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

P. Laborie - IBM Software Group - laborie@fr.ibm.com

Objective

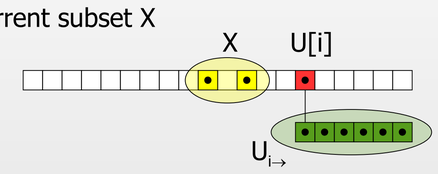
- Given an unfeasible Constrained Optimization problem M, extract a **Minimal Unsatisfiable Core (MUC)** that is, a minimal subset (in the sense of set inclusion) of unfeasible constraints X⊆M
- Example: x, y, z in { 1, 2, 3 }
  constraint C1: AllDifferent(x,y,z)
  constraint C2: z == x + 2y
  constraint C3: y < x
  constraint C4: z == 2x
- This problem is unfeasible. A possible MUC is {C1,C2}.
- Algorithm developed in the context of providing a Conflict Refiner for CP models in IBM ILOG CPLEX Optimization Studio: when designing optimization models, it is usual to face unfeasible instances (due to errors in the model or the data). Identifying a MUC generally helps to explain the unfeasibility
- Given the complexity of the automatic search, we consider the engine as a **black-box** whose answer is Yes/No for the feasibility of a subset of constraints X⊆M

Abstracted problem

- Let U a finite set of cardinality n and P an **upward-closed** (or monotonous) property on its powerset 2^U, that is, such that:
  $(X \subseteq Y \subseteq U) \wedge P(X) \Rightarrow P(Y)$
- Minimal subset: X⊆U is a **minimal subset** satisfying P iff:
  P(X) and $\forall Y \subset X, \neg P(Y)$
- In our original problem:
  - U ≡ Set of constraints in the model
  - P(X) ≡ Subset of constraints X is unfeasible
- Problem definition:
  - Let U such that P(U), **find a minimal subset X satisfying P**
  - The complexity of the resolution algorithm is measured as the **number of property checks**. As this is an homogeneous measure we can use a more fine grain complexity comparison than the traditional *big O* comparison. We use a comparison *on the order of* :
    $f(n) \sim g(n)$ means $\lim_{n\to\infty}(f(n)/g(n))=1$

A family of iterative algorithms

- Elements of U are stored in an array of size n
- Array is shuffled so as to rule out any particular structure
- Algorithm will select the "rightmost" minimal set in the array
- Scan elements from left to right
- Grows a current subset X



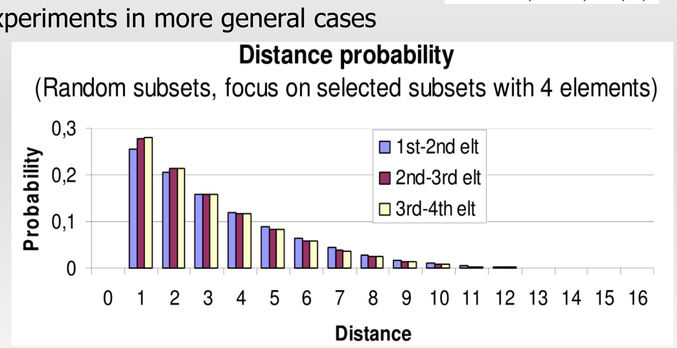
- Next element U[i] to be added to current subset X is the one with the smallest index i such that $P(X \cup U_{i\rightarrow})$
- When a new element U[i] is added, property P can be checked on current subset X (if P(X), algorithm can be stopped)
- Algorithms in this family differs according to the way to find the next element U[i]

Position of consecutive elements

- In the particular case of a single minimal subset of size m **uniformly distributed** in U, the probability p(k) that the distance between the i^{th} and $i+1^{th}$ element is k:
  - Does not depend on the position i
  - Exponentially decreases with k

$$p(k) = \binom{n-k}{m-1} / \binom{n}{m}$$

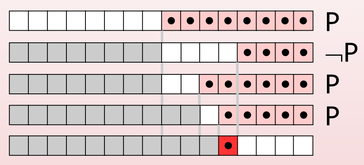
- Experiments in more general cases



- Next element of the subset is often **close** to the current one
- Estimated distance between consecutive elements can be **learned**

Dichotomy

- A dichotomy algorithm DC has been proposed in [2]. It performs a dichotomic search in order to find next element U[i].



- This algorithm is efficient for small minimal subsets. It is optimal for minimal subsets of size 1 with a complexity in $\sim\log_2(n)$.
- On the other side of the spectrum, it is not efficient for large minimal subsets. For a minimal subset of size n it may require $O(n \log_2(n))$ property checks.
- Our algorithm uses a dichotomic search:
  - To select the first element
  - For the following elements: after the acceleration has reduced the scope of the dichotomy

Acceleration / progression

- The probabilistic study of the distance between consecutive elements suggests that it may pay off to look for the next element close to the current one
- Starting from last added element at index i, an acceleration algorithm tries $i+1, i+2, i+4, \ldots, i+2^k$ until first index $j=i+2^k$ such that $\neg P(X \cup U_{j\rightarrow})$
- The algorithm then applies a dichotomy on the reduced index segment $[i+2^{k-1}, i+2^k)$
- A similar approach is used in [3]
- Main differences of our algorithm compared to [3]:
  - The first iteration uses dichotomy, not acceleration
  - The initial step in the acceleration is learned (see below), it is not necessarily 1
  - Up to a certain size, property is checked on current subset X each time a new element is added
- These differences allow showing optimality of the proposed algorithm for small minimal subsets

Estimation (of distance)

- The probabilistic study of the distance between consecutive elements suggests that the distribution of this distance does not depend much on which element is considered
- Let *s* denote the size of the initial acceleration step (acceleration algorithm tries $i+s, i+2s, i+4s, \ldots, i+2^k s$)
- For the first element, we take $s=n$ so the search boils down to a pure dichotomic search (line 4 on Algorithm 1)
- For the next elements, the initial acceleration step is computed as the average distance between past elements (line 10 on Algorithm 1)
- Note that in case of a minimal subset of size n, after first element is found (at position 1), the initial acceleration step is always 1

Lazy checks

- Property P(X) may be checked when a new element is added to X, this allows stopping the search as soon as X satisfies P
- This is efficient for small minimal subsets. Typically, for a minimal subset of size 1, once the initial dichotomy has selected an (the) element, search will be stopped immediately
- This is inefficient for large subsets. Typically, for a minimal subset of size n, n-1 useless checks P(X) will be performed
- When the last element of the minimal subset is added to X, if we do not check P(X), it will take the acceleration about $O(\log_2(n))$ additional checks to show that X satisfies P(X)
- The idea is to check P(X) only for the first $\log_2(n)$ elements. For larger subsets, the $O(\log_2(n))$ price to pay for letting the acceleration show the property will not exceed the $O(\log_2(n))$ we already payed to check the first $\log_2(n)$ elements so the algorithm can stop checking P(X) (line 13 of Algorithm 1)

Detailed **ADEL** algorithm

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

Require: $\mathcal{P}(U)$

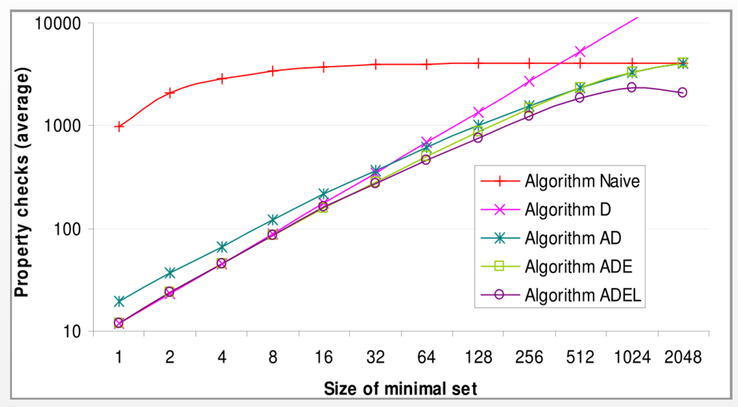
1: Shuffle(U) ▷ Called once: $O(n)$
2: $X \leftarrow \emptyset$ ▷ X: minimal subset under construction
3: $i \leftarrow 0$ ▷ i: index of last element added to X
4: $s \leftarrow n, d_1 \leftarrow 0, d_0 \leftarrow 0$
5: **loop**
6: $j \leftarrow$ FindNext($X, U, i+1, \mathcal{P}, s$)
7: **if** $j > n$ **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

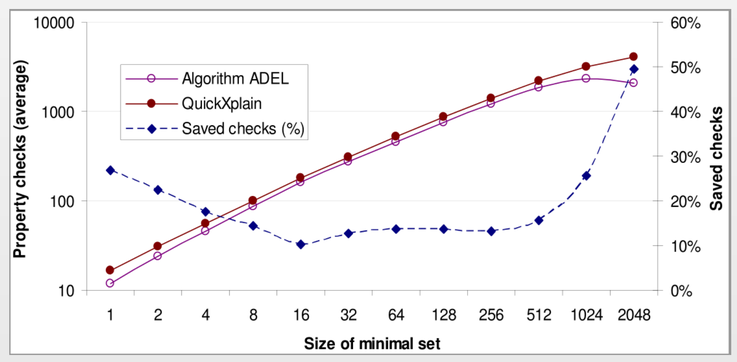
Algorithm 2 FindNext($X, U, i, \mathcal{P}, s$)

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

Results





- Algorithm ADEL is **optimal** in both extreme cases:
  - For small subsets: $\sim\log_2(n)$ for minimal subset of size 1
  - For large subsets: $\sim n$ for minimal subset of size n
- Compared to QuickXplain [1]:
  - For small subsets, it performs 1.5 times less checks
  - For large subsets, it performs twice less checks
- It behaves continuously in between these extreme cases and outperforms all variants as well as QuickXplain
- ADEL algorithm is used as the implementation of the Conflict Refiner functionality for CP models in IBM ILOG CPLEX Optimization Studio since version 12.5

References

[1] U. Junker. *QuickXplain: Preferred explanations and relaxations for over-constrained problems*. In Proc. AAAI-04. 2004.

[2] F. Hemery, C. Lecoutre, L. Sais and F. Boussemart. *Extracting MUCs from constraint networks*. In Proc. ECAI-06. 2006.

[3] J. Marques-Silva, M. Janota and A. Belov. *Minimal Sets over Monotone Predicates in Boolean Formulae*. In Proc. CAV-13. 2013.