Merging the local and global approaches to probabilistic satisfiability

Pierre Hansen, Sylvain Perron *

GERAD and HEC Montréal, 3000, Chemin de la Côte-Sainte-Catherine, Montréal (Qc), Canada H3T 2A7

Received 31 May 2006; received in revised form 19 March 2007; accepted 20 March 2007
Available online 28 March 2007

Abstract

The probabilistic satisfiability problem is to verify the consistency of a set of probability values or intervals for logical propositions. The (tight) probabilistic entailment problem is to find best bounds on the probability of an additional proposition. The local approach to these problems applies rules on small sets of logical sentences and probabilities to tighten the given probability intervals. The global approach uses linear programming to find best bounds. We show that merging these approaches is profitable to both: local solutions can be used to find global solutions more quickly through stabilized column generation, and global solutions can be used to confirm or refute the optimality of the local solutions found. As a result, best bounds are found, together with their step-by-step justification.

© 2007 Elsevier Inc. All rights reserved.

Keywords: Probabilistic satisfiability; Probabilistic entailment; Rule-based approach; Linear programming; Column generation; Stabilization

1. Introduction

The probabilistic satisfiability (PSAT, see, e.g. [22] for a survey) and the (tight) Probabilistic Entailment problems are central in reasoning under uncertainty. The expressions of these problems are based upon propositional logic and standard probability theory (see, e.g. [11] for others approaches).

The probabilistic satisfiability problem can be expressed as follows: let \( S_1, S_2, \ldots, S_m \) denote propositional sentences obtained from the logical variables \( x_1, x_2, \ldots, x_n \) with the usual Boolean operators \( \lor \) (or), \( \land \) (and), and \( \neg \) (not). Let \( \pi_1, \pi_2, \ldots, \pi_m \) be probabilities that these sentences are true. Are these probabilities consistent, i.e., such that there exists a probability distribution over the set of the \( 2^n \) complete products of logical variables \( x_1, x_2, \ldots, x_n \) in direct or complemented form for which the sum of probabilities \( p_1, p_2, \ldots, p_{2^n} \) of the products for which sentence \( S_i \) is true is equal to its probability \( \pi_i \) of being true for all \( i = 1, 2, \ldots, m \)?

* Corresponding author.

E-mail addresses: pierre.hansen@gerad.ca (P. Hansen), sylvain.perron@gerad.ca (S. Perron).

0888-613X/S - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.ijar.2007.03.001
The (tight) probabilistic entailment problem, also usually noted \textsc{psat}, has the following formulation: let $S_{m+1}$ be an additional propositional sentence, also obtained from $x_1, x_2, \ldots, x_n$ with the operators $\lor, \land$ and $\lnot$. Find the best possible lower and upper bounds on its probability $p_{m+1}$ to be true.

Both problems first appear in the work of George Boole, who expresses them in his famous book of 1854, *An Investigation of the Laws of Thought* [4], as particular linear programs (thus providing the most remarkable prefiguration of linear programming up to the work of Kantorovich in the late 1930s [32]). Using $p_1, p_2, \ldots, p_m$ as parameters, Boole characterizes consistent probability vectors by a set of linear inequalities called “Conditions of possible experience”; moreover, he shows how to obtain best possible lower and upper bounds on $p_{m+1}$ as inequalities involving the probabilities $p_1, p_2, \ldots, p_m$. He calls this problem “the general problem in the theory of probabilities.”

While this last qualification is outdated, the \textsc{psat} model has proven to be a remarkably fecund one despite an eclipse of over a century in English-speaking countries [20]. It has been generalized to tackle: (i) interval values for probabilities; (ii) conditional probabilities in the objective and constraints; (iii) operations on conditional events; and (iv) qualitative probabilities. (See the survey [22] for details and references.) Moreover, \textsc{psat} plays an essential role in de Finetti’s theory of subjective probabilities as a special case of testing coherence of a set of bets (see, e.g. [6]) and in Walley’s theory of imprecise probabilities applying natural extension [43].

There are two basic approaches to the solution of \textsc{psat}:

(i) First, there is the global approach, which extends the linear programming approach of George Boole. It was pioneered by Hailperin [19] and rediscovered by several authors (see [22] for references), prominent among them being Nilsson [37]. For extension of this approach to a logic for reasoning about probabilities, see Fagin et al. [12].

The main strength of this approach is that it makes it possible to rigorously prove that probabilities $p_1, p_2, \ldots, p_m$ are consistent or that they are not, and to obtain proven best possible bounds on $p_{m+1}$. Sometimes it has been assumed that because the number of variables in the linear program is exponential in the number of logical variables $x_1, x_2, \ldots, x_n$, this approach is limited to small instances. However, using the powerful column generation technique of linear programming (for example [5, Chapter 26]) together with nonlinear 0–1 programming to find the entering column allows the solution of \textsc{psat} instances with several hundred variables and/or sentences. (See [8, 15, 29, 33] and below.) Consequently, column generation algorithms appear to be the current state of the art in solving \textsc{psat}. Nevertheless, large instances still require a substantial computing time. In this paper, we push this approach a bit further by providing the first application of stabilized column generation [9] to \textsc{psat}.

Another strength of the global approach is that it makes it possible to analytically solve small \textsc{psat} instances, using enumeration of extreme points and extreme rays of the dual linear program [19, 24]. Thus, the best possible rules for tightening probability intervals can be generated in an automated way. The weak point of the global approach is that it does not provide a justification of the bounds obtained which can be followed step-by-step by the user.

(ii) Second, there is the local or rule-based approach, which exploits rules for tightening probability intervals through propagation. Such rules exploit probabilities for premises to be true in order to bound possibilities for a conclusion to be true. They are locally optimal, i.e., best possible for the small system considered. Frisch and Haddawy [14] gather many such rules and apply them to several examples. They propose an “anytime-deduction method” in which one can interrupt the solution process at any desired time, with a probability interval on $p_{m+1}$ that is valid. The result is only significant if the probabilities $p_1, p_2, \ldots, p_m$ are consistent (otherwise any interval could in principle be obtained). Checking this is a \textsc{psat} problem in itself. Recently, some new algorithms [31, 30, 34, 35] that generate rules and apply them in a systematic way have been proposed. The strong points of the local approach are its speed in providing bounds and the fact that it gives an easy-to-follow justification of the bounds attained from a list of the rules applied.

Its weak points are that local methods may not be able to prove that a system is inconsistent or attain the best possible bounds. In other words, they are usually incomplete. Moreover, these defects may be difficult to correct while remaining within the local approach. To illustrate this, Lukasiewicz [34] gives
examples of PSAT problems with conditional probabilities where the rules converge to a probability bound that is not the best possible one, and where finding such a bound would require a number of rules exponential in the number of variables.

A comparison of the local and the global approaches has been made by Lukasiewicz [34]. His conclusion is that even a streamlined local approach cannot compete with the global one. In this paper, we show that, perhaps surprisingly, the local approach is still useful, not as a stand alone approach but in conjunction with the global approach. More precisely, we show that the defects in both the local and the global approaches to PSAT may be alleviated, and for some of them removed, by merging the two approaches.

The paper is organized as follows: a detailed description of the global approach is given in the following section. Then the local approach is described in Section 3. In Section 4, we present the merged approach as well as numerical results illustrating that the local approach can accelerate the global one through stabilized column generation. In Section 4, we also give a small example that illustrates how the global approach can be used to show that the proposed rules in the local approach are not sufficient to provide the best bounds. Conclusions are given in Section 5.

2. Probabilistic satisfiability and linear programming: the global approach

2.1. Problem statement

The values “true” and “false” of the $x_k$ and $S_i$ will be associated with 1 and 0. There are $2^n$ complete products $w_j$, for $j = 1, 2, \ldots, 2^n$, of the variables $x_1, x_2, \ldots, x_n$ in direct or complemented form. These products may be called, following Leibniz, possible worlds. (Note that Nilsson [37] reserves the name “possible worlds” for the distinct $m$-columns that appear in (1) below and calls “impossible worlds” those $m$-columns that do not.) In each possible world $w_j$, any sentence $S_i$ is either true or false. The PSAT problem may then be reformulated: Is there a probability distribution $p_1, p_2, \ldots, p_{2^n}$ on the set of possible worlds such that the sum of the probabilities of the possible worlds in which sentence $S_i$ is true is equal to its probability $\pi_i$ of being true for $i = 1, 2, \ldots, m$?

Defining the $m \times 2^n$ matrix $A = (a_{ij})$ where $a_{ij}$ is equal to 1 if $S_i$ is true in possible world $w_j$ and 0 otherwise, PSAT may be written as

$$
\begin{align*}
\emptyset p &= 1 \\
Ap &= \pi \\
p &\geq 0
\end{align*}$$

(1)

where $\emptyset$ is a $2^n$ unit row vector, and $p$ and $\pi$ are the column vectors $(p_1, p_2, \ldots, p_{2^n})^T$ and $(\pi_1, \pi_2, \ldots, \pi_m)^T$, respectively. The answer is yes if there is at least one vector $p$ satisfying (1) and no otherwise. Note that not all the columns of $A$ need to be different. Moreover, not all of the $2^n$ possible different column vectors of $A$ need to be present, and in most cases they will not all be present.

Considering one more sentence $S_{m+1}$, with an unknown probability $\pi_{m+1}$, leads to the probabilistic entailment problem. Usually, the constraints (1) do not impose a unique value for the probability $\pi_{m+1}$ of $S_{m+1}$. As de Finetti shows [7], this is the case if and only if the line-vector $A_{m+1} = (a_{m+1,1}, \ldots, a_{m+1,n})$, where $a_{m+1,j} = 1$ if $S_{m+1}$ is true in possible world $w_j$ and $a_{m+1,j} = 0$ if not, is a linear combination of the rows of $A$. Otherwise, the constraints (1) imply bounds on the probability $\pi_{m+1}$. The probabilistic entailment problem is to find the best possible bounds. It can be written as

$$
\begin{align*}
\min \text{ or } \max \quad A_{m+1}p \\
\text{s.t.} \quad \emptyset p &= 1 \\
Ap &= \pi \\
p &\geq 0
\end{align*}$$

(2)

Nilsson [37] calls (1) and (2) probabilistic logic and probabilistic entailment. However, while (1) and (2) are very useful inference tools, they do not properly constitute a logic, i.e., a set of axioms and inference rules. The term
probabilistic satisfiability, proposed by Georgakopoulos, Kavvadias, and Papadimitriou [15], appears better suited as it stresses the relationship of (1) with the satisfiability problem, which is the particular case where \( \pi = 1 \) and a solution with a single positive \( p_j \) is required (which can be easily deduced from any other solution of (2) if there are identical columns with positive probabilities). It is shown in [15] that the LP problem (1) is in NP. This implies that it is NP-complete and that (2) is NP-hard.

In order to illustrate problem (2), we next recall a classical example. It will be used later to illustrate how to formulate the subproblem in the column generation algorithms for PSAT.

**Example 1** (*Boole’s challenge problem, 1851*) [3]. Find the best possible bounds on the probability of \( S_4 \equiv x_3 \) subject to

\[
\begin{align*}
\text{prob}(S_1 \equiv x_1) &= \pi_1 \\
\text{prob}(S_2 \equiv x_2) &= \pi_2 \\
\text{prob}(S_3 \equiv x_1 \land x_3) &= \pi_3 \\
\text{prob}(S_4 \equiv x_2 \land x_3) &= \pi_4 \\
\text{prob}(S_5 \equiv x_1 \land x_2 \land x_3) &= 0.
\end{align*}
\]

Considering the eight possible worlds \( w_1 = (1, 1, 1)^T, \ w_2 = (1, 1, 0)^T, \ w_3 = (1, 0, 1)^T, \ w_4 = (1, 0, 0)^T, \ w_5 = (0, 1, 1)^T, \ w_6 = (0, 1, 0)^T, \ w_7 = (0, 0, 1)^T \) and \( w_8 = (0, 0, 0)^T \), the probabilistic entailment problem becomes

\[
\begin{align*}
&\min/\max \quad p_1 + p_3 + p_5 + p_7 \\
&\text{s.t.} \quad p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8 = 1 \\
&\quad p_1 + p_2 + p_3 + p_4 = \pi_1 \\
&\quad p_1 + p_2 + p_5 + p_6 = \pi_2 \\
&\quad p_1 + p_3 = \pi_3 \\
&\quad p_1 + p_5 = \pi_4 \\
&\quad p_7 = 0 \\
&\quad p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8 \geq 0.
\end{align*}
\]

This form will be used in all numerical experiments described in Section 4.

2.2. Column generation

The number of columns in the linear programs (1), (2), and (3) grows exponentially in the minimum of the number \( m \) of sentences and the number \( n \) of logical variables in these sentences. In view of the enormous size of these programs (about \( 10^9 \) columns for \( \min(m, n) = 30 \), \( 10^{18} \) columns for \( \min(m, n) = 60 \), etc.), it is sometimes stated in the literature that they are intractable in a practical sense, not only in the worst case. For instance, Nilsson [38], in a review of work subsequent to his “Probability Logic” paper of 1986 [37], speaks of the “complete impracticability of solving large instances” and recommends looking for heuristics. Such views are overly pessimistic: while writing large probabilistic satisfiability problems explicitly is impossible, they can be solved quite efficiently by keeping them implicit. The tool to be used is an advanced linear
programming tool called column generation [5,16,17]. Two programs are associated with the original linear program: on the one hand, the master problem, which is identical to the original program itself but with only a small number of explicit columns, and on the other hand, the subproblem, whose role is to determine the entering column, as in the simplex or revised simplex algorithms (see e.g. [5]). A specific optimization problem must be solved for this purpose. Once the entering column is determined, its expression in the current master problem is calculated, and a simplex iteration takes place.

Next we recall the principle of the column generation method for linear programming. Consider the linear program

$$\begin{align*}
\min \quad & z = cx \\
\text{s.t.} \quad & Ax = b \\
& x \geq 0
\end{align*}$$

and its solution by the simplex algorithm. Note that by adding slack and surplus variables, (3) can be expressed in the form (4). At a current iteration (after a possible reindexing of the variables), let \( A = (B, N) \), where \( B \) and \( N \) denote the submatrices of basic and nonbasic columns, respectively.

Problem (4) can be expressed as follows:

$$\begin{align*}
\min \quad & z = c_B B^{-1} b + (c_N - c_B B^{-1} N) x_N \\
\text{s.t.} \quad & x_B + B^{-1} N x_N = B^{-1} b \\
& x_B, x_N \geq 0
\end{align*}$$

where \( x_B, x_N \) are the vectors of basic and nonbasic variables and \( c_B, c_N \) the corresponding vectors of coefficients in the objective function. In the revised simplex method, one stores only the matrix \( B^{-1} \) (in compact form), the current basic solution \( B^{-1} b \) and value \( c_B B^{-1} b \), as well as the data. The entering variable is determined by computing the smallest reduced cost, using the initial data, i.e.,

$$c_k - c_B B^{-1} A^k = \min_{j \in N} c_j - c_B B^{-1} A^j = c_j - u^T A^j$$

where \( u = c_B B^{-1} \) is the current column vector of dual variables. This computation is not too time-consuming provided the matrix \( A \) is sparse and the columns are not too numerous. The entering column is then computed as \( B^{-1} A^k \), and the simplex iteration proceeds as usual (optimality check, unboundedness check, choice of leaving variable, updating of solution and basis inverse).

If the number of columns is exponential in the input size, one must compute

$$\min_{j \in N} c_j - u^T A^j$$

without considering the nonbasic columns individually. This is done by solving the subproblem, a specific optimization problem in which the coefficients in the columns \( A^j \) are the variables.

For the minimization form of (2), the subproblem (7) is

$$\min_{j \in N} c_j - u^T A^j = \min_{x \in \{\text{true, false}\}^n} S_{m+1} - u_0 - \sum_{i=1}^m u_i S_i$$

where, as discussed above, the values “true” and “false” for the \( S_i, i = 1, \ldots, m + 1 \) are identified with the numbers 1 and 0. This problem can be viewed as a weighted \textsc{maxsat} problem, as observed in [15]. Then (8) is transformed into an arithmetical expression involving the logical variables \( x_1, \ldots, x_n \) appearing in the \( S_i \), with the values “true” and “false” also associated with 1 and 0. This is done by eliminating the Boolean operators \( \lor, \land, \neg \) using the relations

$$\begin{align*}
x_i \lor x_j & \equiv x_i + x_j - x_i x_j \\
x_i \land x_j & \equiv x_i x_j \\
\bar{x}_i & \equiv 1 - x_i
\end{align*}$$

The resulting expression is a nonlinear (or multilinear) real-valued function in 0–1 variables, or a nonlinear 0–1 function, or a \textit{pseudo-Boolean function} [21].
Example 1 (continued). Subproblem (8) is

$$\begin{align*}
\text{min} & \quad S_6 - u_0 - u_1 S_1 - u_2 S_2 - u_3 S_3 - u_4 S_4 - u_5 S_5 \\
= & \quad x_3 - u_0 - u_1 x_1 - u_2 x_2 - u_3 x_3 x_3 - u_4 x_3 x_3 - u_5 x_3 x_3 \\
= & \quad -u_0 - u_1 x_1 - u_2 x_2 + (1 - u_3) x_3 + (u_3 - u_5) x_3 x_3 + (u_5 - u_4) x_3 x_3 - u_5 x_3 x_3 \\
\text{with} & \quad x_1, x_2, x_3 \in \{0, 1\}.
\end{align*}$$

Problem (8) must be solved at each iteration of the column generation method and may be time-consuming when the number of 0–1 variables is large. Indeed, the minimization of a nonlinear 0–1 function is NP-hard, as numerous NP-hard problems, e.g., INDEPENDENT SET, can be easily expressed in that form. However, for guaranteeing convergence it is not mandatory to solve (8) exactly at all iterations. A heuristic method may be applied as long as it gives a negative reduced cost. Indeed, an iteration of the simplex algorithm can be performed by entering any column with a negative reduced cost. Such a column may differ from the column with the minimum reduced cost.

Various heuristics can be used to solve (8). In our implementation, we use a Variable Neighborhood Search (VNS) heuristic [25,36] which allows easy multiple pricing, i.e., the simultaneous generation of several entering columns [26].

VNS is a metaheuristic based on the idea of systematic change of neighborhood during the search. VNS explores close and then increasingly far neighborhoods of the incumbent (or best known) solution in a probabilistic way. This means that favorable characteristics of the incumbent solution can be kept and used to obtain promising neighboring solutions. VNS applies a local search routine repeatedly in order to go from these neighboring solutions to local optima. Fig. 1 presents the steps of our implementation of VNS where $f(X)$ corresponds to the nonlinear real-valued function in $X = (x_1, x_2, \ldots, x_n) \in \{0, 1\}^n$. The set of solutions in the $k$th neighborhood of a solution is obtained by applying $k$ complementation on the 0–1 vector defining a solution of problem (8). The only move used during the local search phase (step 2.2.2) is complementation of the variable with the most negative partial derivative $D_i$.

Our implementation of VNS exploits the fact that in our column generation algorithm, we wish to generate a high number of columns with negative reduced cost at each iteration. Usually, adding multiple columns at each iteration instead of only one reduces the number of iterations and speeds up the algorithm. The list $L$ gives the set of columns that must be added to the master problem because they have a negative reduced cost. During our experiments with VNS, we have observed that memorizing only local optima with negative values

1 Initialization:
1.1 Let $L = \emptyset$ be the set of solutions with negative values;
1.2 Select an initial solution $X$. If $f(X) < 0$ then add $X$ to $L$;

2 Repeat the following until the stopping condition is met:
2.1 Set $k \leftarrow 1$;
2.2 Until $k = k_{\text{max}}$, repeat the following steps:
   2.2.1 shaking. Generate a vector $X'$ at random by complementing $k$ variables of $X$;
   2.2.2 local search. Apply a steepest descent method with $X'$ as initial solution; denote with $X''$ the obtained local optimum; if $f(X'') < 0$ and $X'' \notin L$ then add $X''$ to $L$;
   2.2.3 move or not. If $f(X'') < f(X)$, then move there ($X \leftarrow X''$), and continue the search with $k = 1$; otherwise, set $k \leftarrow k + 1$.

Fig. 1. Steps of the VNS heuristic for the subproblem.
gives better results than memorizing all the solutions with negative values. This appears to be due to the fact
that columns obtained in the first way differ more than those obtained in the second.

Two stopping conditions are used simultaneously. Limits are imposed on the maximum number of iterations and on the maximum cardinality of \(L\). The heuristic stops when any one of these limits is reached.

For problem (2), an exact algorithm must be applied to the subproblem at least once in order to prove that there are no more columns with a reduced cost of the desired sign when the heuristic fails to find any such columns, that is to say, to prove that the optimality conditions are satisfied. There are various techniques to optimize a nonlinear 0–1 function. These techniques are reviewed in [23]. In our program, linearization is used together with the very efficient and largely used commercial software ILOG CPLEX 8.1 [28] to solve the resulting mixed integer linear program. This procedure proves to be not too time-consuming, as in practice most dual variables are equal to 0 at the optimum, and thus the nonlinear 0–1 function to be optimized contains very few terms.

### 2.3. Stabilization

In order to improve the convergence of the column generation solution process, it is possible to use the technique proposed by du Merle et al. [9] to stabilize the value of the dual variables. In order to do so, we perturb the linear program by adding bounded surplus and slack variables in the primal and penalizing these variables in the objective function. These modifications in the primal correspond to the introduction of bounds on dual variables in the dual of that linear program and to the penalization of these variables when they are outside a given interval. If available, an estimate of the optimal values of the dual variables is used to choose initial intervals. Stabilization often leads to a substantial reduction in the number of iterations necessary to obtain an optimal solution to the original linear program.

Let (10) and (11) be, respectively, the primal and dual formulation of the minimization form of the PSAT problem (2):

\[
\begin{align*}
\min & \quad A_{m+1}p \\
\text{s.t.} & \quad 1p = 1 \\
& \quad Ap = \pi \\
& \quad p \geq 0 \\
\max & \quad u_0 + \pi^T u \\
\text{s.t.} & \quad u_0^T + A^T u \leq A_{m+1}^T
\end{align*}
\]

where \(u\) is the column vector \((u_1, u_2, \ldots, u_m)^T\) and \(u_i\), for \(i = 0, 1, \ldots, m\), is the dual variable associated to the constraint \(i\) of the primal.

By applying stabilization, we obtain the following primal and dual programs:

\[
\begin{align*}
\min & \quad A_{m+1}p \\
& \quad -\delta_0^- y_0^- + \delta_0^+ y_0^+ \\
& \quad -\delta^- y^- + \delta^+ y^+ \\
\text{s.t.} & \quad 1p - y_0^- + y_0^+ = 1 \\
& \quad Ap - y^- + y^+ = \pi \\
& \quad y_0^- \leq \epsilon_0^-, y_0^+ \leq \epsilon_0^+ \\
& \quad y^- \leq \epsilon^-, y^+ \leq \epsilon^+ \\
& \quad p, y_0^-, y_0^+, y^-, y^+ \geq 0
\end{align*}
\]
max \[ u_0 + \pi^T u \]
\[ - \epsilon_0 w_0 - \epsilon_0^+ w_0^+ \]
\[ - \epsilon^- T w^- - \epsilon^+ T w^+ \]
s.t. \[ u_0 f^T + A^T u \leq A_{m+1}^T \]
\[ - u_0 - w_0^- \leq -\delta_0 \]
\[ u_0 - w_0^+ \leq \delta_0^+ \]
\[ - u - w^- \leq -\delta^- \]
\[ u - w^+ \leq \delta^+ \]
\[ w_0, w_0^+, w^-, w^+ \geq 0 \]

where \( y^-, y^+ \) and \( w^+ \) are the column vectors \((y_1^-, y_2^-, \ldots, y_m^-)^T, (y_1^+, y_2^+, \ldots, y_m^+)^T, (w_1^-, w_2^-, \ldots, w_m^-)^T, \) and \((w_1^+, w_2^+, \ldots, w_m^+)^T, \) respectively. In (12), the variables \( y_i^- \) and \( y_i^+ \) are, respectively, the surplus and slack variables of constraint \( i, \) for \( i = 0, 1, \ldots, m. \) They are bounded by \( \epsilon_i^- \) and \( \epsilon_i^+ \), and they are penalized in the objective function by \( \delta_i^- \) and \( \delta_i^+ \). In (13), the variables \( w_i^- \) and \( w_i^+ \) correspond to the violation value of the lower bound \( \delta_i^- \) or upper bound \( \delta_i^+ \) on the dual variable \( u_i \), for \( i = 0, 1, \ldots, m. \) These variables are penalized in the objective function by \( \delta_i^- \) and \( \delta_i^+ \), respectively. Therefore, there are two groups of parameters: (i) the penalties \( \delta_i^- \) and \( \delta_i^+ \), for \( i = 0, 1, \ldots, m, \) which implicitly correspond to bounds on dual variables; and (ii) the bounds \( \epsilon_i^- \) and \( \epsilon_i^+ \), for \( i = 0, 1, \ldots, m, \) on surplus and slack variables, which implicitly correspond to penalties on the violation of the bounds \( \delta_i^- \) and \( \delta_i^+ \) on the dual variables. After solving (12) (or (13)) it is necessary to verify that the slack and surplus variables are all at 0. If they are not at 0, the parameters are updated (see Section 4) and the process is iterated.

3. The local approach

The local approach, also known as the rule-based approach or anytime-deduction approach, is a heuristic method that solves probabilistic satisfiability and probabilistic entailment problems by considering a series of rules instead of using linear programming tools. Applying these rules tightens the given probability intervals on the probability values of the \( m \) propositions and possibly of the objective function proposition \( S_{m+1}. \) If the probability interval for one of the constraints becomes empty, a contradiction has been detected and the corresponding \textsc{psat} problem is inconsistent. The rules can be applied iteratively as in constraint propagation heuristics. Applying such rules is very quick but incomplete: there is no guarantee (except for very particular classes of problems \([14,35]\)) that once they do not improve the probability intervals any more these intervals are the tightest possible. In particular, rules-based heuristics might fail to detect inconsistency of a \textsc{psat} problem.

The rule-based approach to \textsc{psat} and close problems has been explored by Dubois and Prade \([10,11]\), Frisch and Haddawy \([14]\), Lukasiewicz \([34,35]\), and Jaumard et al. \([30,31,39]\), as well as by many others (see, e.g. \([1,2,18,40–42]\), as well as references \([14,34,35]\)).

Here, we focus on the algorithm \textsc{ad-psat} \([31,39]\), which is an improved version of the algorithm \textsc{turbosat} presented in \([30]\). \textsc{ad-psat} considers a series of rules (see Fig. 2 for an illustration of one such rule and \([39]\) for the complete list) expressed by subsets of sentences and probabilities. The rule of Fig. 2 can be interpreted as follows: find a tight entailment on \( x_2 \) given the probability interval \([\pi_1, \pi_1]\) on \( x_1 \) together with the probability interval \([\pi_2, \pi_2]\) on \( x_1 \lor x_2, \) which is equivalent to \( x_1 \) implying \( x_2. \) Then for consistency the probabilities \( \pi_i, \)

i = 1, 2, must satisfy the obvious conditions \( 0 \leq \pi_i \leq 1 \) and \( \pi_2 \leq \pi_i, \) as well as the less obvious condition \( \pi_1 + \pi_2 \geq 1. \) Moreover, the probability of \( x_2 \) is at least the largest of \( +0 \) and \( \pi_1 + \pi_2 - 1 \) and at most the smallest of \( +1 \) and \( \pi_2. \) Note that these last bounds have long been known \([13,27]\).

Rules used by \textsc{ad-psat}, such as the one illustrated in Fig. 2, are obtained in an automated and systematic way by analytically solving the dual of small \textsc{psat} problems, i.e., Problem (11) above. Justification follows from the duality theorem of linear programming as shown by Hailperin \([19]\) and Hansen et al. \([24]\). All consistency conditions are obtained by enumeration of extreme rays of (11) and all lower and upper bounds on the objective by enumerating vertices of that polyhedron. Proofs and a series of examples are given in \([24]\).
Consider a PSAT problem that consists in finding $\pi_3$ and $\overline{\pi}_3$, the best possible lower and upper bounds on $\pi_3 = \text{prob}(S_3 \equiv x_2)$, subject to

\[
\begin{align*}
\text{prob}(S_1 \equiv x_1) & \in [\underline{x}_1, \overline{x}_1] \\
\text{prob}(S_2 \equiv x_1 \lor x_2) & \in [\underline{x}_2, \overline{x}_2].
\end{align*}
\]

If the consistency conditions

\[
\begin{align*}
0 \leq \pi_i \leq 1 & \quad i = 1, 2 \\
\underline{x}_i \leq \pi_i & \quad i = 1, 2 \\
\pi_1 + \pi_2 & \geq 1
\end{align*}
\]

hold, then the optimal bounds on $\pi_3$ are the following:

\[
\begin{align*}
\underline{\pi}_3 &= \max\{0, \underline{x}_1 + \underline{x}_2 - 1\} \\
\overline{\pi}_3 &= \min\{1, \overline{x}_2\}.
\end{align*}
\]

AD-PSAT is a sequential method that tries to tighten the probability interval on a selected variable at each iteration. This is done by solving a reduced probabilistic entailment problem consisting of a subset of logical sentences together with the probability intervals obtained thus far on the variables. We next illustrate the AD-PSAT procedure on a small example.

**Example 2.** Find best possible bounds on the probability of $S_4 \equiv x_3$ subject to

\[
\begin{align*}
\text{prob}(S_1 \equiv x_1) & \in [0.6, 1] \quad (14) \\
\text{prob}(S_2 \equiv x_1 \lor x_2) & \in [0.8, 0.9] \quad (15) \\
\text{prob}(S_3 \equiv x_2 \lor x_3) & \in [0.75, 0.85]. \quad (16)
\end{align*}
\]

AD-PSAT first tries to update the bounds on $\text{prob}(x_2)$ while considering the subsystem composed of (14) and (15). By applying the rule in Fig. 2, it concludes that the subsystem is consistent and obtains

\[
\text{prob}(x_2) \in [0.3, 0.9]. \quad (17)
\]

Then, applying the same rule to the subsystem composed of (17) and (16), AD-PSAT concludes that the subsystem is consistent and updates the probability interval on $S_4$ to $[0.05, 0.85]$. The solution process stops there without detecting inconsistency and finds the optimal solution interval $[0.05, 0.85]$ on $\pi_4$. This solution can be confirmed by the global approach (or the merged approach described below). Therefore, on the one hand, optimality is proved after confirmation and, on the other hand, an explicit justification of the bounds obtained is given in terms of the rules applied.

Usually, for large instances, only a few sentences are used by AD-PSAT to find either the best bounds on $\pi_{m+1}$ or a good approximation of these bounds. AD-PSAT solves large instances very rapidly (less than 2 s for problems with up to 1000 variables and 3000 logical sentences on a single processor of an Entreprise 10,000 computer with 64 processors, 400 MHz, and 64GB RAM).

4. The merged approach

4.1. Description of the merged approach

Algorithm 1 given below presents the detailed steps of the merged approach. In step (a) a local algorithm is used to find, if the local approach did not detect inconsistency, a heuristic evaluation of the bounds on $\pi_{m+1}$. If the heuristic solution corresponds to the optimal solution, step (a) will also give the subset of the only logical
sentences $S_i$ that influence the optimal bounds on $\pi_{m+1}$. Sentences $S_i$ used in this step correspond to a compact linear program (10) and its dual (11). In Step (b), this linear program is solved to optimality and provides a first estimate $\hat{u}_i$ of each dual variable $u_i$ of problem (11): $\hat{u}_i$ is equal to the corresponding optimal dual variable and equal to 0 otherwise. The estimates $\hat{u}_i$ are used to initialize the penalties $\delta^{-}_i$ and $\delta^{+}_i$ of problem (12). The stabilized column generation process (as explained in Section 2.3) is used in step (c) to obtain the exact solution of the original problem (2).

**Algorithm 1.** Steps of the Merged Approach

(a) Use the local approach (e.g., AD-PSAT) to find provisional conclusions for problem (2). If the local approach detects incoherence, stop because problem (2) is incoherent. Otherwise, select the sentences $S_i$ used by this approach to find the final values of $\pi_{m+1}$ and $\pi_{m+1}$, and go to step (b).

(b) Use the global approach (the column generation algorithm) to solve a compact linear program with only the rows associated with the $S_i$ used by the local approach. If this compact linear program is infeasible, stop because the complete problem (2) is incoherent. Otherwise, use the optimal dual variables of this compact linear program as estimates of those of (2), and go to step (c).

(c) Solve the complete problem (2) by stabilized column generation using the estimated dual variables to confirm the results obtained by AD-PSAT, or to show that problem (2) is incoherent or to obtain an optimal solution of this problem.

### 4.2. Contribution of the local approach

In this section, we present numerical results showing that the use of the solution obtained by the local approach can accelerate the solution process of the global approach and also provide a full justification of the results. The test instances have been randomly generated in a similar fashion to those in Jaumard et al. [29]. All the instances correspond to problem (3), and the probability intervals are set in such a way that the problem is consistent. If we were to choose instances purely at random, the local approach would very quickly detect infeasibility in most of the numerous cases in which the problem is infeasible. Moreover, the local approach detects inconsistency very quickly. Numerical results presented in [31] show that the local approach detects inconsistency in all inconsistent instances in less than 1% of the time spent by the global approach. The logical sentences correspond to clauses (disjunctions of literals) with at most three literals. Results on the CPU time (in seconds) and on the number of columns generated are given for 15 different problem sizes. For each size, 30 instances are solved, and the average as well as the standard deviation values are presented. The algorithm is implemented in C and uses CPLEX 8.1 [28] as the linear programming solver. At each iteration of the column generation algorithm, up to 50 columns having a reduced cost of the desired sign can be added to the master problem. All results were obtained on a single processor of an Entreprise 10000 computer with 64 processors, 400 MHz, and 64GB RAM.

The procedure for initializing the parameters of the stabilized column generation algorithm is not the same when the local approach is used before the global one. When this is the case, the estimates $\hat{u}_i$ obtained in step (b) of the merged approach are used to initialize the penalties $\delta^{-}_i$ and $\delta^{+}_i$. Indeed, the interval $[\delta^{-}_i, \delta^{+}_i]$ on the dual variable $u_i$ is initialized to $[\hat{u}_i - 10^{-3}, \hat{u}_i + 10^{-3}]$. When this is not the case, no heuristic solution is available for the initialization of penalties $\delta^{-}_i$ and $\delta^{+}_i$ in the stabilized column generation process. Therefore, the interval $[\delta^{-}_i, \delta^{+}_i]$ on dual variable $u_i$ is initialized to $[-10^{-3}, 10^{-3}]$ to take into account that most dual variables are equal to 0 at the optimum.

The same procedure for updating the parameters of the stabilized column generation algorithm is considered whether or not the local approach is used. The updates are only done when the optimal solution of the current problem (12) is obtained. When a parameter’s update occurs and the current dual variable lies outside its interval, this interval is updated so that its center corresponds to the current value of the dual variable and its length is doubled. The updates of the penalties also exploit an empirical result we have noted: most dual variables have an optimal value that is equal to $-1$, $0$, or $1$. From linear programming sensitivity analysis, this corresponds to a small increase in the right hand side of a constraint of PSAT entailing an equal decrease,
having no influence, or entailing an equal increase in the objective function value, respectively. The value 0 which is the most frequent one implies, barring degeneracy, that the corresponding constraint is not tight at the optimum. Indeed, using the local approach shows that only a few of the rules are applied. If, after updating the same interval several times, the values appear to progress towards one of the values $-1$, 0, or 1, the algorithm will center the interval $[\delta_i^-, \delta_i^+]$ around this value. The bounds $\epsilon_i^-$ and $\epsilon_i^+$, for $i = 0, 1, \ldots, m$, are initialized to $10^{-1}$, and during the update of the parameters, they are decreased progressively to 0.

In a first series of experiments, results of which are presented in Tables 1–3, a comparison is made between the merged approach and the previous global approach, i.e., column generation without any use of the local approach or stabilization.

Table 1 presents detailed step-by-step results for the merged approach, i.e., Algorithm 1. The column Accuracy gives the proportion of the 30 instances for which the solution obtained by the local approach is optimal.

<table>
<thead>
<tr>
<th>Problem size</th>
<th>Step (a)</th>
<th>Step (b)</th>
<th>Step (c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>n m</td>
<td>CPU time (s)</td>
<td>Accuracy</td>
<td>CPU time (s)</td>
</tr>
<tr>
<td>50 50</td>
<td>0.10 0.01</td>
<td>100.0</td>
<td>0.01 0.01</td>
</tr>
<tr>
<td>50 100</td>
<td>0.11 0.01</td>
<td>70.0</td>
<td>0.08 0.01</td>
</tr>
<tr>
<td>50 200</td>
<td>0.17 0.01</td>
<td>53.3</td>
<td>0.11 0.01</td>
</tr>
<tr>
<td>50 400</td>
<td>0.44 0.01</td>
<td>30.0</td>
<td>0.10 0.01</td>
</tr>
<tr>
<td>50 800</td>
<td>0.25 0.01</td>
<td>0.0</td>
<td>0.02 0.01</td>
</tr>
<tr>
<td>100 50</td>
<td>0.12 0.01</td>
<td>96.7</td>
<td>0.09 0.01</td>
</tr>
<tr>
<td>100 100</td>
<td>0.12 0.01</td>
<td>96.7</td>
<td>0.08 0.01</td>
</tr>
<tr>
<td>100 200</td>
<td>0.20 0.01</td>
<td>93.3</td>
<td>0.01 0.01</td>
</tr>
<tr>
<td>100 400</td>
<td>0.43 0.01</td>
<td>83.3</td>
<td>0.01 0.01</td>
</tr>
<tr>
<td>100 800</td>
<td>0.13 0.01</td>
<td>83.3</td>
<td>0.01 0.01</td>
</tr>
<tr>
<td>200 50</td>
<td>0.12 0.01</td>
<td>100.0</td>
<td>0.02 0.01</td>
</tr>
<tr>
<td>200 100</td>
<td>0.19 0.01</td>
<td>96.7</td>
<td>0.01 0.01</td>
</tr>
<tr>
<td>200 200</td>
<td>0.25 0.01</td>
<td>93.3</td>
<td>0.01 0.01</td>
</tr>
<tr>
<td>200 400</td>
<td>0.51 0.01</td>
<td>86.7</td>
<td>0.01 0.01</td>
</tr>
<tr>
<td>200 800</td>
<td>0.11 0.01</td>
<td>86.7</td>
<td>0.01 0.01</td>
</tr>
</tbody>
</table>

Table 2 presents results obtained with the previous global approach.

<table>
<thead>
<tr>
<th>Problem size</th>
<th>CPU time (s)</th>
<th>Nb. cols.</th>
</tr>
</thead>
<tbody>
<tr>
<td>n m</td>
<td>Mean Stdev</td>
<td>Mean Stdev</td>
</tr>
<tr>
<td>50 50</td>
<td>0.50 0.07</td>
<td>313.7 51.11</td>
</tr>
<tr>
<td>50 100</td>
<td>2.85 0.38</td>
<td>652.6 73.92</td>
</tr>
<tr>
<td>50 200</td>
<td>20.37 3.02</td>
<td>1082.8 127.51</td>
</tr>
<tr>
<td>50 400</td>
<td>190.64 30.63</td>
<td>1626.6 161.69</td>
</tr>
<tr>
<td>50 800</td>
<td>2010.73 302.88</td>
<td>2184 199.349</td>
</tr>
<tr>
<td>100 50</td>
<td>0.56 0.07</td>
<td>319.8 43.16</td>
</tr>
<tr>
<td>100 100</td>
<td>3.62 0.48</td>
<td>751.6 96.479</td>
</tr>
<tr>
<td>100 200</td>
<td>26.10 3.12</td>
<td>1499.8 133.274</td>
</tr>
<tr>
<td>100 400</td>
<td>241.90 24.899</td>
<td>2766.2 195.829</td>
</tr>
<tr>
<td>100 800</td>
<td>2660.49 300.12</td>
<td>4054.9 309.804</td>
</tr>
<tr>
<td>200 50</td>
<td>0.76 0.09</td>
<td>326.6 39.63</td>
</tr>
<tr>
<td>200 100</td>
<td>4.13 0.49</td>
<td>739.8 73.192</td>
</tr>
<tr>
<td>200 200</td>
<td>30.60 3.78</td>
<td>1715.8 176.912</td>
</tr>
<tr>
<td>200 400</td>
<td>267.13 24.499</td>
<td>3886.2 306.249</td>
</tr>
<tr>
<td>200 800</td>
<td>2651.48 246.11</td>
<td>6965.6 428.901</td>
</tr>
</tbody>
</table>
In all tables, the CPU times reported are in seconds and the two columns called Nb. Cols gives statistics on the number of columns generated by the column generation algorithm. It appears that

(i) In step (a), the AD-PSAT heuristic finds often, but not always, the optimal solution;
(ii) The probability of finding this optimal solution decreases with the ratio \( n/m \) of variables to logical sentences. The extreme case is for \( n = 50 \) and \( m = 800 \) where the optimal solution is never found;
(iii) Computation times of step (a) are very small and never exceed a quarter of a second;
(iv) The small linear program of step (b) is very easy to solve, regardless of \( n \) and \( m \). The number of columns generated is about 3 and the computing time is less than one fiftieth of a second;
(v) The overwhelming part of the computing time is taken up by step (c), i.e., using stabilized column generation to find the optimal solution and prove its optimality;
(vi) Computing times appear not to be very sensitive to the number \( n \) of variables but augment rapidly with the number \( m \) of logical sentences;
(vii) The number of columns generated increases, but less than linearly, with the number \( m \) of logical sentences.

Results obtained by the previous global approach are presented in Table 2. It appears that

(i) The average computing times of the previous global approach are always larger than those of the merged approach;
(ii) The same appears to be true for the number of columns generated.

The comparison of the CPU times between the merged approach and the previous global approach is refined in Table 3. For each problem size, all cases (columns All) are split into two: those for which the local approach does find the optimal solution (column Op.) and those for which it does not (column Non-op.). It appears that

(i) The average time saving is about 34.1%, which is fairly substantial;
(ii) For each value of \( n \), this average saving increases with \( m \) to reach a maximum of 53.2% (there is one exception: the case \( n = 50 \) and \( m = 800 \) for which the local approach never finds the optimal solution and where an average time loss of 3.1% is observed);
(iii) The average time savings are more substantial when the local approach finds the optimal solution: the average is then 39.5% and the maximum is 57.4%;

(iv) Conversely, time savings are smaller when the local approach does not find the optimal solution: the average is then 16.3% (note however the large influence of the case \( n = 50 \) and \( m = 800 \)) and the maximum is 43.6%. In such a case, the first estimates of the optimal dual variables obtained in step (b) of Algorithm 1 are bad, and the dual variables take more iterations to stabilize to their optimal value.

A second series of experiments aims at finding which part of the time savings is due to the local approach and which part is due to the stabilization. Therefore, in Table 4, the instances are solved through stabilized column generation without the use of the local approach to initialize the parameters. It appears that

(i) The average computing times and the number of columns generated of the stabilized column generation without the use of the local approach are always larger than those of the merged approach;

(ii) In most cases, the average computing times and the number of columns generated of the stabilized column generation without the use of the local approach are smaller than those of the previous global approach.

The comparison of the CPU times of the merged approach and the stabilized global approach without the use of the local approach is refined in Table 5. In fact, the local approach is used here, but only to discriminate cases in which AD-PSAT finds the optimal solution and cases when it does not. It appears that

(i) The average time saving is 20.6%, which is smaller than the average saving obtained when comparing the merged approach to the previous global approach.

(ii) For each value of \( n \), this average saving increases with \( m \) to reach a maximum of 57.9%.

(iii) The average time saving is smaller for instances for which the local approach finds the optimal solution than otherwise: the average is 16.1% compared to 35.8%. However, the average time saving is, for some problem sizes, larger for instances for which the local approach finds the optimal solution.

(iv) In some cases, there is a time loss instead of a time saving for instances for which the local approach does not find the optimal solution.

So both the local approach and the stabilization of column generation contribute to the average time saving.

Table 4
Results obtained with the stabilized global approach without the use of the local approach

<table>
<thead>
<tr>
<th>Problem size</th>
<th>CPU time (s)</th>
<th>Nb. cols.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Stdev</td>
</tr>
<tr>
<td>( n ) ( m )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50 50</td>
<td>0.436</td>
<td>0.052</td>
</tr>
<tr>
<td>50 100</td>
<td>2.405</td>
<td>0.38</td>
</tr>
<tr>
<td>50 200</td>
<td>18.49</td>
<td>3.31</td>
</tr>
<tr>
<td>50 400</td>
<td>267.395</td>
<td>153.5</td>
</tr>
<tr>
<td>50 800</td>
<td>8308.253</td>
<td>6921.047</td>
</tr>
<tr>
<td>100 50</td>
<td>0.497</td>
<td>0.057</td>
</tr>
<tr>
<td>100 100</td>
<td>2.626</td>
<td>0.208</td>
</tr>
<tr>
<td>100 200</td>
<td>17.991</td>
<td>2.095</td>
</tr>
<tr>
<td>100 400</td>
<td>188.364</td>
<td>46.641</td>
</tr>
<tr>
<td>100 800</td>
<td>3843.712</td>
<td>5813.576</td>
</tr>
<tr>
<td>200 50</td>
<td>0.562</td>
<td>0.056</td>
</tr>
<tr>
<td>200 100</td>
<td>2.68</td>
<td>0.254</td>
</tr>
<tr>
<td>200 200</td>
<td>19.21</td>
<td>4.438</td>
</tr>
<tr>
<td>200 400</td>
<td>167.01</td>
<td>26.146</td>
</tr>
<tr>
<td>200 800</td>
<td>2223.978</td>
<td>1570.884</td>
</tr>
</tbody>
</table>
4.3. Contribution of the global approach

Since the local approach finds a heuristic solution, there is no guarantee that this solution is optimal. By applying the global approach after the local one, it is possible (i) to confirm the optimality of the heuristic solution or (ii) to find the optimal one when the local approach fails to do so. It is also possible (iii) to show that the problem is infeasible. In case (i), the local approach provides a full justification of the optimal values obtained, which can be followed step-by-step by the user. We next consider an example from [14] in which the optimal solution was not found by the anytime-deduction heuristic proposed there.

Example 3 (Example 4 of [14]). Find the best possible bounds on the probability of \( S_7 \equiv x_2 \land x_3 \) subject to

\[
\begin{align*}
\text{prob}(S_1 \equiv x_1) & \in [0.6, 1] \quad (18) \\
\text{prob}(S_2 \equiv \bar{x}_1 \lor x_2) & \in [0.8, 0.9] \quad (19) \\
\text{prob}(S_3 \equiv \bar{x}_1 \lor x_3) & \in [0.9, 1] \quad (20) \\
\text{prob}(S_4 \equiv \bar{x}_2 \lor x_4) & \in [0.5, 0.8] \quad (21) \\
\text{prob}(S_5 \equiv \bar{x}_3 \lor x_4) & \in [0.8, 0.9] \quad (22) \\
\text{prob}(S_6 \equiv x_4) & \in [0.0, 0.2]. \quad (23)
\end{align*}
\]

Using the rules presented in [14], and there referred to by roman numerals, one can show the inconsistency of these probability intervals. Applying rule (xx) to (18) and (20), we obtain

\[
\text{prob}(x_3) \in [0.5, 1.0]. \quad (24)
\]

Applying rule (xxi) to (22) and (23), we obtain

\[
\text{prob}(x_3) \in [0.1, 0.4]. \quad (25)
\]

Applying rule (xvii) to (24) and (25), we obtain that \( 0.5 \leq \text{prob}(x_3) \leq 0.4 \), which proves the inconsistency of the system. Frisch and Haddawy did not detect this inconsistency since they did not apply rules to tighten the
probability interval on $x_3$ to obtain best bounds on the probability of $S_7$. AD-PSAT detects this inconsistency because it applies rules to tighten probability intervals on variables even if they do not belong to the logical sentence $S_{m+1}$. In this case, AD-PSAT updates bounds on each variable of the problem since each variable appears in $S_{m+1}$ or is related directly or indirectly to variables in $S_{m+1}$. This example illustrates the importance of using a powerful heuristic in the merged approach in order to avoid recourse to the much slower global approach.

5. Conclusions

Merging the local and global approaches to PSAT is beneficial to both. One may distinguish four cases:

(i) PSAT is inconsistent and the local approach detects this. This is done very quickly compared to using the global approach (less than 1% of the time spent by the global approach). Moreover, a detailed argument proving inconsistency is provided.

(ii) PSAT is inconsistent but the local approach does not detect this. The global approach will prove inconsistency and show that the conclusions of the local approach, although logically correct, should not be relied upon. Note that this case appears to be rare and was not observed to date with AD-PSAT.

(iii) PSAT is consistent and the local approach finds the optimal solution. Then the global approach is applied with best possible estimates of the optimal values of the dual variables. For all instances tested in this paper for which this case occurs, computing times are reduced by 39.5% on average. Moreover, a complete justification of the tightest bounds entailed is available.

(iv) PSAT is consistent but the local approach did not find the optimal solution. Then the global approach is applied with poor estimates of the optimal values of the dual variables. Computing times are slightly reduced by comparison with the standard column generation. For all instances tested in this paper for which this case occurs, computing times are reduced by 16.3% on average. Such examples could be further analyzed in view of devising new rules and so improving the local approach.

This appears to be the current state-of-the-art for solving PSAT problems.

Finally, the merged approach can be extended to problems with conditional probabilities by using local rules for such problems [34,35] and expressing conditional PSAT problems as linear programs [22,44].

Acknowledgements

The research of the first author was supported by the NSERC (Natural Sciences and Engineering Research Council of Canada) Grant 105574-02 and the FCAR (Fonds pour la Formation de Chercheurs et l’Aide à la Recherche) Grant 2002-ER-73226. The research of the second author was supported by the NSERC graduate scholarship 195113 and the FCAR graduate scholarship 67567. We thank Anderson Delcio Parreira for the use of his implementation of the algorithm AD-PSAT presented in [39]. We also thank William Milnes for improvements in the paper’s presentation and two anonymous reviewers for their many constructive remarks and suggestions.

References
