Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information Processing Letters 94 (2005) 43–47
Information Processing Letters
www.elsevier.com/locate/ipl

A generalization of the 0–1 principle for sorting ☆

Sanguthevar Rajasekaran [a,*], Sandeep Sen [b,1]

[a] *257 ITE Building, Department of CSE, University of Connecticut, Storrs, CT 06269, USA*
[b] *Department of Computer Science and Engineering, IIT Delhi, New Delhi 1100116, India*

Received 17 March 2004; received in revised form 1 September 2004

Available online 16 January 2005

Communicated by S.E. Hambursch

Abstract

The traditional *zero–one* principle for sorting networks states that "if a network with n input lines sorts *all* 2^n binary sequences into nondecreasing order, then it will sort any arbitrary sequence of n numbers into nondecreasing order". We generalize this to the situation when a network sorts *almost all* binary sequences and relate it to the behavior of the sorting network on arbitrary inputs. We also present an application to mesh sorting.

Keywords: Sorting; 0–1 principle; Meshes; Average case perfomance; Analysis of algorithms; Parallel algorithms; Randomized algorithms

1. Introduction

We prove a generalization of the 0–1 principle to sorting networks that sort almost all possible binary sequences. This generalization will be useful in the average case analysis of sorting algorithms as well as in the analysis of randomized sorting algorithms. It may be noted that the 0–1 principle extends to *oblivious* sorting algorithms [3] and although our results are stated in the context of sorting networks, they are applicable to *oblivious* sorting algorithms also. The standard 0–1 principle offers great simplicity in analyzing sorting algorithms—it suffices to assume that the input consists of only zeros and ones. The same level of simplicity is offered by the generalization presented in this paper as well.

As an example application, we consider a sorting algorithm for the mesh and analyze its average case performance using the generalized 0–1 principle. This analysis is very simple and the method is applicable to any *oblivious* sorting algorithms. The generalized 0–1 principle has potential benefits for other sorting algorithms and to the best of our knowledge no formal generalization of the 0–1 principle existed in literature.

☆ This research has been supported in part by the NSF Grants CCR-9912395 and ITR-0326155.
* Corresponding author.
E-mail addresses: rajasek@engr.uconn.edu (S. Rajasekaran), ssen@cse.iitd.ernet.in (S. Sen).
[1] Part of the research done when the author was visiting University of Connecticut and supported by NSF Grant ITR-0326155.

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2004.11.013

Chlebus [1] used it in an ad hoc manner without giving any formalization.

1.1. Some definitions and our results

For a self-contained exposition, we define the terms as used in this paper and review a proof of the 0–1 principle that will be convenient for the presentation of our result.

A *Sorting Network* consists of binary comparator modules where each module has two inputs. Each module compares the two inputs and exchanges them in the output if they are out of order. The subsequent comparisons do not depend on the outcome of any specific comparator. A n sorting network realizes a sorted permutation of any n input configuration.

Although the natural realization of a sorting network is a hardwired circuit with comparators and wires connecting comparators, any sorting algorithm that consists solely of prespecified compare–exchange operations can be thought of as a sorting network and is referred to as an *oblivious* sorting algorithm. In future, these terms will be used interchangeably. Note that many of the common sorting algorithms like mergesort, quicksort, heapsort are not *oblivious* whereas bubblesort is *oblivious*.

The traditional *zero–one* principle for sorting networks states that "if a network with n input lines sorts *all* 2^n binary sequences into nondecreasing order, then it will sort any arbitrary sequence of n numbers into nondecreasing order". (The converse is trivial.) In [8], the author shows that this result cannot be strengthened, i.e., no proper subset of the 2^n sequences can have this property.

Bubblesort [4] and shearsort [9] are two classic examples of the applications of the 0–1 principle. We generalize the zero–one principle to situations when a network sorts *almost all* binary sequences and relate it to the behavior of the sorting network on arbitrary inputs. More specifically, we prove the following theorem in Section 2.

Theorem 1.1 (Generalized 0–1 principle). *Let S_k denote the set of length n binary strings with exactly k 0's $0 \leqslant k \leqslant n$. Then, if a sorting circuit with n input lines sorts at least α fraction of S_k for all k, $0 \leqslant k \leqslant n$, then the circuit sorts at least $(1 - (1 - \alpha) \cdot n)$ fraction of the input permutations of n arbitrary numbers.*

Note that the theorem gives nontrivial bounds only when $\alpha > 1 - 1/n$. In Section 3 we present an application of this result to sorting on an $n \times n$ mesh. The algorithm attains the same (optimal) bound as the algorithms of Chlebus [1] and Gu and Gu [2], but the analysis is relatively simpler and cleaner because of the generalized 0–1 principle. It is very likely that this theorem will have further applications to design and analysis of sorting algorithms.

2. Proof of the main result

Definition. A string $s \in \{0, 1\}^n$ is a k-string if it has exactly k 0's , $0 \leqslant k \leqslant n$. The set of all k-strings for a fixed k will be denoted by S_k.

Clearly the S_ks are pairwise disjoint and their union consists of all the 2^n length n strings over $\{0, 1\}$.

Let $A^{(t)}, B^{(t)}$ be two totally ordered *multisets* of n elements each. A mapping $f : A^{(t)} \to B^{(t)}$ is *monotone* if for all $x, y \in A^{(t)}$ and $x \leqslant y$, $f(x) \leqslant f(y)$.

Observation. If $f : A^{(t)} \to B^{(t)}$ is monotone then the inverse of f is also monotone.

Given $I_n = \{1, 2, \ldots, n\}$, the only monotone function between I_n and the multiset $\{0^k, 1^{n-k}\}$ is given by $f_k(j) = 0$ for $j \leqslant k$ and 1 otherwise. We denote the extension of f to a sequence $a^{(t)} = (a_1, a_2, \ldots, a_t)$ by $f^{(t)}(a^{(t)}) = (f(a_1), f(a_2), \ldots, f(a_t))$. The correctness of the standard zero–one principle is often argued based on the following elegant result (see Knuth [4] for a proof).

Lemma 2.1. *For any n input sorting circuit $C^{(n)}$ and a monotone function f,*

$$f^{(n)}\big(C^{(n)}(a^{(n)})\big) = C^{(n)}\big(f^{(n)}(a^{(n)})\big).$$

Remark. For the sake of simplicity, we will avoid using superscripts to denote sequences when it is clear from the context. For instance the previous lemma can be restated as $f(C(a)) = C(f(a))$.

From our previous observation f_k^{-1} is also monotone. From the previous lemma it follows that

Lemma 2.2. *If a sorting network C correctly sorts $f_k(\sigma)$ for all k, for an input permutation σ, then it correctly sorts σ.*

Proof (*sketch*). Suppose i and j $(j > i)$ are interchanged in σ. Consider the binary string $f_i(\sigma)$. Since $f_i(C(\sigma)) = C(f_i(\sigma))$, the circuit C does not sort $f_i(\sigma)$ correctly leading to a contradiction. □

We now turn our attention towards generalizing this argument. For convenience, we define a bipartite graph G_k with $n!$ elements in one set and $|S_k|$ on the other (for each k). The edges of this graph are defined by the mapping between permutations of I_n and strings $a \in S_k$ such that $a_i = f_k(\Pi(i))$, $1 \leqslant i \leqslant n$, for a permutation Π. Here $\Pi(i)$ denotes the element of I_n that is mapped to i. Note that a single permutation maps to exactly one string in S_k, so by simple counting arguments, it follows that

Lemma 2.3. *Each set in the bipartite graph G_k has vertices with equal degree. In particular, the vertices representing S_k have degrees equal to $\frac{n!}{|S_k|}$.*

In graph G_k we mark all the nodes corresponding to the permutations that are not sorted correctly and likewise we mark the nodes of S_k that are not sorted correctly. For a fixed k, if the circuit does not sort $\beta_k|S_k|$ $(\beta_k < 0)$ strings it does not sort at most $\beta_k|S_k| \cdot \frac{n!}{|S_k|}$ permutations. Therefore the total fraction of permutations (over all values of $1 \leqslant k \leqslant n$) that may not get sorted correctly is bounded by $\beta \cdot n$ where $\beta = \max_{i=1}^{k} \beta_k$. Note that we do not have to consider $k = 0$ as there is only one trivial sorted sequence. Setting $\alpha = 1 - \beta$ completes the proof of Theorem 1.1.

An interesting question is if we can improve the bound, namely, the fraction of sorted inputs. The following lemma shows that the fraction of unsorted permutations is at least β.

Lemma 2.4. *If a sorting circuit does not sort some $a \in S_k$ then it does not sort* (*any of the permutations corresponding to*) $f_k^{-1}(a)$.

This can be seen as follows. At least one pair of 0 and 1 in a must be swapped in $C(a)$, say, in positions i and j. The corresponding elements in the inverse map must also have been swapped from Lemma 2.1.

Remark. It is not clear if we can eliminate the multiplicative factor n, namely, are the unsorted permutations disjoint corresponding to the distinct S_ks? Also note that the number of strings in the set $S_p = \bigcup_{i=0.49n}^{0.51n} S_i$ form an overwhelming fraction of all length n binary strings. Even if one can design an ad hoc sorting algorithm that works correctly for S_p (and consequently for a large fraction of all 0–1 inputs), but that sorts only a negligible fraction of $S_{\log n}$ (for example), the algorithm does not sort most input permutations. See the remark following Theorem 3.1.

3. Sorting in an expected $2n + o(n)$ steps on an $n \times n$ mesh

One of the most challenging problems in the context of sorting numbers on an $n \times n$ mesh with one element per processor is to sort them in time $2n$ (plus possibly lower order terms) which is the *distance bound* on the mesh (see [7,10] for detailed surveys). In this problem, in each step, neighboring processors are allowed to communicate and exchange elements but not store more than one element. If we relax the storage requirement, then this time bound can be achieved (see [7,10]). Note that even on the average, the distance bound is $2n - o(n)$. By considering sufficiently large sub-meshes on the opposite corners, it can be seen that at least one of these elements must travel from one sub-mesh to the other with high probability.

A simple modification of the algorithm of [6] gives us a $2n + o(n)$ steps algorithm for sorting on the average and the analysis is based on the generalized 0–1 principle. For completeness, we describe their algorithm briefly. In the remaining description, by $u \times v$ sub-meshes we refer to all the aligned sub-meshes consisting of processors indexed by (x, y) where $iu \leqslant x \leqslant i(u+1) - 1$, $jv \leqslant y \leqslant j(v+1) - 1$ (for some integers i and j). A row is called *dirty* if it consists of both 0's and 1's, otherwise it is *clean*. The significance of this definition stems from the fact that an unsorted mesh contains many dirty rows whereas a sorted mesh (in row major order) contains at most one dirty row. For our algorithm, the sorting order is defined in terms of blocks that are relatively ordered among themselves in a row-major snake-like ordering. The elements within a block can be ordered in any fashion since it does not affect the asymptotic perfor-

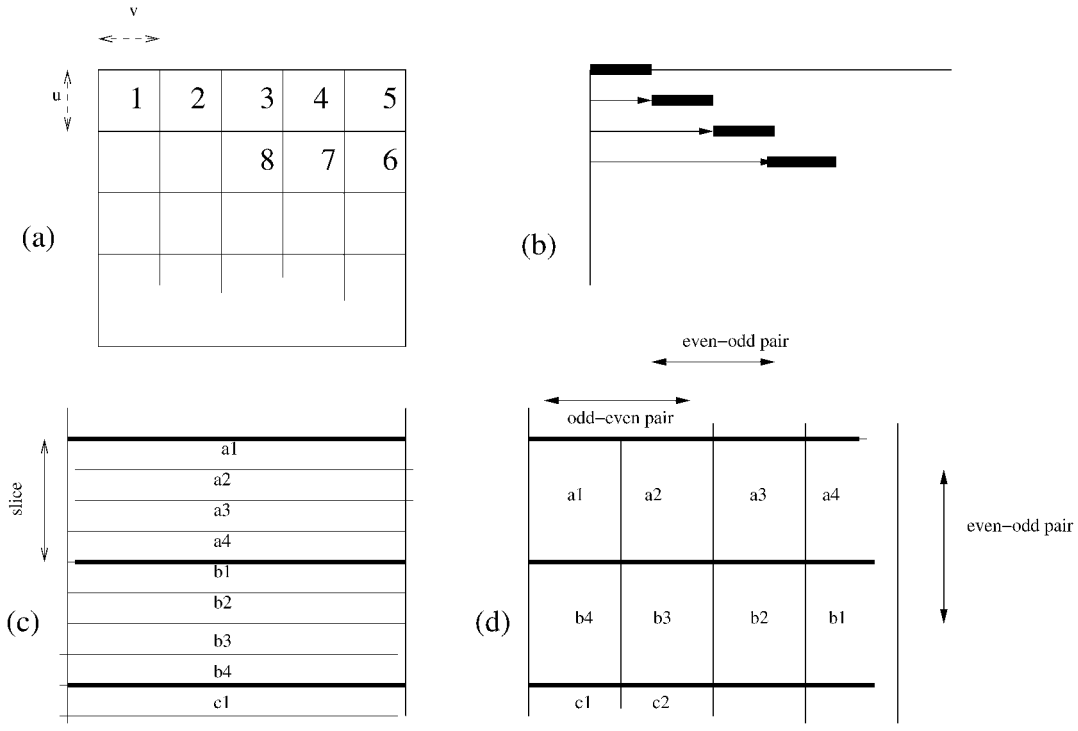

Fig. 1. (a) illustrates $u \times v$ sized blocked snake-like row-major ordering. (b) illustrates distribution of each block by cyclic shifts. (c) illustrates the transformation of slices into blocks in Phase 4. Notice how consecutive slices remain consecutive.

mance and the notion of dirty row is extended to dirty sub-mesh in the obvious manner.

Algorithm MSS [6].

1. Sort all $n^{3/4} \times n^{3/4}$ sub-meshes in a row-major ordering.
2. Distribute each sub-mesh evenly using blocked rotations, i.e., the ith row of every sub-mesh is shifted right by $i \cdot n^{3/4}$ positions.
3. Sort the columns.
4. Transform every $n^{3/4} \times n$ horizontal slice into $n^{1/4}$ $n^{3/4} \times n^{3/4}$ sub-meshes such that consecutive $n^{1/2} \times n$ sub-meshes remain consecutive (within a slice). The row ordering is alternated in every slice so that after transformation, the two dirty sub-meshes are in a snake-like row-major order.
5. Sort pairs of every consecutive (in the blocked snake-like row-major order) sub-meshes—once taking every odd–even pair and then taking even–odd pair.

See Fig. 1 for an illustration of some of the steps.

Each of Phases 2, 3, and 4 can be implemented in n time steps and the others take $o(n)$ time resulting in $3n + o(n)$ time overall (for more details of individual phases see [6]). If we examine each phase more closely, after Phase 3, we have at most two (consecutive) dirty $n^{1/2} \times n$ sub-meshes. These come from dirty rows contributed by each sorted sub-mesh (at most one per sub-mesh) after Phase 2 that become contiguous $\sqrt{n}$ dirty rows following Phase 3.

We modify the above algorithm by eliminating the first two phases, i.e., start with column sort. Then the size of the dirty *band* (contiguous dirty rows) after sorting the columns is the difference between the maximum and the minimum number of 0's among the n columns. Without loss of generality, assume that we have at least $\Omega(n^2)$ 0's—then the expected number of 0's in each column is $\Omega(n)$. From Chernoff [5] bounds it follows that with probability exceeding $1 - \frac{1}{n^\alpha}$, the deviation of the number of 0's is no more than $\theta(\sqrt{\alpha n \log n})$. Therefore, the size of the dirty band

is $\sqrt{c\alpha n \log n} \times n$ for some constant c after sorting columns. By modifying the sizes of the transformed sub-meshes to $(n\sqrt{c\alpha n \log n})^{1/2} \times (n\sqrt{c\alpha n \log n})^{1/2}$, in Phase 4, the 0–1 sequence is correctly ordered in a blocked snake-like indexing after Phase 5. The total number of steps is $2n + o(n)$.

Theorem 3.1. *The (modified) algorithm sorts* $1 - \frac{1}{n^{\alpha}}$ *fraction of all inputs correctly in* $2n + o(n)$ *steps for any fixed* $\alpha > 0$. *With more careful choice of parameters for the sub-mesh sizes the fraction of correctly sorted inputs can be increased to* $1 - 1/2^{n^{\varepsilon}}$ *for some* $\varepsilon > 0$.

Proof The success of this approach depends on the size of the dirty band after the column sort which in turn depends on the discrepancy of the distribution of 0's and 1's among the different columns. The generalized 0–1 principle gives a direct connection between the discrepancy and the fraction of the inputs sorted, viz., it will sort correctly if the discrepancy is not too high. □

Remark. Most 0–1 sequences have balanced number of 0's and 1's and hence we know the rough location of the dirty band following the column sort. Therefore we can clean it by restricting sorting to within the band. This ad hoc approach will be an incorrect application of Theorem 1.1.

References

[1] B.S. Chlebus, Mesh sorting and selection optimal on the average, Comput. and Inform. (Special issue on parallel and distributed computing) 16 (2) (1997).
[2] Q.P. Gu, J. Gu, Algorithms and average time bounds of sorting on a mesh-connected computer, IEEE Trans. Parallel Distrib. Syst. 5 (3) (1994) 308–315.
[3] T. Leighton, Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes, Morgan Kaufmann, San Mateo, CA, 1992.
[4] D.E. Knuth, Sorting and Searching, The Art of Computer Programming, vol. 3, Addison–Wesley, Reading, MA, 1973.
[5] R. Motwani, P. Raghavan, Randomized Algorithms, Cambridge Univ. Press, New York, 1995.
[6] Y. Ma, S. Sen, D. Scherson, The distance bound for sorting on mesh connected processor arrays is tight, in: Proc. 27th Symp. on Foundations of Computer Science, 1986, pp. 255–263.
[7] S. Rajasekaran, Sorting and selection on interconnection networks, DIMACS Ser. Discrete Math. Theor. Comput. Sci. 21 (1995) 275–296.
[8] M.D. Rice, Continuous algorithms, Topology Appl. 85 (1998) 299–318.
[9] I.D. Scherson, S. Sen, A. Shamir, Shear sort: A true two-dimensional sorting technique for VLSI networks, in: Proc. Internat. Conf. on Parallel Processing, 1986, pp. 903–908.
[10] J.F. Sibeyn, Overview of mesh results, Technical Report MPI-I-95-1-018 MPI, Saarbrücken, 1995.