Dynamic Branch&Bound, an optimization counterpart for dynamic backtracking

Marius-Călin Silaghi

Florida Institute of Technology
msilaghi@fit.edu

Abstract. Here we show how the idea of dynamic backtracking can be applied to branch and bound optimization. This is done by exploiting the concept of valued nogood introduced in [6, 16, 17]. However, simple replacement of nogoods with valued nogoods does not lead to a correct algorithm. We show that a way to achieve correctness is to use at each variable a separate nogood storage for each position that the variable holds in the order on assigned variables. This is a slightly different version to the one in [5].
Constraint Satisfaction Problems (CSPs) are typically addressed with some kind of backtracking algorithm. A seminal work that helped to deeply understand and unify many backtracking algorithms was [9]'s dynamic backtracking (DB). Optimization problems are often solved with branch and bound (B&B) algorithms, a family of algorithms related to backtracking. However, no equivalent of DB's heuristic was so far developed for branch and bound. The algorithm proposed here, Dynamic Branch&Bound (DBB), is optimal and guaranteed to terminate, having polynomial space complexity and a reordering policy similar to Dynamic Backtracking. The experimentation performed validates the correctness of the theory. Like Dynamic Backtracking, whose reordering heuristic is known to be rather inefficient, the interest of the proposed algorithm is shown to be mainly theoretical.

1 Introduction

Constraint Satisfaction provides a framework for modelling and solving problems that can be represented by a set of variables taking values from corresponding domains and where the possible assignments are restricted by a set of constraints (predicates). A large set of flexible/general algorithms exist for solving Constraint Satisfaction Problems (CSPs). Some of the most general such algorithms are based on inference with nogoods [9, 13, 8, 1, 3]. CSPs do not model any kind of optimization requirement. An extension allowing for modeling some optimization problems is given by Valued CSPs.

Definition 1 (VCSP). *A Valued CSP is defined by a set, X, of variables $X = \{x_1, ..., x_m\}$ where x_i can take values from a domain D_i, and a set of functions, $f_1, f_2, ..., f_i, ..., f_n$, of type $f_i : X_i \rightarrow \mathbb{R}_+$, where $X_i \subseteq X$.*

The valued constraint satisfaction problem consists in finding $\operatorname{argmin}_{\boldsymbol{x}} \sum_{i=1}^{n} f_i(\boldsymbol{x}_{|X_i})$*, where* $\boldsymbol{x}$ *assigns values for all variables in* X*.*

For discrete problems with binary functions f_i, the functions can be represented by matrices with values. To model CSPs with VCSPs, infeasible tuples can be set to ∞, and feasible ones to 0.

Most algorithms for addressing CSPs are not directly applicable to VCSPs, and in particular no optimization equivalent was known so far for versions of Dynamic Backtracking [9, 13, 8, 3, 10]. Here we show how the first of these algorithms can be adapted to solve valued constrained satisfaction problems by exploiting a new type of nogoods, introduced in [16, 17].

2 Background

Many constraint satisfaction algorithms are based on reasoning with nogoods but only few optimization algorithms boast some usage of nogoods. We first introduce the concept of nogood and nogood-based backtracking algorithms for constraint satisfaction, as well as some well known optimization algorithms and value nogoods.

2.1 Nogoods

A nogood is a list of assignments that was proven impossible, by inference using constraints [18]. It has the form $\neg(\langle x_1, v_1\rangle, ..., \langle x_t, v_t\rangle)$ and is often denoted $\neg N$ where N is a set of assignments.

Given a nogood $\neg(\langle x_1, v_1\rangle, ..., \langle x_t, v_t\rangle)$, one can infer an *elimination explanation* [9] for removing the value v_t from the domain of x_t given the assignments $E = \langle x_1, v_1\rangle, ..., \langle x_{t-1}, v_{t-1}\rangle$. This inference is denoted (v_t, E), and is semantically equivalent to an *applied nogood*, (i.e. the inference): $(\langle x_1, v_1\rangle, ..., \langle x_{t-1}, v_{t-1}\rangle) \rightarrow \neg\langle x_t, v_t\rangle$

Given a set of such applied nogoods, one for each value of a variable x_t with domain $D_t = \{v_1, ..., v_d\}$, the following version resolution can be used to infer new nogoods.

$$
\begin{array}{l}
(\langle x_1^1, v_1^1\rangle, ..., \langle x_{t-1}^1, v_{t-1}^1\rangle) \rightarrow \neg\langle x_t, v_1\rangle \\
... \\
(\langle x_1^d, v_1^d\rangle, ..., \langle x_{t-1}^d, v_{t-1}^d\rangle) \rightarrow \neg\langle x_t, v_d\rangle \\
\hline
\neg(\langle x_1^1, v_1^1\rangle \wedge ... \wedge \langle x_{t-1}^1, v_{t-1}^1\rangle) \wedge ... \wedge (\langle x_1^d, v_1^d\rangle \wedge ... \wedge \langle x_{t-1}^d, v_{t-1}^d\rangle)
\end{array}
$$

2.2 Dynamic Backtracking

Dynamic backtracking [9] is an algorithm where all inferences are performed using a polynomial number of nogoods, leading to a strong logical fundation. The

stength that was proven for such an approach is that it allows very flexible ways to systematically explore the search space, in particular a scheme for reordering of variables that were not possible otherwise. Additional previously impossible heuristics for reordering variables in systematic search, but enabled by [9] nogoods storage, were discovered in [13, 8, 3]. These reordering heuristiques were shown to perform well only an limited types of problems [9, 1, 10]. The basic method is given in Algorithm 1.

1. If all variables are assigned, return. Otherwise select a variable i that is not yet assigned and use a mechanism generating elimination explanations, removing as many values as possible from i using constraints with previously assigned variables.
2. If i has any value that was not eliminated, assign it with such a value and go to step 1.
3. If all values of i were eliminated, apply a resolution step on all elimination explanations of its values to abtain a new nogood N. If N is empty then return failure. Otherwise let j be the last variable in N and transform N into an elimination explanation for the current value of j. Remove any elimination explanation involving j, unassign j and go to step 1 (optionally select i=j at next step).

Algorithm 1: Dynamic Backtracking (informal description)

2.3 Optimization Algorithms

Some of the most well known optimization algorithms guaranteeing optimality are Branch&Bound and A*. Branch&Bound can be seen as a version of backtracking for optimization. While A* requires exponential space, a kind of polynomial space iterative deepening of A* is commonly used in distributed algorithms under the name of ADOPT [14]. An important class of optimization algorithms is based on variable elimination [7]. The algorithms in this class have both space complexity and time complexity that is exponential in the induced width of the graph. The idea of consistency maintenance, that is ubiquitous in constraint satisfaction [15, 2], is shown to be also useful in optimization in the work of Larrosa [11].

2.4 AND-OR Optimization

It was shown in [4] that given a depth first search spanning (DFS) tree of the constraint graph of a problem, search can proceed in parallel and independently on branches of this DFS tree, as long as results are aggregated at the nodes where the branches meet. The concept was shown valid for optimization problems in [14] and was applied to Branch&Bound in [12]. Certain versions of this algorithm can employ nogoods as a way of storing additional constraints. No previous algorithm is known to us whose inference is based solely on nogoods.

2.5 Valued Nogoods

With VCSPs a cost is associated to each partial assignments by valued constraints whose variables are involved in those assignments. To avoid accumulating twice the contribution of the same constraint, one can keep track of the culprit constraints used for inferring each cost. This is done using sets of references to culprit constraints.

Definition 2 (SRC). *A set of references to constraints (SRC) is a set of names of constraints of the distributed VCSP. It is a set of symbols, each of them standing for a distinct constraint.*

An example of a SRC is $\{f_{1,2}, f_{3,4}\}$ where $f_{1,2}$ and $f_{3,4}$ are names of two distinct constraints of a VCSP.

Definition 3 (Valued Nogood). *A valued* nogood *has the form* $[\Gamma, c, N]$ *where* Γ *is a set of references to constraints having cost at least* c*, given a set of assignments,* N*, for distinct variables.*

The (classic/hard) nogood is obtained for $c{=}\infty$.

By deciding that a nogood $[\Gamma, c, (\langle x_1, v_1\rangle, ..., \langle x_i, v_i\rangle)]$ will be applied to a certain variable x_i, one obtains a cost assessment for x_i tagged with the SRC Γ,[1] denoted $(\Gamma, v_i, c, (\langle x_1, v_1\rangle, ..., \langle x_{i-1}, v_{i-1}\rangle))$, and meaning that $(\langle x_1, v_1\rangle, ..., \langle x_{i-1}, v_{i-1}\rangle) \rightarrow$ (assignment $\langle x_i, v_i\rangle$ has cost c for Γ).

Definition 4 (Cost Assesment (CA)). *A cost assesment of variable* x_i *has the form* (Γ, v, c, N) *where* Γ *is a set of references to constraints having cost with lower bound* c*, given a set of assignments,* N*, for distinct variables, and the value* v *in the domain of* x_i*.*

A cost assesment is always specified together with the variable for which it applies.

Proposition 1 (min-resolving). *A set of cost assesments for* x_i*,* $(\Gamma_i, v_i, c_i, N_i)$ *where* $\cup_i\{v_i\}$ *covers the whole domain of* x_i *and assignments are identical in all* N_i*, can be combined into a new valued nogood. The obtained valued nogood is* $[\Gamma, c, N]$ *such that* $\Gamma{=}\cup_i\Gamma_i$*,* $c{=}\min_i(c_i)$ *and* $N{=}\cup_i N_i$*.*

Example 1. For the graph coloring problem in Figure 1, x_1 is colored red, x_2 yellow and x_3 green. Assume that due to its constraints with the three neighbors, one can infer for each of the valued of x_4 the following valued nogoods:

(r): $[\{f_{1,4}\}, 10, \{(x_1, r), (x_4, r)\}]$ obtaining CA $(\{f_{1,4}\}, r, 10, \{(x_1, r)\})$
(y): $[\{f_{2,4}\}, 5, \{(x_2, y), (x_4, y)\}]$ obtaining CA $(\{f_{2,4}\}, y, 5, \{(x_2, y)\})$
(g): $[\{f_{3,4}\}, 7, \{(x_3, g), (x_4, g)\}]$ obtaining CA $(\{f_{3,4}\}, g, 7, \{(x_3, g)\})$

By min-resolution on these CAs one obtains the valued global nogood $[\{f_{1,4}, f_{2,4}, f_{3,4}\}, 5, \{(x_1, r), (x_2, y), (x_3, g)\}]$, meaning that given the coloring of the first 3 nodes there is no solution with cost lower than 5 for the constraints $\{f_{1,4}, f_{2,4}, f_{3,4}\}$.

[1] its extension to set of values is denoted valued conflict list in [16]

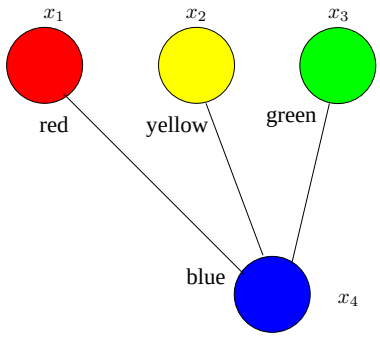

Fig. 1. MIN resolution on valued nogoods

2.6 Combining valued nogoods

If several cost assesments (valued nogoods) for the same value of a variable are infered from disjoint SRCs, then they can be combined into stronger valued nogoods. This mechanism is what enables the usage of the concepts behind AND-OR optimization in our algorithm.

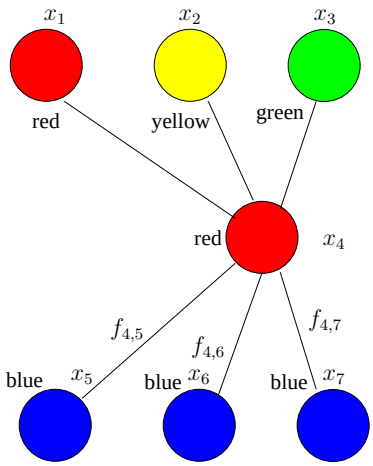

Fig. 2. SUM-inference on CAs

Proposition 2 (sum-inference). *A set of cost assesments,* (Γ_i, v, c_i, N_i) *where* $\forall i, j : (i \neq j) \Rightarrow (\Gamma_i \cap \Gamma_j = \emptyset)$*, and the assignment of any variable* x_k *is identical in any* N_i *where* x_k *is present, can be combined into a new cost assesment.*

The obtained cost assesment is (Γ, v, c, N) *such that* $\Gamma = \cup_i \Gamma_i$, $c = \sum_i (c_i)$, *and* $N = \cup_i N_i$.

Example 2. For the graph coloring problem in Figure 2, x_1 is colored red, x_2 yellow, x_3 green, and x_4 red. Assume that due to its constraints with the three following neighbors x_5, x_6, and x_7, one has infered for (x_4, r) the following valued nogood:

- $[\{f_{4,5}\}, 5, \{(x_2, y), (x_4, r)\}]$ obtaining CA $(\{f_{4,5}\}, r, 5, \{(x_2, y)\})$
- $[\{f_{4,6}\}, 7, \{(x_1, r), (x_4, r)\}]$ obtaining CA $(\{f_{4,6}\}, r, 7, \{(x_1, r)\})$
- $[\{f_{4,7}\}, 9, \{(x_2, y), (x_4, r)\}]$ obtaining CA $(\{f_{4,7}\}, r, 9, \{(x_2, y)\})$

Due to x_4's constraint with X_1, one can infer for (x_4, r) the following valued nogood:

- $[\{f_{1,4}\}, 10, \{(x_1, r), (x_4, r)\}]$ obtaining CA $(\{f_{1,4}\}, r, 10, \{(x_1, r)\})$

By sum-inference on these CAs one obtains for x_4 the CA $[\{f_{1,4}, f_{4,5}, f_{4,6}, f_{4,7}\}, r, 31, \{(x_1, r), (x_2, y)\}]$, meaning that given the coloring of the first 2 nodes, coloring x_4 in red leads to a cost of at least 31 for the constraints $\{f_{1,4}, f_{4,5}, f_{4,6}, f_{4,7}\}$.

2.7 Dynamic Branch&Bound

A mechanism for inferring CAs from constraints is called a *CA mechanism*. Given a correct, complete, and concise *CA mechanism* ε, (where these concepts parallel those for elimination mechanisms in [9]), we can formulate generalizations of dynamic backtracking for solving VCSPs.

Dynamic Branch and Bound (DBB) is an algorithm where we parallel the Dynamic Backtracking (DB) algorithm of [9], having similar strengths and weaknesses [1, 3]. Like with DB, a partial solution is being built, and when it is shown unacceptable (by being more expensive than a current bound, B) a culprit variable is isolated and uninstantiated. Uninstantiating a variable consists of discarding all nogoods based on its previous assignment, and moving the variable to the set of unassigned variables.

Dynamic Backtracking assumes that variables are going to be reordered during the exploration of the search space. We will apply a similar approach to implement a dynamic branch and bound, with initial bound B. The current order on variables is maintained using an array o, where $o[i]$ specifies the variable on position i, i.e., $x_{o[i]}$. The position of a variable's index k in the array o is denoted $pos(k)$, $o[pos(k)] = k$. An array v holds the values of currently assigned variables, $v[i]$ being the value assigned to x_i.

To manage CAs (that replace the nogoods of DB), we use matrices l, h and ca.

- $l[i, k]$ stores a CA for $x_i = k$ that is infered solely from the constraints between x_i and prior variables using a CA mechanism.

```
procedure DBB(B) do
    i := 0; //last instantiated variable;
    k := 1; //variable under inspection;
    forever do
        if (k > n) then
            store solution and compute new B = Σ_y cost(l[y, x[y]]);
            k − −; // or first var x_i with h[i, v[i]] ≥ B;
        y := o[k]; // index of current variable;
        if (k > i) then
            foreach value j of x_y do
                l[y, j] := ε(o, x, k, j); // use constraints with previous variables;
                h[y, j]:=sum-inference(l[y_≤, j], ca[y, j, ∗]); // up to cost B;
            v[y] := select(l, h, y, k); // with minimal cost h(), lower than B;
            if (v[y] valid) i := k; // new variable instantiated;
        vn:=min-resolution(h[y, ∗]);
        j:=target(vn);
        if (vn better than ca[j, x[j], pos(j)]) then
            replace ca[j, v[j], pos(j)] with vn2ca(vn, j);
            update h[j, v[j]] by sum-inference;
        if (cost(h[j, v[j]]) ≥ B) then
            k := pos(j); continue; // optional;
        if (cost(vn) ≥ B) then
            if (j < 0) then
                break;
            reorder x_j after x_i (or after y if v[y] invalid);
            i − −; k := i + 1; continue;
        if (cost(h[y][x[y]]) ≥ B) then
            reorder x_k after x_i; i − −;
        k := i + 1;
```

Algorithm 2: Dynamic Branch&Bound (DBB)

- $ca[i, k, t]$ stores a CA for $x_i = k$ that is computed when i is $o[t]$, by combining different CAs with the mentioned inferences.
- $h[i, k]$ stores a CA for $x_i = k$ that is infered from $ca[i, k, t]$ and $l[j, v[j]]$ for all j positioned before i in the current order.

To denote the set of elements having all possible values of an index, we will use an "*" for that index. E.g., $h[i, *]$ is the set of the CAs for all possible values of x_i.

Remark 1. In computing the CAs in $h[i, v]$ we ensure that their costs are equal or higher to the cost of the constraints between variables on positions smaller or equal to i, given the current assignments. Such a computation of $h[i, v]$ is done by first combining all corresponding CAs with

$$vca = sum\_infernce(ca[i, v, *]).$$

Then compute

$$ld = sum\_inference(\{l[j, v[j]] | pos(j) \leq pos(i), \text{with SRCs disjoint from } vca\}$$

and

$$lo = sum\_inference(\{l[j, v[j]] | pos(j) \leq pos(i), \text{with SRCs not disjoint from } vca\}$$

Finally,

$$h[i, v] = (cost(lo) > cost(vca))?sum\_inference(lo, ld) : sum\_inference(vca, ld).$$

The function $cost(N)$ receives as parameter a CA or a valued nogood and returns its assessed cost. The function $target(N)$ receives as parameter a valued global nogood and returns the position of the latest instantiated variable that is specified in N.

In Algorithm 2 we use an index i to specify the position of the last assigned variable, and an index k to specify the position of the variable that we are analyzing at this cycle. To instantiate a new variable x_j we use function *select* which selects the value v with the smallest (most promissing) cost in its $h[j, v]$. If the smallest such cost is larger than the current bound B, then -1 is returned.

The *target* variable in a valued nogood vn is the one with the largest position. We say that the domain of a variable is *wiped out* when the nogood vn generated by min-resolution on it has a cost higher than B. If a variable's domain is wiped out than the target variable in vn is the one that will be reordered and uninstantiated. Nogoods are used to update CAs of already instantiated variable, which can lead to getting an h with cost higher than B. Any such variable is reordered and unassigned. When the CA of some instantiated variable is modified, a cycle verifying it can be initiated by starting a round with k set to its position.

Theorem 1. *Dynamic B&B is optimal and terminates.*

Proof. Assuming termination, the optimality of DBB is guaranteed by the fact that all used operations (inferences) are logically sound. Therefore if DBB returns that there is no solution with value smaller than a result B, that is founded.

Let us now prove termination (absence of an infinite loop). When a variable is on the first position, it will receive only CAs whose culprit assignments list is empty. These CAs never expire due to a reordering, and they can only have an increasing threshold. As thresholds are limited and increase monotonically, the total possible set of new empty CAs that a variable can receive for all its values is finite. Therefore, eventually, the variable and assignment on the first position will no longer change.

Let us now prove the termination by induction. Assuming that the variable asignments for the variables on the first k positions no longer change. It follows by the same reasoning as above that, eventually, the variable on position $k + 1$ will no longer change.

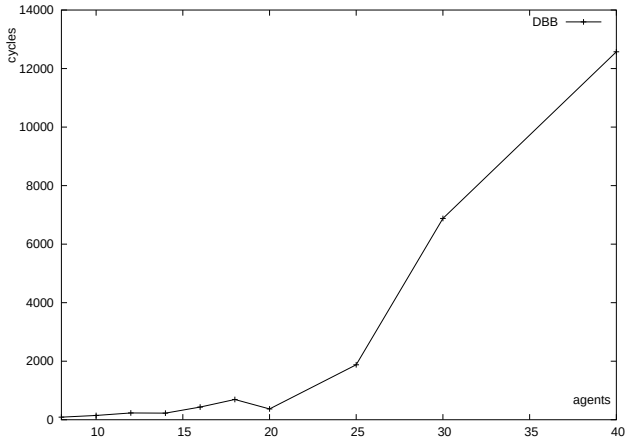

Fig. 3. Number of search nodes on problems with density .2.

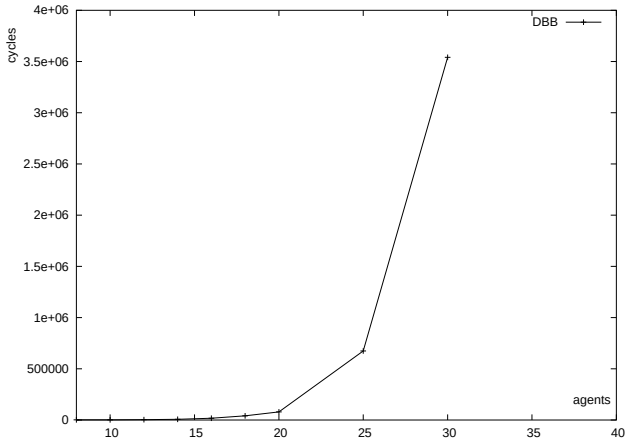

Fig. 4. Number of search nodes on problems with density .3.

3 Experimentation

We have implemented DBB and we tested it on problems with 8, 10, 12, 14, 16, 18, 20, 25, 30, and 40 variables. For each of these problem sizes we performed test on problems with density .2 and respectively with density .3. The density of a (binary) constraint problem's graph with n variables is defined by the ratio between the number of binary constraints and $\frac{n(n-1)}{2}$. Results are averaged on the 25 problems with the same parameters.

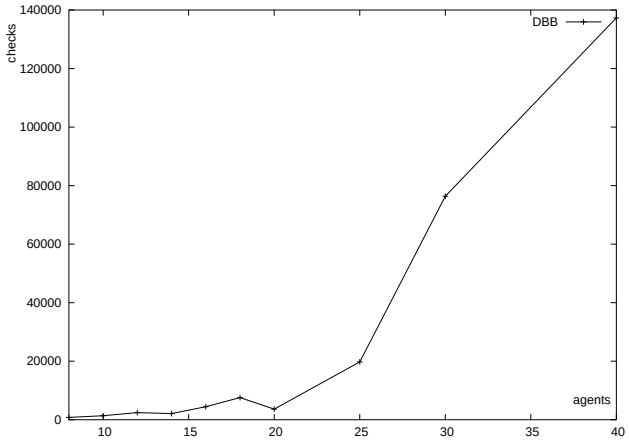

Fig. 5. Number of constraint checks on problems with density .2.

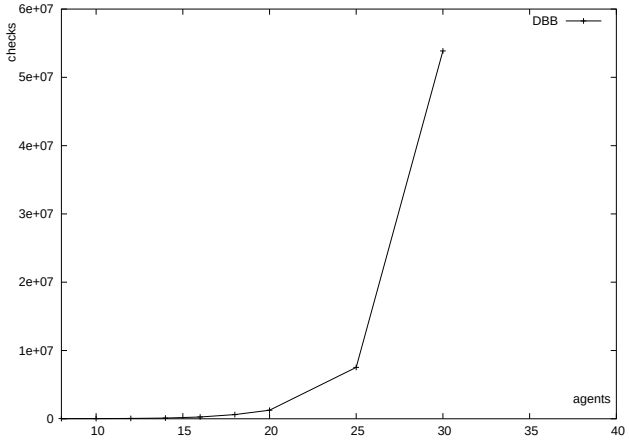

Fig. 6. Number of constraint checks on problems with density .3.

The quality of the found solutions was identical to the quality of the results of other techniques (notably ADOPT and ADOPT-ng), run on the same problems, confirming the optimality of DBB.

4 Conclusions

We have proposed an optimization counterpart for Dynamic Backtracking by paralleling that algorithm using valued nogoods instead of classic nogoods and cost assesments instead of elimination explanations. The obtained technique is

called Dynamic Branch&Bound. The optimality and termination of the algorithm are based on storing a separate set of cost assesments for each position occupied by each variable.

5 Acknowledgements

We want to thank Rina Dechter for comments.

References

1. A. Backer. The hazards of fancy backtracking. In *Proceedings of the 12th National Conference on AI*, pages 288–293. AAAI, 1994.
2. C. Bessière. Arc-consistency in dynamic constraint satisfaction problems. In *AAAI'91*, 1991.
3. C. Bliek. Generalizing partial order and dynamic backtracking. In *AAAI-98 Proceedings*, pages 319–325, Madison,Wisconsin, July 1998. AAAI.
4. Z. Collin, R. Dechter, and S. Katz. On the feasibility of distributed constraint satisfaction. In *Proceedings of IJCAI 1991*, pages 318–324, 1991.
5. Pierre Dago. Backtrack dynamique valué. In *JFPLC*, pages 133–148, 1997.
6. Pierre Dago and Gérard Verfaillie. Nogood recording for valued constraint satisfaction problems. In *ICTAI*, pages 132–139, 1996.
7. Rina Dechter. *Constraint Processing*. Morgan Kaufman, 2003.
8. M. Ginsberg and D. McAllester. Gsat and dynamic backtracking. In J.Doyle, editor, *Proceedings of the 4th IC on PKRR*, pages 226–237. KR, 1994.
9. M. L. Ginsberg. Dynamic backtracking. *Journal of AI Research*, 1, 1993.
10. Narendra Jussien, Romuald Debruyne, and Patrice Boizumault. Maintaining arc-consistency within dynamic backtracking. In *CP'2000*, Singapore, 2000. Springer.
11. J. Larrosa. Node and arc consistency in weighted csp. In *AAAI-2002*, Edmonton, 2002.
12. Radu Marinescu and Rina Dechter. And/or branch-and-bound for graphical models. In *IJCAI*, Edinburgh, 2005.
13. D. A. McAllester. Partial order backtracking. *Research Note: 'ftp://ftp.ai.mit.edu/people/dam/dynamic.ps'*, 1993.
14. Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *AIJ*, 161, 2005.
15. D. Sabin and E. C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *ECAI*, 1994.
16. M.-C. Silaghi. *Asynchronously Solving Distributed Problems with Privacy Requirements*. PhD Thesis 2601, (EPFL), June 27, 2002. `http://www.cs.fit.edu/~msilaghi/teza`.
17. M.-C. Silaghi and M. Yokoo. Nogood-based asynchronous distributed optimization (ADOPT-ng). In *AAMAS (to appear)*, 2005.
18. Richard M. Stallman and Gerald J. Sussman. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9:135–193, 1977.