An Optimal Non-Recursive Algorithm for Finding a Minimal Subset Satisfying an Upward-Closed Property
! WORKING PAPER !

Philippe Laborie
IBM
9 rue de Verdun
94253 Gentilly Cedex, France

Introduction

Given a finite set U, finding a minimal subset of U satisfying a given property $\mathcal{P}$ is a frequent problem in computer science. For instance, this problem can be found in diagnosis (*find a minimal subset of faulty components of a system that explains the current observations*) or in non-monotonic logics (*find a minimal subset of abnormalities in the clauses that restore admissibility*).

In several important applications the property $\mathcal{P}$ is upward-closed that is, whenever it holds for a subset X it also holds for any superset of X. This is typically the case of the identification of minimal infeasible subsets (conflicts) of constraints in optimization models (Chinneck 2007): if a subset X of constraints is infeasible, any superset of X clearly is infeasible too.

In general, checking property $\mathcal{P}$ for a particular subset is an expensive operation (it can be NP-complete) and often the approaches for solving this minimal subset problem rely on the particular features of the property $\mathcal{P}$ being considered. For instance in the optimization context mentioned above techniques such as *elastic filters* can be used for Linear Programming models (Chinneck 1997), *explanations* recording (Jussien & Lhomme 2002) for Constraint Programming models or *no-goods* learning for Boolean Satisfiability models (Marques Silva & Sakallah 1996).

But the logic behind property check $\mathcal{P}(X)$ can be so complex that it may turn out to be necessary or advantageous to consider it as a black box operation. Property $\mathcal{P}(X)$ could for instance be the outcome of a complex simulation process based on a set of input events X. In the context of minimal conflicts identification, the advances in the optimization state-of-the-art result in increasingly sophisticated and efficient engines. Engines sophistication makes it harder to implement intrusive methods to compute minimal conflicts whereas increase of engines efficiency makes infeasibility property check faster. Both aspects tend to make the black box approach more attractive.

In this paper we investigate the problem of finding a minimal subset satisfying an upward-closed property *without any knowledge on the property* except for the assumption that it is upward-closed. The complexity measure we consider is therefore the number of property checks performed. The work closest to ours is probably the `QuickXplain` recursive algorithm (Junker 2004) developed in the context of the computation of minimal conflicts for constraint programming using a black box approach. An iterative algorithm to solve this problem has also been proposed in (Hemery *et al.* 2006).

After introducing some notations and formally defining the problem, the paper presents a statistical study of the relative positions of elements of the selected minimal set. This study highlights a couple of interesting properties that are exploited in the following section of the paper that gradually introduces the proposed algorithm `ADEL` (standing for the four ingredients of the approach: Dichotomy, Acceleration, Estimation and Lazy checks). We show in the last section of the paper that the complexity of this algorithm is optimal both for small and large minimal subsets and that it performs in average between 10% and 50% less property checks than the `QuickXplain` algorithm. Proofs of properties are given in a separate section at the end.

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:

$$(X \subseteq Y \subseteq U) \wedge \mathcal{P}(X) \Rightarrow \mathcal{P}(Y)$$

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

In the sequel of this paper we assume $\mathcal{P}(U)$, that is there exist at least one minimal subset. Of course, in general there is not a unique minimal subset: there can be an exponential number of such subsets, for instance the set of minimal subsets could be all subsets of cardinality $n/2$ which is lower bounded by $2^{n/2}$.

Without loss of generality we assume a total order $\prec$ over the elements in U so that the elements can be indexed: $U = (u_1, ..., u_n)$ with $u_i \prec u_{i+1}$.

The total order $\prec$ over U implies a lexicographic total order $\prec$ over the powerset 2^U. Let $X, Y \subseteq U, X \neq Y$:

$$X \prec Y \Leftrightarrow \exists j \in [1,n] : \begin{cases} \forall i < j, (u_i \in X) \Leftrightarrow (u_i \in Y) \\ u_j \in X \\ u_j \notin Y \end{cases}$$

We denote X^* the unique minimal subset $X^* \in \mathcal{M}$ that is maximal with respect to the total order $\prec$.

The problem studied in this paper is the design of efficient algorithms to compute 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 to compute a minimal subset as the number of property checks it performs.

Note that in practice, relation $\prec$ can be used to express preferences over minimal subsets, in this case, the algorithms presented in this paper will compute a preferred minimal subset. The definition of relation $\prec$ is the same as in (Junker 2004).

Let $X \subseteq U$ and $m = |X|$. Without loss of generality, we can sort the m elements of X by their index in U: $X = \{u_{\pi(X,j)}\}_{j\in[1,m]}$ with $1 \leq \pi(X,j) < \pi(X,j+1) \leq n$. Integer $\pi(X,j)$ denotes the position in U of the j^{th} element of subset X.

Figure 1 illustrates a set U, the set of all minimal subsets ordered by relation $\prec$, subset X^* and the distances between consecutive elements in X^*.

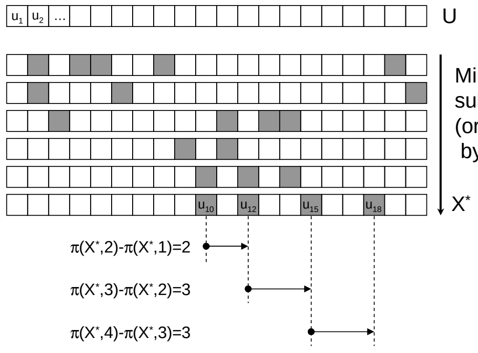

Figure 1: Minimal subsets

For $1 \leq i \leq n$, the subset $\{u_j | i \leq j \leq n\}$ of all u_j with index greater than i is denoted $U_{i\rightarrow}$.

Statistical study

In this section, we study (either analytically or experimentally) the distribution of the positions $\pi(X^*,j)$ and distances $\pi(X^*,j+1) - \pi(X^*,j)$ between the elements of minimal subset X^* in U under some assumptions. The main conclusions of this study will be used to design efficient algorithms for computing X^*.

Unique minimal subset

In this section we assume U contains a unique minimal subset of size m with $0 < m \leq n$ uniformly distributed in U.

Property 1 gives the probability distributions of the j^{th} element of the minimal subset.

Property 1 *In the case of a unique minimal subset of size m, the probability $P\big(\pi(X^*,j) = k\big)$ that the position of the j^{th} element of the minimal subset is k, denoted $p_j(k)$ is given by:*

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

These distributions are illustrated on figure 2 for $n = 20$ and $m = 4$.

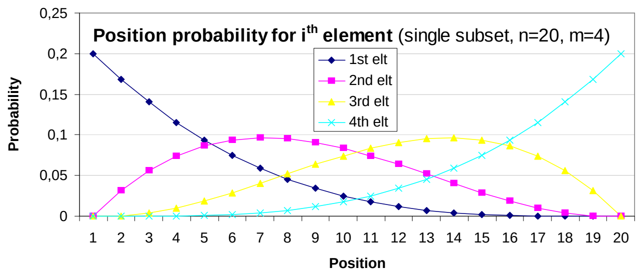

Figure 2: Position probabilities

Property 2 gives the probability distributions of the distances between consecutive elements of the unique minimal subset in U.

Property 2 *In the case of a unique minimal subset of size m, the probability that the distance $\pi(X^*,j+1) - \pi(X^*,j)$ between two consecutive elements is k does not depend on j and is the same as the probability that the first element of the minimal subset is at position k ($p_1(k)$):*
$\forall j \in [1,m), P\big(\pi(X^*,j+1) - \pi(X^*,j) = k\big) = p_1(k)$
With:

$$p_1(k) = \binom{n-k}{m-1}/\binom{n}{m}$$

From the formula, probability $p_1(k)$ can be computed recursively as follows:

- $p_1(1) = m/n$
- $\forall k \in [2, n-m+1], p_1(k+1) = \big(1 - \frac{m-1}{n-k}\big)\, p_1(k)$
- $\forall k > n-m+1, p_1(k) = 0$

The probability distribution $p_1(k)$ is illustrated on figure 3 for $n = 20$ and $m = 4$. This probability exponentially decreases with k. This result can be seen as a discrete version of the probability density of the distance x between two neighbor points from a set of points randomly and uniformly spread on a line which is $p(x) = \frac{1}{d}\, e^{-x/d}$ where d is the mean distance between two neighbor points (Demaret & Gareet 1977).

Disjoint minimal subsets of random size

In this section we assume U contains c uniformly distributed disjoint minimal subsets of random sizes m_i, $i \in [1,c]$ with $0 < \sum_{i\in[1,c]} m_i \leq n$. The following property holds, independently from the probability distribution of the minimal subset sizes m_i:

Property 3 *In the case of disjoint minimal subsets, the probability that the distance $\pi(X^*,j+1) - \pi(X^*,j)$ between two consecutive elements is k does not depend on j and is a non-increasing function of k.*

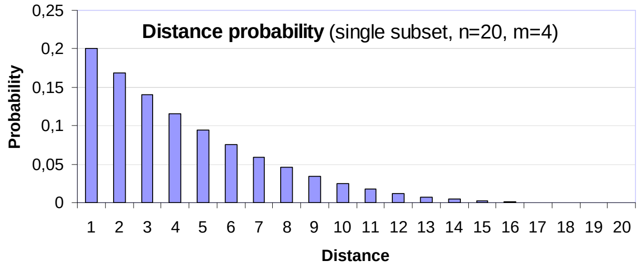

Figure 3: Distance probability between two consecutive elements

Figure 4 gives the distributions of the positions of the elements of minimal subset X^* in U for the particular case of a set of 5 independent minimal subsets of size 4. It is to be noted that in this case, all the elements of U belong to a unique minimal subset. Figure 5 shows the distribution of the distance between consecutive elements of X^* in this case.

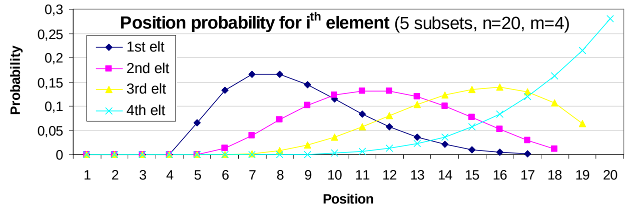

Figure 4: Position probabilities

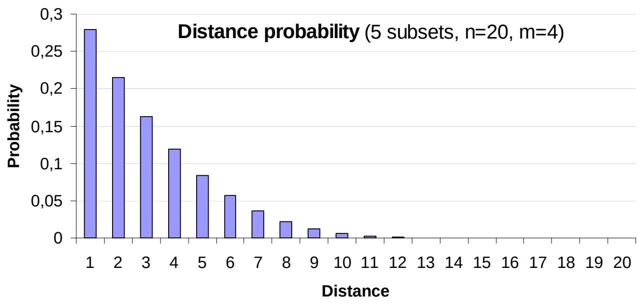

Figure 5: Distance probability between two consecutive elements

Random minimal subsets of random size

In this section, we experimentally study the distributions of random minimal subsets of random size. For each sample, a set of c subsets of size uniformly selected in $[1, n]$ is generated and the non-minimal subsets are eliminated.

For $n = 20$ and $c = 15$, Figure 6 shows the probability distribution of (1) the size of all generated minimal subsets and (2) the size of the selected minimal subset X^*. As we could expect, selected subsets X^* are usually smaller than the average size of all minimal subsets. This is because large minimal subsets are in general lower than small minimal subsets in the sense of total order $\prec$.

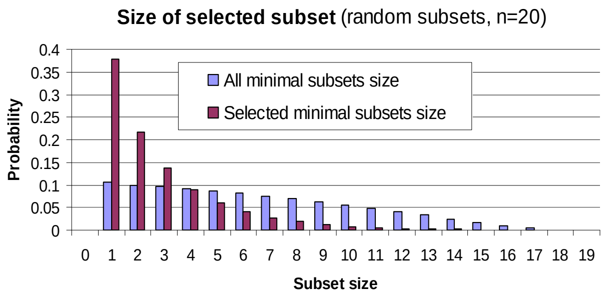

Figure 6: Minimal subset sizes

For the same configuration and only considering the samples for which the selected minimal subset X^* is of size 4, Figure 7 displays the probability distributions of the distances between consecutive elements in X^*. Here, unlike for disjoint subsets, we see that these distributions slightly depend on which consecutive elements of X^* are considered. Elements of X^* tend to be slightly more concentrated in the end. It is to be noted that, here again, all those probability distributions are non-increasing functions of the distance.

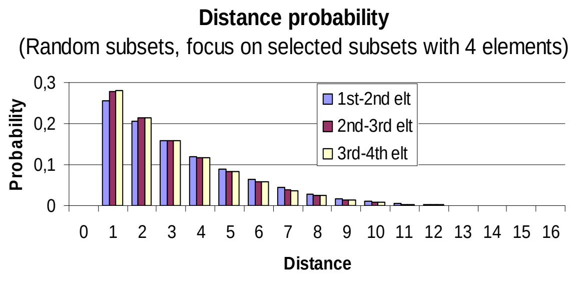

Figure 7: Distance probability between two consecutive elements

Conclusion of the statistical study

We can draw three conclusions from this short study on uniformly distributed minimal subsets:

1. When there are several minimal subsets, the selected one X^* tend to be smaller than the average. This is a good property as in most of the applications one is in general interested in minimal subsets of small cardinality.
2. The probability distribution of the distance between consecutive elements of the selected minimal subset X^* is in general a non-increasing function. This was formally shown in the case of disjoint minimal sets. Stated otherwise, the $j+1^{th}$ element of the selected minimal subset is likely to be close to the j^{th} element.
3. The probability distributions of the distance between consecutive elements of the selected minimal subset X^* do not heavily depend on which element is considered. In

case of disjoint minimal subsets it can even be shown that they are independent.

Algorithms

The input problem is given by:

- The finite set $U = (u_1, ..., u_n)$
- The upward-closed property $\mathcal{P}$

All the algorithms studied in this paper share the same framework presented on Algorithm 1.

The input set U is stored as an array of size n where $U[i] = u_i, i \in \{1, ..., n\}$.

Algorithm 1 FindMinimalSubset(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: repeat
5:     $i \leftarrow \mathrm{FindNext}(X, U, i+1, \mathcal{P})$
6:     $X \leftarrow X \cup \{U[i]\}$
7: until $(i = n) \vee \mathcal{P}(X)$
8: return $X$
```

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. This shuffling defines a total order $\prec$ among the elements of U. Procedure $\mathrm{FindNext}(X, U, i, \mathcal{P})$ is in charge of finding and returning the largest index j ($i \leq j \leq n$) such that $\mathcal{P}(X \cup U_{j\rightarrow})$. The algorithms described in this paper differ according to their implementation of this procedure.

Figure 8 illustrates a set U, the set of all minimal subsets ordered by relation $\prec$ and the subsets X and $U_{i\rightarrow}$ at the iteration of algorithm 1 when $U[i]$ is added to the current minimal subset X.

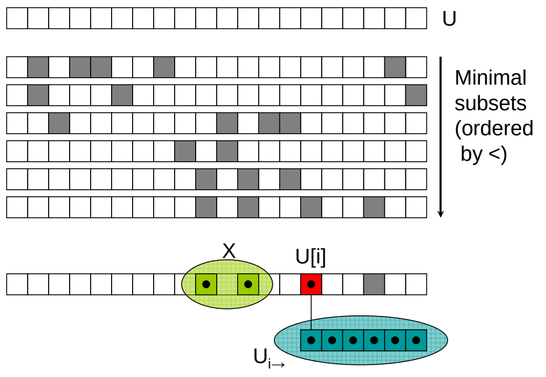

Figure 8: Minimal subsets

Property 4 *Algorithm 1 is correct and returns the unique minimal subset $X^* \in \mathcal{M}$ that is maximal with respect to the total order $\prec$.*

Naive Algorithm

The naive algorithm simply iterates over all elements $u_j, i < j$ until the first j such that $\neg\mathcal{P}(X \cup U_{j\rightarrow})$ in order to return index $j-1$ (See Algorithm 2). This algorithm is clearly a correct implementation for procedure FindNext.

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

```
Require: $\mathcal{P}(X \cup U_{i\rightarrow})$
1: $j \leftarrow i$
2: repeat
3:     $j \leftarrow j+1$
4: until $(j = n+1) \vee \neg\mathcal{P}(X \cup U_{j\rightarrow})$
5: return $j-1$
```

Property 5 *If $\pi(X^*, |X^*|) < n$, the naive algorithm performs $\pi(X^*, |X^*|) + |X^*|$ property checks. Otherwise, if $\pi(X^*, |X^*|) = n$, the naive algorithm performs $n + |X^*| - 2$ property checks.*

Dichotomy Algorithm

The dichotomy algorithm corresponds to the DC algorithm proposed in (Hemery *et al.* 2006). It performs a dichotomic search, see Algorithm 3.

Algorithm 3 FindNext-D(X, U, i, $\mathcal{P}$)

```
Require: $\mathcal{P}(X \cup U_{i\rightarrow})$
1: $l \leftarrow i, r \leftarrow n$
2: while $l \neq r$ do
3:     $m \leftarrow \lceil (l+r)/2 \rceil$
4:     if $\mathcal{P}(X \cup U_{m\rightarrow})$ then
5:         $l \leftarrow m$
6:     else
7:         $r \leftarrow m-1$
8:     end if
9: end while
10: return $l$
```

The dichotomy algorithm is clearly a correct implementation for procedure FindNext.

Property 6 *In average, the dichotomy algorithm performs $O(|X^*| \times \log_2(n))$ property checks.*

If $|X^*| = 1$, the dichotomy algorithm performs $O(\log_2(n))$ checks which is clearly better than the $O(n)$ checks of the naive algorithm. At the other side of the spectrum if $|X^*| = n$, the dichotomy algorithm performs worse ($O(n \log_2(n))$) than the naive one ($O(n)$). The purpose of the *accelerate & dichotomize* algorithms described in next sections is to take the best of both worlds.

Accelerate & Dichotomize Algorithm

The idea of the basic *accelerate & dichotomize* algorithm is to exploit the second conclusion of the statistical study that shows that the index j returned by procedure $\mathrm{FindNext}(X, U, i, \mathcal{P})$ 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 may pay off to search for such an index starting from index i with an acceleration phase (trying $i+1$, $i+2$, $i+4$, ..., $i+2^k$) until the first index such that $\neg\mathcal{P}(X \cup U_{i+2^k\rightarrow})$ and then applying a dichotomic search on the index segment $[i+2^{k-1}, i+2^k]$. This idea leads to Algorithm 4.

Algorithm 4 FindNext-AD(X, U, i, $\mathcal{P}$)

```
Require: $\mathcal{P}(X \cup U_{i\rightarrow})$
 1: $l \leftarrow i, r \leftarrow n$
 2: $s \leftarrow 1$                                      ▷ Initial step
 3: $s_{max} \leftarrow \lfloor (n-i)/2 \rfloor$          ▷ Maximal step
 4: while $(s \leq s_{max}) \wedge \mathcal{P}(X \cup U_{i+s\rightarrow})$ do   ▷ Accelerate
 5:     $l \leftarrow i+s, s \leftarrow s*2$
 6: end while
 7: if $s \leq s_{max}$ then
 8:     $r \leftarrow i+s-1$
 9: end if
10: while $l \neq r$ do                                  ▷ Dichotomize
11:     $m \leftarrow \lceil (l+r)/2 \rceil$
12:     if $\mathcal{P}(X \cup U_{m\rightarrow})$ then
13:         $l \leftarrow m$
14:     else
15:         $r \leftarrow m-1$
16:     end if
17: end while
18: return $l$
```

Accelerate, Dichotomize & Estimate Algorithm

The *estimate, accelerate & dichotomize* algorithm elaborates further on the previous algorithm by learning the initial step s of the acceleration phase from the previous calls to the $\mathrm{FindNext}$ procedure. This is supported by the third conclusion of the statistical study that shows that the probability distribution of the distances between consecutive elements of X^* does not 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 minimal set. For the first call to $\mathrm{FindNext}$, we take $s = n$ so, as $s > s_{max}$, 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. See Algorithms 5 and 6.

Accelerate, Dichotomize, Estimate & Lazy checks Algorithm

We finally propose a last improvement to the previous algorithm. Property checks on the subset X under construction in Algorithm 5, line 11 will be called m times for a minimal set of size m. If m is large and typically, getting close to n, this may represent a large proportion of the property checks of algorithm `ADE`. 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 the function FindNext-ADE show that the current subset does not have to be extended. It will show it with approximatively $\log_2(n)$

Algorithm 5 FindMinimalSubset-ADE(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: $s_0 \leftarrow n$, $d_1 \leftarrow 0$, $d_0 \leftarrow 0$
 5: repeat
 6:     $j \leftarrow$ FindNext-ADE$(X, U, i+1, \mathcal{P}, s_0)$
 7:     $d_0 \leftarrow d_0 + 1$
 8:     $d_1 \leftarrow d_1 + j - i$, $s_0 = \lfloor d_1/d_0 \rfloor$
 9:     $i \leftarrow j$
10:     $X \leftarrow X \cup \{U[i]\}$
11: until $(i = n) \vee \mathcal{P}(X)$
12: return $X$
```

Algorithm 6 FindNext-ADE(X, U, i, $\mathcal{P}$,s_0)

```
Require: $\mathcal{P}(X \cup U_{i\rightarrow})$
 1: $l \leftarrow i, r \leftarrow n$
 2: $s \leftarrow s_0$                                    ▷ Initial step
 3: $s_{max} \leftarrow \lfloor (n-i)/2 \rfloor$          ▷ Maximal step
 4: while $(s \leq s_{max}) \wedge \mathcal{P}(X \cup U_{i+s\rightarrow})$ do   ▷ Accelerate
 5:     $l \leftarrow i+s, s \leftarrow s*2$
 6: end while
 7: if $s \leq s_{max}$ then
 8:     $r \leftarrow i+s-1$
 9: end if
10: while $l \neq r$ do                                  ▷ Dichotomize
11:     $m \leftarrow \lceil (l+r)/2 \rceil$
12:     if $\mathcal{P}(X \cup U_{m\rightarrow})$ then
13:         $l \leftarrow m$
14:     else
15:         $r \leftarrow m-1$
16:     end if
17: end while
18: return $l$
```

property checks in the acceleration step. So the idea is to 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 X^* with function $\mathrm{FindNext\text{-}ADE}$ 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 5, line 11. This idea results in a *lazy* version of algorithm `ADE` denoted `ADEL`[1] and shown in Algorithms 7 and 8.

Algorithm 7 FindMinimalSubset(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: $s_0 \leftarrow n$, $d_1 \leftarrow 0$, $d_0 \leftarrow 0$
 5: repeat
 6:     $j \leftarrow$ FindNext($X, U, i+1, \mathcal{P}, s_0$)
 7:     $d_0 \leftarrow d_0 + 1$
 8:     $d_1 \leftarrow d_1 + j - i$, $s_0 = \lfloor d_1/d_0 \rfloor$
 9:     $i \leftarrow j$
10:     $X \leftarrow X \cup \{U[i]\}$
11: until $(i = n) \vee (i \leq \log_2(n) \wedge \mathcal{P}(X))$
12: return $X$
```

Algorithm 8 FindNext(X, U, i, $\mathcal{P}$,s_0)

```
Require: $\mathcal{P}(X \cup U_{i\rightarrow})$
 1: $l \leftarrow i$, $r \leftarrow n$
 2: $s \leftarrow s_0$                                        ▷ Initial step
 3: $s_{max} \leftarrow \lfloor (n-i)/2 \rfloor$               ▷ Maximal step
 4: while $(s \leq s_{max}) \wedge \mathcal{P}(X \cup U_{i+s\rightarrow})$ do   ▷ Accelerate
 5:     $l \leftarrow i + s$, $s \leftarrow s * 2$
 6: end while
 7: if $s \leq s_{max}$ then
 8:     $r \leftarrow i + s - 1$
 9: end if
10: while $l \neq r$ do                                        ▷ Dichotomize
11:     $m \leftarrow \lceil (l+r)/2 \rceil$
12:     if $\mathcal{P}(X \cup U_{m\rightarrow})$ then
13:         $l \leftarrow m$
14:     else
15:         $r \leftarrow m - 1$
16:     end if
17: end while
18: return $l$
```

Experimental Study

In this section, we experimentally compare the performances of the algorithms described in previous section under the different assumptions analyzed in the statistical study.

[1]Note that the same idea can apply to algorithms `D` and `AD`.

Unique minimal subset

We assume U, with $n = |U|$, contains a unique minimal subset of size m uniformly distributed in U.

The curves on Figures 9 and 11 show the performance of the different algorithms (number of property checks) as a function of the size of the minimal subset m and the size n of set U. Each point is computed as the average over 100 samples for the position of the minimal set in U. Note that both axis use a logarithmic scale. For the x-axis this is to give a special importance to small values of m.

Figure 9 shows for a fixed value of n ($n = 2048$) the average number of property checks performed by the different algorithms as a function of the size m of the unique minimal subset of U.

For the naive algorithm, for minimal subsets of unit size ($m = 1$), the average number of checks is roughly $n/2$ which is the average position of the element of the minimal set in U. For a minimal subset of size n, the naive algorithm has to perform $2\,n$ checks: n checks to verify that removing any element of U leads to a subset that do not check the property and n checks to verify whether the current subset under construction satisfies the property.

The dichotomy algorithm `D` roughly performs $m\,(1 + \log_2(n))$ checks which is linear with m for a fixed n. For $m = 1$ it performs $1 + \log_2(n)$ checks and for $m = n$, it requires about $n\,(1 + \log_2(n))$ checks.

Algorithm `AD` performs worse than `D` for small values of m. This is because when the elements of the selected minimal subset are sparse in U, the acceleration phase that starts with a unit step need to perform about $\log_2(n)$ iterations before to give the hand to the dichotomy phase that also will run in about $\log_2(n)$. For instance, for $m = 1$, it requires about $1 + 2\,\log_2(n)$ steps and `AD` is about twice worse as `D`. The situation is the opposite when m is getting closer to n: starting the acceleration phase with small values pays off and when $m = n$, the number of check of `AD` is $2\,n$ (same as the naive algorithm) which is a $n/\log_2(n)$ improvement factor over `D`.

As expected, algorithm `ADE` takes the best of both algorithms `D` and `AD`: by learning the initial acceleration step, it can achieve the same performance as `D` for small m and as `AD` for large m values.

The last improvement in algorithm `ADEL` pays off for large values of m by replacing the m checks in the FindNext functions by $\log_2(n)$ checks. For $m = n$, the number of checks is about $n + \log_2(n)$ where $\log_2(n)$ is the initial dichotomy to find the first element of the subset. So the improvement factor compared to `ADE` is about 2.

In fact, `ADEL` has *optimal complexity* with n both for *small* minimal sets ($m = 1$) and *large* ones ($m = n$). When $m = 1$ it boils down to a binary search in $\log_2(n)$. When $m = n$, complexity is in $n + 2\ \log_2(n)$ and any algorithm need indeed to at least check that each of the n elements is member of the minimal subset by performing a check without it. The number of useless checks, due to the fact we do not know *a priori* that $m = n$, is only $2\ \log_2(n)$ and is negligible compared to n: $\log_2(n)$checks for identifying the first element with the initial binary search and the $\log_2(n)$ initial checks in algorithm 5, line 11.

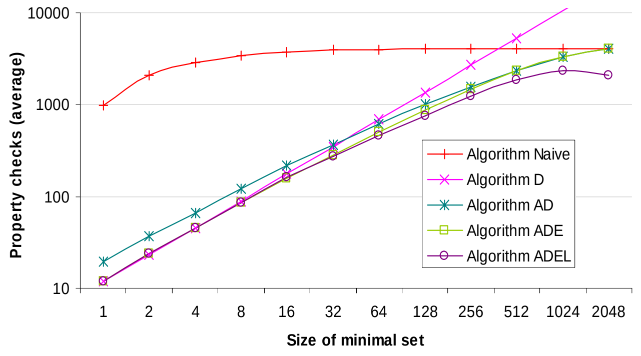

Figure 9: Number of property checks for $n = 2048$

Figure 10 compares the number of property checks performed by algorithms ADEL and QuickXplain (Junker 2004). QuickXplain is recapped in Algorithm 9.

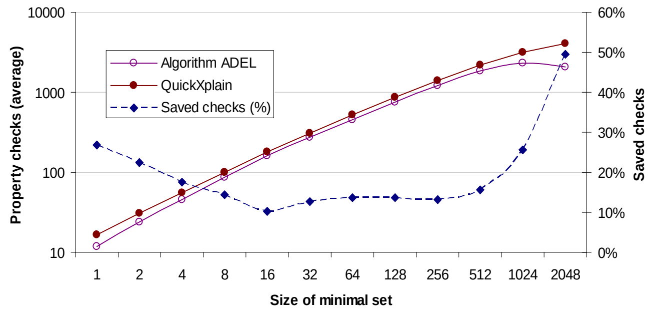

Figure 10: Number of property checks for $n = 2048$

We see that ADEL consistently performs less checks than QuickXplain (between 10% and 50% less checks).

Property 7 shows the average complexity for QuickXplain in case of a unique minimal subset of unit size. Note that this average complexity turns out to be the average value of the best and worst case complexities (resp. $\log_2(n)$ and $2\ \log_2(n)$) shown in (Junker 2004). Compared to the $\log_2(n)$ complexity of algorithm ADEL, this gives a 3/2 factor in favor of ADEL.

Property 7 *In case of a unique minimal set of size* 1*, the average complexity of the* QuickXplain *algorithm is* $3/2\ \log_2(n)$.

In case of a minimal subset of size n, QuickXplain need to perform $2\ n - 2$ property checks that is, the number of nodes of a binary tree with n leaves minus one because the top level node is not checked. Compared with the $n + 2\ \log_2(n)$ complexity of ADEL in this case this gives, for large values of n a factor 2 in favor of ADEL.

Figure 11 shows, for a fixed value of m ($m = 8$) the average number of property checks performed by the different algorithms as a function of the size n of U.

Random minimal subsets of random size

For each sample, a set of c subsets of size uniformly selected in $[1, n]$ is generated and the non-minimal subset are eliminated.

Algorithm 9 QuickXplain(U, $\mathcal{P}$)

```
Require: $\mathcal{P}(U)$
 1: return DoQuickXplain($\emptyset$, $\emptyset$, $U$, $\mathcal{P}$)

 2: function DoQuickXplain($B$, $\Delta$, $C$, $\mathcal{P}$)
 3:     if $\Delta \neq \emptyset$ and $\mathcal{P}(B)$ then
 4:         return $\emptyset$
 5:     end if
 6:     if $C = \{\alpha\}$ then
 7:         return $\{\alpha\}$
 8:     end if
 9:     Let $C = \{\alpha_1, ..., \alpha_n\}$ with $\forall i \in [1, n), \alpha_i \prec \alpha_{i+1}$
10:     $k \leftarrow \lceil n/2 \rceil$                                    ▷ Split
11:     $C_1 \leftarrow \{\alpha_k, ..., \alpha_n\}$
12:     $C_2 \leftarrow \{\alpha_1, ..., \alpha_{k-1}\}$
13:     $\Delta_2 \leftarrow$ DoQuickXplain($B \cup C_1$, $C_1$, $C_2$, $\mathcal{P}$)
14:     $\Delta_1 \leftarrow$ DoQuickXplain($B \cup \Delta_2$, $\Delta_2$, $C_1$, $\mathcal{P}$)
15:     return $\Delta_1 \cup \Delta_2$
16: end function
```

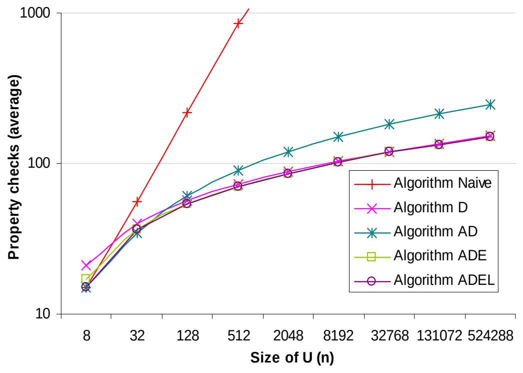

Figure 11: Number of property checks for $m = 8$

Curve on Figure 12 shows the performance of the different algorithms (number of property checks) for $n = 2048$ as a function of the number of subsets c generated. When c grows, the size of the selected minimal subset X^* tends to decrease. The average size of the selected subset is also shown on the figure as an indicator. We can make similar observations as in the previous section: algorithm ADEL outperforms all other algorithms.

The comparison with algorithm QuickXplain is shown on Figure 13. We see that ADEL consistently performs less checks than QuickXplain (between 10% and 30% less checks).

Conclusion

This article studies the problem of finding a minimal subset satisfying an upward-closed property. No assumption is made on the property being checked and the objective is to minimize the number of property checks. The proposed ADEL algorithm exploits the statistical properties of the subset element positions. We show that:

- it is *optimal* for small subsets, with a complexity domi-

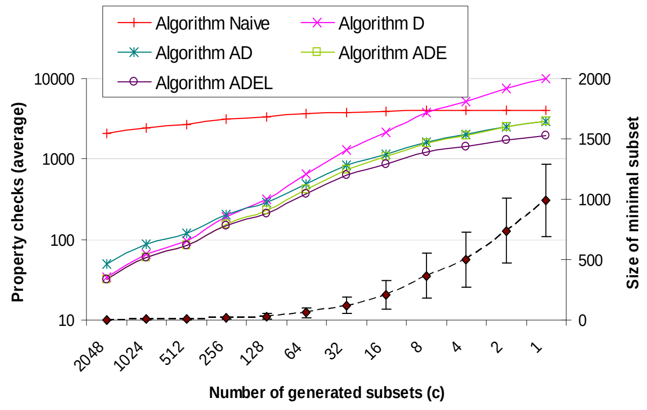

Figure 12: Number of property checks for $n = 2048$

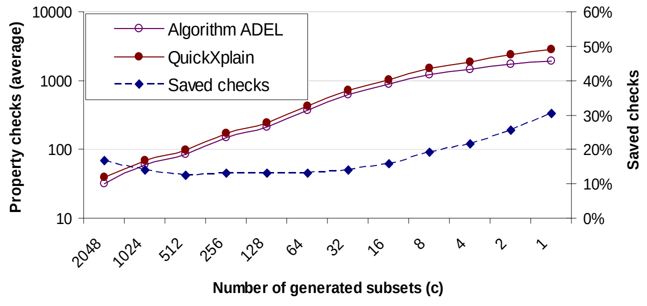

Figure 13: Number of property checks for $n = 2048$

nated by $\log_2(n)$ for a unique minimal subset of size 1.

- it is *optimal* for large subsets, with a complexity dominated by 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 studied in this paper.
- in the case of a unique minimal set of size one, we show that it performs in average 3/2 times less checks than the `QuickXplain` algorithm (33% less checks).
- in the case of a unique minimal set of size n, for large values of n, we show that it performs about twice less checks than the `QuickXplain` algorithm (50% less checks).
- in the case of a unique minimal set of size n, for large values of n, we show that it performs about $\log_2(n)$ less checks than the `DC` algorithm.
- 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).
- it can trivially be adapted for the case we are looking for a *preferred* subset.

Proofs

Property 1 *In the case of a unique minimal subset of size m, the probability $P\big(\pi(X^*, j) = k\big)$ that the position of the j^{th} element of the minimal subset is k, denoted $p_j(k)$ is given by:*

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

Proof: The total number of realizations for positioning the m elements of the unique minimal subset in U is $\binom{n}{m}$.

From these realizations, the ones that put the j^{th} element of the subset at position k are $\binom{k-1}{j-1}\binom{n-k}{m-j}$. The first term is the number of ways to select the $j-1$ elements before the j^th one and put them before position k while the second term is the number of ways to select the $m-j$ elements after the j^th one and put them after position k. ■

Property 2 *In the case of a unique minimal subset of size m, the probability that the distance $\pi(X^*, j+1) - \pi(X^*, j)$ between two consecutive elements is k does not depend on j and is the same as the probability that the first element of the minimal subset is at position k ($p_1(k)$):*
$\forall j \in [1, m), P\big(\pi(X^*, j+1) - \pi(X^*, j) = k\big) = p_1(k)$
With:

$$p_1(k) = \binom{n-k}{m-1}/\binom{n}{m}$$

Proof: The total number of realizations for positioning the m elements of the unique minimal subset in U is $\binom{n}{m}$.

Let $j \in [1, m)$. The number of realizations $r(j, k, i)$ for which the distance between the j^{th} and $j+1^{th}$ element of the subset is k and the j^{th} element is at position i is:

$$r(j, k, i) = \binom{i-1}{j-1}\binom{n-(i+k)}{m-(j+1)}$$

The first factor is the number of ways to distribute the $j-1$ elements before the j^{th} element in the interval $[1, i-1]$ and the second factor is the number of ways to distribute the $m-(j+1)$ elements of the subset after the $j+1^{th}$ element in the interval $[i+k+1, n]$.

Thus, the total number of realizations $r(j, k)$ for which the distance between the j^{th} and $j+1^{th}$ element of the subset is k is a sum over all possible positions i:

$$r(j, k) = \sum_i r(j, k, i)$$

Using the following change of variables: $u = i-1$, $v = j-1$, $\alpha = m-2$, $\beta = n-k-1$, we obtain:

$$r(j, k) = \sum_u \binom{u}{v}\binom{\beta-u}{\alpha-v}$$

Now, we can use the following identity for binomial coefficients:

$$\forall v \in [0, \alpha], \sum_u \binom{u}{v}\binom{\beta-u}{\alpha-v} = \binom{\beta+1}{\alpha+1}$$

It gives:

$$r(j, k) = \binom{n-k}{m-1}$$

■

Property 3 *In the case of disjoint minimal subsets, the probability that the distance $\pi(X^*, j+1) - \pi(X^*, j)$ between two consecutive elements is k does not depend on j and is a non-increasing function of k.*

Proof: Let's take a realization for the minimal subset sizes m_i, $i \in [1, c]$ so that the minimal subsets are $\{X_1, ..., X_c\}$ with $|X_i| = m_i$ and assume $X^* = X_c$. It is clear that if the property holds for each individual realization of subset sizes, it also holds in general.

For $l \in [1, n]$, the number of realizations $r_l(j, k, i)$ for which: (1) the first element of X^* is at position l and (2) the j^{th} element of X^* is at position i and (3) the distance between the j^{th} and $j + 1^{th}$ element of X^* is k is:

$$r_l(j, k, i) = \binom{i-l}{j-1}\binom{n-(i+k)}{m-(j+1)} A_l$$

The first factor is the number of ways to distribute the $j-1$ elements before the j^{th} element in the interval $[l, i-1]$. The second factor is the number of ways to distribute the $m-(j+1)$ elements of the subset after the $j+1^{th}$ element in the interval $[i+k+1, n]$. Finally, A_l is the number of ways to distribute the elements of $X_1 \cup ... \cup X_{c-1}$ such that at least one element of each subset is before position l given that there are $l-1$ free positions before l and $n-l-m_c+1$ free positions after l. The key point is that because the minimal subsets are disjoint, A_l does not depend on i, j or k.

Thus, the number of realizations such that the first element of X^* is at position l and the distance between the j^{th} and $j+1^{th}$ element of X^* is k is:

$$r_l(j, k) = \sum_i r_l(j, k, i)$$

Using the following change of variables: $u = i - l$, $v = j - 1$, $\alpha = m - 2$, $\beta = n - k - l$, we obtain:

$$r_l(j, k) = A_l \sum_u \binom{u}{v}\binom{\beta-u}{\alpha-v}$$

Now, we can use the following identity for binomial coefficients:

$$\forall v \in [0, \alpha], \sum_u \binom{u}{v}\binom{\beta-u}{\alpha-v} = \binom{\beta+1}{\alpha+1}$$

It gives:

$$r_l(j, k) = A_l \binom{n-l-k+1}{m-1}$$

The total number of realizations such that the distance between the j^{th} and $j+1^{th}$ element of X^* is k is:

$$\sum_l r_l(j, k) = \sum_l A_l \binom{n-l-k+1}{m-1}$$

This number clearly does not depend on j and is a non-increasing function of k. ∎

Property 4 *Algorithm 1 is correct and returns the unique minimal subset $X^* \in \mathcal{M}$ that is maximal with respect to the total order $\prec$.*

Proof: We need to show that:

1. the algorithm terminates
2. the subset X returned by Algorithm 1 is minimal
3. if X and Y are two minimal subsets such that $X \prec Y$ then X won't be produced by the algorithm

To facilitate the proof we slightly rewrite the algorithm to identify the different iterations s as well as some relevant subsets visited during the iterations (see Algorithm 10).

Algorithm 10 FindMinimalSubset(U, $\mathcal{P}$)

```
Require: $\mathcal{P}(U)$
 1: Shuffle($U$)
 2: $s \leftarrow 0$                                   ▷ Current step
 3: $X_0 \leftarrow \emptyset$
 4: $i_0 \leftarrow 0$  ▷ $i_s$: index of element added to $X_{s-1}$ at step $s$
 5: repeat
 6:     $s \leftarrow s + 1$
 7:     $i_s \leftarrow \mathrm{FindNext}(X_{s-1}, U, i_{s-1} + 1, \mathcal{P})$
 8:     $X_s \leftarrow X_{s-1} \cup \{U[i_s]\}$
 9:     $V_s \leftarrow X_{s-1} \cup U_{i_s+1\rightarrow}$
10:     $W_s \leftarrow X_{s-1} \cup U_{i_s\rightarrow}$ $(= X_s \cup U_{i_s+1\rightarrow})$
11: until $(i_s = n) \vee \mathcal{P}(X_s)$
12: return $X_s$
```

We recap that procedure $\mathrm{FindNext}(X, U, i, \mathcal{P})$ is in charge of finding and returning the largest index j ($i \leq j \leq n$) such that $\mathcal{P}(X \cup U_{j\rightarrow})$. This procedure is well defined and guaranteed to return an index as soon as $\mathcal{P}(X \cup U_{i\rightarrow})$.

Termination.

We will first prove by recursion the existence of i_s computed by FindNext and the fact $\mathcal{P}(W_s)$.

- For iteration $n = 1$, $\mathrm{FindNext}(\emptyset, U, 1, \mathcal{P})$ works on the full set $U_{1\rightarrow} = U$ so, given that $\mathcal{P}(U)$, there exists a largest index i_1, $0 < i_1$ such that $\mathcal{P}(\emptyset \cup U_{i_1\rightarrow})$. We have $W_1 = \emptyset \cup U_{i_1\rightarrow}$ thus, $\mathcal{P}(W_1)$.
- Assuming i_{s-1} exists and $\mathcal{P}(W_{s-1})$, $\mathrm{FindNext}(X_{s-1}, U, i_{s-1} + 1, \mathcal{P})$ works on the set $X_{s-1} \cup U_{i_{s-1}+1\rightarrow} = W_{s-1}$. As $\mathcal{P}(W_{s-1})$, there exists a largest index i_s, $i_{s-1} < i_s$ such that $\mathcal{P}(X_{s-1} \cup U_{i_s\rightarrow})$. We precisely define W_s as $W_s = X_{s-1} \cup U_{i_s\rightarrow}$ thus, $\mathcal{P}(W_s)$.

As for all s, i_s is well defined and $i_{s-1} < i_s$ this shows that $card(X_s) = s$ and that the algorithm necessarily terminates with less than n iterations.

Note that the termination condition $i_s = n$ simply avoids checking $\mathcal{P}(X_s)$ in a situation when X_s clearly satisfies the property because from the definition of FindNext we have $\mathcal{P}(X_{s-1} \cup U_{i_n\rightarrow})$ and thus $\mathcal{P}(X_s)$ because $X_s = X_{s-1} \cup \{U[i_s]\}$.

Minimality of the returned subset.

Let s^* denote the number of iterations until termination. We necessarily have $\mathcal{P}(X_{s^*})$ as this is the stopping condition. Let's show that X_{s^*} is minimal. Suppose there exists a subset $X \subset X_{s^*}$ such that $\mathcal{P}(X)$. Let i denote the smaller index i such that $u_i \in X_{s^*} \setminus X$ and s the iteration when u_i was added to X_{s^*} ($i_s = i$). Let $V_s = X_{s-1} \cup U_{i_s+1\rightarrow}$.

Because until index i the sets X and X_{s^*} are the same, we have $X \subseteq V_s$. As $\mathcal{P}(X)$, we should also have $\mathcal{P}(V_s)$ but this is inconsistent with the fact i_s is the largest index such that $\mathcal{P}(X_{s-1} \cup U_{i_s \rightarrow})$ and thus $V_s = X_{s-1} \cup U_{i_s+1\rightarrow}$ does not satisfy $\mathcal{P}$.

Maximality with respect to $\prec$.

Let X and Y, $X \neq Y$ be two minimal subsets such that $X \prec Y$. We suppose that X is the subset returned by Algorithm 5. Let $j \in [1, n]$ be the index such that:

$$u_j \in X, u_j \notin Y, \forall i < j, u_i \in X \Leftrightarrow u_i \in Y$$

Let suppose u_j was added to subset X in iteration s of the algorithm. At iteration $s-1$ we had: $X_{s-1} = \{u_i \in X | i < j\}$. At iteration s, as j was selected as the largest index such that $\mathcal{P}(X_{s-1} \cup U_{j\rightarrow})$, it means $\neg\mathcal{P}(X_{s-1} \cup U_{j+1\rightarrow})$. But we have $Y \subseteq X_{s-1} \cup U_{j+1\rightarrow}$ and this is inconsistent with $\mathcal{P}(Y)$. ■

Property 5 *If $\pi(X^*, |X^*|) < n$, the naive algorithm performs $\pi(X^*, |X^*|) + |X^*|$ property checks. Otherwise, if $\pi(X^*, |X^*|) = n$, the naive algorithm performs $n + |X^*| - 2$ property checks.*

Proof: Let $l = \pi(X^*, |X^*|)$ and $k = |X^*|$.

If $l < n$, algorithm 2 will perform k checks in line 7 of algorithm 1 (all index i are such that $i < n$) and l checks in line 4 of algorithm 2.

If $l = n$, the last check in line 7 of algorithm 1 as well as the last check in line 4 of algorithm 2 are not performed thus the total number of checks is $k + l - 2$. ■

Property 7 *In case of a unique minimal set of size* 1*, the average complexity of the* QuickXplain *algorithm is* $3/2\ \log_2(n)$.

Proof: We suppose n is a power of 2: $n = 2^m$. We first compute $c(k, n)$ the number of property checks performed by algorithm QuickXplain to find a minimal subset of size 1 at position $k \in [0, n)$. We clearly have $c(0, 1) = 0$. If $k \geq n/2$, the minimal subset belongs to C_1 so the first call to DoQuickXplain for the first split to compute Δ_2 (left branch) will perform a single check and return $\emptyset$. The call to DoQuickXplain to compute Δ_1 (right branch) will require $c(k - n/2, n/2)$ checks. If $k < n/2$, the left branch will require $1 + c(k, n/2)$ checks whereas the right one will only require 1 check. So we have:

$$c(k, n) = \begin{cases} 1 + c(k - n/2, n/2) & \text{if } k \geq n/2 \\ 2 + c(k, n/2) & \text{if } k < n/2 \end{cases}$$

For computing the average over all positions k, we compute $s(n) = \sum_{k=0}^{n-1} c(k, n)$.

$$\begin{aligned} s(n) &= \sum_{k=0}^{n/2-1} c(k, n) + \sum_{k=n/2}^{n-1} c(k, n) \\ &= 2(\frac{n}{2}) + \sum_{k=0}^{n/2-1} c(k, \frac{n}{2}) + (\frac{n}{2}) + \sum_{k=n/2}^{n-1} c(k - \frac{n}{2}, \frac{n}{2}) \\ &= 3(\frac{n}{2}) + 2\, s(\frac{n}{2}) \end{aligned}$$

So we have $s(n) = 3/2\, n\ \log_2(n)$ and the average number of checks is $3/2\ \log_2(n)$. ■

References

Chinneck, J. W. 1997. Finding a useful subset of constraints for analysis in an infeasible linear program. *INFORMS Journal on Computing* 9:164–174.

Chinneck, J. W. 2007. *Feasibility and Infeasibility in Optimization: Algorithms and Computational Methods*. Springer.

Demaret, J., and Gareet, A. 1977. Sum of exponential random variables. *Electronics and Communication (AE)* 31:445–448.

Hemery, F.; Lecoutre, C.; Sais, L.; and Boussemart, F. 2006. Extracting MUCs from constraint networks. In *Proc. 17th European Conference on Artificial Intelligence (ECAI'06)*.

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

Jussien, N., and Lhomme, O. 2002. Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence* 139(1):21–45.

Marques Silva, J. P., and Sakallah, K. A. 1996. Conflict analysis in search algorithms for satisfiability. In *Proceedings of the 8th International Conference on Tools with Artificial Intelligence (ICTAI'96)*.