Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Algorithms for Deciding Counting Quantifiers over Unary Predicates

Marcelo Finger
Department of Computer Science
University of Sao Paulo, Brazil

Glauber De Bona
Department of Computer Science
University College London, UK

Abstract

We study algorithms for fragments of first order logic extended with counting quantifiers, which are known to be highly complex in general. We propose a fragment over unary predicates that is NP-complete and for which there is a normal form where Counting Quantification sentences have a single Unary predicate, thus call it the CQU fragment. We provide an algebraic formulation of the CQU satisfiability problem in terms of Integer Linear Programming based on which two algorithms are proposed, a direct reduction to SAT instances and an Integer Linear Programming version extended with a column generation mechanism. The latter is shown to lead to a viable implementation and experiments shows this algorithm presents a phase transition behavior.

1 Introduction

The need to combine deductive reasoning with counting and cardinality capabilities in a principled way has been recognized since the beginning of automated reasoning. However, the complexity of the resulting systems has perhaps precluded a wider presence of deductive systems that provide those counting capabilities.

The basic approach for adding counting capabilities extends first-order logic with some Lindström-type generalized quantifiers (Lindström 1966) that could express at least the counting notions of "there are at least/most n elements with property P". The satisfiability of a logic with that expressivity, but limited to fragments with at most binary predicates was first shown to be decidable (Grädel and Otto 1999) and was later shown in NEXPTIME (Pratt-Hartmann 2005) with an EXPTIME-hard lower bound (Baader, Buchheit, and Hollander 1996); related studies are found in (Martin, Madelaine, and Stacho 2015; Bulatov and Hedayaty 2015).

If the attention is reduced to counting quantifiers over unary predicates, the decision problem was shown to be "only" NP-complete, even when restricted only to a fragment called Syllogistic Logic (Pratt-Hartmann 2008). However, the algorithm presented for that complexity proof was intrinsically non-deterministic, with no clear reduction to known feasible algorithms.

This paper aims at presenting an expressive fragment of first-order logic with counting quantifiers over unary predicates called CQU, developing decision algorithms for it and providing an effective implementation using open-source software. This fragment is more expressive than Syllogistic Logic and bears considerable similarity in format to that of Probabilistic Satisfiability (PSAT) problems (Nilsson 1986), an NP-complete problem (Georgakopoulos, Kavvadias, and Papadimitriou 1988) for which there has been considerable algorithmic development in the literature (Hansen and Jaumard 2000; Finger and De Bona 2011; 2015). The algorithms rely on an Integer Linear Programming presentation of CQU satisfiability; a viable implementation is obtained with a SAT-based column generation strategy.

We start by describing an example of reasoning on the CQU fragment.

Example 1 Consider the following statements about people's hobbies

(i) At most 15 people are astronomers, ball players, choir singers or dancers
(ii) At least 12 astronomers are not dancers
(iii) At least 10 astronomers play ball or sing
(iv) All ball players dance
(v) At least 7 astronomers sing in a choir
(vi) At most 6 astronomers sing in a choir

We would like to reason that (i)–(iv) are satisfiable and that from them one can infer (v) or, equivalently, as (vi) is the negation of (v), that (i)–(iv) and (vi) are unsatisfiable. □

An implementation of a CQU decision procedure should allow us to solve problems like the one in Example 1 and, furthermore, it should also allows us to empirically investigate if a *phase-transition phenomenon* occurs for CQU decision problems. Cheeseman *et al.* (Cheeseman, Kanefsky, and Taylor 1991) conjectured that a phase-transition phenomenon is a property of all NP-complete problems. Hard and easy instances of SAT distributions problems were described by (Mitchell, Selman, and Levesque 1992). Gent and Walsh (Gent and Walsh 1994) described a phase transition for SAT problems dependent on the rate m/n, where m is the number of clauses in a 3-SAT instance and n is the number of variables, describing that the hardest instances concentrate around the *phase transition point*, at which the

number of expected satisfiable instances is 50%, which for 3-SAT occurs at $m/n \approx 4.3$.

To the best of our knowledge, no phase transition has been described for counting quantifiers so far. We implement a CQU-SAT solver using open-source software, which we plan to make publicly available, on which phase-transition detection experiments can be done.

The rest of this paper is organized as follows. The CQU fragment is defined in Section 2, formalizing the satisfiability problem and presenting an algebraic formulation for it. Two algorithms are presented in 3.1 and their basic properties are studied. The implementation, experiments and results are described in 4. We terminate with conclusion and suggestions for further works.

2 Counting Quantifiers over Unary Predicates

We deal with a function-free first-order fragment over a signature containing only unary predicates and constants, extended with explicit counting quantifiers $\exists_{\leq n}$ (atmost n) and $\exists_{\geq n}$ (at least n), where $n \in \mathbb{N}$ is a non-negative integer.

We consider two forms of sentences. A *counting sentence* has the form $\exists_{\leq n} x\, \varphi(x)$ or $\exists_{\geq n} x\, \varphi(x)$; $\varphi(x)$ is a Boolean combination of unary atomic formulas $p(x), q(x)$, etc. A *universal sentence* has the form $\forall x \varphi(x)$, where $\varphi(x)$ is restricted as in counting sentence. A formula φ of the fragment of counting quantifiers over unary predicates (CQU), is a conjunction of any finite number of counting sentences $\mathcal{Q}$ and universal sentences $\mathcal{U}$, $\varphi = \langle \mathcal{Q}, \mathcal{U} \rangle$.

The semantics is the usual one, briefly presented here. Let V be a countable set of variables, let D be a non-empty set (domain). Let a *term* denote a constant or a variable. Consider an interpretation $\mathcal{I}$ such that for every term $t, \mathcal{I}(t) \in D$ and for every unary predicate symbol p, $\mathcal{I}(p) \subseteq D$; we write $\mathcal{I}_{|x}$ for an interpretation that agrees with $\mathcal{I}$ for any term $t \neq x$. Let φ be a CQU-formula; we write $D, \mathcal{I} \models \varphi$ to indicate that φ is satisfiable over domain D and interpretation $\mathcal{I}$, defined as

$$
\begin{array}{ll}
D, \mathcal{I} \models p(t) & \textit{iff } \mathcal{I}(t) \in \mathcal{I}(p) \\
D, \mathcal{I} \models \neg\varphi & \textit{iff } D, \mathcal{I} \not\models \varphi \\
D, \mathcal{I} \models \varphi \wedge \psi & \textit{iff } D, \mathcal{I} \models \varphi \text{ and } D, \mathcal{I} \models \psi \\
D, \mathcal{I} \models \exists_{\leq n} x\, \varphi & \textit{iff } |\{\mathcal{I}_{|x}(x) \in D | D, \mathcal{I}_{|x} \models \varphi\}| \leq n \\
D, \mathcal{I} \models \exists_{\geq n} x\, \varphi & \textit{iff } |\{\mathcal{I}_{|x}(x) \in D | D, \mathcal{I}_{|x} \models \varphi\}| \geq n
\end{array}
$$

A formula φ is *satisfiable* if there are D and $\mathcal{I}$ such that $D, \mathcal{I} \models \varphi$; otherwise it is unsatisfiable. A formula φ *entails* ψ ($\varphi \models \psi$) iff $\varphi \wedge \neg\psi$ is unsatisfiable. φ is logically equivalent to ψ ($\varphi \equiv \psi$) iff $\varphi \models \psi$ and $\psi \models \varphi$. The problem of determining whether a given formula is satisfiable is called CQU-SAT.

Other Boolean connectives are defined in the usual way. The semantics above shows that the negation of counting quantifiers is such that $\neg\exists_{\leq n} x\, \varphi \equiv \exists_{\geq n+1} x\, \varphi$ and $\neg\exists_{\geq n+1} x\, \varphi \equiv \exists_{\leq n} x\, \varphi$. If we define the usual existential quantifier as $\exists x \varphi \equiv \exists_{\geq 1} x\, \varphi$ it follows that $\forall x \varphi \equiv \exists_{\leq 0} x\, \neg\varphi$. The exact counting quantifier is defined as $\exists_{=n} x\, \varphi \equiv \exists_{\leq n} x\, \varphi \wedge \exists_{\geq n} x\, \varphi$.

The fragment which allows for the free application of syntactic rules without the separation between universal and quantified sentences is called $\mathcal{C}^1$ and was investigated in (Pratt-Hartmann 2008). The difference between $\mathcal{C}^1$ and CQU lies in the fact that the former allows for disjunctions between quantified formulas, such as $\exists_{\geq 7} x\, \varphi \vee \exists_{\leq 9} y\, \psi$, which is not allowed in CQU. The following result follows from the study of $\mathcal{C}^1$.

Proposition 1 ***(Pratt-Hartmann 2008)*** *The CQU fragment has the finite model property. Moreover, the satisfiability problem for it is strongly NP-complete.* □

Strong NP-completeness means that NP-completeness holds even when n in $\exists_{\leq n}, \exists_{\geq n}$ is given in unary notation. The proof of Proposition 1 is based on non-deterministic manipulations of the formula and does not provide an effective (non-exponential) algorithm. We note that the fragment described above contains Syllogistic Logic (Pratt-Hartmann 2008), which contains counting quantifiers over only a conjunction of two literals over unary predicates. In the search for feasible algorithms we propose a normal form for formulas in the CQU fragment.

Definition 1 A CQU formula $\varphi = \{\mathcal{Q}, \mathcal{U}\}$ is in *normal form* if the quantified sentences in $\mathcal{Q}$ are restricted to atomic unary predicates, that is, quantified formulas are of the form $\exists_{\leq n} x\, p(x)$ or $\exists_{\geq n} x\, p(x)$, where p is an atomic unary predicate. □

In the following we will write $\bowtie$ to refer to $\leq$ or $\geq$, so the CQU normal form is characterized by counting quantifier sentences of the form $\exists_{\bowtie n} x\, p(x)$. By adding a small number of extra variables, any CQU formula can be brought to normal form.

Lemma 1 *For every CQU formula φ there exists a normal form formula φ' such that φ is a satisfiable iff φ' is; the atomic instance φ' can be built from φ in polynomial time.* □

PROOF Consider $\varphi = \langle \mathcal{Q}, \mathcal{U} \rangle$. We build $\varphi' = \langle \mathcal{Q}', \mathcal{U}' \rangle$ starting with $\mathcal{Q}' = \varnothing$ and $\mathcal{U}' = \mathcal{U}$. Then, for every quantified formula $\exists_{\bowtie n} x\, \psi$, if ψ is an atomic predicate, just add $\exists_{\bowtie n} x\, \psi$ to $\mathcal{Q}'$; otherwise, create a new unary predicate p_{new} and add $\forall x (p_{\text{new}}(x) \leftrightarrow \psi)$ to $\mathcal{U}'$ and $\exists_{\bowtie n} x\, p_{\text{new}}(x)$ to $\mathcal{Q}'$; at every step φ' is in normal form, and at its end, by contruction, φ' is satisfiable iff φ is. ■

Example 2 Consider Example 1, which can be formalized as follows:

(i) $\exists_{\leq 15} x\, (a(x) \vee b(x) \vee c(x) \vee d(x))$
(ii) $\exists_{\geq 12} x\, (a(x) \wedge \neg d(x))$
(iii) $\exists_{\geq 10} x\, (a(x) \wedge (b(x) \vee c(x))$
(iv) $\forall x (b(x) \rightarrow d(x))$
(v) $\exists_{\geq 7} x\, (a(x) \wedge c(x))$
(vi) $\exists_{\leq 6} x\, (a(x) \wedge c(x))$

Clearly, (vi) is the negation of (v); we use only the latter. To place counting formulas in normal form, we introduce 4 new predicates, p_1, p_2, p_3, p_4. Let $\mathcal{U} = \{\forall x (p_1(x) \leftrightarrow (a(x) \vee b(x) \vee c(x) \vee d(x))), \forall x (p_2(x) \leftrightarrow a(x) \wedge \neg d(x)), \forall x (p_3(x) \leftrightarrow (a(x) \wedge (b(x) \vee c(x)))), \forall x (b(x) \rightarrow d(x)), \forall x (p_4(x) \leftrightarrow (a(x) \wedge c(x)))\}$ so that we can have counting quantification over unary predicates only; let $\mathcal{Q} =$

$\{\exists_{\leq 15} x\, p_1(x), \exists_{\geq 12} x\, p_2(x), \exists_{\geq 10} x\, p_3(x)\}$, such that we expect $\langle \mathcal{Q}, \mathcal{U}\rangle$ to be satisfiable and $\langle \mathcal{Q} \cup \{\exists_{\leq 6} x\, p_4(x)\}, \mathcal{U}\rangle$ to be unsatisfiable. □

For the rest of this work we always assume that formulas are in normal form, which allow for a convenient presentation of CQU satisfiability in terms of integer linear algebra.

2.1 Algebraic Formulation of CQU-SAT

Consider a normal form CQU formula $\varphi = \langle \mathcal{Q}, \mathcal{U}\rangle$ whose satisfiability we want to determine. Consider the $k = |\mathcal{Q}|$ unary predicates that are quantified in $\mathcal{Q}$, $p_1(x), \ldots, p_k(x)$, where the order is fixed; then there are 2^k *elementary terms* of the form $e(x) = \lambda_1(x) \wedge \ldots \wedge \lambda_k(x)$, where each $\lambda_i(x)$ is either $p_i(x)$ or $\neg p_i(x)$; an elementary term $e(x)$ is called *coherent* if it is consistent with the universal sentences, that is, the set $\{\exists x e(x)\} \cup \mathcal{U}$ has a model.

Semantically, each elementary term is interpreted as a subset E of the domain D, $E = L_1 \cap \ldots \cap L_k$, where each L_i is either the interpretation of p_i or its complement with respect to the domain D. In any interpretation that satisfies $\varphi = \langle \mathcal{Q}, \mathcal{U}\rangle$, only coherent elementary terms may be interpreted as non-empty subsets, otherwise the interpretation would contradict $\mathcal{U}$.

An integer linear algebraic presentation of CQU-SAT follows from encoding elementary terms as a $\{0,1\}$-vector e of size k, in which $e_i = 1$ if in the term $e(x)$ λ_i is p_i and $e_i = 0$ otherwise. We consider only the set of coherent elementary terms so let $k_m \leq 2^k$ be the number of coherent elementary terms. Let A be a $k \times k_m$ $\{0,1\}$-matrix, where each column encodes a coherent elementary term; note that the ith line corresponds to the i counting quantifier expression in $\mathcal{Q}$. Let the ith element in $\mathcal{Q}$ be $\exists_{\bowtie_i n_i} x\, p_i(x)$, $\bowtie_i \in \{\leq, \geq\}$; let b be $k \times 1$ integer vector, such that $b_i = n_i$, and let x be a $k_m \times 1$ vector of integer variables. Then the (potentially exponentially large) integer linear system that corresponds the CQU-SAT problem $\varphi = \langle \mathcal{Q}, \mathcal{U}\rangle$ is:

$$\begin{array}{rcl} Ax & \bowtie & b \\ x & \geq & 0 \\ x_j & & \text{integer} \end{array} \qquad (1)$$

Lemma 2 *A normal form $\varphi = \langle \mathcal{Q}, \mathcal{U}\rangle$ is CQU satisfiable iff its corresponding system given by* (1) *has a solution.* □

PROOF ($\Rightarrow$) If φ has an interpretation, we instantiate x_j with the number of elements in the subset corresponding to the jth coherent elementary term; clearly x_j is a non-negative integer. As all elements in $\mathcal{Q}$ are satisfied, all inequalities in $Ax \bowtie b$ must be satisfied.

($\Leftarrow$) If system (1) has a solution, we construct a finite interpretation by inserting x_j elements in each subset corresponding to a coherent elementary term. From that point we can compute an interpretation for all predicates in $\mathcal{Q}$, and as all inequalities in (1) are satisfied, so is $\mathcal{Q}$; furthermore, as only coherent elementary terms have non-zero elements, $\mathcal{U}$ is also satisfied. ■

To determine if an elementary term is coherent, we transform the set $\mathcal{U}$ into a propositional formula, by deleting the external $\forall x$ quantifiers and considering each unary predicate $p(x)$ as a propositional symbol p. Then we can generate all satisfying Boolean assignments using any SAT solver (Biere 2014; Eén and Sörensson 2003), and consider the part of each assignment corresponding to the predicates in $\mathcal{Q}$, as illustrated in the following example.

Example 3 Consider the normal form formula $\varphi = \langle Q, U\rangle$ presented in Example 2. The linear algebraic rendering of the problem shows it is CQU satisfiable:

$$\begin{array}{lc} \mathbf{p_1} & \left[\begin{array}{ccc} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{array}\right] \cdot \left[\begin{array}{c} 10 \\ 2 \\ 0 \end{array}\right] \begin{array}{c} \leq \\ \geq \\ \geq \end{array} \begin{array}{c} 15 \\ 12 \\ 10 \end{array} \\ \mathbf{p_2} & \\ \mathbf{p_3} & \\ \hline \mathbf{a} & \begin{array}{ccc} 1 & 1 & 0 \end{array} \\ \mathbf{b} & \begin{array}{ccc} 0 & 0 & 0 \end{array} \\ \mathbf{c} & \begin{array}{ccc} 1 & 0 & 1 \end{array} \\ \mathbf{d} & \begin{array}{ccc} 0 & 0 & 0 \end{array} \end{array}$$

Each line corresponds to a predicate, indicated on the left. The first 3 lines correspond to the quantified restrictions in $\mathcal{Q}$, and the product of the matrix by the vector satisfies the corresponding inequalities. Each of the first 3 $\{0,1\}$-columns of size 7 correspond to a $\mathcal{U}$-satisfying valuation of all predicates. Thus this solution shows that the first four conditions of Examples 1 and 2 are satisfiable.

However, if we want to show that adding the last condition makes the example inconsistent, we would have to consider the 2^4 valuations over the predicates $p_1, \ldots, p_4$ and show that no combination is capable of satisfying 4 inequalities, a clearly exponential problem. □

The example shows that there is the potential of an exponentially sized search for solutions for CQU-SAT. In the search for effective algorithms, we need a guarantee that all satisfiable CQU formulas have polynomial-sized models. In the case of Probabilistic Satisfiability (PSAT), this existence of small models is guaranteed via Caratheodory's Theorem (Eckhoff 1993). In the discrete case, we have the following anologue.

Proposition 2 *Let $\mathcal{E}$ be system of the form* (1) *that has a solution. Then $\mathcal{E}$ has a solution over $\mathbb{N}$ with at most $\frac{5}{2} k \log k + 1$ non-zero entries.* □

This result is proven by (Pratt-Hartmann 2008) adapting a result by (Eisenbrand and Shmonin 2006). It gives us a polynomial-sized bound for models of satisfiable CQU-SAT instances, as follows.

A universal formula over unary predicates $\forall x \psi(x)$ can be *seen as a propositional formula ψ'*, obtained by deleting $\forall x$ and replacing each unary predicate $p(x)$ by a corresponding propositional atom p'. Let $\mathcal{U}$ be seen as a set of propositional formulas, $\{0,1\}$-matrix A be as in (1); A's jth column A^j is *$\mathcal{U}$-satisfying* if there is a valuation satisfying $\mathcal{U}$ that, when applied to $\mathcal{Q}$'s predicates seen as propositions, produces A^j. Lemma 2 and Proposition 2 yield the following.

Lemma 3 *Consider a normal form CQU-SAT instance $\varphi = \langle \mathcal{Q}, \mathcal{U}\rangle$, where $\mathcal{Q} = \{\exists_{\bowtie_i b_i} x\, p_i(x) | 1 \leq i \leq k\}$, and $\mathcal{U}$ is seen as a set of propositional restrictions. Then φ is satisfiable iff there exists a solvable system of inequalities of the form*

$$A_{k \times k_m} \cdot x_{k_m \times 1} \bowtie b_{k \times 1} \qquad (2)$$

where $k_m = \left\lceil \frac{5}{2}(k \log k + 1) \right\rceil$*, A is a* $\{0,1\}$*-matrix whose columns are* $\mathcal{U}$*-satisfying.* ■

This serves as a basis for effective algorithms for CQU-SAT.

3 Algorithms

We present two algorithms for deciding the satisfiability of normal form CQU-SAT instances based on Lemma 3.

3.1 Canonical Reduction to SAT

The polynomial-size format of solutions given by Lemma 3 provides a way to reduce an instance of CQU-SAT. In this approach, each instance $\varphi = \langle \mathcal{Q}, \mathcal{U} \rangle$ of a CQU-SAT decision problem is polynomially translated to an instance of SAT by encoding the set of inequalities in (2). In the following, $\mathcal{U}$ is seen as a set of classical propositional restrictions.

Consider the set of inequalities in $\mathcal{Q}$, represented in (2) by $\bowtie\ b$, where b is a vector of size k containing the upper and lower limits defined by the k-sized vector of inequality symbols $\bowtie$. We search for a $\{0,1\}$-matrix $A_{k \times k_m}$ whose columns, according to Lemma 3, are $\mathcal{U}$-satisfying valuations. Similarly, we encode x as a vector of size k_m and b as a vector of size k of positions consisting of a fixed number in binary positional system with y_b bits, as in (3).

$$x_j = \boxed{x_{j,1}} \boxed{\cdots} \boxed{x_{j,y_b}} \qquad b_i = \boxed{b_{i,1}} \boxed{\cdots} \boxed{b_{i,y_b}} \tag{3}$$

Linear programming guarantees that, if there is a solution to (2), then there is a solution x in which $x_j \leq \max(b_i)$ for all j; so we fix $y_b = \lceil \log_2(\max b_i + 1) \rceil$ as the number of bits that encode the largest element of b. According to (Warners 1998), each linear inequality can be brought to a conjunction of conjunctive normal form formulas in linear time in $k_m \times y_b$. Thus, this enconding is polynomial in k and in the number of bits needed to represent b. We have thus generated a *CQU-SAT-to-SAT encoding*.

Lemma 4 *The formula obtained by the CQU-SAT-to-SAT encoding is satisfiable iff the the input CQU-SAT formula* $\varphi = \langle \mathcal{Q}, \mathcal{U} \rangle$ *is.* □

PROOF Just notice that the CQU-SAT-to-SAT formula encodes a direct solution to (2), and by Lemma 3 this solution exists iff φ is CQU satisfiable. ■

However, as discussed in (Finger and De Bona 2011), the number of variables in the CQUSAT to SAT enconding is $O(y_b \cdot k^2 \cdot \log k)$, in analogy to the reduction of probabilistic satisfiability to SAT. As a result, experiments with parameters as low as the number of inequalities $k = 6$ and the number of variables in $\mathcal{U}$ $n = 40$ are almost impractical around the critical points. This is why we turn to other algorithms for CQU-SAT solving.

3.2 CQU-SAT via Integer Linear Programming

The formulation of CQU-SAT for the decision of $\varphi = \langle \mathcal{Q}, \mathcal{U} \rangle$ given by (1) is apparently suited for solving via Integer Linear Programming (ILP), as it deals with the existence of a solution of a set of inequalities of the form $Ax \bowtie b$, where $x_j \in \mathbb{N}$. However, there are two peculiarities in (1) that have to be addressed, namely

- We do not produce A explicitly.
- If we construct it, it may be exponentially large.

In fact, A's columns consists of $\mathcal{U}$-satisfying valuations, which are costly to compute and there may be exponentially many, e.g. when $\mathcal{U} = \varnothing$. To avoid these problems, we propose to solve the ILP problem via a simplified version of the *branch-and-bound* algorithm (Schrijver 1986), which solves relaxed (continuous) linear programs. For the latter, we generate A's column as needed, in a process known as *column generation* (Jaumard, Hansen, and Poggi de Aragão 1991). For an ILP of the form (1), it is not necessary to search for an optimal integer solution, one only needs to find one feasible integer solution or show none exists.

The top level of the *CQUBranchAndBound* method is shown in Algorithm 3.1. It starts with a unary set of problems containing the input CQU formula, and it loops until either a feasible integer solution to the corresponding linear algebraic problem given by (2) is found or the set of problems becomes empty, in which case unsatisfiability was determined. In the main loop (lines 3–15), we first heuristically remove one problem from the set of problems (line 4, and solve its *relaxed version*, that is, the problem without the restriction of solutions being integral. The heuristic implemented looks for the problem whose current solution has the least number of non-integral components.

If the relaxed problem has no solution, the problem is just removed from the set and the next iteration starts. If there is a solution that is integral, the problem is satisfiable and the loop ends. Otherwise, a solution with at least one non-integral variable exists. A second heuristic is used to find

Algorithm 3.1 CQUBranchAndBound(φ)

Input: A normal form CQU formula $\varphi = \langle \mathcal{Q}, \mathcal{U} \rangle$.
Output: A solution satisfying (2); or "No", if unsatisfiable.

```
1:  CQUSet = {φ}
2:  SAT = false
3:  while not SAT and CQUSet is not empty do
4:      CQUProblem = RemoveHeuristically(CQUSet)
5:      solution = SolveRelaxedViaColGen(CQUProblem)
6:      if no solution then
7:          continue
8:      else if integral solution then
9:          SAT = true
10:     else
11:         var = choseBranchVar(solution)
12:         newCQUs = boundedProblems(CQUProblem, var)
13:         CQUSet = CQUSet ∪ newCQUs
14:     end if
15: end while
16: if SAT then
17:     return solution
18: else
19:     return "No"
20: end if
```

Algorithm 3.2 *SolveRelaxedViaColGen*(φ)

Input: A normal form CQU formula $\varphi = \langle \mathcal{Q}, \mathcal{U} \rangle$.
Output: A relaxed solution, if it exists; or "No", if unsatisfiable.

1: $A_{(0)} = I$; compute cost vector $c^{(0)}$; $x^{(0)} = b$
2: **for** $s = 0$; $c^{(s)\prime} \cdot x^{(s)} > 0$; $s{+}{+}$ **do**
3: $z^{(s)} = \mathit{DualSolution}(A_{(s)}, \bowtie b, c^{(s)})$
4: $y^{(s)} = \mathit{GenerateColumn}(z, \mathcal{U})$
5: **return** "No" **if** column generation failed
6: $A_{(s+1)} = \mathit{merge}(A_{(s)}, y^{(s)})$
7: $c^{(s+1)} = \mathit{append}(c^{(s)}, 0)$
8: **end for**
9: **return** $A_{(s)}$, $x^{(s)}$ such that $A_{(s)} x^{(s)} \bowtie b$

a variable x_i with a non-integral solution z_i on which to branch (line 11). This heuristic chooses x_{i^*} for which the non-integral z_{i^*} is closer to either $\lfloor z_{i^*} \rfloor$ or $\lceil z_{i^*} \rceil$.

Usually, two new bounded problems $\varphi' = \langle \mathcal{Q}', \mathcal{U}' \rangle$, $\varphi'' = \langle \mathcal{Q}'', \mathcal{U}'' \rangle$ are generated (line 12), with the creation of a new unary predicate p_{new}. We make $\mathcal{U}' = \mathcal{U}'' = \mathcal{U} \cup \{\forall x (p_{\text{new}}(x) \leftrightarrow e_{i^*}(x))\}$, where $e_{i^*}(x)$ is the elementary term corresponding to column i^* and $\mathcal{Q}' = \mathcal{Q} \cup \{\exists_{\leq \lfloor z_{i^*} \rfloor} x\, p_{\text{new}}(x)\}$ and $\mathcal{Q}'' = \mathcal{Q} \cup \{\exists_{\geq \lceil z_{i^*} \rceil} x\, p_{\text{new}}(x)\}$. These new formulas φ' and φ'' are then dealt with as integer linear problems of a larger number of rows and columns. However, if the size of the generated node exceeds that given by Lemma 3, the problem is pruned and not inserted.

The largest part of the processing in the *CQUBranchAndBound* algorithms occurs in the calls to the relaxed solver (line 5), *SolveRelaxedViaColGen*(φ), in which column generation takes part. This process takes as input a CQU formula, eventually expanded by the bounding operation and is described in Algorithm 3.2. Its output may contain some non-integral values, but the objective function, which minimizes the cost of the solution, has to be 0 for success to be achieved. Thus the *SolveRelaxedViaColGen*(φ) aims at solving the following linear program (Bertsimas and Tsitsiklis 1997):

$$\begin{array}{ll} \text{minimize} & c' \cdot x \\ \text{subject to} & A \cdot x \bowtie b \text{ and } x \geq 0 \end{array} \tag{4}$$

In the linear program (4), $\{0,1\}$-matrix A's columns consist of all possible valuations and it has 2^k columns, and cost vector c and solution vector x also have size 2^k. So neither is represented explicitly. Instead, Algorithm 3.2 starts with a square matrix and iterates by generating the columns of A in such a way as to decrease the objective function (lines 2–8).

At step 0, we start A with a $k \times k$-identity matrix I; the initial cost function c is such that $c_j = 1$ iff A's jth column is not $\mathcal{U}$-satisfying; otherwise $c_j = 0$, $1 \leq j \leq k$. As all generated columns at the following steps are $\mathcal{U}$-satisfying, all added elements of the cost vector c are 0.

At each step s, we start by solving the linear program $A_{(s)} \cdot x^{(s)} \bowtie b$ (line 3); so we suppose there is a linear programming solver available. We require that the solution generated contains the *primal solution* $x^{(s)}$ as well as the *dual solution* $z^{(s)}$ (Bertsimas and Tsitsiklis 1997); but note that the method for solving the linear program is not fixed. These are used in the column generation process (line 4) described below. If column generation fails, than the process cannot decrease current cost and Algorithm 3.2 is terminated with a negative decision. Otherwise a new column is generated, and A and c are updated. At the end, when the objective function has reached 0, the final values of A and x are returned.

We now describe the column generation process.

3.3 SAT-Based Column Generation

The idea of the SAT-based column generation is to map a linear inequality over a set of $\{0,1\}$-variables into a SAT-formula, in exactly the same way as a set of inequalities was mapped into a SAT-formula in Section 3.1 by using the $O(n)$ method described in (Warners 1998).

The inequality is obtained from the condition for choosing a column in the *Simplex Method* for solving linear programs (Bertsimas and Tsitsiklis 1997; Papadimitriou and Steiglitz 1998). The basic idea, given a linear program of the form (4), the *reduced cost* $\bar{c}_y$ of inserting a column y from A in a simplex basis is

$$\bar{c}_y = c_y - z' \cdot y \tag{5}$$

where c_y is the cost associated with column y and z is the dual solution of the system $A \cdot x \bowtie b$ of size k. As the generated column y is always $\mathcal{U}$-satisfying, $c_y = 0$, so to ensure a non-increasing value in the objective function we need a non-positive reduced cost, $\bar{c}_y \leq 0$, which leads us to

$$z' \cdot y \geq 0 \tag{6}$$

As y is a $\{0,1\}$-vector, inequality (6) can be transformed into a SAT-formula; that formula is added with the elements of $\mathcal{U}$, and an appropriate renaming of variables so as to force the resulting SAT-formula χ to represent the fact that the solution to (6) is $\mathcal{U}$-satisfying. We can then send the formula χ to a SAT-solver. If it is unsatisfiable, this means that there is no way to reduce the cost of the linear program's objective function. If it is satisfiable, we obtain an instance of the y variables that consists of a column that, when added to the basis of a simplex solution of linear program (4), is expected decrease the value of the objective function. It may maintain it with the same value, in which case the simplex process has reached a plateau; but it will never increase the objective's value.

Thus, we have shown how to construct a SAT-based column generation function $\mathit{GenerateColumn}(\mathit{solution}, \mathcal{U})$, provided we are given a (dual) solution for the corresponding linear program.

Theorem 1 *Algorithms 3.1, 3.2 and GenerateColumn provide a decision procedure for the CQU-SAT problem.* □

PROOF (SKETCH) The proof is a simplification of the correctness of the branch-and-bound method for ILP (Schrijver 1986), due to the fact that CQU-SAT requires only a single feasible integer solution instead of searching for optimality in the lattice of feasible integer solutions. Details omitted. ■

4 Experiments and Results

We have implemented the CQU-SAT via ILP method, that is, a CQU-SAT solver using Integer Logic Programming implementing Algorithms 3.1, 3.2 and *GenerateColumn* previously described. The implementation was developed in C++ [1] using the following open source software:

- Column generation implemented using MINISAT 2.2[2]
- we implemented the *SolveRelaxedViaColGen* method using the linear solver **Clp** of the COIN-OR Project[3].

The CQU-SAT decision method *CQUBranchAndBound* was implemented using those modules. The heuristics employed for choosing a node to solve (Algorithm 3.1, line 4) consists on finding a solutions that is "closer to integral", that is, a solution with the least number of non-integral solution; in case of a draw, we choose the node with the largest number of lines, that is, with the largest number of bounding refinement; if the draw persists, one node is chosen randomly. A second heuristic is needed for choosing the variable on which to branch (Algorithm 3.1, line 11), for which we chose, among the variables with non-integer values, the one whose value is closer to an integer. Values are considered integer up to precision $\epsilon = 10^{-}8$.

For the experiments, we created a normal form cqu format as a variant of the DIMACS cnf format[4]. We generate uniformly distributed random CQU-SAT instances in normal form with $k = |\mathcal{Q}|$ counting quantifier inequalities, and the set of universal restrictions $\mathcal{U}$ was generated in propositional CNF format with n variables and m 3-SAT clauses. Counting quantifier sentences of the form $\exists_{\bowtie \ell} p_i(x)$, $1 \leq i \leq k$, were generated by $\bowtie = \leq$ or $\bowtie = \geq$ with uniform distribution, and ℓ uniformly distributed in the interval $[1, \ldots, L]$.

For the first experiment, we fixed $L = 100$, $k = 10$ and $n = 250$ and varied the rate $\frac{m}{n}$ from 0.5 to 8 in steps of 0.05. At each step, we generated randomly 100 cqu-instances and computed the percentage of satisfiable instances (%SAT, left axis) and the average time of computation in seconds (right axis). We then obtained the graphic illustrated in Figure 1.

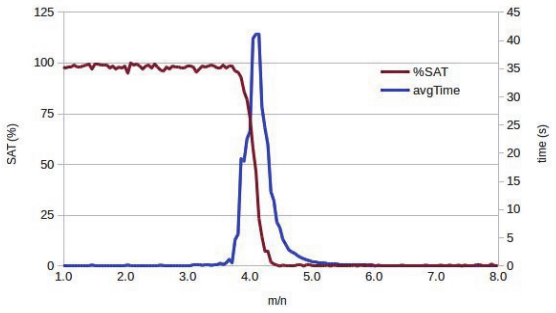

Figure 1: CQU-SAT Phase Transition, for $k = 10$, $n = 250$

[1] Available at http://cqu.sourceforge.net
[2] http://minisat.se/
[3] http://www.coin-or.org/
[4] http://www.satlib.org

Figure 1 clearly illustrates a phase transition behaviour, with the transition point around $\frac{m}{n} = 4.1$. We know that when $k = 0$, CQU-SAT becomes SAT, with phase transition point around 4.3 (Gent and Walsh 1994). As the rate of satisfiable formulas decreases when k increases, due to the extra constraints, it is expected that the rate of satisfiable formulas decreases for a given rate $\frac{m}{n}$, so the phase transition point is supposed to dislocate to the left, as observed.

It is obvious that the larger the value of n, the higher the time curve. However, we hypothesized that for a fixed value of $\frac{k}{n}$ the shapes of the curves of percentage of CQU-SAT formulas is about the same. To test this hypothesis, a second experiment was performed, as shown in Figure 2.

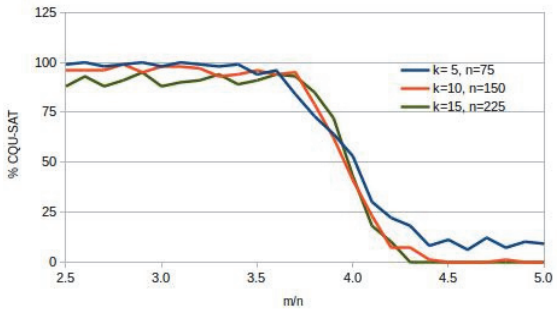

Figure 2: Percentage of CQU-SAT formulas, for fixed $\frac{k}{n} = \frac{1}{15}$, $k = 5, 10, 15$ and $n = 75, 150, 225$, respectively

In that experiment, we fixed $\frac{k}{n} = \frac{1}{15}$ and we drew three curves for $k = 5, 10, 15$ and $n = 75, 150, 225$, respectively, generating 100 formulas for each value of $\frac{m}{n}$, set from 2.5 to 5.0 in 0.1 steps; the percentage of CQU satisfiable formulas at each step was measured. The result is shown in Figure 2, which is consistent with the hypothesis, displaying three similar curves with similar phase-transition points; experiments with a larger number of points at shorter intervals should lead to a better agreement among the curves of CQU-SAT percentage.

5 Conclusion

We have proposed an expressive fragment of counting quantifiers over unary predicates, CQU, for which the satisfiability problem is NP-complete and an integer linear algebraic formulation is possible. We have proposed a CQU-SAT solving method by direct reduction to SAT problems, which proved unfeasible, and a branch-and-bound method which employs SAT-based column generation, which appears to be feasible and allows us to show a phase transition behaviour for the CQU-SAT problem. The phase-transition shows there is predominance of feasible CQU-SAT instances.

Future research aims at extending the counting quantifier fragment with possible applications to feasible Description Logic handling of counting; we also plan to study inference from CQU consistent and inconsistent sets of formulas.

Acknowledgements This work was supported by Fapesp proj. 2015/21880-4 and CNPq grant 306582/2014-7. GDB is supported by CNPq grant PDE 200780/2015-8.

References

Baader, F.; Buchheit, M.; and Hollander, B. 1996. Cardinality restrictions on concepts. *Artificial Intelligence* 88(1):195–213.

Bertsimas, D., and Tsitsiklis, J. N. 1997. *Introduction to Linear Optimization*. Athena Scientific, 1st edition.

Biere, A. 2014. Lingeling essentials, a tutorial on design and implementation aspects of the the sat solver lingeling. In *POS@ SAT*, 88. Citeseer.

Bulatov, A. A., and Hedayaty, A. 2015. Galois correspondence for counting quantifiers. *Multiple-Valued Logic and Soft Computing* 24(5–6):405–424.

Cheeseman, P.; Kanefsky, B.; and Taylor, W. M. 1991. Where the really hard problems are. In *12th IJCAI*, 331–337. Morgan Kaufmann.

Eckhoff, J. 1993. Helly, Radon, and Caratheodory type theorems. In Gruber, P. M., and Wills, J. M., eds., *Handbook of Convex Geometry*. Elsevier Science Publishers. 389–448.

Eén, N., and Sörensson, N. 2003. An extensible SAT-solver. In *SAT 2003*, volume 2919 of *LNCS*, 502–518. Springer.

Eisenbrand, F., and Shmonin, G. 2006. Carathéodory bounds for integer cones. *Operations Research Letters* 34(5):564–568.

Finger, M., and De Bona, G. 2011. Probabilistic satisfiability: Logic-based algorithms and phase transition. In *IJCAI*, 528–533.

Finger, M., and De Bona, G. 2015. Probabilistic satisfiability: algorithms with the presence and absence of a phase transition. *Annals of Mathematics and Artificial Intelligence* 75(3):351–379.

Gent, I. P., and Walsh, T. 1994. The SAT phase transition. In *ECAI94 – Proceedings of the Eleventh European Conference on Artificial Intelligence*, 105–109. John Wiley & Sons.

Georgakopoulos, G.; Kavvadias, D.; and Papadimitriou, C. H. 1988. Probabilistic satisfiability. *J. of Complexity* 4(1):1–11.

Grädel, E., and Otto, M. 1999. On logics with two variables. *Theoretical Computer Science* 224(1):73–113.

Hansen, P., and Jaumard, B. 2000. Probabilistic satisfiability. In *Handbook of Defeasible Reasoning and Uncertainty Management Systems, vol.5*. Springer.

Jaumard, B.; Hansen, P.; and Poggi de Aragão, M. 1991. Column generation methods for probabilistic logic. *ORSA Journal on Computing* 3(2):135–148.

Lindström, P. 1966. First order predicate logic with generalized quantifiers. *Theoria* 32(3):186–195.

Martin, B.; Madelaine, F. R.; and Stacho, J. 2015. Constraint satisfaction with counting quantifiers. *SIAM J. Discrete Math.* 29(2):1065–1113.

Mitchell, D.; Selman, B.; and Levesque, H. 1992. Hard and easy distributions of SAT problems. In *AAAI92 – Proceedings of the 10th National Conference on Artificial Intelligence*, 459–465.

Nilsson, N. 1986. Probabilistic logic. *Artificial Intelligence* 28(1):71–87.

Papadimitriou, C., and Steiglitz, K. 1998. *Combinatorial Optimization: Algorithms and Complexity*. Dover.

Pratt-Hartmann, I. 2005. Complexity of the two-variable fragment with counting quantifiers. *Journal of Logic, Language and Information* 14(3):369–395.

Pratt-Hartmann, I. 2008. On the computational complexity of the numerically definite syllogistic and related logics. *The Bulletin of Symbolic Logic* 14(1):1–28.

Schrijver, A. 1986. *Theory of Linear and Integer Programming*. New York, NY, USA: John Wiley & Sons, Inc.

Warners, J. P. 1998. A linear-time transformation of linear inequalities into conjunctive normal form. *Inf. Process. Lett.* 68(2):63–69.