Theory Comput Syst (2009) 44: 269–288
DOI 10.1007/s00224-008-9113-2

Pictures from Mongolia. Extracting the Top Elements from a Partially Ordered Set

Paolo Boldi · Flavio Chierichetti · Sebastiano Vigna

Published online: 30 April 2008

Abstract You are back from that very long, marvellous journey. You have a thousand pictures, but your friends and relatives will stand just a few dozens. Choosing is a painful process, in particular when you cannot decide between the silent vastness of that desert and the idyllic picture of that tranquil, majestic lake. We are going to help.

Keywords Partial orders · Preorders · Algorithms · Ranking

1 Pictures, Pictures and More Pictures

In the summer of 2004, Paolo, Sebastiano and four more friends took part to a long journey in Mongolia, with a local guide, a driver and an old Uaz van. Five of the participants were fond of taking pictures, and Mongolia met their match: long rivers flowing through luxuriant meadows, high peaks covered with pine trees, wild horses running free, children laughing at our passage, camels in sandy deserts .... On our trip back to Italy, when we were in Moscow for a short stop, we started to put together all the pictures we had been taking for three weeks and discovered that we had about six hundred of them! (We were using mainly digital cameras and we had an old light laptop with us, so we had essentially no limit to the number of pictures we could take.)

The number was scary, not only for us but also, and importantly, for all of our friends. If you ever happened to be invited to one of those just-after-the-holidays

A preliminary version of this paper appeared in [1].

P. Boldi (✉) · S. Vigna
Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, via Comelico 39/41, 20135 Milano MI, Italy
e-mail: boldi@dsi.unimi.it

F. Chierichetti
Dipartimento di Informatica, Sapienza Università di Roma, via Salaria 113, 00198 Roma, Italy

dinners at one of your friend's, you know what we mean: at the end of the dinner, no doubt your friend will "propose" to look at the pictures they have been taking during their holidays in Guatemala, and those pictures turn out to be three hundred, and most of them about a special tree growing in the jungle, and ....

Yet, choosing the best representatives from a set of objects is not an easy task; even the very nature of preference is questionable, and has been largely debated by psychometricians and economists (see, e.g., [2]). In particular, a much argued point is whether personal preference is a transitive relation or not; even if some experiments actually prove that preference may indeed not be transitive, it is commonly agreed that intransitive preference lead to irrational behaviour. For this reason, it is by now accepted that preference can be modelled as a *preorder* (a.k.a. *quasiorder*), that is, a transitive, reflexive relation that is not required to be antisymmetric; for sake of simplicity, in this paper we shall mainly focus our attention on *partial orders*, although all our algorithms can be easily extended to preorders (we discuss this issue in Sect. 6).

So, in an abstract sense, we shall consider the problem of choosing the "best" t elements from a set X of n elements subject to a given partial order; the exact notion of "best elements" will be discussed shortly.[1] In our intended application, X is the set of all pictures, whereas t is the number of pictures we are going to put in our album, and the order reflects the preference of some given, rational subject.

An algorithm for this problem will present a sequence of pairs of pictures to the subject and ask her to choose which of the two pictures she likes best, or if she is completely indifferent about the two pictures. The computational cost we are going to take into consideration is the number of comparisons the subject is requested to perform; notice, in particular, that we shall not be considering the number of instructions executed or the space occupied by the data (even though all our algorithms have a polynomial running time).

Apart for pictures, the problem may find more interesting applications to other areas; for example, our algorithms suggest a new way to perform rank aggregation in information-retrieval tasks. Rank aggregation is the problem of combining ranking results from various sources; in the context of the Web, the main applications include building meta-search engines, combining ranking functions, selecting documents based on multiple criteria, and improving search precision through word associations. Here, by rank we mean an assignment of a relevance score to each candidate. Traditional approaches to rank aggregation [4, 6] assume that ranks be aggregated in a single rank value that satisfies suitable properties. Instead, we propose to produce a partial order by intersection, and then identify suitable "best elements" of the partial order obtained in this way.

For more information on the possible applications of the problem we discuss, we direct the reader to a very recent, independently developed, paper by Daskalakis, Karp, Mossel, Riesenfeld and Verbin [3] that contains some results similar in nature to the ones discussed here.

[1] Note that this problem is significantly different from the complete reconstruction of the partial order, a task whose complexity has been largely studied in the literature, starting from the seminal paper of Faigle and Turán [7].

2 Basic Setup

A *partial order* on a set X is a binary relation $\preceq$ on X that is reflexive ($x \preceq x$ for all $x \in X$), antisymmetric ($x \preceq y$ and $y \preceq x$ imply $x = y$) and transitive (if $x \preceq y$ and $y \preceq z$ then $x \preceq z$); a *poset* is a set endowed with a partial order; in this paper, all posets will be finite. We let $x \prec y$ mean $x \preceq y$ and $x \neq y$.

Two elements $x, y \in X$ such that either $x \preceq y$ or $y \preceq x$ are called *comparable*, otherwise they are called *incomparable*. A set of pairwise comparable (incomparable, resp.) elements is called a *chain* (*antichain*, resp.). The *width* of a poset is the maximum cardinality of an antichain.

Antichains are "no-clue" sets: we have no freakin' idea as to which picture we would prefer out of the set. Chains are "I-got-all-clues" sets: we have a precise idea about which picture is better out of any pair.

We say that x *covers* y iff $y \prec x$ and, for every z, $y \preceq z \preceq x$ implies either $y = z$ or $z = x$. Intuitively, x is "just better" than y.

A *maximal* (*minimal,* resp.) *element* in a subset Y of a poset $(X, \preceq)$ is any element $y \in Y$ such that $y \preceq y' \in Y$ ($Y \ni y' \preceq y$, resp.) implies $y = y'$. Maximal elements are pictures for which we cannot find a better one (but there might be many incomparable maximal elements).

The notation $x{\downarrow}$ denotes the set of all elements smaller than or equal to x.

3 Setting Our Goal

We wish to give a sound and meaningful definition of what are the "best" elements of a given poset; this notion is quite obvious in the case of a total order, but turns out to be much subtler when partial orders are involved.

As a first attempt, an *upper set* seems a good idea. Given a poset $(X, \preceq)$, a set $Y \subseteq X$ is an *upper set* iff $y \preceq z$ and $y \in Y$ imply $z \in Y$. Indeed, if we choose an upper set we are sure that no picture more valuable than the ones chosen has been missed.

There is however a significant problem with this definition when we try to give an intuitive, user-related meaning to incomparability. Our friends coming from Mongolia have pictures from Lake Hovsgol and also from the Gobi Desert.[2] It is reasonable to assume that it is very difficult to define preferences between pictures containing just sand and pictures containing just water, so every picture from Gobi could be incomparable with every picture from Lake Hovsgol. But in this case the pictures from Gobi would be an upper set, and a subset of Gobi pictures that is also an upper set could provide a valid answer.

In other words, if we interpret incomparability as "different setting" or "different topic" or "different subject", we would like, say, to have at least the best picture for every setting, or topic, or subject. Once we have the best pictures, if we still have room we would like to get the *second* best pictures, and so on: intuitively, we would like to *peel* the top of the poset by iteratively choosing the best pictures available. This idea suggests a more stringent definition:

[2]Incidentally, "Gobi" in Mongolian means "desert", which makes the expression "Gobi Desert" bizarre at best.

Definition 1 Let $(X, \preceq)$ be a poset, and define a sequence $X_0, X_1, X_2, \ldots$ of disjoint subsets of X, called *levels*, where X_t is the set of maximal elements in $X \setminus (X_0 \cup X_1 \cup \cdots \cup X_{t-1})$ (the sequence is ultimately $\emptyset$). Let us define a new order $\sqsubseteq$ on X by setting $X_0 \sqsupseteq X_1 \sqsupseteq X_2 \sqsupseteq \cdots$ (elements within each X_i are incomparable). A *top set* is an upper set w.r.t. $\sqsupseteq$; a top set of size t is also called a t-top set.

More explicitly, top sets are exactly sets of the form $X_0 \cup X_1 \cup \cdots \cup X_{i-1} \cup X'_i$, where $X'_i \subseteq X_i$.

Note that $\sqsubseteq$ is an extension of $\preceq$ (if $x \preceq y$ and $y \in X_i$ necessarily $x \in X_j$ for some $j > i$): as a consequence, all top sets are upper sets (for $\preceq$). The extension is in general proper: if we consider the set $X = \{x, y, x', y'\}$ ordered by $x \preceq x'$, $y \preceq y'$, we have that additionally $x \sqsubseteq y'$.

At this point, we are in a position to give a more precise definition of our problem: we are given a poset $(X, \preceq)$ of n elements, and an integer $t \leq n$, and we must output a t-top set;[3] the poset is available only implicitly, in the sense that we do not know $\preceq$ in advance, but our algorithm can use an oracle that, given two distinct elements $x, y \in X$, answers either "$x \prec y$", or "$x \succ y$" or "x and y are incomparable": such a call to the oracle is denoted by $x :: y$ and named *comparison* (or query), and we want our algorithm to perform as few comparisons as possible.

To measure the loss w.r.t. a hypothetical optimal algorithm, we will say that an algorithm is θ-*slow* if it never makes more than

$$\theta \cdot \min_{\mathcal{A}} \max_{|P|=n} q(\mathcal{A}, P, t)$$

queries to return the t-top set of a poset of size n, where $q(\mathcal{A}, P, t)$ is the number of queries performed by algorithm $\mathcal{A}$ on the poset P of size n to return a t-top set.

Of course, there is a trivial algorithm that performs $\binom{n}{2}$ comparisons and rebuilds the whole poset, and in the worst case we cannot hope to perform a smaller number of comparisons: every algorithm must perform at least $\binom{n}{2}$ comparison on a poset made of n incomparable elements to output a top set of $n-1$ elements. Indeed, if x and y are never compared and, say, y is not in the output, the algorithm would fail for a poset in which $x \prec y$ (note that for this proof to work the choice to use top sets vs. upper sets is essential). This consideration can be generalised to any $t < n$:

Proposition 1 *Every correct algorithm must, in the worst case, perform at least*

$$\tfrac{1}{2}t(2n - t - 1) \geq \tfrac{1}{2}tn$$

comparisons $(0 < t < n)$ *to determine a* t*-top set of a* n*-elements poset.*

Proof Assume that the algorithm outputs a t-top set T of a poset of n elements without comparing every element of T with every element of P; then, there is some $x \in T$ and some $y \in P$ that have not been compared, and the algorithm would fail on a poset in which the only comparable pair is $x \prec y$. Indeed, in that case x should

[3] It may be worth noting that almost all the algorithms we are going to present can be modified so to output not only a t-top set T, but also, for each $x \in T$, the level i to which x belongs (i.e., such that $x \in X_i$).

not be in the output, as $t < n$ and there are $n-1$ maximal elements. Hence, the algorithm performed at least $\binom{t}{2} + t(n-t)$ comparisons, which is the left-hand side of the inequality in the statement (the right-hand side follows from $t < n$). □

We note in passing that a similar lower bound holds for upper sets. The previous proof can be easily amended to provide a $t(n-t)$ lower bound on the number of comparisons (if you output t elements and you did not check against some of the remaining $n-t$, your output might not be an upper set). On the other hand, finding an upper set in $O(t(n-t))$ comparisons is trivial: if $t \leq n/2$ just look in a brute-force manner for a maximal element ($n-1$ comparisons), then find another maximal element in the remaining $n-1$ elements ($n-2$ comparisons) and so on. This requires $(n-1)+(n-2)+\cdots+(n-t) = tn - t(t+1)/2 \leq \frac{3}{2}t(n-t)$ comparisons. If $t > n/2$ we can work backwards, eliminating iteratively minimal elements, and returning the remaining ones.

Maybe surprisingly, an absolutely analogous approach yields a 2-slow algorithm for top sets. The only difference is that once we find a maximal element m, we start searching for elements incomparable with m in the poset, and we output them. Some care, however, must be exercised, as once an incomparable element has been output, all smaller elements are not good candidates for the output, even if they are incomparable with m. So we must keep track of which elements of the poset should be checked for incomparability with m, and update them each time we output a new incomparable element. The details are given in Algorithm 1.

Algorithm 1 (A 2-slow algorithm)

top(P: a poset, t: an integer)

```
1:  C ← ∅ {Candidate set: elements of the level to be peeled are to be found in here.}
2:  L ← P {Elements left for the next round.}
3:  for t times do
4:     if C = ∅ then
5:        {We prepare to peel a new level}
6:        C ← L
7:        L ← ∅
8:     end if
9:     set m to a maximal element of C
10:    output m
11:    add m↓ minus m to L {Keep all elements smaller than m for the next round
       ...}
12:    remove m↓ from C {...but remove m and all smaller elements from the can-
       didates.}
13: end for
```

Theorem 2 *Algorithm 1 outputs a t-top set.*

Proof Let O be the set of already output elements. We show that at the start of the loop, $C \cup L = P \setminus O$, and the maximal elements of C are the maximal elements of

$P \setminus (O \cup L)$. Indeed, the execution of the if statement (i.e., when C is empty, hence $L = P \setminus O$) does not change these facts. When we output m we make it disappear from C and appear in O, preserving the equation above (the elements strictly smaller than m are transferred from C to L). Finally, since we remove $m{\downarrow}$ from C, m also disappears from the set of maximal elements of C, and no new maximal element appears. This implies that the second property is preserved, too.

To conclude the proof, we note that just after the i-th execution of the if statement, C will contain the entire poset P minus the first i levels. This is certainly true at the first reassignment, and thus the maximal elements of C are exactly the first level of P. Since we remove them one by one, by the time we execute the second reassignment we will have output exactly the first level, so we will now assign to C the entire poset minus the first level, and so on. □

Theorem 3 *Algorithm 1 requires no more than* $t(2n - t - 1)$ *comparisons to output a* t*-top set of a poset with* n *elements. Thus, it is* 2*-slow.*

Proof The algorithm performs some comparisons only when finding maximal elements, and when computing $x{\downarrow}$. In both cases at most $|C| - 1$ comparisons are needed, using a brute-force approach. Note that at the start of the i-th iteration of the loop $|C| + |L| = n - i$, as several elements are moved between C and L, but exactly one element is output (and removed from C) at each iteration, so in particular $|C| \leq n - i$. The total number of comparison is then bounded by $2((n-1) + \cdots + (n-t)) = 2tn - t(t+1) = t(2n - t - 1)$. The last claim then follows immediately from Proposition 1. □

4 Small-Width Posets

The reader might have noticed that our lower bound uses very wide posets (i.e., with width $\Theta(n)$). It is thus possible that by limiting the width we can work around the lower bound, getting a better algorithm for small-width posets. Indeed, we expect that settings, topics or subjects should be significantly fewer than the number of pictures, or it would be very difficult to choose a small best subset.

Looking into Algorithm 1, it is clear that we are somehow wasting our time by scanning for one maximal element at a time. A more efficient strategy could be building the set of maximal elements in one shot, peeling them, building the new set of maximal elements, and so on.

There are two difficulties with this approach: first of all, if we need a very small top set we could make much more queries than necessary, as the set of maximal elements could be significantly larger than the required top set. Second, rebuilding the set of maximal elements after each peeling could be expensive.

There is not much we can do about the first problem, but our next algorithm (Algorithm 2) tries to address the second one. If we divide P into two subsets and compute the respective sets of maximal elements, the union of these two sets will certainly contain all maximal elements of P (plus some spurious elements). We can apply this reasoning recursively, building the set of maximal elements incrementally. To avoid

an expensive recomputation after each peeling, we will arrange the elements of the poset P arbitrarily on the leaves of a complete binary tree T. The binary tree then induces naturally a hierarchical partition of the elements of P—just look at the leaves of each node of given depth. We will keep track of the maximal elements of each subset of the partition, by suitably labelling each node of the tree. Initially, we will find at the root the set M of maximal elements of P, which we can output. Then, we will remove the elements of M from all labels in the tree, and recompute the new labels. The process will continue until we have output enough elements.

Note that the width of the poset does not appear explicitly in the description of the algorithm. Indeed, it will surface only during the analysis, when we shall put to good use the fact that the labels on each node cannot be of cardinality larger than the poset width.

In what follows we shall tacitly assume that the algorithm uses some data structure to keep track of the comparisons that have already been performed; in other words, for every pair $x, y \in P$, we are able to tell whether x, y have ever been compared and, if so, the result of the comparison. This assumption makes it possible to bound the number of comparisons using just the *number of pairs ever compared*.

Algorithm 2 (An algorithm for small-width posets)

top(t: an integer)

```
let T be a complete binary tree labelled on subsets of P and with n leaves
label each leaf of T with a singleton containing a distinct element of P
label the remaining nodes with ∅
while t > 0 do
    completeLabelling(T)
    let A be the label of the root of T
    output u ← min{t, |A|} elements from A
    t ← t − u
    remove the elements of A from all the labels
end while
```

completeLabelling(T: a binary tree)

```
recursively consider all non-leaf nodes v of T, bottom-up
let z_0, z_1 be the two children of v
for k ← 0, 1 do
    for x a label of z_k do
        if no label of z_{1−k} is greater than x add x to the label of v
    end for
end for
```

Theorem 4 *Algorithm 2 is correct.*

Proof To prove the statement, it is sufficient to show that after the i-th call to *completeLabelling* the label of the root is the i-th level of the input poset P. Given a

node v of T, we define the v-dominated poset $v{\Downarrow}$ as the subset of elements of P contained in leaves dominated by v with the order inherited from P. It is immediate to show by induction that if each node v of T is labelled by a (possibly empty) set of maximal elements of $v{\Downarrow}$, then after a call to *completeLabelling* each node will be labelled by the set of *all* maximal elements of $v{\Downarrow}$. Indeed, during the recomputation of the labels we compute the maximal elements of the union of the labels of the children. But the labels of the children contain (by induction) all maximal elements of the respective dominated posets, and all maximal elements of $v{\Downarrow}$ must appear in the labels of the children of v (if we partition a partial order in two disjoint subsets, all maximal elements of the partial order are still maximal elements of the subset they belong).

Thus, the label of the root after the first iteration is the first level of P. Then, either the algorithm stops, or we remove the first level and call again *completeLabelling*. Since now the tree contains just P minus its first level, and all labels are still maximal, the label of the root will be the second level, and so on. □

Theorem 5 *Algorithm 2 finds a t-top set of a poset with n elements and width at most w using*

$$\frac{3}{2}wn + wt\left(\lfloor \log n \rfloor - \lceil \log w \rceil\right)$$

comparisons.

Proof We split the estimate of the number of comparisons in two parts. Define the *depth* of a node in the standard way (the root has depth 0, and the children of nodes at depth d have depth $d+1$). We shall call the nodes of depth smaller than $\lfloor \log n \rfloor - \lceil \log w \rceil$ *interesting*, and the remaining nodes *uninteresting*. Note that the depth of the deepest node performing comparisons is $\lceil \log(n+1) \rceil - 2$, and that

$$\lceil \log(n+1) \rceil - \lfloor \log n \rfloor = 1 \quad \text{for all integers } n.$$

Since we never repeat a comparison, all comparisons that could be ever performed on uninteresting nodes are very easily bounded:

$$\sum_{i=\lfloor \log n \rfloor - \lceil \log w \rceil}^{\lceil \log(n+1) \rceil - 2} 2^i \left(2^{\lceil \log(n+1) \rceil - 2 - i}\right)^2 \leq n 2^{\lceil \log w \rceil - 1}.$$

We now focus the rest of the proof on the interesting nodes. The number of elements contained in each child of an interesting node can be just bounded by w, so in principle each time we need to update the labels of an interesting node we might need w^2 comparisons. This very rough estimate can be significantly improved noting that we cannot increase indefinitely the number of elements in a child (as it is bounded by w). Indeed, each time we add new elements in a node, this generates some new comparisons in its parent. Nonetheless, as long as we *add* elements, the number of overall comparisons performed by the parent approaches w^2, but can never get past it.

So if we estimate an initial cost

$$\sum_{i=0}^{\lfloor \log n \rfloor - \lceil \log w \rceil - 1} 2^i w^2 \leq 2^{\lfloor \log n \rfloor - \lceil \log w \rceil} w^2 \leq n 2^{-\lceil \log w \rceil} w^2,$$

this estimate will cover all *additions*, as long as elements are never deleted.

Before considering the deletion of elements, let us bound the sum $n2^{\lceil \log w \rceil - 1} + n2^{-\lceil \log w \rceil} w^2$:

$$\begin{aligned} &n2^{\lceil \log w \rceil - 1} + n2^{-\lceil \log w \rceil} w^2 \\ &\quad = n\bigl(2^{\lceil \log w \rceil - 1} + w^2 2^{-\lceil \log w \rceil}\bigr) = nw\left(\frac{1}{w} 2^{\lceil \log w \rceil - 1} + w 2^{-\lceil \log w \rceil}\right) \\ &\quad = nw\bigl(2^{\lceil \log w \rceil - \log w - 1} + 2^{\log w - \lceil \log w \rceil}\bigr). \end{aligned}$$

We let $x = \lceil \log w \rceil - \log w$ (note that $x \in [0..1)$) so the previous expression becomes $nw(2^{x-1} + 2^{-x})$. The function $g(x) = 2^{x-1} + 2^{-x}$ decreases for $x < \frac{1}{2}$ and increases for $x > \frac{1}{2}$. Thus, in the interval $[0..1)$, $g(x) \leq \max\{g(0), g(1)\} = \frac{3}{2}$. We conclude that

$$n2^{\lceil \log w \rceil - 1} + n2^{-\lceil \log w \rceil} w^2 \leq \frac{3}{2} nw.$$

Now, what happens when elements are deleted? In this case, the deleted element can be replaced by a new one, and its cost is not included in our previous estimate. So we must fix our bound by adding w comparisons for each node in which a deletion happens.

Let us make the above considerations formal. Each interesting node v has a *bonus* of w^2 comparisons included in the estimate above. Let v ambiguously denote the number of labels of a node v *before* a call to *completeLabelling* (so in particular $v = 0$ for all interesting v at the first iteration), and v^* the number of labels of v after the call. If ℓ and r are the left and right child of v we show that, provided $w^2 - \ell r$ bonus comparisons are available before the call, $w^2 - \ell^* r^*$ are still available afterwards.

When we call *completeLabelling*, the labels of each node v must be updated. The update involves comparing new elements that appeared in the children of v; at most $(\ell^* - \ell)r + (r^* - r)\ell + (\ell^* - \ell)(r^* - r)$ comparison are needed to update v. But since $w^2 - \ell r - (\ell^* - \ell)r - (r^* - r)\ell - (\ell^* - \ell)(r^* - r) = w^2 - \ell^* r^*$ the invariant is preserved. The invariant is trivially true before the first call, so we conclude that the costs of *completeLabelling* are entirely covered by the bonus.

Finally, when we remove elements from the tree, each removed element potentially alters $\lfloor \log n \rfloor - \lceil \log w \rceil$ interesting nodes, reducing by 1 the number of its labels. Thus, to keep the invariant true we might need to cover some costs: with the same notation as above, if, for instance, the set of labels of ℓ becomes smaller we need to compensate $r \leq w$ comparisons to keep the invariant true for node v (symmetrically for r). Thus, removing an element requires patching the bonus by at most $w(\lfloor \log n \rfloor - \lceil \log w \rceil)$ additional comparisons.

All in all, emitting t elements will require a fixed cost of $\frac{3}{2}nw$ comparisons, plus at most $w(\lfloor \log n \rfloor - \lceil \log w \rceil)$ comparisons for each deleted element; since the deleted elements are bounded by t the result follows easily. □

Note that if $w = o(t)$ Algorithm 2 is advantageous over Algorithm 1, as

$$wt \log \frac{n}{w} = nt \left(\log \frac{n}{w} \right) \Big/ \left(\frac{n}{w} \right) = o(nt).$$

The opposite happens if $t = o(w)$. To see that this is not an artifact of the analysis of the algorithm, just note that on a poset formed by w chains, each of height $\lceil n/w \rceil$, if we distribute each level (which has width w) on a set of adjacent leaves *all* possible comparisons on uninteresting nodes will be performed during the first call to *completeLabelling*, so the algorithm actually requires $\Omega(wn) = \omega(tn)$ comparisons if $t > 0$.

As we will now show, the difference between the case $t = o(w)$ and the case $w = o(t)$ is not an artifact of our analysis or of our algorithms, but rather an intrinsic feature of the problem.

Theorem 6 *Every correct algorithm must, in the worst case, perform at least*

$$\frac{w}{2}(n - t)$$

queries to get a t-top set of a n-elements poset of width $w \leq t$.

Proof Suppose that a certain algorithm is trying to determine the t-top set of a n-elements poset of width w; we build the poset adversarially using a w-colourable graph G with n vertices $\{0, 1, \ldots, n-1\}$ (each vertex represents an element of the poset). At the beginning, the graph G contains $n - t$ isolated vertices $\{0, 1, \ldots, n - t - 1\}$, $\lfloor t/w \rfloor$ cliques containing w vertices each and possibly (if w does not divide t) a clique containing the last t mod w elements; let us denote by U the elements of this last clique.

Every time the algorithm queries $x :: y$, we decide the answer as follows:

- if one element belongs to U and the other does not, we answer that the element in U is larger;
- otherwise, if x and y are adjacent or can be made adjacent while still keeping the graph w-colourable, we answer "incomparable" and add the edge $\{x, y\}$, if it is not already in the graph;
- otherwise, we answer $\prec$ or $\succ$ depending on whether $x < y$ or $x > y$; note that in such a case every w-colouring assigns the same colour to x and y: we say that x and y are *tainted*; when a vertex becomes tainted, it has degree at least $w - 1$, and it obviously remains tainted thereafter.

At any time, you can build a poset of width w compatible with all the answers given so far as follows: take any w-colouring of G, say $c : \{0, 1, \ldots, n-1\} \to \{0, 1, \ldots, w-1\}$ and consider the partial order $\preceq_c$ defined by $x \preceq_c y$ iff $c(x) = c(y)$ and $x \leq y$, or $x < n - t \bmod w \leq y$. In such a poset, the only t-top set is

$\{n-t, n-t+1, \ldots, n-1\}$, so this must be the correct output: indeed, the last $n-t \bmod w$ elements are larger than all the other ones; once these are removed, the elements $\{n-t, \ldots, n-t \bmod w - 1\}$ form $\lfloor t/w \rfloor$ cliques, each corresponding to a layer in the poset.

Note that if the algorithm gives an output when a vertex z out of the first $n-t$ ones is not tainted, the output would be wrong if we made z the largest among the elements not in U with the same colour as z. But a vertex z out of the first $n-t$ ones must have been subject of at least w queries to become tainted. We conclude that $w(n-t)/2$ queries are necessary to guarantee that all of the first $n-t$ vertices are tainted. □

With the same construction as in the previous proof, but using a t-colourable graph and adding a w-sized clique of *smallest* elements, we can obtain the following result for the case $t < w$:

Corollary 7 *Every correct algorithm must, in the worst case, perform at least*

$$\frac{t}{2}(n-t-w)$$

queries to get a t-top set of a n-elements poset of width $w > t$.

The previous theorems imply that, when $t \geq w$ and $t = o\left(n/\log \frac{n}{w}\right) \supseteq o\left(n/\log n\right)$, Algorithm 2 is $(3+\epsilon)$-slow; when $t < w$ and $w = o(n)$ Algorithm 1 is $(4+\epsilon)$-slow. Here $\epsilon = \epsilon(n)$ and $\lim_{n\to\infty} \epsilon(n) = 0$.

5 A Probabilistic Algorithm

We now attack the problem from a completely different viewpoint. Since our lower bounds are based on very peculiar posets, we resort to a wonderful asymptotic structure theorem proved by Kleitman and Rothschild [8]: almost all posets of n elements are *good*, that is, they are made of three levels of approximate size $n/4$, $n/2$, and $n/4$, respectively.[4] The idea is that of writing the algorithm as if *all* posets were good. In this way we will get an asymptotically very fast (albeit "probably" useless) algorithm that returns top sets on almost all posets.

Before embarking on this task, we should point out that we are not claiming that the structure of good posets reflects a real-world situation, especially not in the case of pictures; rather, it should be considered a promising starting point for further investigation, and a base case that is useful when nothing is known about the actual distribution of inputs.

[4]The original paper creates levels by stripping *minimal* elements, but by duality all results are valid also in our case.

Stated in a slightly more precise form, Kleitman and Rothschild prove the following property. If you draw at random a poset $(X, \preceq)$ uniformly among all posets of n elements, then with probability $1 - O(1/n)$ the poset is *good*, that is:[5]

- X can be partitioned into three antichains L_1, L_2 and L_3;
- $||L_i| - \frac{n}{4}| \leq \sqrt{n} \log n$ for $i = 1, 3$;
- every element in L_i only covers elements in L_{i+1} (for $i = 1, 2$); hence, in particular, every element in L_3 is minimal and every element in L_1 is maximal;
- every non-maximal (non-minimal, resp.) element is covered by (covers, resp.) at least $n/8 - n^{7/8}$ elements.
- every element in L_1 (L_3) covers (is covered by, resp.) at least $n/4 - n^{7/8}$ elements of L_2.

In other words, almost every poset is made by just three antichains, and contains about $n/4$ minimal and maximal elements, whereas the remaining elements are just "sandwiched" between a maximal and a minimal element. The main idea of our first algorithm is that if we need no more than $\big(1/4 - \epsilon\big)n$ top elements, they are easy to find if the poset is good—and almost all posets *are* good. Indeed, top elements are so easy to find that we can just extract three elements at random, hoping that they form a chain, and in that case take the largest element of the chain, as described in Algorithm 3 (the bound on the number of comparisons can be immediately derived from the condition of the while loop).

Algorithm 3 (An $O(t + \log n)$ algorithm for good posets and $t \leq (1/4 - \epsilon)n$)

top(P: a poset of order n, t: a positive integer, with $t \leq \big(1/4 - \epsilon\big)n$)

```
t' ← max(64t/(5ε), 256 ln n/(5ε))
T ← ∅
i ← 0
while |T| < t and i < t' do
    choose u.a.r. without replacement 3 elements x_1, x_2, x_3 from P
    perform all comparisons between x_1, x_2, x_3
    if the x_i's form a chain and x_ī is the maximum then
        T ← T ∪ {x_ī}
        P ← P \ {x_ī}
    end if
    i ← i + 1
end while
if |T| = t then
    return T
else
    fail
end if
```

[5]The result proved in [8] is actually stronger, but we are only quoting the properties of good posets that we will be needing in our proof.

Let ξ_{fail} be the event that the algorithm fails, and let ξ_{good} be the event that the chosen poset is good. Then

$$P[\xi_{\text{fail}}] = P[\xi_{\text{fail}} \mid \xi_{\text{good}}]P[\xi_{\text{good}}] + P[\xi_{\text{fail}} \mid \overline{\xi_{\text{good}}}]P[\overline{\xi_{\text{good}}}]$$
$$\leq P[\xi_{\text{fail}} \mid \xi_{\text{good}}] + P[\overline{\xi_{\text{good}}}] \leq P[\xi_{\text{fail}} \mid \xi_{\text{good}}] + O\left(\frac{1}{n}\right).$$

So, $P[\xi_{\text{fail}} \mid \xi_{\text{good}}] = O(1/n)$ entails $P[\xi_{\text{fail}}] = O(1/n)$.

The following lemma can be shown using Chernoff's bounds, with the observation that during the execution of the algorithm, every time we choose x_1, x_2, x_3, such a triple of elements with constant probability will be a chain.

Lemma 8 *The probability that Algorithm 3 fails is at most* $O(1/n)$.

It is an easy observation that we can modify Algorithm 3 to select *minimal* elements, hence we can use it when $t \geq (\frac{3}{4} + \epsilon)n$ by selecting $n - t$ minimal elements and giving in output the rest of the poset.

We point out that the algorithm can fail in two different ways: unconsciously, if the given poset was not good, or consciously, if the poset was good but not enough maximal elements were found.

5.1 An $O(n)$ Algorithm for Every t

We just presented fast algorithms for small t (less than $(\frac{1}{4} - \epsilon)n$) and large t (more than $(\frac{3}{4} + \epsilon)n$). For the remaining cases, we provide an alternative linear algorithm.

Consider Algorithm 4: it tries to classify the elements of an (assumed good) poset into its three levels (P_1 should contain maximal elements, P_3 minimal elements, and P_2 the elements in the intermediate layer).

Let us first observe that if we run Algorithm 4 on a good poset P the elements put in the sets P_1, P_2 and P_3 are classified correctly after the loop at line 3. Moreover:

Lemma 9 *In a good poset, the loop at line 3 performs* $O(n)$ *comparisons with probability* $1 - O(1/n)$.

Proof Consider a single execution of the loop at line 4, and let X be the random variable denoting the number of times the condition at line 7 was true. For any given x_1, the probability to extract a chain involving x_1 is at least $(1/8 - o(1))^2 \geq 1/64 - o(1)$; hence $\mu = E[X] \geq |P'|/65$. Chernoff's bound, if $|P'| \geq 524 \ln n$, implies

$$P\left[X < \frac{|P'|}{130}\right] \leq P[X < \mu/2] \leq e^{-\mu/8} \leq e^{-|P'|/520} = O(n^{-\rho})$$

for some $\rho > 1$. Letting $P'_0, P'_1, \ldots$ be the cardinality of P' after $0, 1, \ldots$ executions of the loop at line 3, we have $|P'_0| = n$ and for each $i > 0$ the size reduction $|P'_i| \leq 129|P'_{i-1}|/130$ happens with probability $1 - O(n^{-\rho})$.

Notice that as long as the loop is repeated for a number of times that is $o(n^{\rho-1})$, the probability that the size reduction does not happen at some step is bounded by

Algorithm 4 An $O(n)$ algorithm for good posets and every t.

top(P: a poset of order n, t: a positive integer)

$P_1 \leftarrow \emptyset$, $P_2 \leftarrow \emptyset$, $P_3 \leftarrow \emptyset$
$P' \leftarrow P$
while $|P'| \geq 524 \ln n$ **do**
  for all $x_1 \in P'$ **do**
    choose u.a.r. without replacement two elements x_2, x_3 from P
    perform all comparisons between x_1, x_2, x_3
    if there is a permutation $\pi \in S_3$ such that $x_{\pi(1)} \succ x_{\pi(2)} \succ x_{\pi(3)}$ **then**
      for $i = 1, 2, 3$ **do**
        $P_i \leftarrow P_i \cup \{x_{\pi(i)}\}$
      end for
    end if
    $P' \leftarrow P - (P_1 \cup P_2 \cup P_3)$
  end for
end while
for all $x \in P'$ **do**
  $\ell \leftarrow$**getLevel**(x)
  $P_\ell \leftarrow P_\ell \cup \{x\}$
end for
given the subdivision of P into its three layers P_1, P_2, P_3, determine a t-top set T by first drawing elements P_1, then possibly from P_2, and finally, if necessary, from P_3
return T

getLevel(x)

choose u.a.r. with replacement $q = \lceil 7.5 \ln n \rceil$ elements $y_1, y_2, \ldots, y_q$ from the poset
if x is comparable with at least one y_i **then**
  return 1, 3 or 2 depending on whether x is maximal, minimal, or neither in $\{x, y_1, \ldots, y_q\}$
else
  fail
end if

$O(1/n)$. Since after $k = O(\log n)$ steps $|P'_k| < 524 \ln n$, we conclude that with probability $1 - O(1/n)$ the loop is exhausted having performed a size reduction at each step, which also imply that the overall number of comparisons performed will be at most

$$\sum_{i=0}^{k-1} 3|P'_i| \leq \sum_{i=0}^{k-1} 3\left(\frac{129}{130}\right)^i n = O(n). \qquad \square$$

We are now ready to prove correctness of Algorithm 4:

Theorem 10 *With probability* $1 - O(1/n)$, *Algorithm* 4 *finds a* t*-top set performing* $O(n)$ *comparisons.*

Proof As before, we will just show that $P[\xi_{\text{fail}} \mid \xi_{\text{good}}] = O(1/n)$, which is enough to obtain the result. In the case of good posets, by Lemma 9 during the loop at line 3 $O(n)$ comparisons are performed with probability $1 - O(1/n)$. In the second loop (at line 15) no more than $O(\log^2 n)$ comparisons are performed. In this part of the algorithm, though, it may happen that some element is misclassified: the probability that x is non-maximal (non-minimal) and still no element y that x covers (covered by x) is found is at most

$$\left(1 - \frac{1}{8} + O\left(n^{-1/8}\right)\right)^{\lceil 7.5 \ln n \rceil} \leq n^{7.5 \ln(\frac{7}{8} + O(n^{-1/8}))} \leq O\left(n^{-1.001}\right).$$

So, once more, with probability $1 - O(1/n)$, all elements of P' are correctly classified at the end of the second loop. □

5.2 A Probabilistic Lower Bound and a Proof of Optimality

In this section, we will prove that no algorithm that fails with small probability can perform less than $\Omega(\min(t, n - t) + \log n)$ queries, for $0 < t < n$ (this assumption is necessary, because for $t = 0$ and $t = n$ zero queries are sufficient). As a consequence, we shall obtain that our probabilistic algorithms can be merged to obtain an asymptotically optimal one.

We say that a probabilistic algorithm is *successful* if, for some $\eta > 0$, its failure probability is at most $O(n^{-\eta})$. We will show that no successful algorithm can perform less than $\Omega(\min(t, n - t) + \log n)$ queries.

Our first lemma states that, for a probabilistic algorithm to be successful, it must perform $\Omega(\min(t, n - t))$ queries; otherwise, indeed, the failure probability does not even go to zero:

Lemma 11 *Suppose that a probabilistic algorithm is given that never performs more than* $q(n, t)$ *queries. If* $q(n, t) = o(\min(t, n - t))$ *and* $0 < t < n$ *then with nonvanishing probability the algorithm fails to return a* t*-top set when the input is chosen u.a.r. among the posets of cardinality* n.

Proof We can restrict without loss of generality to good posets, since they are a nonvanishing fraction of all posets.

Let us call "compared" an element that has been used in a comparison at least once during the execution of the algorithm. As $q(n, t)$ is $o(n)$, $o(t)$ and $o(n - t)$, we have that the compared elements are at most:

- a $o(1)$ fraction of the elements in the output;
- a $o(1)$ fraction of the elements not in the output;
- a $o(1)$ fraction of the elements in the poset level L_i, for each $i = 1, 2, 3$.

Now, for a given execution of the algorithm on an input poset of n elements, let $\mathcal{P}$ be the class of good posets of n elements that are compatible with the answers

provided to the algorithm. The third property above, via relabelling (and the random choice of the poset), says that, for every uncompared element u, the number of posets in $\mathcal{P}$ where $u \in L_1$ ($u \in L_2$, $u \in L_3$, respectively) is $1/4 + o(1)$ ($1/2 + o(1)$, $1/4 + o(1)$, respectively) of the total.

The assumption $0 < t < n$ implies that there is some uncompared element x in the output, and some uncompared element y not in the output. But now:

- if $t < (3/4 - \epsilon)n$, the output is wrong for all posets in $\mathcal{P}$ where x is in L_3;
- if $t > (1/4 + \epsilon)n$, the output is wrong for all posets in $\mathcal{P}$ where y is in L_1.

Each of the two latter events ($x \in L_3$ and $y \in L_1$) has probability at least $1/4 + o(1)$, so in both cases the algorithm would fail with probability at least $1/4 + o(1)$. □

We need something more than this, though: we need to prove that, if just $o(\log n)$ queries are performed, then the probability that the algorithm fails is still too high. To show this result, we first state a combinatorial lemma concerning the number of good posets with prescribed sets of elements in their middle layer: for a given set X, let $\mathcal{M}_X$ be the set of good posets of n elements in which X is contained in the middle layer.

Lemma 12 *Let $A \subseteq \{0, 1, \ldots, n-1\}$ and $x \in \{0, 1, \ldots, n-1\} \setminus A$; if $a = |A| = o(n)$ then*

$$\frac{|\mathcal{M}_{A\cup\{x\}}|}{|\mathcal{M}_A|} = \frac{1}{2} + o(1).$$

Proof Let us first restrict our attention to the part of $\mathcal{M}_A$ with prescribed cardinalities ℓ_1, ℓ_2, ℓ_3 of the three layers (of course, $\ell_2 \geq a$). We can divide this set into classes, according to the three sets of elements L_1, L_2, L_3 that fall in each layer; these classes will all have the same cardinality, by a simple relabelling argument, so we can limit ourselves to studying how many of these classes will be such that $x \in L_2$. This is equivalent to counting the number of ways of dividing an $n - a - 1$ elements set into three parts of cardinalities $\ell_1, \ell_2 - a - 1, \ell_3$ over the number of ways of dividing an $n - a$ elements set into three parts of cardinality $\ell_1, \ell_2 - a, \ell_3$, so:

$$\frac{|\mathcal{M}_{A\cup\{x\}}|}{|\mathcal{M}_A|} = \binom{n-a-1}{\ell_1, \ell_2 - a - 1, \ell_3} \Big/ \binom{n-a}{\ell_1, \ell_2 - a, \ell_3} = \frac{\ell_2 - a}{n-a} = \frac{1}{2} + o(1),$$

recalling that in a good poset $\ell_2 = n/2 + o(n)$, and $a = o(n)$ by hypothesis. □

We can now prove our second probabilistic lower bound:

Lemma 13 *Suppose that a probabilistic algorithm is given that never performs more than $q(n,t)$ queries. If $q(n,t) = o(\log n)$ and $0 < t < n$ then, for all $\eta > 0$, with probability $\omega(n^{-\eta})$ the algorithm fails to return a t-top set when the input is chosen u.a.r. among the posets of cardinality n.*

Proof Let $x_1, x_2, \ldots, x_k$, with $k = o(\log n)$, be the elements queried by the algorithm, in the order in which they are first queried; let ξ_{good} be the event that the poset is good, and ξ_i be the event that the poset is good and $x_i \in L_2$.

We start by showing that all the different elements queried by the algorithm will belong to the middle layer L_2 with nonnegligible probability, that is,

$$P[\xi_1 \cap \cdots \cap \xi_k] = \omega(n^{-\eta}), \quad \forall \eta > 0. \tag{1}$$

By the chain rule we get,

$$\begin{aligned} P[\xi_1 \cap \cdots \cap \xi_k] &= P[\xi_{\text{good}}] P[\xi_k \cap \xi_{k-1} \cap \cdots \cap \xi_1 \mid \xi_{\text{good}}] \\ &= P[\xi_{\text{good}}] \prod_{i=1}^{k} P[\xi_i \mid \xi_{i-1} \cap \cdots \cap \xi_1 \cap \xi_{\text{good}}]. \end{aligned}$$

By Lemma 12, $P[\xi_i \mid \xi_{i-1} \cap \cdots \cap \xi_1 \cap \xi_{\text{good}}] = 1/2 + o(1)$, as long as $i = o(n)$. Since $P[\xi_{\text{good}}] = 1 - O(1/n)$, we have, for all $\eta > 0$,

$$P[\xi_1 \cap \cdots \cap \xi_k] = \left(1 - O\left(\frac{1}{n}\right)\right)\left(\frac{1}{2} + o(1)\right)^{o(\log n)} = \omega(n^{-\eta}).$$

Now, to obtain the contradiction, consider two cases:

- if $t < (3/4 - \epsilon)n$, we can assume that at least one output element x has not been compared (because otherwise with the above probability the output contains only elements from the middle layer, missing the elements from the top layer);
- if $t > (1/4 + \epsilon)n$, we can assume that at least one element y not in the output has not been compared (because otherwise the output contains all elements from the top and bottom layers, but misses some elements from the middle layer).

In both cases, we can proceed as in the last part of the proof of Lemma 11. □

It is now straightforward to obtain the following theorem:

Theorem 14 *Every successful algorithm performs at least* $\Omega(\min(t, n - t) + \log n)$ *queries to return a* t*-top set* $(0 < t < n)$ *of a poset of order* n *chosen uniformly at random.*

Now, we can merge Algorithms 3 and 4 as follows:

- if $0 < t < (1/4 - \epsilon)n$ we apply Algorithm 3, performing $O(t + \log n)$ queries;
- if $(1/4 - \epsilon)n \le t \le (3/4 + \epsilon)n$ we apply Algorithm 4, performing $O(n)$ queries;
- if $(3/4 + \epsilon)n < t < n$ we apply a modified version of Algorithm 3 that searches for $n - t$ minimal elements, and returns the top set containing all the elements but those found; in this case, we perform $O((n - t) + \log n)$ queries.

The algorithm obtained in this way performs $O(\min(t, n - t) + \log n)$ queries, so it is optimal in the class of successful algorithms (i.e., it is θ-slow with respect to the optimal algorithm in the class, for some suitable constant θ).

6 Preorders

All our algorithms can be extended to work on preorders, as we informally show in this section. A *preorder* is a set endowed with a reflexive, transitive relation $\preceq$ (which however might fail to be antisymmetric). In a preorder, if $x \preceq y$ and $y \preceq x$ we say that x and y are *equivalent* and write $x \sim y$.

Suppose we take a picture of the same scene twice, to increase the probability that at least one is not blurred, and that both copies turn out to be perfect shots. In this case, $x \sim y$, as we can clearly compare the two pictures, and we have no preference about one or the other. This situation is very different from the sea/desert dilemma, in which we are not able to express an opinion about which picture is better: in the case of the two shots, we want just one of the two pictures, whereas in the incomparability case we would like to keep both.

There are at least two possible ways to extend the notion of t-top sets to preorders. One could just search for a t-top set of the poset $P/\sim$ obtained by quotienting the original preorder P with respect to the equivalence relation $\sim$. In this case, the output of our algorithms should be a set of equivalence classes. Alternatively, as a perhaps more sensible solution, we might return an arbitrarily chosen representative for each equivalence class.

To treat preorders, we add a new possible answer to queries (denoted, of course, by $x \sim y$). We change the first two algorithms in the following way: instead of handling antichains, we manipulate quasi-antichains, that is, sets of $\sim$-equivalent classes that are pairwise *incomparable* (i.e., any two elements from distinct classes are incomparable, and any two elements within the same class are $\sim$-equivalent). When the algorithm wants the query $X :: Y$ to be answered (where X and Y are $\sim$-equivalence classes), we choose arbitrarily an element in $x \in X$ and an element $y \in Y$ and we pass the query $x :: y$ to the oracle. If the result of the query is $x \sim y$, we just merge the two classes X, Y, adding a class $X \cup Y$ and removing both X and Y. Otherwise, we let the algorithm proceed as before.

It is easy to check that this strategy works and that it does not increase the number of comparisons of the algorithms. The deterministic lower bounds we gave trivially hold for preorders as well, since each partial order is also a preorder.

6.1 Preorders in the Probabilistic Setting

To handle preorders in the probabilistic setting, we use Marcel Erné's results [5, Lemma 4.6, p. 253] stating that the ratio between the number of preorders of order n (therein labelled $A(n)$) and the number of posets of order n ($A_0(n)$) is

$$\frac{A(n)}{A_0(n)} = 1 + 2^{-n/2+O(\sqrt{n}\log n)} \leq \frac{1}{1-O(1/n)}.$$

As a consequence, almost all preorders are good posets; more precisely, since the ratio between the numbers of posets and preorders is at least $1 - O(1/n)$, and at least $1 - O(1/n)$ of these posets are good, the fraction of preorders that turn out to be good posets is at least $(1 - O(1/n))^2 = 1 - O(1/n)$.

This observation is sufficient to show that both the probabilistic algorithms and the corresponding lower bounds hold for preorders as well.

7 Conclusions

We have introduced the problem of finding a set of t best elements (that we called top set) from a poset of n elements. The first two solutions we have provided are deterministic: one requires $O(tn)$ comparisons, while the other needs $O(wn + wt \log(n/w))$ (where w is the width of the poset).

In general, the former algorithm is 2-slow (meaning that, for all posets, it does not pose more than twice the number of queries that the best algorithm would pose in the worst case).

For posets of limited width w, under the assumption that t, w are not too large, we showed an inherent difference between the cases $t < w$ and $t \geq w$: if $t < w = o(n)$, the first algorithm is $(4 + \epsilon)$-slow, whereas if $o(n/\log n) = t \geq w$ the second algorithm is $(3 + \epsilon)$-slow (here, $\epsilon = \epsilon(n)$ and $\lim_{n\to\infty} \epsilon(n) = 0$).

Then, to beat the lower bounds used to determine the slowness of the two deterministic algorithms, we assumed that the input poset was chosen uniformly at random between all posets of size n. Under this assumption, we were able to obtain a probabilistic algorithm capable of determining t-top sets in time $O(\min(t, n - t) + \log n)$ with probability $1 - O(1/n)$. This probabilistic algorithm happens to be $O(1)$-slow in the class of all algorithms that choose their input u.a.r. and fail with at most an inverse polynomial probability (that is, $O(n^{-\eta})$ for some $\eta > 0$). In other words, our algorithm is asymptotically optimal under the stated success probability.

Finally, we considered preorders instead of just posets. A top set of a preorder is a top set of the poset obtained by quotienting the preorder with respect to its equivalence relation. We showed that all our upper and lower bounds hold also in the case of preorders.

Acknowledgements P.B. and S.V. were partially supported by the MIUR PRIN Project "Automi e linguaggi formali: aspetti matematici e applicativi".

F.C. was partially supported by a grant of Yahoo! Research and by the MIUR PRIN Project "Web Ram: web retrieval and mining".

We started chatting about this problem when we were still in Mongolia: we wish to thank all our travel companions, namely, Roberto Cuni, Corrado Manara, Roberto Radicioni and Lorenzo Virtuoso, our guide and translator (and cook!) Miss "it's-possible" Nora, and, last but not least, our tireless driver, mechanic, human-GPS, heroic Puujee.

We also want to thank Mongolia itself: among other things, for that August starry night in the desert that we shall never forget ....

References

1. Boldi, P., Chierichetti, F., Vigna, S.: Pictures from Mongolia—partial sorting in a partial world. In: Proceedings of FUC with Algorithms 2007. Lecture Notes in Computer Science, vol. 4475, pp. 66–77. Springer, Berlin (2007)
2. Cowan, T.A., Fishburn, P.C.: Foundations of preference. In: Essays in Honor of Werner Leinfellner, pp. 261–271. Reidel, Dordrecht (1988)
3. Daskalakis, C., Karp, R.M., Mossel, E., Riesenfeld, S., Verbin, E.: Sorting and selection in posets. Computing Research Repository (CoRR) (July 2007). Available at http://arxiv.org/pdf/0707.1532
4. Dwork, C., Kumar, R., Naor, M., Sivakumar, D.: Rank aggregation methods for the web. In: WWW '01: Proceedings of the 10th International Conference on World Wide Web, pp. 613–622. ACM, New York (2001)

5. Erné, M.: Struktur- und Anzahlformeln für Topologien auf endlichen Mengen. Manuscripta Mathematica **11**, 221–259 (1974)
6. Fagin, R., Kumar, R., Sivakumar, D.: Efficient similarity search and classification via rank aggregation. In: SIGMOD '03: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, pp. 301–312. ACM, New York (2003)
7. Faigle, U., Turán, G.: Sorting and recognition problems for ordered sets. SIAM J. Comput. **17**(1), 100–113 (1988)
8. Kleitman, D.J., Rothschild, B.L.: Asymptotic enumeration of partial orders on a finite set. Trans. Am. Math. Soc. **205**, 205–220 (1975)