Generalised Predicates in Lazy Clause Generation using Global Propagation

Julius Gvozdiovas
Delft University of Technology, Netherlands

Emir Demirović
Delft University of Technology, Netherlands

Abstract

Lazy clause generation (LCG) has proven to be an effective framework for constraint programming solvers. A key component of LCG is explanations: propositional statements that capture the logical reasons for conflicts or propagations. Considerable effort is devoted to constructing explanations that are as general as possible, as solver efficiency critically depends on their generality. Conventionally, these explanations are represented as conjunctions of literals, where each literal is a predicate over a single integer variable. In this work, we outline our plans to study a natural generalisation in which literals may involve multiple variables, allowing arbitrary linear inequalities within explanations. Such explanations are strictly more general than standard LCG explanations and have the potential to significantly improve solving performance. Our approach is closely related to extended resolution, in which new variables are introduced as functions over existing variables. While this technique is known to be theoretically powerful, it has not yet been well understood to be integrated into constraint programming solvers. To further support this approach, we propose several propagation schemes to ensure efficient channelling and outline an experimental plan to be used in the future to evaluate the resulting framework.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming

Keywords and phrases constraint programming

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

Constraint programming (CP) is a paradigm that allows solving problems composed of variables and constraints over those variables. The problem has to be modelled by specifying the domains of the variables and the constraints, to which a solver then either finds a satisfiable solution or proves infeasibility.

CP solvers support a variety of constraints that can be used in models. Each constraint has an associated propagator, which prunes values from variable domains. Modern CP solvers incorporate lazy clause generation [24] (LCG) – each propagator generates a clausal explanation for an inference, which is then given to a conflict-driven SAT solver, which can learn new clauses, called nogoods, from conflicts. The quality of the explanations greatly impact learned nogoods, as general explanations allow pruning the search space more.

Prior attempts to extend LCG have been made. Baauw et al. [1] introduces lazy linear generation (LLG), which generates linear inequalities instead of boolean clauses over predicate literals – however, it sometimes fails to learn asserting nogoods, due to the so-called rounding problem [23]. Their approach is similar to pseudo-Boolean approach [3], where Boolean clauses are replaced by 0-1 integer inequalities, and non-clausal learning is performed. LLG was further expanded on by Flippo et al. [11] in their work on hypercube linears, which extend LCG explanation by allowing linear inequalities on the right side of the inferences – this permits more general explanations, but requires weakening during conflict resolution.

Chu and Stuckey [4] incorporated extended resolution [29] (ER) into CP, which is known

to be exponentially more powerful than regular resolution [13] (which is what the underlying SAT solvers generally simulate). Their approach requires explicitly channelling the meaning of the extended variables, resulting in large overhead for ad-hoc encodings.

We extend the latter approach, generalizing LCG, by permitting arbitrary linear inequalities as predicate literals, while still using clausal learning. Explanations generated this way can be strictly stronger than classic LCG counterparts.

In order to support the extension, we consider multiple propagation schemes. We incorporate ideas from satisfiability modulo theories (SMT) to use linear programming to unit propagate nogoods. We compare such an approach to a simple propagation scheme, where each linear inequality literal is channelled to the corresponding bounds-consistent propagator.

We have devised a number of empirical experiments to evaluate the effect of the more expressive literals, to determine the impact of different propagation schemes on the performance of our solver, the generality of the produced explanations, and the overhead of using the linear inequality propagation in the main propagation loop.

The rest of the paper is structured as follows. Section 2 explains the prerequisite concepts required to understand the rest of the paper. In Section 3, we give the connection between existing work and our approach. Section 4 describes our main theoretical contributions, and the plan for empirical evaluation is in Section 5. Finally, we conclude in Section 6.

2 Background

Constraint satisfaction problems (CSP) are composed of a set of variables $\mathcal{X}$, integer domains of those variables $\mathcal{D}$ (domains of the individual variables are $\mathcal{D}(x), x \in \mathcal{X}$), and constraints over (subsets of) those variables. A constraint is defined as a general predicate over variables, restricting the final values. An example constraint $\langle x + y = z \rangle$ would require that the final assignment of variables maps z such that it is equal to the sum of assignments of x and y. The overall goal of constraint programming (CP) is to solve these problems, i.e. to find a satisfying assignment from each variable to a value in its domain, or else to show that this is not possible.

CP solvers are typically comprised of two parts: backtracking search, where the solver attempts to assign variables, and backtracks if that eventually leads to infeasibility; and propagation, where search space is pruned by removing infeasible assignments. Propagation is carried out by specialized propagators, which transform the CSP without changing the overall feasibility of the CSP, by removing values from variable domains which are infeasible based on the constraints. For example, the constraint $\langle x + y = z \rangle$ might be propagated by setting $D'(z) := D(z) \cap \{x' + y' \mid x' \in D(x), y' \in D(y)\}$.

Modern CP solvers use lazy clause generation [24] (LCG) to combine the strength of conflict-driven clause learning [22] (CDCL) present in Boolean satisfiability (SAT) solving, with constraint propagation in CP. The key idea is that propagators, instead of directly pruning domains, instead generate clauses, which are then propagated by the underlying CDCL SAT solver. In LCG, integer domains are represented in propositional logic using predicate literals of the form $[x \oplus v]$, where x is a variable, v is an integer value, and $\oplus \in \{\leq, \geq, =, \neq\}$ is a permitted operator. An example of a generated explanation could be $[x \leq 2] \wedge [y \leq 5] \rightarrow [z \leq 7]$, to explain why $8 \notin D(z)$ when the upperbound (the largest value in the domain) of x is 2 and the upperbound of y is 5. Note that the explanation is given in the form of $\bigwedge l_i \rightarrow r$, which is equivalent to clausal form $(\bigvee \neg l_i) \vee r$.

In this work, we reference linear programming (LP), thus we cover it here briefly as

well (we refer the reader to introductory textbooks for a better introduction [30]). LP concerns itself with systems of linear inequalities over real-valued variables, and seeks to maximize a linear function over those variables. LP problems are commonly stated as: maximize $\sum_{j=1}^{n} c_j x_j$, subject to $\sum_{j=1}^{n} a_{i,j} x_j \leq b_i, i \in \{1, 2, \ldots, m\}$, with variables $x_j \in \mathbb{R}$, and parameters $c_j, b_i, a_{i,j} \in \mathbb{R}$. The commonly used *simplex algorithm* can solve linear programming problems on average in polynomial time, by traversing the vertices of the polyhedra implied by the system of linear inequalities. For a given (primal) LP problem, a dual LP problem exists: minimize $\sum_{i=1}^{m} b_i y_i$, subject to $\sum_{i=1}^{m} y_i a_{i,j} \leq c_j, j \in \{1, 2, \ldots, n\}$, with $y_i \in \mathbb{R}$ being the variables now. If a given primal LP problem is infeasible, then its dual will be feasible, with its optimal solution providing a linear combination of the linear inequalities in the primal problem that cause the infeasibility.

3 Related Work

Lazy clause generation [24] (LCG) was introduced as a way to incorporate CDCL [22] learning in CP. It has stood the test of time among other approaches [6, 18, 17]. Since then, alternative approaches that generalize LCG have been proposed.

Veksler and Strichman [31] introduce a scheme where specific combinations of constraints are combined together to learn new constraints, which often results in stronger learning compared to LCG. However, not all possible cases are covered, thus sometimes the scheme must fall back to clausal learning.

Baauw et al. [1] extend existing alternative approaches of learning in SAT [3, 16, 23] to CP, using cutting planes to learn integer linear inequality constraints. However, after conflict analysis, their learned nogood sometimes fails to propagate, meaning that search space is not meaningfully reduced. We build upon the explanations for propagations that they derived in their work. Some of their explanations require big-M formulations, which we can avoid by using disjunctions.

Flippo et al. [11] extend LCG by permitting linear inequalities on the right side of explanations. They devise a scheme which always produces asserting nogoods after conflict analysis, however, they perform a weakening of the explanation in certain cases, to stick with the hypercube linear format. Our approach can be seen as a generalization of their approach, further permitting linear inequalities on the left side of the explanation.

Chu and Stuckey [4] have introduced the concept of extended variables [29] into CP, which are known to be exponentially more powerful than regular resolution [13], but have only been applied to a limited extent in CDCL solvers [15, 2, 5]. The main issue that they face is that each introduced variable must be channelled individually, resulting in a large overhead. Their work was expanded on to implement specific constraint propagators, such as `Cumulative` [19], to mixed results. Our work conceptually builds upon this, but we aim to address the channelling problem by reasoning about the newly introduced literals in a global manner using linear programming techniques.

Usage of the simplex algorithm to propagate linear inequalities has been widely studied in SMT literature [9, 27, 7, 21, 20]. Existing linear programming approaches, such as the simplex algorithm, do not directly and efficiently compute the reduced search space [28], and thus cannot be directly integrated into CP solvers, which require access to individual variable bounds. Dutertre and De Moura [9] show that in SMT context, full propagation using the simplex algorithm is not efficient, and that it is better to use heuristic propagation, such as bounds reasoning. We investigate this claim in constraint programming context, specifically when generalising LCG.

There are existing approaches to integrating global propagators in CP. Solver Tempo [14] uses difference logic to globally reason over predicate literals of the form $[x - y \leq v]$. Devriendt et al. [8] use a global LP relaxation to help solve SAT in pseudo-Boolean [3] context. LCG solvers Chuffed and OR-Tools [26] incorporate some degree of global propagation via linear programming, but the effectiveness of the technique has not been published. Our work seeks to fill in this gap by investigating the impact, but we further extend the concept by permitting explanations to contain linear inequalities.

4 Our contribution

We propose an extension to LCG, by permitting not only atomic predicate literals ($[x \oplus v], \oplus \in \{\leq, \geq, =, \neq\}$), but also arbitrary linear inequalities, i.e. $[\sum a_i x_i \leq b]$, as predicate literals. For simplicity of implementation, we remove the use of $\{=, \neq\}$ operators as they can be simulated using inequalities.

The key benefit of our approach is that propagators can use these extended predicate literals in their explanations, enabling much more general explanations. For example, explanations such as $[2x - y \leq 1] \wedge [z - x \geq 3] \rightarrow [x + y + z \leq 7]$ are now permitted, with linear inequalities on both the left-hand and right-hand sides. During conflict analysis, produced nogoods will contain such linear inequality literals. We consider multiple approaches to permit unit propagation.

One approach, previously explored by Chu and Stuckey [4], is to have channelling constraints for each literal. For example, $[2x - y \leq 1]$ would have a corresponding boolean variable, $[2x - y \leq 1] = b_1$, and a reified linear constraint, $b \leftrightarrow 2x - y \leq 1$. If the literal is set to true, the linear inequality is propagated. Similarly, if the linear inequality would conflict, the literal is set to false – allowing nogood unit propagation. A benefit of this approach in context of CP is that individual variable domains are always propagated, without special upkeep, by propagating the linear inequality.

Another approach, inspired by SMT, is to consider the LP problem consisting of the currently held true linear inequalities, and detect feasibility using the simplex algorithm. We dub such propagation scheme as the *simplex propagator* and discuss it in Subsection 4.1.

4.1 Simplex propagator

The simplex propagator reasons over the entire CSP as a whole. During search, the underlying SAT solver might force certain literals to hold true, through unit propagation. Thus, a partial assignment (in SAT view), consisting of linear inequality predicate literals that hold true, represent a system of linear inequalities – an LP problem. Thus, the simplex algorithm can be ran to detect infeasibility in such state.

Suppose that $[x + y \geq 5], [z \geq 0], [w \geq 0]$ all hold in the current assignment. The simplex propagator should infer that in the clause $[x + y + z + w \leq 4] \vee [a + b \geq 1]$, the literal $[x + y + z + w \leq 4]$ is infeasible, in order to propagate $[a + b \geq 1]$ as true. A naive method to achieve this is to run the current assignment, but with the literal added to the system – to detect whether infeasibility is reached. This method would potentially require running the simplex algorithm for each literal in each clause.

Our suggestion is then to store a *witness* value for each literal, which is initially calculated by running the simplex algorithm for each literal, storing a feasible solution for that literal.

In order to efficiently globally propagate, we attach a witness to each literal in each clause, based on a feasible solution for that literal within the current partial assignment. When a linear inequality literal is assigned as true by unit propagation, each witness can be

checked for infeasibility by plugging in the witness into the newly propagated inequality – which is faster and simpler than running the entire simplex algorithm to detect feasibility. By using the two-watched literal scheme [12], this requires only checking $2n$ literals, where n is the number of clauses. If a witness does not violate the inequality, then in the system of linear inequalities with that literal, the witness is still a valid solution. However, if the witness violates the inequality, then the literal might have become infeasible in the current assignment. In that case, the simplex algorithm is run assuming the literal for that witness. It is either found infeasible (possibly permitting unit propagation), or feasible – in that case, the newly found feasible solution is assigned as the new witness. The witnesses, similar to watched literals, have a nice property that they remain valid even after backtracking.

A possible optimization to finding new feasible witnesses for literals is that after each pivot in the simplex algorithm, it can be checked against some or all literals whose witnesses are not currently valid, to check if that would be a valid new witness. This can, in some cases, skip some of the runs of the simplex algorithm, by finding a feasible witness before then. Even for literals which already have a witness, it might be possible to find a better witness (e.g. by distance to the linear equality line), skipping a simplex algorithm run later, or by keeping track of multiple witnesses.

When a literal is found to be infeasible, we need to mark it as false for unit propagation. However, for CDCL conflict analysis procedure, the unit propagation must have a reason. We adopt a technique from SMT [9] to generate such reasons using Farkas' Lemma composed of linear inequalities in the current system. The explanation, together with the new propagation, can then be added to the clause database, causing unit propagation to propagate the literal as false, possibly prompting more propagation.

For usage in CP, propagators generally query the bounds of variables. However, as noted before [28], simplex algorithm by default does not calculate the reduced search space, including the variable bounds. For now, our approach is to keep track of currently known variable bounds. Suppose that $x \leq v$ is known as the lowest upper bound of x. Then a vacuously true clause $[x \leq v - 1] \vee [x \geq v]$ will not propagate. After propagation, if $x = v$ is no longer a viable assignment, then unit propagation on that clause would propagate $[x \leq v - 1]$. We can detect such propagations, and run the simplex algorithm on the current partial assignment, maximizing x – thus finding[1] the new lowest upper bound of x, say u, after which the former clause can be replaced by $[x \leq u - 1] \vee [x \geq u]$. This procedure can be run incrementally over each variable, since only the objective function, not the overall LP problem, is changed.

Our propagation scheme only detects conflicts when it is impossible to assign an integer value to some variable x, while assigning real number values to the rest of the variables. It thus solves a relaxation, at the benefit of running on average in polynomial time. On the other hand, when searching for witnesses, instead of accepting a real-valued solution from the simplex algorithm, we can run branch-and-cut [25] to find an integral solution, albeit possibly in exponential time. This is a classic trade-off between algorithmic complexity and propagation strength. We plan to empirically evaluate both approaches, as described in Section 5.

[1] In practice, the simplex algorithm will not find an integer, but a real-valued number u'. The true upper bound can then be assumed to be $u = \lfloor u' \rfloor$. If this new value is not viable, then the clause that is generated will cause a conflict.

4.2 Semantic nogood minimization

In LCG, semantic minimization is used to ensure that nogoods unit propagate as soon as possible [10]. For example, for the nogood $[x \leq 5] \wedge [x \leq 10] \rightarrow \bot$ to unit propagate, $x \geq 6$ must be assigned. However, $[x \leq 5] \rightarrow \bot$ can unit propagate directly. The latter nogood can replace the former, since $x \leq 5 \Rightarrow x \leq 10$, thus $[x \leq 5] \wedge [x \leq 10] \rightarrow \bot \equiv [x \leq 5] \rightarrow \bot$. In general, such nogoods can be minimized by individually considering domains of variables in the nogood.

The example can be generalized to linear inequalities (e.g. with $[x + y \leq 5]$ and $[x + y \leq 10]$). However, a distinct case occurs with arbitrary linear inequalities. Consider the nogood $[x + y \leq 8] \wedge [y - z \leq 5] \wedge [x + z \leq 1] \rightarrow \bot$. Here, inequalities $y - z \leq 5$ and $x + z \leq 1$ can be combined to infer $x + y \leq 6$, which directly implies $x + y \leq 8$. We can thus minimize the nogood to $[y - z \leq 5] \wedge [x + z \leq 1] \rightarrow \bot$. In general, we use linear inequality propagation to determine whether a literal in a nogood is redundant, by asserting it as false, and the remaining nogoods as true, and checking for infeasibility.

There are multiple qualitative differences arising between LCG and our system. While in LCG, removal of a literal can be justified with only a single other literal, in our system, we might need to consider an arbitrary number of other literals. This means that detecting redundancy is not trivial, and requires running the simplex algorithm to detect indirect relations. Furthermore, the final nogood in LCG is canonical (there is an objective best choice for the final nogood), whereas there might be multiple choices in our system, when reasoning over integers.

Consider the nogood $[3x - y \geq 1] \wedge [3x + y \geq 2] \wedge [y \geq 0] \wedge [y \leq 1] \rightarrow \bot$. Exactly one of the first two literals can be removed, since they are both implying that $x \geq 1$ when $y \in [0, 1]$. However, it is not obvious which literal should be removed, or even, if it would be preferable to replace them by $[x \geq 1]$. Over the real numbers, none of the choices dominate each other, but detecting such infeasibilities over the integers is NP-hard in general.

5 Experiments

No implementation exists yet, but is planned as our work progresses. This section thus describes the planned experiments.

Impact of generalized explanations Our primary motivation is that linear inequality predicates permit more general explanations. We thus plan to implement two versions of each propagator – one with generalized explanations using arbitrary linear inequality predicate literals, and one with atomic predicate literals. In order to assess the impact, we plan to measure the overall number of conflicts, the average number of literals in each propagation, and the average literal block distance of produced nogoods, comparing the two solvers using different types of explanations.

Impact of semantic minimization Semantic minimization in the context of LCG is known to increase solver performance [10]. When it is computationally easy to execute, it offers a theoretical basis for permitting nogoods to propagate earlier, with empirical results showing that such propagations matter in practice. With linear inequality literals, the theoretical grounding is the same, however, the computational task of minimizing nogoods is considerably more difficult. In the worst case, there may be redundant literals which can only be detected as redundant over the integers (and thus being NP-hard to recognize).

We plan to evaluate multiple semantic minimization schemes: no minimization, minimization considering pairs of inequalities over the same sets of variables (i.e. $[x + y \leq 4]$ vs. $[x + y \leq 7]$ are both over $\{x, y\}$), full simplex-based minimization, full simplex + branch-and-cut minimization. For each minimization scheme, we plan to run the solver over multiple instances, measuring how many literals, on average, were removed from nogoods, as well as average number of times nogoods propagate.

Impact of propagation strength Existing work in SMT literature [9, 21] suggests that full propagation using the simplex propagator is not worth the overhead. We seek to evaluate this claim in CP context, where some constraints are decomposed into linear inequality clauses, and where propagators generate linear inequality clauses.

We compare multiple propagation strengths: bound-consistent propagation over individual linear inequalities (channelling), simplex propagation, and simplex + branch-and-cut propagation. Each setup propagates strictly stronger than the previous ones, thus we keep track of which propagations are missed in each case.

For each setup, we measure the total number of conflicts (representing size of the search graph), as well as average depth loss, measured in the number of decisions made after a stronger propagator would have cut that branch of the search space.

Furthermore, we measure the overhead of each method, using the following metrics:

- Bound-consistent channelling – total number of calls to each propagator.
- Simplex – total number of simplex pivots.
- Simplex and branch-and-cut – total number of simplex pivots plus number of branch-and-cuts.

We heuristically assume that these metrics represent the actual runtime overhead of each method, without being subject to the usual problems with runtime measurement.

We plan to run each instance with each propagation strength multiple times, averaging the results, in order to compare the propagation strengths on a per instance basis. The overall goal is to compare the benefit of the stronger propagation versus detriment of the additional overhead.

6 Conclusion

We extended lazy clause generation with arbitrary linear inequality literals, permitting more general explanations, with the support of multiple propagation schemes. We have devised a number of experiments to measure the impact of more general explanations, and to test the effectiveness and overhead of supporting propagators. Future work could expand with different generalized literal scheme, coupled together with a corresponding propagation scheme.

References

1 Robbin Baauw, Maarten Flippo, and Emir Demirović. Conflict analysis based on cutting-planes for constraint programming. In *31st International Conference on Principles and Practice of Constraint Programming (CP 2025)*, pages 4:1–4:19. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2025.

2 Sam Buss, Jonathan Chung, Vijay Ganesh, and Albert Oliveras. Extended resolution clause learning via dual implication points. *arXiv preprint arXiv:2406.14190*, 2024.

3 Donald Chai and Andreas Kuehlmann. A fast pseudo-boolean constraint solver. In *Proceedings of the 40th annual Design Automation Conference*, pages 830–835, 2003.

4 Geoffrey Chu and Peter J Stuckey. Structure based extended resolution for constraint programming. *arXiv preprint arXiv:1306.4418*, 2013.

5 Jonathan Chung. Prioritized unit propagation and extended resolution techniques for SAT solvers. 2023.

6 Rina Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41(3):273–312, 1990.

7 David Detlefs, Greg Nelson, and James B Saxe. Simplify: a theorem prover for program checking. *Journal of the ACM (JACM)*, 52(3):365–473, 2005.

8 Jo Devriendt, Ambros Gleixner, and Jakob Nordström. Learn to relax: Integrating 0-1 integer linear programming with pseudo-Boolean conflict-driven search. *Constraints*, 26(1):26–55, 2021.

9 Bruno Dutertre and Leonardo De Moura. A fast linear-arithmetic solver for DPLL (T). In *International Conference on Computer Aided Verification*, pages 81–94. Springer, 2006.

10 Thibaut Feydy, Andreas Schutt, and Peter Stuckey. Semantic learning for lazy clause generation. In *TRICS workshop, held alongside CP*. Citeseer, 2013.

11 Maarten Flippo, Peter J. Stuckey, and Emir Demirović. Resolution meets cutting planes: Introducing hypercube linear resolution. Not yet published.

12 Ian P Gent. Optimal implementation of watched literals and more general techniques. *Journal of Artificial Intelligence Research*, 48:231–252, 2013.

13 Armin Haken. The intractability of resolution. *Theoretical computer science*, 39:297–308, 1985.

14 Emmanuel Hebrard. Disjunctive scheduling in Tempo. In *31st International Conference on Principles and Practice of Constraint Programming (CP 2025)*, pages 13–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2025.

15 Jinbo Huang. Extended clause learning. *Artificial Intelligence*, 174(15):1277–1284, 2010.

16 Dejan Jovanović and Leonardo De Moura. Cutting to the chase: solving linear integer arithmetic. *Journal of automated reasoning*, 51(1):79–108, 2013.

17 George Katsirelos. *Nogood processing in CSPs*. PhD thesis, 2009.

18 George Katsirelos and Fahiem Bacchus. Generalized nogoods in CSPs. In *AAAI*, volume 5, pages 390–396, 2005.

19 MAA Kienhuis. Solving Cumulative with Extended Resolution in LCG solvers. *TU Delft repository*, 2025.

20 Tim King, Clark Barrett, and Cesare Tinelli. Leveraging linear and mixed integer programming for SMT. In *2014 Formal Methods in Computer-Aided Design (FMCAD)*, pages 139–146. IEEE, 2014.

21 David Monniaux. A survey of satisfiability modulo theory. In *International Workshop on Computer Algebra in Scientific Computing*, pages 401–425. Springer, 2016.

22 Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535, 2001.

23 Robert Nieuwenhuis. The IntSat method for integer linear programming. In *International Conference on Principles and Practice of Constraint Programming*, pages 574–589. Springer, 2014.

24 Olga Ohrimenko, Peter J Stuckey, and Michael Codish. Propagation = lazy clause generation. In *International Conference on Principles and Practice of Constraint Programming*, pages 544–558. Springer, 2007.

25 Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33(1):60–41, 03 1991. URL: `https://www.proquest.com/scholarly-journals/branch-cut-algorithm-resolution-large-scale/docview/926130616/se-2`.

26 Laurent Perron, Frédéric Didier, and Steven Gay. The CP-SAT-LP solver (invited talk). In *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*, pages 3:1 – 3:2. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2023.

27 Harald Rueß and Natarajan Shankar. Solving linear arithmetic constraints. *SRI International, Computer Science Laboratory*, 2004.

28 Kish Shen and Joachim Schimpf. Eplex: Harnessing mathematical programming solvers for constraint logic programming. In *International Conference on Principles and Practice of Constraint Programming*, pages 622–636. Springer, 2005.

29 Grigori S Tseitin. On the complexity of derivation in propositional calculus. In *Automation of reasoning: 2: Classical papers on computational logic 1967–1970*, pages 466–483. Springer, 1983.

30 Robert J Vanderbei. Linear programming: foundations and extensions. *Journal of the Operational Research Society*, 49(1):94–94, 1998.

31 Michael Veksler and Ofer Strichman. Learning general constraints in CSP. *Artificial Intelligence*, 238:135–153, 2016.