Core-Guided and Core-Boosted Search for CP

Graeme Gange[1], Jeremias Berg[2], Emir Demirović[3], and Peter J. Stuckey[1,4]

[1] Monash University, Australia, {graeme.gange,peter.stuckey}@monash.edu
[2] HIIT, Department of Computer Science, University of Helsinki, Finland, jeremias.berg@helsinki.fi
[3] University of Melbourne, Australia, emir.demirovic@unimelb.edu.au
[4] Data61, CSIRO, Australia

Abstract. Core-guided search has proven to be the state-of-the-art in finding optimal solutions for maximum Boolean satisfiability and these techniques have recently been successfully imported in constraint programming. While effective on a wide range of problems, the methods are direct translations of their propositional logic counterparts. We propose two reformulation techniques that take advantage of the rich formalism offered by constraint programming rather than relying on propositional logic strategies, and generalise two existing techniques to improve core-extraction and the overall performance. Our experiments demonstrate the effectiveness of our approaches over the conventional (core-guided) CP methods, both in terms of proving optimality and quickly computing high-quality solutions.

1 Introduction

Discrete optimisation problems are ubiquitous: they include scheduling, rostering, production planning, and many other important questions. Optimal or good solutions to these problems result in more efficient use of scarce resources, saving time, money and the environment. Because of their importance, there are many paradigms to solve optimisation problems, including Mixed Integer Programming (MIP), Constraint Programming (CP), Maximum Satisfiability (MaxSAT) and local search. In this work, we focus on constraint programming, and in particular on improving core-guided search for CP.

There are two main approaches to optimization in constraint programming: 1) *branch-and-bound*, that iteratively improves a best known solution during search, and 2) *core-guided search*, where the algorithm assumes all constraints can be satisfied, and upon detecting infeasibility, relaxes the assumptions and reiterates. Branch-and-bound and core-guided search can be seen as upper and lower bounding methods, respectively. Branch and bound is by far the most used approach in CP.

Core-guided search originates from the MaxSAT community, where problems are specified as propositional logic formulae. It is one of the central approaches to *complete* MaxSAT solving, as a large portion complete solvers in the annual MaxSAT Evaluation use the core-guiding methodology. In contrast, in CP, core-guided approaches have only recently been developed. In particular, the solver

LCG-Glucose-UC, based on core-guiding, achieved the third highest score in the MiniZinc Challenge 2016, and OR-tools has introduced core-guided search in 2018 as part of their parallel solver. While these techniques have seen success in CP, they do not, in fact, use the expressivity offered by the constraint programming framework. Indeed, the methods are direct translations of the MaxSAT approaches, which were originally developed for the low-level language of propositional logic.

In this work, we advance the state-of-the-art for core-guided search in CP by exploiting the high-level language constructs offered by constraint programming. We provide two novel reformulation techniques for CP which are unique to constraint programming. Moreover, we generalise two existing techniques for CP, namely assumption probing to improve core-extraction and core-boosting [1] to increase the overall performance. We also discuss an issue with using explanation lifting with the conventional CP core-guided approach and note a number of techniques adapted from the MaxSAT community which play an important role in obtain high quality results. Our experiments on benchmarks from the MiniZinc Challenge show improvements over the conventional (core-guided) CP approaches, both in terms of the number of instances solved to optimality and the ability to quickly produce high-quality solutions.

The rest of the paper is organised as follows. In the next section, we introduce basic notations; constraint programming solvers with explanations, and core-guided MaxSAT methods along with their conventional translation for CP. Our main contributions are given in Section 3, together with additonal techniques that improve empirical performance. A report on the experimental evaluation is given in Section 4. We conclude in Section 5.

2 Preliminaries

Notation: A Boolean variable can take values *true* (1) or *false* (0). A literal is a Boolean variable x or its negation $\neg x$. A clause is a disjunction (set of) of literals. The set $\text{VAR}(\mathcal{F})$ and $\text{LIT}(\mathcal{F})$ contain all variables (resp. literals) of the formula $\mathcal{F}$. The binary variable $\langle x \sqcap k \rangle$ is an indicator for the condition $x \sqcap k$ being true, e.g., $\langle x > k \rangle$ is true if the integer variable x is assigned a value greater than k. For two formulas R and L, $R \Rightarrow L$ denotes logical entailment, i.e., all models R are also models of L. We use the notation $unsat(L)$ to indicate that formula L is unsatisfiable (i.e. has no models). We will use $[\![x]\!]_l^u$ to denote the function returning the *excess* of x above l, up to u. That is, $[\![x]\!]_l^u = \max(0, \min(u, x) - l)$. For convenience, we write $[\![x]\!]_l^\infty$ as $[\![x]\!]_l$.

CP solvers typically (implicitly) reason about an integer variables x taking values in $[l..u]$ by using the *atomic constraints*: $x \geqslant d$, $x \leqslant d$, $x = d$, $x \neq d$ and *false* for $d = l..u$. Given a set of constraints $\mathcal{F}$, the current domain D, seen as a formula containing conjunction of atomic constraints, represents all possible values that each variable $x \in \text{VAR}(\mathcal{F})$ can take. We denote the upper and lower bound (in the current domain) of an integer x by $\mathsf{ub}(x)$ and $\mathsf{lb}(x)$, respectively. A propagator f_c for a constraint $c \in \mathcal{F}$ takes the current domain D and returns

a set $f_c(D)$ of atomic constraints such that each $r \in f_c(D)$ is entailed by $D \wedge c$ but not D alone, i.e. $D \not\Rightarrow r$ and $D \wedge c \Rightarrow r$. If $\mathit{false} \in f_c(D)$, the propagator has detected unsatisfiability, i.e. that the current domain is inconsistent with the constraint c. For example, if c is a clause, then $f_c(D) = \{\mathit{false}\}$ if the current domain sets all literals in c to 0. If instead, D sets all but one literal $l \in c$ to false, then $f_c(D) = \{l = 1\}$.

We consider the problem of minimising a linear objective function $\sum_i w_i \cdot x_i$, subject to a set of constraints $\mathcal{F}$. Here each x_i is an integer variable taking values in some domain $[l...u]$ assigned a weight w_i. Whenever all variables are binary and all constraints in $\mathcal{F}$ are clauses, we talk about a (weighted partial) MaxSAT problem. We say that a literal $x \in \text{LIT}(\mathcal{F})$ for which $w(x) > 0$ is *soft* and denote the set of all soft variables by $\text{S}(\mathcal{F})$. A model τ of $\mathcal{F}$ is a solution and has cost $\text{COST}(\mathcal{F}, \tau) = \sum_{x_i \in \text{S}(\mathcal{F})} w(x_i)\tau(x_i)$. A solution τ is optimal if $\text{COST}(\mathcal{F}, \tau) \leqslant \text{COST}(\mathcal{F}, \tau')$ holds for all solutions τ' to $\mathcal{F}$. Note that the traditional description of MaxSAT is somewhat different to the above, but this is closer to the mathematical view of optimisation problems, and more closely reflects how MaxSAT solvers (including core-guided solvers) work internally [2].

An important concept in this work is that of an (unsatisfiable) *core*. Given a set $\mathcal{F}$ of constraints, a core κ is a set $\kappa \subset \text{LIT}(\mathcal{F})$ of literals s.t. $\mathcal{F} \wedge \bigwedge_{l \in \kappa}(l) \Rightarrow \mathit{false}$. In other words, cores are sets of literals for which no assignment satisfying $\mathcal{F}$ also satisfy all literals (seen as unit clauses) of κ. A key observation for core-guided search methods is that the existence of a core κ that only contains negations of soft variables implies a lower bound $w^\kappa = \min\{w_i \mid \neg x_i \in \kappa\}$ on the objective function $\sum_{i=1}^{n} w_i x_i$. Core-guided search methods make use of this fact by reformulating the instance during search. More specifically, given a lower bound l_i on each soft variable x_i, the objective function can be rewritten as $\sum_{i=1}^{n} w_i x_i = \sum_{i=1}^{n} w_i l_i + \sum_{i=1}^{n} w_i [\![x_i]\!]_{l_i} = C_{lb} + \sum_{i=1}^{n} w_i [\![x_i]\!]_{l_i}$ where C_{lb} is constant. We also note that, for an integer x, the following holds: $[\![x]\!]_k = \langle x > k \rangle + [\![x]\!]_{k+1}$, and $\langle x > k \rangle = [\![x]\!]_k^{k+1}$. For an integer x with initial domain $l..u$ then $x = [\![x]\!]_l^u$.

Lifting Explanations in CP solving: Similarly to how core-guided MaxSAT solvers make extensive use of conflict driven clause learning (CDCL) SAT solving under assumptions [3], core-guided CP solvers make extensive use of *lazy clause generation solving* (LCG) under assumptions [4]. Given a set F of propagators (representing a set of constraints), a current domain D_{orig} and a set A of assumptions (in the form of atomic constraints) over integer variables, an LCG solver $\mathsf{LCG}(F,D,A)$ determines if there exists an assignment θ to the variables that entails: (i) all constraints, (ii) all assumptions and (iii) the original domain D_{orig}. If so, the solver returns $\text{SAT}(\theta)$. Otherwise the solver returns $\text{UNSAT}(\bar{\kappa})$ where $\bar{\kappa} \subseteq A$ is a subset of variables that represents a core of the instance (more precisely, the set $\kappa = \{\neg l \mid l \in \bar{\kappa}\}$ is a core of the instance).

As we use LCG solvers in a black-box manner, we will not go into detail on how such solvers operate (see e.g. [5] for more details). A central concept for applying LCG solvers in core-guided CP is that of *explanations for propagations*.

Each propagator f_c in a LCG solver must be able to *explain* its propagation of an atomic constraint r in the form of a clause, i.e., compute an explanation clause $\mathsf{expl}(c, D, r) \equiv (E \rightarrow r)$ where E is a conjunction of atomic constraints and $D \Rightarrow E$, as well as $c \Rightarrow E \rightarrow r$. During conflict analysis, a learnt clause is derived by starting from the conflict C, and repeatedly replacing some atom $r \in C$ with its reason, i.e. E. Analogously to how learned clauses over assumptions represent cores in core guided MaxSAT, the explanations for failure over assumptions represent cores in core-guided CP. We note that for a single propagation there can be (and often are) are many different explanations; the solver does not need to use the current strongest information about the domain to explain the failure. Instead any explanation that is correct in the current domain suffices. The following examples demonstrate that some of the explanations are better for core guided CP than others.

Example 1. Consider the propagation of the linear inequality $2x + 3y + 4z \leqslant 27$ with the current domain $x \in [5..7]$, $y \in [4..9]$, $z \in [5..8]$. The propagator detects unsatisfiability, a simple explanation is: $\langle x \geqslant 5\rangle \wedge \langle y \geqslant 4\rangle \wedge \langle z \geqslant 5\rangle \rightarrow \mathit{false}$ i.e. the clause $C_1 = (\langle x < 5\rangle \vee \langle y < 4\rangle \vee \langle z < 5\rangle)$. This is not however, the only explanation. By relaxing bounds we obtain the *lifted* explanation $\langle x \geqslant -2\rangle \wedge \langle y \geqslant 4\rangle \wedge \langle z \geqslant 5\rangle \rightarrow \mathit{false}$ i.e. the clause $C_2 = (\langle x < -2\rangle \vee \langle y < 4\rangle \vee \langle z < 5\rangle)$, for the same propagation. Observe that the lifted explanation is stronger than the simple one as $C_2 \Rightarrow C_1$. Some lifted explanations can be particularly attractive, for example if the original domain sets $x \geqslant 0$, then the lifted explanation is equivalent to $\langle y \geqslant 4\rangle \wedge \langle z \geqslant 5\rangle \rightarrow \mathit{false}$, containing one less literal. □

One way of obtaining stronger explanations is through *lifting*, informally speaking a lifted explanation can be computed by making use of the original propagator in order to compute an explanation for an atom in the (partial) learned clause, instead of the explanation graph. The reason computed by the propagator will frequently be *weaker* than the atom that was originally inferred, and allow the construction of a more general explanation.

Example 2. Consider a propagator f_c for the linear inequality $c = 7y + 4t \geqslant 34$, and a current domain $y \in [0..10]$ and $t \in [0..2]$. The propagator can propagate $\langle y \geqslant 4\rangle$ with explanation $\langle t \leqslant 2\rangle \rightarrow \langle y \geqslant 4\rangle$. Now suppose a partial learned clause of form $\neg\langle y \geqslant 1\rangle \vee Q$, is encountered during conflict analysis. Since $\langle y \geqslant 4\rangle \Rightarrow \langle y \geqslant 1\rangle$ the original explanation can be used to obtain $C_1 = \neg\langle t \leqslant 2\rangle \vee Q$. However, the propagator f_c can return the lifted explanation $\langle t \leqslant 8\rangle \rightarrow \langle y \geqslant 1\rangle$ which allows deriving the learned clause $C_2 = \neg\langle t \leqslant 8\rangle \vee Q$. We observe that $C_2 \Rightarrow C_1$, i.e. the learned clause obtained with the lifted explanation is stronger than the original one. □

Objective probing: Branch-and-bound (B&B) CP solvers iteratively search for better solutions by constraining that the objective value must be better than in the previous found solution. A common issue that arises is slow convergence:

after finding a solution, B&B solvers typically generate many incrementally better solutions before reaching the true optimal solution. One strategy for improving convergence rate is *optimistic partitioning* [6]: after finding a solution with objective value $\hat{z}$ given lower bound z_{lb}, optimistic partitioning speculatively posts a constraint $\left\langle z < \frac{\hat{z}+z_{lb}}{2} \right\rangle$ (instead of $\langle z < \hat{z} \rangle$). If this succeeds, we have a much better solution; otherwise, the lower bound is greatly increased.

Core-Guided MaxSAT: that originated with the Fu-Malik algorithm [7] is today one of three central approaches to complete industrial MaxSAT solving, together with the implicit hitting set approach [8, 9] and the model improving (corresponding to branch-and-bound in CP) approach [10–13]. Core-Guided search is a lower bounding approach based on first assuming that all soft literals can be set to false and relaxing the assumption whenever new cores (i.e. sources of unsatisfiability) are detected.

In more detail, when minimising an objective $\sum_i w(x_i)x_i$ subject to a set $\mathcal{F}$ of clauses, modern core-guided MaxSAT solvers maintain a working instance $\mathcal{F}^w$, initialised to $\mathcal{F}$. During each iteration, a SAT solver is used to determine if there exists a solution of cost 0 to $\mathcal{F}^w$ by querying it on the clauses of $\mathcal{F}^w$ while assuming $A = \{\neg x | x \in \mathrm{S}(instance^w)\}$, that is all soft variables are false. If so, that solution will be an optimal solution to $\mathcal{F}$. Otherwise, the solver returns a set of literals $\bar{\kappa} \subset A$ that represents a core $\kappa = \{\neg l \mid l \in \kappa\}$ of $\mathcal{F}^w$. Next $\mathcal{F}^w$ is relaxed (reformulated) based on κ. First, the weight of each soft literal in κ is lowered by $w^\kappa = \min\{w(x) \mid x \in \kappa\}$. Second, new soft variables and clauses $\textsc{reform}(\kappa)$ that rule out κ as a source of unsatisfiability are added to $\mathcal{F}^w$.

Most core-guided solvers differ mainly in the instantiation of $\textsc{reform}(\kappa)$. We detail the OLL algorithm [14, 15] as it been shown to be the most effective in the MaxSAT evaluations and can be naturally extended to CP. Assuming $|\kappa| = n$ OLL adds new soft variables $\langle o_\kappa > 1 \rangle, ..., \langle o_\kappa > n-1 \rangle$, each of weight w^κ and clauses corresponding to AS-CNF$(\sum_{l \in \kappa}(l) > k \rightarrow \langle o_\kappa > k \rangle)$ for each $k = 1..n-1$. Assuming one of the commonly used encodings [16, 17], the clauses enforce that setting $k > 1$ literals of κ to true propagates the literals $\langle o_\kappa > 1 \rangle, ..., \langle o_\kappa > k-1 \rangle$ to true, incurring $(k-1)w^\kappa$ additional cost. Informally, the new clauses allow setting one of the soft literals in κ to true for free while incurring more cost for any additional ones.

Example 3. Consider the following problem:

$$\begin{aligned} \min z = {} & 3x_1 + 2x_2 + 2x_3 + 4x_4 \\ s.t. \quad & \max(x_1, x_2) \geqslant 2 \\ & \max(x_2, x_3) \geqslant 2 \\ & \max(x_3, x_4) \geqslant 2. \end{aligned}$$

where each x_i is an integer variable with domain 0..3. To solve this problem with the MaxSAT OLL algorithm, we consider the equivalent problem:

$$\begin{aligned}
\min z = \; & \sum_{k=1}^{3} 3\langle x_1 \geqslant k\rangle + \sum_{k=1}^{3} 2\langle x_2 \geqslant k\rangle + \sum_{k=1}^{3} 2\langle x_3 \geqslant k\rangle + \sum_{k=1}^{3} 4\langle x_4 \geqslant k\rangle \\
s.t \;\; & \text{AS-CNF}(\langle x_i \geqslant k\rangle) \text{ for } i = 1..4, k = 1..3 \text{ [18]} \\
& \langle x_1 \geqslant 2\rangle \vee \langle x_2 \geqslant 2\rangle \\
& \langle x_2 \geqslant 2\rangle \vee \langle x_3 \geqslant 2\rangle \\
& \langle x_3 \geqslant 2\rangle \vee \langle x_4 \geqslant 2\rangle .
\end{aligned}$$

We sketch an execution of the MaxSAT OLL algorithm. The initial solver call is made assuming $\langle x_i \geqslant k\rangle$ to false for all $i = 1..4$, and $k = 1..3$. Let $\kappa_1 = \{\langle x_2 \geqslant 2\rangle, \langle x_3 \geqslant 2\rangle\}$ be the first core extracted. First the weights of both variables in the core are lowered by $\min\{w(\langle x_2 \geqslant 2\rangle), w(\langle x_3 \geqslant 2\rangle)\} = 2$. Then a new soft variable $\langle o_1 > 1\rangle$ (with weight 2) defined with the clauses AS-CNF $\left(\neg\langle o_1 > 1\rangle \rightarrow \sum_{l\in\kappa_1}(l) \leqslant 1\right)$ is introduced to the instance before the solver reiterates. In the next iteration, the soft variables of the instance are $\langle o_1 > 1\rangle$, $\langle x_i \geqslant k\rangle$ for $k = 1..3$ and $i \in \{1, 4\}$ as well as $\langle x_j \geqslant 1\rangle$, $\langle x_j \geqslant 3\rangle$ for $j \in \{2, 3\}$. Let $\kappa_2 = \{\langle x_1 \geqslant 2\rangle, \langle o_1 > 1\rangle, \langle x_4 \geqslant 2\rangle\}$ be the next core. As in the first iteration, the weight of each soft variable in the core is lowered by 2 and new soft variables $\langle o_2 > 1\rangle$ and $\langle o_2 > 2\rangle$ (weight 2) are introduced and defined with constraints AS-CNF $\left(\neg\langle o_2 > k\rangle \rightarrow \sum_{l\in\kappa_2}(l) \leqslant k\right)$.

Next the algorithm extracts two unit cores arising due to the order-encoding of integers before terminating with the satisfying assignment that sets $\langle x_2 \geqslant 1\rangle$, $\langle x_3 \geqslant 1\rangle$, $\langle x_2 \geqslant 2\rangle$, $\langle x_3 \geqslant 2\rangle$ and $\langle o_1 > 1\rangle$ to true and all other variables to false. This assignment has cost 8 and corresponds to $x_1 = x_4 = 0$ and $x_2 = x_3 = 2$, an optimal assignment to the original problem. □

Core-boosting is a recently proposed [1] search strategy for MaxSAT that combines core-guided search with an anytime approach (originally a B&B type search for MaxSAT). The intuition underlying core-boosting is that core-guided search is mostly an "all-or-nothing" strategy. In its most basic form, core-guided search only finds one feasible solution during search, an optimal one. Furthermore, core-guided search tends to be somewhat bimodal [1] , either proving optimality fairly quickly or not terminating within a reasonable time. Core-boosted search is designed to take advantage of the fact that, even if core-guided search doesn't terminate within a reasonable time, it might still rule out a significant number of cores from the instance that would cause trouble for approaches like B&B.

More specifically, given a total resource budget, core-boosting spends a small fraction of its budget running in a core-guided mode. If this budget is exhausted and optimality has not yet been proven it rebuilds the objective based on the cores found so far, and then spends its remaining time optimizing the *reformulated* objective in a branch-and-bound mode.

3 Advancing Core-Guided search for CP

In this section, we overview core-guided search for CP and discuss our contributions toward advancing its performance. We begin with what we call *slice-based* reformulation, i.e. the conventional translation of the OLL algorithm for MaxSAT to CP. We also discuss potential issues when applying explanation lifting to the slice based formulation. Motivated by these we then detail our main contributions: *coefficient elimination* and *variable-based reformulation*, two novel core-guided reformulations specific for CP. Finally, we also discuss improvements and generalisations of existing search heuristics from MaxSAT and CP; assumption probing and core-boosting.

3.1 Slice-based reformulation

The following restatement of Example 3 for CP provides intuition for the slice based reformulation.

Example 4. In contrast to MaxSAT, the OLL algorithm with slice based reformulation works directly on the original problem of Example 3. Initially, the LGC solver is called while assuming all variables to their current lower bounds, i.e. $\langle x_i \leqslant 0\rangle$ for $i = 1..4$. Let $\kappa_1 = \{\neg\langle x_2 \leqslant 0\rangle, \neg\langle x_3 \leqslant 0\rangle\}$, i.e. $\{\langle x_2 \geqslant 1\rangle, \langle x_3 \geqslant 1\rangle\}$ be the first core obtained. The algorithm now introduces a new *integer* variable $o_1 \geqslant \langle x_2 \geqslant 1\rangle + \langle x_3 \geqslant 1\rangle$ with an initial domain $o_1 = \{1, 2\}$ (notice that $o_1 \neq 0$ as at least one of the literals in the core has to be false). For an alternative view, o_1 could be seen as the variable $o_1 = \langle x_2 \geqslant 1\rangle + \langle x_3 \geqslant 1\rangle$ that has its upper bounds enforced by assumptions. Next the objective is reformulated using o_1 to $z = 2 + 2[\![o_1]\!]_1 + 3x_1 + 2[\![x_2]\!]_1 + 2[\![x_3]\!]_1 + 4x_4$. For some intuition, notice for example that the term $2[\![x_2]\!]_1$ corresponds to the term $2\sum_{k=2}^{3}\langle x_2 \geqslant k\rangle$ in the MaxSAT objective and that the term $2[\![o_1]\!]_1 + 2$ can be seen as $2\langle x_2 \geqslant 1\rangle + 2\langle x_3 \geqslant 1\rangle$ plus the lower bound implied by κ_1.

In the next iteration, all variables are again assumed to their current lower bounds, i.e. $\langle x_1 \leqslant 0\rangle$, $\langle x_2 \leqslant 1\rangle$, $\langle x_3 \leqslant 1\rangle$, $\langle x_4 \leqslant 0\rangle$, and $\langle o_1 \leqslant 1\rangle$. Notice how the current lower bound for x_2 and x_3 is 1, conceptually, the (potential) weight for $x_2 = x_3 = 0$ is accounted for by the variable o_1. Assume the next core obtained is $\kappa_2 = \{\langle x_2 \geqslant 2\rangle, \langle x_3 \geqslant 2\rangle\}$. Similarly to before, a new variable o_2 and constraint $o_2 \geqslant \langle x_2 \geqslant 2\rangle + \langle x_3 \geqslant 2\rangle$ is introduced, and the objective reformulated to $z = 4 + 2[\![o_1]\!]_1 + 2[\![o_2]\!]_1 + 3x_1 + 2[\![x_2]\!]_2 + 2[\![x_3]\!]_2 + 4x_4$.

In the next iteration, the lower bounds for the variables are $\langle x_1 \leqslant 0\rangle$, $\langle x_2 \leqslant 2\rangle$, $\langle x_3 \leqslant 2\rangle$, $\langle x_4 \leqslant 0\rangle$, $\langle o_1 \leqslant 1\rangle$, and $\langle o_2 \leqslant 1\rangle$. Let $\kappa_3 = \{\langle x_1 \geqslant 1\rangle, \langle o_2 \geqslant 2\rangle, \langle x_4 \geqslant 1\rangle\}$ be the next core extracted, after which the constraint $o_3 \geqslant \langle x_1 \geqslant 1\rangle + \langle o_2 \geqslant 2\rangle + \langle x_4 \geqslant 1\rangle$ is introduced and the objective reformulated to $z = 6 + 2[\![o_1]\!]_1 + [\![x_1]\!]_0^1 + 3[\![x_1]\!]_1 + 2[\![x_2]\!]_2 + 2[\![x_3]\!]_2 + 2[\![x_4]\!]_0^1 + 4[\![x_4]\!]_1$.

After this, the solver still extracts the core $\kappa_4 = \{\langle o_1 \geqslant 2\rangle\}$ and reformulates the instance one last time before terminating with the solution $x_1 = x_4 = 0$, $x_2 = x_3 = 2$ and $o_1 = o_2 = 2$, $o_3 = o_4 = 1$. □

Algorithm 1: OLL for constraint programming using slice-based reformulation

Data: Constraints $\mathcal{F}$, an original domain D_o and objective $z = w_1x_1 + \ldots + w_mx_m$.
Result: Optimal solution θ^*.

```
z_lb ← Σ_{i=1}^{n} w_i lb(x_i)
switch LCG(F,D_o,∅) do
    case UNSAT(∅) do
        return UNSAT
    case SAT(θ) do
        E ← {x_i ↦ (lb(x_i), w_i, w_i) | i ∈ 1...n}
        while true do
            switch LCG(F,D_o,{⟨x_i ⩽ l_i⟩ | E[x_i] = (l_i, u_i, w_i)}) do
                case SAT(D) do
                    return D, z_lb
                case UNSAT(κ̄) do
                    F, z_lb, E ← REFORMULATE-SLICE(F, z_lb, E, κ)
```

Algorithm 2: REFORMULATE-SLICE$(\mathcal{F}, z_{lb}, E, \kappa)$

```
w^κ ← min{u_i | ⟨x_i < k_i⟩ ∈ κ, E[x_i] = (lb_i, u_i, w_i)}
o_κ ← NEW-VAR(F, [1, |κ|])
R ← 0
for ⟨x_i < k_i⟩ ∈ κ do
    (lb_i, u_i, w_i) ← E[x_i]
    R ← R + ⟨x_i > lb_i⟩
    if u_i = w^κ then
        E[x_i] ← (lb_i + 1, w_i, w_i)
    else
        E[x_i] ← (lb_i, u_i − w^κ, w_i)
E[o_κ] ← (1, w^κ, w^κ)
F ← F ∧ (o_κ ⩾ R)
return F, z_lb + w^κ, E
```

Example 4 gives some intuition for the term *slice based* reformulation. In each iteration the algorithm *slices off* the current lower bound of all variables appearing in a core κ, packaging the removed values into a new penalty term $o_\kappa = \sum_{l \in \kappa}(l)$.

Algorithms 1 and 2 detail OLL with slice based reformulation for CP. Given a set $\mathcal{F}$ of constraints and an objective z to minimize, the algorithm initially checks the feasibility of the problem by calling the LCG solver on the constraints without assumptions. If the problem has feasible solutions, the algorithm enters its main search loop. On each iteration, the LCG solver is invoked on the instance

while assuming all soft variables to their current lower bounds. These lower bounds are maintained in a mapping E that maps each variable x_i to a tuple (l_i, u_i, w_i) containing its current lower bound (l_i), its residual weight u_i and its full weight w_i. If the solver returns SAT(D) the obtained domain will be an optimal solution to the problem so the algorithm terminates. Otherwise, the solver returns a set κ of assumptions corresponding to the core $\{\neg l \mid l \in \kappa\}$. The algorithm then reformulates the instance using Algorithm 2. Analoguously to MaxSAT, slice based reformulation of the instance means: (i) computing w^κ, the minimum residual weight of all literals in the core, (ii) lowering the (residual) weight of each literal in the core by w^κ and (iii) introducing a new variable o_κ with lower bound 1 and full weight w^κ as well as new constraints $o_\kappa \geqslant \sum_{l \in \kappa}(l)$. Any variable whose residual weight gets lowered to 0 during step (ii) gets its lowerbound increased by one and residual weight reset to its full weight

The following example demonstrates a potential weakness of slice-based reformulation, motivating the novel reformulation strategies we propose in the next section.

Example 5. Consider the problem defined in Example 3 and the initial LCG call made by OLL for CP with the assumptions $\langle x_i \leqslant 0 \rangle$ for $i = 1..4$. Assume now that we obtain the lifted core $\kappa'_1 = \{\langle x_2 \geqslant 2 \rangle, \langle x_3 \geqslant 2 \rangle\}$ and introduce a single new variable $o_\kappa = \langle x_2 \geqslant 1 \rangle + \langle x_2 \geqslant 2 \rangle + \langle x_3 \geqslant 1 \rangle + \langle x_3 \geqslant 2 \rangle = [\![x_2]\!]_0^2 + [\![x_3]\!]_0^2$ If the solver later derives $\langle x_2 \geqslant 1 \rangle$ and $\langle x_3 \geqslant 1 \rangle$, the lower bound on o_κ is set to 2. However, with the reformulations performed in Example 4 the algorithm has already derived $\langle x_2 \geqslant 2 \rangle \vee \langle x_3 \geqslant 2 \rangle$, implying a lower bound of 3 on o_κ □

In other words, slice based reformulation makes using lifted cores difficult. Hence, instead of the approach presented in Example 5 we instead perform reformulation similarly to Example 4 instead. We do however add the lifted core to the model, thus allowing the algorithm to extract it later without search.

3.2 Novel Core-Guided Reformulations for CP

Next we detail the main contribution of this work, two novel reformulation strategies for the CP OLL algorithm: 1) *Coefficient Elimination* and 2) *Variable-based reformulation.* Coefficient elimination seeks to increase the number of variables whose lower bounds are increased during reformulation steps., thus increasing the rate at which the lower bounds of the variables increase. Variable-based reformulation attempts to make better use of the information provided by lifted cores in order to increase the lower bound on the objective faster.

Coefficient Elimination Let κ be a set of literals corresponding to a core obtained during an iteration of OLL for CP and w^κ the smallest (residual) weight of the literals in the core. Consider now the weighted sum of the literals in the core, i.e the variable $o_\kappa = \sum_{x_i \in \kappa} w_i x_i$. Since κ corresponds to a core, the lower bound of o_κ is w^κ and the objective could be reformulated using $\sum_{x_i \in \kappa} w_i [\![x_i]\!]_{l_i} = w^\kappa + [\![o_\kappa]\!]_{w^\kappa} + \sum_{x_i \in \kappa} w_i [\![x_i]\!]_{l_i+1}$. Notice how, in contrast to

the strategy described in Section 3.1 and Example 4, coefficient elimination in this form results in the lower bound of *all* variables in κ being increased by one. The drawback is instead the (potential) increase in complexity of the subsequent LCG solver calls.

Example 6. Consider the following problem:

$$\textbf{minimize } 1000p + \sum_{i=1}^{n} x_i + y_i \;\; \textbf{s.t.} \qquad \neg p \rightarrow x_i + y_i \geqslant 1, \;\; \forall i \in 1 \ldots n$$

With weight splitting, the OLL algorithm for CP generates $\max(n, 1000)$ cores of form $\langle p \leqslant 0 \rangle \wedge \langle x_i \leqslant 0 \rangle \wedge \langle y_i \leqslant 0 \rangle$ $i = 1..$, each time decreasing the coefficient of p by 1. Since p is never removed as a soft variable, we expect extracting each core to require approximately similar amounts of computational effort (see also independent core extraction detailed later in this section).

With coefficient elimination, the algorithm instead introduces the variable $o_\kappa = 1000p + x_1 + y_1$ with a lower bound of 1 when reformulating $\kappa = \langle p \leqslant 0 \rangle \wedge \langle x_1 \leqslant 0 \rangle \wedge \langle y_1 \leqslant 0 \rangle$. In the next iteration, the variable p is not soft anymore. Instead the next core extracted will be $\langle o_\kappa \leqslant 1 \rangle \wedge \langle x_2 \leqslant 0 \rangle \wedge \langle y_2 \leqslant 0 \rangle$ instead. Informally speaking, all of the subsequent cores will depend on the reformulation variables introduced in previous iterations thus making extracting cores require increasing amounts of computational effort. [19] □

The version of coefficient elimination that we consider is a hybrid strategy designed to balance the number of variables whose lower bounds can be increased during each reformulation with the potential of extracting independent cores during subsequent iterations. More specifically, when reformulating on a set κ of literals having minimum weight w^κ, we fully reformulate all literals in κ that have weight less than Tw^κ where T is a threshold parameter, and slice the rest. More formally, coefficient elimination introduces a variable $o_\kappa = \sum_{x_i \in \kappa} c_i x_i$ where $c_i = \max(w_i, Tw^\kappa)$ and reformulates the objective using $\sum_{x_i \in \kappa} w_i [\![x_i]\!]_{l_i} = w^\kappa + [\![o_\kappa]\!]_{w^\kappa} + \sum_{x_i \in \kappa} \max(0, w_i - Tw^\kappa)[\![x_i]\!]_{l_i} + \sum_{x_i \in \kappa} w_i [\![x_i]\!]_{l_i+1}$.

Variable-based reformulation attempts to overcome the difficulties that slice based reformulation has with exploiting the full potential of lifted cores, i.e. that slice-based reformulation can only ever increase the lb of variables by 1 and thus the objective by the minimum (residual) weight of the variables in the core.

Recall for example the lifted core $\{\langle x_2 \geqslant 2 \rangle, \langle x_3 \geqslant 2 \rangle\}$ discussed in Example 5. When reformulating with variable-based reformulation, the OLL algorithm for CP introduces the variable $o_\kappa \geqslant x_2 + x_3$ with an initial domain of $[2 \ldots)$. In more general terms, variable based reformulation merges all variables in a core into a single new variable, and assigns the new variable an initial lower bound equal to the lowest bound appearing in the core. Notice that the potential benefits of variable based elimination are directly related to the size of the domains of the involved variables. In this particular case, the approach lifts the lower bound by 2 but it is easy to create examples where the increase is higher, which we observed to also occur frequently in practice.

In addition to more effectively exploiting information in lifted cores, we often observed that variable based reformulation resulted in unit cores being extracted in subsequent iterations. Unit cores are particularly attractive for the OLL algorithm for CP as no new variables nor constraints need to be introduced. Instead it suffices to increase the lower bound of the variable in the core.

3.3 Generalisations of Existing Techniques

Before reporting on an experimental evaluation of the new reformulation strategies, we briefly describe new generalisations and improvements to existing heuristics in both CP and MaxSAT solving that we make use of in this work.

Core-Boosting for CP We extended core-boosted search from MaxSAT to CP. A key difference when applying core boosting in CP compared to MaxSAT is the need to explicitly encode the objective function (which is only implicitly defined during the core-guided phase) before switching to branch-and-bound search.

Explicitly encoding the objective function when using variable based reformulation is fairly straightforward. During each reformulation step, a set $x_1, \ldots, x_k$ of variables in the objective are replaced with a variable o representing their sum. When switching to B&B search, the same procedure is used to remove all remaining terms and merging them into a single new variable. In contrast, combining core-boosted search with slice-based reformulation is more intricate. Consider a possible (implicit) objective:

$$\begin{aligned} z = \quad & c_1 [\![x_1]\!]_{d_1} \quad + c_2 [\![x_2]\!]_{d_2} + \ldots \quad + c_k [\![x_m]\!]_{d_m} \\ & + b_1 \langle x_1 \geqslant d_1 \rangle + b_2 \langle x_2 \geqslant d_2 \rangle + \ldots + b_k \langle x_m \geqslant d_m \rangle + z_{lb} \end{aligned}$$

obtained after several iterations of core-guided search with sliced based reformulation. A simple approach to making z explicit is to introduce fresh variables for each sub-term, i.e. let $x'_i = \mathbf{max}(0, x_i - d_i)$, $x''_i = \langle x_i \geqslant d_i \rangle$ and $z = z_{lb} + \sum_{i=1}^{m} c_i x'_i + \sum_{i=1}^{m} b_i x''_i$.

A more efficient method makes use of the monotonicity of $[\![x_i]\!]_{d_i}$ which in turn implies that any atomic constraint $\left\langle [\![x_i]\!]_{d_i} \geqslant c \right\rangle$ can be expressed as an equivalent atom $\langle x_i \geqslant c' \rangle$:

$$\left\langle [\![x_i]\!]_{l_i}^{u_i} \geqslant c \right\rangle = \begin{cases} \mathit{true} & \textbf{if } c \leqslant 0 \\ \langle x_i \geqslant l_i + c \rangle & \textbf{if } 0 < c \leqslant u_i - l_i \\ \mathit{false} & \textbf{if } c > u_i - l_i \end{cases}$$

Hence we can use a form of variable view [20] to encode the expressions $[\![x_i]\!]_{d_i}$ and $\langle x_i \geqslant d_i \rangle \equiv [\![x_i]\!]_{d_i}^{d_i+1}$, thus avoiding the need to introduce new variables.

Progressive probing Recall that when given an incumbent solution with cost $\hat{z}$ and a lower bound z_{lb} on the objective, objective (optimistic) probing attempts to improve the solution and find a solution of cost $(\hat{z} + z_{lb})/2$. In practice, we observed that objective probing is a risky strategy since the jump from

$\hat{z}$ to $(\hat{z} + z_{lb})/2$ is quite aggressive, and thus can result in difficult unsatisfiable subproblems. Instead we consider a more conservative strategy that we call *progressive probing*, an idea resembling to the use of progression in MaxSAT solving [21] For geometrically increasing values of δ (more precisely, in iteration i $\delta_i = 2^i stepsize$ where $stepsize$ is a parameter) the solver is queried for a solution of cost $\hat{z} - \delta_i$. The procedure reiterates until the solver either returns UNSAT or runs out of the resources allocated for probing.

In addition to B&B search, we also make use of probing during core-guided search, inspired by techniques from the MaxSAT community [22, 23]. Anytime a singleton core $\langle x > k \rangle$ is extracted , the bound k is probed by repeated invocations of $\mathsf{LCG}(F,D,\{\langle x \leqslant k + \delta_i \rangle\})$ for the geometrically increasing values of δ_i defined above. Core-probing like this is particularly effective in combination with variable-based reformulation: if $\mathsf{lb}(x) + \mathsf{lb}(y)$ is much smaller than the true lower bound of $o_\kappa = x + y$, core probing will quickly push up the bound of z, skipping many of the intermediate steps. Recall also that variable based reformulation often results in unit cores being extracted in subsequent iterations.

3.4 Additional Techniques

Finally, we also considered a number of fairly direct translations of MaxSAT techniques to CP.

Independent core extraction [24] is a strategy for obtaining *simpler* cores. Given some core κ of instance, the reformulation (i.e. introduction of new variables and constraints) is delayed and instead only the assumptions in the core are relaxed (lower their weight by w^κ or Tw^κ in the case of variable based reformulations). Since at least one of the weights will be lowered to zero, the solver can be invoked to extract another core without needing to reformulate. Note that reformulating the instance makes it more complicated, and thus delaying is beneficial. The process continues until no more cores can be found, at which point all found cores are reformulated.

Stratification [25] starts by posting assumptions using only literals with high weights, and throughout the search introduces the remaining literals. This allows high-weighted core to be extracted early in the search. As a side effect, feasible solutions can be generated in the process.

Hardening [25] can be used to enforce satisfaction for certain literals. Given an upper and lower bound z_{ub} z_{lb} on the optimal cost, *hardening* will set false any Booleans with weight $w > z_{ub} - z_{lb}$. The same rule can be generalised to integer variables x_i by setting $\mathsf{ub}(x_i) = lb_i + \lfloor \frac{z_{ub} - z_{lb}}{c_i} \rfloor$ where c_i is the coefficient and lb_i a (relaxed) lower bound of x_i.

Solution-guided search [13, 26] is a value-selection heuristic that assigns a branching-variable the value it takes in the current best solution if possible, and otherwise resorts to the default value-selection strategy. This focuses the search around the best solution, quickly finding local improvements.

4 Experiments

In order to experimentally evaluate the improvements to core-guided search we integrated the described core-guided optimisation methods into `geas` (`https://bitbucket.org/gkgange/geas`), a lazy-clause generation solver. The core engine of `geas` is written in C++, with a `FlatZinc` frontend written in OCaml. The core-guided optimisation techniques are entirely implemented in the OCaml frontend, using the engine's assumption interface to handle the cores and reformulation. Propagators in `geas` implement lazy explanation with lifting, so it can extract lifted cores.

As benchmarks, we took the set of models and instances from the MiniZinc Challenge [27] for years 2015–2018, and selected all optimization models with a linear objective. This resulted in 48 models, and 249 instances. We then ran `geas` on this data-set, comparing its branch-and-bound configuration (bb), with all combinations of the following core-based configurations:

- core-guided (core), or core-boosted (boost), using a 10% of time limit (i.e. 60s) core-guided phase before switching to branch and bound search
- slice-based (slice), or variable-based (var) reformulations.
- weight splitting (split) or coefficient elimination with a threshold of $T = 2$ (elim).

All core-based methods were run with stratification, independent core extraction and hardening. Each instance was run with a 600 second time-limit, reporting the time to prove optimality as well as the best objective value found.

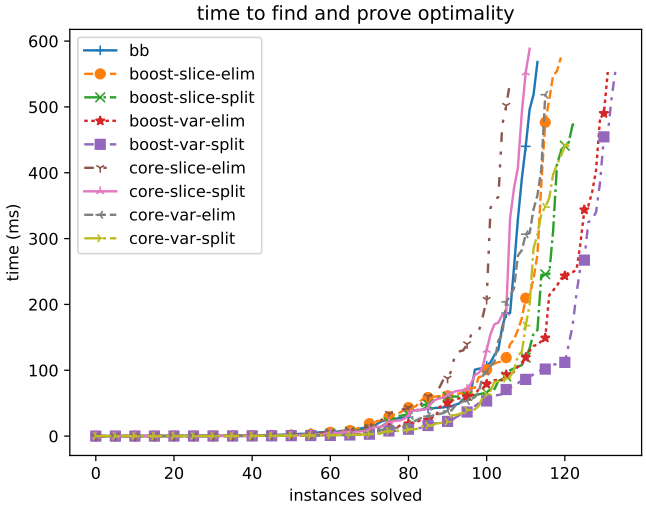

Fig. 1: Time to prove optimality for all methods.

Figure 1 compares the overall performance of each set of parameters across the dataset. We observe that branch and bound performs slightly better than "the basic version" of core guided search, i.e. core-slice-split and core-slice-elim. Variable based reformulation improves over slice based reformulation, obtaining performance superior to B&B. The best overall performance is obtained by boost-var-split making use of core-boosted search, variable based reformulation and weight splitting, although the difference between coefficient elimination and weight splitting is minor.

Figure 2 gives a per-instance breakdown of the results, comparing core-guided search with branch-and-bound as well as the reformulation strategies. We observe

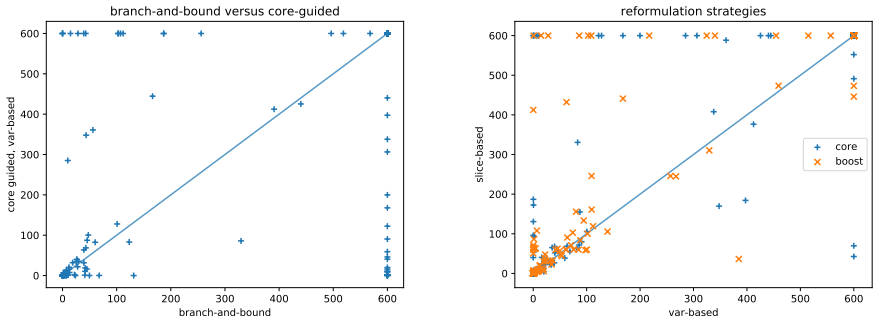

Fig. 2: Comparing time-to-optimality. Left: branch-and-bound versus core guided using a variable-based reformulation. Right: slice-based versus variable-based reformulation.

that B&B and core-guide search are fairly orthogonal in the sense that there are many instances on which B&B search finished quickly while core-guided search times out and vice versa. This observation provides a possible explanation for the good overall performance of core-boosted search, notice that most of the instances where core-guided outperforms branch-and-bound are clustered in the bottom-right of the figure. The other side of the figure also clearly demonstrates the superior performance of variable based reformulation compared to slice based reformulation.

In addition to proving optimality we investigate the anytime behaviour of the methods i.e. how good are the solutions obtained when optimality is not proven? Tables 1 and 2 compare the quality of solutions found by each method across all (Table 1) and a representative (Table 2) set of benchmarks. The tables again demonstrates the orthogonality of the methods we consider, no individual method dominates all others. However, pure core-guided methods were much less competitive as anytime methods supporting the intuition of core-guided methods typically either proving optimality quickly, or failing to produce solutions of reasonable quality.

Table 1: How many times did each each method (row) report a *strictly better* objective value than each other method (column) and the best objective value found overall (column BEST).

				BOOST				CORE				
				VAR		SLICE		VAR		SLICE		
			BB	ELIM	SPLIT	ELIM	SPLIT	ELIM	SPLIT	ELIM	SPLIT	BEST
		BB	0	25	22	33	38	100	93	89	87	196
BOOST	VAR	ELIM	39	0	27	36	36	102	95	93	91	211
		SPLIT	39	23	0	33	36	103	93	94	91	205
	SLICE	ELIM	37	18	20	0	25	101	92	91	88	196
		SPLIT	33	22	20	28	0	101	93	92	87	197
CORE	VAR	ELIM	20	4	4	10	8	0	22	39	34	144
		SPLIT	19	4	5	9	8	24	0	37	31	152
	SLICE	ELIM	22	8	8	8	8	50	47	0	13	151
		SPLIT	22	7	8	9	8	57	54	25	0	154

Table 2: Quality scores for selected models. Quality of a solution with objective value z is defined as the ratio of the distance between the initial lower bound z_{lb} and z to the distance between the best solution obtained by any method and z_{lb}

		BOOST				CORE			
		VAR		SLICE		VAR		SLICE	
MODEL	BB	ELIM	SPLIT	ELIM	SPLIT	ELIM	SPLIT	ELIM	SPLIT
cargo_coarsePiles	0.99	**1.00**	0.98	0.90	0.89	0.81	0.81	0.79	0.79
celar	0.59	0.97	0.98	**0.99**	**0.99**	0.68	0.68	0.94	0.91
oc-roster	0.93	**1.00**	0.97	**1.00**	0.96	0.66	0.51	0.55	0.52
seat-moving	0.84	0.95	0.95	0.95	0.95	**1.00**	**1.00**	**1.00**	**1.00**
vrplc_service	**1.00**	0.99	0.99	0.99	**1.00**	0.80	0.80	0.74	0.74

5 Conclusion

In this paper, we revisit the use of unsatisfiable core approaches for CP – both standalone, and as part of a hybrid (core-boosted) strategy. We exploit the extra expressiveness of lazy clause generation solvers to build more compact OLL-style reformulations, and to opportunistically tighten lower and upper bounds. We experimentally evaluated the new methods and draw the following conclusions 1) Core-boosting is generally worthwhile, both for anytime performance *and* proving optimality. 2) Variable-based reformulations are typically better for proving optimality, but this is model-dependent. 3) If using core-boosting, variable-based reformulations also produce better solutions. 4) Surprisingly, slice-based reformulations yield better solutions for core-guided; but still not as good as those for core-boosted. 5) Coefficient elimination finds the best solution slightly more frequently in combination with variable-based core-boosting (but is slightly worse at proving optimality). In other configurations, it is worse than weight splitting.

References

1. Berg, J., Demirović, E., Stuckey, P.J.: Core-boosted linear search for incomplete MaxSAT solving. In Rousseau, L.M., Stergiou, K., eds.: Proceedings of Sixteenth International Conference on Integration of Artificial Intelligence and Operations Research techniques in Constraint Programming (CPAIOR2019). Number 11494 in LNCS, Springer (2019) 39–56
2. Berg, J., Järvisalo, M.: Unifying reasoning and core-guided search for maximum satisfiability. In: JELIA. Volume 11468 of Lecture Notes in Computer Science., Springer (2019) 287–303
3. Eén, N., Sörensson, N.: Temporal induction by incremental SAT solving. Electr. Notes Theor. Comput. Sci. **89**(4) (2003) 543–560
4. Ohrimenko, O., Stuckey, P., Codish, M.: Propagation via lazy clause generation. Constraints **14**(3) (2009) 357–391
5. Feydy, T., Stuckey, P.J.: Lazy clause generation reengineered. In Gent, I.P., ed.: Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009, Lisbon, Portugal, September 20-24, 2009, Proceedings. Volume 5732 of Lecture Notes in Computer Science., Springer (2009) 352–366
6. Marriott, K., Stuckey, P.: Programming with Constraints: an Introduction. MIT Press (1998)
7. Fu, Z., Malik, S.: On solving the partial MaxSAT problem. In: Proc. SAT. Volume 4121 of Lecture Notes in Computer Science., Springer (2006) 252–265
8. Davies, J., Bacchus, F.: Exploiting the power of MIP solvers in MaxSAT. In: Proc. SAT. Volume 7962 of Lecture Notes in Computer Science., Springer (2013) 166–181
9. Saikko, P., Berg, J., Järvisalo, M.: LMHS: A SAT-IP hybrid MaxSAT solver. In: Proc. SAT. Volume 9710 of LNCS., Springer (2016) 539–546
10. Martins, R., Manquinho, V.M., Lynce, I.: Improving linear search algorithms with model-based approaches for MaxSAT solving. J. Exp. Theor. Artif. Intell. **27**(5) (2015) 673–701
11. Morgado, A., Heras, F., Marques-Silva, J.: Model-guided approaches for MaxSAT solving. In: Proc ICTAI, IEEE Computer Society (2013) 931–938
12. Koshimura, M., Zhang, T., Fujita, H., Hasegawa, R.: Qmaxsat: A partial max-sat solver. Journal on Satisfiability, Boolean Modeling and Computation **8** (2012) 95–100
13. Demirović, E., Stuckey, P.J.: Techniques inspired by local search for incomplete maxsat and the linear algorithm: Varying resolution and solution-guided search. In: CP. Volume 11802 of Lecture Notes in Computer Science., Springer (2019) 177–194
14. Andres, B., Kaufmann, B., Matheis, O., Schaub, T.: Unsatisfiability-based optimization in clasp. In: Proc. ICLP Technical Communications. Volume 17 of LIPIcs., Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2012) 211–221
15. Morgado, A., Dodaro, C., Marques-Silva, J.: Core-guided MaxSAT with soft cardinality constraints. In: Proc. CP. Volume 8656 of Lecture Notes in Computer Science., Springer (2014) 564–573
16. Asín, R., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: Cardinality networks and their applications. In: Proc SAT. (2009) 167–180
17. Bailleux, O., Boufkhad, Y.: Efficient CNF encoding of boolean cardinality constraints. In: Proceedings of CP-03. (2003) 108–122

18. Tamura, N., Taga, A., Kitagawa, S., Banbara, M.: Compiling finite linear CSP into SAT. Constraints **14**(2) (2009) 254–272
19. Bacchus, F., Narodytska, N.: Cores in core based MaxSat algorithms: An analysis. In: Proc. SAT. Volume 8561 of Lecture Notes in Computer Science., Springer (2014) 7–15
20. Schulte, C., Tack, G.: View-based propagator derivation. Constraints **18**(1) (2013) 75–107
21. Ignatiev, A., Morgado, A., Manquinho, V.M., Lynce, I., Marques-Silva, J.: Progression in maximum satisfiability. In: ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014). (2014) 453–458
22. Ansótegui, C., Gabàs, J.: Wpm3: An (in)complete algorithm for weighted partial maxsat. Artificial Intelligence **250** (2017) 37 – 57
23. Ignatiev, A., Morgado, A., Marques-Silva, J.: Rc2: a python-based maxsat solver. MaxSAT Evaluation 2018 22
24. Berg, J., Järvisalo, M.: Weight-aware core extraction in SAT-based MaxSAT solving. In: Proc CP. Volume 10416 of Lecture Notes in Computer Science., Springer (2017) 652–670
25. Ansótegui, C., Bonet, M.L., Gabàs, J., Levy, J.: Improving SAT-based weighted MaxSAT solvers. In: Proc. CP. Volume 7514 of Lecture Notes in Computer Science., Springer (2012) 86–101
26. Demirović, E., Stuckey, P.J.: Local-style search in the linear maxsat algorithm: A computational study of solution-based phase saving. In: Pragmatics if SAT Workshop. (2018)
27. Stuckey, P.J., Feydy, T., Schutt, A., Tack, G., Fischer, J.: The minizinc challenge 2008-2013. AI Magazine **35**(2) (2014) 55–60

Appendix: Solution quality across all models

model	bb	b-v-el	b-v-sp	b-s-el	b-s-sp	c-v-el	c-v-sp	c-s-el	c-s-sp
cargo_coarsePiles	0.99	1.00	0.98	0.90	0.89	0.81	0.81	0.79	0.79
celar	0.59	0.97	0.98	0.99	0.99	0.68	0.68	0.94	0.91
city-position	0.94	0.92	0.95	0.84	0.83	0.01	0.01	0.97	0.97
community-detection	0.83	0.87	0.71	0.61	0.69	0.41	0.34	0.26	0.33
concert-hall-cap	1.00	1.00	1.00	1.00	1.00	0.88	0.98	0.94	0.98
crossword_opt	0.99	0.98	0.99	0.98	0.98	0.90	0.91	0.96	0.96
cvrp	0.96	0.99	0.98	0.97	1.00	0.78	0.78	0.78	0.78
dcmst	1.00	1.00	1.00	1.00	1.00	0.92	0.89	0.91	0.94
freepizza	0.99	0.97	0.98	0.98	0.97	0.83	0.83	0.83	0.83
gbac	0.76	0.92	0.78	0.85	0.92	0.36	0.28	0.45	0.45
gfd-schedule	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
gfd-schedule2	1.00	1.00	0.99	1.00	1.00	0.98	0.98	0.99	0.99
group	0.96	0.99	0.97	0.97	1.00	0.92	0.92	1.00	0.99
handball	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
hrc	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
jp-encoding	0.87	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
mapf	1.00	1.00	1.00	1.00	1.00	0.89	0.90	0.99	0.95
mario	0.96	0.91	0.97	1.00	0.97	0.82	0.82	0.82	0.85
maximum-dag	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
mknapsack_global	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
model	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.98
mznc2017_aes_opt	0.98	0.98	0.98	0.99	1.00	0.86	0.86	0.91	0.91
mznc2017_cargo	0.95	0.86	0.94	0.77	0.90	0.68	0.85	0.72	0.72
neighbours-rect	1.00	1.00	1.00	1.00	0.99	0.98	0.97	0.98	0.99
nfc	0.97	1.00	0.99	1.00	0.98	0.96	0.96	0.98	0.98
oc-roster	0.93	1.00	0.97	1.00	0.96	0.66	0.51	0.55	0.52
p1f	0.96	0.96	0.96	0.96	0.96	1.00	1.00	1.00	1.00
pc	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
rcpsp-wet	1.00	1.00	1.00	1.00	1.00	0.98	1.00	1.00	1.00
rel2onto	1.00	1.00	1.00	1.00	1.00	0.95	0.92	0.92	0.90
road_naive	0.40	0.66	0.66	0.46	0.46	0.50	0.50	0.50	0.50
roster_model	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
seat-moving	0.84	0.95	0.95	0.95	0.95	1.00	1.00	1.00	1.00
spot5	0.89	0.95	1.00	0.98	0.99	0.94	0.93	0.96	0.96
steelmillslab	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
steiner-tree	1.00	1.00	1.00	1.00	1.00	0.94	0.96	0.91	0.99
tcgc2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
tpp	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
train	1.00	1.00	1.00	1.00	1.00	0.98	0.96	0.98	0.98
triangular	1.00	1.00	0.99	0.99	0.99	0.97	0.97	0.97	0.97
ttppv	0.99	0.98	0.99	0.99	0.98	0.90	0.90	0.89	0.88
vrplc_service	1.00	0.99	0.99	0.99	1.00	0.80	0.80	0.74	0.74
wcsp	0.95	1.00	1.00	1.00	1.00	0.78	0.87	0.95	0.95
zephyrus	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00