Application of Constraint Networks Filtering Techniques to Truth Maintenance Systems

Christian Bessiere

Application of Constraint Networks Filtering Techniques to Truth Maintenance Systems

Christian Bessière
LIRMM, University of Montpellier II
860, rue de Saint Priest, 34090 Montpellier, France
Phone: (33) 67 14 85 63 Email: bessiere@crim.fr

Abstract

A truth maintenance system (TMS) is a general problem solving facility designed to work in tandem with an inference engine. Justification-based TMSs (the most commonly used TMSs) have not a sufficient expressive power for many applications. Logic-based TMSs overcome this limitation but are untractable. So, McAllester, or Forbus and de Kleer, have proposed to encode formulae in clauses and to run the efficient Boolean Constraint Propagation algorithm (BCP), what lose the completeness of the deductions. In this paper, we introduce the model of Constraint Networks (CNs), underlining that the notion of local completeness (called local-consistency in CNs) has been widely treated in CNs. We then propose an encoding of McAllester's TMS in Dynamic CN to be able to use CNs techniques. We show that not only we can compute as many deductions as BCP for the same time-consuming, but also we can outperform the local completeness computed for a slight more running-time.

AI topic: reasoning
Domain area: Expert systems, problem solving
Impact: The estimated benefit is 40% more information deduced (in average) in a problem solver, while keeping reasonable time-consuming.

1. Introduction

In this paper, we briefly recall that Justification-based TMSs have not a sufficient expressive power for many applications (see [1]). We present Logic-based TMSs (LTMSs) and remark that efficiency and completeness cannot hold together since any SAT problem can be encoded in an LTMS. Then, the problem is to find a compromise between efficiency and completeness. The usual way is to encode arbitrary formulae into Conjunctive Normal Form (CNF) and to run the efficient Boolean Constraint Propagation algorithm (BCP). De Kleer has shown in [2] that encoding formulae into CNF results in losing locality properties of the original formulae and leads BCP to fail on some deductions. He proposed to solve local incompleteness by locally compiling formulae into their prime implicates.

We propose here another alternative, starting from the observation that the notions of incompleteness or local completeness are well known and have been widely treated in Constraint Networks (CNs). The idea is to encode an LTMS into a CN and to find CNs filtering techniques which compute the same deductions as the TMS's algorithm BCP, for the same cost. Then, we will look for another filtering technique which provides more deductions (i.e. less incomplete) for a sligth more running-time.

The rest of the paper is organized as follows: Section 2 presents the basic definitions on TMSs and underlines the drawbacks and advantages of each class saying why McAllester's TMS is interesting. Section 3 introduces CNs definitions necessary to understand the next section. Section 4 begins with a representation of McAllester's TMS in a Dynamic CN, gives properties of this representation and shows that we can improve the deduction algorithm using a well known CNs filtering technique. An experimental comparison of BCP and this new approach is given. Section 5 is a conclusion on the interest of this approach.

2. Definitions and preliminaries on truth maintenance systems

A **truth maintenance system** (**TMS**) is a general problem solving facility designed to work in tandem with an inference engine. A TMS takes assertions from the problem solver (or inference engine). It produces deductions from the set of assertions and maintains justifications of its deductions. It incrementally updates its beliefs when assumptions are added or removed. It does dependency directed backtracking (DDB) when a contradiction arises to track down the assumptions which underlie that contradiction.

A large part of the definitions given in sections 2.1. and 2.2. are taken from [3], [2] and [1]. We have limited the discussion to monotonic TMSs. Doyle's TMS with non-monotonic justifications is not considered here, involving problems out of the topic of this paper.

2.1. Logic-based TMS (LTMS)

Justification-based truth maintenance system (**JTMS**) is the simplest and most commonly used type of TMS. A JTMS deals only with Horn clauses. It is sufficient for some applications but the expressive power is very limited.

Many applications need to express more than just Horn clauses. **Logic-based TMSs** (**LTMSs**) overcome this limitation. Unlike the JTMS, the LTMS allows one to express negation explicitly and to use the four usual binary connectives ($\vee$, $\wedge$, $\rightarrow$ and $\equiv$). Therefore, it can represent any propositional formula.

Definition 2.1. A LTMS involves a set of *propositional symbols* S, a set of *propositional formulae* F and a set of *assumption literals* A. A *literal* is either a propositional symbol σ of S or the negation $\neg\sigma$ of a symbol σ. A formula is defined in the usual way with the connectives $\neg$, $\vee$, $\wedge$, $\rightarrow$ and $\equiv$ in terms of propositional symbols. The reason for distinguishing A from F is that F grows monotonically while assumptions may be added or removed from A at any time.

Definition 2.2. A *labeling* λ is a mapping from S to the set {**T**, **F**, **U**} (**T** for true, **F** for false, and **U** for unknown). If for all σ in S, $\lambda(\sigma)\neq$**U** then λ is *complete*; otherwise λ is *partial*. A *completion* of a partial labeling is a complete labeling which relabels all the **U** symbols **T** or **F**. The label of the literal σ is the label of the symbol σ and the label of the literal $\neg\sigma$ is **F**, **T** or **U** depending on whether the label of σ is **T**, **F** or **U**. The label **T** or **F** for a literal is also called the **truth value** of the literal. For any labeling λ, literal l, we define $\lambda[l\leftarrow v]$ as the labeling which is identical to λ except that it assigns the literal l the truth value v.

Since in an LTMS the set F can contain any propositional formula, determining the satisfiability of $F \cup A$ or discovering if a label for a symbol logically follows from $F \cup A$ will be NP-complete tasks. So, LTMSs are usually implemented with the useful **Boolean Constraint Propagation** algorithm (**BCP**) to reduce the cost of deductions updating. BCP is sound but incomplete. Propositional formulae of F are called now *constraints*.

For BCP, given any labeling λ, each constraint is in one of 4 possible states:

- the constraint is *satisfied*: for every completion of λ the constraint is true. For example, if $\lambda(x)$=**T**, then the constraint $x \vee y$ is satisfied.
- the constraint is *violated*: there is no completion of λ which satisfies the constraint. For example, if $\lambda(x)$=**T**, then the constraint $\neg x \wedge y$ is violated.
- the constraint *forces* a symbol's label: for every completion of λ which makes the constraint true, the symbol is always labeled **T** or always **F**. There may be multiple such symbols. For example, if $\lambda(x)$=**T**, then the constraint $(x \vee y) \rightarrow z$ forces z to be labeled **T**.
- otherwise the constraint is *open*.

BCP labels **T**, **F** or **U** every symbol of S. BCP processes the initial set of assumption literals A; if $\sigma \in A$ then σ is labeled **T**, and if $\neg\sigma \in A$ then σ is labeled **F**. All remaining symbols are initially labeled **U**. BCP processes the constraints one at a time monotonically expanding the current labeling λ:

Algorithm BCP($F \cup A, \lambda$)

1. if $F \cup A$ contains any violated constraint then **exit**.

2. if $F \cup A$ contains any constraint that forces a label for a symbol, then select such a constraint C, select a symbol σ that C forces to the truth value v and **return** BCP($F \cup A, \lambda[\sigma\leftarrow v]$).

3. otherwise **return** the labeling λ.

The problem of this algorithm is that determining whether a constraint is violated or whether a constraint forces a symbol's label are NP-complete tasks (since constraints are arbitrary formulae). So, each of these steps in BCP is exponential in the size of the constraint considered.

2.2. Clause-based TMS (CTMS)

BCP on arbitrary formulae is not very efficient since it is exponential in the size of the formulae. So, in practice, LTMSs are often restricted to **clause-based TMSs** (abbreviated **CTMSs**).

Definition 2.3. A CTMS involves a set of propositional symbols S, a set of *clauses* F_C instead of a set of arbitrary formulae F, and a set of assumption literals A. A clause is a disjunction of literals.

The algorithm BCP is still available in CTMSs. But tasks that were expensive on arbitrary formulae become straightforward on clauses.

Given a labeling λ, a constraint $C \in F_C$ is:

- *Satisfied* if some literal of C is labeled **T**.

• *Violated* if every literal of C is labeled **F**.

• *Unit open* if one literal is labeled **U** and the remainder **F** (i.e. C forces the label **T** for this literal).

• *Open* if more than one literal is labeled **U** and the remainder **F**.

Therefore, BCP on clauses (which is similar to unit resolution) is very efficient. Even finding the justification for a deduced literal is done by simply taking the other literals of the clause where the deduction has been made (all of them must be labeled **F** to deduce the label **T** for the last one). So, it is sufficient to record the clause as the justification of the deduction. If we denote $|C|$ the number of literals in a clause and $|F_c|$ the sum of the $|C_i|$ for all $C_i \in F_c$ then the complexity of BCP on $F_c \cup A$ is $O(|F_c|+|A|)$.

Remark. JTMSs are particular CTMSs. BCP detects all contradictions in JTMSs.

2.3. McAllester's TMS (MTMS)

CTMSs implemented with BCP are very efficient but have a limited expressive power compared to general LTMSs. Hence, some authors (McAllester in [4], Forbus and deKleer in [1]) have proposed TMSs allowing any propositional formulae (in what they look like general LTMSs) but translating these arbitrary formulae in clauses before running BCP (in what they work like CTMSs). The difference between McAllester's TMS and Forbus's and deKleer's one is that in [1] formulae are directly translated into **Conjonctive Normal Form** (**CNF**), when in [4], formulae are decomposed into ternary formulae with the help of new symbols before translating them into CNF.

In this section, we try to give a formal definition of the TMS presented by McAllester and denoted **MTMS** in the rest of this paper.

Definition 2.4. A *well-formed formula (wff)* is a formula or the negation of a formula involving a binary connective ($\vee$, $\wedge$, $\rightarrow$ or $\equiv$) and two terms that are literals or wffs:
Let f be a formula, f is a wff iff

• $f = (f_1 \vee f_2)$, or $f = (f_1 \wedge f_2)$, or $f = (f_1 \rightarrow f_2)$, or $f = (f_1 \equiv f_2)$ with f_1, f_2 wffs or literals (f is a *positive* wff)

• or $f = \neg f_1$ with f_1 a wff.

The main idea in a MTMS is the decomposition of any formula given as an assumption (or assertion) to the MTMS, in a set of ternary formulae. Each positive wff included in the formula given to the MTMS is associated to a new symbol by the connective $\equiv$. For example, the formula $(\neg(x \vee y) \rightarrow z))$ will be decomposed in the set of ternary formulae $\{\phi_2 \equiv (x \vee y)) \ ; \ \phi_1 \equiv (\neg \phi_2 \rightarrow z))\}$, ϕ_1 and ϕ_2 being two new symbols. ϕ_1 represents the formula $(\neg(x \vee y) \rightarrow z))$.

Definition 2.5. A MTMS involves in an external level, a set of propositional symbols S_s and a set of *assumptions* F_a which are visible to the user. Any literal or wff can be an assumption.

In an internal level, the kernel, not visible to the user, it involves another set S_{wff} of symbols (in addition to S_s), a set of clauses F_c and a set of assumption literals A.

• There is a one to one mapping between S_{wff} and the set of the positive wffs which appear in formulae of F_a[1]. Each symbol ϕ in S_{wff} is associated to a positive wff $f=(f_1 * f_2)$ by the ternary formula $\phi \equiv (l_1 * l_2)$ where l_1 and l_2 are the literals associated to the wffs f_1 and f_2 (with $* \in \{\vee, \wedge, \rightarrow, \equiv\}$)[2].

• These ternary formulae are encoded in F_c by their CNF.

• Every literal in F_a or associated to a wff element of F_a is put in A.

Property 2.6. In a MTMS, $F_c \cup A$ and F_a are proportional in size.

Property 2.7. In a MTMS, $F_c \cup A$ and F_a are logically equivalent (i.e. the labelings satisfying $F_c \cup A$ restricted to the symbols in S_s are the labelings satisfying F_a).

Example. Suppose the assumption formula $(\neg(x \vee y) \rightarrow z)$ arrives in F_a, x, y and z being symbols of S_s. The symbol associated to the formula must be put in A. If the formula was not already known, a symbol ϕ_1 is created in S_{wff}. $(x \vee y)$ is a positive wff too, for which a symbol ϕ_2 is created in S_{wff}. ϕ_2 and ϕ_1 are associated to the formulae by the connective $\equiv$ CNF($\phi_2 \equiv (x \vee y)$) and CNF($\phi_1 \equiv (\neg \phi_2 \rightarrow z)$) are added to F_c. Now, ϕ_1 can be added to A. Suppose that the formula $(\neg(x \vee y) \vee z)$ arrives now in F_a. A symbol ϕ_3 must be associated to the formula and put in A. The positive wff $(x \vee y)$ which appear in the formula was already known, so we need only to put CNF($\phi_3 \equiv (\neg \phi_2 \vee z)$) in F_c and ϕ_3 in A. Now, if the assumption $(\neg(x \vee y) \rightarrow z)$ is removed from F_a, we simply retract ϕ_1 from A.

---

[1] The number of symbols in S_{wff} is linear in $|F_a|$, the total size of the formulae in the MTMS.

[2] A literal l is associated to a wff f if $l=\sigma$, where σ is the symbol associated to f which is a positive wff, or if $l=\neg\sigma$ and σ is associated to $\neg f$, f being a negative wff.

A MTMS seen from the outside is powerful enough to encode any propositional formula, but inside is built like a CTMS and so is very efficient. What are the drawbacks ? When a formula is encoded in CNF it is decomposed in several clauses. Now, BCP is inherently local, considering only one propositional formula at a time. So, BCP is complete when applied to one formula, but not when applied to several formulae because of its locality. For example, BCP applied to the formula $((p \rightarrow q) \wedge (p \vee q))$ deduces q, but applied to the set of clauses $\{(\neg p \vee q); (p \vee q)\}$ it does not deduce anything. So, BCP applied to the CNF of a formula is less powerful than applied to the original formula. The clauses of the CNF of a formula taken one after one forget local properties of the formula. The solution given by de Kleer in [2] was to decompose formulae not into CNF but into prime implicates. BCP applied to the set of the prime implicates of a set of formulae is complete. Since there can be an exponential number of prime implicates for a set of formulae, deKleer proposed to build prime implicates only for formulae taken individually. Running BCP on the prime implicates of the individual formulae is the same as running BCP on the original formulae and he can exploit the efficiency of BCP on clauses. But computing prime implicates of a formula can be exponential in the size of the formula.

3. Constraint networks basic definitions

3.1. Constraint networks

A **static** constraint network (CN) (X, dom, c) involves a set of **variables**, $X = \{i, j, ...\}$, each taking **value** in its respective **domain**, $dom(i)$, $dom(j)$,..., and a set of **constraints** c. Each constraint $C(i_1,...,i_q)$ constraining the subset $\{i_1,...,i_q\}$ of X, is a set of tuples, subset of the Cartesian product $dom(i_1)$ x...x $dom(i_q)$, that specifies which values of the variables are compatible with each other.

A **solution** of a CN is an assignment of values to all the variables such that all the constraints are satisfied.

3.2. Dynamic constraint networks

A **dynamic** constraint network (DCN) p is a sequence of static CNs $p_{(0)},..., p_{(\alpha)}, p_{(\alpha+1)},...$, each resulting from a change in the preceding one imposed by "the outside world". This change can be a **restriction** (a new constraint is imposed on a subset of variables) or a **relaxation** (a constraint which was present in the CN is removed).

So, if we have $p_{(\alpha)}=(X_{(\alpha)}, dom_{(\alpha)}, c_{(\alpha)})$, we will have $p_{(\alpha+1)}=(X_{(\alpha+1)}, dom_{(\alpha+1)}, c_{(\alpha+1)})$ where $c_{(\alpha+1)}=c_{(\alpha)} \pm C(i_1,...,i_q)$ and $X_{(\alpha+1)}=X_{(\alpha)} \cup \{i_1,...,i_q\}$, $C(i_1,...,i_q)$ being a constraint.

3.3. Local consistency properties

Since the task of finding a solution in a CN is NP-complete, local consistency algorithms were proposed by many authors ([5], [6], [7]). These algorithms do not solve a CN completely but they eliminate once and for all local inconsistencies that cannot participate in any solutions.

The most widely used consistency algorithms are those achieving **arc-consistency**. Arc-consistency checks the consistency of values for each pair of variables linked by a constraint and removes the values that cannot satisfy this local condition.

Definition 3.1. A CN is arc-consistent iff for all i in X, $dom(i) \neq \varnothing$ and for all v_i in $dom(i)$, for all $C(i_1,..., i_q)$ in c with i in $\{i_1,..., i_q\}$, there exists $v_{i1}...v_{iq}$ in $dom(i_1)...dom(i_q)$ such that $(v_{i1}...v_i...v_{iq})$ is a tuple in C.

Arc-consistency is very simple to implement and has a good efficiency. Its complexity is $O(K)$ with K the sum of the lengths of all admissible tuples for all constraints in c [8].

Pairwise-consistency is another local consistency property which checks the consistency of tuples for pairs of constraints involving common variables. Pairwise-consistency has first been introduced in [9] on database schemes and has been applied to CNs in [10].

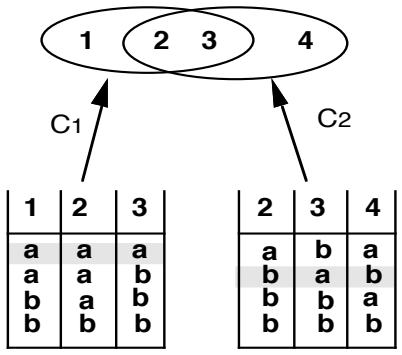

Figure 1. This CN is not pairewise-consistent. We have to discard the tuple (*a*, *a*, *a*) from the constraint $C_1(1, 2, 3)$ and the tuple (*b*, *a*, *b*) from the constraint $C_2(2, 3, 4)$

Definition 3.2. A CN is pairwise-consistent iff for all $C(i_1,...,i_q)$ in c, $C(i_1,...,i_q) \neq \varnothing$ and for all $C(i_1,...,i_q)$ in c, for all $C'(j_1,..., j_p)$ in c, with $I=\{i_1,...,i_q\} \cap \{j_1,..., j_p\} \neq \varnothing$, we have $C[\,I\,]=C'[\,I\,]$, where $C[\,I\,]$ is the projection of the tuples of C on I.

4. Encoding a TMS in dynamic CN

We have seen in section 2 that in a TMS we have to choose between expressive power and efficiency. If we choose to keep both of them like McAllester, or Forbus and de Kleer, we lose completeness. The compromise made by de Kleer when computing prime implicates, is to lose a bit of efficiency to earn local completeness. This approach is very similar to consistency algorithms used in CNs.

So, we propose a representation of McAllester's TMS in dynamic CN (called **coupled-MTMS** in the following) to be able to use CNs resolution techniques.

4.1. The transformation

Definition 4.1. A coupled-MTMS is a MTMS in which the kernel is represented by the dynamic CN $(s=s_S \cup s_{wff}, \{\mathbf{T}, \mathbf{F}\}^S, f_C \cup a)$. The sets of symbols S_S and S_{wff} are replaced by the sets of bi-valued variables s_S and s_{wff}, the set of clauses F_C is replaced by a set of ternary constraints f_C and the set of assumption literals A by the set of unary constraints a. Every time CNF($\phi \equiv (l_1 * l_2)$) was added to F_C in the MTMS's kernel (with $* \in \{\rightarrow, \vee, \wedge, \equiv\}$), the ternary constraint *CNconstraint*($\phi \equiv (l_1 * l_2)$) is added to f_C in the kernel of the coupled-MTMS[3]. Every time a literal σ (resp. $\neg\sigma$) was added or retracted from A, then the unary constraint $V_\sigma = \mathbf{T}$ (resp. $V_\sigma = \mathbf{F}$) is added or retracted from a, V_σ being the variable in s corresponding to the symbol σ.

Remark. The domain of every variable of s_S or s_{wff} is $\{\mathbf{T}, \mathbf{F}\}$. So, if V_σ is the variable in the CN representation of the kernel corresponding to the symbol σ in the MTMS, we can define the labeling λ in the MTMS corresponding to the domain *dom* in the CN representation of the kernel, by:

$dom(V_\sigma)=\{\mathbf{T}\} \Leftrightarrow \lambda(\sigma)=\mathbf{T}$
$dom(V_\sigma)=\{\mathbf{F}\} \Leftrightarrow \lambda(\sigma)=\mathbf{F}$
$dom(V_\sigma)=\{\mathbf{T}, \mathbf{F}\} \Leftrightarrow \lambda(\sigma)=\mathbf{U}$

4.2. Properties of the CN representation

The proofs of all properties and theorems which follow are given in the full paper [11].

Property 4.2. For all wff f, its representation in a MTMS or in a coupled-MTMS are logically equivalent and proportional in size.

Theorem 4.3. BCP in a MTMS or an algorithm that achieve arc-consistency in a coupled-MTMS produce exactly the same deductions.

Property 4.4. Using the incremental algorithm DnGAC4 ([12], [13]) to achieve arc-consistency in a coupled-MTMS or using BCP in a MTMS give proportional time consuming for any addition or retraction of assumption in F_a. More, with DnGAC4, finding the justification of a deduced label for a symbol is as simple as in BCP on clauses.

We have defined a representation of a MTMS in terms of CNs (the coupled-MTMS) without any loss of efficiency. All the tasks processed by a MTMS are still available in a coupled-MTMS, and for the same cost.

4.3. Enhancing completeness

Having a representation of a MTMS in terms of CNs without loss of expressive power or efficiency, we can look for using CN techniques to achieve a local consistency stronger than arc-consistency without losing too much efficiency. Pairwise-consistency is a consistency property a "little stronger" than arc-consistency. Arc-consistency was complete for an individual constraint, pairwise-consistency is complete for a pair of constraints. Used in a coupled-MTMS it seems that pairwise-consistency provides a good improvement in the number of deductions without involving a big loss of efficiency. The complexity of pairwise-consistency in a coupled-MTMS is $O(m \cdot log(m))$ with m the number of CN constraints[4]. By properties 2.6. and 4.2., m is proportional to $|F_a|$, the sum of the sizes of the formulae in the coupled-MTMS. So, the complexity of pairwise-consistency in a coupled-MTMS is $O(|F_a| \cdot log|F_a|)$.

Example. Let the two formulae $f_1=(\neg(x \vee y) \rightarrow z)$ and $f_2=(\neg(x \vee y) \vee z)$. BCP on the CNF of the two formulae in Forbus's and de Kleer's TMS or BCP in a MTMS with $F_a=\{f_1, f_2\}$ as input does not deduce anything for z.

But let see what happens in a coupled-MTMS achieving arc plus pairwise consistencies.

After the addition of f_1 and f_2 in F_a, we have:
$s_{wff}=\{\phi_1, \phi_2, \phi_3\}$, three new symbols created to be associated to the positive formulae in F_a (see the example of section 2.3.).

---

[3] The function *CNconstraint*($\phi \equiv (l_1 * l_2)$) returns in extension the definition of the constraint, i.e. the disjunctive normal form of the formula $\phi \equiv (l_1 * l_2)$. In our case, all the constraints will have four tuples.

[4] We obtain a so good complexity because there are no constraints larger than ternary. In this case, it is sufficient to create a variable for each couple of variables in a constraint, to add the correct values to these variables domains, to extend the existent constraints to these variables and to acheive arc-consistency; this provides the same result as pairewise-consistency on the original CN. The logarithm comes from an existence test when we create new variables.

f_C={*CNconstraint*($\phi_2 \equiv (x \vee y)$), *CNconstraint*($\phi_1 \equiv (\neg\phi_2 \rightarrow z)$), *CNconstraint*($\phi_3 \equiv (\neg\phi_2 \vee z)$)}
a={ϕ_1=**T**, ϕ_3=**T**}

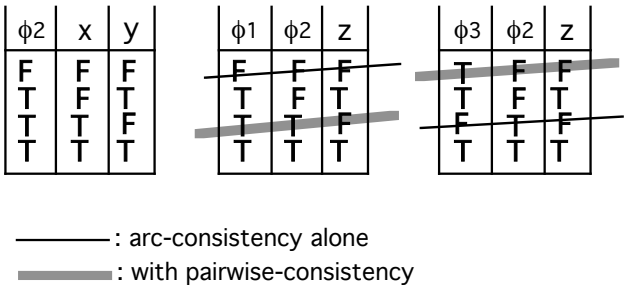

Figure 2. The ternary constraints in the coupled-MTMS and the deduction processes

Pairwise-consistency plus arc-consistency (see fig. 2) deduce the label **T** for z from the formulae f_1 and f_2. We can remark that arc-consistency alone (like BCP) does not deduce z's value.

4.4. Experimental results

We have compared the number of deductions made by BCP in a MTMS and by pairewise-consistency in a coupled-MTMS on randomly generated sets of clauses. We have made these tests on 3 classes of problems, with ten instances in each class (we give only the average of the ten instances).

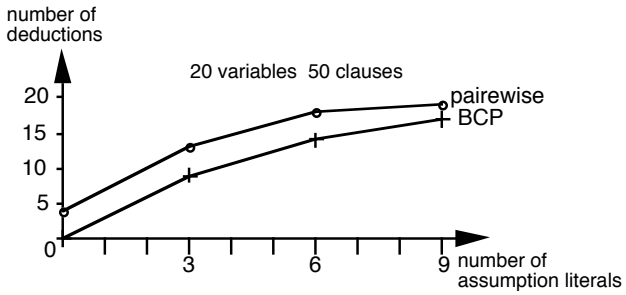

Figure 3. Comparison of BCP and pairewise-consistency deductive power on problems with 20 variables

The problems have a fixed number of variables and a fixed number of binary and ternary clauses. We compare the number of deductions made by each algorithm when the number of unary clauses added (assumption literals) increases. We have generated only satisfiable problems.

Seeing figures 3 and 4, the first remark, evident, is that BCP cannot deduce anything if there does not exist unary clauses at the begining of the deduction process. BCP needs unit open clauses to deduce values. Pairewise-consistency can deduce values when there are no unit open clauses.

The number of deductions made by pairewise-consistency is almost always more than 40% better than BCP on the same problem. The difference between the two algorithms is the largest when there are more non-unary clauses and less assumption literals. This is what is wanted in a problem solver: adding less assumptions and having more informations.

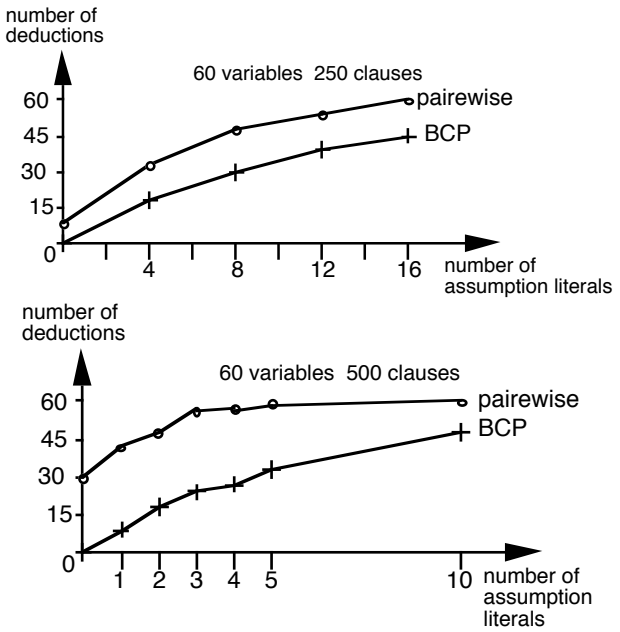

Figure 4. Comparison of BCP and pairewise-consistency deductive power on problems with 60 variables

On all the classes tested, BCP needs more than twice as many assumption literals as pairewise-consistency to deduce the same number of values. It means, for example, that contradictions will be discovered twice later (in number of assumptions) than with pairewise-consistency.

On all these tests, time-consuming of pairewise-consistency remains between 1.8 and 2.7 times as much as BCP's.

5. Conclusion

In this paper, we have presented the main classes of TMSs and focused on one, interesting when a large expressive power is needed without losing too much efficiency. We have shown its drawbacks and recalled one solution given by deKleer to reach stronger completeness with prime implicates. In section 4, we have introduced a new approach, encoding a TMS into a Dynamic CN to apply CNs techniques. We have presented CNs local consistency techniques that in essence, look like some techniques used in TMSs. So, the comparison was easy. The first one (arc-consistency) is as powerful as BCP the main used algorithm on TMSs. The second one (pairwise-

consistency) outperforms BCP for a slight more running time. Experimental results show that pairewise-consistency is an interesting method to maintain a local completeness in a TMS.

References

[1] K. Forbus, J. de Kleer: "*Truth Maintenance Systems*"; Tutorial in Proceedings AAAI-91, Anaheim CA

[2] J. de Kleer: "*Exploiting locality in a TMS*"; Proceedings AAAI-90, Boston MA, 264-271

[3] R. Zabih, D. McAllester: "*A Rearrangement Search Strategy for Determining Propositional Satisfiability*" ; Proceedings AAAI-88, St-Paul MN, 155-160

[4] D.A. McAllester: "*An Outlook on Truth Maintenance*"; MIT, Boston MA, Tech.Rep. AI Memo No.551, 1980

[5] U. Montanari: "*Networks of Constraints: Fundamental Properties and Applications to Picture Processing*"; Information Science 7, 95-132 (1974)

[6] A.K. Mackworth: "*Consistency in Networks of Relations*"; Artificial Intelligence 8 (1977) 99-118

[7] E.C. Freuder: "*Synthesizing Constraint Expressions*"; Communications of the ACM, Nov. 1978, Vol.21 No.11

[8] R. Mohr, G. Masini: "*Good Old Discrete Relaxation*" ; Proceedings ECAI-88, München, FRG, 651-656

[9] C. Beeri, R. Fagin, D. Maier, M. Yannakakis: "*On the desirability of acyclic database schemes*"; Journal of the ACM Vol.30, 1983, 479-513

[10] P. Janssen, P. Jégou, B. Nouguier, M.C. Vilarem: "*A Filtering Process for General Constraint-Satisfaction Problems: Achieving Pairewise-Consistency Using an Associated Binary Representation*"; Proceedings of the IEEE Workshop on Tools for Artificial Intelligence (TAI), Fairfax VA, 1989, 420-427

[11] C. Bessière: "*Using CSPs to encode TMSs*"; Tech. Rep. n° 92-001, LIRMM Montpellier II, France, January 1992

[12] C. Bessière: "*Arc-Consistency in Dynamic Constraint Satisfaction Problems*"; Proceedings AAAI-91, Anaheim CA, 221-226

[13] C. Bessière: "*Arc-Consistency for Non-Binary Dynamic CSPs*"; Proceedings ECAI-92, Vienna, Austria, 23-27