Theoretical Computer Science 110 (1993) 99–129
Elsevier

Time–space tradeoffs for set operations

Boaz Patt-Shamir and David Peleg*

Department of Applied Mathematics, The Weizmann Institute, Rehovot 76100, Israel

Communicated by E. Shamir
Received August 1990
Revised August 1991

Abstract

Patt-Shamir, B. and D. Peleg, Time–space tradeoffs for set operations, Theoretical Computer Science 110 (1993) 99–129.

This paper considers time–space tradeoffs for various set operations. Denoting the time requirement of an algorithm by T and its space requirement by S, it is shown that $TS=\Omega(n^2)$ for set complementation and $TS=\Omega(n^{3/2})$ for set intersection, in the R-way branching program model. In the more restricted model of comparison branching programs, the paper provides two additional types of results. A tradeoff of $TS=\Omega(n^{2-\varepsilon(n)})$, derived from Yao's lower bound for element distinctness, is shown for set disjointness, set union and set intersection [where $\varepsilon(n)=O((\log n)^{-1/2})$]. A bound of $TS=\Omega(n^{3/2})$ is shown for deciding set equality and set inclusion. Finally, a classification of set operations is presented, and it is shown that all problems of a large naturally arising class are as hard as the problems bounded in this paper.

1. Introduction

The study of lower bounds is one of the main avenues taken in the quest for understanding the complexity of computations. Much effort has been directed toward the goal of proving nontrivial lower bounds on the resources required to compute various functions. In its basic form, the question involves establishing a lower bound on each resource separately. However, measuring a single resource (such as time or space) does not always correctly capture the entire picture regarding the complexity of the problems at hand. Consequently, some work was directed to considering two resources simultaneously, most commonly time and space product, denoted by TS.

Correspondence to: B. Patt-Shamir, Laboratory for Computer Science, MIT, 545 Technology Square, Cambridge, MA 02139, USA.

* Supported in part by an Allon Fellowship, by a Bantrell Fellowship and by a Walter and Elise Haas Career Development Award.

0304-3975/93/$06.00 © 1993—Elsevier Science Publishers B.V. All rights reserved

In 1966 Cobham [6] showed that any computational device with one read-only input tape must satisfy $TS=\Omega(n^2)$ in order to be able to recognize the set of perfect squares (n is the length of the input number). Although working within the severe limitation of tape input (and deriving his result from the crossing sequence argument), Cobham pioneered a new combinatorial concept for the workspace required by a nonoblivious program, that combined the traditional notion of "worktape" (which can be viewed as data space) with "control space" (which can be viewed as the space required for the instruction pointer). This concept was further abstracted and developed by Borodin et al. [5], where they show that any comparison-based "conservative" computational device requires $TS=\Omega(n^2)$ for sorting n inputs. The term *conservative* means here that the inputs are considered to be indivisible elements, and the branching of control is determined solely by the results of comparisons between pairs of input values. The basic concepts and techniques employed in TS lower-bound proofs were laid in that paper. Some other variants of the branching program model were considered by Yao [9], in which a linear type of queries is allowed, and by Karchmer [7], in which queries involving a comparison of more than two elements are allowed.

A significant generalization of the model was introduced by Borodin and Cook [3], where the R-way model is defined. In this model the flow of control may be effected by the inputs in any possible way, so long as the input values are in the range $[1..R]$. In [3], it was proved that sorting in such a "general sequential" model requires $TS=\Omega(n^2/\log n)$ for $R\geqslant n^2$. The general scheme of the proof is the one used in [5]. The bound for sorting was later increased (by improving one of the key lemmas of [3]) by Reisch and Schnitger [8] to $TS=\Omega(n^2\log\log n/\log n)$ for $R\geqslant(n\log n/\log\log n)$. Recently, Beame [2] proved that finding the unique elements in a list of n values (i.e., the elements appearing exactly once in the input) requires $TS=\Omega(n^2)$, for $R\geqslant n$. As a corollary, he deduced that sorting requires $TS=\Omega(n^2)$ for $R\geqslant n$.

Another area in which TS lower bounds are known in the R-way model is computing algebraic functions. Yesha [11] showed, basing his arguments on [3], that over finite fields, computing the Fourier transform of n elements requires $TS=\Omega(n^2)$, and multiplying two $n\times n$ matrices requires $TS=\Omega(n^3)$. Abrahamson [1] proved that over a finite field of size R, the convolution of two n-vectors requires $TS=\Omega(n^2\log R)$, and multiplying two $n\times n$ matrices requires $T^2S=\Omega(n^6\log R)$.

All the above results are based on the technique of [5], which seemed to yield bounds not better than $TS=\Omega(nr)$, where n is the number of inputs, and r is the number of outputs. But in 1987, Borodin et al. [4] were able to establish a nontrivial lower bound for a *decision* problem ($r=1$) within the "reasonable" comparison branching program model. Specifically, it is shown in [4] that deciding element distinctness (i.e., deciding whether the input values are all distinct) requires $TS=\Omega(n^{3/2}\sqrt{\log n})$. This result was accomplished by defining a new measure for the progress of a program, and applying again the general scheme of [5]. This bound was improved by Yao [10] to $TS=\Omega(n^{2-\varepsilon(n)})$, where $\varepsilon(n)=5/\sqrt{\ln n}$, by using the same arguments of [4], and very careful accounting.

In this paper we study the complexity of computational and decision problems concerning several set operations. The input to these problems is typically one or two sets of integers, although some of the bounds are derived for set operations of arbitrary (fixed) arity. The problems involve either computing the result of some unary or binary set operation, or deciding whether a certain property holds. Specifically, we consider the following operations. Let A and B denote sets of integers with $|A|=n$, $|B|=m$, and assume $n \geqslant m$. Let COMP^R denote the problem of computing $\{1, \ldots, R\} \setminus A$ (where we assume that $A \subseteq \{1, \ldots, R\}$). Let IS, UN, SUB and XOR denote the problems of computing the intersection $A \cap B$, the union $A \cup B$, the difference $A \setminus B$ and the symmetric difference $A \oplus B$, respectively. As for decision problems, let DIS, EQ and INC denote the problems of deciding whether A and B are disjoint, equal, or whether A contains B, respectively (for EQ assume $|A|=|B|=n$). We seek bounds on the minimal product TS required by algorithms solving these problems.

Naturally, the model of computation is a central issue when discussing lower bounds. As described above, much of the earlier results were derived for models of a very restrictive type, such as tape input, or oblivious programs. The models adopted in this paper are variants of the quite general branching program model, as introduced by Cobham [6], described by Borodin et al. [5], and generalized by Borodin and Cook [3]. This model is formally defined in Section 2.

Our results are based on techniques that were used previously to provide bounds for two problems, namely element distinctness (ED) and unique elements (UE). Following [4, 10, 2], we derive three types of results for set operations. In Section 2 we define the models of computation and agree upon some notational conventions. In Section 3 we follow [2] and prove two lower bounds in the general model of R-way branching programs. It is shown that set complementation (COMP^R) admits $TS=\Omega(n^2)$, deducing this bound for set subtraction (SUB) and symmetric difference (XOR) as direct corollaries. We also prove that set intersection (IS) admits $TS=\Omega(m\sqrt{n})$. Both bounds hold for $R=O(n)$. We then restrict our attention to comparison branching programs. Section 4 gives a "near optimal" bound of $TS=\Omega(n^{2-\varepsilon(n)})$, where $\varepsilon(n)=5/\sqrt{\ln n}$, for set disjointness (DIS), deducing this bound for set union (UN) and set intersection (IS) as direct corollaries. These bounds are derived by generalizing the technique of [10]. In Section 5 we show, as a generalization of the proof in [4], that the time and space required to decide whether two sets are equal, satisfy $TS=\Omega(n^{3/2})$ in the comparison branching program model. Let us remark that some of our results are stated also for the *average* time, $\bar{T}$.

Finally, in Section 6 we attempt to extend and unify the above results into a more general statement holding for a wide class of set operations. We begin by providing a classification of set operations by some natural properties. It is then shown that all set problems from a large "natural" class are as hard as either set complementation or set intersection (for computational problems), or as hard as set disjointness or set equality (for decision problems). In Section 7 it is shown that all definable set operations can be computed in time–space product $O(n^2)$ in the R-way model, and that another "natural" class of set problems can be computed by a random-access

machine (RAM) algorithm that require $O(S)$ space and $O(nm \log n/S)$ time for all $\log n \leqslant S \leqslant m$.

2. The model

2.1. Branching programs

We first describe the general model of branching programs [6, 5, 3]. Branching programs model algorithms by labelled directed multigraphs representing the flow of control. Formally, we assume that the input and the output domains are known, and a branching program is defined by a seven-tuple $P=(X, V, E, v_0, Q, A, O)$, where

- $X=\{x_1, \ldots, x_n\}$ is the set of input variables;
- V and E are the sets of nodes and edges of a directed multigraph;
- $v_0 \in V$ is the *root* node;
- $Q: V \to X^i$ (for some integer $i \geqslant 1$) is a mapping associating a query concerning the input variables with every node that has outgoing edges, in a way that will be explained later;
- A is a mapping associating with every edge (v, u), a possible outcome (an *answer*) of the query associated with its starting node, v. A associates *all* the possible outcomes of $Q(v)$ with edges outgoing from v, and every such outcome is associated with exactly *one* of the outgoing edges;
- O is the output mapping associating with every edge a subset of the output domain.

Essentially, the nodes represent possible configurations of the computation (excluding the input, unlike instantaneous descriptions of Turing machines), and the edge set represents the transition function. The *root* node, v_0, corresponds to the initial configuration. Define an *input instance* to be an assignment of values from the input domain to the input variables. Given such an input instance, its corresponding *computation* is the path in the program that starts at the root, and consists of the edges labelled by the answers to the queries associated with the nodes on its way. In a correct program, every path followed by an input instance ends in a sink (a node with no outgoing edges), whereby the computation halts. For a computation that consists of the edges $e_1, \ldots, e_t$, the output is defined by the output mapping O as the union $\bigcup_{i=1}^{t} O(e_i)$. When we are dealing with a decision problem, we may label some of the sinks as *accepting* and the others as *rejecting*, which can be treated as identical to "YES" or "NO" output, respectively, associated with the edges leading to these sinks.

There are differences in the type of queries allowed in the different variants of the branching program model, that effect the computational power and the generality of the variants. In the comparison branching program model, the input domain is assumed to be linearly ordered, and the queries are of the type "$x_i : x_j$", where x_i and x_j are input variables (formally, $Q: V \to X^2$). The answer mapping, A, is defined formally to be $A: E \to \{<, =, >\}$. The queries and the answers are interpreted in the obvious

way. In this model, the output may consist of indices of the input variables and constants. This model was employed in the proofs of lower bounds on sorting [5] and element distinctness [4, 10].

The most general model is called the R-way model, introduced by Borodin and Cook [3]. In this model the input domain, D, is of size R. The nodes are labelled by variables $(Q: V \to X)$, and the edges are labelled by the R possible values the queried variable may have $(A: E \to D)$. Among the results in this model there are lower bounds for sorting [3, 8], matrix multiplication and the discrete Fourier transform [1, 11], and unique elements [2].

It should be emphasized that the only restriction on the way the graph is specified is the one imposed by the answer mapping A, which is necessary to make the notion of computation well-defined. There are no assumptions whatsoever concerning the way the computation is realized. This feature is to be contrasted with the conventional models of Turing machine, or RAM, which are defined in a structured fashion, with a small repertoire of basic moves. This makes the branching program model (and especially the R-way model) a very general one and, hence, adequate for proving lower-bound results.

An important aspect of this model is its nonuniformity. Denote the number of variables in an input instance I by $|I|$. Branching programs have a fixed number of input variables for a single program. We say that a problem Π is computable by a family $\{P_n\}$ of branching programs if for every admissible input instance I, the program $P_{|I|}$ outputs $\Pi(I)$. Whenever we discuss the asymptotic complexity of branching programs, it is to be understood with respect to such a family $\{P_n\}$ of programs that solve the problem in question. Note, in addition, that the actual input length is $|I| \log R$ (assuming $|D| = R$). We denote $|I| = n$, and the latter quantity of the bit-cost input length is denoted $N = n \log R$.

In many senses, the model of R-way branching programs is "more powerful" than the Turing machine model, due to its nonuniformity, random-access ability and the non-structured way in which a program is specified. The only aspect in which this model is weaker than Turing machines is the way the results of the computation are output; producing an output value is an atomic step.

2.2. *Basic properties*

Let us now define the complexity measures relevant to branching programs, and state some of their basic properties.

The *running time* of a branching program is defined to be the length of the longest path from the root that some input follows. This is a unit-cost worst-case time measure.

The *workspace* S of a branching program (called also its *capacity*) is defined to be the logarithm (to base 2) of the number of all nodes reachable (by some input) from the root. This definition is appropriate for lower-bound results, since regardless of the way the space is utilized, it is necessary at least to be able to distinguish among the different

configurations. This definition captures the space required for storing intermediate values, as well as the control space. See [5, 3] for a more detailed discussion and justification of this definition. Note, for example, that branching programs do not have an explicit notion of internal variables. If we want to simulate some variable that can have k distinct values, then we can take k copies of the program, where each copy corresponds to some possible value. This costs $O(\log k)$ additive space, which is the minimal storage required to store the value of the variable by a logarithmic-cost space measure for RAMs, or Turing machines.

A branching program whose underlying graph is a directed tree is called a *tree program*, or a *computation tree*. Note that, for any problem and any input length n, there exists an R-way tree program with running time $T=n$ and capacity $S=O(n \log R)$ and, hence, in the R-way model $TS=O(n^2 \log R)$ for *all* definable problems.

We state some of the basic properties concerning the time and space requirements of branching programs (see [5]). First note that the length of any path cannot exceed the total number of nodes, which is 2^S. Consequently, we have Property 1.

Property 1. $T \leqslant 2^S$.

We remark that all the problems discussed in this paper cannot be computed in sublinear worst-case time, that is, $T \geqslant n$ and, by Property 1, also $S \geqslant \log n$.

A branching program with running time T is called *levelled* if its node set can be partitioned into T disjoint subsets labelled $0, 1, \ldots, T$, in such a way that every edge outgoing from a node in subset i is incoming into a node in subset $i+1$. We now consider the conversion of an arbitrary program P into a levelled one. This can be done by combining T copies of P, and the capacity of the resulting program is bounded by $\log(T \cdot 2^S) \leqslant 2S$, by Property 1. Therefore, we have Property 2.

Property 2. For every S-space, T-time branching program P there exists a levelled program with identical output, running time T and $O(S)$ space.

Property 2 is of special importance, since it allows us to consider only levelled programs when we deal with the asymptotic complexities of branching programs. Throughout the remainder of this paper we assume, without loss of generality, that all branching programs considered are levelled.

Finally, consider an R-way branching program solving some problem. Let $R' < R$. The deletion of all edges labelled by $R' < r \leqslant R$, followed by the deletion of all unreachable nodes, can decrease only the running time and the capacity of the problem. Hence, we have Property 3.

Property 3. For every T-time, S-space, R-way branching program and for all $R' < R$, there exist a T'-time, S'-space, R'-way program solving the same problem for $D' = \{1, \ldots, R'\}$ that satisfies $T' \leqslant T$ and $S' \leqslant S$.

Note that by Property 3, any TS lower bound for R_0-way branching programs applies to all R-way branching programs satisfying $R \geq R_0$.

2.3. Notations

Since we are focusing on set problems, let us agree upon the following conventions to hold throughout this paper. Denote by Π_n a problem Π whose input consists of a set of variables, $X = \{x_1, \ldots, x_n\}$, that take values from some linearly ordered domain D, and, w.l.o.g., we generally assume $D = \{1, \ldots, R\}$. Given an input instance, denote the set of values assigned to X by A. Since in this paper we are dealing with set operations, when the set operation in question is k-ary, we partition the variable set into k sets, $X_1, \ldots, X_k$, and the corresponding value-sets are denoted by $A_1, \ldots, A_k$. If the problem Π is binary, we denote Π_{nm} for input variable sets $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_m\}$, and the sets of values are denoted A and B. We always assume $|A| \geq |B|$, i.e., $n \geq m$.

Whenever we define an input instance to consist of *sets*, it is to be interpreted that all the variables in a single variable set are assigned distinct values. The input instance is said to consist of *multisets* or *lists* when the same value may be assigned to different variables in a single variable set. For most set problems, one may consider versions in which input instances consist of either sets or multisets, and likewise the output. We comment here that most of our lower bounds (excluding union) are derived in the weakest framework, i.e., the framework in which the input instance is guaranteed to consist of sets, and the output is allowed to be a multiset. Therefore, the bounds hold also in the stronger frameworks.

Let τ be a path in a branching program. Denote by $|\tau|$ the *length* of τ, i.e., the number of edges it contains. Denote by $Q(\tau)$ the set of variables queried in τ, and for an input variable set X let $X_\tau = X \cap Q(\tau)$. When the identity of τ is clear from the context, we denote $t = |\tau|$ and $t_X = |X_\tau|$.

A notation often used in the sequel is $(a)_b$, denoting the number of ways to choose b ordered elements out of a set of size a,

$$(a)_b = a \cdot (a-1) \cdots (a-b+1).$$

We use probabilistic language in forming our arguments. It is generally assumed that all admissible input instances have equal probability, unless explicitly indicated otherwise.

Throughout, we omit floors and ceilings, for simplicity of presentation. All occurrences of "log" denote logarithm to base 2, and "ln" is logarithm to base e.

3. Lower bounds in the R-way model

This section presents the general technique in which time–space tradeoffs are derived for branching programs. This technique is applied in the general R-way model to establish the bounds $TS = \Omega(n^2)$ for set complementation and $TS = \Omega(mn^{1/2})$ for set intersection.

3.1. A generic lemma

We open with a generic lemma that outlines the scheme of the lower-bound proofs for computational problems. The basic idea, due to Borodin et al. [5], is that if one can show, for a given problem Π, that every shallow tree-program cannot output correctly "too many" values, and that there are "many" values to be output by "many" of the input instances, then the time–space product of any branching program solving Π can be bounded from below. This idea is used in [2, 4, 7, 10, 11]. We closely follow [2].

Lemma 3.1. *Let Π be a set problem with a given probability distribution over its domain of instances. Suppose that for sufficiently large input length n there exist $0<\varepsilon_n\leqslant 1$, $\beta_n, \delta_n>0$ (all may depend on n), and a constant $\gamma<1$ (independent of n), such that the following conditions hold:*

(1) *For any R-way tree branching program P of length $t\leqslant\beta_n$ and for all $r>0$, the probability that P outputs r distinct correct values for Π is less than γ^r.*

(2) *The probability for an input instance picked at random (according to the given distribution) to have at least δ_n distinct output values is at least ε_n.*

Then for n sufficiently large, any R-way branching program solving Π_n with capacity S and time $T\geqslant\beta_n$ satisfies $S\geqslant(\log(1/\gamma))\beta_n\delta_n/T-(\log(1/\varepsilon_n))$. Furthermore, if $\varepsilon_n>q$ for some constant $q>0$, then $\bar{T}S=\Omega(\beta_n\delta_n)$, where $\bar{T}$ denotes the average running time according to the given distribution.

Proof. Let P be a branching program solving Π_n in time T and capacity S. Consider P in stages of β_n steps each. There are T/β_n such stages. For every input instance of Π_n with δ_n or more output values, there must be a stage in which more than $\delta_n/(T/\beta_n)=\delta_n\beta_n/T$ values are output. Denote $r_n=\delta_n\beta_n/T$. Regard the subprograms rooted at each node in the start of a stage, truncated to length β_n, as computation trees (this may be done by duplicating the subprograms rooted at nodes with more than one incoming edge). By assumption (1), the probability that a random input instance will output r_n values in such a subprogram is less than γ^{r_n}. Since there are 2^S nodes, and by assumption (2) there is a subset of the input instances with probability ε_n for which P outputs at least r_n values, P cannot solve Π_n correctly unless $2^S\gamma^{r_n}\geqslant\varepsilon_n$, i.e.,

$$2^S\gamma^{\delta_n\beta_n/T}\geqslant\varepsilon_n,$$

which implies

$$S\geqslant(\log 1/\gamma)\frac{\delta_n\beta_n}{T}-(\log 1/\varepsilon_n).$$

Note that both parenthesized quantities are nonnegative.

To see that the second assertion holds, we remark that the average case time complexity $\bar{T}$ satisfies $\bar{T}\geqslant\varepsilon_n T\geqslant qT=\Theta(T)$. □

Given Lemma 3.1, in order to obtain a lower bound for the time–space product it suffices to show that the assumptions of the lemma hold for the problem in question.

3.2. Set complementation

We first turn to set complementation, formally defined as follows:

SET COMPLEMENTATION (COMP^R)
Instance: A set of integers $A \subseteq \{1, \ldots, R\}$.
Output: $\{1, \ldots, R\} \setminus A$.

Recall that the input to COMP_n^R consists of the variables $X = \{x_1, \ldots, x_n\}$, whose contents represent the elements of A.

Lemma 3.2. *Let P be an R-way computation tree of height at most βn, where $0 < \beta < 1$ is a constant, and $R \leqslant cn$ for some constant $c > 1$. Let $r \geqslant 0$. Assume that all the input instances of COMP_n^R have equal probability. Then the probability that a random input instance of COMP_n^R follows a path in P that outputs more than r distinct correct values is bounded by γ^r, where $\gamma = (c-1)/(c-\beta)$.*

Proof. Fix a computation path τ in P. Recall that t_X denotes the number of distinct X-variables queried in τ. Since $|\tau| \leqslant \beta n$, we have that $t_X \leqslant \beta n$. Denote $a = R - t_X$ and $b = n - t_X$. The total number of input instances that follow τ (i.e., agree with the outcomes of the queries in τ) is $(a)_b$, since there are t_X variables whose values are determined by the answers in τ. Consider now only the first r distinct output values. The number of input assignments that follow τ and have correct output is $(a-r)_b$, because the remaining b variables are not allowed to take the r values that are output in τ. Therefore, denoting by E the event that an input instance correctly outputs r values, we have that

$$\begin{aligned}\Pr\{E\} &= \frac{(a-r)_b}{(a)_b} \\ &= \frac{(a-r)!}{a!} \cdot \frac{(a-b)!}{(a-b-r)!} \\ &= \frac{(a-b-r+1)(a-b-r+2)\cdots(a-b)}{(a-r+1)(a-r+2)\cdots a}.\end{aligned}$$

This probability can now be bounded by

$$\Pr\{E\} \leqslant \left(\frac{a-b}{a}\right)^r = \left(\frac{R-n}{R-t_X}\right)^r \leqslant \left(\frac{c-1}{c-\beta}\right)^r = \gamma^r. \qquad \square$$

Theorem 3.3. *Any T-time, S-space R-way branching program that solves COMP_n^R for $R \geqslant cn > n$ satisfies $\bar{T}S = \Omega(n^2)$, where all the input instances of COMP_n^R have equal probability.*

Proof. We assume that $R = cn$ and, w.l.o.g., $c > 1$. As mentioned in Section 3.1, all we need to show is that the conditions of Lemma 3.1 hold. Lemma 3.2 fulfills condition (1)

with $\beta_n = \beta n$ for any constant $0 < \beta < 1$, and $\gamma = (c-1)/(c-\beta) < 1$. As to condition (2), we note that *all* input instances of COMP_n^R must output $R-n$ distinct values and, thus, we set $\delta_n = R - n = (c-1)n$ and $\varepsilon_n = 1$. In addition, we remark that, since all branching programs that solve COMP_n^R must query every variable at least once, their running time T must satisfy $T \geqslant n > \beta n$. The bound $\bar{T}S = \Omega(n^2)$ now follows from Lemma 3.1. □

Consider the following problems:

SET SUBTRACTION (SUB)
Instance: Two sets of integers, A and B.
Output: $A \setminus B$.

SYMMETRIC DIFFERENCE (XOR)
Instance: Two sets of integers, A and B.
Output: $(A \cup B) \setminus (A \cap B)$.

As a direct corollary of Theorem 3.3, we have the following.

Corollary 3.4. *Any R-way branching program that solves* XOR *or* SUB *in time T and space S satisfies $TS = \Omega(n^2)$.*

Proof. By the definitions of the problems, it is clear that COMP^R is the restriction of XOR, or of SUB, to the case where $A = \{1, \ldots, R\}$. Therefore, either XOR or SUB solve COMP^R by the trivial reduction. □

Note that the distribution implicitly assumed by Corollary 3.4 for the instances of XOR and SUB is the one assumed for COMP^R, i.e., the distribution giving equal probability to all instances in which $B \subseteq A$, and zero probability to all other cases. Therefore, the result for the average case does not follow directly from Theorem 3.3 (although it can be obtained by mimicking the technique of Lemma 3.2).

3.3. Set intersection

The set intersection problem is formally defined as follows.

SET INTERSECTION (IS)
Instance: Two sets of integers, A and B.
Output: $A \cap B$.

We continue using the framework of Lemma 3.1. The following lemma provides us with condition (1) of Lemma 3.1 for set intersection.

Lemma 3.5. *Let $0 \leqslant r \leqslant m$ and $c_1 > 2$, and let P be an R-way computation tree of height $T \leqslant \sqrt{n}$, where $R \geqslant c_1 n$. Suppose that all the input instances of* IS_{nm} *have equal probability. Then given a random input instance of* IS_{nm}*, the probability that P outputs more than r distinct correct values is bounded by $(2/\sqrt{c_1 - 1})^r$.*

Proof. For a computation path τ in P, denote by r_τ the number of values output in τ. Denote by k_X the number of output values that appear in τ as values of variables in X_τ, by k_Y the number of output values that appear in τ as values of variables in Y_τ, and by k_τ the number of output values that appear in τ as values of variables in both X_τ and Y_τ. Clearly, $r_\tau \geqslant k_X + k_Y - k_\tau$.

Let $r \geqslant 0$. Denote by E the event in which a random input instance follows a path τ that correctly outputs more than r distinct values (i.e., $r_\tau > r$), and by E' the event in which a random input instance follows a path τ with $k_\tau > r/2$ (i.e., the outcomes of the queries in the path suffice to ensure that more than half of its output values are correct). With these notations,

$$\Pr\{E\} = \Pr\{E \cap E'\} + \Pr\{E \cap \neg E'\}. \tag{1}$$

We bound the probability $\Pr\{E \cap E'\}$ by $\Pr\{E'\}$, i.e., the probability that a random input instance of IS_{nm} follows a path τ with $k_\tau = k > r/2$ internal X–Y equalities. Fix X_τ and Y_τ, the sets of variables queried in τ. The total number of possible assignments for these variables is $(R)_{t_X}(R)_{t_Y}$. To count the number of such assignments with exactly k X–Y equalities, we choose values for all the t_X variables of X_τ, choose the k variables from Y_τ to be in the intersection with the values of the X_τ variables and choose their values among the t_X values and, finally, choose values for the $t_Y - k$ remaining Y_τ-variables among the $R - t_X$ remaining values. Thus, the number of such assignments is

$$(R)_{t_X}\binom{t_Y}{k}(t_X)_k(R-t_X)_{t_Y-k}$$

and, so, we have that

$$\begin{aligned}
\Pr\{E \cap E'\} &\leqslant \Pr\{E'\} \\
&= \frac{(R)_{t_X}\binom{t_Y}{k}(t_X)_k(R-t_X)_{t_Y-k}}{(R)_{t_X}(R)_{t_Y}} \\
&\leqslant \frac{\binom{t_Y}{k}(t_X)_k}{(R)_k} \\
&\leqslant \left(\frac{t_Y t_X}{R}\right)^k.
\end{aligned}$$

But $t_X + t_Y \leqslant T$ implies $t_X t_Y \leqslant T^2/4$; since $k = k_\tau > r/2$, it follows that

$$\Pr\{E \cap E'\} \leqslant \left(\frac{T^2}{4R}\right)^{r/2} \leqslant \left(\frac{1}{2\sqrt{c_1}}\right)^r. \tag{2}$$

We now turn to bound the second summand in (1). That is, we wish to bound the probability that more than r output values are correct in a path with less than $r/2$ internal equalities. For this, we use the fact that $\Pr\{E \cap \neg E'\} \leqslant \Pr\{E \mid \neg E'\}$. Given

that E' is not the case in a path τ, i.e., $k_\tau \leqslant r/2$, we calculate the probability that a random input instance follows τ and outputs r correct values as follows. Let $a = n - t_X$, the number of X-variables not queried in τ, and let $b = m - t_Y$, the number of Y-variables not queried in τ. The total number of input instances that follow τ is

$$(R - t_X)_a (R - t_Y)_b.$$

Let us count the number of inputs for which the outputs are correct. There are t_X values of X_τ-variables that are determined by the queries. Let $s = r - k_X$, the number of values output in τ that do not appear in τ as values of X-variables, and let $t = r - k_Y$, the number of values output in τ that do not appear in τ as values of Y-variables. By the definition of set intersection, s X-variables must take values from the values output in τ. All the other $a - s$ variables in X can take any of the remaining $R - t_X - s$ values and, thus, the total number of input assignments for the X-variables is

$$(a)_s (R - t_X - s)_{a-s} \leqslant n^s (R - t_X)_{a-s}.$$

The counting for the Y-variables is dual, when we substitute m for n, t_Y for t_X, b for a and t for s. So, we have

$$\Pr\{E \mid \neg E_1\} \leqslant \frac{n^{s+t} (R - t_X)_{a-s} (R - t_Y)_{b-t}}{(R - t_X)_a (R - t_Y)_b}.$$

But $R - t_X - a + s = R - n + s$ and $R - t_Y - b + t = R - m + t$ and, hence,

$$\Pr\{E \mid \neg E_1\} \leqslant \frac{n^{s+t}}{(R-n+1)\cdots(R-n+s)\cdot(R-m+1)\cdots(R-m+t)}$$

$$\leqslant \left(\frac{n}{R-n}\right)^{s+t};$$

since $k_\tau \leqslant r/2$ implies $s + t = 2r - k_X - k_Y \geqslant r - k_\tau \geqslant r/2$ and $c_1 > 2$, we have

$$\Pr\{E \mid \neg E_1\} \leqslant \left(\frac{n}{R-n}\right)^{s+t} \leqslant \left(\frac{1}{c_1 - 1}\right)^{2r - k_X - k_Y} \leqslant \left(\frac{1}{\sqrt{c_1 - 1}}\right)^r.$$

Thus, we get

$$\Pr\{E \cap \neg E'\} \leqslant \Pr\{E \mid \neg E'\} \leqslant \left(\frac{1}{\sqrt{c_1 - 1}}\right)^r. \tag{3}$$

Combining (1)–(3) together we get, for sufficiently large n,

$$\Pr\{E\} = \Pr\{E \cap E'\} + \Pr\{E \cap \neg E'\}$$

$$\leqslant \left(\frac{1}{2\sqrt{c_1}}\right)^r + \left(\frac{1}{\sqrt{c_1 - 1}}\right)^r \leqslant \left(\frac{2}{\sqrt{c_1 - 1}}\right)^r. \qquad \square$$

The following lemma shows that is satisfies condition (2) of Lemma 3.1 for $R = O(n)$.

Lemma 3.6. *Let* $R \leqslant c_2 n$ for $c_2 \geqslant 1$. *Suppose that all the instances of* IS_{nm} *have equal probability, where all the variables take values from the input domain* $\{1, \ldots, R\}$. *Then* $\Pr\{|A \cap B| \geqslant nm/2R\} \geqslant 1/(2c_2 - 1)$.

Proof. The expected number of intersections is

$$E(|A \cap B|) = \sum_{i=1}^{m} \Pr\{y_i \in A\} = \frac{nm}{R}.$$

Let $\varepsilon = \Pr\{|A \cap B| \geqslant nm/2R\}$. The fact that $|A \cap B| \leqslant |B| = m$ implies

$$E(|A \cap B|) \leqslant \varepsilon m + (1 - \varepsilon)\frac{nm}{2R}$$

and, hence,

$$\varepsilon \geqslant \frac{n}{2R - n} \geqslant \frac{1}{2c_2 - 1}. \qquad \square$$

Theorem 3.7. *Any R-way branching program that solves* IS_{nm} *in time T and capacity S for* $R \geqslant cn > 5n$ *satisfies* $\bar{T}S = \Omega(m\sqrt{n})$, *when all instances of* IS_{nm} *are considered to have equal probability.*

Proof. We establish the bound by applying Lemma 3.1. Assume $R = cn$ for $c > 5$. Let $\beta_n = n^{1/2}$. Then Lemma 3.5 provides us with condition (1) of Lemma 3.1, setting $\gamma = 2/\sqrt{c-1} < 1$. Lemma 3.6 ensures us that at least $1/(2c-1)$ of the inputs have at least $mn/2R = m/2c$ output values and, hence, we set $\delta_n = m/2c$ and $\varepsilon_n = 1/(2c-1)$. It is clear that the running time of any R-way program solving IS_{nm} is at least $n + m > \beta_n$. Since c and ε_n are constants, we get from Lemma 3.1

$$\bar{T}S = \Omega(m\sqrt{n}),$$

when all instances of IS_{nm} are equally likely. $\square$

Comment: The lower bounds for COMP^R and IS were proved for the case of set input and multiset output. By the trivial reduction, the bounds hold when the input consists of multisets.

4. Strong lower bounds in the comparison model

Consider the well-known element distinctness problem.

ELEMENT DISTINCTNESS (ED)

Instance: A list of integers.

Output: YES iff all input integers are distinct.

In this section, we base upon known bounds for ED [4, 10] and derive bounds on the time–space product for set operations. In the problems discussed so far, we allowed the output to contain repetitions. If we would have insisted that the output must be a set, then the element distinctness problem could be reduced to both COMP^R and IS (by setting $A=\{1,\ldots,R\}$ and counting the outputs) and, hence, both problems would have admitted the ED lower bounds. Here we follow the technique of [10], and derive a "near optimal" bound of $\Omega(mn^{1-5/\sqrt{\ln n}})$ for deciding set disjointness, and deduce that the bound holds for set union, set intersection and for the problem of deciding whether two sets have at least k elements in common (where k is fixed). Set disjointness and set union are formally defined as follows.

SET DISJOINTNESS (DIS)

Instance: Two sets of integers, A and B.

Output: YES iff $A\cap B=\emptyset$, A and B are disjoint.

SET UNION (UN)

Instance: Two sets of integers, A and B.

Output: $A\cup B$, when every input value appears exactly once.

Note that union is trivial (i.e., solvable in linear time and logarithmic space) if the set-output constraint is not imposed. However, ED cannot be reduced directly to union, because the instances of ED are multisets, whereas UN is defined for instances that consist of sets.

In this section (and in Section 5) we restrict the discussion to comparison branching programs. We often identify the input instance with the mapping associating each input variable with its rank in the input set. There is no loss of generality, since in the comparison branching program model, the computation is effected solely by the ranks assigned to the input variables.

4.1. Adjacent pairs and the AC property

The general idea we follow in this section, due to Borodin et al. [4], is that in order to solve the above problems correctly, any comparison branching program must compare certain pairs of input values. We need the following definitions.

Definition 4.1. Let $\pi:\{1,\ldots,n\}\to\{1,\ldots,R\}$ be a one-to-one mapping. A pair (i,j) is an *adjacent pair* if $\pi(i)<\pi(j)$ and there is no $k\in\{1,\ldots,n\}$ such that $\pi(i)<\pi(k)<\pi(j)$. A comparison of an adjacent pair is an *adjacent comparison.*

Note that the only way a comparison branching program can get any information concerning the mutual ranking of an adjacent pair is by an adjacent comparison.

Definition 4.2. Let $\mathcal{I}$ be a subset of the input instances of a comparison branching program P. P is said to have the *m-AC property* (or P is an *m-AC program*) *with respect*

to $\mathcal{I}$ if, for every input instance $I \in \mathcal{I}$, P compares at least m adjacent pairs ($m \leqslant n-1$, where n is the number of the input variables). We call $(n-1)$-AC programs shortly *AC programs.*

Definition 4.3. A *permutation instance* is an instance with all input variables assigned distinct values.

We often identify a permutation instance with its corresponding permutation.

We show that any program solving the problems in question has to have the AC property with respect to the permutation instances.

The two previous proofs of lower bounds on the time–space tradeoff for ED in the comparison branching program model [4, 10] share the same overall structure:

(1) Show, in a main lemma, a bound on the rate of progress of the program. Progress is measured by the number of adjacent comparisons made so far.

(2) Conclude a lower bound for the time complexity as a function of the input length and the capacity of any branching program having the AC property.

(3) Finally argue that any program solving ED must be an AC program with respect to the YES instances. This is done in the following way. Suppose that P does not have the AC property (with respect to the YES instances). That is, there exists a computation path τ in P and a YES instance $I=(x_1, \ldots, x_n)$ which follows τ, and a pair (i_0, j_0) adjacent in I, such that P does not compare x_{i_0} with x_{j_0}. Then one can define a NO instance $I'=(x'_1, \ldots, x'_n)$, where $x'_i = x_i$ for all $i \neq j_0$, and $x'_{j_0} = x_{i_0}$. Clearly, the only comparison effected by this change is $x'_{i_0} : x'_{j_0}$, and since P, by the assumption, does not test this pair, I' follows the same path I follows and, hence, outputs the same answer, proving that P does not solve ED.

Thus, these proofs can be viewed essentially as lower-bound proofs for programs with the AC property, augmented by the fact that in order to answer the ED question, a program must have the AC property with respect to the YES instances.

We apply the results for AC programs and derive a bound for set disjointness. We make use of the corollary of the main lemma of [10].

Let us first provide a glossary for the necessary notations. Let n be the number of input variables, and let S be some fixed positive number. Denote

$$\eta(n)=\frac{1}{\sqrt{\ln n}}, \qquad \varepsilon(n)=5\eta(n), \qquad y=n^{\eta(n)},$$

$$k_0=\log_y \frac{n}{4}, \qquad a_{k_0}=2^{5k_0}yS, \qquad p_{k_0}=(4y)^{k_0}16^{-S}.$$

Corollary 4.4 (Yao [10]). *Let P be a comparison branching program of capacity $S \geqslant \log n$ and running time no more than $n/4$. Then for sufficiently large n, the probability that more than a_{k_0} comparisons of adjacent pairs are made on a randomly chosen permutation of $\{1, \ldots, n\}$ is less than p_{k_0}.*

Yao makes use of this corollary in a theorem that bounds the time–space product of ED programs. As we are motivated by other problems, we generalize that theorem in two aspects. First, we deal with any sufficiently large number of adjacent comparisons. Second, we extend the result to the average case of YES instances.

Theorem 4.5. *Let P be a comparison branching program of capacity $S \geqslant \log n$, and let $r \geqslant a_{k_0}$. Then the running time required to compare r adjacent pairs satisfies $\bar{T}S = \Omega(r \cdot n^{1-\varepsilon(n)})$ in the average case, where all permutation instances are considered to be equally likely.*

Proof. We show that at least a half of the permutation instances require that much time. Denote by K_v the set of permutations for which P performs more than a_{k_0} adjacent comparisons, when the computation starts from node v (in the branching program), and its length is no more than $n/4 = y^{k_0}$ steps. Then, by Corollary 4.4, $|K_v| \leqslant p_{k_0} n!$. Hence,

$$\left| \bigcup_{v \in P} K_v \right| \leqslant 2^S p_{k_0} n! = 2^S (4y)^{k_0} 2^{-4S} n! = 2^{-3S} 4^{k_0} \frac{n}{4} n!.$$

Now, since

$$4^{k_0} = 4^{\log_y (n/4)} = \left(\frac{n}{4}\right)^{\log_y 4} = \left(\frac{n}{4}\right)^{\ln 4 \eta(n)} \leqslant n^2$$

and $2^{-S} \leqslant n^{-1}$, we get

$$\left| \bigcup_{v \in P} K_v \right| \leqslant \frac{n!}{2}.$$

That is, at least half of the instances are not in $\bigcup_{v \in P} K_v$, and every such instance makes no more than $a_{k_0} T/(n/4)$ adjacent comparisons, where T is the total running time of P. Since we require P to execute at least r adjacent comparisons, it follows that $4a_{k_0} T/n > r$ for at least half of the permutation inputs and, therefore,

$$\bar{T} \geqslant \frac{1}{2} \cdot \frac{nr}{4a_{k_0}}$$

or, in other words,

$$\bar{T} = \Omega\left(\frac{nr}{S} \frac{1}{y 2^{5k_0}}\right) = \Omega\left(\frac{r n^{1-\varepsilon(n)}}{S}\right),$$

because

$$y 2^{5k_0} = e^{\sqrt{\ln n}} e^{(\ln 32 \ln n/4)/\sqrt{\ln n}} \leqslant e^{\sqrt{\ln n}(\ln 32 + 1)} \leqslant n^{5/\sqrt{\ln n}}. \qquad \square$$

4.2. Set disjointness and corollaries

Consider now the set disjointness problem (DIS), as defined earlier. In order to bound the average case complexity of YES instances for DIS_{nm} by the lower bound for AC programs, we make the following definition.

Definition 4.6. Let (X, Y) be an arbitrary YES instance of DIS_{nm}, and let A and B be their respective sets of values. Let $x_{i_1}, x_{i_2}, \ldots, x_{i_{m+n}}$ be the complete ordering of $A \cup B$. We call the set

$$\{1 \leqslant i_j < m+n \mid x_{i_j} \in A \text{ and } x_{i_{j+1}} \in B, \text{ or } x_{i_j} \in B \text{ and } x_{i_{j+1}} \in A\}$$

the *alternation set*, and its size is the *number of alternations.*

Lemma 4.7. *The average number of alternations of a* YES *instance of* DIS_{nm} *is* $\Theta(m)$.

Proof. Let Z_i be a random variable, having value 1 if i is in the alternation set and 0 otherwise, and let

$$Z = \sum_{i=1}^{m+n-1} Z_i.$$

We assume, without loss of generality, that $A \cup B = \{1, \ldots, m+n\}$; further, we assume that all the YES instances satisfying $|A| = n$ and $|B| = m$ have equal probability. Then

$$\begin{aligned} E(Z) &= \sum_{i=1}^{m+n-1} E(Z_i) \\ &= \sum_{i=1}^{m+n-1} \left(\frac{n}{m+n} \cdot \frac{m}{m+n-1} + \frac{m}{m+n} \cdot \frac{n}{m+n-1} \right) \\ &= \frac{2nm}{n+m}. \end{aligned}$$

As $n \leqslant m+n \leqslant 2n$, it follows that $E(Z) = \Theta(m)$. □

Lemma 4.8. *Let P be a comparison branching program that solves* DIS_{nm}, *and let $k \leqslant m+n-1$. Then P has the k-AC property with respect to the* YES *instances with k alternations.*

Proof. Suppose that P does not have the k-AC property with respect to some YES instance (X, Y) with k alternations. Then, without loss of generality, we may assume that there exists an adjacent pair (i, j), such that P does not compare x_i with y_j. Define a NO instance (X, Y') by assigning $y_j = x_i$, and assigning to all other variables of Y' the same values as in Y. By the adjacency of (i, j), all comparisons other than $x_i : y_j$ are not effected by this substitution and, therefore, (X, Y') follows the same path that (X, Y) follows, thus reaching the same answer, contradicting the hypothesis that P solves DIS_{nm}. □

We deduce bounds on the average case of a YES instance for DIS. Note that Theorem 4.5 is restricted to the case of more than a_{k_0} adjacent comparisons. We use the estimate $a_{k_0} < Sn^{\varepsilon(n)}$.

Theorem 4.9. *Let P be a comparison branching program solving DIS_{nm} in time T and capacity S. Then the time–space product of the average case of the* YES *instances of DIS_{nm} with $m \geqslant a_{k_0}$ satisfies $\bar{T}S = \Omega(mn^{1-\varepsilon(n)})$.*

Proof. Since, by Lemma 4.7, there are $\Theta(m)$ alternations in the average and since the YES instances of DIS are permutation instances, it follows from Theorem 4.5 that the time–space product for any program solving DIS_{nm} satisfies

$$\bar{T}S = \Omega(m(m+n)^{1-\varepsilon(n)}) = \Omega(mn^{1-\varepsilon(n)}). \qquad \square$$

Bounds on the problems of set union and set intersection can now be proven as well.

Corollary 4.10. *Any comparison branching program that solves UN_{nm} in time T and capacity S for $m \geqslant a_{k_0}$, satisfies $TS = \Omega(mn^{1-\varepsilon(n)})$.*

Proof. We show that DIS reduces to UN by counting the outputs. More specifically, given a branching program P that solves UN_{nm}, one can construct a program P' that solves DIS_{nm} in the following way. Take $m+n+1$ copies of P labelled $0, 1, \ldots, m+n$. For all $0 \leqslant i < m+n$ and all $r > 0$, divert all edges in copy i with r output values to the corresponding endpoint in copy $i+r$. Discard all the outputs. Mark all the sinks in copy $m+n$ as accepting, and all other sinks as rejecting. Let the root of P' be the root of copy 0. Denote by T' and S' the running time and the capacity of P', respectively. By the construction, $T' = T$ and $S' = S + \log(m+n+1) = O(S)$. Clearly, the input sets are disjoint if and only if UN_{nm} outputs exactly $m+n$ values. Therefore, the existence of P with $TS = o(mn^{1-\varepsilon(n)})$ would contradict Theorem 4.9. $\square$

Corollary 4.11. *Any comparison branching program that solves IS_{nm} in time T and space S for $m \geqslant a_{k_0}$, satisfies $TS = \Omega(mn^{1-\varepsilon(n)})$.*

Proof. As in Corollary 4.10, we argue that any program for IS can be transformed into a program that solves DIS. This is done in the following way. All edges associated with output values (i.e., all the edges $e \in E$ such that $O(e) \neq \emptyset$) are diverted to a rejecting sink. All other sinks are labelled as accepting. The output values are discarded. It is clear that IS have an output value on an input instance (X, Y) if and only if $\text{DIS}(X, Y) = \text{NO}$. $\square$

Our last application of Theorem 4.9 concerns a somewhat different problem, defined as follows.

k-INTERSECTION ($k-$IS)

Instance: Two sets of integers, A and B.

Output: YES iff $|A \cap B| \geqslant k$ for some fixed constant k.

Corollary 4.12. *Any comparison branching program solving* k-IS$_{nm}$ *in time* T *and capacity* S *satisfies* $TS = \Omega(n^{2-\varepsilon(n)})$.

Proof. The assertion follows immediately from the fact that DIS reduces to k-IS: given an instance (A, B) of DIS, augment both A and B by $k-1$ new elements, getting an instance (A', B'). Clearly, DIS$(A, B) = k$-IS(A', B'). □

5. A weaker lower bound for set equality

This section presents lower bounds for the time–space product required by any comparison branching program that decides whether two given sets are equal. The result is obtained by a straightforward adaptation of the proof of [4] for element distinctness.

We begin with the formal definition of EQ.

SET EQUALITY (EQ)

Instance: Two sets of integers, A and B.

Output: YES iff $A = B$.

The bound, as before, stems from the fact that the only way a comparison branching program can know whether certain pairs of variables are equal, is by comparing them directly.

In the following proof, we assume that the probability distribution of the input instances is defined as follows.

(a) The assignments to the X-variables are fixed.

(b) The value set of the Y-variables is the same value set assigned to the X-variables, and all permutations assigning these ranks to the Y-variables are equally likely.

Lemma 5.1. *Let* P *be a comparison branching tree program of height* $t < \sqrt{n}$. *Then for all* $0 \leqslant r \leqslant t$ *the probability that a randomly chosen* YES *instance of* EQ$_n$ *follows a path in* P *with at least* r *distinct equality answers is less than* $(2t^2/n)^r$.

Proof. Fix a computation path τ with at least r equality answers. We use t, Y_τ and t_Y with the usual interpretation. The number of ways to assign to the Y_τ-variables ranks is $\binom{n}{t_Y}$. The number of ways to do it meeting the constraint that at least r of the ranks assigned to Y_τ express $X-Y$ equalities is $\binom{t}{r}\binom{n}{t_Y - r}$. Therefore, denoting by E the event

that a random input follows a path with at least r equality answers, we have, for sufficiently large n,

$$\Pr\{E\} = \frac{\binom{t}{r}\binom{n}{t_Y - r}}{\binom{n}{t_Y}}$$

$$= \binom{t}{r} \frac{t_Y \cdots (t_Y - r + 1)}{(n - t_Y) \cdots (n - t_Y - r + 1)}$$

$$\leqslant t^r \left(\frac{2t_Y}{n}\right)^r$$

$$\leqslant \left(\frac{2t^2}{n}\right)^r. \qquad \square$$

Theorem 5.2. *Every T-time, S-space comparison branching program that solves* EQ_n *satisfies* $TS = \Omega(n^{3/2})$.

Proof. The proof resembles that of Lemma 3.1. We first note that in order to answer YES to the EQ_n question, n distinct equalities must occur in the computation path of a comparison branching program (or otherwise some NO instance could follow the same path). Now, let $\beta_n < \sqrt{n}/2$. Consider P in stages of β_n steps each. There are T/β_n such stages. For every YES instance, there must be a stage with more than $n\beta_n/T$ distinct equality answers. Denote $q = n\beta_n/T$. Regarding the subprograms rooted at the nodes at the start of a stage as computation trees and, applying Lemma 5.1, we obtain that the probability for a random input instance to follow a path in such a subprogram with at least q equalities is less than $(2\beta^2/n)^q$. Since there are no more than 2^S nodes at the start of each stage, and since all YES instances of EQ_n must get n distinct equality answers, it must be the case that

$$2^S \left(\frac{2\beta^2}{n}\right)^q \geqslant 1,$$

which implies

$$S = \Omega\left(\frac{n\beta_n}{T}\right) = \Omega\left(\frac{n^{3/2}}{T}\right). \qquad \square$$

Consider the following problem.

SET INCLUSION (INC)

Instance: Two sets of integers, A and B.

Output: YES iff $B \subseteq A$, B is contained in A.

We have the following immediate corollary.

Corollary 5.3. *Let P be a comparison branching program with running time T and capacity S. If P solves* INC$_{nm}$ *then* $TS=\Omega(n^{3/2})$.

Proof. Follows from the fact that EQ reduces to INC: if $|A|=|B|$, then EQ$(A, B)=$YES iff INC$(A, B)=$YES. □

Note also that the time–space tradeoff for set union can be bounded using the bound for EQ by such a direct reduction. The bound obtained this way is for the instances of equal input sets, whereas the bound of Corollary 4.10 is for disjoint input sets.

6. Classification of set operations

In this section we provide a general classification of set operations by defining several interesting classes of set operations of arbitrary (fixed) arity. We then show that computing operations of some of the "natural" classes is as hard as complementation or intersection (for computational problems), or as hard as deciding equality or disjointness (for decision problems).

6.1. The classification

Assume the existence of a finite universal input and output domain D, with $|D|=R$. Let f be a k-ary computational set operation.

Denote

$$A|_a^{a'}=\begin{cases} A\cup\{a'\}\setminus\{a\} & \text{if } a\in A \text{ and } a'\notin A,\\ A\cup\{a\}\setminus\{a'\} & \text{if } a'\in A \text{ and } a\notin A,\\ A, & \text{otherwise.}\end{cases}$$

Definition 6.1.

- The operation f is *conserving* if for every $A_1, \ldots, A_k\subseteq D$,

$$f(A_1, \ldots, A_k)\subseteq \bigcup_{i=1}^{k} A_i.$$

- The operation f is *anonymous* if for every two elements $a, a'\in D$,

$$f(A_1, \ldots, A_k)|_a^{a'}=f(A_1|_a^{a'}, \ldots, A_k|_a^{a'}).$$

- The operation f is a *template* operation if there exists a *truthset* T_f of words from $\{0, 1\}^k$ such that $a\in f(A_1, \ldots, A_k)$ if and only if there exists a word $w=b_1\ldots b_k\in T_f$ for which

$$\forall 1\leqslant i\leqslant k\ [a\in A_i \Leftrightarrow b_i=1].$$

- The operation f is *accumulative* if it is conserving and for all subsets $D' \subseteq D$,

$$f(A_1 \cap D', \ldots, A_k \cap D') = f(A_1, \ldots, A_k) \cap D'.$$

- The operation f is *basic* if it is template and conserving.
- The operation f is *trivial* if there exists a *projection set* $J_f \subseteq \{1, \ldots, k\}$ such that

$$f(A_1, \ldots, A_k) = \bigcup_{i \in J_f} A_i.$$

To gain some intuition for the above definitions, we make the following remarks.

(1) Comparison branching programs with no output of constants can compute only conserving operations.

(2) In a certain sense, template operations comprise a natural class of operations. We demonstrate the "naturality" of template set operations by redefining some operations using their truthsets.

Unary operations. The only unary basic computational set operations are trivial: the identity operation, $I(A) = A$, has projection set $J_I = 1$ and truthset $T_I = 1$, and the null operation, $v(A) = \emptyset$, has empty projection set and empty truthset. Set Complementation is of course nonconserving, but it is a template operation, and $T_{\text{COMP}} = \{0\}$.

Binary operations. Intersection, symmetric difference, subtraction, and union can be defined by $T_{\text{IS}} = \{11\}$, $T_{\text{XOR}} = \{01, 10\}$, $T_{\text{SUB}} = \{10\}$ and $T_{\text{UN}} = \{01, 10, 11\}$, respectively. The projection operation, defined by $\pi_1(A, B) = A$, is a trivial set operation with projection set $J_{\pi_1} = \{1\}$ and truthset $T_{\pi_1} = \{10, 11\}$.

Note, however, that the definition of a problem by its truthset does not restrict the way the outcome is represented and, hence, it does not directly capture the difficulty of computing set union as discussed in Section 4.

(3) Accumulative set operations can be computed "slice by slice" and, hence, they admit a time–space tradeoff spectrum.

Motivated by the above concepts, we define a dual classification for decision operations. Let g be a k-ary decision set operation.

Definition 6.2.

- The operation g is *conserving* if for every nonempty subset of the domain $D' \subseteq D$ there exist $A_1, \ldots, A_k \subseteq D'$ such that

$$g(A_1, \ldots, A_k) = \text{YES}.$$

- The operation g is *anonymous* if for every two elements $a, a' \in D$,

$$g(A_1, \ldots, A_k) = g(A_1|_a^{a'}, \ldots, A_k|_a^{a'}).$$

- The operation g is a *template* operation if there exists a *truthset* $T_f = \{w_1, \ldots, w_t\}$, where $w_j = b_{j_1} \ldots b_{j_k}$ for $1 \leqslant j \leqslant t$ such that

$$g(A_1, \ldots, A_k) = \text{YES} \Leftrightarrow \forall a \in D \left[\bigvee_{j=1}^{t} \left(\bigwedge_{i=1}^{k} a \in A_i \Leftrightarrow b_{j_i} = 1 \right) \right].$$

- The operation g is *accumulative* if for all subsets $D_1, \ldots, D_p$ satisfying $\bigcup_{i=1}^{p} D_i = D$ and for all instances $A_1, \ldots, A_k \subseteq D$

$$g(A_1, \ldots, A_k) = \bigwedge_{i=1}^{p} g(A_1 \cap D_i, \ldots, A_k \cap D_i).$$

- The operation g is *trivial* if there exists a *projection set* $J_g \subseteq \{1, \ldots, k\}$ such that

$$g(A_1, \ldots, A_k) = \text{YES} \Leftrightarrow \bigwedge_{i \in J_g} (A_i = \emptyset).$$

Let us give some examples of decision problems and their appropriate classification. The problems of deciding whether two sets are equal, disjoint, or whether the first set contains the second can be described by the truthsets $T_{\text{EQ}} = \{00, 11\}$, $T_{\text{DIS}} = \{00, 01, 10\}$ and $T_{\text{INC}} = \{00, 10, 11\}$, respectively. The problem χ_D, deciding whether the input elements comprise the whole domain, can be defined as an template operation with $T_{\chi D} = \{01, 10, 11\}$. Note that χ_D is not conserving.

The nature of the duality between the properties of decision and computational problems is described by the following definition.

Definition 6.3. Let f be a k-ary computational set operation. Its *dual decision problem* $\hat{f}$ is defined by

$$\hat{f}(A_1, \ldots, A_k) = \text{YES} \Leftrightarrow f(A_1, \ldots, A_k) = \emptyset.$$

With this definition, the following lemma is immediate.

Lemma 6.4. *Let f be a k-ary computational set operation. Then f is a template (anonymous, accumulative) operation if and only if its dual decision operation $\hat{f}$ is a template (anonymous, accumulative) operation. If f is a template operation with truthset T_f then $T_f = \{0, 1\}^k \setminus T_f$. In particular, f is trivial with projection set J_f if and only if $\hat{f}$ is trivial with the same projection set.*

We proceed with some additional examples. As mentioned above, set complementation is not conserving (COMP^R may be considered as the nonconserving version of SUB, with the first operand fixed to be $\{1, \ldots, R\}$). All operations defined so far in this paper are anonymous. The following operation is not: $\chi_Z(A) = A \cap Z$, for some fixed set Z satisfying $\emptyset \subset Z \subset D$.

An interesting example of a nontemplate operation is the decision problem $odd(A)$ defined by $odd(A) = \text{YES}$ iff $|A|$ is odd.

The operation $\chi_Z(A)$ is not a template operation either, as a consequence of the next easy lemma.

Lemma 6.5. *Let f be a template (computational or decision) set operation. Then f is anonymous.*

Proof. The assertion follows immediately from the fact that for all a, a',

$$a \in A \Leftrightarrow a' \in A|_a^{a'}. \qquad \square$$

The following lemma characterizes the class of basic set operations.

Lemma 6.6. *Let f be a k-ary conserving computational set operation. Then f is a template operation (hence, basic) if and only if it is both accumulative and anonymous.*

Proof. Assume first that f is a template operation. By Lemma 6.5, f is anonymous. To see that f is accumulative, let $D' \subset D$, and let $a \in D$. We need to show that

$$a \in f(A_1 \cap D', \ldots, A_k \cap D') \Leftrightarrow a \in f(A_1, \ldots, A_k) \cap D'.$$

There are two cases to consider. If $a \notin D'$ then, on one hand, $a \notin f(A_1, \ldots, A_k) \cap D'$ and, on the other hand, $a \notin f(A_1 \cap D', \ldots, A_k \cap D')$, by the fact that f is conserving.

Suppose now that $a \in D'$. If $a \in f(A_1 \cap D', \ldots, A_k \cap D')$, then there exists some word $w = b_1 \ldots b_k \in T_f$ such that $a \in A_i \cap D'$ iff $b_i = 1$ for all $1 \leqslant i \leqslant k$. Since $a \in D'$, we have $a \in A_i \Leftrightarrow b_i = 1$ and, therefore, $a \in f(A_1, \ldots, A_k) \cap D'$. The argument can be reversed in the case of $a \in f(A_1, \ldots, A_k) \cap D'$ to show that necessarily $a \in f(A_1 \cap D', \ldots, A_k \cap D')$.

For the other direction of the lemma, assume f is accumulative and anonymous. Define the set T_f in the following way. Let a' be any fixed element in D. Denote by P the set of all 2^k k-tuples consisting of the singletons $\{a'\}$ and empty sets, i.e., $P = \{\{a\}, \emptyset\}^k$. For every such tuple $p = (B_1, \ldots, B_k) \in P$ define $w_p = (b_1 \ldots b_k)$ by

$$b_i = \begin{cases} 1 & \text{if } B_i = \{a'\}, \\ 0 & \text{if } B_i = \emptyset, \end{cases}$$

for all $1 \leqslant i \leqslant k$. Now define

$$T_f = \{w_p \mid p \in P \text{ and } f(p) = \{a'\}\}.$$

To see that f is template with truthset T_f, let $(A_1, \ldots, A_k)$ be any k-typle of subsets of D. Denote $p = (A_1|_a^{a'} \cap \{a'\}, \ldots, A_k|_a^{a'} \cap \{a'\})$. Clearly, $p \in P$. By the hypothesis f is anonymous and accumulative; hence, for all $a \in D$, $a \in f(A_1, \ldots, A_k)$ if and only if $a' \in f(p)$, and this is true iff $w_p \in T_f$. $\square$

Corollary 6.7. *Let g be a k-ary conserving decision set operation. Then g is basic if and only if g is accumulative and anonymous.*

The following immediate lemma characterizes the template operations by their truthsets. Let $\cdot$ denote concatenation of bit strings.

Lemma 6.8. *Let f be a k-ary computational set operation. Then*

(1) *f is basic if and only if it is a template operation with truthset T_f such that $0^k \notin T_f$.*

(2) *f is trivial if and only if it is basic with truthset T_f and projection set J_f such that*

$$T_f = \bigcup_{i \in J_f} \{0,1\}^{i-1} \cdot 1 \cdot \{0,1\}^{k-i}.$$

Proof. Let us prove claim (1). Assume that f is conserving. Then $f(\emptyset, \ldots, \emptyset) = \emptyset$ and, hence, $0^k \notin T_f$. Assume now $0^k \notin T_f$. We need to show that

$$a \in f(A_1, \ldots, A_k) \Rightarrow a \in \bigcup_{i=1}^{k} A_i.$$

By the definition of template operations, $a \in f(A_1, \ldots, A_k)$ implies the existence of a word $w = b_1 \ldots b_k \in T_f$ such that $a \in A_i \Leftrightarrow b_i = 1$ and, since $w \neq 0^k$, there exists an index $1 \leqslant j \leqslant k$ such that $a \in A_j$; hence, $a \in \bigcup_{i=1}^{k} A_i$.

Claim (2) is immediate from the definitions. □

Corollary 6.9. *Let g be a k-ary decision set operation. Then*

(1) *g is basic if and only if it is a template operation with truthset T_g such that $0^k \in T_g$.*

(2) *g is trivial if and only if it is basic with truthset T_g and projection set J_g such that*

$$T_g = \bigcap_{i \in J_g} \{0,1\}^{i-1} \cdot 0 \cdot \{0,1\}^{k-i}.$$

Figure 1 summarizes graphically the classification of set operations. (It deals with computational operations, the classification of decision operations being dual.) Lemma 6.5 shows that the class of template operations is contained in the class of anonymous operations. Complementation is the obvious template nonconserving operation. The predicate $odd(A)$ defined below demonstrates the strictness of the inclusion. Define

$$odd'(A) = \begin{cases} A & \text{if } odd(A), \\ \emptyset & \text{otherwise,} \end{cases}$$

and

$$odd''(A) = \begin{cases} \text{Comp}(A) & \text{if } odd(A), \\ D & \text{otherwise.} \end{cases}$$

Clearly, odd' is anonymous, nontemplate and conserving, whereas odd'' is anonymous, nontemplate and nonconserving. The class of accumulative operations was defined as a subclass of the conserving operations. Lemma 6.6 shows that the basic operations can be defined equivalently as either the conserving template operations, or as the accumulative anonymous operations and, further, that they are *precisely* the accumulative template operations. In other words, it follows that there are no template operations that are conserving but nonaccumulative, neither are there accumulative operations that are anonymous but nontemplate. An example of an accumulative, yet not template, operation is $\chi_Z(A)$, as defined above. Lastly, an example of a conserving,

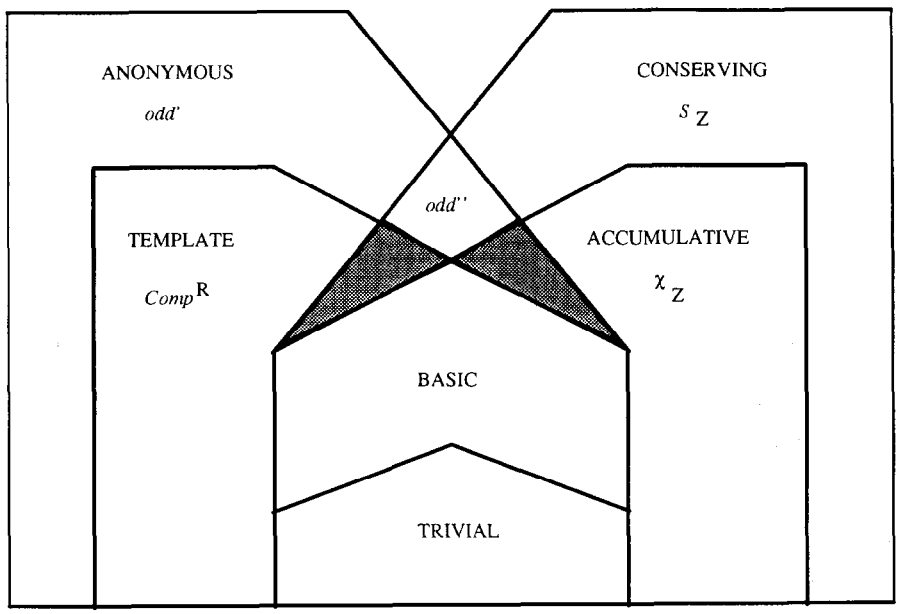

Fig. 1. Schematic classification of set operations. (The shaded areas represent empty subclasses).

nonaccumulative, nonanonymous operation is the following. Let Z be a fixed, nonempty subset of the domain ($\emptyset \subset Z \subset D$).

$$s_Z(A) = \begin{cases} A \cap Z & \text{if } odd(A), \\ A & \text{otherwise.} \end{cases}$$

6.2. A time–space lower bound for nontrivial basic operations

We conclude this section by showing that all nontrivial basic set operations are hard.

Theorem 6.10. *Let f be a k-ary basic computational set operation. Then one of the following holds:*

(i) *f is trivial.*
(ii) COMP^R *can be reduced to f.*
(iii) IS *can be reduced to f.*

Proof. Without loss of generality, consider the first operand of f. Denote the truthset of f by T_f. Let $u = b_2 \ldots b_k \in \{0,1\}^{k-1}$. There are the following cases to consider.

If $0 \cdot u \in T_f$, and $1 \cdot u \notin T_f$, then COMP^R can be reduced to f in the following way. Let A be an instance of COMP^R. Define the instance of f by setting $A_1 = A$, and letting

$$A_i = \begin{cases} D & \text{if } b_i = 1, \\ \emptyset & \text{if } b_i = 0, \end{cases}$$

for $2 \leqslant i \leqslant k$. Clearly, $f(A_1, \ldots, A_k) = D \setminus A = \text{COMP}^R(A)$.

Otherwise, it is the case that for any $u \in \{0, 1\}^{k-1}$ we have that if $0 \cdot u \in T_f$, then $1 \cdot u \in T_f$. If this is the case for all operands, then there are some "minimal" words in T_f, in the sense that if a 1 bit of such a word is flipped, then the result is not in T_f. If T_f can be expressed by a collection of such minimal words $\{w_i\}_{i=1}^{l}$, satisfying $\| w_i \| = 1$ for all $1 \leqslant i \leqslant l$, then f is trivial (with projection set defined by the indices of the 1 bits in these minimal words).

The remaining case is where there exists a minimal word $w = b_1 \ldots b_k$ (in the above sense), with $\| w \| > 1$. Then set intersection is reducible to f by the following construction. Let i_0 be the smallest index for which $b_{i_0} = 1$, and let (A, B) be an instance of IS. Define the instance for f by setting $A_{i_0} = A$, and for $i \neq i_0$ defining

$$A_i = \begin{cases} B & \text{if } b_i = 1, \\ \emptyset & \text{if } b_i = 0. \end{cases}$$

When $A_1, \ldots, A_k$ are defined this way, we have $f(A_1, \ldots, A_k) = A \cap B$. □

An analogous proof shows that all basic nontrivial decision set operations are hard.

Theorem 6.11. *Let g be a k-ary basic decision set operation. Then one of the following holds:*

(i) *g is trivial.*
(ii) DIS *can be reduced to g.*
(iii) EQ *can be reduced to g.*

Proof. First note that if $k = 1$, then g is trivial. So, assume $k \geqslant 2$. Denote the truthset of g by T_g, and let $u = b_3 \ldots b_k \in \{0, 1\}^{k-2}$. Without loss of generality, we consider the first two operands of g.

If $01 \cdot u \notin T_g$ and $11 \cdot u \in T_g$, then EQ reduces to g: given an instance (A, B) of EQ, define $A_1 = A$, $A_2 = B$, and for $i \geqslant 2$ define

$$A_i = \begin{cases} A & \text{if } b_i = 1, \\ \emptyset & \text{if } b_i = 0. \end{cases}$$

These definitions ensure us that $g(A_1, \ldots, A_k) = \text{EQ}(A, B)$.

Otherwise, if for all $u \in \{0, 1\}^{k-2}$, $11 \cdot u \in T_f$ implies that $01 \cdot u \in T_f$, and this holds for all pairs of operands, then it is the case that there are some "maximal" words in T_g, in the sense that flipping any of their 0 bits to 1 yields a word not in T_g. Let $u = b_1 \ldots b_k$ be such a maximal word with the minimum $\| w \|$. If there is only one such maximal word, then g is trivial, with projection set defined by the indices of the 0 bits in this word.

So, suppose the existence of some $u = b'_1 \ldots b'_k \in T_g$ such that $\| u' \| = \| u \|$. Consider the word $u'' = b''_1 \ldots b''_k$ defined by $b''_i = 1$ iff $b_i = 1$ or $b'_i = 1$. By the maximality of u, it follows

that $u'' \notin T_g$. Now, let (A, B) be an instance of DIS. Define

$$A_i = \begin{cases} A & \text{if } b_i = 1 \text{ and } b_i' = 0, \\ B & \text{if } b_i = 0 \text{ and } b_1' = 1, \\ B & \text{if } b_i = b_i' = 1, \\ \emptyset & \text{if } b_i = b_i' = 0. \end{cases}$$

By the construction, $g(A_1, \ldots, A_k) = \text{DIS}(A, B)$. □

7. Upper bounds

This section contrasts the picture depicted by Sections 3–6 by presenting upper bounds for the time–space product required by set operations. First we show that *all* set operations can be computed in linear time and space in the R-way model and, hence, the best lower bound one can hope to establish on the time–space product for any set operation in this model is $TS = \Omega(n^2)$. We proceed with a scheme of RAM algorithms that compute set problems in time–space product $TS = O(n^2 \log n)$ for the natural class of accumulative operations.

7.1. Quadratic upper bound on arbitrary set operations

Theorem 7.1. *Let f denote an arbitrary k-ary set operation. Let n be the total number of input elements. Then f can be computed by an R-way program for $R = O(n)$ with $T = n$ and $S = O(n)$.*

Proof. Assume first $k = 1$, and consider the following algorithm for computing $f(A)$.

(1) Determine the contents of A.

(2) Output $f(A)$.

Determining the contents of A is carried out in the following levelled way. Let $0 \leqslant i < |A|$ denote the level (i.e., step) number. All nodes in level $i-1$ query the variable x_i. The nodes at level $i-1$ represent all possible sets consisting of i elements. The only node at level $|A|$ is a sink. The edges are defined in the obvious way. The number of nodes required to determine A this way is

$$\sum_{i=0}^{|A|} \binom{R}{i} \leqslant 2^R.$$

If $k > 1$, determine the contents of $A_2, \ldots, A_k$ in the same fashion, and attach a copy of the subprogram determining A_{j+1} to every node at the last level determining A_j for $1 \leqslant j < k$.

The output is made in the last step, when the contents of $A_1, \ldots, A_k$ is fully known. The time required is

$$T = \sum_{i=1}^{k} |A_i| = n,$$

and the capacity of the program is

$$S = O(\log(2^{kR})) = O(n). \qquad \square$$

We remark that the quadratic upper bound for an arbitrarily defined set operation is achievable because in the R-way model, there is no requirement to specify basic moves. This implies the absence of the log n factor (in this model, for example, sorting can be computed in linear time.) Another point to be noted is that if $n = o(R)$, then applying the scheme above yields $TS = O(n^2 \log R)$.

However, since the main interest in upper bounds concerns uniform, structured models (in which a program consists of a limited repertoire of basic moves, and one program solves a problem for any input length), we must settle for another $\log n$ factor, and only for "reasonable" functions.

7.2. Accumulative operations and time–space tradeoffs

It is clear that accumulative operations can be computed using any storage amount

$$\log R \leqslant S \leqslant R \log R,$$

by partitioning the domain D into blocks small enough to reside in the workspace, and computing the results for each block successively. Hence, by Lemma 6.6 we have that the following generic RAM algorithm applies to all basic set operations. The algorithm depends on a parameter S for controlling the time–space tradeoff. Assume that the domain is $D = \{1, \ldots, R\}$, where $R = O(|A_1| + \cdots + |A_k|)$, and $A_m = \{a_{m_1}, \ldots, a_{m_{|A_m|}}\}$ for $m = 1, \ldots, k$.

The algorithm uses a bit array B of size $S \times k$, and a fixed truth set T_f. We denote by $B[i]$ the string of bits $B[i] \ldots B[i, k]$.

For $j = 0$ to $\lfloor R/S \rfloor$ **do**:

1. **For** each set operand A_m **do**:
   Let $l = a_{m_i} \bmod S + 1$.
   if $jS < a_{m_i} \leqslant (j+1)S$ **then** $B[l, m] = 1$ **else** $B[l, m] = 0$.
2. **For** $1 \leqslant l \leqslant S$ **do**:
   if $B[l] \in T_f$ **then** output $jS + l$.

Clearly, the running time satisfies

$$T = O\left(\frac{R}{S} n\right) = O\left(\frac{n^2}{S}\right),$$

and the capacity required is $O(S)$.

Remarks.

- The result applies only to basic operations since the anonymity of f is required for computing in a uniform model. In a nonuniform model, any accumulative operation can be computed in sublinear space with time–space product $O(n^2 \log n)$.
- The above algorithm has the same asymptotic complexity for nonconserving template operations, so long as $R = O(n)$.

We now turn to the case when $n = o(R)$. We present a generic RAM algorithm that computes any basic operation in space $O(S)$ and time

$$T = O\left(\frac{n^2 \log^2 S}{S}\right) = O\left(\frac{N^2}{S}\right),$$

for all $\log n \leqslant S \leqslant n$, where n is the total number of input elements and N is the total length of the input. We continue using the bit array B as before, and we use another pointer array C of size S, where each entry is capable of storing a value in the range $\{1, \ldots, S\}$.

For $i = 0$ to $\lfloor n/S \rfloor$ **do**:

1. Initialize all bits in B to 0.
2. Sort the input elements numbered $x_{iS+1}, \ldots, x_{(i+1)S}$, storing their relative order in C.
3. **For** every input element a **do**:
   if $a \in C$ **then** mark the corresponding bit in B.
4. **For** $1 \leqslant l \leqslant S$ **do**:
   if $B[l] \in T_f$ **then** output $x_{iS+C[l]}$.

The question whether $a \in C$ in step 3 is answered by a binary search.

8. Conclusion

The key difficulty in computing a template set operation is the disorder in which the elements may appear. Indeed, if the sets are given in any sorted way, then all the template set operations could be computed trivially, i.e., in linear time and logarithmic space.

The only binary operations for which we were able to establish a tight bound in a general model are COMP^R and its derivatives, SUB and XOR. Nevertheless, we conjecture that all nontrivial template binary set operations admit the bound $TS = \Omega(nm)$, where $|A| = n$ and $|B| = m$.

Many interesting questions that concerns the classification of set operations are left open. Extending the classification to "composite" operations is the next natural step. Extending the scheme should be considered too. On the one hand, our classification does not apply to multiset inputs and, on the other hand, it does not restrict the output to be a set. Our partial results give rise to the question whether a "comprehensive" classification can be defined in a way that corresponds to the (known and

conjectured) complexity bounds. Lastly, it is interesting whether a "unifying" classification can be defined, deleting the distinction between computational and decision problems.

As to the model of R-way branching programs, there still exists the problem of establishing a lower bound for a decision problem (or, loosely speaking, bounding TS away from $\Omega(rn)$, where n is the number of inputs and r is the number of outputs).

References

[1] K. Abrahamson, Time–space tradeoffs for branching programs contrasted with those for straight-line programs, in: *Proc. 27th IEEE Symp. on Foundations of Computer Science* (1986) 402–409.

[2] P. Beame, A general sequential time–space tradeoff for finding unique elements, in: *Proc. 21st ACM Symp. on Theory of Computing* (1989) 197–203.

[3] A. Borodin and S. Cook, A time–space tradeoff for sorting on a general sequential model of computation, *SIAM J. Comput.* **11** (1982) 287–297

[4] A. Borodin, F. Fich, F. Meyer auf der Heide, E. Upfal and A. Wigderson, A time–space tradeoff for element distinctness, *SIAM J. Comput.* **16** (1987) 97–99.

[5] A. Borodin, M.J. Fischer, D.G. Kirkpatrick, N.A. Lynch and M. Tompa, A time–space tradeoff for sorting on non-oblivious machines, *J. Comput. System Sci.* **22** (1981) 351–364.

[6] A. Cobham, The recognition problem for the set of perfect squares, in: *Proc. 7th IEEE Symp. on Switching and Automata Theory* (1966) 78–87.

[7] M. Karchmer, Two time–space tradeoffs for element distinctness, *Theoret. Comput. Sci.* **47** (1986) 237–246.

[8] S. Reisch and G. Schnitger, Three applications of Kolmogorov complexity, in: *Proc. 23rd IEEE Symp. on Foundations of Computer Science* (1982) 45–52.

[9] A.C. Yao, On time–space tradeoff for sorting with linear queries, *Theoret. Comput. Sci.* **19** (1982) 203–218.

[10] A.C. Yao, Near-optimal time–space tradeoff for element distinctness, in: *Proc. 29th IEEE Symp. on Foundations of Computer Science* (1988) 91–97.

[11] Y. Yesha, Time–space tradeoffs for matrix multiplication and the discrete Fourier transform on any general sequential random access computer, *J. Comput. System Sci.* **29** (1984) 183–197.