Hybridization of CP and VLNS for Eternity II.

Pierre Schaus **Yves Deville**

Department of Computing Science and Engineering,
University of Louvain,
Place Sainte Barbe 2,
B-1348 Louvain-la-Neuve, Belgique

{pierre.schaus,yves.deville}@uclouvain.be

Abstract

Eternity II is an edge-matching puzzle created by Christopher Monckton for the game editor Tomy(TM). Given 256 squared pieces with a color on each of the four sides of pieces and a 16×16 board, the goal is to place all the pieces such that two adjacent pieces have their common side of same color. This problem is NP-complete, has very few structure and is highly combinatorial ($256! \cdot 4^{256}$ possible combinations). Christopher Monkton is so confident in the difficulty of the problem that he promises a \$2 million prize to the first person who finds the solution. We don't have any hope (anymore) to find a solution to this problem, but we nevertheless decided to explain our strategy which might be useful for someone else still believing in his/her chances of success. Our procedure first initializes the board with constraint programming by relaxing the problem. Then we improve the solution with a very large neighborhood stochastic local search. Our neighborhood is very large (i.e. exponential) but can be explored in polynomial time by solving an assignment problem. Our procedure allows us to obtain rapidly good solutions with scores reaching 458/480 number of satisfied junctions. Our very large neighborhood can also be used in a brute-force approach to test $256!/128! \cdot 4^{128}$ combinations instead of $256! \cdot 4^{256}$.

This paper is also an example of VLNS that can be applied on other matching problems.

1 Introduction

The Eternity II (E2) puzzle consists of n^2 square pieces that are bordered by one color on each side and, a $n \times n$ board game. The pieces must be placed on a board such that two adjacent pieces have aligned colors. A same color on the extremity of the board is imposed.

E2 is a 16×16 instance. Since the extremity color is imposed, these constraints are not difficult to satisfy. This is why the score is always given in terms of the number of inside connections, that is $2n \cdot (n-1)$ for a $n \times n$ puzzle (480 inside connections for the 16×16).

Many people are actively working on E2. The most significant group of discussion on the web counts about 2000 members [1]. Two distributed computing softwares were developed using a brute-force backtracking approach :

– The first one was developed by Dave Clark. About 1000 registered computers were permanently active. This project was the most popular one and to the best of our knowledge have submitted the best known solution : 463/480. This project lasted 6 months. Dave Clark has now give up because he doesn't believe any more in the brute-force approach to solve the puzzle.
– The second one is a French project [2] which seems to have less members. They did not published their highest score.

E2 is an edge-matching puzzle, that is a tiling puzzle involving tiling an area with (typically regular) polygons whose edges are distinguished with colours or patterns, in such a way that the edges of adjacent tiles match. These categories of puzzles were proved to be NP-complete [4]. Tetravex [3] and E2 [4] are both edge-matching puzzles. In particular Tetravex was proved to be NP-complete by reduction of 1in3-SAT [10].

[1] http://games.groups.yahoo.com/group/eternity_two/
[2] http://www.eternity2.fr/
[3] http://live.gnome.org/Tetravex/
[4] http://uk.eternityii.com/

E2 has not a lot of structure : the constraints are very local. The only constraint implying all the pieces is that no two pieces can be placed on a same position of the board. The colors of the tiles of E2 have been chosen by Christopher Monckton such that the backtracks occur deeply in the search tree (typically after the placement of about 160 pieces). The distribution of the colors is given on Figure 1. There are 22 colors. Color 0 is for the countour of the board. Colors 1-5 are only present on contour pieces (with a 0) and can only be used for the matching edges along the contour pieces. As can be seen, the distribution of the colors could not be more equally distributed. The number of colors has also been probably chosen such that there are probably very few solutions and such that the problem is not too constrained. Actually, it is very easy to make a valid contour even by hand and then to place about hundred of pieces. To reduce the number of solutions and the possibility of symmetrical solutions, Monckton designed the pieces all different and imposes the position of a clue piece is in the center of the board. This clue piece makes the solution even more difficult to find since even the rotational symmetries of the board are suppressed.

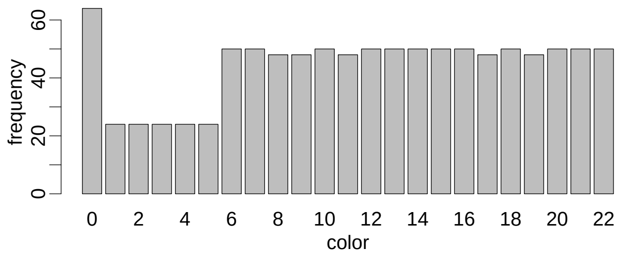

FIG. 1 – Distribution of the colors over the 256 pieces of E2

A lot of standard techniques to tackle combinatorial optimization problems can be used to solve E2. We think this challenging game might become a standard benchmark for combinatorial optimization.

A summary of our attempts to reach our best score of 458/480 follows :

1. Using an exact Constraint Programming (CP) model, we were not able to solve exactly instances with the same characteristics than E2 with n larger that 8. We had to relax about 80 junction constraints to be able to solve it. Hence we were not able to reach a score larger than 400/480 with a pure CP approach.
2. We tried a standard tabu stochastic local search with moves switching a pair of pieces. Starting from a random placement of the pieces, this approach was not able to reach scores larger than 410/480. Using the solution of the relaxed CP solution to initialize the local search we could raise our best score to about 425/480.
3. Finally we improved the local search using a very large neighborhood. Our neighborhood is able to move optimally more than two pieces at once. The only constraint is that the moved pieces must not be edge-adjacent. The number of moved pieces can hence be up to $n^2/2$ at once. This large neighborhood allowed us to raise our score to 458/480 in less than one day of computation on a standard computer. This score can be considered as state-of-the-art and demonstrates that the approach is promising to solve E2 and can certainly be improved to reach larger scores.

Contributions : The main contribution is the design of a large neighborhood that can be explored efficiently by solving an assignment problem. Our neighborhood can potentially be applied to any edge-matching puzzle and probably to other placement problems as well. We also show that a local search based on this neighborhood can reach higher scores when initialized with a (partial) solution obtained with CP on a (relaxed) E2 problem.

Outline : Section 2 describes the CP model. Section 3 explains how to build and solve the very large neighborhood. Section 4 gives our tabu procedure using the very large neighborhood. Section 5 presents a possible hybridization of CP and the local search to solve E2. Finally Section 6 concludes by giving experimental results.

2 Constraint Programming Model

Constraint Programming is a paradigm where a problem is modeled by declaring variables with their domains of possible values ,and stating constraints among the variables. The solver then tries to find an assignment of the variables to values of their domains such that every constraints are satisfied. The constraints are responsible to remove as much inconsistent values as possible from the domains of variables (propagation part). When a fixed point of the propagation is reached and that all the domains are not singletons and non empty, two branches are created. The first branch reduces the domain of a variable to a single value and the alternative branch removes this value from the domain. The search tree is usually explored in depth first way. In summary, the search of a solution is nothing else than exploring a search tree interleaving assignment and propagation and backtracking when a domain becomes empty. Of course one can hope to find more rapidly solutions by deciding heuristically which variable to instantiate to which value at each node. More details about CP in general can be found in [9].

The Variables : The different colors are $\{0, ..., c\}$. For each of the 16x16 positions (i, j) of the board we define the following variables.

- U_{ij}, R_{ij}, D_{ij}, L_{ij} with domains $\{0, ..., c\}$ represent respectively the colors of the up, right, down and left side of the piece coming in position (i, j) of the board,
- $I_{ij} \in \{0, ..., n^2 - 1\}$ is the identifier of the piece coming in that position.

We also add the following variables allowing a more efficient branching heuristic :

- $O_{ij} \in \{0..3\}$ represents the orientation of the piece coming in that position (number of clockwise quarter rotations),
- $IO_{ij} \in \{0, .., 4n^2 - 1\}$ represents together the piece identifier and its orientation in that position.

These variables are redundant and are linked to the first ones with

$$\forall (i, j) \in [1..n] \times [1..n] : \quad IO_{ij} = 4 \cdot I_{ij} + O_{ij}. \qquad (1)$$

Alternatively, one can also use element constraints [6] specifying that $IO_{ij} \in [4k..4k + 3]$ if and only if $I_{ij} = k$ and, $IO_{ij} \bmod 4 = l$ if and only if $O_{ij} = l$. This second set of channeling constraints are preferable to the arithmetic constraint (1). Indeed if the variable IO_{ij} is assigned, the values taken by I_{ij} and O_{ij} cannot be deduced if bound-consistency is achieved on (1) [5]. On the contrary, for the element constraints, as soon as IO_{ij} is assigned, the identifier of the piece $I_{ij} = IO_{ij}/4$ and the orientation $O_{ij} = IO_{ij} \bmod 4$ can be deduced.

The constraints : First we must have that the I_{ij}'s must be different. This can be enforced with an AllDifferent global constraint [8]. We must also have that the four colors U_{ij}, R_{ij}, D_{ij}, L_{ij} correspond to a physical piece.

A first way is to use extensional constraints. Since, n^2 physical pieces are given, for each one, four 4-tuples of colors can be created corresponding the four possible orientations of the piece. Hence, a total of $4n^2$ 4-tuples can be created and one must constraint $[U_{ij}, R_{ij}, D_{ij}, L_{ij}]$ to be one of these tuples. This can be realized with extensional constraints from [2].

Alternatively, one can also use element constraints. Indeed when the piece and its orientation are known for a position (i, j) one can pickup the values for U_{ij}, R_{ij}, D_{ij}, L_{ij} in four different arrays of constants encoding all the valid tuples.

The edge-matching constraints are simply expressed as $D_{i,j} = U_{i+1,j}$ and $R_{i,j} = L_{i,j+1}$. There is a constraint that the sides on the contour must be of color 0.

Branching Heuristics : Our experiments showed that it is more advantageous to branch on the IO_{ij} variables rather than, first fixing the pieces I_{ij}'s then the orientations O_{ij}'s. The heuristic we used is a classical first fail : the choice of the next variable to instantiate is the IO_{ij} with the smallest domain size. Ties are broken randomly. The choice of the value is chosen randomly.

Possible improvements : A big issue with our model is the depth first search used with the branching heuristic described above. The search can always place a large number of pieces (typically 160) before backtracking. The backtracks never occurs at the level of the first placed pieces. Hence these early choices are never reconsidered. A more clever search could use impacts and restarts to better guide the search and reconsider more rapidly the early choices [7]. Another possible improvement is to increase the filtering. For example, one could imagine to maintain arc-consistency on *domino* global constraints on the rows and columns of the board.

3 A Polynomial Time Very Large Neighborhood

Most combinatorial problem are intractable and E2 is one of these. Nevertheless we can generally obtain near optimal solutions using improving algorithms. The idea is to start from a solution an successively apply improving modifications on this solution. The possible modifications on a solution is called a neighborhood. The problem is that the solution can rapidly be trapped in a local optimum or in a cycling state. These issues can be avoided using various methaheuristics. A very popular and simple one, also very efficient in practice, is the tabu search [5] were some moves are forbidden for a while to avoid cycling and try to diversify the search space.

Whichever methaheuristic is used, there is no hope to reach good solutions without a pertinent neighborhood. The neighborhood is also very important to avoid local optimum. The larger is better, since there is more chance to escape from local optimum. A neighborhood is said to be very large with respect to the input data when it is exponential. Unfortunately, is can be very expensive to explore a large neighborhood at each iteration. There is generally a tradeoff between the speed and the size of the neighborhood. For certain problems, we are lucky and one can imagine very large neighborhoods that can be explored rapidly (in polynomial time). For the well known traveling salesman problem, useful very large neighborhoods have been designed [1]. The selection of the neighborhood is typically solved by an optimization problem such as finding a minimum path or cycle length, or solving a matching or assignment problem.

A small neighborhood For E2, the first neighborhood that comes to mind is probably to exchange two pieces and possibly rotate them. Let us call it the *swap and rotate* move. For a solution, there are $n^2 \cdot (n^2 - 1) \cdot 16$ possible

[5] Arc-consistency for arithmetic constraints is too costly

swap and rotate moves. To reduce this complexity, one can proceed in two steps by first choosing the first piece (typically one of the most violated one) and let the freedom on the other. The time complexity to explore the neighborhood is of $O(n^2)$ rather than $O(n^4)$.

A very large neighborhood We suggest to generalize the *swap and rotate* to more that two pieces. We consider swaps and rotates of a set of pieces. Unfortunately optimally swap and rotate a set of pieces is in general as difficult as solving E2. However, by choosing carefully our set of pieces we can replace them optimally into the holes in polynomial time. Indeed, if there are no edge-adjacent pieces in the set of removed pieces on the board, they can be replaced optimally in the holes by solving an assignment problem. Figure 2 shows a set of pieces without edge-adjacent removed from the current solution.

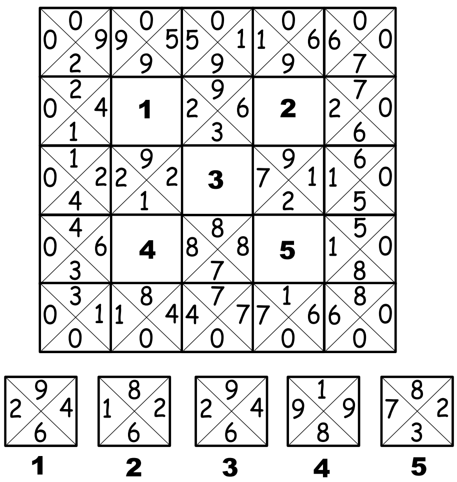

FIG. 2 – 5 non edge-adjacent pieces are removed from the current solution creating five holes.

The removed pieces can be reallocated optimally to the created holes by solving an assignment problem. The assignment problem is defined on a complete weighted bipartite graph between the set of removed pieces and the holes. An arc between a piece an a hole is labelled with a couple (r, w) :

- r is an optimal rotation of the piece when placed in this position and
- w is the number of matching edges of the piece when placed inside the hole with rotation r.

We have $r \in [0..3]$ (number of clockwise quarter of rotation) and $w \in [0..4]$. Table 1 gives arcs labels for the example of Figure 2.

When the optimal weights and rotations from the pieces to the holes are computed, one can compute a matching of maximal weight on this bipartite graph. The selected edges and the labels on the edges tell us how to place and rotate the pieces optimally in the holes. Finding a maximum weight matching is called an assignments problem. It can be solved in polynomial time for example with the Hungarian algorithm or the primal dual method in $O(m^4)$ [6] for a $m \times m$ weighted bipartite graph (see [3] for a dedicated book on assignment problems).

TAB. 1 – Labels (orientation,weight) of the arcs of the bipartite graph from the pieces to the holes of example of Figure 2.

	Holes				
Pieces	1	2	3	4	5
1	2,3	2,2	0,1	1,1	1,1
2	0,1	0,1	2,2	1,3	3,3
3	0,1	0,1	0,1	1,1	1,1
4	1,2	1,2	0,1	0,2	1,2
5	0,1	0,1	2,4	1,1	3,2

On our example, the selected edges (piece $\mapsto$ hole) are $(1 \mapsto 1)$, $(2 \mapsto 4)$, $(3 \mapsto 3)$, $(4 \mapsto 5)$ and $(5 \mapsto 3)$.

In summary, the steps to build our neighborhood are :

1. Select a set S of non edge-adjacent positions.
2. Compute the labels (r, w) for each of the $|S|^2$ arcs from pieces to the holes.
3. Compute the maximum weight matching solving an assignment problem.
4. The arcs of the matching give the permutation of the positions and the rotations are determined by the labels of the selected arcs.

The size of the explored neighborhood is $|S|! \cdot 4^{|S|}$ and $|S|$ can be up to $n^2/2$. The neighborhood is thus exponential.

Figure 3 gives more insight on the influence of size of the set S. We applied successively the large neighborhood move during 30 seconds by selection of random sets of non-edge adjacent pieces of size k. For each k we made 10 runs, each one starting from a random placement of the pieces. We tried the values 2, 4, 8, 16, 32 and 64 for k. Figure 3 gives the average evolution of the score (480 is a perfect solution) over the 30 seconds. The standard deviations are represented only for $k = 2$ and $k = 4$ for clarity reasons. The quality of the local optimum is not really improved for $k \geq 16$.

4 A Tabu Search

A tabu search tries to diversify the search making some moves tabu for some iterations (see [5] for more information on tabu search). Typically, the tabu moves are conceptually stored in tabu list. The number of iterations a move

[6] faster algorithms exist (e.g. $O(m^3)$) but are also more complicated [3].

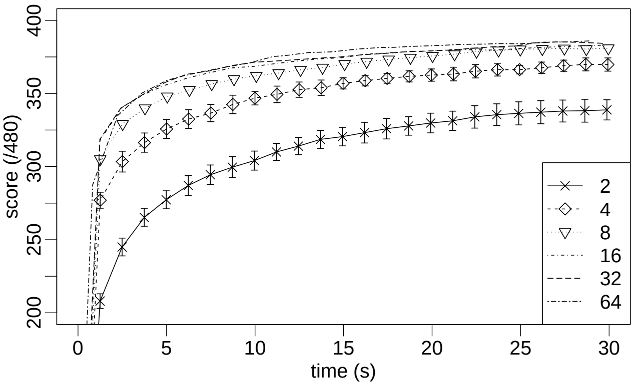

FIG. 3 – Influence of the number of selected pieces with the large neighborhood move on the quality of the local optimum

remains in the tabu list is called the tenure of the tabu list. Two different tabu lists can be imagined for the large neighborhood described above :

– A first tabu list can store during some iterations the positions that were recently chosen in the set of non-edge adjacent positions S. The positions will then be diversified over the board along the iterations.
– A second tabu list can forbid some permutations of positions. Assume that at the current iteration, a piece at position i is moved to the position j (with $i \neq j$). In order to avoid cycling effects, it is desirable to avoid the opposite move during some iterations. The pair (j, i) is added to the tabu list specifying that a piece in position j cannot move to position i while (j, i) is in the tabu list. This can be easily achieved by giving a very small weight to the arc (j, i) during the construction of the bipartite graph.

Algorithm 1: VLN tabu search algorithm for E2

$tabu_1 \leftarrow list()$.
$tabu_2 \leftarrow list()$.
while *not end condition* **do**
  if *diversify condition* **then**
    diversify
  1. Select a subset S of non edge-adjacent pieces not in $tabu_1$.
  2. Add elements of S in $tabu_1$ for some iterations.
  3. Compute an apply the optimal move on S by solving the assignment problem and by penalizing arcs in $tabu_2$.
  4. Add the reverse arcs of moved pieces to $tabu_2$.

  if *intensification condition* **then**
    restore the best solution

Algorithm 1 is a quite high level description. We give a more detailed view of some parts of our implementation.

– In the selection of S, most violated positions are preferably chosen.
– The diversify condition occurs when a plateau of given length is met.
– The diversification consists of making a given number of random swap and rotate moves.
– The intensification condition occurs when the best score is not improved for a given number of consecutive plateau detections.
– The end condition occurs after a given number of intensification's.
– The time period move remains in a tabu list is chosen randomly between two bounds.

5 Hybridization

The large neighborhood described in previous section allows to replace optimally in polynomial time non edge-adjacent pieces on the board. This neighborhood can also be used to reduce the cost of a brute-force approach. The number of possible placement of the pieces on the board is $n^2! \cdot 4^{n^2}$. It is possible to reduce drastically this number by placing only one over two pieces like black squares of a chess board. The number of possible placements is then $\frac{n^2!}{(n^2/2)!} \cdot 4^{n^2/2}$. It remains $n^2/2$ non edge-adjacent empty positions. These positions are optimally completed with the matching move.

Another exact approach can combine constraint programming and the large neighborhood :

– E2 is too hard for constraint programming. Nevertheless a solution can be found by allowing some non matching edges. If we choose to relax non-matching edges of some non edge-adjacent positions (e.g. a subset of the black pieces of a chess like board), one can hope to find a solution with CP.
– The relaxed positions can be filled optimally with the large neighborhood based on the assignment problem.
– If they are some violations, repeat the procedure for the next solution given by CP.

We relaxed a number of non edge-adjacent positions chosen randomly. For each number, we generated 30 relaxed instances. Figure 4 gives the number of instances that could be solved within 30 seconds. Clearly the number of relaxed pieces must be larger than 24 to have a good chance of solving the relaxed problem.

The procedure described above can also generate good initial solutions. Our hybridization starts a local search with Algorithm 1 on such solutions. If CP is not able to find a solution in a given limit of time, we complete randomly the largest partial assignment (deepest node of the search tree).

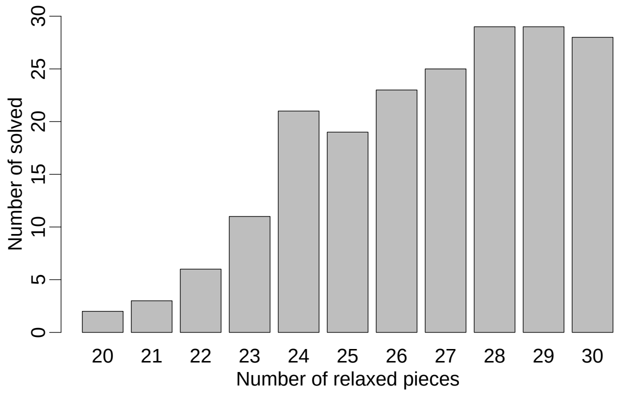

FIG. 4 – Number of solved relaxed problems on 30 runs.

6 Experimental Results and Conclusion

We experiment in this section the initializations with CP for a varying number of relaxed positions. For each number of relaxed positions we made 20 runs with the hybridization described in previous section. We also tried with a random initialization, that is by placing randomly the pieces on the board. The boxplots are drawn on Figure 5. The boxes represent the median and quartiles and the whiskers extend to the extreme values.

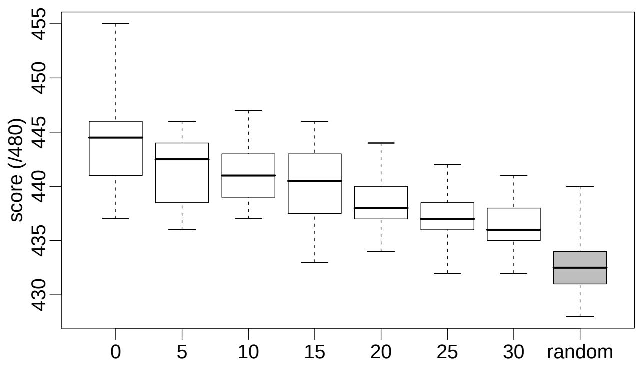

FIG. 5 – Boxplots of the scores on 20 runs with CP initialization using a varying number relaxed pieces (0,5,10,15,20,25,30) and a random initialization

It seems that the local search Algorithm 1 obtains better results with a CP initialization than with a random one (gray box versus other boxes). What is more surprising is that higher scores are obtained with less relaxed positions. Remember that for less than 20 relaxed positions, CP is not able to find a solution. In this case, the initial state is the largest partial assignment completed randomly with remaining pieces. Intuitively we thought that higher was the score of the initial state, better it was to start the local search and finally reach very good scores. It is not the case. Starting with a good small partial assignment gives better results than starting with larger partial assignment of worse quality. Our explanation is that if we relax say 30 positions and find a solution to this relaxed problem then replace optimally the relaxed pieces, we start precisely in a local optimum and it is very difficult for our local search to improve it.

By letting the program run 24 hours, we were able to generate several solutions with a score of 458 which can be considered as a state-of-the-art result for E2.

All experiments were realized on CPU Intel Xeon(TM) 2.80GHz. We used the Gecode 2.0.1 CP library [11]. The local search with large neighborhood was implemented in C++. Our implementation maintains incrementally the violations along the moves.

Conclusions and Perspectives We have designed a very large neighborhood for E2 that can be efficiently explored by solving an assignment problem. We also explained how this neighborhood can be used to reduce the number of combinations in an exact brute-force method. We gave a basic tabu search procedure using the neighborhood and showed that an initialization using a (partial) solution generated with CP can enhance the results obtained by the local search.

We don't believe that E2 will be solved with exact methods nor by heuristic methods in the near future. But we think that the combinations of the two can help to reach better solutions. Standard CP is maybe not a good choice for the initialization because CP backtracks as soon as a domain is empty. The approaches used by most people is rather to place as many pieces as possible with matching edges. Backtracking when a domain is empty is not a good idea to this end. Indeed, one could continue with the non empty domains until all the domains are empty. People pretend to be able to place about 210 pieces with these approaches while CP is not able to produce partial solutions with more than 170 placed pieces. As future work we would like to try such initializations before starting our local search procedure. The local search algorithm can also be imagined with other methaheuristics and alternative moves.

Références

[1] Ravindra K. Ahuja, Özlem Ergun, James B. Orlin, and Abraham P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1-3) :75–102, 2002.

[2] Christian Bessière, Jean-Charles Régin, Roland H. C. Yap, and Yuanlin Zhang. An optimal coarse-grained arc consistency algorithm. *Artif. Intell.*, 165(2) :165–185, 2005.

[3] R. Burkard, M. DellAmico, and S. Martello. *Assignment Problems*. SIAM Monographs on Discrete Mathematics and Applications, 2008.

[4] Erik D. Demaine and Martin L. Demaine. Jigsaw puzzles, edge matching, and polyomino packing : Connections and complexity. *Graph. Comb.*, 23(1) :195–208, 2007.

[5] Fred Glover. *Tabu Search*. Kluwer Boston, Inc., 1998.

[6] Van Hentenryck P. and Carillon J.-P. Generality versus specificity : an experience with ai and or techniques. *In Proceedings of AAAI-88*, 1988.

[7] Philippe Refalo. Impact-based search strategies for constraint programming. In *CP*, pages 557–571, 2004.

[8] Jean-Charles Régin. A filtering algorithm for constraints of difference in csps. In *AAAI '94 : Proceedings of the twelfth national conference on Artificial intelligence (vol. 1)*, pages 362–367, Menlo Park, CA, USA, 1994. American Association for Artificial Intelligence.

[9] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.

[10] Yasuhiko Takenaga and Toby Walsh. Tetravex is np-complete. *Inf. Process. Lett.*, 99(5) :171–174, 2006.

[11] Gecode Team. Gecode : Generic constraint development environment. available from http ://www.gecode.org. 2006.