An Optimal Iterative Algorithm for Extracting MUCs in a Black-box Constraint Network

Philippe Laborie[1]

Abstract. We present a non-intrusive iterative algorithm for extracting Minimal Unsatisfiable Cores in black-box constraint networks. The problem can be generalized as the one of finding a minimal subset satisfying an upward-closed property $\mathcal{P}$. If performance is measured as the number of infeasibility property checks, we show that the proposed algorithm, ADEL, is optimal both for small and for large MUCs and that it consistently outperforms existing approaches in between those two extremal cases.

1 Introduction

When a Constraint Satisfaction Problem is infeasible, providing a minimal explanation for infeasibility in the form of a minimal subset of constraints that are mutually contradictory (known as Minimal Unsatisfiable Core or MUC) helps identifying the causes of the infeasibility [2]. Many existing approaches for MUC extraction exploit the particular features of the problem or of the resolution engine. For instance techniques such as *elastic filters* can be used for LP models [1], *explanations* recording for CP models [5] or *no-goods* learning for SAT models [6]. In this paper, we tackle the problem of MUC extraction in a *non-intrusive* way with respect to the resolution engine by considering that checking for the (in)feasibility of a subset of constraints is a *black-box* operation. Advances of the state-of-the-art in Optimization result in increasingly sophisticated and efficient engines. Engines sophistication makes it harder to implement intrusive methods to compute MUCs whereas increase of engines efficiency makes infeasibility property checks faster. Both aspects tend to make the black-box approach attractive. The problem studied in this paper is thus more general than MUC extraction and can be defined as follows: given a finite set U and a property $\mathcal{P}$ on the subsets of U that is upward-closed (that is, whenever it holds for a subset X it also holds for any superset of X), find a *minimal subset* of U that satisfies property $\mathcal{P}$. For MUC extraction the property $\mathcal{P}(X)$ is the *infeasibility* of a subset of constraints X, it clearly is upward-closed as the superset of any infeasible subset of constraints is infeasible too. The works closest to ours are the `QuickXplain` recursive algorithm [4] developed in the context of MUC extraction for constraint programming using a black-box approach and the iterative dichotomy algorithm `DC` proposed in [3]. After introducing some notations and formally defining the problem, we describe the proposed algorithm `ADEL` (standing for the four ingredients of the approach: Acceleration, Dichotomy, Estimation and Lazy checks). Last section reports some performance results showing that the complexity of ADEL is optimal both for small and large minimal subsets and that it outperforms both the `QuickXplain` and the `DC` algorithms.

[1] IBM Software Group, France, email: laborie@fr.ibm.com

2 Problem Definition and Notations

Let U be a finite set of cardinality n and $\mathcal{P}$ an *upward-closed* property on its powerset 2^U that is, a property such that: $\big(X \subseteq Y \subseteq U\big) \wedge \mathcal{P}(X) \Rightarrow \mathcal{P}(Y)$.

Definition 1 (Minimal Subset) *A subset $X \subseteq U$ is said to be* **minimal** *if and only if: $\mathcal{P}(X) \wedge \forall Y \subset X : \neg\mathcal{P}(Y)$.*

In the sequel of this paper we assume $\mathcal{P}(U)$ that is, there exist at least one minimal subset. Without loss of generality we assume a total order over the elements in U so that the elements can be indexed: $U = (u_1, ..., u_n)$. For $1 \leq i$, $U_{i\rightarrow}$ denotes the subset $\{u_j | i \leq j \leq n\}$ of all u_j with index j greater than or equal to i (note that if $i > n$, then $U_{i\rightarrow} = \emptyset$).

The problem studied in this paper is the design of an efficient algorithm to compute a minimal subset X without any knowledge about property $\mathcal{P}$ beside the assumption that it is upward-closed. We estimate the complexity of an algorithm as the number of property checks it performs. Note that for comparing the different algorithms we use in this paper a more fine-grain measure than the traditional *big O* comparison because we count the number of property checks which is an homogeneous measure for all the approaches. That's why we use a comparison *on the order of*: $f(n) \sim g(n)$ meaning $\lim_{n\rightarrow+\infty}\big(f(n)/g(n)\big) = 1$. This allows constant factors to be taken into account.

3 ADEL Algorithm

We now describe the ADEL algorithm for computing a minimal subset. The input problem is given by: (1) the finite set $U = (u_1, ..., u_n)$ and (2) the upward-closed property $\mathcal{P}$. The input set U is stored as an array of size n where $U[i] = u_i, i \in \{1, ..., n\}$.

In Algorithm 1, at line 1, the array U is shuffled. This will rule out any particular structure in the set U and, informally speaking, will ensure that minimal subsets are uniformly distributed in U. Procedure $\mathrm{FindNext}(X, U, i, \mathcal{P}, s)$ is in charge of finding and returning the largest index j ($i \leq j$) such that $\mathcal{P}(X \cup U_{j\rightarrow})$.

The ideas of *acceleration* and *dichotomy* exploited in the $\mathrm{FindNext}$ function in Algorithm 2 rely on the fact that the index j returned by this procedure is likely to be quite close to index i. This is of course especially true if the size of the selected minimal subset is large. So it generally pays off to search for such an index j starting from index i with an acceleration phase (trying $i+s$, $i+2s$, $i+4s$, ..., $i+2^k s$) until the first index such that $\neg\mathcal{P}(X \cup U_{i+2^k s\rightarrow})$ and then applying a dichotomic search on the index segment $[i+2^{k-1}s, i+2^k s)$. We can elaborate further on this idea by learning the initial step s of the acceleration phase from the previous calls to the $\mathrm{FindNext}$ procedure. Indeed, as the initial set U has been shuffled we can expect

the probability distribution of the distances between consecutive elements of X not to depend much on which element is considered, thus this information can be estimated from the previous elements. In this context, the initial value of s represents the expected average distance between two successive elements of the selected minimal subset. For the first call to $\mathrm{FindNext}$, we take $s = n$ so this first call boils down to the pure dichotomy algorithm. For the later calls, the initial s is computed as the average of the distance between successive elements already added to the minimal set X under construction.

When the selected minimal subset is small, it's worth to check property $\mathcal{P}$ on the current subset X each time a new element u_i is added. But when the minimal subset is large, with a size typically getting close to n, this may represent a large proportion of the property checks of the algorithm. On the other side, it is not necessary to stop the algorithm as soon as one has proved the current subset X satisfies the property: one can let function $\mathrm{FindNext}$ show that the current subset does not have to be extended. It will show it with approximately $\log_2(n)$ property checks in the acceleration step. So we consider that once the size of the current subset X is larger than $\log_2(n)$, as the effort to be spent for proving the property for the selected minimal subset with function $\mathrm{FindNext}$ won't exceed the effort already spent checking the property for each elements added to X so far, we can stop checking the property in Algorithm 1, line 13. This is the idea of *lazy checks*.

Algorithm 1 ADEL($U, \mathcal{P}$)

Require: $\mathcal{P}(U)$
1: $\mathrm{Shuffle}(U)$ ▷ Called once: $O(n)$
2: $X \leftarrow \emptyset$ ▷ X: minimal subset under construction
3: $i \leftarrow 0$ ▷ i: index of last element added to X
4: $s \leftarrow n, d_1 \leftarrow 0, d_0 \leftarrow 0$
5: **loop**
6: $j \leftarrow \mathrm{FindNext}(X, U, i+1, \mathcal{P}, s)$
7: **if** $j > n$ **then** ▷ Last acceleration showed $\mathcal{P}(X)$ holds
8: **return** X
9: $d_0 \leftarrow d_0 + 1$
10: $d_1 \leftarrow d_1 + j - i, s = \lfloor d_1/d_0 \rfloor$ ▷ Distance Estimation
11: $i \leftarrow j$
12: $X \leftarrow X \cup \{U[i]\}$
13: **if** $i \leq \log_2(n) \wedge \mathcal{P}(X)$ **then** ▷ Lazy check
14: **return** X

Algorithm 2 FindNext(X, U, i, $\mathcal{P}$,s)

Require: $\mathcal{P}(X \cup U_{i\rightarrow})$
1: $l \leftarrow i, r \leftarrow n$
2: **while** $(l \leq n) \wedge \mathcal{P}(X \cup U_{i+s\rightarrow})$ **do** ▷ Accelerate
3: $l \leftarrow i + s, s \leftarrow s * 2$
4: **if** $l > n$ **then**
5: **return** l ▷ Acceleration showed $\mathcal{P}(X)$ holds
6: **else**
7: $r \leftarrow i + s - 1$
8: **while** $l \neq r$ **do** ▷ Dichotomize
9: $m \leftarrow \lceil (l + r)/2 \rceil$
10: **if** $\mathcal{P}(X \cup U_{m\rightarrow})$ **then**
11: $l \leftarrow m$
12: **else**
13: $r \leftarrow m - 1$
14: **return** l

4 Performance Study

A complexity study of ADEL as well as some experiments not reported here by lack of space show that:

- it is *optimal* for small subsets, with a complexity in the order of $\log_2(n)$ for a unique minimal subset of size 1.
- it is *optimal* for large subsets, with a complexity in the order of n for a minimal subset of size n.
- the average number of checks behaves continuously in between those two extremal cases and outperforms all variants of the ADEL algorithm obtained by switching off any of its features (acceleration, distance estimation, lazy checks).
- in the case of a unique minimal subset of size one, it performs in average ³⁄₂ times less checks than the recursive `QuickXplain` algorithm [4] (33% less checks).
- in the case of a unique minimal subset of size n, for large values of n, it performs about twice less checks than the `QuickXplain` algorithm (50% less checks).
- in the case of a unique minimal subset of size n, for large values of n it performs about $\log_2(n)$ times less checks than the `DC` algorithm that implements a pure dichotomical search [3].
- in between those two extremal cases it consistently performs less property checks than the `QuickXplain` algorithm (between 10% and 50% less checks in the instances generated for our experiments).

5 Conclusion

This paper tackles the problem of finding a minimal subset satisfying an upward-closed property. No other assumption is made about the property being checked and the objective is to minimize the number of checks. The approach can be applied to the extraction of MUCs in Constraint Networks. The proposed `ADEL` algorithm initially shuffles the set of elements and then exploits the probabilistic properties of the position of consecutive elements in the selected subset. The algorithm is shown to be optimal on both extreme cases with minimal subsets of size 1 and n and to perform less property checks than existing `DC` and `QuickXplain` approaches. An additional interest of ADEL relies on the fact it is by nature an iterative algorithm and is thus easier to implement than a recursive one like `QuickXplain` if recursion is to be avoided. Algorithm `ADEL` is used as the implementation of the MUC extraction functionality (Conflict Refiner) in IBM ILOG CP Optimizer since version 12.5.

REFERENCES

[1] John W. Chinneck, 'Finding a useful subset of constraints for analysis in an infeasible linear program', *INFORMS Journal on Computing*, **9**, 164–174, (1997).
[2] John W. Chinneck, *Feasibility and Infeasibility in Optimization: Algorithms and Computational Methods*, Springer, 2007.
[3] Fred Hemery, Christophe Lecoutre, Lakhdar Sais, and Frédéric Boussemart, 'Extracting mucs from constraint networks', in *Proc. 17th European Conference on Artificial Intelligence (ECAI'06)*, (2006).
[4] Ulrich Junker, 'QuickXplain: Preferred explanations and relaxations for over-constrained problems', in *Proc. AAAI-04*, (2004).
[5] Narendra Jussien and Olivier Lhomme, 'Local search with constraint propagation and conflict-based heuristics', *Artificial Intelligence*, **139**(1), 21–45, (July 2002).
[6] Joao P. Marques Silva and Karem A. Sakallah, 'Conflict analysis in search algorithms for satisfiability', in *Proceedings of the 8th International Conference on Tools with Artificial Intelligence (ICTAI'96)*, (1996).