. 1. A CSP divided into two semi-independent Sub-CSPs.

3.1. Partition Size

The goal of Metis is to partition the original CSP, but the question is to determine how many partitions must be generated to optimize the solving process. This number must be given by the user. It depends on several parameters such as the number of variables, the number of constraints, and the domain size. However, it can also depend on the constrainedness of the problem [14]. If the CSP is easy to solve (under-constrained), no partition is necessary to achieve a solution; if the problem is hard to solve, the number of partitions must be greater. Nevertheless, a huge number of partitions is not desirable, due to the fact that time must be spent in the exchange of information.

As we will analyze in evaluation, the number of partitions will depend mainly on the number of variables n, the domain size d, and also, but less important, on the number of constraints. An appropriate threshold is necessary to achieve optimality in the solving process. A good approach for achieving the number of partitions is:

$$\#\text{partitions} = \lfloor \ln(n) + \ln(d) + \log(gd) \rfloor, \tag{1}$$

where n is the number of variables, d is the domain size and gd is the graph degree. For instance, a problem with 200 variables, a domain size of 40 and a graph degree of 25, the number of partitions is $\lfloor \ln(200) + \ln(40) + \log(25) \rfloor = 10$. This formula gives us an estimator of an appropriate number of partitions (see Table 3 in the evaluation section).

3.2. The block agents

Block agents are agents whose aim is to solve subproblems. Following, we present the behavior and characteristics of *block agents* (Fig. 2):

- Each *block agent* a_j has an identifier j.
- There is a partition of the set of constraints $C \equiv \bigcup_{i=1}^{m} C(i)$ generated by the graph partitioning heuristic, and each *block agent* a_j is committed to the block of constraints $C(j)$.
- Each *block agent* a_j has a set of variables V_j involved in its block of constraints $C(j)$. These variables fall into two different sets: *used variables* set ($\bar{v}_j$) and *new variables* set (v_j), that is: $V_j = \bar{v}_j \cup v_j$. *Used variables* correspond to previously assigned variables.
- The domain D_i (corresponding to variable x_i) is maintained in the first *block agent* a_t in which x_i is involved, (i.e.), $x_i \in v_t$.
- Each *block agent* a_j assigns values to variables that have not yet been assigned, that is, a_j assigns values to new variables $x_i \in v_j$ because variables $x_k \in \bar{v}_j$ have already been assigned by previous *block agents*.
- Each *block agent* a_j maintains a storage of partial problem solutions generated by its neighbors and previous *block agents*. Therefore, the last *block agent* can return a solution for the global problem.

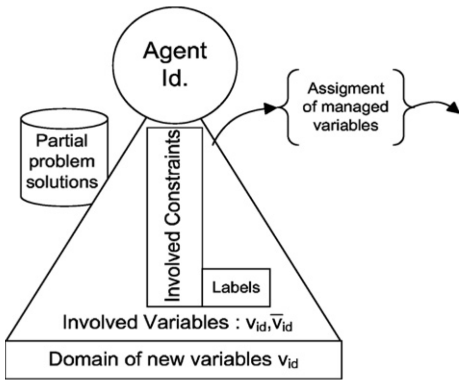
Agent Id.
Assigment of managed variables
Partial problem solutions
Involved Constraints
Labels
Involved Variables : $v_{id}, \bar{v}_{id}$
Domain of new variables v_{id}

Fig. 2. Properties and characteristics of block agents.

The goal of these *block agents* is to solve CSPs that represent subproblems of the main CSP. These *block agents* must cooperate with each other by sending messages with consistent partial states.

4. General system architecture

The general scheme of our system is presented in Fig. 3. As a preprocessing step, the Graph Partitioning Heuristic is carried out to partition the problem into a set of semi-independent subproblems. To this end, given the number of variables n, the domain size d, and the number of constraints c, we can use formula (1) to obtain the estimated number of partitions (p). Thus, given the CSP and the number of partitions, Metis selects the appropriate program (*kmetis*, *pmetis*) to partition the problem. In this way, the original problem is divided into different subproblems. These subproblems can be concurrently solved using different techniques.

Once, each subproblem is generated, *block agents* concurrently manage each block of constraints (sub-CSP). Each *block agent* is free to select any search algorithm to find a consistent partial state. It can select a local search algorithm, a backtracking-based algorithm, or any other algorithm, depending on the problem topology. In any case, each *block agent* must find a solution to its particular subproblem. This subproblem is composed of its CSP, which is subject to the variable assignment generated by the previous *block agents*.

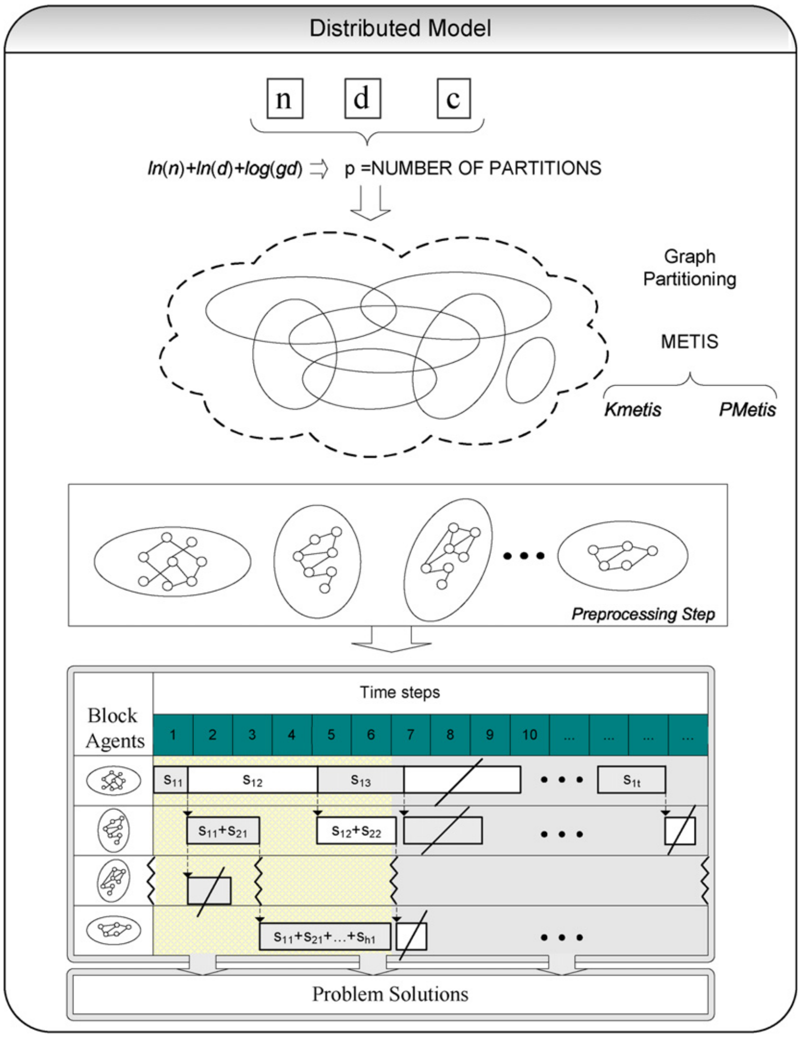

Fig. 3. General system architecture.

Table 1
Random instances $\langle n, 40, 25, 10\rangle$: variables = n, domain size = 40, graph density = 25, and partition size = 10

Problem	Distributed model run-time (s)	Centralized model run-time (s)
$\langle 50, 40, 25, 10\rangle$	12	3
$\langle 100, 40, 25, 10\rangle$	12	14
$\langle 150, 40, 25, 10\rangle$	13	37
$\langle 200, 40, 25, 10\rangle$	14	75
$\langle 250, 40, 25, 10\rangle$	17	98
$\langle 300, 40, 25, 10\rangle$	19	140
$\langle 350, 40, 25, 10\rangle$	23	217
$\langle 400, 40, 25, 10\rangle$	30	327
$\langle 450, 40, 25, 10\rangle$	32	440
$\langle 500, 40, 25, 10\rangle$	42	532

Generally, Sub-CSPs are classified by their degree of connectivity; that is, Sub-CSP1 is connected with other sub-CSPs besides Sub-CSP2 and, so on. Thus, *block agent* 1 works on its block of constraints. If *block agent* 1 finds a solution to its subproblem, then it sends the consistent partial state to the neighboring *block agents* that are connected to *block agent* 1. All of them work concurrently to solve their specific subproblems: *block agent* 1 tries to find another solution and the neighboring *block agents* try to solve their subproblems knowing that their *used variables* have been assigned by *block agent* 1. Thus, in any time step (see Fig. 3), any *block agent* j, that has the instantiated variables generated by the previous *block agents* works concurrently with the previous *block agents*, and it tries to find a consistent state using a search algorithm. Finally, the last *block agent* m, which works concurrently with *block agents* $1, 2, \ldots, (m-1)$, tries to find a consistent state in order to find a problem solution.

5. Evaluations

In this section, we evaluate and compare our distributed model and the centralized model. To do this, we have used a well-known CSP solver called forward checking (FC).[1]

In our evaluations, each set of random binary CSPs was defined by the 4-tuple $\langle n, d, gd, p\rangle$, where n was the number of variables, d the domain size, gd the graph density (constraints) and p the number of partitions. The problems were randomly generated by modifying these parameters. We evaluated 100 test cases for each type of problem and each value of the variable parameter. All random binary constraints are in the form $c{:}v_i\langle\rangle v_j{:}v_i, v_j \in V$. We compare the run-time (in seconds) of the distributed model against the centralized model. In the distributed model, the run-time is composed by the run-time of the preprocessing step plus the run-time of the solving process of each sub-problem. There are three different evaluations presented in the tables.

In the first evaluation, the domain size, the graph density, and the size of the partition were fixed, and the number of variables was increased from 50 to 500. As Table 1 shows, the run-time for small problems was worse in the distributed model due to the fact that, in the preprocessing step, the number of partitions was greater than the number estimated by formula (1). For example, the random instances $\langle 50, 40, 25, 10\rangle$ were solved by the distributed model in 12 s, while the same problems with 5 partitions ($\langle 50, 40, 25, 5\rangle$) were solved in 7 s (see Table 2). However, as the number of variables increased, the behavior of the distributed problem improved. For instance, in random instances $\langle 500, 40, 25, 10\rangle$, the centralized model maintained 500 variables, whereas each subproblem in the distributed model maintained an average of 50 variables. Thus, our system is appropriate for large CSPs where a centralized model is unable to manage these type of problems.

In the second evaluation, the number of variables and the graph density were fixed, and the domain size was increased from 20 to 300. As Table 2 shows, our proposal maintained lower run-times than the centralized model. For small domains the centralized model was better, but in more realistic problems (with large domains), the distributed model was better. We also increased the domain size to 400 and 500, and we

[1] Forward Checking was obtained from CON'FLEX. It can be found in: http://www.inra.fr/bia/T/rellier/Logiciels/conflex/welcome.html.

Table 2
Random instances $\langle 50, d, 25, p\rangle$: variables = 50, domain size = d, graph density = 25, and partition size = p

Problem	Distributed model run-time (s)	Centralized model run-time (s)
$\langle 50,20,25,5\rangle$	7	2
$\langle 50,30,25,5\rangle$	7	3
$\langle 50,40,25,5\rangle$	7	4
$\langle 50,50,25,5\rangle$	7	5
$\langle 50,100,25,5\rangle$	10	22
$\langle 50,150,25,5\rangle$	14	49
$\langle 50,200,25,5\rangle$	29	123
$\langle 50,300,25,5\rangle$	35	287
$\langle 50,400,25,11\rangle$	34	515
$\langle 50,500,25,11\rangle$	75	933

Table 3
Random instances $\langle 200, 40, 25, p\rangle$: variables = 200, domain size = 40, graph density = 25, and partition size = p

Problem	Distributed model run-time (s)	Centralized model run-time (s)
$\langle 200,40,25,2\rangle$	51	75
$\langle 200,40,25,3\rangle$	26	75
$\langle 200,40,25,4\rangle$	20	75
$\langle 200,40,25,5\rangle$	19	75
$\langle 200,40,25,6\rangle$	16	75
$\langle 200,40,25,8\rangle$	15	75
$\langle 200,40,25,10\rangle$	14	75
$\langle 200,40,25,12\rangle$	16	75
$\langle 200,40,25,15\rangle$	18	75
$\langle 200,40,25,20\rangle$	22	75

increased the number of partitions to 11. This table illustrates the usefulness of an appropriate number of partitions. For random instances $\langle 50, 400, 25, p\rangle$, if formula (1) is used, then $p = \lfloor \ln(50) + \ln(400) + \log(25)\rfloor = 11$ and better results were obtained than random instances $\langle 50, 300, 25, 5\rangle$.

In the last evaluation, the number of variables, the domain size, and the graph density were fixed, and the size of the partition was increased from 2 to 20. This evaluation shows the importance of selecting an appropriate number of partitions. Table 3 shows an empirical evaluation of the appropriate number of partitions for binary problems with 200 variables, a domain size of 40 and a graph density of 25. The most appropriate number of partitions was 10, due to the fact that the run-time was the lowest (14 s). This number of partitions coincides with the number of partitions obtained using formula (1). Furthermore, in all the random instances, the distributed model obtained a lower run-time than the centralized model. However, it is still useful to obtain a formula that estimates an appropriate number of partitions.

6. Conclusions and future work

Nowadays, many real problems can be modelled as constraint satisfaction problems (CSPs) and solved using constraint programming techniques. These problems may be soluble or insoluble and they may be hard or easy to solve. In this paper, we have presented a preprocessing technique to break a single, large problem into a set of smaller loosely connected ones, where a set of agents are committed to solving their own subproblems. Our distributed model takes advantage of advanced on graph partitioning techniques to distribute a CSP in semi-independent sub-CSPs. Our evaluations show that our distributed model significantly improves runtimes for large CSPs.

We are currently applying our distributed model to the railway scheduling problem. This problem is very hard to solve in a centralized way, but it can be easily distributed by means of trains and track stations.

Acknowledgement

This work has been partially supported by the research projects TIN2004-06354-C02- 01 (Min. de Educacin y Ciencia, Spain-FEDER), FOM-70022/T05 (Min. de Fomento, Spain) and FP6-021235-2, IST-STREP (UE).

References

[1] R. Dechter, J. Pearl, Network-based heuristics for constraint satisfaction problems, Artificial Intelligence 34 (1988) 1–38.
[2] E. Freuder, A sufficient condition for backtrack-free search, Journal of the ACM 29 (1982) 24–32.
[3] D. Frost, R. Dechter, Look-ahead value orderings for constraint satisfaction problems, Proceedings of IJCAI-95 (1995) 572–578.
[4] R. Wallace, E. Freuder, Ordering heuristics for arc consistency algorithms, Proceedings of Ninth Canadian Conferences on A.I. (1992) 163–169.
[5] M.A. Salido, F. Barber, A constraint ordering heuristic for scheduling problems, Proceeding of the First Multidisciplinary International Conference on Scheduling : Theory and Applications 2 (2003) 476–490.
[6] M. Yokoo, E.H. Durfee, T. Ishida, K. Kuwabara, The distributed constraint satisfaction problem: formalization and algorithms, IEEE Transactions on Knowledge and Data Engineering 10 (1998) 673–685.
[7] B. Hendrickson, R.W. Leland, A multi-level algorithm for partitioning graphs, Supercomputing (1995).
[8] B.W. Kernighan, S. Lim, An efficient heuristic procedure for partitioning graphs, Technical Report 1 (1970).
[9] G. Karypis, V. Kumar, A parallel algorithm for multilevel graph partitioning and sparse matrix ordering, Journal of Parallel and Distributed Computing 48 (1998) 71–95.
[10] G. Karypis, V. Kumar, Using METIS and parMETIS, Technical report (1995).
[11] B. Burg, Parallel forward checking: First part, Technical Report TR-594, Institute for New Generation Computer Technology (1990).
[12] W. Lin, B. Yang, Probabilistic performance analysis for parallel search techniques, International Journal of Parallel Program 23 (2) (1995) 161–189.
[13] G. Karypis,V. Kumar, Multilevel algorithms for multi-constraint graph partitioning, Technical Report TR 98-019, Department of Computer Science, University of Minnesota, 1998.
[14] M.A. Salido, F. Barber, Exploiting the constrainedness in constraint satisfaction problems, Artificial Intelligence: Methodology, Systems, and Applications LNAI 3192 (2004) 126–136.