Discrete Applied Mathematics 107 (2000) 27–40

algorithms

Richard Génisson, Philippe Jégou *

LIM-ESA CNRS 6077, CMI-Université de Provence, 39, rue Jolliot Curie-13453 Marseille Cedex 13, France

Received 14 March 1997; revised 7 April 1999; accepted 21 June 1999

Abstract

We show the equivalence between the so-called Davis–Putnam procedure (Davis et al., Comm. ACM 5 (1962) 394–397; Davis and Putnam (J. ACM 7 (1960) 201–215)) and the Forward Checking of Haralick and Elliot (Artificial Intelligence 14 (1980) 263–313). Both apply the paradigm *choose and propagate* in two different formalisms, namely the propositional calculus and the constraint satisfaction problems formalism. They happen to be strictly equivalent as soon as a compatible instantiation order is chosen. This equivalence is shown considering the resolution of the clausal expression of a CSP by the Davis–Putnam procedure. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Proportional calculus; Constraint satisfaction; Algorithms; Complexity

1. Introduction

A large number of problems in AI and computer science can be viewed as special cases of the constraint satisfaction problem. Some examples are machine vision, belief maintenance, scheduling, temporal reasoning, graph problems, circuits verification, etc. The satisfiability problem also allows to express and solve these problems in a bit more rough manner. Their two respective formalisms gave arise to two procedures being considered as the best complete methods to solve problems (see for example [7,13]). They are, respectively, the Davis–Putnam procedure for SAT [4] and [3] and the Forward Checking procedure for CSPs [9]. The last being the heart and soul of several provers in constraint logic programming languages (see for example [6]).

Our goal here is to show that these two procedures are equivalent, and so that, in some sense, Forward Checking brings nothing new although it came 20 years after the Davis–Putnam procedure. This is mainly an epistemological issue.

* Corresponding author.
E-mail address: jegou@lim.univ-mrs.fr (P. Jégou)

On the other hand, there might be a practical interest in this investigation. Knowing that a problem is the clausal expression of a CSP could lead to develop some specific DP-like procedures much more efficient than DP itself. This could be done by improving the domain shortening after a choice is made and by performing an FC-like domain exploration.

To prove the equivalence of the two procedures we shall first recall in Section 2.1 how a CSP can be expressed as a SAT problem, owing to de Kleer's translation [5]. In Section 3, we will discuss two other procedures, the Quine algorithm of propositional calculus [11], and the Backtrack algorithm of CSPs. These procedures can be seen, respectively, as the Davis–Putnam procedure and the Forward Checking procedure with a trivial consistency checking (i.e. without any propagation). Once we have studied their respective behaviour, introducing the propagation in both cases will lead us to claim the equivalence of DP and FC in Section 4.

2. Preliminaries

In the sequel, we shall assume the reader familiar with both formalisms. However, for sake of readability, we have to fix some notations that will be used further.

A CSP involves a set of n variables $\{x_1,\dots,x_n\}$ taking their values in as many finite domains $\{D_1,\dots,D_n\}$. The problem is to decide whether these variables in their respective domains can satisfy a set of m constraints $\{C_1,\dots,C_m\}$ where each constraint is a subset of variables $C_i=\{x_{i1},\dots,x_{in_i}\}$. To satisfy a constraint, one has to find a value in the domain of each concerned variable to satisfy the relation associated to the constraint. We note the set of relations $R=\{R_1,\dots,R_m\}$ and consider they are given as sets of allowed tuples that are of course included in the cartesian product of the associated variables domains. The binary constraint $\{x_i,x_j\}$ will be denoted C_{ij} and its associated relation R_{ij}. That problem is NP-complete, even if all the constraints are binary (on two variables). For the purpose of this paper, we shall only consider binary CSPs.

The Constraint Satisfaction Problem came as a generalization of the SAT problem. The SAT problem is to decide whether a formula in propositional calculus has solutions or not. This can be formulated as the problem of the existence of an assignment of the variables in $\{1,0\}$ which makes the formula evaluate to 1. Without loss of generality, we shall only consider CNF formulas. They are a conjunction of disjunctions of literals called *clauses*. A literal is either a propositional variable or its complement. Indeed, this problem is equivalent to a CSP in which all the variables take their values in the domain $\{1,0\}$. There are as many constraints as clauses. The relations can be seen as the allowed tuples by each clause.

2.1. Writing a CSP as SAT

We first recall how any CSP can be considered as a SAT problem, then relate the CSP variables instantiations with propositional assignments and finally introduce the notion of compatible instantiation orders.

In [5], de Kleer introduced an algorithm to express any CSP as a set of propositional clauses. We briefly recall it here.

For each variable x_i we introduce a boolean variable for each value $a_{ij} \in D_i$. The propositional symbol $x_i a_{ij}$ stands for the assertion $x_i = a_{ij}$. For each variable x_i we write the clause expressing that it has to take a value in its domain: $x_i a_{i1} \vee x_i a_{i2} \vee \cdots \vee x_i a_{id_i}$.

We also express that a variable can take only one value in its domain. For each x_i, for each a_{ij}, for each a_{ik} such that $j \neq k$, we add the following negative binary clause: $\neg x_i a_{ij} \vee \neg x_i a_{ik}$.

Each constraint is coded in the following manner. For a constraint C_{ij}, the set of clauses representing it is: $\{\neg x_i a \vee \neg x_j b$: $(a,b) \in (D_i \times D_j) \setminus R_{ij}\}$. If $|R_{ij}|$ is the size of the relation then $|D_i \times D_j| - |R_{ij}|$ is the size of its propositional coding.

If we consider that the size of the CSP is $m.d^2$ (where m is the number of constraints and d the maximum size of a domain) as indicated by Montanari in his historical paper [10], the cost of this transformation is then linear. Note also that, as we shall see further, any instantiation of the CSP variables gives a truth assignment of the boolean variables, but that the converse is false as indicated at the end of Section 2.2. However, these two problems have of course the same sets of solutions.

As a matter of fact, de Kleer did not even mention that there could be strong similarities between the Davis and Putnam's procedure and the Forward Checking procedure.

2.2. Propositional assignments and variables instantiations

Let us first remark that given a CSP and its clausal expression S, one can easily associate to each partial assignment of the CSP variables an unique truth assignment of the boolean variables of S.

Definition 1. The partial interpretation I associated to a partial instantiation $(v_1, v_2, \ldots, v_i)$ of some CSP variables on the domains $D_1, D_2, \ldots, D_i$ is defined by: $\forall j, 1 \leqslant j \leqslant i$, $I[x_j v_j] = 1$ and $\forall v \in D_j$ such that $v \neq v_j$, $I[x_j v] = 0$.

Conversely, it is not possible to associate a CSP instantiation to any propositional assignment. For example, no assignment such that $\exists v_1, v_2 \in D_j$ with $I[x_j v_1] = 1$ and $I[x_j v_2] = 1$ can be associated to a CSP instantiation since it would force a single variable to take two values in its domain, namely v_1 and v_2.

2.3. Compatible orders

The comparison of the Quine algorithm and the Backtrack algorithm requires to relate the respective instantiation and assignment orders. We shall see later that this relation induces some conditions on the the domains exploration which brings no loss of generality. These conditions require to interpret consecutively the boolean variables coding a same domain D_i, e.g. in an order like $x_i v_1, x_i v_2, \ldots, x_i v_d$.

Definition 2. An interpretation order I is called compatible with a CSP instantiation if and only if, $\forall i,j$: $1 \leqslant i < j \leqslant n$, $\forall a,b$: $v_a \in D_i, v_b \in D_j$, then $x_i v_a$ precedes $x_j v_b$ in the order.

Considering a compatible order will lead the d propositions coding the d values of a domain D_i to be consecutively interpreted. Note that a compatible order can be found by a basic first-fail principle that consists in choosing to evaluate a literal in the shortest shortened clause. This clause will, in any case, be a domain clause, one can easily get convinced of this fact. Once a literal is chosen (and so a domain clause), suppose that the exploration begins on the sub-tree corresponding to its assignment to 1. So, each negative exclusion clause of this domain will be reduced to unitary negative clauses. So, after the exploration of this sub-tree, the exploration is going on the assignment to 0. This assignment to 0 will cause the corresponding domain clause to be again the shortest shortened one seeing that it is the unique clause which has been reduced since this domain has been choosed (no other domain clause has been reduce).

3. From the Quine algorithm to the CSP enumeration

As a preliminary to the comparative study of DP and FC, we first establish the strong similarities between two other procedures: the Quine algorithm of propositional calculus and the backtrack algorithm of CSPs.

3.1. The Quine algorithm

The version of the Quine algorithm we give here takes as parameters a set of clauses S and a partial assignment I. The function *Quine* calls the function *Wash*(S, p) which result is a set of clauses S' built from S by the suppression of the clauses containing p and the reduction of those where $\neg p$ occurs.

```
function Quine(S,I):boolean;
begin
  if $S = \emptyset$ then Quine:=true {I is a model}
  else if ($S$ contains an empty clause) then Quine:=false
  else begin
    Choice of a boolean variable p occurring in S;
    Quine:=Quine(Wash(S,p),I $\cup$ {I[p] = 1})$\vee$ Quine(Wash(S,$\neg$ p),I $\cup$ {I[p] = 0})
  end
end;
```

Note that the instruction *Choice of a boolean variable p occurring in S* must satisfy the compatible order. This algorithms explores a tree the nodes of which are labelled by the current set of clauses S and the arcs of which, issued from a same node are labelled either by a boolean variable p, for its assignment to 1, or by its negation $\neg p$ for the assignment of p to 0. To each path issued from the root corresponds a partial assignment of the boolean variables.

3.2. The Backtrack algorithm in CSPs

The version of the Backtrack algorithm we give here takes as parameters a CSP P and a partial instantiation of the variables I. The CSP considered at each node of the tree is supposed to be "washed" by I, that is the domains of the instantiated variables were reduced to the fixed value whereas the relations were restricted to the remaining values in the domains. Let us consider for example a partial instantiation I of the form $\{I[x_1]=v_1,\ldots,I[x_i]=v_i\}$. We have then as current CSP, if $1 \leqslant j,j' \leqslant i < k,k' \leqslant n$:

- $D'_j=\{v_j\}$
- $D'_k=D_k$
- $R'_{jj'}=\{(I[x_j],I[x_{j'}])\}\cap R_{jj'}$
- $R'_{jk}=\{(v_j,v_k)\in R_{jk} : v_j=I[x_j]\}$
- $R'_{kk'}=R_{kk'}$

Saying that I is compatible with P simply means that no relation is empty. The algorithm can be sketched as follows:

```
function Backtrack(P,I):boolean;
begin
   if (no relation is empty)∧(|I|=n) then Backtrack:=true {I is a solution}
   else if (a relation is empty) then Backtrack:=false
   else begin
      Choice of $x_i$ not yet instantiated;
      Backtrack:=false;
      for $v_i \in D_i$ do Backtrack:=Backtrack ∨ Backtrack(Washing(P,$v_i$),I ∪{I[$x_i$] = $v_i$})
   end
end;
```

The function *Backtrack* calls the function *Washing*(P,v_i) the result of which is a CSP washed as indicated earlier. This washing turns to be a simple consistency test, as it allows to verify that the last instantiation satisfied the involved constraints.

As for the Quine function, the Backtrack algorithm develops a semantic tree. Its nodes correspond to the current problem, e.g. the one induced by the instantiation associated to the labelling of the arcs of the path from the root to the considered node.

3.3. The Quine semantic tree on the clausal expression of a CSP

The purpose of this section is to relate the semantic trees developed by the Quine and Backtrack algorithms. We will show here that the dead ends of the Quine algorithm are of two types. They are either a contradiction of the clauses coding the domains, or a violation of the constraints of the CSP. In that case, we will see that the dead ends corresponding to some constraints violation occur at the same depth in both algorithms. In the following, we will always refer to an order supposed compatible with a CSP instantiation. A first property gives us already the possible dead ends in the tree developed by the Quine algorithm. To simplify, we shall consider that $\forall i,\ |D_i| = d$.

Property 1. *Let S be the current set of clauses supposed not to contain the empty clause, I the current partial assignment, and x_iv_j the next variable to interpret, e.g. the variable representing the jth value of the ith CSP variable. The empty clause can be produced by the function Wash in three cases*:

- $\forall k, 1 \leqslant k < j, I[x_iv_k] = 0, I[x_iv_j] = 0$ *and* $j = d$.
- $\exists k, 1 \leqslant k < j$ *such that* $I[x_iv_k] = 1$ *and* $I[x_iv_j] = 1$.
- $\forall k, 1 \leqslant k < j, I[x_iv_k] = 0,\ I[x_iv_j] = 1$ *and the unit clause* $\neg x_iv_j$ *is present.*

Proof. The proof of this property being as simple as fastidious, this proof can be left to the reader, we just give here remarks about the three assertions (the interested reader can refer to [8]). Without loss of generality, suppose that the current partial assignment I relates the varibles $x_1v_1, x_1v_2, \ldots, x_1v_d, x_2v_1, \ldots, x_2v_d, \ldots, x_iv_1, \ldots, x_iv_{j-1}$, the current state of S being then a function of I. Actually, the clauses coding the domains D_k, for $1 \leqslant k < i$, are not present in S anymore since all their literals are now interpreted (and none of these interpretations leaded to the production of the empty clause). We have so for all k such that $1 \leqslant k < i$, one and only one x_kv such that $I[x_kv] = 1$. Concerning the boolean variables coding the current domain D_i, there is at most one positive interpretation among $I[x_iv_1], I[x_iv_2], \ldots, I[x_iv_{j-1}]$. So, we comment the three cases given in the property:

- $\forall k, 1 \leqslant k < j, I[x_iv_k] = 0, I[x_iv_j] = 0$ and $j = d$. This case relates the fact that no value from the domain D_i is assigned to the CSP variable x_i. So, the positive domain clause $x_iv_1 \vee x_iv_2 \vee x_iv_d$ will be reduced to the empty clause.
- $\exists k, 1 \leqslant k < j$ such that $I[x_iv_k] = 1$ and $I[x_iv_j] = 1$. This case relates the fact that two values from the domain D_i are assigned to the CSP variable x_i. So, the negative domain clause $\neg x_iv_k \vee \neg x_iv_j$ will be reduced to the empty clause.
- $\forall k, 1 \leqslant k < j, I[x_iv_k] = 0,\ I[x_iv_j] = 1$ and the unit clause $\neg x_iv_j$ is present. This case relates the fact that the domain D_i is properly assigned but since the unit clause $\neg x_iv_j$ is present, we know that a constraint clause of the form $\neg x_\alpha v \vee \neg x_iv_j$ with $1 \leqslant \alpha < i$ has been reduced by the assignment $I[x_\alpha v] = 1$. So, the constraint between the CSP variables x_α and x_i is now violated, and then the empty clause will be produced. □

The following property can then be deduced from this first result. It involves a current problem P, an instantiation I_{B} associated to a Backtrack resolution, a set of clauses S corresponding to P and the interpretation I_{Q} associated to I_{B} by a resolution of the clausal expression of P by the Quine algorithm:

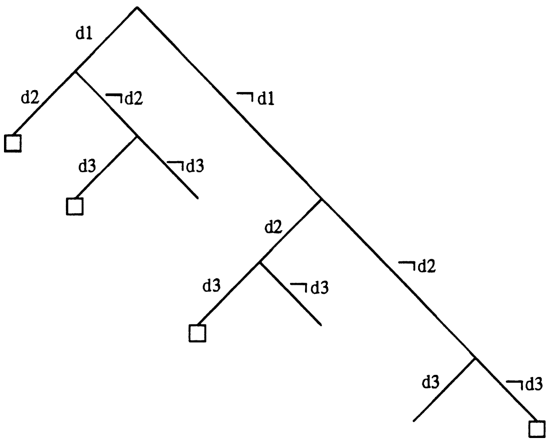

Fig. 1. Examination of a domain of size three by the Quine algorithm.

Property 2. *A constraint of the problem P is violated by the last assignment in* I_{B} *if and only if the empty clause is produced by the shortening of the clauses representing this constraint. This shortening comes from the washing performed on S with the last instantiation of* I_{Q}.

Proof. Let $I_{\mathrm{B}} = (v_1, v_2, \ldots, v_i)$ be such that $I_{\mathrm{B}}[v_i]$ causes the inconsistency. This corresponds to the presence of a constraint C_{ji} with $j < i$ such that $(v_j, v_i) \notin R_{ji}$. This constraint is coded by a clause $\neg x_j v_j \vee \neg x_i v_i$. On the other hand, I_{Q} is such that $I_{\mathrm{Q}}[x_j v_j] = I_{\mathrm{Q}}[x_i v_i] = 1$. Thus, $\neg x_j v_j \vee \neg x_i v_i$ was shortened once to $\neg x_i v_i$ and has to be shortened again, producing the empty clause. □

The semantic tree developed for the exploration of the domain D_i is so of the form shown in Fig. 1. Its leaves are either dead ends denoted by a box, or potential continuation points. $d1, d2, d3$ stand for the literals associated to the three values in the CSP variable domain.

One can clearly see that the number of nodes out of the root is at least $2d$ if d represents a domain size. That is the case when no value is possible for the current CSP variable (no possible continuation). The worst case occurs when all the potential continuations are possible. We show then that in that case the number of nodes is $d^2 + d$.

Property 3. *The number of nodes* $N(d)$ *necessary to examinate a domain of size* d *by the Quine algorithm verifies* $2d \leqslant N(d) \leqslant d^2 + d$.

Proof. The lower bound $2d \leqslant N(d)$ is obvious. One still has to establish the upper bound. For a domain of size one, only two nodes have to be examined, one corresponding to the interpretation $I[x_i v_j] = 1$, the other to $I[x_i v_j] = 0$. For a domain of size d, let us note $\mathrm{Max}(d)$ the maximum number of developed nodes. From property 1, and for a domain of size d, there are d potential continuation nodes, exactly one by value, and a dead-end node, due to the interpretation $I[x_iv_1]=I[x_iv_2]=\cdots=I[x_iv_d]=0$, because of the positive domain clause $x_iv_1 \vee x_iv_2 \vee \cdots \vee x_iv_d$ that is not satisfied. For a domain of size $d+1$, each continuation node has to be developed by $I[x_iv_{d+1}]=1$ and $I[x_iv_{d+1}]=0$. There are two cases:

- $I[x_iv_{d+1}]=1$ produces the empty clause and so a dead end for $\exists j, 1 \leqslant j \leqslant d$ such that $I[x_iv_j]=1$ and the clause $\neg x_iv_j \vee \neg x_iv_{d+1}$ already reduced to $\neg x_iv_{d+1}$ has to be reduced again.
- $I[x_iv_{d+1}]=0$ deleted the unit clause $\neg x_iv_{d+1}$; this is a continuation node.

Thus, $2d$ new nodes are then developed below the continuation nodes. For a dead-end node deep-rooted under the interpretation $I[x_iv_1]=I[x_iv_2]=\cdots=I[x_iv_d]=0$, we have now in S the positive clause x_iv_{d+1}, issued of the progressive reduction of the positive domain clause $x_iv_1 \vee x_iv_2 \vee \cdots \vee x_iv_d \vee x_iv_{d+1}$. This is no more dead-end node. Two nodes are then developed, one for $I[x_iv_{d+1}]=1$, another for $I[x_iv_{d+1}]=0$. This last node being necessarily a dead-end node. Thus, we establish the induction equation:

- $\mathrm{Max}(1)=2$.
- $\mathrm{Max}(d+1)=\mathrm{Max}(d)+2d+2$ for $d \geqslant 1$.

The solution of this induction equation is $\mathrm{Max}(d)=d^2+d$. Thus we have $2d \leqslant N(d) \leqslant d^2+d$. □

The value of $N(d)$ has to be compared to the number of nodes generated by the examination of a domain of size d by the Backtrack algorithm. It will be, in any case, equal to d. This remark leads us to the following property:

Property 4. *In the worst case, the Quine algorithm develops at most* $(d+1)$ *times more nodes than the CSP Backtrack.*

The worst case occurs when all the domain values are explored. In that case, the examination of a domain by the Backtrack algorithm requires d nodes whereas the cost occasionned by the Quine algorithm is d^2+d that is $d(d+1)$. Claiming the equivalence of the two procedures is so impossible simply because of the wider expressivity of the CSP formalism. Actually, in the CSP formalism, it is implicit that a variable takes one and only one value in its domain, whereas the Quine algorithm has to discover it along its search. Considering now the Davis–Putnam procedure will lead this implicit knowledge to be actually taken into account via the unit-resolution mechanism.

4. From DP to FC

The Quine algorithm and the Backtrack algorithm can be seen, respectively, as the Davis–Putnam procedure and the Forward Checking procedure without any propagation. We shall now show that DP and FC perform the same filterings. Let us first recall the two procedures and their filterings.

4.1. The Davis–Putnam procedure

The function we introduce here is slightly different from those presented in [4]. Actually, it is the same algorithm, but for the purpose of the paper, we want to put the stress on the filtering.

```
function DP(S,I):boolean;
begin
  if $S = \emptyset$ then DP:=true {I is a model}
  else if ($S$ contains an empty clause) then DP:=false
  else begin
    Choice of a boolean variable p occurring in S;
    DP:=DP(Propagate(S,p),I $\cup$ {I[p] = 1}) $\vee$ DP(Propagate(S,$\neg$ p),I $\cup$ {I[p] = 0})
  end
end;
```

The only difference with the Quine algorithm is the function *Propagate*(S,p). This function is commonly known as the unit resolution rule. It takes in input a set of clauses S and a literal p. Its output is a set S' reduced by the assignment of p to 1 and saturated for the unit clauses resolution. That is, if a clause becomes unitary, its remaining literal is assigned to 1, and so on until no more unit clause is present.

We have to mention that Davis and Putnam also consider so-called monotone literals in their algorithm. These literals occur only positively (or negatively). They can be satisfied without changing the satisfiability of the problem and so should be considered by the filtering. The point is that during the resolution of the clausal expression of a CSP, they do not occur.

4.2. The Forward Checking procedure

The idea of the Forward Checking procedure is to consider the variables directly constrained with the most recently assigned variable. If one of their values is not compatible with the last instantiation, it is deleted. No further propagation is done.

```
function FC(P,I):boolean;
begin
  if (no relation is empty) $\wedge$(|I| = n) then FC:=true {I is a solution}
  else if (an ulterior domain is empty) then FC:=false
  else begin
    Choice of $x_i$ not yet instantiated;
    FC:=false;
    for $v_i \in D_i$ do FC:=FC$\vee$FC(Propagation(P,$v_i$), I $\cup$ {I[$x_i$] = $v_i$})
  end
end;
```

In this algorithm, the filtering is done by the function *Propagation*(P, v_i). In input, P is the current CSP no domain of which is empty, and v_i a value of D_i. Its output is the CSP P' verifying the following property: $\forall k, i < k \leqslant n, D'_k = \{v_k \in D_k : (v_i, v_k) \in R_{ik}\}$.

Moreover, the relations induced by this filtering are restricted to the tuples defined on the filtered domains. Let us consider for example a partial instantiation I of the form $\{I[x_1] = v_1, \ldots, I[x_i] = v_i\}$. We have then as current CSP, if $1 \leqslant j, j' \leqslant i < k, k' \leqslant n : D'_k$ defined as indicated above, $R'_{kk'} = \{(v_k, v_{k'}) \in R_{kk'} : v_k \in D'_k \text{ and } v_{k'} \in D'_{k'}\}$.

The compatibility of the current instantiation I with P simply expresses by the fact that no domain is empty.

4.3. Towards an equivalence of the approaches

We shall now show that the introduction of the filtering in both algorithms leads to a total equivalence.

4.3.1. Effects of propagations

We have to distinguish two cases, whether a domain becomes unitary or not. We shall show that, if no domain becomes unitary then unit resolution performs exactly the same domain reductions as forward propagation. On the other hand, if a domain becomes unitary, then unit resolution will propagate this fact, hence forcing a variable to take a value in its domain which may lead to some other domain reductions. As a matter of fact, any FC-like algorithm would choose to instantiate a variable the domain of which becomes unitary since it is the basic first-fail heuristic. Unit resolution is so equivalent to several FC propagations and choices in unitary domains. These cannot be considered as real choices, since they do not generate any new branch in the search tree.

The following property involves a CSP instantiation I_F that can be reduced to an unique variable instantiation and a truth assignment I_D hence reduced to a single literal.

Property 5. *If the value v of the domain of x is suppressed by forward propagation on P with I_F, then $\neg xv$ is produced by unit resolution applied on S with I_D.*

Proof. Let $I_F[x_i] = v_j$ and so $I_D[x_i v_j] = 1$. The deletion of v in the domain of x means that these values are incompatible in the CSP. Thus, there exists a binary clause $\neg xv \vee \neg x_i v_j$. Propagating the assignment of $x_i v_j$ to 1 will of course produce $\neg xv$. $\square$

This property leads us to the following theorem:

Theorem 1 (No unitary domain). *If no domain becomes unitary, the forward propagation on P with I_F deletes a value v in the domain of x if and only if the unit resolution applied on S with I_D produces the literal $\neg xv$.*

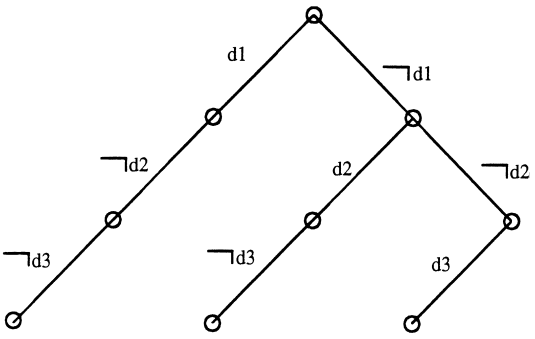

Fig. 2. Exploration of a domain of size three by DP.

Proof. $\Leftarrow$: See Property 5.

$\Rightarrow$: If $\neg xv$ is produced, the positive domain clause is reduced. Since the two problems are semantically equivalents, v is removed from the current domain of x. □

In case a domain becomes unitary, let us consider the following procedure we call Iterated_Propagation. It takes as parameters a CSP P and an instantiation I and applies the forward propagation once. If a domain becomes unitary, it instantiates the corresponding variable to its remaining value and iterates.

```
Procedure Iterated_Propagation(P,I);
begin
  repeat
    P:=Propagation(P,I)
    if (there is an $x_i$ the domain of which is reduced to $\{v_i\}$) then I:=I $\cup$ $\{I[x_i]=v_i\}$
  until (there is no unitary domain)
end;
```

Theorem 2 (Possible unitary domains). *The iterated forward propagation on P with I_F deletes a value v in the domain of x if and only if the unit resolution applied on S with I_D produces the literal $\neg xv$.*

Proof. Actually, the domain $v_1,\dots,v_n$ of the variable x becomes unitary if and only if all its values but one were deleted, say v_1. From Property 5, the negative literals $\neg xv_2,\dots,\neg xv_n$ were produced. Hence, the domain clause of x is reduced to xv_1 and unit resolution propagates this literal. The two procedures perform well the same deletions/instantiations. □

4.3.2. Semantic trees and search trees

Fig. 2 shows the semantic tree of DP exploring a domain of size three. The leaves are potential continuation points, there are as many leaves as values in the domain of the current variable. At each leaf unit resolution will be performed. This leads us to the following property:

Property 6. *The number of nodes out of the root for DP to examine a domain of size d is at most $N(d) = (d \times (d+1))/2 + d - 1$.*

Proof. For a domain of size one (that is, an unit literal), only one node is necessary. Let us note $\mathrm{Max}(d)$ the maximum number of nodes developed by the Davis–Putnam procedure to examine a domain of size d. For a domain of size $d+1$, each continuation node has to be developed by the assignment of the variable coding the $(d+1)$th value of the domain to 0, this gives d new nodes. The assignment of the variable coding the dth value to 0 will generate two new nodes since the $(d+1)$th variable becomes an unit literal. Thus, we establish the induction equation:
- $\mathrm{Max}(1) = 1$.
- $\mathrm{Max}(d+1) = \mathrm{Max}(d) + d + 2$ for $d > 1$.

Its solution is: $\mathrm{Max}(d) = (d \times (d+1))/2 + d - 1$. □

This value has to be compared to the number of nodes generated by the examination of a domain of size d by FC. It will be, in any case, equal to d.

We know from Theorem 2 that the remaining problem at each leaf after unit resolution will be equivalent to that induced by the iterated propagation of FC. Since the choice of a variable the domain of which is unitary cannot be considered as a real choice, both algorithms will make choices at the same depth. This leads us to the following property:

Property 7. *Both algorithms develop the same number of branches in their respective search trees.*

4.3.3. Cost of propagations

The complexities of the filterings in both procedures need to be compared to claim the two algorithms are equivalent. A good measure is the number of consistency checks done by both algorithms. A consistency check is, for CSPs, an access to a relation and, for SAT, an access to a clause. We evaluate the cost of all the propagations performed to go from the root to a node where the CSP variables $x_1, x_2, \dots, x_k$ are instantiated.

Forward propagation: After each instantiation, forward propagation consists in testing the domains of the ulterior connected variables. Hence, if $T(k)$ denotes the number of consistency checks after the kth instantiation, we have: $T(k) = \sum_{1 \leqslant i \leqslant k} (\sum_{j: x_j \in \Gamma^+(x_i)} d_j^i) \leqslant m_k d$ where $\Gamma^+(x_i)$ denotes the set of the ulterior variables connected to x_i. d_j^i stands for $|D_j^i|$, D_j^i being the domain of x_j before the ith filtering. m_k stands for the number of vertices of the sub-graph induced by the instantiation of the variables $x_1, \dots, x_k$ plus the number of outgoing vertices.

Unit resolution: We have to evaluate the treatment of the domain clauses as well as of the constraint clauses.

- Concerning the treatment of the current domain D_k when assigning x_k, the number of accessed clauses by reduction/suppression is: $\frac{1}{2}(d_k^{k-1}+1)d_k^{k-1}$. This value corresponds to the number of exclusion clauses considered when satisfying the domain clause of x_k. Hence, if $T_{\mathrm{dom}}(k)$ denotes the number of consistency checks to the domain and exclusion clauses after the kth instantiation, we have $T_{\mathrm{dom}}(k)=\sum_{1<i\leqslant k}(\frac{1}{2}(d_i^{i-1}+1)d_i^{i-1})\leqslant\frac{1}{2}kd^2$
- For the constraint clauses, let us note $T_{\mathrm{C}}(k)$ the number of consistency checks needed to reach the kth instantiation, we have:

$$T_{\mathrm{C}}(k)=\sum_{1\leqslant i<j\leqslant k}2|\bar{R}_{ij}|+\sum_{1\leqslant i\leqslant k<j}|\bar{R}_{ij}|\leqslant 2m_k d^2,$$

  where $\bar{R}_{ij}$ stands for the complement of the relation R_{ij} and so, $|\bar{R}_{ij}|$ is the number of clauses necessary to express the relation R_{ij}. Factor 2 comes from the fact that a binary clause is accessed at most twice in any propositional assignment.

5. Conclusion

In this paper, we have shown that the Davis and Putnam's procedure is strictly equivalent to the Forward Checking procedure. That is, the Davis Putnam procedure has exactly the same behaviour on the clausal expression of a CSP than the Forward Checking procedure on the same CSP. This theoretical result might have a practical application. When solving the clausal expression of a CSP, DP should consider its special structure to improve its filtering and so perform as fast as FC. Note that we have restricted our analysis to binary relations. It is because FC has been originally defined for binary CSPs. Several generalizations has been proposed, e.g. [14]. In [14], one can find a generalization of FC for non-binary constraints which is equivalent to DP, considering the same translation from CSP to SAT.

Recently, Freuder and Sabin [12] studied an algorithm to solve CSPs they called MAC. MAC uses the same basic framework as Forward Checking, alternating search and consistency inference steps, but differs conceptually in two aspects:

- The constraint network is made arc-consistent initially.
- When during the search a new variable x is instantiated to a value v, all other values in the domain are eliminated and the effects of removing them are propagated through the constraint network as necessary to restore full arc-consistency.

In their paper, they showed that maintaining full arc-consistency during search is often in fact very cost effective. That is why MAC is nowadays considered even better than FC. As a matter of fact, MAC is also a rewriting of well-known things in propositionnal calculus. In [2], Dalal introduced a way to code a CSP as a SAT instance and showed that on this clausal expression, unit resolution was strictly equivalent to arc-consistency. With the insights given in this paper, the reader should easily get convinced that the Davis–Putnam procedure on Dalal's expression of a CSP will behave exactly as MAC, etc.

To conclude on these epistemological considerations, we want to mention that, as was already indicated by Chvátal and Szemerédi [1]: "The Davis–Putnam procedure was incidentaly proposed some 50 years earlier (in 1910) by Löwenheim".

References

[1] V. Chvátal, E. Szemerédi, Many hard examples for resolution, J. ACM 35 (4) (1988) 759–768.
[2] M. Dalal, Tractable deduction in knowledge representation systems, in: Proceedings of the third International Conference on Principles of Knowledge Representation and Reasoning, KR'92, Boston, MA, 1992, pp. 393–402.
[3] M. Davis, G. Logemann, D. Loveland, A machine program for theorem proving, Comm. ACM 5 (1962) 394–397.
[4] M. Davis, H. Putnam, A computing procedure for quantification theory, J. ACM 7 (1960) 201–215.
[5] J. de Kleer, A comparison of ATMS and CSP techniques, Proceedings IJCAI'89, 1989.
[6] M. Dincbas, P. van Hentenryck, H. Simonis, A. Aggoun, T. Graf, F. Berthier, The constraint logic programming language CHIP, Proceedings of the second International Conference on Fifth Generation Computer Systems, 1988, pp. 249–264.
[7] O. Dubois, P. André, Y. Boufkhad, J. Carlier, SAT versus UNSAT, in: D.S. Johnson, M.A. Trick (Eds.), Cliques, Coloring, and Satisfiability, DIMACS Series in DM and TCS, Vol. 26, AMS, Providence, RI, 1996.
[8] R. Génisson, On enumerative algorithms and polynomial classes in SAT and constraint satisfaction problems, Ph.D. Dissertation, Laboratoire d'Informatique de Marseille, Université de Provence, France, 1996 (in French).
[9] R.M. Haralick, G.L. Elliot, Increasing tree search efficiency for constraint satisfaction problem, Artificial Intelligence 14 (1980) 263–313.
[10] H. Montanari, Network of constraints: fundamental properties and applications to picture processing, Inform. Sci. 7 (1974) 95–132.
[11] W.V. Quine, A proof procedure for quantification theory, J. Symbolic Logic june (1955) 141–149.
[12] D. Sabin, E. Freuder, Contradicting conventional wisdom in constraint satisfaction, in: Proceedings of the 11th European Conference on Artificial Intelligence, ECAI'94, 1994, pp. 125–129.
[13] P. van Beek, G. Kondrak, A theoretical evaluation of selected backtracking algorithms, Proceedings of the IJCAI'95, Montreal, Canada, 1995, pp. 541–547.
[14] P. van Hentenryck, Constraint Satisfaction in Logic Programming, Logic Programming Series, MIT Press, Cambridge, 1989.