- A third question is raised by our local sorting algorithm, which makes comparisons between distant elements of the graph in order to finally determine the local order relationships. Is this necessary? Specifically: Is there a local sorting algorithm which uses $\Theta(\log \alpha(G))$ comparisons but never compares elements over a distance greater than some constant *comparison horizon* h? Our algorithm, modified so that $E(|R_k|) = n/\log n$, needs $h(n) = \sqrt{\log \log n}$. The question is particularly intriguing because our algorithm actually makes rather few long-distance comparisons (e.g. only linearly many beyond distance 4).

  We know only that $h = 1$ is not sufficient. This answers a question of Linial [L]. Consider a bipartite graph where all values in one color class are greater than those in the other: all edges must be checked in order to certify this.

- A more combinatorial question is that of obtaining better upper and lower bounds on $\alpha(G)$.

Acknowledgements

Claire Kenyon's suggestions resulted in the saving of a factor of $\log^* n$ in the complexity of the set-maxima algorithm. The authors are grateful for her careful review of that algorithm and analysis. They would also like to thank Michael Sipser for many discussions and considerable advice. Further, they would like to thank Michelangelo Grigni, Mark Haiman, Russell Impagliazzo and Mauricio Karchmer for helpful discussions.

References

[BMN] A. Bar-Noy, R. Motwani, and J. Naor, "A Linear Time Approach to the Set Maxima Problem," *manuscript.*

[GYY] R. Graham, F. Yao, and A. Yao, "Information Bounds are Weak in the Shortest Distance Problem," *JACM* **27** (1980) pp. 428–444.

[GS] W. Goddard and L. Schulman, "An Optimal Probabilistic Algorithm for Local Sorting," *unpublished manuscript.*

[KK] C. Kenyon and V. King, "Verifying Partial Orders," *Proceedings of the 21st ACM Symposium on Theory of Computing*, 1989, pp. 367–374.

[K] J. Komlós, "Linear Verification for Spanning Trees," *Combinatorica* **5** (1985) pp. 57–65.

[L] N. Linial, "Legal Coloring of Graphs," *Combinatorica* **6** (1986) pp. 49–54.

[MT] U. Manber and M. Tompa, "The Effect of Number of Hamiltonian Paths on the Complexity of a Vertex-Coloring Problem," *SIAM J. Comp.* **13** (1984) pp. 109–115

[S] R. Stanley, "Acyclic Orientations of Graphs," *Discrete Mathematics* **5** (1973) pp. 171-178.

[T] R. Tarjan, *private communication.*

[Y] A. Yao, *private communication.*

the bounds of Lemma 2 show that

$$\begin{aligned} E[|Y|\log(\textstyle\sum_j \mathrm{rank}(r_j)/|Y|)] \\ &\le E[|Y|\log(10m(\log^{(i)} n)^4/|Y|)] \\ &\le E(|Y|)\log(10m(\log^{(i)} n)^4/E(|Y|)). \end{aligned}$$

Similarly, the expectation of $|Y|\log(\sum_j(\mathrm{rank}(r_j))^2/|Y|)$ is bounded above by

$$E(|Y|)\log(40m(\log^{(i)} n)^{12}/E(|Y|)).$$

Hence, the expected sum of (2) and (3) is no greater than

$$cE(|Y|)\log(m/E(|Y|)) + c'E(|Y|)\log^{(i+1)} n, \qquad (4)$$

for c, c' constants. This expression has a positive derivative with respect to $E(|Y|)$ as $E(|Y|) \le n \le m$. Thus, we may substitute the upper bounds of Lemma 3 to compute upper bounds on (4).

Now, we evaluate (4) at $|Y| = n$ and $i = i_0$ to count the cost of the first call to the procedure at level i_0. This is at most $cn\log(m/n) + c'n\log^{(i_0+1)} n$ which is bounded above by $(c+c')n\log(m/n)+O(n)$, since either $m \ge n\log^{(i_0)} n$ or $n\log^{(i_0)} n = n\log^{(\log^* n-5)} n \le 2^{2^{16}} n$.

At level $i_0 - 1$, $E(|Y|) \le n/\log^{(i_0)} n$. Hence the three calls to A at this level each use less than

$$\frac{cn}{\log^{(i_0)} n}\log\left(m\frac{\log^{(i_0)} n}{n}\right) + \frac{c'n\log^{(i_0)} n}{\log^{(i_0)} n} \le (2c+c')n$$

comparisons, since $m \le n\log^{(i_0-1)} n$.

There are 3^{i_0-i} calls at levels i, for $i = 1$ to $i_0 - 2$. An upper bound on the cost of each is given by the following where we have substituted $n/(\log^{(i+1)} n\,\log^{(i+2)} n)$ for $E(|Y|)$ and $n\log^{(i+1)} n$ for m:

$$\frac{cn\log\left((\log^{(i+1)} n)^2\log^{(i+2)} n\right)}{\log^{(i+1)} n\log^{(i+2)} n} + \frac{c'n}{\log^{(i+2)} n} \le \frac{(c+c')n}{\log^{(i+2)} n}.$$

Finally, we determine the cost of the sorts in level 1 calls. Using Lemma 3, we see that if $i_0 \ge 2$ then the expected size of R in a level 1 call is the expectation of $|Y|/\log n$ which is at most $n/(\log n\log\log n)$. If $i_0 = 1$ then R has expected size $n/\log n$. As there are 3^{i_0-1} level 1 calls, the expected cost for these sorts is at most $O(n)$, for any i_0.

Hence the expected total comparisons for A is at most:

$$\begin{aligned} (c+c')n\log(m/n) + O(n) + \sum_{i=1}^{i_0-2} 3^{i_0-i}(c+c')n/\log^{(i+2)} n \\ \le O(n\log(m/n)+n). \end{aligned}$$

We may conclude:

Theorem 4 *Let m sets be given from a totally ordered n-element universe, with $m \ge n$. Then the expected number of comparisons required by the set-maxima algorithm is at most:*

$$\min\{O(n\log(m/n)+n), O(n\log n)\}.$$

A matching lower bound is easily observed:

Theorem 5 *If m is $O(n^2)$ then there exist collections of m sets such that*

$$\Omega(n\log(m/n)+n)$$

comparisons are required for set-maxima.

Proof: If one set contains all elements then $n-1$ comparisons are necessary. Now, let us assume that $2m/n$ and $n^2/2m$ are integers (though the argument can be modified for any m). Then, if we partition the universe into $n^2/(2m+n)$ blocks of size $(2m+n)/n$, and take as our sets all pairs of each block, then we have m sets arranged so that set-maxima sorts each block. This yields a lower bound of $\Omega(n\log(m/n))$. □

8 Open Questions

We list below some questions which we have encountered.

- What about deterministic algorithms? No nontrivial deterministic algorithm for local sorting is known other than on dense graphs where one might as well totally sort the elements. The same is true of set-maxima with the exception of the restricted set systems discussed in [K] and [BMN].
- Can the time complexity of the set-maxima and local sorting algorithms (currently polynomial) be brought into line with their comparison model complexity?

in the interval between 2^{k-1} and 2^k until x is placed among the elements of T_x,

or

(b) x is greater than the largest sorted element of T_x, and x's rank in T_x cannot be determined. (This can only occur if $i \geq 2$ and $|T_x| > (\log^{(i-1)} n)^2$.) Then set x aside.

Let Y' be the set of all x set aside in (b). If $i = 1$ then Y' is empty.

Let $W_j' = \{ x \in (Y - Y') \cap W_j : x \geq r_j \}$.

Step 4. If $i \geq 2$ do $A^L(i-1, Y', \{W_1, W_2, \ldots, W_{m'}\})$, to find and sort the largest $(\log^{(i-1)} n)^2$ elements of $Y' \cap W_j$.

Step 5. Let $W_j'' = W_j' \cup \{$ the $(\log^{(i)} n)^2$ largest elements of $Y' \cap W_j \}$. Apply the set-sort algorithm (Corollary 2) to the sets $W_1'', W_2'', \ldots, W_{m'}''$.

7 Analysis of the Set-Maxima Algorithm

We analyze the expected number of comparisons used by $A(i_0, X, \{S_1, S_2, \ldots, S_m\})$. All comparisons occur in Steps **3, 5**, and, at level 1, in Step **1**. We start by analyzing those occurring in Steps **3** and **5**.

We examine the number of comparisons of a single call of the form $A(i, Y, \{W_1, W_2, \ldots, W_{m'}\})$, where $i \leq i_0$, $m' \leq m$, and $Y, W_1, W_2, \ldots, W_{m'} \subseteq X$. We will refer to any such call of the form $A(i, Y, \ldots)$ as a "level" i procedure call with "domain" Y.

Let $\text{rank}(r_j) = |\{ x \in W_j \cap Y : x \geq r_j \}|$.

The number of comparisons needed for each element $x \in Y$ in Step **3**, the doubling search, is the cost of searching for the interval in which x lies, and, if one is found, the cost of locating x within the interval.

Thus, Step **3** requires at most

$$\sum_{x \in Y} 2 \log(|\{ r_j : r_j \leq x, x \in W_j \}|)$$

comparisons. But $\sum_{x \in Y} |\{r_j : r_j \leq x, x \in W_j\}|$ is at most $\sum_j |\{x \in W_j \cap Y : x \geq r_j\}|$, the sum of the ranks of the representatives. Hence by the concavity of the log function, Step **5** requires at most

$$2|Y| \log \left(\sum_j \text{rank}(r_j)/|Y| \right) \qquad (2)$$

comparisons.

In Step **5**, the m' sets have size at most $\text{rank}(r_j) + (\log^{(i)} n)^2$, and are contained in Y. Therefore the set-sort algorithm (Corollary 2) takes at most

$$c_0 |Y| \log \left(\sum_j \left(\text{rank}(r_j) + (\log^{(i)} n)^2 \right)^2 / |Y| \right) \qquad (3)$$

comparisons, for c_0 a constant. Thus the total number of comparisons required in a single call to this recursive procedure is the sum of terms (2) and (3).

To compute the expectation of these terms, we determine upper bounds on $E(\text{rank}(r_j))$, $E((\text{rank}(r_j))^2)$, and $E(|Y|)$. The proofs of the first two are straightforward and are omitted in this abstract.

Lemma 2
a) $E[\text{rank}(r_j)] \leq 10(\log^{(i)} n)^4$, *independent of* $|Y|$.
b) $E[(\text{rank}(r_j))^2] \leq 40(\log^{(i)} n)^{12}$, *independent of* $|Y|$.

Lemma 3 *The expected size of any domain in a level i procedure call is no greater than:*
a) n for $i = i_0$;
b) $n/\log^{(i_0)} n$ for $i = i_0 - 1$;
c) $n/(\log^{(i+1)} n \log^{(i+2)} n)$ for $i \leq i_0 - 2$.

Proof: Every level $i-1$ call initiated in Step **4** has domain Y' where $x \in Y'$ only if x is greater than at least $(\log^{(i-1)} n)^2$ of the $r_j \in T_x$. But there are at most $\sum_j \text{rank}(r_j)/(\log^{(i-1)} n)^2$ such x's; hence this is an upper bound on $|Y'|$ and

$$E(|Y'|) \leq E(\sum_j \text{rank}(r_j))/(\log^{(i-1)} n)^2.$$

By Lemma 2a, and because $m \leq n \log^{(i-1)} n$, the above is at most $10(\log^{(i)} n)^4 / \log^{(i-1)} n$. But $i \leq \log^* n - 5$, hence $\log^{(i-1)} n \geq 10(\log^{(i)} n)^5 (\log^{(i+1)} n)$, and so $E(|Y'|) \leq n/(\log^{(i)} n \log^{(i+1)} n)$.

When the level i_0 call makes the level $i_0 - 1$ calls in Steps **1** and **2**, the domain R has expected size $n/\log^{(i_0)} n$.

If we assume that any domain Z for any level i call has expected size at most $n/\log^{(i+1)} n$, then the level $i-1$ calls at Steps **1** and **2** have domain R with expected size $|Z|/\log^{(i)} n \leq n/(\log^{(i+1)} n \log^{(i)} n)$. □

We may now evaluate the expected values of (2) and (3). Calculus and concavity arguments together with

2. *Insert x in T_x*

We determine for each x, the set $\{ r_j \in T_x : r_j < x \}$; this is equivalent to determining for each j, the reduced set S'_j.

We perform a "doubling" search, rather than a binary search, to find the interval of T_x in which x lies: One compares x with the element of T_x of rank $1, 2, 4, \ldots, 2^l$, until an element greater than x, if there is any, is found, and then performs a binary search in the interval $[2^{l-1}, 2^l)$ to find where x lies.

Starting the search at the bottom of T_x is more efficient because the average x is likely to be smaller than most of the representatives, and thus towards the bottom of T_x.

3. *Find the maximum of each S'_j using brute force.*

The major change we make is to avoid the complete sort of the sample R. We used the fact that R was sorted to (i) find the representative of each S_j in R, and (ii) perform doubling searches on the T_x's. Instead, we accomplish (i) by recursively solving set-maxima for the sets $S_j \cap R$ in the universe R.

One could accomplish (ii) by sorting the sets T_x. However, these are too large to efficiently apply set-sort. Instead, we note that for most $x \in X$, the search of T_x ends after x has been compared with only the smallest elements of T_x. This motivates an extension of set-maxima: for given set system and $f(n)$, find and sort the $f(n)$ largest elements of each set. We generalize our approach accordingly (see below), and find and sort the smallest elements in each T_x.

Now we are able to complete the doubling search for most x's; call such an x "good". This yields for each S_j a reduced set consisting of the good x's that are at least as big as r_j. For the few x's for which the doubling search fails, we solve separately the set-maxima problem (over a much smaller universe). Then for each S_j we add its largest bad elements to its reduced set, obtaining a collection of small sets $\{S'_j\}$.

Finally, we set-sort the collection $\{S'_j\}$. In particular this finds the maximum of each S_j.

The extension of the above to general $f(n)$ requires only minor changes. For example, rather than taking the representative to be the maximum sample element in each set, we choose the $f(n)^{\text{th}}$ largest value in order to ensure that each reduced set contain at least $f(n)$ elements (if it had that many to start with).

6 The Set-Maxima Algorithm

We are given a set X of n elements from some total order, and $S_1, S_2, \ldots, S_m$, subsets of X, for $n \leq m < n^2$.

Let $\log^{(i)} n$ denote the i^{th} iteration of $\log n$, and $\log^* n$ denote the minimum i such that $\log^{(i)} n \leq 1$. Let

$$i_0 = \min\{\log^* n - 5, \min\{i : n \log^{(i)} n \leq m\}\}.$$

(We assume that n is large enough that $i_0 \geq 1$.) Note that $\log^{(\log^* n - 5)} n$ is less than a constant, $2^{2^{16}}$, for all n, but it is sufficiently large so that $\log^{(i)} n / \log^{(i-1)} n$ is small enough to ensure the bounds used in later calculations.

The algorithm A^L is defined below, recursively for Y and $W_1, W_2, \ldots, W_{m'}$ subsets of X, and $i \leq i_0$. A^L finds and sorts the largest $(\log^{(i)} n)^2$ elements in $Y \cap W_j$ for $1 \leq j \leq m'$. A^S is defined analogously for the smallest $(\log^{(i)} n)^2$ elements.

To solve the set-maxima problem on X and $S_1, S_2, \ldots, S_m$, we call $A^L(i_0, X, \{S_1, S_2, \ldots, S_m\})$.

$A^L(i, Y, \{W_1, W_2, \ldots, W_{m'}\})$

Step 1. Let R be the sample generated by choosing each $y \in Y$ independently with probability $1/\log^{(i)} n$.

If $i \geq 2$ do $A^L(i-1, R, \{W_1, W_2, \ldots, W_{m'}\})$ which finds and sorts the $(\log^{(i-1)} n)^2$ largest elements in $R \cap W_j$, for $1 \leq j \leq m'$.

If $i = 1$, sort R.

For $1 \leq j \leq m'$, let r_j be the element of largest rank $(\log^{(i)} n)^2$ in $R \cap W_j$, or $-\infty$ if there is no such element.

Step 2. For each $x \in Y$, let $T_x = \{ r_j : x \in W_j \}$. If $i \geq 2$, do $A^S(i-1, R, \{T_x\}_{x \in Y})$. This finds and sorts the smallest $(\log^{(i-1)} n)^2$ elements in $R \cap T_x = T_x$, for all $x \in Y$. If $i = 1$, then R is sorted.

Step 3. For each $x \in Y$, we do a doubling search on the sorted portion of T_x. We compare x to the elements of T_x of smallest ranks $2^0, 2^1, \ldots$ until either:

(a) x is less than some element of rank $2^k \leq (\log^{(i-1)} n)^2$. Then perform a binary search

set of F_0 and an edge of G appears exactly once, either in $G_0 = G - U_0$ or in F_0.

We continue by iterating the above procedure. At step i ($i \geq 0$), let U_i be the set of vertices of degree at most $a_i = 2^{2^i} - 1$ in G_{i-1}. Let W_i be their neighbors, and form F_i as the graph on $U_i \cup W_i$ with edges interior to U_i or between U_i and W_i; let $G_i = G_{i-1} - U_i$. Halt when G_i is the null graph.

One modification remains before we are ready to locally sort each F_i. Color red, and denote by RW_i, all vertices of W_i that are of degree at most a_i. Then split every remaining vertex of W_i into several vertices each retaining between $a_i/2$ and a_i of its originator's edges. Color these new vertices green and call them GW_i. Denote the new graph H_i and let $h_i = |H_i|$. (The coloring will be used in the analysis.)

We now apply the limited-degree local sorting algorithm (§2) to each H_i. We require an expected $O(\sum_i h_i \log(a_i + 1))$ comparisons to locally sort $H = \bigcup H_i$. Since this is equivalent to locally sorting G, this yields:

Theorem 3 *Given any graph G, the local sorting algorithm takes an expected $\Theta(\log \alpha(G))$ comparisons and is optimal.*

Proof Summary: We know from Theorem 2 that $\log \mathcal{D}(G) = \sum_{v \in G} \log(d_v + 1)$ is at most twice $\log \alpha(G)$, the information-theoretic lower bound for local sorting on G. It is therefore sufficient to establish:

$$\sum_i h_i \log(a_i + 1) \leq 8 \log \mathcal{D}(G).$$

Every non-isolated vertex v of G shows up, perhaps lacking some of its original edges, in just one U_i; here $i \leq \lceil \log\log(d_v+1) \rceil$. For each smaller i, W_i may contain points corresponding to v: either several green vertices, or just one red vertex.

We separate the contributions to $\sum_i h_i \log(a_i+1)$ into the contributions of vertices in $\bigcup U_i$, the contributions of red vertices, and those of green vertices.

The contribution $\log(a_i + 1)$ associated with a vertex v' in U_i, derived from v in G, is at most $2\log(d_v + 1)$. Thus the contribution of $\bigcup U_i$ is at most $2 \log \mathcal{D}(G)$. Also, because the series $\{a_i\}$ is doubly exponential, the total contribution of all red vertices associated with a vertex v in G is at most $2\log(d_v + 1)$. Therefore the contribution of $\bigcup RW_i$ is at most $2 \log \mathcal{D}(G)$.

In order to account for the contribution of the green vertices, we first recall that vertices of GW_i are adjacent only to vertices of U_i. Further, vertices of GW_i are of degree at least $a_i/2$ while those of U_i are of degree at most a_i. Therefore $|GW_i| \leq 2|U_i|$, and so the contribution of all green vertices is at most twice that from $\bigcup U_i$.

Combining all the above we find that $\sum_i h_i \log(a_i + 1) \leq 8 \log \mathcal{D}(G)$. □

Recall that the set-sort problem entails sorting each of several, possibly intersecting, sets. The above theorem implies:

Corollary 2 *Let $S_1, S_2, \ldots, S_m$ be sets from a totally ordered universe of size n. Then these sets can be sorted using an expected $O(n \log(\sum_j |S_j|^2/n))$ comparisons.*

5 The Ideas Behind the Set-Maxima Algorithm

The following approach to set-maxima was devised in [KK]:

1. *Choose a random sample, and define for each set a "representative" by taking the maximum sample point in that set.*
2. *Reduce each set by discarding those elements less than the representative.*
3. *Solve the reduced system.*

In this paper we retain this strategy, but improve the implementations of the three phases.

In order to introduce the specifics of our algorithm, we sketch the [KK §2] implementation. That algorithm finds the maximum element in each of $S_1, S_2, \ldots, S_m$, sets in a universe X of size n.

1. *Take a random sample R by including each $x \in X$ with some probability $p(n)$, and sort R.*

   Let r_j be the maximum element of S_j in R, if there is such an element; r_j is the representative of S_j in R. We reduce each S_j to $S'_j = \{ x \in S_j : x \geq r_j \}$, where the expected size of S'_j is "small." Now, let T_x be the set of representatives of all sets containing x, i.e., $\{ r_j : x \in S_j \}$.

3 The Acyclic Orientations of a Graph

Let $\alpha(G)$ denote the number of acyclic orientations of a graph G: the number of directed acyclic graphs having the same underlying edge set as G. (The parameter α has been studied for instance in [S].) We shall provide a close estimate of $\alpha(G)$ based solely on the degree sequence of G. The upper bound is due to Manber and Tompa [MT].

Define

$$\mathcal{D}(G) = \prod_{v \in G} (d_v + 1)$$

and

$$\mathcal{F}(G) = \prod_{v \in G} f(d_v + 1),$$

where d_v is the degree of v in G, and $f(x) = (x!)^{1/x}$.

Theorem 2 *For any graph G,*

$$\sqrt{\mathcal{D}(G)} \leq \mathcal{F}(G) \leq \alpha(G) \leq \mathcal{D}(G).$$

Proof: The proof that $\alpha(G) \leq \mathcal{D}(G)$ is in [MT]. For positive integral x, $\sqrt{x+1} \leq f(x+1)$, hence $\sqrt{\mathcal{D}(G)} \leq \mathcal{F}(G)$. Now we show that $\mathcal{F}(G) \leq \alpha(G)$.

The proof is by induction on the number of vertices of G. The case of a single vertex is trivial.

Let v be a vertex of minimum degree δ. If $\delta = 0$ then $\alpha(G) = \alpha(G - v)$ and $\mathcal{F}(G) = \mathcal{F}(G - v)$; so hereafter we assume that $\delta \geq 1$. We prove first that

$$\alpha(G) \geq (\delta + 1)\alpha(G - v). \tag{1}$$

Take any acyclic orientation $\mathcal{A}$ of $G - v$ and look at v's neighbors N in G. If N is totally ordered by $\mathcal{A}$ then there are $\delta + 1$ ways of extending $\mathcal{A}$ to an acyclic orientation of G (by choosing v's rank with respect to N); otherwise there are even more ways to extend $\mathcal{A}$.

The next thing to notice is that

$$\mathcal{F}(G) = \mathcal{F}(G - v) f(\delta + 1) \left(\prod_{v \in N} \frac{f(d_v + 1)}{f(d_v)} \right),$$

where $\{d_v\}$ indicate degrees in G (each greater than or equal to δ). Now, $f(x+1)/f(x)$ is monotone decreasing for positive integral x. So

$$\mathcal{F}(G) \leq \mathcal{F}(G - v) f(\delta + 1) \left(\frac{f(\delta + 1)}{f(\delta)} \right)^{\delta}.$$

Now from inequality 1 above and our induction hypothesis,

$$\alpha(G) \geq (\delta + 1)\mathcal{F}(G - v) \geq \mathcal{F}(G) \frac{(\delta + 1)(f(\delta))^{\delta}}{(f(\delta + 1))^{\delta + 1}}.$$

The last fraction equals 1. □

The bounds on $\alpha(G)$ are the best possible of the form $\prod_v g(d_v)$. The lower bound is attained by any disjoint union of cliques. For the upper bound consider $G = K(a, b)$ with $b \gg a$: here $\alpha(G) \sim a!(a+1)^b$. (We would like to thank Mark Haiman for discussions on this point.) The latter graphs also disprove the upper bound $\prod_v \max\{2, d_v\}$ claimed in [L].

Now, since $\log \alpha(G)$ is a lower bound on the number of comparisons required to locally sort G, we have:

Corollary 1 *A local sorting algorithm requires*

$$\Omega(\sum_{v \in G} \log(d_v + 1))$$

comparisons on G.

For example, a Δ-regular graph requires $\Omega(n \log(\Delta + 1))$ comparisons—as attained in the algorithm of §2. However, the upper bound on $\log \alpha(G)$ also suggests that for other graphs that algorithm is not optimal.

4 Local Sorting for Arbitrary Graphs

We will locally sort an arbitrary graph as follows: form several smaller graphs out of it, and then locally sort each graph separately with the limited-degree algorithm. We will be able to control the increase in $\log \alpha(G)$ in the construction and argue thereby that our algorithm is optimal. Our construction is deterministic and requires no comparisons.

Given a graph G, first discard all isolated vertices. Then let U_0 be the set of vertices of degree 1. We wish to split G into $G - U_0$ and the induced graph on U_0, and locally sort these separately (but this of course neglects the edges between these parts). So for every neighbor w of U_0 in G, we create a copy w_0 (call the set of all these W_0). Then we define the graph F_0 on $U_0 \cup W_0$ by including all edges of G that are interior to U_0 or that connect U_0 and W_0. Thus W_0 is an independent

Figure 1: Omitted

The initial step is to choose a random sample R_k of expected size $O(1)$ which is then sorted using an expected $O(1)$ comparisons.

The main part of the algorithm is an iterative process in which the partial order on R_i is used to obtain the requisite partial order on R_{i-1}. The R_i are used as "signposts" for the two parts of this process. Let B_x^r denote the set of vertices of distance at most r from vertex x. Proceed as follows.

(0) *Choose R_{i-1} by independently including each vertex with probability p_{i-1}.*

(1) *For each $x \in R_{i-1}$ do:*
- *Find the rank of x in $R_i \cap B_x^{2^{2i-1}}$ using a binary search.*

Any two elements of R_i in $B_x^{2^{2i-1}}$ are within a distance of 2^{2i} of each other—so, since R_i is locally sorted to distance 2^{2i}, their relative order is already known (i.e. $R_i \cap B_x^{2^{2i-1}}$ has been totally ordered). This allows us to find x's rank with a binary search.

(2) *For each $x \in R_{i-1}$ do:*
- *Determine C_x, the set of all $y \in R_{i-1} \cap B_x^{2^{2i-2}}$ that have the same rank as x with respect to $R_i \cap B_x^{2^{2i-2}}$.*
- *Compare x with every element in $C_x - \{x\}$.*

Note that for y in $B_x^{2^{2i-2}}$, $B_y^{2^{2i-1}}$ contains $B_x^{2^{2i-2}}$. Therefore, C_x can be constructed using no further comparisons. Furthermore, the relation of every element of $(R_{i-1} \cap B_x^{2^{2i-2}}) - C_x$ to x is known, since there is at least one intervening signpost (element of $R_i \cap B_x^{2^{2i-2}}$). Thus in order to locally sort R_{i-1} to radius 2^{2i-2}, we need only compare each x in R_{i-1} with the elements of $C_x - \{x\}$.

2.2 Analysis

How many comparisons do we use? For each $x \in R_{i-1}$, the number of comparisons in step (1) is at most $\log |B_x^{2^{2i-1}}|$ which, by our assumption on the degrees, is at most $2^{2i-1} \log(\Delta + 1)$.

For each $x \in R_{i-1}$, the expected number of comparisons in step (2) is the expected size of C_x. Intuitively the expected size of C_x should be of the order of the expected value of $|R_{i-1} \cap B_x^{2^{2i-2}}|/|R_i \cap B_x^{2^{2i-2}}|$, which one would hope to be something like p_{i-1}/p_i. This intuition is supported in the following lemma whose proof we omit.

Lemma 1 *The expected size of C_x is at most $2p_{i-1}/p_i$.*

Now, a good choice of the $\{p_i\}$ is obtained by balancing the upper bounds on the expected number of comparisons for the two steps associated with each $x \in R_{i-1}$:

$$2^{2i-1} \log(\Delta + 1) = 2p_{i-1}/p_i.$$

Solving for p_i in terms of p_{i-1} and noting that $p_0 = 1$, this leads us to choose:

$$p_i = \frac{1}{(\log(\Delta + 1))^i 2^{i^2 - i}}.$$

Thus the expected number of comparisons in going from R_i to R_{i-1} is:

$$\begin{aligned} &E(|R_{i-1}|)E(\text{comparisons for } x \in R_{i-1}) \\ &\quad \le np_{i-1}(2^{2i-1} \log(\Delta + 1) + 2p_{i-1}/p_i) \\ &\quad = \frac{n}{(\log(\Delta + 1))^{i-2} 2^{i^2 - 5i + 2}}. \end{aligned}$$

Now, we require that $E(|R_k|)$ be $O(1)$; for this it is sufficient that $k \ge \sqrt{\log n}$.

We can bound the expected number of comparisons in going from R_k to R_1 by

$$\sum_{i=2}^{k} \frac{n}{(\log(\Delta + 1))^{i-2} 2^{i^2 - 5i + 2}} \le n \sum_{i \ge 2} 2^{-i^2 + 5i - 2} < 37n.$$

(This bound is independent of Δ: this is possible because Δ determined the size of R_1.) Thus the overall number of comparisons is dominated by the transition from R_1 to R_0, and we have:

Theorem 1 *The limited-degree local sorting algorithm requires*

$$E(\textit{comparisons}) \le 4n \log(\Delta + 1) + 37n$$

on a graph with n vertices and maximum degree Δ.

As an example, if the degree of every vertex is polylogarithmic in n, then the procedure runs in $O(n \log \log n)$ comparisons.

Another application is to the *set-sort* problem: sort each of a family of possibly overlapping sets. In fact, set-sort is equivalent to locally sorting the graph in which each set induces a clique.

Set-maxima was introduced in [GYY] where an argument of Fredman was presented showing that, given m sets over an n-element universe, at most $\binom{m+n-1}{n-1}$ arrangements of maxima are possible. We in fact observe an $\Omega(n\log(m/n)+n)$ information-theoretic lower bound matching this limitation. The first randomized algorithm for set-maxima, presented by Kenyon and King [KK] in 1988, used an expected $O((m+n)(\log n)^{1/3})$ comparisons provided m was $O(n(\log n)^c)$. Their algorithm was improved in 1989 by Goddard and Schulman [GS] who, using local sorting, solved set-maxima with an expected $O(n\log\log n)$ comparisons for m in the same range.

In the present paper we describe a randomized algorithm for set-maxima which uses an expected $O(n\log(m/n)+n)$ comparisons. By the above this is optimal.

Set-maxima can be applied to the *partial order verification* problem suggested by A. Yao [Y]: given a directed graph G with elements of a total order at each of the n vertices, verify that the orientation of every edge agrees with the order. In the reduction each set consists of the immediate predecessors of a vertex of the graph. Thus our set-maxima algorithm yields an $O(n)$ comparison algorithm for this problem.

Komlós [K] considered the *minimum spanning tree verification* problem, and solved it by a reduction to set-maxima. His deterministic algorithm finds the tree edge of largest weight in every simple cycle containing exactly one non-tree edge. It solves this instance of set-maxima in $O(n\log(m/n))$ comparisons, where n and m are the number of tree and non-tree edges. Then it verifies that the every non-tree edge is greater than the appropriate maximum. We have been informed of an $O(m+n)$ time implementation of this algorithm by Tarjan [T].

Recently Bar-Noy, Motwani, and J. Naor [BMN] gave deterministic algorithms which run in time $O(n)$ for two cases of set-maxima on n sets. The first is where the n sets are the hyperplanes in a projective geometry. In the second, the n sets are chosen randomly, so that each element appears in each set with probability $p(n)$.

We will proceed in §2 to describe a randomized algorithm for local sorting which, for a graph with n vertices and maximum degree Δ, uses an expected $O(n\log(\Delta+1))$ comparisons. In §3 we obtain our estimate of $\alpha(G)$. Then in §4 we provide a deterministic procedure, involving no comparisons, to reduce the local sorting of an arbitrary graph to that of a graph H in which each component H_i has maximum degree a_i (say). Local sorting of H is achieved in $O(\sum_i |H_i|\log(a_i+1))$ comparisons using the algorithm of §2. We show that this number is $O(\log\alpha(G))$.

In §5 we explain the outline of our set-maxima algorithm. We present it in detail in §6. In §7 we analyze the algorithm and show that it uses an expected $O(n\log(m/n)+n)$ comparisons for m sets from an n-element universe.

Finally in §8 we discuss some open problems.

2 Local Sorting for Graphs of Limited Degree

We show here how to locally sort in a graph with vertex set X ($|X|=n$) and maximum degree Δ. The idea is to take a series of increasingly large random samples of the vertices and partially order each sample using the information given by the partial order on the previous one.

We will present our results under the assumption that the values at the vertices are distinct. Minor modifications suffice to cover the more general case where equalities are allowed, and "local sorting" includes reporting the equalities as such.

2.1 Algorithm

We produce a series of samples $R_k, R_{k-1}, \ldots, R_0 = X$. We then:

- Partially order each R_i so that it is *locally sorted to radius* 2^{2i}.

By this we mean that the relative order of vertices in R_i is known if, as measured in G, they are within distance 2^{2i} of each other. The end result is that $R_0 = X$ is locally sorted to radius 1, as called for.

We construct each sample of expected size np_i by choosing each vertex independently with probability p_i.

Optimal Randomized Algorithms for Local Sorting and Set-Maxima

Wayne Goddard
Massachusetts Institute of Technology

Valerie King
University of Toronto

Leonard Schulman
Massachusetts Institute of Technology

Abstract

We present randomized algorithms for two sorting problems. In the local sorting problem, a graph is given in which each vertex is assigned an element of a total order, and the task is to determine the relative order in every pair of adjacent vertices. In the set-maxima problem, a collection of sets whose elements are drawn from a total order is given, and the task is to determine the maximum element in each set. We describe lower bounds for the problems in the comparison model, and show that the algorithms are optimal within a constant factor.

1 Introduction

In this paper we study two sorting problems. The first is the *local sorting* problem: given a graph in which each vertex is assigned an element of a total order, one must determine the relative order in every pair of adjacent nodes. This problem restricts to standard sorting when the graph is complete, but in general its complexity depends on the graph selected. The second problem is *set-maxima*, where the task is to identify the maximum element in each of a collection of sets drawn from a total order. Set-maxima was investigated in [GYY], [K], [KK] and [BMN]. Local sorting appears to be new, although a restricted version was suggested in [L].

We present randomized algorithms for these problems and measure their complexity in the comparison (decision-tree) model. In this model, one pays only for comparisons between elements of the total order. The complexity of the algorithm is the expected number of comparisons made on a worst-case input. We also obtain information-theoretic lower bounds for each of these problems, and show that our algorithms attain these bounds. In contrast to the situation for randomized algorithms, no nontrivial deterministic algorithms are known for local sorting or set-maxima on arbitrary graphs or set systems.

An information-theoretic lower bound on the complexity of local sorting is given by the logarithm of the number, $\alpha(G)$, of acyclic orientations of the graph G. We present an algorithm which is optimal—it makes an expected $\Theta(\log \alpha(G))$ comparisons.

We derive an estimate of $\alpha(G)$ in terms of the degree sequence of G: specifically $\log \alpha(G)$ is $\Theta(\sum_{v \in G} \log(d_v + 1))$, where d_v is the degree of vertex v in G. Thus the number of comparisons our algorithm makes contrasts with the $\Theta(\sum_{v \in G} d_v)$ comparisons made by the naive algorithm which compares along all edges.

One application of local sorting is to the *element uniqueness problem on a graph*: given a graph G with each vertex assigned an element of a total order, are all pairs of adjacent values distinct? Manber and Tompa [MT] introduced this question and showed that $\log \alpha(G)$ comparisons are necessary for this decision problem; up to a constant factor, this is the bound achieved by our local sorting procedure (which can identify every equality).

The first and third authors' research was supported by NSF grant 8912586-CCR, DARPA contract N00014-89-J-1988, and the third author's ONR Graduate Fellowship. They may be contacted at the Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139. The second author was supported by a grant from NSERC and ITRC. She may be contacted at the Department of Computer Science, Toronto, Canada M5S 1A4.