Pushing the Boundaries of Reasoning about Qualified Cardinality Restrictions

Jelena Vlasenko, Volker Haarslev, Brigitte Jaumard

Concordia University, Montréal, Québec, Canada

Abstract. We present a novel hybrid architecture for reasoning about description logics supporting role hierarchies and qualified cardinality restrictions (QCRs). Our reasoning architecture is based on saturation rules and integrates integer linear programming. Deciding the *numerical* satisfiability of a set of QCRs is reduced to solving a corresponding system of linear inequalities. If such a system is infeasible then the QCRs are unsatisfiable. Otherwise the numerical restrictions of the QCRs are satisfied but unknown entailments between qualifications can still lead to unsatisfiability. Our integer linear programming (ILP) approach is highly scalable due to integrating learned knowledge about concept subsumption and disjointness into a column generation model and a decomposition algorithm to solve it. Our experiments indicate that this hybrid architecture offers a better scalability for reasoning about QCRs than approaches combining both tableaux and ILP or applying traditional (hyper)tableau methods.

1 Introduction

The performance of the original $\mathcal{ALCQ}$ tableau algorithm [19] that is implemented by most description logic (DL) reasoners covering qualified cardinality restrictions[1] (QCRs) is not optimal. To perform a concept satisfiability test, the tableau algorithm creates role successors to satisfy at-least restrictions, e.g., $\geq 20\, R.C$. Given at-most restrictions, e.g., $\leq 10\, R.D$, $\leq 10\, R.E$, the algorithm resolves each R-successor as either D or $\neg D$, and E or $\neg E$. If an at-most restriction for R is violated ($\leq 10\, R.D$), the algorithm nondeterministically merges two R-successors that are instances of D. This uninformed process is highly inefficient, especially when the algorithm has to deal with larger cardinalities and/or large sets of related QCRs. In [11, Sec. 4.1.1] it was shown that if a set of QCRs contains p at-least ($\geq n_i\, R_i.C_i$) and q at-most restrictions ($\leq m_j\, R'_j.C'_j$), then roughly $2^{qN} \prod_{i=0}^{M-2} \binom{M-i}{2} / (M-1)!$ branches need to be explored in the worst case by the standard $\mathcal{ALCQ}$ algorithm (assuming that M R'_j-successors exist in C'_j with $M > m_j$ and $N = \sum_{i=1}^{p} n_i$).

In our previous work (inspired by [26]) we have shown that algebraic tableaux can improve reasoning on QCRs dramatically for DLs such as $\mathcal{SHQ}$ [11], $\mathcal{SHIQ}$ [28], and $\mathcal{SHOQ}$ [9, 10]. The basic idea in these calculi is to transform a set of QCRs into a linear optimization problem that will be solved accordingly. These algorithms need to explore $2^{2^{p+q}}$ branches in the worst case but they are independent of N, M. If there is a feasible solution to the problem then the corresponding set of QCRs is satisfiable

[1] Also known as graded modalities in modal logics.

provided completion rules encounter no logical clashes for the returned solution. The prototypes implementing the above-mentioned approaches on algebraic tableaux [11, 28, 10] could demonstrate runtime improvements by several orders of magnitude for reasoning about QCRs (and nominals). However, we identified the following two disadvantageous characteristics.

(i) Given n QCRs (and nominals) the naive encoding of the corresponding system of inequalities requires n rows and 2^m columns, where m is the cardinality of the set P of all pairs of roles and their qualifications occurring in the n given QCRs. Let us illustrate this with a small example: $\geq 2R.C \sqcap \geq 2R.D \sqcap \leq 2R.E$. In this case, $P = \{R_C, R_D, R_E\}$, $n = 3$, $m = 3$. In order to represent the QCRs as inequalities we create $\sum x_{C_i} \geq 2$, $\sum x_{D_j} \geq 2$, and $\sum x_{E_k} \leq 2$. For instance, the variables x_{C_i} represent the cardinalities of all elements in the power set of P that contain R_C. The same holds for the other variables respectively. As an additional constraint we specify that all variables must have values in $\mathbb{N}$. Our objective function minimizes the sum of all variables. Intuitively speaking, the above-mentioned concept conjunction is feasible and also satisfiable in this trivial case if the given system of inequalities has a solution in $\mathbb{N}$. It is easy to see that the size of such an inequality system is exponential with respect to m. Furthermore, in order to ensure completeness, in our previous work we required a so-called choose rule that implements a semantic split that nondeterministically adds for each variable x either the inequality $x \leq 0$ or $x \geq 1$. Unfortunately, this uninformed choose-rule could fire 2^{2^m} times in the worst case and cause a severe performance degradation.

(ii) The employed integer linear programming (ILP) algorithms were best-case exponential in the number of occurring QCRs due to the explicit representation of 2^m variables. In [9, 10] we developed an optimization technique called lazy partitioning that tries to delay the creation of ILP variables but it cannot avoid the creation of 2^m variables in case m QCRs are part of a concept model. Our experiments in [9–11] indicated that quite a few ILP solutions can cause clashes due to lack of knowledge about known subsumptions, disjointness, and unsatisfiability of concept conjunctions. This knowledge can help reducing the number of variables and eliminating ILP solutions that would fail logically. For instance, an ILP solution for the example presented in the previous paragraph might require to create an R-successor as an instance of $C \sqcap D$. However, if C and D are disjoint this ILP solution will cause a clash (and fail logically).

Characteristic (i) can be avoided by eliminating the choose-rule for variables. This does not sacrifice completeness because the algorithms implementing our ILP component are complete (and certainly sound) for deciding (in)feasibility. In case a system is feasible (or numerically satisfiable), dedicated saturation rules determine whether the returned solutions are logically satisfiable. In case of logical unsatisfiability a corresponding unsatisfiable concept conjunction is added to the input of the ILP component and therefore monotonically constrains the remaining feasibility space. Consequently, previously computed solutions that result in unsatisfiability are eliminated. For instance, the example above would be deemed as infeasible once ILP knows that C, D are subsumed by E and C, D are disjoint.

The avoidance of characteristic (ii) is motivated by the observation that only a small number of the 2^m variables will have non-zero values in the optimal solution of the

linear relaxation, i.e., no more variables than the number of constraints following the characteristics of the optimal solution of a linear program, see, e.g., [6]. Moreover, in practice, only a limited number of variables have a non-zero value in the integer optimal solution. In addition, linear programming techniques such as column generation [7, 13] can operate with as few variables as the set of so-called basic variables in linear programming techniques at each iteration, i.e., nonbasic variables can be eliminated and are not required for the guarantee of reaching the conclusion that a system of linear inequalities is infeasible, or for reaching an optimal LP solution. Although the required number of iterations varies from one case to another, it is usually extremely limited in practice, in the order of few times the number of constraints. The efficiency of the branch-and-price method, which is required in order to derive an optimal ILP solution, e.g., [3, 31, 23], depends on the quality of the integrality gap (i.e., how far the optimal linear programming solution is from the optimal ILP solution in case the system of inequalities is feasible, and on the level of infeasibility otherwise). Furthermore, our new ILP approach considers known subsumptions, disjointness, and unsatisfiability of concept conjunctions and uses a different encoding of inequalities that already incorporates the semantics of universal restrictions. We delegate the generation of inequalities completely to the ILP component.

To summarize, the novel features of our architecture are (i) saturation rules that do not backtrack to decide subsumption (and disjointness) [32]; (ii) feasibility of QCRs is decided by ILP (in contrast to [4]); (iii) our revised encoding of inequalities, which incorporates role hierarchies, the aggregation of information about subsumption, disjointness, and unsatisfiability of concept conjunctions, allows a more informed mapping of QCR satisfiability to feasibility and reduces the number of returned solutions that fail logically; (iv) the best-case time complexity of our ILP feasibility test is polynomial to the number of inequalities [24]. This work extends our previous research on the $\mathcal{ELQ}$ Avalanche reasoner [32].

2 Preliminaries

Description logics are a family of knowledge representation languages that form a basis for the Web Ontology Language (OWL). The DL $\mathcal{ALCHQ}$, which is a core subset of OWL, allows role hierarchies ($\mathcal{H}$) and the concept-forming constructors conjunction, disjunction, negation, at-least and at-most restriction ($\mathcal{Q}$). The semantics of $\mathcal{ALCHQ}$ concepts and roles is defined by an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ that maps a concept C to $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and a role R to $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. For convenience we use the concepts $\top$ and $\bot$ with $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$ and $\bot^{\mathcal{I}} = \emptyset$.

$\mathcal{ALCHQ}$ concepts are inductively defined from concept and role names using the constructors as follows ($n, m \in \mathbb{N}$, $n \geq 1$, $\| \cdot \|$ denotes set cardinality, $F^{R,C}(x) = \{y \in C^{\mathcal{I}} \,|\, (x, y) \in R^{\mathcal{I}}\}$): (i) $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$; (ii) $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$; (iii) $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$; (iv) $(\geq n\, R.C)^{\mathcal{I}} = \{x \,|\, \|F^{C,R}(x)\| \geq n\}$; (v) $(\leq m\, R.C)^{\mathcal{I}} = \{x \,|\, \|F^{C,R}(x)\| \leq m\}$. The latter two constructors are called QCRs. A concept C is satisfiable if there exists an $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \emptyset$.

An $\mathcal{ALCHQ}$ Tbox $\mathcal{T}$ is defined as a finite set of axioms of the form $C \sqsubseteq D$ or $R \sqsubseteq S$, where C, D are concepts and R, S roles, and such axioms are satisfied by $\mathcal{I}$ if

$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ or $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$. We call $\mathcal{I}$ a model of $\mathcal{T}$ if it satisfies all axioms in $\mathcal{T}$. A Tbox $\mathcal{T}$ entails an axiom if all models of $\mathcal{T}$ satisfy that axiom.

One of the main tasks of a DL reasoner is to classify a Tbox by computing all subsumptions between named concepts. Tableau-based algorithms [2] are the most applied reasoning algorithms to date. Consequence-based or saturation-based algorithms [4, 30] are algorithms that accumulate or *saturate* entailed knowledge in a bottom-up way while tableaux attempt to prove entailment in a goal-oriented or top-down way. The idea of saturating knowledge comes from the one-pass saturation algorithm for the DL $\mathcal{EL}^{++}$ [1]. $\mathcal{EL}$ only allows conjunction and existential value restriction ($\exists R.C \equiv \geq 1\, R.C$). Different optimization techniques exist for different types of tasks performed by DL reasoners. In this work, we are interested in applying linear optimization in order to handle qualified number restrictions [9–11] in ontologies expressed in $\mathcal{ELQ}$, which is a superset of $\mathcal{EL}$ that additionally allows QCRs. It is well-known that $\mathcal{ELQ}$ is a syntactic variant of $\mathcal{ALCQ}$ [1].

Atomic decomposition was initially proposed in [26] to reason about sets, however it can also be used to reason about role fillers of qualified number restrictions in description logics. This technique allows us to reduce the problem of deciding feasibility of qualified number restrictions to solving a linear program. The example below illustrates how to transform qualified number restrictions into inequalities. Let us assume the following three qualified number restrictions $\geq 3\, hasColor.Blue$, $\geq 4\, hasColor.Red$, $\leq 5\, hasColor.Green$. We denote the partition of the set $\{b, r, g\}$ ($b = blue$, $r = red$, $g = green$) as $\{b, r, g, br, bg, rg, brg\}$ where the absence of a letter indicate the implied presence of its negation, e.g., b stands for the intersection of blue, not red, not green. Then, we get the corresponding inequalities ($|\,.\,|$ denotes set cardinality).

$$
\begin{aligned}
|b| + |br| + |bg| + |brg| &\geq 3 \\
|r| + |br| + |rg| + |brg| &\geq 4 \\
|g| + |bg| + |rg| + |brg| &\leq 5
\end{aligned}
$$

In such a way we preserve the semantics of qualified number restrictions and reduce a satisfiability problem to a feasibility problem, i.e., whether a 0-1 linear program is feasible.

3 Column Generation and Branch-and-Price Methods

We discuss here how to check the feasibility of a 0-1 linear program with a column generation model, i.e., with an exponential number of variables. It consists in first checking whether its linear relaxation, a linear program with an exponential number of variables, is feasible. We next provide a brief overview of linear programming (LP for short) and column generation.

Linear Programming was recognized as one of the most important scientific achievements of the 20th century. It allows the solution of complicated problems that concern allocation of scarce resources with respect to various constraints in a minimum amount of time. There exist different approaches to solve linear programs. One of them is the simplex method that was proposed in 1947 by George B. Dantzig. Although the simplex method requires an exponential number of iterations in the worst case, it performs

very well in practice. In 1979 the ellipsoid method was proposed by Leonid Khachiyan, and could solve linear programming problems in polynomial time. Nevertheless, the simplex method, despite being worst-case exponential is more efficient in practice than the ellipsoid method. The interior-point method is another polynomial-time algorithm that was proposed in 1984 by Narendra Karmarkar and its recent refinements [12] are competitive with the simplex algorithm.

The algorithms mentioned above have been integrated into different commercial and open source solvers, e.g., CPLEX, Gurobi, XPRESS. These solvers are capable of solving very large linear programming problems, i.e., with up to hundreds of thousands of variables. When it comes to millions of variables, their performance starts to deteriorate. We can then use the column generation method. The idea behind this method is that only a subset of all variables (columns) have non zero values in the optimal LP solution. Indeed, there are no more than the number of constraints, i.e., the number of so-called basic variables in linear programming. Numerous large-scale optimization problems are now using it [23].

Back to feasibility checking, column generation can easily detect infeasible linear programs. However, infeasible integer linear programs do not necessarily have an infeasible linear programming relaxation. In order to detect infeasibility in such cases, it is then required to use a branch-and-price algorithm [3] (i.e., a branch-and-bound algorithm [25] in which the linear relaxation is solved with a column generation method).

In the context of our work, we create very large integer linear programs with numerous variables. Therefore, we choose to solve the continuous relaxation with the column generation method to address scalability issues, following the success of using it for, e.g., deciding the consistency of a set of clauses in the context of probabilistic logic [20, 17]. In order to produce integer solutions, column generation is combined with a branch-and-price algorithm [3] to either conclude that the model has no solution or to obtain an integer solution.

Column generation together with the branch-and-price method have been implemented into a system that we decided to call QMediator. QMediator is a middle layer or a mediator that facilitates communication between Avalanche and CPLEX. This process will be described in detail in Section 5. In short, whenever Avalanche needs to process qualified number restrictions it calls QMediator. QMediator in turn creates a corresponding integer linear program based on the received information and solves it by means of column generation and branch-and-price methods. To actually solve the integer linear program QMediator calls CPLEX.

The example below illustrates how column generation works in practice, without the need of a branch-and-price method for this particular example.

Consider the axiom $D \sqsubseteq \geq 2\, R.A \sqcap \geq 3\, R.B \sqcap \leq 4\, R.C$. Initially we assume that there is no known subsumption relationship between the QCR qualifications A, B, C. Since we have only one role, we can ignore its name and only focus on $Q_D = Q_D^{\geq} \cup Q_D^{\leq}$, which is our base set for partitioning, with $Q_D^{\geq} = \{A, B\}$ and $Q_D^{\leq} = \{C\}$. The complete decomposition set (or partition) is $\mathcal{D}_D = \{\{A\}, \{B\}, \{C\}, \{A, B\}, \{A, C\}, \{B, C\}, \{A, B, C\}\}$ where each partition element p represents the intersection of p's elements plus the intersection of all $\neg e_i$ with $e_i \in Q_D \setminus p$. We denote the elements of $\mathcal{D}_D$ by the variables $x_A, x_B, x_C, x_{AB}, x_{AC}, x_{BC}, x_{ABC}$. In the context of the ILP

model, note that each variable is associated with a column, so we may use either terms in the sequel.

First example The QCRs for concept D result in the following ILP problem.

$$\min \quad \sum_{p \in \mathcal{D}_D} x_p \tag{1}$$

subject to:

$$x_A + 0x_B + 0x_C + x_{AB} + x_{AC} + 0x_{BC} + x_{ABC} \geq 2 \quad \rightsquigarrow \geq 2\, R.A \tag{2}$$
$$0x_A + x_B + 0x_C + x_{AB} + 0x_{AC} + x_{BC} + x_{ABC} \geq 3 \quad \rightsquigarrow \geq 3\, R.B \tag{3}$$
$$0x_A + 0x_B + x_C + 0x_{AB} + x_{AC} + x_{BC} + x_{ABC} \leq 4 \quad \rightsquigarrow \leq 4\, R.C \tag{4}$$
$$x_p \in \mathbb{N}, \qquad \text{for } p \in \mathcal{D}_D.$$

The optimal solution is $x_B = 1$, $x_{AB} = 2$, and all other variables are equal to zero.

However, since the size of $\mathcal{D}_D$ is exponential with respect to the size of Q_D, in general one cannot afford to enumerate all variables. In order to use a column generation modelling, model (1)-(4) is decomposed into a restricted master problem (RMP), made of a subset of columns, and the pricing problem (PP), which can be viewed as a column generator. The RMP contains the inequalities (rows) representing the QCRs, with a very restricted set of variables. Initially one can start with an empty set P of variables x_p, and a set of artificial variables h_q, one for each constraint, i.e., for each element in Q_D (here $n = 3$) using an arbitrarily large cost W (here $W = 10$). Those artificial variables define an initial artificial feasible solution, however, in order to be feasible, the QCR set must not use any of them in its solution.

The cost of a partition element p is defined as the number of elements it contains. Consequently, the objective function of the RMPs is defined as $\sum_{p \in P} \text{cost}_p x_p + W \sum_{i=1}^{n} h_i$. The choice of the cost is related to the selection of partition elements of smaller sizes and thus of less restricted solutions. Indeed, it promotes the reuse of nodes in Avalanche's saturation graph. The optimal solution[2] of (RMP 1) has a cost 50, and contains two non-zero artificial variables.

Minimize $10h_1 + 10h_2 + 10h_3$ subject to (RMP 1)

$h_1 \geq 2$

$h_2 \geq 3$

$h_3 \leq 4$

Solution: cost $= 50; h_1 = 2, h_2 = 3;$ Dual: $\pi_A = 10, \pi_B = 10$

The objective of the PP is equal to the so-called reduced cost in linear programming (see, e.g., Chvatal [6] if not familiar with linear programming). It uses the dual price values as coefficients of the variables associated with a potential partition element, i.e., binary variables b_q, r_q ($q \in Q_D$) to ensure the description logics semantics

[2] The value of variables not listed in a solution are equal to zero.

of QCRs. The variables b_q indicate whether role successors must be an instance of q and r_q whether an R-successor that is an instance of q must exist. For each at-least QCR with a role and its qualification, P must contain a corresponding variable, e.g., for $\geq 2\, R.A$ if $r_A = 1$ a variable b containing A in its subscript must exist ($r_A - b_A \leq 0$). If P contains a qualification of an at-most QCR, then a corresponding variable must be present, e.g., if C occurs in P ($b_C = 1$), then a variable r containing C in its subscript must exist ($b_C - r_C \leq 0$). The objective function of the PP can then be written as

$$\sum_{q \in Q_D} b_q - \sum_{q \in Q_D^{\geq}} \pi_q r_q - \sum_{q \in Q_D^{\leq}} \omega_q r_q \tag{5}$$

Based on this formula we can define (PP 1). In its objective function the only non-zero dual price values (coefficients) are π_A, π_B due to (RMP 1) .

Minimize $b_A + b_B + b_C - 10r_A - 10r_B$ subject to (PP 1)

$$\begin{aligned} r_A - b_A &\leq 0 \\ r_B - b_B &\leq 0 \qquad \text{(CPP 1)} \\ b_C - r_C &\leq 0 \end{aligned}$$

Solution: $cost = -18, r_A = 1, r_B = 1, b_A = 1, b_B = 1$.

Since the values of r_A, r_B are 1, we add the variable x_{AB} to the next RMP ($P = \{\{A, B\}\}$). The cost of its solutions is reduced, from 50 in (RMP 1) to 6 in (RMP 2).

Minimize $2x_{AB} + 10h_1 + 10h_2 + 10h_3$ subject to: (RMP 2)

$$\begin{aligned} x_{AB} + h_1 &\geq 2 \\ x_{AB} + h_2 &\geq 3 \\ h_3 &\leq 4 \end{aligned}$$

Solution: $cost = 6, x_{AB} = 3;$ Dual: $\pi_B = 2$

In the objective of (PP 2) the only non-zero dual price value is π_B (see also (5)).

Minimize $b_A + b_B + b_C - 2r_B$ subject to (CPP 1) (PP 2)

Solution: $cost = -1, r_B = 1, b_B = 1$

Since the value of r_B is 1, we add the variable x_B to the next RMP ($P = \{\{B\}, \{A, B\}\}$), whose cost is further reduced, from 6 in (RMP 2) to 5 in (RMP 3).

Minimize $x_B + 2x_{AB} + 10h_1 + 10h_2 + 10h_3$ subject to (RMP 3)

$$\begin{aligned} x_{AB} + h_1 &\geq 2 \\ x_B + x_{AB} + h_2 &\geq 3 \\ h_3 &\leq 4 \end{aligned}$$

Solution: $cost = 5, x_B = 1, x_{AB} = 2;$ Dual: $\pi_A = 1, \pi_B = 1$

In the objective of (PP 3) the only non-zero dual price values are π_A, π_B.

$$\text{Minimize } b_A + b_B + b_C - r_A - r_B \text{ subject to (CPP 1)} \qquad \text{(PP 3)}$$
$$\text{Solution: } cost = 0$$

At this point all variables h_i in (RMP 3) and r_q in (PP 3) are zero indicating that we have reached a feasible solution. Moreover, since the reduced cost of the problem is always positive no "improving" column can be added. This allows us to conclude that we have reached the optimal solution of the LP. Lastly, as this LP optimal solution is integer, we can also claim that it defines the optimal set of partition elements. The inequality system (1) is feasible and the solution in (RMP 3) results in creating one R-successor that is an instance of B with cardinality 1 ($x_B = 1$) and one R-successor that is an instance of $A \sqcap B$ with cardinality 2 ($x_{AB} = 2$). Obviously, this solution satisfies the initial inequalities since the successor $A \sqcap B$ satisfies $\geq 2\, R.A$ and $\geq 2\, R.B$. Thus, the B successor together with the $A \sqcap B$ successor will satisfy $\geq 3\, R.B$.

Second example This example adds to the first example the axioms $A \sqsubseteq C$ and $B \sqsubseteq C$. The original inequality system (1) and (RMP 1) remain unchanged. The new pricing problem below accommodates the added subsumptions, e.g., $A \sqsubseteq C$ is modelled as $b_A \leq b_C \iff b_A - b_C \leq 0$.

$$\text{Minimize } b_A + b_B + b_C - 10r_A - 10r_B \text{ subject to} \qquad \text{(PP 4)}$$
$$\begin{aligned} r_A - b_A &\leq 0 \\ r_B - b_B &\leq 0 \\ b_C - r_C &\leq 0 \qquad \text{(CPP 4)} \\ b_A - b_C &\leq 0 \\ b_B - b_C &\leq 0. \end{aligned}$$
$$\text{Solution: } cost = -17, r_A = 1, r_B = 1, r_C = 1, b_A = 1, b_B = 1, b_C = 1.$$

Since the values of r_A, r_B, r_C are 1 we add the variable x_{ABC} to the next version of our RMP ($P = \{\{A, B, C\}\}$), which reduces the cost from 50 to 9 in (RMP 5).

$$\text{Minimize } 3x_{ABC} + 10h_1 + 10h_2 + 10h_3 \text{ subject to} \qquad \text{(RMP 5)}$$
$$\begin{aligned} x_{ABC} + h_1 &\geq 2 \\ x_{ABC} + h_2 &\geq 3 \\ x_{ABC} + h_3 &\leq 4 \end{aligned}$$
$$\text{Solution: } cost = 9, x_{ABC} = 3; \text{ Dual: } \pi_B = 3$$

$$\text{Minimize } b_A + b_B + b_C - 3r_B \text{ subject to (CPP 4)} \qquad \text{(PP 5)}$$
$$\text{Solution: } cost = -1, r_B = 1, r_C = 1, b_B = 1, b_C = 1$$

Since the values of r_B, r_C are 1 we add the variable x_{BC} to the next version of our RMP ($P = \{\{B, C\}, \{A, B, C\}\}$), which reduces the cost from 9 to 8 in (RMP 6).

$$\begin{aligned} &\text{Minimize } 2x_{BC} + 3x_{ABC} + 10h_1 + 10h_2 + 10h_3 \text{ subject to} \qquad \text{(RMP 6)} \\ &\qquad x_{ABC} + h_1 \geq 2 \\ &x_{BC} + x_{ABC} + h_2 \geq 3 \\ &x_{BC} + x_{ABC} + h_3 \leq 4 \\ &\text{Solution: } cost = 8, x_{BC} = 1, x_{ABC} = 2; \text{ Dual: } \pi_A = 1, \pi_B = 2 \end{aligned}$$

$$\begin{aligned} &\text{Minimize } b_A + b_B + b_C - r_A - 2r_B \text{ subject to (CPP 4)} \qquad \text{(PP 6)} \\ &\text{Solution: } cost = 0 \end{aligned}$$

All variables r_q are zero, so, no variable can be added to minimize (RMP 6) further. The inequality system (1) is feasible and according to (RMP 6) we create an R-successor that is an instance of $B \sqcap C$ with cardinality 1 and an R-successor that is an instance of $A \sqcap B \sqcap C$ with cardinality 2. Since we have 3 R-successors that instances of C, the QCR $\leq 4\, R.C$ is satisfied.

Third example This example adds to the second example the axiom $A \sqcap B \sqsubseteq \bot$. The original inequality system (1) and (RMP 1) remain unchanged. The new pricing problem below accommodates the added disjointness, i.e., $A \sqcap B \sqsubseteq \bot$ is modelled as $b_A + b_B \leq 1$.

$$\begin{aligned} &\text{Minimize } b_A + b_B + b_C - 10r_A - 10r_B \text{ subject to} \qquad \text{(PP 7)} \\ &\quad r_A - b_A \leq 0 \\ &\quad r_B - b_B \leq 0 \\ &\quad b_C - r_C \leq 0 \\ &\quad b_A - b_C \leq 0 \\ &\quad b_B - b_C \leq 0 \\ &\quad b_A + b_B \leq 1 \qquad \text{(CPP 7)} \\ &\text{Solution: } cost = -8, r_A = 1, r_C = 1, b_A = 1, b_C = 1 \end{aligned}$$

Since the values of r_A, r_C are 1 we add the variable x_{AC} to the next version of our RMP ($P = \{\{A, C\}\}$), which reduces the cost from 50 to 34 in (RMP 8).

$$\begin{aligned} &\text{Minimize } 2x_{AC} + 10h_1 + 10h_2 + 10h_3 \text{ subject to} \qquad \text{(RMP 8)} \\ &\quad x_{AC} + h_1 \geq 2 \\ &\qquad h_2 \geq 3 \\ &\quad x_{AC} + h_3 \leq 4 \\ &\text{Solution: } cost = 34, x_{AC} = 2, h_2 = 3; \text{ Dual: } \pi_A = 2, \pi_B = 10 \end{aligned}$$

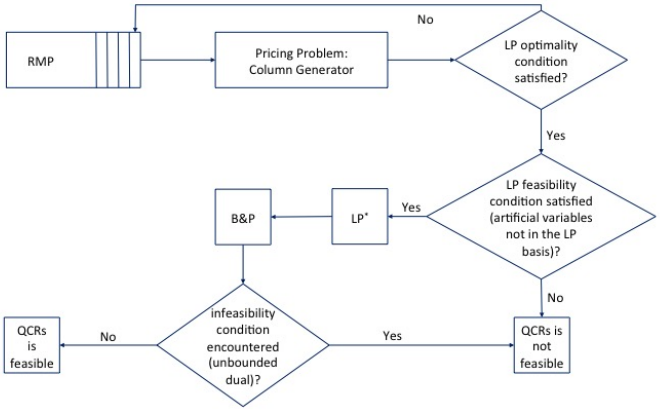

Fig. 1. Optimization Chart

Minimize $b_A + b_B + b_C - 2r_A - 10r_B$ subject to (CPP 7) (PP 8)

Solution: $cost = -8, r_B = 1, r_C = 1, b_B = 1, b_C = 1$

Since the values of r_B, r_C are 1 we add the variable x_{BC} to the next version of our RMP ($P = \{\{B, C\}, \{A, C\}\}$), which reduces the cost from 34 to 14 in (RMP 9).

Minimize $2x_{AC} + 2x_{BC} + 10h_1 + 10h_2 + 10h_3$ subject to (RMP 9)

$$x_{AC} + h_1 \geq 2$$
$$x_{BC} + h_2 \geq 3$$
$$x_{AC} + x_{BC} + h_3 \leq 4$$

Solution: $cost = 14, x_{BC} = 2, x_{AC} = 2, h_2 = 1$; Dual: $\pi_A = 8, \pi_B = 10, \omega_C = -8$

Minimize $b_A + b_B + b_C - 8r_A - 10r_B + 8r_C$ subject to (CPP 7) (PP 9)

Solution: $cost = 0$

All variables r_q are zero, so, no variable can be added to minimize (RMP 9) further. However, the inequality system (1) is now infeasible because (RMP 9) still contains the non-zero artificial variable h_2. The infeasibility is caused by the disjointness between the QCR qualifications A and B.

The process described above is summarized in Figure 1. We first define RMP and apply column generation to produce new columns until we obtain an optimal solution. Then, if the solution is infeasible the submitted QCRs are infeasible as well. Otherwise, if the solution is feasible then we proceed with applying the branch-and-price method. If the problem is not feasible, then it will be detected at some iteration in the branch-and-price, while solving the linear relaxation with the column generation algorithm. Otherwise, if the problem is feasible, then the branch-and-price will output a feasible solution.

4 Role Hierarchies

Role hierarchies can easily be mapped to ILP. We illustrate the methodology first with a small example and later with one that entails role subsumption. However, please note that they have not yet been integrated in the current version of Avalanche.

Simple example. Let $A \sqsubseteq \geq 2\, S.B \sqcap \geq 2\, U.C \sqcap \leq 3\, R.\top$ with S, U subroles of R. The concept A is satisfiable and its least constrained model must have at least one SU-successor that is an instance of $B \sqcap C$. Role hierarchies only need to be considered if an at-most QCR referring to a superrole (R) is restricting other QCRs referring to subroles (S, U) of R. The semantics of role hierarchies is encoded in the inequalities generated for the corresponding at-most QCRs.

We define $Q_A^{\geq} = \{S_B, U_C\}$ and $Q_A^{\leq} = \{R\}$. Since S, U are subroles of R, any partition element containing a subrole and its superrole can be simplified by removing the superrole because their intersection is equal to the subrole, e.g., $\{S_B, R\}$ is equal to $\{S_B\}$. Additionally, R can be removed from Q_A because no at-least QCR mentioning R exists. The complete decomposition set is $\mathcal{D}_A = \{\{S_B\}, \{U_C\}, \{S_B, U_C\}\}$. We denote these partition elements by the variables $x_{S_B}, x_{U_C}, x_{S_B U_C}$.

The QCRs for concept A result in the following ILP problem.

$$
\begin{aligned}
&\text{Minimize } x_{S_B} + x_{U_C} + 2x_{S_B U_C} \text{ subject to} \qquad (6)\\
&\quad x_{S_B} + 0x_{U_C} + x_{S_B U_C} \geq 2 \quad \rightsquigarrow \geq 2\, S.B\\
&\quad 0x_{S_B} + x_{U_C} + x_{S_B U_C} \geq 2 \quad \rightsquigarrow \geq 2\, U.C\\
&\quad x_{S_B} + x_{U_C} + x_{S_B U_C} \leq 3 \quad \rightsquigarrow \leq 3\, R.\top\\
&\text{with } x_{S_B}, x_{U_C}, x_{S_B U_C} \in \mathbb{N}
\end{aligned}
$$

The optimal solution is $x_{S_B} = 1$, $x_{U_C} = 1$, $x_{S_B U_C} = 1$. We create one SU-successor that is an instance of $B \sqcap C$, one S-successor that is an instance of B, and one U-successor that is an instance of C. All three successors have a cardinality of 1.

Example with entailed role subsumption The combination of role hierarchies and QCRs can be used to entail role subsumptions. Let us assume a Tbox $\mathcal{T} = \{\top \sqsubseteq \leq 1\, R.\top, \geq 1\, S.\top \sqsubseteq C, C \sqsubseteq \geq 1\, U.\top, A \sqsubseteq \geq 1\, S.B \sqcap \leq 0\, U.B\}$ with S, U subroles of R. $\mathcal{T}$ entails that S is a subrole of U and thus $A \sqsubseteq \bot$. It is easy to see that A is subsumed by C via the role S. Thus the QCRs applicable to A are $\leq 1\, R.\top, \geq 1\, S.B, \leq 0\, U.B$, $\geq 1\, U.\top$.

We define $Q_A^{\geq} = \{S_B, U\}$ and $Q_A^{\leq} = \{R, U_B\}$. After applying the simplifications from above we get $\mathcal{D}_A = \{\{U\}, \{S_B\}, \{U_B\}, \{U, S_B\}, \{S_B, U_B\}\}$. We denote these partition elements by the variables $x_U, x_{S_B}, x_{U_B}, x_{US_B}, x_{S_B U_B}$.

The QCRs for concept A result in the following ILP problem.

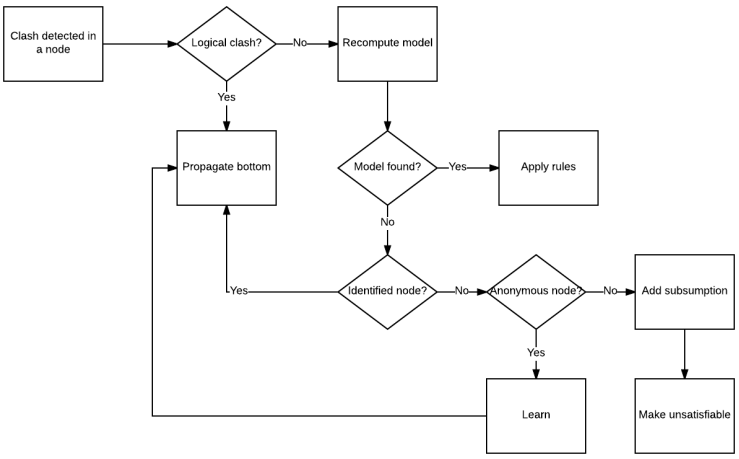

Fig. 2. Clash Detection

$$\begin{aligned}
&\text{Minimize } x_U + x_{S_B} + x_{U_B} + 2x_{US_B} + 2x_{S_BU_B} \text{ subject to} \qquad (7)\\
&\quad 0x_U + x_{S_B} + 0x_{U_B} + x_{US_B} + x_{S_BU_B} \geq 1 \quad \rightsquigarrow \geq 1\, S.B\\
&\quad x_U + 0x_{S_B} + x_{U_B} + x_{US_B} + x_{S_BU_B} \geq 1 \quad \rightsquigarrow \geq 1\, U.\top\\
&\quad x_U + x_{S_B} + x_{U_B} + x_{US_B} + x_{S_BU_B} \leq 1 \quad \rightsquigarrow \leq 1\, R.\top\\
&\quad 0x_U + 0x_{S_B} + x_{U_B} + x_{US_B} + x_{S_BU_B} \leq 0 \quad \rightsquigarrow \leq 0\, U.B\\
&\text{with } x_U, x_{S_B}, x_{U_B}, x_{US_B}, x_{S_BU_B} \in \mathbb{N}
\end{aligned}$$

The system's infeasibility is caused by the encoding of the entailed role subsumption $S \sqsubseteq U$ (first three inequalities) and $\leq 0\, U.B$ (fourth inequality). If any of these four inequalities is removed, the remaining system becomes feasible.

5 Communication of Avalanche with QMediator

Avalanche is a complex rule-based system that implements a consequence-based reasoning algorithm presented in [32]. The algorithm manages the application of rules to an input ontology by traversing the completion graph. A dedicated module QMediator is called when a rule needs to expand the underlying graph or when a clash has been detected in a node due to the presence of qualified number restrictions. With the help of the module we can reduce the problem of deciding satisfiability of qualified number restrictions to the feasibility of inequalities, which gives us a clear advantage over other existing systems. To avoid circular dependencies between the two systems (considered an anti-pattern in software design) QMediator cannot call or access any data from Avalanche. Avalanche in its turn cannot directly call CPLEX.

During the execution of the algorithm the rules are being applied to an input ontology and a directed completion graph is constructed to store the inferred information.

There can be four types of nodes in the graph – identified nodes, anonymous nodes, auxiliary nodes, and two types of cloned nodes – a positive clone to test subsumption between concepts and a negative clone to test disjointness between concepts. If a positive/negative node becomes unsatisfiable then the subsumption/disjointness holds. Each node in a given completion graph is uniquely identified by its representative concept. A representative concept is either a concept (a concept name) declared in the original ontology or a concept created during the reasoning process. All nodes contain a set of subsumers and only identified nodes contain a set of possible subsumers. Subsumers are other concepts that subsume the representative concept of a node. Possible subsumers are collected by a dedicated rule and are needed for subsumption testing. As it can be guessed from their name, possible subsumers represent the subsumers that can possibly subsume the representative concept of a node. Thus, we can avoid performing unnecessary subsumption tests.

When a node is subsumed by qualified number restrictions it has to call the graph expansion rule. The rule in its turn will call QMediator and pass the corresponding information: the qualified number restrictions, the subsumers of the qualifications and their unsatisfiable concept conjunctions. After that, the mediator will transform this information into a linear program, and it will call CPLEX to solve it or in other words to find a model. The result of this call is returned to the rule. Thus, the rule will have all the necessary information to expand the graph or to make the node unsatisfiable by adding $\bot$ (bottom) to its subsumers. As a result, the expansion rule may create additional nodes in the graph - the anonymous nodes. An anonymous node represents a situation when a role filler is not a single concept (e.g., A) but rather an intersection of concepts ($A \sqcap B$).

In Figure 2 we show how the call to QMediator is integrated into the clash detection process. If a node becomes unsatisfiable, then the cause of the unsatisfiability has to be identified. If there is a logical clash (e.g., A and $\neg A$ are present in the node) then the corresponding ancestors of the node will be made unsatisfiable. However, if the clash is due to the presence of qualified number restrictions then the mediator should be called and it should be asked to recompute a more constrained model. If a new model was computed, the rules can continue to be applied. If there is no model and the node is an identified node, then the corresponding ancestors of the node should be made unsatisfiable. If the node is an anonymous node this information will be recorded to avoid having the QMediator recompute the same model. Otherwise, the node in question must be a positive/negative cloned node. In this case it can be concluded that the subsumption/disjointness holds and the node will be marked as unsatisfiable.

6 Performance Evaluation

We extended the test suite that we used in our previous work to evaluate the performance of Avalanche [32]. The test suite is composed of three different collections of test ontologies that will be presented below. We chose these ontologies to be classified to stress test the performance of Avalanche with respect to different applications of qualified cardinality restrictions. Some metrics about the test ontologies are shown in Figure 3. The ontologies differ by the number of axioms, concepts, roles, and qualified number restrictions.

Ontology Name	#A	#C	#R	#QCRs
canadian-parliament-factions-1	48	21	6	19
canadian-parliament-factions-2	56	24	7	25
canadian-parliament-factions-3	64	27	8	30
canadian-parliament-factions-4	72	30	9	35
canadian-parliament-factions-5	81	34	10	40
canadian-parliament-factions-10	121	49	15	54
canadian-parliament-full-factions-1	51	22	6	22
canadian-parliament-full-factions-2	60	25	7	30
canadian-parliament-full-factions-3	69	28	8	36
canadian-parliament-full-factions-4	78	31	9	42
canadian-parliament-full-factions-5	87	34	10	48
canadian-parliament-full-factions-10	132	49	15	69
C-SAT-exp-ELQ	26	10	4	13
C-UnSAT-exp-ELQ	26	10	4	13
genomic-cds rules-ELQ-fragment-1	716	358	1	357
genomic-cds rules-ELQ-fragment-2	718	359	1	357
genomic-cds rules-ELQ-fragment-3	718	359	1	357
genomic-cds rules-ELQ-fragment-4	1691	2775	1	8172

Fig. 3. Metrics of benchmark ontologies (#=Number of ..., A=Axioms, C=Concepts, R=Roles)

The first test collection models the House of Commons of the Canadian parliament [5] (see top part of Figure 4). It is our own collection of $\mathcal{ELQ}$ ontologies where we represent a real-world situation. There are two versions of this benchmark - short and full, each consisting of six variants. The variants differ by the number of included factions [5]. The only reasoners that can classify all variants of the simplest of these ontologies within the given time limit are Avalanche and Racer. Avalanche is the only reasoner that can classify all variants of these ontologies.

The second test collection (see middle part of Figure 4) uses synthetic concept templates. The original $\mathcal{ALCQ}$ concepts are shown below the table. They were manually rewritten into normalized $\mathcal{ELQ}$. The concept templates use a variable n that is increased exponentially. The numbers used in the template are bounded by the value of $2n$. The first template is satisfiable and the second one is unsatisfiable. Only Avalanche and Racer can classify all variants of these small ontologies within the time limit.

The third test collection (see bottom part of Figure 4) uses four $\mathcal{ELQ}$ fragments of a real world ontology, genomic-cds_rules [29], which was developed for pharmacogenetics and clinical decision support. It contains many concepts using QCRs of the form $= 2\, has.A_i$. However, in these fragments the concepts (A_i) occurring in the qualification of the QCRs do not interact with one another. This simplifies reasoning and all reasoners except Racer perform well. Avalanche (with the exception of the fourth fragment) and HermiT as well as FaCT++ and Konclude have similar runtimes. These fragments are interesting because the concept `<#human>` contains several hundred QCRs using the same role. This is one of the reasons why Racer times out for all fragments. At the moment we can classify only the above mentioned fragments of the ontology

Canadian Parliament										
Factions only						Full				
#F	**Ava**	**Fac**	**Her**	**Kon**	**Rac**	**Ava**	**Fac**	**Her**	**Kon**	**Rac**
10	1.1	TO	TO	TO	0.12	1.4	TO	TO	TO	TO
5	0.56	TO	TO	TO	0.12	0.73	TO	TO	TO	TO
4	0.46	TO	TO	TO	0.11	0.58	TO	TO	TO	TO
3	0.36	TO	TO	TO	0.07	0.43	TO	TO	TO	TO
2	0.26	TO	TO	TO	0.07	0.33	TO	TO	TO	10.5
1	0.18	TO	TO	7.3	0.05	0.24	TO	TO	TO	0.44

	C-SAT-exp-ELQ					C-UnSAT-exp-ELQ				
n	**Ava**	**Fac**	**Her**	**Kon**	**Rac**	**Ava**	**Fac**	**Her**	**Kon**	**Rac**
40	1.3	TO	TO	TO	0.01	1.6	TO	TO	TO	0.01
20	1.3	TO	TO	TO	0.01	1.5	TO	TO	TO	0.01
10	1.2	TO	TO	TO	0.01	1.6	TO	TO	TO	0.01
5	0.95	6.3	4.4	0.91	0.01	1.8	TO	TO	784	0.01
3	1.1	0.17	0.18	0.33	0.01	1.6	0.25	1.15	1.18	0.01

Sat: $C \sqsubseteq (\leq n\, R.\neg A \sqcup \leq n{-}1\, R.\neg B) \sqcap \geq 2n\, R.\top \sqcap \leq n\, R.A \sqcap \leq n\, R.B$
Unsat: $C \sqsubseteq (\leq n{-}1\, R.\neg A \sqcup \leq n{-}1\, R.\neg B) \sqcap \geq 2n\, R.\top \sqcap \leq n\, R.A \sqcap \leq n\, R.B$

Satisfiability of concept `<#human>`					
Name	**Ava**	**Fac**	**Her**	**Kon**	**Rac**
genomic-cds_rules-ELQ-fragment-1	0.75	27.7	0.87	27.7	TO
genomic-cds_rules-ELQ-fragment-2	1.2	28.2	1.14	28.3	TO
genomic-cds_rules-ELQ-fragment-3	4.8	28.8	1.27	26.3	TO
genomic-cds_rules-ELQ-fragment-4	26.7	28.8	4.4	29.4	TO

Fig. 4. Benchmark runtimes in seconds with a timeout of 1000 seconds (TO=timeout, #F=Number of Factions, Ava=Avalanche, Fac=FaCT++, Her=HermiT, Kon=Konclude, Rac=Racer)

in question. Our ultimate goal is classify the entire ontology. As we know, no other reasoner can do it yet.

As compared to our previous work [32], the performance of Avalanche has already been greatly improved. However, we expect to achieve even better results in future. Avalanche's speed for the Canadian Parliament ontologies has been improved by several orders of magnitude. Previously it could not classify the version of Canadian Parliament with 10 factions within 10,000 seconds. The reason for such a change is mainly due to the improved communication with QMediator. Previously we delegated all computations that concerned qualified number restrictions to the dedicated module. After a scrupulous analysis of Avalanche's runtime we noticed that a lot of time is spent in the module to solve rather simple cases. It appears that Linear Programming methods are typically used to solve feasible problems. If a problem is infeasible then it should be considered as erroneous and it has to be remodelled. However, in our case we do not consider infeasible models to be erroneous. On the contrary, they help us to discover valuable knowledge about the ontology in question, e.g subsumption or disjointness. As a result, we identified several cases where feasibility/infeasibility information can be discovered without CPLEX. For example, we do not need to call QMediator when

we have only at-least or only at-most restrictions. In the former case we simply connect with an edge the node that contains the number restrictions and the nodes that contain role fillers as their representative concepts. In the latter case we do not need to do anything because at-most inequalities do not require us to create successors (remember that 0 will satisfy $\leq 5R.C$). We have a special treatment when we have a set of at-least and at-most inequalities and all at-most inequalities are of cardinality 0. We also check if we can reduce all at-least inequalities and all at-most inequalities to only one inequality. For example, $\geq 3R.C$ and $\geq 5R.C$ could be replaced by $\geq 5R.C$. Similarly, $\leq 0R.C$ and $\leq 6R.C$ could be replaced by $\leq 0R.C$. Further, we have other ways to determine early infeasibility. For example, $\leq 3R.C$ and $\geq 1R.D$ would be infeasible if C is subsumed by D. Thus, QMediator is now called only when it cannot be avoided.

Although Racer can classify some of the test ontologies faster than Avalanche, we are not discouraged by this fact because we know exactly how we have to improve Avalanche in order to achieve comparable or even better results. In particular, we will be working on a reimplementation of the strategy that is used to apply rules to nodes.

The experiments were performed on a MacBook Pro (2.6 GHz Intel Core i7 processor, 16GB memory). The comparison results (average of 3 runs) are shown in Figure 4. We compared Avalanche with major OWL reasoners: FaCT++ (1.6.4) [8], HermiT (1.3.8) [18], Konclude (0.6.2) [22], and Racer (3.0) [15, 14, 27]. In fact, Racer is the only other available OWL reasoner using an ILP component for reasoning about QCRs in contrast to [21] where ILP is used in the context of probabilistic reasoning. The algorithms implementing Racer's ILP component are in general best-case exponential with respect to the number of QCRs given for one concept. Another reasoning approach for $\mathcal{ALCQ}$ [16] used SMT with a theory that is a specific and computationally much cheaper subcase of Linear Arithmetic under the Integers but this approach suffers from inefficiencies for nested QCRs where reasoning involves backtracking. It would also not scale well for role hierarchies and its extension to inverse roles is an open problem.

7 Conclusion

In this work we presented a hybrid architecture for reasoning about description logics supporting role hierarchies and QCRs. It allows us to reduce the QCR satisfiability problem to a feasibility problem. We tested our system and identified ontologies that cannot be classified by other reasoners in a reasonable amount of time. We almost finished extending the architecture to cover $\mathcal{ALCHQ}$. Our ultimate goal is to extend our architecture to the DL $\mathcal{ALCHIQ}$ by adding inverse roles ($\mathcal{I}$).

References

1. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Proc. of IJCAI. pp. 364–369 (2005)
2. Baader, F., Sattler, U.: An Overview of Tableau Algorithms for Description Logics. Studia Logica 69(1), 5–40 (2001)
3. Barnhart, C., Johnson, E.L., Nemhauser, G.L., Savelsbergh, M.W.P., Vance, P.H.: Branch-and-price: Column generation for solving huge integer programs. Operations Research 46(3), 316–329 (1998)

4. Bate, A., Motik, B., Cuenca Grau, B., Simančík, F., Horrocks, I.: Extending consequence-based reasoning to $\mathcal{SRIQ}$. In: Proc. of KR. pp. 187–196 (2016)
5. Canadian Parliament: https://en.wikipedia.org/wiki/House_of_Commons_of_Canada
6. Chvatal, V.: Linear Programming. Freeman (1983)
7. Dantzig, G.B., Wolfe, P.: Decomposition principle for linear programs. Operations research 8(1), 101–111 (1960)
8. FaCT++: http://owl.cs.manchester.ac.uk/tools/fact/
9. Faddoul, J., Haarslev, V.: Algebraic tableau reasoning for the description logic $\mathcal{SHOQ}$. Journal of Applied Logic 8(4), 334–355 (2010)
10. Faddoul, J., Haarslev, V.: Optimizing algebraic tableau reasoning for $\mathcal{SHOQ}$: First experimental results. In: Proc. of DL. pp. 161–172 (2010)
11. Farsiniamarj, N., Haarslev, V.: Practical reasoning with qualified number restrictions: A hybrid Abox calculus for the description logic $\mathcal{SHQ}$. AI Communications 23(2-3), 334–355 (2010)
12. Freund, R., Mizuno, S.: Interior point methods: current status and future directions. Optima 51, 1–9 (1996)
13. Gilmore, P.C., Gomory, R.E.: A linear programming approach to the cutting-stock problem. Operations research 9(6), 849–859 (1961)
14. Haarslev, V., Hidde, K., Möller, R., Wessel, M.: The RacerPro knowledge representation and reasoning system. Semantic Web 3(3), 267–277 (2012)
15. Haarslev, V., Möller, R.: RACER system description. In: Proc. of IJCAR. pp. 701–705 (2001)
16. Haarslev, V., Sebastiani, R., Vescovi, M.: Automated reasoning in $\mathcal{ALCQ}$ via SMT. In: Proc. of CADE. pp. 283–298 (2011)
17. Hansen, P., Jaumard, B., de Aragão, M.P., Chauny, F., Perron, S.: Probabilistic satisfiability with imprecise probability. International Journal of Approximate Reasoning 24(2-3), 171–189 (May 2000)
18. HermiT: http://www.hermit-reasoner.com/download.html
19. Hollunder, B., Baader, F.: Qualifying number restrictions in concept languages. In: Proc. of KR. pp. 335–346 (1991)
20. Jaumard, B., Hansen, P., de Aragão, M.P.: Column generation methods for probabilistic logic. ORSA Journal on Computing 3(2), 135–148 (1991)
21. Klinov, P., Parsia, B.: Pronto: A practical probabilistic description logic reasoner. In: Uncertainty Reasoning for the Semantic Web II, pp. 59–79. Springer (2013)
22. Konclude: http://www.derivo.de/en/produkte/konclude/
23. Lübbecke, M., Desrosiers, J.: Selected topics in column generation. Operations Research 53, 1007–1023 (2005)
24. Megiddo, N.: On the complexity of linear programming. In: Advances in Economic Theory, pp. 225–268. Cambridge University Press (1987)
25. Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization. Wiley (1988)
26. Ohlbach, H., Köhler, J.: Modal logics, description logics and arithmetic reasoning. Artificial Intelligence 109(1-2), 1–31 (1999)
27. Racer: https://www.ifis.uni-luebeck.de/index.php?id=385
28. Roosta Pour, L., Haarslev, V.: Algebraic reasoning for $\mathcal{SHIQ}$. In: Proc. of DL. pp. 530–540 (2012)
29. Samwald, M.: Genomic CDS: an example of a complex ontology for pharmacogenetics and clinical decision support. In: 2nd OWL Reasoner Evaluation Workshop. pp. 128–133 (2013)
30. Simančík, F., Motik, B., Horrocks, I.: Consequence-based and fixed-parameter tractable reasoning in description logics. Artificial Intelligence 209, 29–77 (2014)
31. Vanderbeck, F.: Branching in branch-and-price: a generic scheme. Mathematical Programming 130(2), 249–294 (2011)
32. Vlasenko, J., Daryalal, M., Haarslev, V., Jaumard, B.: A saturation-based algebraic reasoner for $\mathcal{ELQ}$. In: PAAR@IJCAR. pp. 110–124. Coimbra, Portugal (2016)