Sorting Networks of Logarithmic Depth, Further Simplified

Joel Seiferas

August 8, 2006; revised: July 12, 2007

Abstract. We further simplify Paterson's version of the Ajtai–Komlós–Szemerédi sorting network, and its analysis, mainly by tuning the invariant to be maintained.

1. Introduction

For each n, the celebrated sorting network of Ajtai, Komlós, and Szemerédi [1–2] permutes into sorted order an arbitrarily scrambled sequence of length n in just $\mathcal{O}(\log n)$ oblivious parallel steps. Each of the steps involves simultaneous "comparator" operations on disjoint pairs of registers, with each such operation permuting the contents of its register pair into order. Every step is completely prearranged and not at all adaptive to the particular length-n sequence. Since the total number of two-outcome *comparisons* involved is $\mathcal{O}(n \log n) = \mathcal{O}(\log_2 n!)$, the result is within a constant factor of the smallest possible number of such parallel steps.

Although the $\mathcal{O}(\log n)$ depth of the AKS sorting network is less by a factor of $\Theta(\log n)$ than that of the other famous sorting networks—the elegant networks of Batcher [6]—the hidden constant far exceeds this factor for mundane values of n, and the expositional complexity is much higher. For such reasons, there have been efforts to improve or more simply describe the AKS networks [2, 4, 7, 8, 10, 13–16]. Paterson's version [14] has been the most successful and influential, serving as the basis for most of the others.

Our version is aimed at simplicity, and not at depth reduction. Some of our simplifications, such as combining related parameters, will actually *impede* depth reduction. On the whole, however, a rationale from Paterson's abstract does again apply here: "While the constants obtained for the depth bound still prevent the construction being of practical value, the structure of the presentation offers a convenient basis for further development."

The version we describe here simplifies Paterson's in several ways: The general setting and loop invariant have been carefully tuned so that they are cleaner and easier to work with, and so that the algorithm can follow the same simple procedure on every iteration, without regard for special initial or final stages. Being similar enough to ε and λ, respectively, δ and μ have been eliminated as independently named parameters. The invariant now uses capacities only as *upper* bounds,

Key words and phrases. sorting network, comparison network, comparator network, oblivious sorting, parallel sorting, analysis of algorithms.

I thank Mike Paterson for encouragement, and Piotr Faliszewski for careful reading and helpful feedback, and Reviewer #2 for stimulating a final round of correction, clarification, and simplification.

Typeset by $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX

eliminating concern on whether or when bags are full to capacity. The bags corresponding to singleton intervals have been eliminated, so that rounding becomes a minor issue, not requiring any special patching. The "cold storage" patch is no longer needed.

Although much of Paterson's approach remains unchanged, the version here is intended to stand alone, without necessary reference to any previous version. More on comparator networks can be found in standard textbooks [9, 11].

2. The general approach

The AKS algorithm is based on a "sorting-by-splitting" or "sorting-by-classifying" approach that amounts to an ideal version of "quicksort" [9, 11]: Separate the items to be sorted into a smallest half and a largest half, and continue recursively on each half. But this seems to require shallow networks to obliviously separate the halves of scrambled sequences of items. Such networks do not exist [5, 15], so we resort to imperfect versions, of constant-depth, and a strategy that can undo the errors as it proceeds.

Without loss of generality, assume we have n distinct items to sort, where n is a nontrivial power of 2. To keep track of the items, we consider each of them always to occupy one of $n-1 = 1+2+4+\cdots+n/2$ *bags*—one bag corresponding to each nonsingleton *binary subinterval* of the length-n index sequence: 1 whole, its first and second halves (2 halves), their first and second halves (4 quarters), ... , their first and second halves ($n/2$ contiguous pairs). These binary subintervals, and hence the associated bags, correspond nicely to the nodes of a complete binary tree, with each nontrivial binary subinterval serving as *parent* for its first and second halves, which serve respectively as its *left child* and *right child.* (The sets, and even the numbers, of *registers* (the "wires" in the standard representation of such comparator networks) corresponding to these bags will change with time, according to time-determined baggings and rebaggings by the algorithm.)

Based on its actual rank, each item can be considered *native* to one bag at each level of the tree. For example, if it lies in the second quartile, it is native to the second of the four bags that are grandchildren of the root. Sometimes an item will occupy a bag to which it is *not* native, where it is a *stranger*; more specifically, it will be j*-strange* if its bag is *at least* j steps off its native path down from the root. (So the 1-strangers are *all* the strangers; and the *additional* 0-strangers are actually native, and not strangers at all.)

Initially, we consider all n items to be in the root bag (the one corresponding to the whole sequence of indices), to which all are native. We design a schedule, oblivious to the particular data, of applications of certain comparator networks and conceptual rebaggings of the results that is guaranteed to leave all items in the bags of constant-height subtrees to which they are native. Then we will be able to finish by applying a separate sorting network to the $\mathcal{O}(1)$ items in the bags of each of these constant-size subtrees.

3. The simplified sorting network

Each stage of our network acts separately on the contents of each nonempty bag, whose contents occupy an inductively predictable subsequence of the n registers. In terms of the bag's current "capacity" b (an upper bound on its number of items), a certain fixed fraction λ, and other notions and parameters to be described and chosen later (in retrospect), it applies an *approximate separator* to that sequence of registers, and it evacuates the results to the parent and children bags as follows: If there is a parent, then separate and "kick back" to it $\lfloor \lambda b \rfloor$ items (or as many as are available) from each end of the results. If the excess (between these ends) is odd (which will be inductively impossible at the root), then kick back any one additional item (the middle result, say) to the parent. Send the first and second halves of any remaining excess down to the respective children. Note that this plan can fail to be feasible in only one case: the number of items to be evacuated exceeds $2\lfloor\lambda b\rfloor + 1$, but the bag has no children (i.e., it is a leaf).

The approximate separator, of depth that will depend only on λ and a certain small positive fraction ε, will do a slightly rough job of permuting smaller and larger items, respectively, into the first and second halves of its registers, with ε parameterizing the relative error in a way that will suffice for our argument in Section 5. The parameter b is an imposed *(dynamic) capacity* that will increase with the depth of the bag in the tree but decrease with time.

Here is a preview of one set of adequate parameters:

$$\lambda = \varepsilon = 1/99 \text{ and } b = n \cdot 10^d (.65)^t,$$

where $d \geq 0$ is the depth and $t \geq 0$ is the number of previous stages. If we adopt these parameters, then it will turn out that we can continue until $b_{\text{leaf}} < 99$, at which time every item will be in a height-1 subtree to which it is native, so that we can finish the job with disjoint 4-sorters. Except for the approximate separators, which we review for completeness in Section 6, this already completes a description of a network of depth $\mathcal{O}(\log n)$ to sort n items. Sections 4 and 5 provide the proof.

4. The tuned invariant

We carefully reintroduce our parameters here, planning to accumulate for later examination the constraints on them needed for our analysis to work out. For A comfortably larger than 1 (10 in our preview above) and ν less than but close to 1 (.65 in our preview above), define the *(dynamic) capacity* (b above) of a depth-d bag after t stages to be $n\nu^t A^d$. (Again note the dynamic reduction with time, so that this capacity eventually becomes small even compared to the number of items native to the bag.) Let $\lambda < 1$ be chosen as indicated later, and let $\varepsilon > 0$ be small.

Subject to the constraints we accumulate in Section 5, we show there that each successful iteration of the separation–rebagging procedure described in Section 3 (i.e., each stage of the network) preserves the following four-clause invariant.

(1) Alternating levels are entirely empty.
(2) On each level, the number of items currently in each bag (or in the entire subtree below) is the same (and hence at most the size of the corresponding binary interval).
(3) The number of items currently in each bag is bounded by the current capacity of the bag.
(4) The number of j-strangers ($j = 1, 2, 3, \ldots$) currently in each bag is bounded by $\lambda\varepsilon^{j-1}$ times the bag's current capacity.

How much successful iteration is enough? Until the leaf capacities b dip below the constant $1/\lambda$. At that point, the subtrees of smallest height k such that $(1/\lambda)(1/A)^{k+1} < 1$ contain all the items (since $b/A^{k+1} < (1/\lambda)(1/A)^{k+1} < 1$), correctly classified (since the number of $(k-i+1)$-strangers in each bag at each height $i \leq k$ is bounded by $\lambda\varepsilon^{k-i} b/A^i \leq \lambda b < 1$), so that we can finish the job with independent 2^{k+1}-sorters, of depth at most $(k+1)(k+2)/2$ [6]. (For the right choice of parameters (holding $1/\lambda$ to less than A^2), k as small as 1 can suffice, so that we can finish the job with mere 4-sorters, of depth just 3.) So the number t of successful iterations need only exceed $\big((1+\log_2 A)/\log_2(1/\nu)\big)\log_2 n - \log_2(A/\lambda)/\log_2(1/\nu) =$

$\mathcal{O}(\log n)$, since that is (exactly) enough to get the leaf capacity $n\nu^t A^{(\log_2 n)-1}$ down to $1/\lambda$.

As we noted in the previous section, only one thing can prevent successful iteration: $2\lfloor \lambda b \rfloor + 1$ being less than the number of items to be evacuated from a bag with current capacity b but with no children. Since such a bag is a leaf, it follows from Clause (2) of the invariant that the number of items is at most 2. Thus the condition is $2\lfloor \lambda b \rfloor + 1 < 2$, implying $b < 1/\lambda$, which we have just noted is already sufficient for our goal.

Therefore, it remains *only* to show that each successful iteration does preserve the entire invariant.

5. Argument that the invariant is maintained

We adapt or simplify Paterson's main arguments to the setting of our simplified algorithm and tuned invariant.

Only Clauses (3) and (4) are not immediately clear. We first consider the former—that capacity will continue after the next iteration to bound the number of items in each bag. This is nontrivial only for a bag that is currently empty. If the current capacity of such a bag is $b \geq A$, then the next capacity can safely be as low as

$$\begin{aligned}
&(\text{no. items from below}) + (\text{no. items from above})\\
&\qquad \leq (4\lambda b A + 2) + b/(2A)\\
&\qquad = b(4\lambda A + 1/(2A)) + 2\\
&\qquad \leq b(4\lambda A + 1/(2A) + 2/A), \text{ since } b \geq A.
\end{aligned}$$

In the remaining Clause (3) case, the current capacity of the empty bag is $b < A$. Therefore, all higher bags' capacities are bounded by $b/A < 1$, so that the n items are equally distributed among the subtrees rooted on the level below, an even number of items per subtree. Since each root on that level has passed down an equal net number of items to each of its children, it currently holds an even number of items and will not kick back an odd item to our bag of interest. In this case, therefore, the next capacity can safely be as low as just $4\lambda b A$.

In either Clause (3) case, therefore, any $\boxed{\nu \geq 4\lambda A + 5/(2A)}$ will work.

Finally we turn to restoration of Clause (4). First we consider the relatively easy case of $j > 1$. Again, this is nontrivial only for a bag that is currently empty. Suppose the current capacity of such a bag is b. How can we bound the number of j-strangers after the next step? It is at most

$$\begin{aligned}
&(\textit{all } (j+1)\text{-strangers currently in children})\\
&\quad + ((j-1)\text{-strangers currently in parent, } \textit{and not filtered out by the separation})\\
&\qquad < 2bA\lambda\varepsilon^j + \varepsilon((b/A)\lambda\varepsilon^{j-2})\\
&\qquad \leq \lambda\varepsilon^{j-1}\nu b, \text{ provided } \boxed{2A\varepsilon + 1/A \leq \nu}.
\end{aligned}$$

Note that the bound $\varepsilon((b/A)\lambda\varepsilon^{j-2})$ also places our first constraint on the ("filtering") performance of an approximate separator: Roughly, at most fraction ε of the "few" smallest (or largest) are permuted "far" out of place. More precise sufficient specifications will be the subject of Section 6.

All that remains is the more involved Clause (4) case of $j = 1$. Consider any currently empty bag B, of current capacity b. At the root there are always *no* strangers; so assume B has a parent, D, and a sibling, C. Let d be the number of items in D.

There are three sources of 1-strangers in B after the next iteration, two previously strange, essentially as above, and one newly strange:

1. Current 2-strangers at children (at most $2\lambda\varepsilon bA$).
2. Unfiltered current 1-strangers in D (at most $\varepsilon(\lambda(b/A))$, as above).
3. Items in D that are native to C but that now get sent to B instead.

For one of these last items to get sent down to B, it must get permuted by the approximate separator into "B's half" of the registers. The number that do is at most the number of C-native items in excess of $d/2$, plus the number of "halving errors" by the approximate separator. By a second natural constraint on our approximate separators, the latter is at most $\varepsilon b/(2A)$, leaving only the former to estimate.

For this remaining estimate, we *compare* the current "actual" distribution with a more symmetric "benchmark" distribution that has an unchanged number of items in each bag, but that, for each bag C' on the same level as B, has only C'-native items below C' and has $d/2$ C'-native items in the parent D' of C'. (If d is odd, then the numbers of items in D' native to its two children will be $\lfloor d/2 \rfloor$ and $\lceil d/2 \rceil$, in the order of our choice.) That there *is* such a redistribution follows from Clause (2) of the invariant: Start by partitioning the completely sorted list among the bags on B's level, and then move items down and up in appropriate numbers to fill the budgeted vacancies.

In the benchmark distribution, the number of C-native items in excess of $d/2$ is 0. If our actual distribution is to have an excess, where can the excess C-native items come from, in terms of the benchmark distribution? They can come only from C-native items on levels above D's and from a net reduction in the number of C-native items in C's subtree. The latter can only be via the introduction into C's subtree of items *not* native to C. By Clause (4) of the invariant, the number of these can be at most

$$\begin{aligned}
&2\lambda\varepsilon bA + 8\lambda\varepsilon^3 bA^3 + 32\lambda\varepsilon^5 bA^5 + 128\lambda\varepsilon^7 bA^7 + \dots \\
&\quad = 2\lambda\varepsilon bA(1 + (2\varepsilon A)^2 + ((2\varepsilon A)^2)^2 + ((2\varepsilon A)^2)^3 + \dots) \\
&\quad < 2\lambda\varepsilon bA/(1-(2\varepsilon A)^2).
\end{aligned}$$

If 2^i is the number of bags C' on the level of C, then the *total* number of items on levels above D's is at most

$$2^{i-3}b/A^3 + 2^{i-5}b/A^5 + 2^{i-7}b/A^7 + \dots.$$

Since the number native to each such C' is the same, the number native to C is at most $1/2^i$ times as much:

$$\begin{aligned}
&b/(2A)^3 + b/(2A)^5 + b/(2A)^7 + \dots \\
&\quad < b\Big/\Big((2A)^3(1-1/(2A)^2)\Big) \\
&\quad = b/(8A^3 - 2A).
\end{aligned}$$

So the total number of 1-strangers from all sources is at most

$$2\lambda\varepsilon bA + \varepsilon(\lambda(b/A)) + \varepsilon b/(2A) + 2\lambda\varepsilon bA/(1-(2\varepsilon A)^2) + b/(8A^3-2A),$$

This total is indeed bounded, as needed, by $\lambda\nu b$ (λ times the new capacity), provided

$$\boxed{2\lambda\varepsilon A + \varepsilon\lambda/A + \varepsilon/(2A) + 2\lambda\varepsilon A/(1-(2\varepsilon A)^2) + 1/(8A^3-2A) \le \lambda\nu}.$$

This completes the argument, subject to (Section 6's design of approximate separators and) the following accumulated constraints:

$$\begin{aligned}
&A > 1,\\
&\nu < 1,\\
&\varepsilon > 0,\\
&\lambda < 1,\\
&\nu \geq 4\lambda A + 5/(2A),\\
&\nu \geq 2A\varepsilon + 1/A,\\
&\lambda\nu \geq 2\lambda\varepsilon A + \varepsilon\lambda/A + \varepsilon/(2A) + 2\lambda\varepsilon A/(1-(2\varepsilon A)^2) + 1/(8A^3 - 2A).
\end{aligned}$$

Here is one choice order and strategy that works out neatly:

1. Choose A big.
2. Choose λ between $1/(8A^3 - 2A)$ and $1/(8A)$.
3. Choose ε small.
4. Choose ν within the resulting allowed range.

For example, $A = 10$, $\lambda = 1/100$, $\varepsilon = 1/100$, and $\nu = 13/20$. In fact, if we perturb λ and ε to $1/99$, then we can hold the parameter "k" of Section 4 to 1 and get by with 4-sorters at the very end, as previewed in Section 3.

Note: Seeing that $\lambda = \varepsilon$ is feasible, we could actually commit to that in order to eliminate one more parameter, say λ. Then Clause (4) of the invariant could be extended to the case $j = 0$ to neatly subsume Clause (3), and the simplified constraint $\nu \geq 4\epsilon A + 5/(2A)$ would subsume the constraint $\nu \geq 2A\varepsilon + 1/A$. We have chosen to keep both λ and ε, however, since their roles are conceptually so different.

6. The needed approximate separators

To complete the story, we include a sketch of comparator networks that can serve as the approximate separators needed in the preceding section. The simplest notion of approximate separation is approximate *halving*, the criterion for which is a relative bound on the number of misplacements from each prefix or suffix of a completely sorted result to the wrong *half* of the result actually produced. For approximate *separation*, we strengthen the undesirable fraction one-half to even larger fractions ($1 - \lambda$ below). After giving more precise definitions, we sketch the design and use of constant-depth halvers to construct the desired constant-depth separators.

Definitions. For each $\varepsilon > 0$ and $\lambda \leq 1/2$, the criterion for *ε-approximate λ-separation* of a sequence of n input elements is that, for each $\lambda' \leq \lambda$, at most $\varepsilon\lambda' n$ of the $\lfloor \lambda' n \rfloor$ smallest (resp., largest) elements do not get placed among the $\lfloor \lambda n \rfloor$ first (resp., last) positions. For $\lambda = 1/2$, this is called *ε-halving.*

Although these definitions do not restrict n, we will need and use them only for *even n*. When the number of items was *odd* in the preceding section, we could have immediately considered any one of them to be the "odd" one, definitely to be kicked back to the parent bag.

Lemma 1. *For each $\varepsilon > 0$, ε-halving can be performed by comparator networks (one for each even n) of constant depth (independent of n).*

Lemma 2. *For each $\varepsilon > 0$ and $\lambda \leq 1/2$, ε-approximate λ-separation can be performed by comparator networks (one for each even n) of constant depth (independent of n). As a function of λ (fixing ε), the depth is $\mathcal{O}(\log 1/\lambda)$.*

In our analysis in the preceding section, we assumed *both* ε-approximate λ-separation *and* ε-halving (in constant depth), neither of which quite implies the other. To get the *combination*, first use Lemma 1, and then carefully use Lemma 2, "once for the sake of each half": once simplified as if the second half of the n elements were all some very large element, and once simplified as if the first half of the n elements were all some very small element, leaving out comparators involving the very large or very small elements.

Actually, we need the above combination for *various* fractions "λ," each at least as large as the chosen one, for the following reason: Our kick-back numbers were based on *capacity*, but our bags may not have been full to capacity. (Even fractions greater than 1/2 can result; but then all items will be moved to the parent bag, completely obviating the need for any special approximate separator.) The deepest total depth from our lemmas, however, will be the one for the chosen λ: $\mathcal{O}(1) + \mathcal{O}(\log 1/\lambda) = \mathcal{O}(\log 1/\lambda) = \mathcal{O}(1)$, as required.

Proof sketch for Lemma 1. From the (omitted) study of "expander graphs" [15], start with a constant d, determined by ε, and a d-regular $n/2$-by-$n/2$ bipartite graph with the following expansion property: Each subset S of either part has more neighbors than the smaller of $|S|(1-\varepsilon)/\varepsilon$ and $(1-\varepsilon)n/2$.

The ε-halver uses a comparator corresponding to each edge in the expander graph. This requires depth only d, by repeated application of a minimum-cut argument [9, Exercise 26.3-5, for example].

To see that the result is indeed an ε-halver, consider the final positions of the strays from the $m \leq n/2$ smallest elements. Those positions *and all their neighbors* must finally contain elements among the m smallest. If the fraction of strays were more than ε, then this would add up to more than m, a contradition. $\square$

Proof sketch for Lemma 2. For ε_0 small in terms of ε and λ, make use of the ε_0-halvers already provided by Lemma 1 (the result for $\lambda = 1/2$).

First apply the ε_0-halver for length n to the whole sequence, and then work separately on each resulting half.

The halves are handled symmetrically, in parallel; so let us focus on the first half. In terms of $m = \lfloor \lambda n \rfloor$ (assuming $n \geq 1/\lambda$), apply ε_0-halvers to the $\lceil \log_2(n/m) \rceil - 1$ prefixes of lengths $2m, 4m, 8m, \dots, 2^{\lceil \log_2(n/m)-1 \rceil}m$, in reverse order, where the last one listed (first one performed) is simplified as if the inputs beyond the first $n/2$ were all some very large element. Then the total number of elements from the smallest $\lfloor \lambda' n \rfloor$ (for any $\lambda' \leq \lambda$) that do not end up among the first $\lfloor \lambda n \rfloor = m$ positions is at most $\varepsilon_0 \lambda' n$ in each of the $\lceil \log_2(n/m) \rceil$ intervals $(m, 2m]$, $(2m, 4m]$, $(4m, 8m]$, $\dots$, $(2^{\lceil \log_2(n/m)-2 \rceil}m, n/2]$, and $(n/2, n]$, for a total of at most $\lceil \log_2(n/m) \rceil \varepsilon_0 \lambda' n$.

For the $c < 1$ of our choice (close to 1, making $\log_2(1/c)$ close to 0), we have the following: Unless n is small $(n < (1/\lambda)(1/(1-c)))$,

$$2^{1+\log_2(n/m)} = 2(n/m) = 2n/\lfloor \lambda n \rfloor < 2n/(\lambda n - 1) \leq 2n/(c\lambda n) = (2/c)(1/\lambda),$$

so that $\log_2(n/m)$ is less than $\log_2(1/c) + \log_2(1/\lambda)$, and the number of misplaced elements above is at most $\lceil \log_2(1/c) + \log_2(1/\lambda) \rceil \varepsilon_0 \lambda' n$. This is bounded by $\varepsilon \lambda' n$ as required, provided we chose $\boxed{\varepsilon_0 \leq \varepsilon / \lceil \log_2(1/c) + \log_2(1/\lambda) \rceil}$. $\square$

Note that the particular construction given for Lemma 2 itself follows the strategy sketched above for our *combination* of Lemmas 1 and 2, so that the result is itself already also an ε-halver, and so that the results for larger values of λ are no deeper.

7. Discussion

The goal here has been to give a newly clear and simple modular demonstration of the remarkable big-$\mathcal{O}$ version of the AKS result [1–2]. We have omitted only a proof of the existence of expander graphs, a general lemma with a variety of widely available approaches and proofs, some nonconstructive and some explicit.

Although we have not aimed to achieve a small big-$\mathcal{O}$ constant, it may be instructive now to bring together an estimate of the constant(s) we do obtain, especially in terms of the various parameters, in order to focus on tradeoffs and bottlenecks that could be addressed.

Of course iteration of the separation–rebagging procedure accounts for almost all of the network depth. We have seen that the number of iterations of that procedure need only exceed

$$\big((1+\log_2 A)/\log_2(1/\nu)\big)\log_2 n - \log_2(A/\lambda)/\log_2(1/\nu),$$

and that the network depth per iteration is at most $\lceil\log_2(1/c)+\log_2(1/\lambda)\rceil$ times the depth of an $\varepsilon/\lceil\log_2(1/c)+\log_2(1/\lambda)\rceil$-halver, for $\log_2(1/c)$ as close to 0 as we like. The depth of the final Batcher sorters can be bounded by $k(k+1)/2$ for $k=\lceil\log(1/\lambda)/\log A)\rceil$. So the only remaining issue is the depth of the approximate halvers, which depends on the cited expanders.

For our sample choice of parameters ($A=10$, $\lambda=\varepsilon=1/100$, $\nu=13/20$), this estimate evaluates to about $6.95\log_2 n$ iterations, each seven $1/700$-halvers deep. An alternative of $A=6.785$, $\lambda=1/(96.4)$, $\varepsilon=1/(57.45)$, $\nu=13/20$ reduces the number of iterations to about $6.05\log_2 n$, each seven $1/(402.15)$-halvers deep.

The shallowest halvers can be produced by more direct arguments, rather than via expander graphs. By a direct and careful counting argument, Paterson [14] (the theorem in his appendix, specialized to the case $\alpha=1$) proves that the depth of an ε-halver need be no more than $\lceil(2\log\varepsilon)/\log(1-\varepsilon)+2/\varepsilon-1\rceil$. Others [3, 12] have less precisely sketched or promised asymptotically better results, but we do best with relatively modest values of ε—no smaller than, say, 1/1000. For our two values of ε above, Paterson's respective ε-halver depths are 10564 and 5621. The resulting depth bounds for the sorting network are still huge: $514247\log_2 n$ and $238198\log_2 n$, respectively; but redistinguishing some of the parameters we have deliberately combined for simplicity would bring this estimate somewhat closer to Paterson's comparable one of $30000\log_2 n$.

Even better, one can focus on the construction of approximate separators. Paterson [14], for example, notes the usefulness in that construction (as presented above, for Lemma 2) of somewhat weaker (and thus shallower) halvers, which perform well only on extreme sets of sizes bounded in terms of an additional parameter $\alpha\le 1$. Tailoring the parameters to the different levels of approximate halvers in the construction, he manages to reduce the depth of separators enough to reduce the depth of the resulting sorting network to less than $6100\log_2 n$.

Ajtai, Komlós, and Szemerédi [3] announce that design in terms of generalized, *multi-way* comparators (i.e., M-sorter units) can lead to drastically shallower approximate halvers and "near-sorters" (their original version [2] of separators). Chvátal [7] pursues this idea carefully and arrives at an improved final bound less than $1830\log_2 n$, although still somewhat short of the $100\log_2 n$ privately claimed by Komlós.

The tightest analyses have appeared only as sketches in preliminary conference presentations or mere promises of future presentation [3], or as unpublished technical reports [7]; but no recent journal versions have appeared. If unwieldiness of the argument has played a role in delaying such expositions, we hope the simplicity of our current effort will inspire their renewed pursuit, improvement, and completion.

References

1. M. Ajtai, J. Komlós, and E. Szemerédi, *Sorting in $c \log n$ parallel steps*, Combinatorica **3**, 1 (1983), 1–19.
2. M. Ajtai, J. Komlós, and E. Szemerédi, *An $\mathcal{O}(n \log n)$ Sorting Network*, Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, 1983, pp. 1–9.
3. Miklós Ajtai, János Komlós, and Endre Szemerédi, *Halvers and expanders*, Proceedings, 33rd Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, 1992, pp. 686–692.
4. Selim G. Akl, *Parallel Computation: Models and Methods*, Prentice Hall, 1997.
5. V. E. Alekseev, *Sorting algorithms with minimum memory*, Kibernetika **5**, 5 (September–October, 1969), 99–103.
6. K. E. Batcher, *Sorting networks and their applications*, AFIPS Conference Proceedings Volume 32, Spring Joint Computer Conference, Thompson Book Company, 1968, pp. 307–314.
7. V. Chvátal, *Lecture notes on the new AKS sorting network*, DCS-TR-294 (October, 1992), Computer Science Department, Rutgers University.
8. Richard Cole and Colm Ó'Dúnlaing, *Note on the AKS sorting network (expository note)*, Technical Report #243 (1986), Computer Science Department, New York University.
9. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to Algorithms*, Second Edition, The MIT Press, 2001.
10. Alan Gibbons and Wojciech Rytter, *Efficient Parallel Algorithms*, Cambridge University Press, 1988.
11. Donald E. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching*, Second Edition, Addison–Wesley, 1998.
12. H. Manos, *Construction of halvers*, Information Processing Letters **69**, 6 (26 March 1999), 303–307.
13. Ian Parberry, *Parallel Complexity Theory*, Pitman Publishing, 1987.
14. M. S. Paterson, *Improved sorting networks with $\mathcal{O}(\log N)$ depth*, Algorithmica **5**, 1 (1990), 75–92.
15. Nicholas Pippenger, *Chapter 15: Communication networks*, Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (Jan van Leeuwen, ed.), Elsevier/The MIT Press, 1990, pp. 805–833.
16. Justin R. Smith, *The Design and Analysis of Parallel Algorithms*, Oxford University Press, 1993.

Computer Science Department, University of Rochester, Rochester, New York, U. S. A. 14627-0226
E-mail address: `joel@cs.rochester.edu`