Dyn-Fo-
A Dynamic Complexity Class

Sushant Patnaik & Neil Immerman

CMPSCI Technical Report 96-24
April, 1996

Abstract

Traditionally, computational complexity has considered only static problems. Classical Complexity Classes such as NC, P, NP, and PSPACE are defined in terms of the complexity of checking – upon presentation of an entire input – whether the input satisfies a certain property. These complexity classes are not completely appropriate for database systems. Unfortunately, appropriate Database Complexity Classes have not yet been defined. This thesis makes a step towards correcting this situation.

Here, we introduce the complexity class, Dynamic First-Order Logic (Dyn-FO). We define it to be the set of dynamic problems that can be expressed in first-order logic. What this means is that we maintain a database of relevant information so that the action invoked by each insert, delete, and query is first-order expressible.

This corresponds to the sets of properties that can be maintained and queried in first-order logic, i.e., relational calculus, on a relational database. This is very natural in the database setting. In fact, Dyn-FO is really the set of queries that are computable in a traditional first-order query language.

Our results shed light on some of the interesting differences between static and dynamic complexity. We show the surprising fact that a wealth of problems, including connectivity, are in Dyn-FO. Thus, considered as dynamic problems – and that is what database problems are – these problems are already first-order computable. The problems we show to be in Dyn-FO include: reachability in undirected graphs, maintaining minimum spanning forest, k-edge connectivity and bipartiteness. All regular languages are shown to be in Dyn-FO. We even show that decent approximation algorithms for several NP complete problems are in Dyn-FO. The static versions of all these problems are, of course, not first-order.

1 Introduction

In our view, the main two differences between database complexity and traditional complexity are:

1. Databases are **dynamic**. The work to be done consists of a long sequence of small updates and queries to a large database. Each update and query should be performed very quickly in comparison to the size of the database.
2. Computations on databases are for the most part **disk access bound**. The cost of computing a request is usually tied closely to the number of disk pages that must be read or written to fulfill the request.

Of course, a significant percentage of all uses of computers have the above two features. In this thesis, we focus on the first issue. Dynamic Complexity is quite relevant in most day to day tasks. For example: Texing a file, Compiling a program, Processing a visual scene, Performing a complicated calculation in Mathematica, etc. Yet an adequate theory of dynamic complexity is lacking. (Recently, there have been some significant contributions in this direction, e.g. [MSTV93]. Note that dynamic complexity is different although somewhat related to On-line complexity which is receiving a great deal of attention lately.)

For many, if not most, applications of computers including: databases, text editors, program development, it is more appropriate to model the process as a dynamic one. There is a fairly large object being worked on over a period of time. The object is repeatedly modified by users and computations are performed. Thus a dynamic algorithm for a certain class of queries is one that can maintain an input object, e.g. a database, and process changes to the database as well as answering queries about the current database.

11. Related Work

In [DST93], Dong, Su and Topor consider the incremental evaluation problem for Datalog queries, namely, repeatedly evaluating the same Datalog query to a database that is being updated between successive query requests. They define a first order incremental evaluation system (FOIES), with respect to a given Datalog query, where the incremental evaluation is carried out by a non-recursive Datalog program. They point out that non-recursive Datalog programs are much better than recursive ones using elaborate data structures for database applications, since they reduce the number of relational join operations.

Our approach is similar to their approach in the sense that both store derived relations for reuse after updates. However, Dyn-FO is a general complexity class and it involves both insertions and deletions. Monotone Dyn-FO (which allows only insertions) is equivalent to FOIES. Defining the arity of a Dyn-FO expression as the arity of the auxiliary (non-input) relations used in the first order logic formulae for handling updates, we see that the notion of *space-free* FOIES is related to devising minimum arity Dyn-FO[1] expressions, in the sense that problems in *space-free* FOIES, by definition, have minimum arity Dyn-FO expressions.

Previously, Dong and Su, in [DS93] have shown that reachability in directed, acyclic graphs and in function graphs is in Dyn-FO. But they do not consider a general framework for dynamic complexity.

[1]We shall use Dyn-FO interchangably to denote a class of decision problems and a language.

The incremental approach, namely, to use the difference between successive database states and the answer to the query in one state to reduce the cost of evaluating the query in the next state, plays a role in maintaining materialized views upon updates ([J92], [GMS93], [I85]), and in integrity constraint simplification ([LST87], [N82]). In these studies, the authors investigate how to maintain first-order definable views efficiently under updates to the underlying database. In our framework, we can interpret their approaches as determining ways to implement fast Dyn-TIME solutions for a subclass of Dyn-FO. For example, [GMS93] show that a very restricted subclass of Dyn-FO is in Dyn-TIME[1].

The design of dynamic algorithms is an active field. See, for example, [E*92a], [E*92b], [R92], [CT91], [F83a], [F83b] amongst others. Our work is also informed by [MSTV93] which does some of the ground work for a complexity theory of dynamic complexity.

This chapter is organized as follows. In section 2, for any static complexity class $\mathcal{C}$, we define the corresponding dynamic class, Dyn-$\mathcal{C}$. The class Dyn-FO is the case we emphasize. Then, in section 3, we present several of the above mentioned Dyn-FO algorithms.

2 Dynamic Complexity Classes

We think of an implementation of a problem $S \subseteq \text{STRUCT}[\sigma]$ as a mapping, I, from $\text{STRUCT}[\sigma]$ to $\text{STRUCT}[\tau]$ where $T \subseteq \text{STRUCT}[\tau]$ is an easier problem. The map I should be a many-one reduction from S to T meaning that any structure $\mathcal{A}$ has the property S iff $I(\mathcal{A})$ has the property T. (Actually, in our definition below, the mapping I will map a sequence of inserts and deletes $\bar{r}$ to a structure. In the interesting special case when $I(\bar{r})$ depends only on the corresponding structure $\mathcal{A}$ and not which sequence of inserts and deletes created it, we call I *memoryless.*)

We are thinking and talking about a structure $\mathcal{A} \in \text{STRUCT}[\sigma]$, but the structure that we actually have in memory and disk and are manipulating is $I(\mathcal{A}) \in \text{STRUCT}[\tau]$. In this way, each insert or delete on $\mathcal{A}$ is interpreted as a corresponding series of actions on $I(\mathcal{A})$. The fact that I is a many-one reduction insures that the query asking whether $\mathcal{A} \in S$ can be answered according to whether $I(\mathcal{A}) \in T$.

In traditional static complexity, the entire input structure $\mathcal{A}$ is fixed and we are interested in deciding whether $A \in S$ for a relevant property, S. In the dynamic case, the structure changes over time. The actions we have in mind are a sequence of insertions and deletions of tuples in the input relations. We will usually think of our dynamic structure, $\mathcal{A} = \langle\{0, 1, \ldots n-1\}, R_1, \ldots, R_s, c_1, \ldots, c_t\rangle$, as having a fixed size potential universe, $|\mathcal{A}| = \{0, 1, \ldots n-1\}$, and a unary relation R_1, specifying the elements in the active domain. The initial structure of size n for this vocabulary will be taken to be $\mathcal{A}_0^n = \langle \{0, 1, \ldots n-1\}, \{0\}, \emptyset, \ldots, \emptyset, 0, \ldots, 0 \rangle$, having $R_1 = \{0\}$ indicating that the single element 0 is in the active domain and all the other relations are empty.

Next we give the formal definition of dynamic complexity classes. The issue is that the structure $I(\mathcal{A})$ can be updated efficiently in response to any insert or delete to $\mathcal{A}$. In particular, if $T \in \text{FO}$ and all such inserts and deletes are first-order computable, then $S \in \text{Dyn-FO}$.

21. Definition of Dyn-$\mathcal{C}$

For any complexity class $\mathcal{C}$ we define its dynamic version, Dyn-$\mathcal{C}$, as follows. Let $\sigma = \langle R_1^{a_1} \ldots R_s^{a_s}, c_1, \ldots, c_t \rangle$ be a vocabulary and let $S \subseteq \text{STRUC}[\sigma]$ be any problem. Let

$$\mathcal{R}_{n,\sigma} = \{\text{ins}(i, \bar{a}), \text{del}(i, \bar{a}), \text{set}(j, a) \mid 1 \le i \le s, \bar{a} \in \{0, \ldots, n-1\}^{a_i}, 1 \le j \le t\}$$

be the set be possible requests to insert tuple $\bar{a}$ into the relation R_i, delete tuple $\bar{a}$ from relation R_i, or set constant c_j to a.

Let $\text{eval}_{n,\sigma} : \mathcal{R}_{n,\sigma}^\star \to \text{STRUCT}[\sigma]$ be the naturally defined evaluation of a sequence of requests, initialized by $\text{eval}_{n,\sigma}(\emptyset) = \mathcal{A}_0^n$.

Define, $S \in \text{Dyn-}\mathcal{C}$ iff there exist another problem $T \subset \text{STRUC}[\tau]$ such that $T \in \mathcal{C}$ and there exist maps,

$$f : \mathcal{R}_{n,\sigma}^\star \to \text{STRUCT}[\tau]; \quad g : \text{STRUCT}[\tau] \times \mathcal{R}_{n,\sigma} \to \text{STRUCT}[\tau]$$

satisfying the following properties.

1. For all $\bar{r} \in \mathcal{R}_{n,\sigma}^\star$, $(\text{eval}_{n,\sigma}(\bar{r}) \in S) \Leftrightarrow (f(\bar{r}) \in T)$
2. For all $s \in \mathcal{R}_{n,\sigma}$, and $\bar{r} \in \mathcal{R}_{n,\sigma}^\star$, $f(\text{eval}_{n,\sigma}(\bar{r}s)) = g(f(\text{eval}_{n,\sigma}(\bar{r})), s)$
3. $\|f(\bar{r})\| = \|\text{eval}_{n,\sigma}(\bar{r})\|^{O(1)}$, where for any structure, $\mathcal{A}$, $\|\mathcal{A}\|$ denotes the size of $\mathcal{A}$ [2]
4. The functions g and the initial structure $f(\emptyset)$ are computable in complexity $\mathcal{C}$, (as a function of n).

[2]This expects that the complexity class $\mathcal{C}$ is closed under polynomial increases in the input size. For more restricted classes $\mathcal{C}$, such as linear time, we insist that $\|f(\bar{r})\| = O(\|\text{eval}_{n,\sigma}(\bar{r})\|)$.

We will say that the above map f is *memoryless* if the value of $f(\bar{r})$ depends only on $\mathrm{eval}_{n,\sigma}(\bar{r})$.

In the above, if only inserts and queries are considered, i.e., no deletes, then we get the class Dyn$_s$-$\mathcal{C}$, the semi-dynamic version of $\mathcal{C}$. One can also consider amortized versions of these two classes. Furthermore, there are some cases where we would like extra, but polynomial, precomputation to compute the initial structure $f(\emptyset)$. If we relax condition (4) in this way, then the resulting class is called Dyn-$\mathcal{C}^+$ – Dyn-$\mathcal{C}$ with polynomial precomputation.

We have thus defined the dynamic complexity classes Dyn-$\mathcal{C}$ for any static class, $\mathcal{C}$. Two particularly interesting examples are Dyn-FO and Dyn-TIME$[t(n)]$ for $t(n) \in o(n)$, where the latter is the set of problems computable dynamically on a RAM (with word size $O(\log n)$) in time $t(n)$.

In [TY79], Tarjan and Yao propose a dynamic model whose complexity measure is the number of probes into a data structure and any other computation is for free. The idea is to be able to optimize the number of disk input-outputs. There are two versions of this model: Bit Probe model, in which the word size of the RAM is 1 bit and the Uniform Probe model in which the word size is $O(\log n)$. Recently Miltersen in [M2] renewed investigation into the Bit probe model and got some interesting results. In our framework, Dyn-PROBE$[t(n)]$, and Dyn-BIT-PROBE$[t(n)]$ refer to the set of problems computable dynamically on a RAM with word size $O(\log n)$ and $O(1)$, respectively, making at most $t(n)$ probes.

3 Problems in Dyn-FO

Let graph reachability denote the following problem: given a graph, G, and vertices x, y, determine if there is a path from x to y in G. We shall use 1GAP, UGAP, GAP (acyclic), respectively, to denote graph reachability on directed graphs with out-degree at most 1, undirected graphs and acyclic directed graphs where the inserts preserve acyclicity. It is well known that the graph reachability problem is not first-order expressible and this has often been used as a justification for using database query languages more powerful than FO [CH1]. Thus, the following two theorems are striking.

Theorem 3.1 *UGAP is in* Dyn-FO.

Proof: We maintain a spanning forest of the underlying graph via relations, $\mathrm{F}(x, y)$ and $\mathrm{PV}(x, y, u)$ and the input relation, E. $\mathrm{F}(x, y)$ means that the edge (x, y) is in the current spanning forest. $\mathrm{PV}(x,y,u)$ means that there is a (unique) path in the forest from x to y via vertex u. The vertex, u, may be one of the endpoints. So, for example, if $\mathrm{F}(x,y)$ is true, then so are $\mathrm{PV}(x,y,x)$ and $\mathrm{PV}(x,y,y)$. We maintain the undirected nature of the graph by interpreting insert(E,a,b) or delete(E,a,b) to do the operation on both (a,b) and (b,a).

Insert(E,a,b): We denote the updated relations as E′, F′ and PV′. In the sequel, we shall use $\mathrm{P}(x,y)$ to abbreviate $(x = y \vee\ \mathrm{PV}(x,y,x))$, and $\mathrm{Eq}(x,y,c,d)$ to abbreviate the formula,

$$((x = c \wedge y = d) \vee (x = d \wedge y = c)).$$

Maintaining the input edge relation is trivial:

$$\mathrm{E}'(x,y) \equiv\ \mathrm{E}(x,y) \vee\ \mathrm{Eq}(x,y,a,b)$$

The edges in the forest remain unchanged, if vertices, a and b, were already in the same connected component. Otherwise, the only new forest edge is (a,b).

$$\mathrm{F}'(x,y) \equiv\ \mathrm{F}(x,y) \vee (\mathrm{Eq}(x,y,a,b) \wedge \neg\mathrm{P}(a,b))$$

Now all that remains is to compute PV′. The latter changes iff edge (a,b) connects two formerly disconnected trees. In this case, the new tuples (x,y,z) have x and y coming from one each of the trees containing a and b.

$$\begin{aligned}\mathrm{PV}'(x,y,z) \equiv\ & \mathrm{PV}(x,y,z) \vee (\mathrm{Eq}(x,y,a,b) \wedge (z = a \vee z = b)) \\ & \vee(\neg\mathrm{P}(x,y) \wedge (\exists u,v)\ \mathrm{Eq}(u,v,a,b) \wedge P(x,u) \wedge P(v,y) \\ & \qquad \wedge(z = a \vee z = b \vee\ \mathrm{PV}(x,u,z) \vee \mathrm{PV}(v,y,z)))\end{aligned}$$

Delete(E,a,b): If edge (a,b) is not in the forest ($\neg\ \mathrm{F}(a,b)$), then the updated relations are unchanged, except that $\mathrm{E}'(a,b)$ is set to false. Otherwise, we first identify the vertices of the two trees in the forest created by the deletion, and then we pick an edge, say e, out of all the edges (if any) that run between the two trees and *insert* e into the forest, updating the relations, PV and F, appropriately.

We define a temporary relation T to denote the PV relation after (a,b) is deleted, before the new edge, e, is inserted.

$$\mathrm{T}(x,y,z) \equiv\ \mathrm{PV}(x,y,z) \wedge \neg(\mathrm{PV}(x,y,a) \wedge\ \mathrm{PV}(x,y,b))$$

Using T, we then pick the new edge that must be added to the spanning forest. $\mathrm{New}(x,y)$ is true if and only if edge (x,y) is the minimum[3] edge that connects the ~~two disconnected components:~~

[3]Note that this uses an ordering on the vertices. If no such ordering is given, then we can order edges by their order of insertion. In either case, this reduction is not *memoryless* because it depends on the history. It is open as to whether there exists a *memoryless* first order formula.

$$\begin{aligned}\mathrm{New}(x,y) \equiv\ & \mathrm{E}(x,y) \wedge \mathrm{T}(a,x,a) \wedge \mathrm{T}(b,y,b)\\ & \wedge (\forall u,v)(\mathrm{E}(u,v) \wedge \mathrm{T}(a,u,a) \wedge \mathrm{T}(b,v,b)) \rightarrow (x < u \vee (x = u \wedge y \leq v))\end{aligned}$$

E′, F′ and PV′ are then defined as follows:

$$\mathrm{E}'(x,y) \equiv \mathrm{E}(x,y) \wedge \neg\mathrm{Eq}(x,y,a,b)$$

We remove (a,b) from the forest and add the new edge.

$$\mathrm{F}'(x,y) \equiv (\mathrm{F}(x,y) \wedge \neg\mathrm{Eq}(x,y,a,b)) \vee \mathrm{New}(x,y) \vee \mathrm{New}(y,x)$$

The paths in the forest, from x to y via z, that did not pass through a and b, are valid. Also, new paths have to be added as a result of the insertion of a new edge in the forest.

$$\begin{aligned}\mathrm{PV}'(x,y,z) \equiv\ & \mathrm{T}(x,y,z) \vee [(\exists u,v)(\mathrm{New}(u,v) \vee \mathrm{New}(v,u)) \wedge \mathrm{T}(x,u,x)\\ & \wedge \mathrm{T}(y,v,y) \wedge (\mathrm{T}(x,u,z) \vee \mathrm{T}(y,v,z))]\end{aligned}$$

∎

We give a new Dyn-FO algorithm for the following result.

Theorem 3.2 ([**DS93**]) *1GAP and GAP (acyclic) are in* Dyn-FO.

Proof: 1GAP follows easily from UGAP. For the GAP (acyclic) case, the inserts are assumed to always preserve acyclicity. We maintain the path relation, $\mathrm{P}(x,y)$ which means that there is a path from x to y in the graph. (see figure 1).

Insert(E,a,b):

$$\mathrm{P}'(x,y) \equiv \mathrm{P}(x,y) \vee (\mathrm{P}(x,a) \wedge \mathrm{P}(b,y))$$

Delete(E,a,b):

$$\begin{aligned}\mathrm{P}'(x,y) \equiv\ & \mathrm{P}(x,y) \wedge [\neg(\mathrm{P}(x,a) \wedge \mathrm{P}(b,y)) \vee (\exists u,v)\ \mathrm{P}(x,u) \wedge \mathrm{P}(u,a) \wedge \mathrm{E}(u,v)\\ & \wedge \neg\mathrm{P}(v,a) \wedge \mathrm{P}(v,y) \wedge (v \neq b \vee u \neq a)]\end{aligned}$$

In the case where there is a path from x to y using the edge (a,b), consider any path not using this edge. Let u be the last vertex along this path from which a is reachable. Note that $u \neq y$ because the graph was acyclic even before the deletion

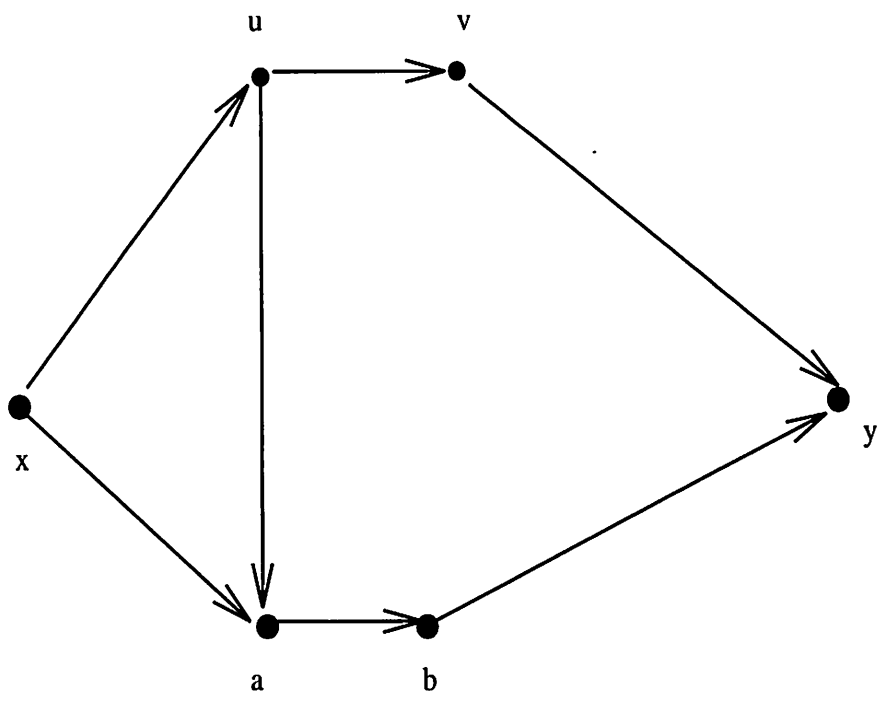

Figure 1: GAP

of edge, (a,b). Thus, the edge, (u,v), described in the above formula must exist and acyclicity insures that the path $x \to u \to v \to y$ does not involve the edge, (a,b). ∎

Let TR denote the following problem: For G, a directed acyclic graph, recall that the *Transitive Reduction*, TR(G), is the minimal subgraph of G having the same transitive closure as G.

Corollary 3.3 *Transitive Reduction for directed acyclic graphs is in memoryless* Dyn-FO.

Proof: We maintain the path relation, P, as in Theorem 3.2.
Insert(E,(a,b)): If $P(a,b)$ already holds, then there is no change. Otherwise, we may have to remove some edges from TR. $TR'(x,y)$ is given by

$$(TR(x,y) \wedge P(a,b)) \vee [\neg P(a,b) \wedge (x = a \wedge y = b) \vee [TR(x,y) \wedge \neg(P(x,a) \wedge P(b,y))]]$$

Delete(E,(a,b)): We have to determine the new edges that might be added in TR. New(x,y) holds if there was a path from x to y via (a,b) and no path of length > 1 remains when (a,b) is deleted.

$$\begin{aligned}\mathrm{New}(x,y) \equiv\ & \mathrm{E}(x,y) \wedge \neg\mathrm{TR}(x,y) \wedge \mathrm{P}(x,a) \wedge \mathrm{P}(b,y) \\ & \wedge (\forall u,v)\ \neg[\mathrm{P}(x,u) \wedge \mathrm{P}(u,a) \wedge \mathrm{E}(u,v) \\ & \wedge \neg\mathrm{P}(v,a) \wedge \mathrm{P}(v,y) \wedge (v \neq b \vee u \neq a)]\end{aligned}$$

$$\mathrm{TR}'(x,y) \equiv (\mathrm{TR}(x,y) \wedge \neg(x = a \wedge y = b)) \vee \mathrm{New}(x,y)$$

∎

Let Minimum Spanning Forest denote the following problem: given an undirected graph G with weights on the edges, determine the minimum weighted spanning forest of G.

Theorem 3.4 *Minimum Spanning Forest is in* Dyn-FO.

Proof: The general idea is to maintain the forest edges and non-forest edges dynamically and to maintain the relations $\mathrm{PV}(x,y,e)$ and $\mathrm{F}(x,y)$ as in the case of UGAP. Let $\mathrm{W}(a,b)$ denote the weight of edge (a,b). The difference from UGAP is that we have to maintain the minimum weighted forest. That changes our update procedures in the following way.

Deletion of an edge, say (a,b), is handled as follows. We determine, using PV, all the vertices that can be reached from a in the tree and all those that can be reached from b. These give the vertices in the two trees that the original tree splits into. Then, instead of choosing the lexicographically first non-forest edge that reconnects the two pieces, we choose the *minimum weight* such edge, and insert it. If there is more than one such minimum edge, then we break the tie with the ordering. PV is updated accordingly to reflect the merging of two disconnected trees into one.

When an edge,say (a,b) is inserted, we determine if there exists a path between a and b. If there is no path, then (a,b) merges two trees into one, and PV is updated as before for UGAP. Otherwise, using PV, we can determine the forest-edges that appear in the unique path in the forest between b and a, and check to see if weight of the new edge, (a,b) is less than the weight of any of these edges. If not, then (a,b) is not a forest edge and nothing changes. Otherwise, let (c,d) be the maximum weight edge on the path from a to b. We make $\mathrm{F}'(c,d)$ false and $\mathrm{F}'(a,b)$ true and update PV accordingly. It is easy to see that if the weights are all distinct, or in the presence of an ordering on the edges, this construction is *memoryless*. ∎

We next show that similar algorithms exist for Bipartiteness, Edge Connectivity, Least Common Ancestor queries (in rooted trees), Maximal Matching and Maximal Independent Set (in bounded degree graphs). These latter two have no known sublinear time fully dynamic solutions.

Theorem 3.5 *Let k be a fixed constant.* Dyn-FO *algorithms exist for the following problems:*

1. *Bipartiteness,*
2. *k-Edge Connectivity,*
3. *Maximal Matching in undirected graphs,*
4. *Least Common Ancestor in rooted trees.*[4]

Proof:

1. We maintain bipartiteness in undirected graphs by maintaining relations $\mathrm{PV}(x,y,z)$ and $\mathrm{F}(x,y)$, as for UGAP, and also, $\mathrm{Odd}(x,y)$, which means that there exists a path of odd length from x to y in the spanning forest. The graph is bipartite iff $(\forall x,y)\ \mathrm{E}(x,y) \rightarrow \mathrm{Odd}(x,y)$.

   We show how to update Odd in Dyn-FO.
   Insert(E,a,b): If the new edge, (a,b) becomes a forest-edge, we determine for all newly connected vertices x and y whether the new path is odd: If on the other hand, (a,b) is added as a non-forest edge, Odd is unchanged.

   $$\begin{aligned}\mathrm{Odd}'(x,y) \equiv \mathrm{Odd}(x,y)\ \vee[\neg\mathrm{PV}(a,b,a)\wedge \\ (\exists u,v)\mathrm{Eq}(u,v,a,b)\wedge\ \mathrm{PV}(x,u,x)\wedge\ \mathrm{PV}(y,v,y) \\ \wedge((\mathrm{Odd}(x,u)\wedge\ \mathrm{Odd}(y,v))\vee(\neg\mathrm{Odd}(x,u)\wedge\neg\mathrm{Odd}(y,v)))]\end{aligned}$$

   Delete(E, a,b): If (a,b) is a non-forest edge, Odd is unchanged. Otherwise, for all vertices x,y, which are in the two disconnected trees that result from deletion of (a,b), make $\mathrm{Odd}(x,y)$ and $\mathrm{PV}(x,y,z)$ false. Then, we select some edge (if any) that spans the disconnected components and insert it in and update Odd and PV exactly as for the insertion case. The Dyn-FO expressions for doing that are as shown before.

2. As before, we maintain the relations, E,F and PV. Insertions and deletions are handled as for UGAP. The query is handled as follows. Since k is constant, we universally quantify over k edges, say, $(x_1,y_1),\ldots,(x_k,y_k)$, and then, for every pair of vertices, x and y, check for a path between x and y in the graph that is obtained after deletion of edges, $(x_1,y_1),\ldots,(x_k,y_k)$, by composing the Dyn-FO formula (for a single deletion) k times.

[4]Rooted trees are trees where all the edges have an orientation toward a fixed vertex called the root.

3. We maintain a maximal matching in DYN-FO by maintaining, under insertions and deletions of edges, a relation, Match(x, y) which means that the edge (x, y) is in the matching. Initially, for the empty graph, Match(x, y) is false for all x and y. As usual, all relations are symmetric. We shall use MP(x) to abbreviate the formula:

$$(\exists z)\mathrm{Match}(x, z)$$

Insert(E,a, b): The matching remains unchanged except that the edge (a, b) is checked to see whether it can be added. Match$'(x, y)$ is given by

$$\mathrm{Match}(x, y) \vee (\mathrm{Eq}(x, y, a, b) \wedge \neg\mathrm{MP}(a) \wedge \neg\mathrm{MP}(b))$$

Delete(E,a, b): If (a, b) is not in the matching, then Match is unchanged. Otherwise, we remove (a, b) from the matching. We pick vertices (the lexicographically minimum) adjacent to a and b, if any, and add the corresponding edge(s) to the new matching. Match$'(x, y)$ is given by

$$(\mathrm{Match}(x, y) \wedge \neg\mathrm{Eq}(x, y, a, b)) \vee (\mathrm{Match}(a, b) \wedge (New(x, y) \vee \ \mathrm{New}(y, x)))$$

where New(x, y) means that $x = a$ and y is the new vertex matched with a or $x = b$ and y is the new vertex matched with b. Let New$_1(u, v)$ denote that $u = a$ and v is the new vertex matched with a.

$$\mathrm{New}_1(u, v) \equiv (u = a \wedge \neg\mathrm{MP}(v) \wedge \ \mathrm{E}(u, v) \wedge (\forall z)[(\mathrm{E}(a, z) \wedge \neg\mathrm{MP}(z)) \rightarrow v \leq z])$$

Then, New(u, v) can be expressed as follows: either New$_1(u, v)$ is true or $u = b$ and v is the new vertex matched with b.

$$\begin{aligned}\mathrm{New}(u, v) \equiv \mathrm{New}_1(u, v) \vee (u = b \wedge \neg\mathrm{MP}(v) \wedge \ \mathrm{E}(u, v) \wedge \ \neg\mathrm{New}_1(z) \\ \wedge(\forall z)[(\mathrm{E}(b, z) \wedge \neg\mathrm{MP}(z) \wedge \neg\mathrm{New}_1(a, z)) \rightarrow v \leq z])\end{aligned}$$

The solution above can be maintained in Dyn-TIME[1] if for each vertex we maintain separate linked lists of the matched and unmatched neighbouring vertices.

4. Least common ancestors in a forest of rooted trees are readily maintained, under deletions and insertions of edges that preserve acyclicity, using PV and the relations, Level(x, i) meaning that vertex x is at level i from the root of the unique tree to which it belongs, and Root$(x) \leftrightarrow$ Level$(x, 0)$, meaning that vertex x is a root of a tree. F is not needed because the inserts preserve the acyclicity of the graph. In the following expressions, we shall frequently use PV(x, y, y)

meaning that there is a path from x to y. Now, LCA(x, y, a) meaning a is the least common ancestor of vertices, x and y iff

$$(\exists i)\ \mathrm{PV}(x,a,a) \wedge\ \mathrm{PV}(y,a,a) \wedge\ \mathrm{Level}(a,i) \wedge$$

$$(\forall b \neq a)((\exists j)\ \mathrm{PV}(x,b,b) \wedge\ \mathrm{PV}(y,b,b) \wedge\ \mathrm{Level}(b,j) \rightarrow j > i)$$

We know how to maintain PV. Level and Root can be easily maintained by first-order formulae. Level changes only if an insert removes a root from the forest, or a delete adds a root to the forest. In these cases, Root changes appropriately. Let New(x) denote that x is a new vertex that is an end point of the edge added in: $\mathrm{New}(x) \equiv (\forall v)\neg \mathrm{E}(v,x) \wedge \neg \mathrm{E}(x,v)$.

Insert(E,a, b): Note that since inserts preserve the acyclicity of the graph, insert never adds an edge between two old nodes in the same tree. There are several other cases to consider. Level(x,i) is easily updated in each case as follows.

- If a is a new node and b is any (root or non-root) old node (the opposite case is analogous): Level is unchanged except for the new vertex.

$$\mathrm{Level}'(x,i) \equiv (x = a \wedge ((\exists j)\mathrm{Level}(b,j) \wedge j+1 = i)) \vee\ \mathrm{Level}(x,i)$$

- If both a and b are new nodes: In this case, edge (a, b) is the sole edge in the tree containing a and b. Level is unchanged except for the levels of nodes, a and b. We arbitrarily choose one to be the root (in this case, the lexicographically smaller node). Then, Level$'(x,i)$ is given by:

$$(x = a \wedge i = 1 \wedge a > b) \vee (x = b \wedge i = 0 \wedge a > b)$$

$$\vee (x = b \wedge i = 1 \wedge b > a) \vee (x = a \wedge i = 0 \wedge b > a) \vee\ \mathrm{Level}(x,i)$$

- If both a and b are roots: We arbitrarily choose the lexicographically smaller one to be the root and increment the levels of all the vertices in the other tree by 1. The level of other vertices is unchanged. Level$'(x,i)$ is as follows:

$$\begin{aligned}&((\exists y)((a > b \wedge y = a) \vee (b > a \wedge y = b)) \wedge \\ &\qquad\qquad (\mathrm{PV}(x,y,y) \wedge (\exists k)\mathrm{Level}(x,k) \wedge\ k+1 = i)) \\ &\quad \vee (x = y \wedge\ i = 1) \vee\ (\mathrm{Level}(x,i) \wedge\ x \neq y \wedge \neg \mathrm{PV}(x,y,y))\end{aligned}$$

- If both a and b are old nodes and say, a is a root node and b is a non-root node (the opposite case is analogous): In this case, the level of nodes in the forest remain unchanged except in b's tree where the levels are recomputed to reflect the new orientation of the tree wherein a is the new root and b is its child, and all the nodes in b's tree are linked to a via b. The new level of any node, say x, in b's tree is computed from the level of b, level of $\text{LCA}(b,x)$ and level of x. Thus, $\text{Level}'(x,i)$ is given by:

  $$\begin{aligned}&(\neg\text{PV}(x,b,b)\wedge \text{ Level}(x,i))\vee(x=b\wedge i=1)\vee\\ &((\exists z,j,k,l)\text{LCA}(x,b,z)\wedge \text{ Level}(b,k)\wedge \text{ Level}(x,j)\wedge \text{ Level}(z,l)\\ &\qquad\qquad\wedge i=j+k-2l)\end{aligned}$$

  In the case when both a and b are non-root old nodes, we choose the lexicographically greater of the two as the parent node and proceed as above, but taking care now that a is not the root and hence it is at some non-zero level, i_0, to begin with.

Delete(E,a,b**)**: In this case, the tree containing a and b breaks into two new trees, one containing a and the other, b. Suppose level of a is greater than b. The opposite case is analogous. Node a becomes a new root of a tree in the forest. The level of the nodes in the other subtrees of b's tree and all other trees remain unchanged. The new level of a is 0 and the new level of any node in its tree is just the previous value minus the previous value of a's level. We need the following predicate: Thus, in this case, $\text{Level}'(x,i)$ is given by

$$(\text{Level}(x,i)\wedge\neg\text{PV}(x,a,a))\vee(\text{Level}(x,i)\wedge \text{ PV}(x,a,b))\vee(x=a\wedge i=0)$$

$$\vee((\exists j,k)\text{Level}(a,j)\wedge \text{ Level}(x,k)\wedge \text{ PV}(x,a,a)\wedge\neg\text{PV}(x,a,b)\wedge i=k-j)$$

∎

It is an interesting phenomenon that constant-approximations to certain NP complete optimization problems can be maintained by FO relational formulae. We will see this in a future paper.

4 Dyn-FO versus NC^1, L and NL

We have shown that 1GAP and GAP(acyclic) are in Dyn-FO. These problems are hard for L and NL respectively via ordinary first order reductions that do not preserve dynamic complexity in general. Hence, it does not follow that NL, or even L, or even NC^1 is contained in Dyn-FO. The relationship between Dyn-FO and even a low level parallel complexity class such as TC^0, that is contained in NC^1, is not clear. Problems such as Addition and Multiplication of two n bit numbers, x, y, which are both in TC^0, are in Dyn-FO, under updates such as Change(x, i, b) or Change(y, i, b) meaning that the i-th bit (from the right/least significant position starting at 0) of x is set to b, for $1 \leq i \leq n, b = 0/1$.

Proposition 4.1 *Multiplication is in* Dyn-FO.

Proof: Given two n bit numbers, x, y, their addition can be expressed in FO $\subseteq$ Dyn-FO. We maintain the product in a bit array, P. Suppose the update operation is Change(x, i, b). (Change (y, i, b) is analogous.) There are two cases:
If the bit is changed from 0 to 1, then P′ is given by shifting y by i bits to the right and then adding it to P. It is easily accomplished by a first-order formula.
If the bit is changed from 1 to 0, then P′ is given by shifting y by i bits to the right and then adding the 2's complement of the resulting number to P. Again this is easily accomplished by a first-order formula. ∎

However, UGAP and Transitive Reduction which are suspected not to be in NC^1 can be expressed in Dyn-FO. We can prove the following, but most relations between Dyn-FO and static complexity classes are open.

Theorem 4.2 *The following classes of problems are in* Dyn-FO*:*

1. *All regular languages,*

2. *D^k, the Dyck language on k parentheses, for any constant k.*

Proof:

1. Given any regular language, let $Q, \Sigma, \Delta, q_0, F$ denote the state set, the alphabet, the transition function, the initial state and the set of final states of the DFA that accepts L. We show two different solutions. The first uses $O(n^2)$ and the second $O(n)$ bits of memory.

   The first idea is to maintain the partial product of the fixed sized mapping $Q \rightarrow Q$ that is induced by every sub-sequence of the input word, $w_i \ldots w_{j-1}$, for all i and j, in a relation, A, and update it in parallel. Let $w = w_1 w_2 \ldots w_n$. Let

$\sigma_i : Q \rightarrow Q$ be the map induced by the transition function on w_i i. e. $\sigma_i(p) = \Delta(p, w_i)$, and let $\sigma_{i,j} : Q \rightarrow Q$ be the map

$$\sigma_i \cdot \ldots \cdot \sigma_j,$$

where $\cdot$ denotes composition. We shall use Boolean constants $\delta_{x,p,q}$ to encode Δ.

For all $x \in \Sigma$ and $p, q \in Q$, $\delta_{x,p,q}$ equals 1 iff $\Delta(p, x) = q$.

A will have the following property: For all $p, q \in Q$ and $n \leq i \leq j \leq 1$,

$$\mathrm{A}(i, j, p, q) \leftrightarrow \sigma_{i,j-1}(p) = q.$$

The query, "Is w in L?", is answered by checking whether

$$\exists f(\mathrm{A}(1, n+1, q_0, f) \wedge f \in F).$$

The update operation is Change(m, x) meaning change the mth input symbol to x, where $1 \leq m \leq n$ and $x \in \Sigma$. Then, A is updated as follows:

$$\begin{aligned}\mathrm{A}'[i,j,p,q] \equiv \quad & (i = j \wedge p = q) \vee [(i < j) \wedge \\ & [((i > m \vee j \leq m) \wedge \ \mathrm{A}(i,j,p,q)) \vee \\ & (i \leq m < j \wedge \\ & \textstyle\bigvee_{s,t \in Q} \ \mathrm{A}(i,m,p,s) \wedge \delta_{x,s,t} \wedge \ \mathrm{A}(m+1,j,t,q))]]\end{aligned}$$

To reduce the number of bits of storage in the second solution, we maintain the fixed sized mapping $Q \rightarrow Q$ that is induced by every symbol of the input word, w ($|w| = n$) at the leaves of a balanced binary tree, and at every internal node the composition of the mappings of its children. The height of the tree is $\log n$. Changing any symbol in the input word, say w_i, changes the induced map on the state set $\sigma_i{:}Q \rightarrow Q$, and in effect that changes all the maps stored in the nodes along the path from the leaf corresponding to i to the root. Since, the information stored at any node is only of constant size ($2|Q|$), on changing symbol, $w[i]$, a first-order formula can guess the $O(\log n)$ bits along the unique path from leaf i to the root and verify it in parallel against the values of the stored tree relation and then update it in parallel (by computing the new composition at the internal nodes).

Assume for convenience that $n = 2^k - 1$ for some k. Let $q = 2|Q|$. Let $\sigma : Q \rightarrow Q$ denote a mapping from Q to Q. We can encode σ by a *block pair* of constant size, viz., $2 \log q$ bits. We shall denote the vertices in the tree as pairs (x, i), for $1 \leq x \leq n$ and $0 \leq i \leq \log n$ and the input bits being $(x, 0)$. We shall use the relations TreePath$(x, 0, y, i)$, Contents(v, i, σ) and Child$(v, i, u, i-1)$ to maintain the balanced binary tree and the contents at each node. TreePath$(x, 0, y, i)$ means that there is a path in the tree from the input node, $(x, 0)$, to node, (y, i). Contents(v, i, σ) means that node, (v, i), stores the mapping, σ. Child$(v, i, u, i-1)$ means that node $(u, i-1)$ is a child of node (v, i). Then, the query, "Is w in L?" is answered by checking whether

$$\bigvee_{\sigma(s)\in F} \text{Contents}(root, i, \sigma)$$

i. e. if the root of the tree stores an induced map that maps the initial state to a final state. This is a finite disjunction over a constant number $= |Q|^{|Q|}$ possibilities.

Since Q is constant, it is trivial to encode a mapping $\sigma : Q \rightarrow Q$ by Boolean variables $\bar{b}_1, \ldots, \bar{b}_s$ where $s = 2q$ and for all i, $|\bar{b}_i| = q$ and $\sigma(\bar{b}_{2i-1}) = \bar{b}_{2i}$. Let $q' = 2q^2$. We can then define a formula Value$(x_1, \ldots, x_{q'}, k, \sigma)$ meaning that the k-th bits of first-order variables x_i, for $1 \leq i \leq q'$, encode the mapping, σ, for $0 \leq k \leq \log n - 1$. Using Value, TreePath, Contents and Child, we can then define the updated relation, Content$'$.

Change(w_m) to w'_m. Let σ'_m denote the new induced map on the state set at the leaf-node, m:

$$\begin{aligned}\text{Content}'(v, i, \sigma) \equiv\ & (\neg\text{TreePath}(m, 0, v, i) \wedge \text{Content}(v, i, \sigma)) \\ & \vee (\exists x_1, \ldots, x_{q'})\text{R}(x_1, \ldots, x_{q'}, m, \sigma'_m) \wedge \ \text{Value}(x_1, \ldots, x_{q'}, i, \sigma)\end{aligned}$$

where R$(x_1, \ldots, x_{q'}, m, \sigma')$ is a first-order formula that checks in parallel that for every level, l, of the tree, σ_l (such that Value$(x_1, \ldots, x_{q'}, l, \sigma_l)$) is encoded in $x_1, \ldots, x_{q'}$ and that $\sigma_l \cdot \sigma_{l1} = \sigma_{l+1}$ (or, $\sigma_{l1} \cdot \sigma_l = \sigma_{l+1}$) where σ_{l1} is the left (resp., right) sibling of the vertex at level $l+1$. We leave the remaining details as an exercise for the interested reader.

2. We show the D^2 case. The theorem follows by an easy adaptation. D^2 can be parsed using the level trick: assign a level to each parenthesis starting at one and ignoring the differences in parenthesis type. The *level* of a parenthesis equals the number of left parentheses to its left (including it) minus the number of right parentheses strictly to its left. A right parenthesis matches a left one if it is the closest parenthesis to the right on the same level. A string is in D^2 iff all parentheses have a positive level and each left parenthesis has a matching right parenthesis of the same type.

In [BC89], the authors showed that $D^2 \in TC^0$. We basically note that the relations they used can be updated in FO. Let

$$\text{LEFT(RIGHT)}(i) \equiv \text{ parenthesis at position } i \text{ is a left (right)type}$$

$$\text{LEVEL}(i,l) \equiv \text{ parenthesis at position } i \text{ is at level } l$$

$$\text{MATCH}(i,j) \equiv \text{ parenthesis at position } i \text{ matches parenthesis at } j$$

$$\text{D2} \equiv \text{input is in } D^2.$$

LEFT, RIGHT, MATCH and D2 can be described by FO formulae. For example, $D^2 \equiv$

$$(\forall i)(\exists l)(l > 0 \wedge \text{ LEVEL}(i,l)) \wedge (\forall j)(\exists k)\text{LEFT}(j) \wedge \text{ RIGHT}(k) \wedge \text{ MATCH}(j,k)$$

Let $\{\#x : f(x)\}$ denote the number of x's such that $f(x)$ is true. LEVEL is then expressed as:

$$\begin{aligned}\text{LEVEL}(i,l) \equiv{} & (\exists y)(y = \#x : x \leq i \wedge \text{ OPEN}(x)) \\ & \wedge(\exists z)(z = \#x : x < i \wedge \text{ CLOSE}(x)) \wedge y \geq z \wedge l = y - z\end{aligned}$$

Clearly LEVEL is in Dyn-FO, i.e., LEVEL can be updated in FO: under an $\text{insert}(x,'[_1')$ operation, for example, we have for $i < x$, $\text{LEVEL}'(i,l)$ same as $\text{LEVEL}(i,l)$, for $i > x$, $\text{LEVEL}'(i,l)$ iff $\text{LEVEL}(i,l-1)$, and for $i = x$, $\text{LEVEL}'(x,l)$ iff for some m, $\text{LEVEL}(x-1,m)$ and $((l = m-1$ and $\text{OPEN}(x-1))$ or $(\text{CLOSE}(x-1)$ and $l = m))$. The deletes and insert of a closed parenthesis can be handled similarly. ∎

Theorem 4.3 *The following complexity classes are contained in* Dyn-FO$^+$*:*

1. *Read-k times only L, for any constant k,*
2. *Read-k times NL, for any constant k.*

Proof: We shall use the following lemma. Let LAYERED-GAP denote the reachability problem in a *layered* (with edges between vertices in adjacent layers) acyclic directed graph.

Lemma 4.4 *LAYERED-GAP is in Dyn-FO.*

Proof: Assume w.l.o.g. that the number of vertices in each level is the same. Let l, n denote the number of levels and number of vertices in each level. We denote the graph as G = (V,E) where V is given by a pair (x, i), where $x \in [n]$ and $i \in [l]$. We maintain the relations E and P as before.
Consider an insert$(E, (a, k), (b, m))$ operation: if $m = k + 1$, we disregard the update, otherwise we insert the new edge into E′ and update P as follows:

$$\begin{aligned} \mathrm{P}'((x,i),(y,j)) \equiv\ & [(\mathrm{P}((x,i),(a,k)) \vee (x = a \wedge i = k)) \wedge \\ & (\mathrm{P}((b,m),(y,j)) \vee (b = y \wedge m = j))] \\ & \vee \mathrm{P}((x,i),(y,j)) \end{aligned}$$

Consider a delete$(E, (a, k), (b, m))$ operation: if $m = k + 1$, we disregard the update, otherwise we update E easily and P as follows:

$$\begin{aligned} \mathrm{P}'((x,i),(y,j)) \equiv\ & (\mathrm{P}((x,i),(y,j)) \wedge (j \le k \vee i \ge m)) \vee \\ & (\exists u, v)(\mathrm{E}'((u,k),(v,k+1)) \wedge \\ & \mathrm{P}((x,i),(u,k)) \wedge \mathrm{P}((v,k+1),(y,j))) \end{aligned}$$

∎

Actually the proof shows that polynomially many edge insertions and deletions can be handled simultaneously in parallel as long as they occur between two adjacent layers.

1. Given any logspace Turing machine, M, in polynomial time (actually FO) we (see, for example [I87], [P]) build a *layered* acyclic directed graph, G, (with out-degree at most 1) corresponding to M's computation on blank input, such that there is an edge from vertex x to y iff a valid move takes M from the configuration corresponding to x to that corresponding to y. We precompute G and the reachability predicates on G in polynomial time. Then, for any single bit change in the input, possibly a large number of edges are deleted and/or inserted in G. By the lemma above, for edges occurring between adjacent levels, we can easily maintain the reachability predicates on G in Dyn-FO. We complete the proof by noting that since any input bit is read only a constant number of times along any single path in the graph, only constantly many levels are altered in G.

2. The proof for read-k times non-deterministic logspace is similar. The computation graph in this case has outdegree greater than 1, but the lemma still holds and the proof goes through as for the deterministic case. ∎

References

[AHU] Aho, A., Hopcroft, J. and Ullman, J. *The Design and Analysis of Algorithms*, 1979, McGraw Hill.

[AU79] Aho, A. and Ullman, J. "Universality of data retrieval languages," *Proceedings of the 6th ACM Symposium on POPL*, 1979, 110-117.

[A*90] Alpern, B., Hoover, R., Rosen, B. K., Sweeney, P. F. and Zadeck, F.K. "Incremental Evaluation of computational circuits," *Proceedings of the 1st ACM-SIAM Symposium on Discrete Algorithms*, 1990, 32-42.

[AV89] Abiteboul, S. and Vianu, V. "Fixpoint extensions of first-order logic and datalog-like languages," *Proceedings of the Symposium on Logic in Computer Science*, 1989, 2-11.

[ACF90] Alpern, B., Carter, L. and Feig, E. "Uniform Memory Hierarchies," *Proceedings of the 31st IEEE Symposium on Foundations of Computer Science*, 1990, 600-608.

[AV91] Abiteboul, S. and Vianu, V. "Generic Computation and its Complexity," *Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science*, 1991, 209-219.

[AV88] Agarwal, A. and Vitter, J.S. "The Input/Output Complexity of Sorting and Related Problems," *Communications of the ACM* **31** No. 9, 1988.

[A84] Ajtai, M. "A lower bound for finding predecessors in Yao's probe model," *Combinatorica 8*, 3, 1988, 235-247.

[AFK83] Ajtai, M., Fredman, M. and Komlos, J. "Hash functions for Priority Queues," *Proceedings of the 24th IEEE Conference on Foundations of Computer Science*, 1983, 299-303.

[AC89] Arafati, F. and Cosmadakis, S. "Expressiveness of restricted recursive queries," *Proceedings of the 21st ACM Symposium on Theory of Computing*, 1989, 113-126.

[AL*92] Arora, S., Lund, C., Motwani, R., Szegedy, M. and Sudan, M. "Proof Verification and Hardness of Approximation Problems," *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science*, 1992, 14-23.

[AI*90] Ausiello, G., Italiano, G. F., Marchetti-Spaccamela, A. and Nanni, U. "Incremental Algorithms for Minimal Length Paths," *Proceedings of ACM-SIAM Symposium on Discrete Algorithms*, 1990, 12-21.

[B87] Barrington, D. M. "Bounded-width Polynomial-size Branching Programs Recognize Exactly Those Languages in NC^1," *Journal of Computer System and Sciences*, **38**, 1989, 150-164.

[BC89] Barrington, D. M. and Corbett, J. C. "On the Relative Complexity of some Languages in NC^1," *Technical Report* $89 - 22$, Department of Computer Science, University of Massachusetts, Amherst.

[DT89] Di-Battista, G. and Tamassia, R. "Incremental Planarity Testing," *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science*, 1989, 436-441.

[B84] Beame, P. "A General Sequential Time-space Tradeoff for Finding Unique Elements," *Proceedings of the 21st ACM Symposium on Theory of Computation*, 1989, 197-203.

[BC92] Bellantoni, S. and Cook, S. "A New Recursion-Theoretic Characterization of Polynomial Time", *Proceedings of the 24th Annual ACM Symposium on the Theory of Computing*, 1992, 283-293.

[B92] Bellantoni, S. "Predicative Recursion and Computational Complexity," *Technical Report* 264/92, University of Toronto, September 1992.

[BK90] Broder, A. and Karlin, A. "Multilevel Adaptive Hashing," *Proceedings of the 1st ACM-SIAM Symposium on Discrete Algorithms*, 1990, 43-53.

[BRS91] Borodin, A., Razborov, R. and Smolensky, S. "On Lower Bounds for Read-k Times Branching Programs," Preprint, 1991.

[BM] Boyer, R. S. and Moore, J. S. *A Computational Logic*, 1979, Academic Press.

[CFI89] Cai, J. Y., Fürer, M., Immerman, N. "An Optimal Lower Bound on the Number of Variables for Graph Identification," *Combinatorica* **12**:4, 1992, 389-410.

[CH1] Chandra, A. and Harel, D. "Structure and Complexity of Relational Queries," *Journal of Computer and System Sciences* **25**, 1982, 99-128.

[CH2] Chandra, A. and Harel, D. "Horn Clause Queries and Generalizations," *Journal of Logic Programming* **1**, 1985, 1-15.

[CH82b] Chandra, A. and Harel, D. "Horn Clauses and Fixpoint Query Hierarchy," *Proceedings of the 14th ACM Symposium on Theory of Computing*, 1982, 158-163.

[Ch81] Chandra, A. "Programming Primitives for Database Languages," *Proceedings of the ACM Symposium on POPL*, 1981, 50-62.

[CSV] Chandra, A., Stockmeyer ,L. J. and Vishkin, U. "Constant Depth Reducibility," *SIAM Journal of Computing* **13**, No. 2, 1984, 423-439.

[CJ90] Cheng, S.W. and Janardan, R. "Efficient Maintenance of the Union of Intervals on a Line, with Applications," *Proceedings of the 1st ACM-SIAM Symposium on Discrete Algorithms*, 1990, 74-83.

[Co64] Cobham, A. "The Intrinsic Computational Difficulty of Functions," *Proceedings of the 1964 Congress for Logic, Philosophy and Methodology of Science*, North Holland, 24-30.

[C70] Codd, E. "A Relational Model for Large Shared Databanks," *Communications of the ACM*, 13(6), 1977, 377-387.

[C72a] Codd, E. "Further Normalization of the Database Relational Model," In R. Rustin, ed., *Database systems*, 1972, Prentice Hall, 33-64.

[C72b] Codd, E. "Relational Completeness of Database Sublanguages," In R. Rustin, ed., *Database systems*, 1972, Prentice Hall, 65-98.

[CT91] Cohen, R. and Tamassia, R. "Dynamic Expression Trees and their Applications," *Proceedings of the 2nd Annual ACM-SIAM Symposium on Discrete Algorithms*, 1991.

[CK85] Cosmadakis, S. and Kanellakis, P. "Parallel Evaluation of Recursive Rule Queries," *Proceedings of the 5th ACM SIGACT-SIGART-SIGMOD Symposium on PODS*, 1986, 280-290.

[CLR] Cormen, T., Leicerson, C. E. and Rivest, R. A. *Introduction to Algorithms*, 1990, McGraw Hill.

[CP84] Cosmadakis, S. S. and Papadimitriou, C.H. "Updates of Relational Views," *Journal of the ACM*, 1984.

[D84] Dahlhaus, E. "Reduction to NP-Complete Problems by Interpretations," *Logic and Machines: Decision Problems and Complexity,* Börger, Rödding, and Hasenjaeger eds., Lecture Notes In Computer Science 171, Springer-Verlag, 1984, 357-365.

[D93] Dawar, A. "Feasible Computation through Model Theory", *Phd thesis*, University of Pennsylvania, Philadelphia, 1993.

[D89] Dietz, P.F. "Fully Persistent Arrays," *Proceedings of the Conference on Foundations of Software Technology and Theoretical Computer Science*, 1989, Springer Verlag, 67-73.

[DS88] Dietz, P. F. and Sleator, D. D. "Two Algorithms for Maintaining Order in a List", *Technical Report CMU-CS-88-113*, 1988, Carnegie Mellon University.

[D*88] Dietzfelbinger, M., Karlin, A., Mehlorn, K., Meyer auf der Heide, F., Rohnert, H. and Tarjan, R. E. "Dynamic Perfect Hashing: Upper and Lower Bounds," *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science*, 1988, 524-531.

[DPZ91] Djidev, H. N., Pantziou, G. E. and Zaroliagis, C. D. "Computing Shortest Paths and Distances in Planar Graphs," *Proceedings of the 18th International Colloqium on Automata, Languages and Programming*, 1991, 327-338.

[DS93] Dong, G., Su, J. W. "Incremental and Decremental Evaluation of Transitive Closure by First-Order Queries," University of California, Santa Barbara, Preprint, 1993.

[DST93] Dong, G., Su, J. and Topor, R. "First-Order Incremental Evaluation of Datalog Queries", *Proceedings of the 1992 International Conference on Database Theory*, LNCS 646, Springer Verlag.

[D*86] Driscoll, J. D., Sarnak, N., Sleator, D. D. and Tarjan R. E. "Making data structures persistent," *Journal of Computer and System Sciences* **38**, 1989, 125-133.

[DR85] Dymond, P. W. and Ruzzo, W. L. "Parallel RAMs with Owned Global Memory and Deterministic Context-free Language Recognition," *Proceedings of the Conference on ICALP*, 1986.

[EF75] Elias, P. and Flower, R. A. "The Complexity of Some Simple Retrieval Problems," *Journal of the Association for Computing Machinery (ACM)*, **22** No. 3, 1975, 367-379.

[E*90] Eppstein, D., Italiano, G. F., Tamassia, R., Tarjan, R. E., Westbrook, J. and Yung, M. "Maintenance of a Minimum Spanning Forest in a Dynamic Planar Graph," *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, 1990.

[E*92a] Eppstein, D., Galil, Z., Italiano, G. F. and Nissenzweig, A. "Sparsification - A Technique for Speeding up Dynamic Graph Algorithms," *Proceedings of 33rd IEEE Symposium on Foundations of Computer Science*, 1992, 60-69.

[E*92b] Eppstein, D., Galil, Z., Italiano, G. F. and Nissenzweig, A. "Sparsification and Planarity Testing," *Proceedings of 24th Annual ACM Symposium on Theory of Computing*, 1993.

[Fa74] Fagin, R. "Generalized First-Order Spectra and Polynomial-Time Recognizable Sets," in *Complexity of Computation,* (ed. R. Karp), *SIAM-AMS Proc. 7,* 1974, (27-41).

[F92] Fegaras, L. "A Tranformational Approach to Database System Implementation," Technical Report 92-68, Computer Science Department, University of Massachusetts, Amherst, 1992.

[FS1] Fegaras, L. and Sheard, T. "A Fold for All Seasons," Department of Computer Science and Engineering, Oregon Graduate Institute of Science and Technology, Preprint, 1993.

[FS2] Fegaras, L. and Sheard, T. "Automated Term Rewriting in an Algebra Derived from the Structure of Types," Department of Computer Science and Engineering, Oregon Graduate Institute of Science and Technology, Preprint, 1993.

[FS2] Fegaras, L., Maier, D. and Sheard, T. "Specifying Rule-based Query Optimizers in a Reflective Framework," Department of Computer Science and Engineering, Oregon Graduate Institute of Science and Technology, Preprint, 1993.

[FSS92] Fegaras, L., Sheard, T. and Stemple, D. "Uniform Traversal Combinators: Definition, Use and Properties", *Proceedings of* 11*th Conference on Automated Deduction (CADE-11)*, Saratoga Springs, New York, 1992.

[F83a] Frederickson, G. F. "Data Structures for On-line Updating of Minimum Spanning Trees," *SIAM Journal on Computing*(14), 1985, 781-798.

[F83b] Frederickson, G. F. "Ambivalent Data Structures for Dynamic 2-edge Connectivity and k Smallest Spanning Trees," *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science*, 1991.

[F80] Fredman, M. L. "The Inherent Complexity of Dynamic Data Structures which Accomodate Range Queries," *Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science*, 1980, 191-200.

[FS89] Fredman, M. and Saks, M. "The Cell Probe Complexity of Dynamic Data Structures," *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, 1989, 345-354.

[FKS84] Fredman, M., Komlos, J. and Szemeredi, E. "Storing a Sparse Table with O(1) Worst Case Access Time," *Journal of the ACM* **31**(3), 1984, 538-544.

[GG81] Gabor, O. and Galil, Z. "Explicit Constructions of Linear Size Superconcentrators," *Journal of Computer and System Sciences* **22**, 1981, 407-420.

[GI91] Galil, Z. and Italiano, G F. "Reducing Edge Connectivity to Vertex Connectivity", *SIGACT News* **22**(1), 1991, 57-61.

[GIS92] Galil, Z. and Italiano, G. F. "Fully Dynamic Planarity Testing," *Proceedings of the 24th Annual ACM Symposium on the Theory of Computing*, 1992, 495-506.

[GJ] Garey, M. R. and Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP Completeness*, 1979, Freeman and Company.

[GW94a] Goemans, M. X. and Williamson, D. P. "A New $\frac{3}{4}$-Approximation Algorithm for MAX SAT," *SIAM Journal on Discrete Mathematics*, 1994.

[GW94b] Goemans, M. X. and Williamson, D. P. ".878-Approximation Algorithms for MAX CUT and MAX 2SAT," Manuscript, 1994.

[G91] Grädel, E. "Capturing Complexity Classes by Fragments of Second Order Logic," *Proceedings of the 6th IEEE Symposium on Structure in Complexity Theory*, 1991, 341-352.

[GMS93] Gupta, A., Mumick, I. S. and Subrahmanian, V.S. "Maintaining Views Incrementally," *Proceedings of the ACM SIGMOD*, 1993, 157-166.

[GKM92] Gupta, A., Katiyar, D. and Mumick, I. S. "Counting Solutions to the View Maintenance Problem," In K. Ramamohanrao, J. Harland, G. Dong, editors. *Proceedings of the JICSLP Workshop on Deductive Databases*, Washington DC, November 1992.

[Gul] Gurevich, Y. "Algebras of Feasible Functions," *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science*, 1983, 210-214.

[HRS] Hershberger, J., Rauch, M. and Suri, S. "Fully Dynamic 2-connectivity in Planar Graphs", Preprint, 1992.

[HK81] Hong, J. W. and Kung, H. T. "I/O Complexity: the Red-blue Pebble Game," *Proceedings of the 13th ACM Symposium on Theory of Computing*, 1981, 326-333.

[HS89b] Hull, R. and Su, J. W. "On Bulk Data Type Constructors and Manipulation Primitives - a Framework for Analyzing Expressive Power and Complexity," *Proceedings of 2nd International Workshop on Database Programming Languages*, 1989, 396-410.

[I81] Immerman, N. "Number of Quantifiers is Better than Number of Tape Cells," *Journal of Computer and System Sciences* **22**, No. 3, 1981, 65-72.

[I82] Immerman, N. "Upper and Lower Bounds for First Order Expressibility," *Journal of Computer and System Sciences* **25**, No. 1, 1982, 76-98.

[I86] Immerman, N. "Relational Queries Computable in Polynomial Time," *Information and Control,* **68**, 1986, 86-104.

[I87] Immerman, N. "Languages that Capture Complexity Classes," *SIAM J. Comput.* **16**, No. 4, 1987, 760-778.

[I5] Immerman, N. "Expressibility as a Complexity Measure: Results and Directions," *Second Structure in Complexity Theory Conf.*, 1987, 194-202.

[I89a] Immerman, N. "Descriptive and Computational Complexity," in *Computational Complexity Theory,* ed. J. Hartmanis, *Proc. Symp. in Applied Math.,* 38, American Mathematical Society, 1989, 75-91.

[I89b] Immerman, N. "Expressibility and Parallel Complexity," *SIAM Journal of Computing* **18**, 1989, 625-638.

[I91] Immerman, N. "DSPACE[n^k] = VAR[k+1]," *Proceedings of the 6th IEEE Symposium on Structure in Complexity Theory*, 1991, 334-340.

[IL89] Immerman, N. and Landau S. "The Complexity of Iterated Multiplication," *Proceedings of the Symposium on Structure in Complexity Theory*, 1989.

[IPS] Immerman, N., Patnaik, S. and Stemple D. "The Expressiveness of a Family of Finite Set Languages," *Proceedings of the 10th ACM SIGART-SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS)*, 1991, 37-52. To appear in *Theoretical Computer Science.*

[I85] Iananadis, Y. "A Time Bound on the Materialization of Some Recursively Defined Views," *Proceedings of International Conference on Very Large Data Bases*, 1985.

[J92] Jakobsson, H. "On Materializing Views and On-line Queries," *Proceedings of International Conference on Database Theory*, Berlin, 1992, 407-420.

[Jo74] Johnson, D. S. "Approximation Algorithms for Combinatorial Problems," *Journal of Computer and System Sciences* **9**, 1974, 256-278.

[JL77] Jones, N. D. and Laaser, W. T. "Complete Problems for Deterministic Polynomial Time," *Theoretical Computer Science* **3**, 1977, 105-117.

[K93] Kahale, N. "Dynamic Expanders," Manuscript, 1993.

[K72] Karp, R. M. "Reducibility Among Combinatorial Problems," *Complexity of Computer Computations*, Editors: R. E. Miller and J. W. Thatcher, Plenum Press, 1972, 85-95.

[KS93] Klein, P. N. and Sairam, S. "A Fully Dynamic Approximation Scheme for All-pairs Shortest Paths in Planar Graphs,"*Proceedings of the Workshop on WADS*, 1993.

[KT90] Kolaitis, P. G. and Thakur, M. N. "Logical Definability of NP optimization problems," *Technical Report UCSC-CRL-90-48*, Dept. of Computer and Information Sciences, University of California at Santa Cruz, February, 1990.

[KT90] Kolaitis, P. G. and Thakur, M. N. "Logical Definability of NP Optimization Problems," Technical Report UCSC_CRL-93-10, Computer and Information Sciences, University of California, Santa Cruz, 1993. To appear in *Information and Computation.*

[Kr88] Krentel, M. W. "The Complexity of Optimization Problems," *Journal of Computer and System Sciences* **36**, 1988, 490-509.

[L92] Leivant, D. "A Foundational Delineation of Poly-time", *Proceedings of Sixth Annual IEEE Symposium on Logic in Computer Science*, 1991.

[L87] Lindell, S. "The Logical Complexity of Queries on Unordered Graphs," PhD Thesis, University of California, Los Angeles, 1987.

[LST87] LLoyd, J. W., Sonenberg, E. A. and Topor, R. W. "Integrity Constraint Checking in Stratified Databases," *Journal of Logic Programming*, 4(4):334-343, 1987.

[LPS86] Lubotzky, A., Phillips, R. and Sarnak, P. "Explicit Expanders and the Ramanujan Conjectures," *Proceedings of the 18th ACM Symposium on Theory of Computation*, 1986, 240-246.

[MI94] Medina, J. A. and Immerman, N. "On NP-completeness under First Order Projections," *Proceedings of 9th IEEE Symposium on Logic in Computer Science*, 1994.

[MP93] Medina, J. A. and Patnaik, S. "Expressiveness of Logics Extended with Reflection," *Technical Report*, Computer Science Department, University of Massachusetts, Amherst.

[MNR89] Mehlhorn, K., Naher, S. and Rauch, M. "On the Complexity of a Game Related to the Dictionary Problem," *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science*, 1989, 546-548.

[M88] Margulis, G. A. "Explicit Group-theoretical Constructions of Combinatorial Schemes and their Applications to the Design of Expanders and Concentrators," *Problemy Peredači Informacii*, 24(1):51–60, 1988.

[M1] Miltersen, P. B. "Online Complexity of Functions", Preprint, 1992.

[M2] Miltersen, P. B. "The Bit Probe Complexity Measure Revisited," *Proceedings of the 10th Symposium on Theoretical Aspects of Computer Science* (1993).

[M3] Miltersen, P. B. "Dynamic Word Problems," Technical Report DAIMI PB - 438, Computer Science Department, Aarhus University, 1993. Appeared in *Proceedings of the 34th IEEE Symposium on Foundations of Computer Science*, 1993.

[MSTV93] Miltersen, P. B, Sairam, S., Tamassia, R. and Vitter, J. S. "A Complexity Theoretic Approach to Incremental Computation," *Proceedings of 10th Symposium on Theoretical Aspects of Computer Science*, 1993.

[N82] Nicolas, J-M. "Logic for Improving Integrity Checking in Relational Databases," *Acta Informatica*, **18**(3):227-253, 1982.

[NLV89] Nodine, M. H., Lopresti, D. P. and Vitter, J. S. "I/O overhead and Parallel VLSI Architectures for Lattice Computations," *Technical Report No. CS-89-18*, Dept. of Computer Science, Brown University, March 1989.

[NV89] Nodine, M. H. and Vitter, J. S. "Greed sort : An Optimal External Sorting Algorithm for Multiple Disks," *Technical Report No. CS-90-04*, Dept. of Computer Science, Brown University.

[OBB] Ohori, A., Buneman, P. and Breazu-Tannen, V. "Database Programming In Machiavelli,"*Proceedings of the ACM SIGMOD*, 1989, 46-57.

[O83] Overmars, M. H. "The Design of Dynamic Data Structures", *Lecture Notes in Computer Science 156*, Springer Verlag, 1983.

[PR90] Panconesi, A. and Ranjan, D. "Quantifiers and Approximation," *Proceedings of the 22nd ACM Symposium on the Theory of Computing*, 1990, 446-456.

[P] Papadimitriou, C. H. *Computational Complexity*, Academic Press, 1994.

[PS84] Papadimitriou, C. H. and Sipser, M. "Communication Complexity," *Journal of Computer and System Sciences* **28**, 1984, 260-269.

[PY88] Papadimitriou, C. H. and Yannakakis, M. "Optimization, Approximation and Complexity Classes," *Proceedings of the 20th ACM Symposium on the Theory of Computing*, 1988, 229-234.

[P93] Patnaik, S. "The Expressiveness of Uniform Traversal Combinators," *Technical Report*, 1994, Computer Science Department, University of Massachusetts, Amherst.

[P94] Patnaik, S. "The Dynamic Complexity of Approximation," *Technical Report*, 1994, Computer Science Department, University of Massachusetts, Amherst.

[PI94] Patnaik, S. and Immerman, N. "DYN-FO: A Parallel Dynamic Complexity Class", *Proceedings of the 13th ACM SIGACT-SIGART-SIGMOD Symposium on PODS*, 1994.

[PF79] Pippenger, N. and Fischer, M. J. "Relations among complexity measures", *Journal of the ACM*, **26**, No. 2, 1979.

[PT88] Preparata, F. P. and Tamassia, R. "Fully Dynamic Techniques for Point location and Transitive Closure in Planar Structures," *Proceedings of 29th IEEE Symposium on Foundations of Computer Science*, 1988, 558-567.

[R92] Rauch, M. "Fully Dynamic Biconnectivity in Graphs," *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science*, 1992, 50-59.

[RR91] Ramalingam, G. and Reps, T. "On the Computational Complexity of Incremental Algorithms," *Technical Report TR-1033*, Computer Sciences Department, University of Wisconsin, Madison, 1991.

[R] Rose, H. E. *Subrecursion: Functions and Hierarchies*, Oxford Logic Guides **9**, 1984, Clarendon Press, Oxford.

[SS93] Sairam, S. "A Fully Dynamic Data Structure for Reachability in Planar Digraphs," *Proceedings of the European Symposium on Algorithms*, 1993.

[SS89] Sheard, T. and Stemple, D. "Automatic Verification of Database Transaction Safety," *ACM Transactions on Database Systems* **14**, No. 3, 1989, 322-368.

[ST85] Sleator, D. D. and Tarjan, R. E. "Amortized Efficiency of List Update and Paging Rules," *Communications of the ACM*, **28** No. 2, 1985, 202-208.

[S*92] Stemple, D., Stanton, R. B., Sheard, T., Philbrow, P., Morrison, R., Kirby, G. N. C., Fegaras, L., Cooper, R. L., Connor, R. C. H., Atkinson, M. P., and Alagic, S. "Type-safe Linguistic Reflection", *Research report CS/92/6*, Department of Mathematical and Computational Sciences, University of St. Andrews, North Haugh, Scotland, 1992.

[SM*93] Stemple, D., Morrison, R., Kirby, G. N. C. and Connor, R. C. H. "Integration, Reflection, Strong Typing and Static Checking", *Research report CS/93/6*, Department of Mathematical and Computational Sciences, University of St. Andrews, North Haugh, Scotland, 1993.

[S91] Stewart, I. A. "Complete Problems Involving Boolean Labelled Structures and Projection Translations," *Journal of Logic Computation*, **1** No. 6, 1991, 861-882.

[S74] Stockmeyer, L. J. "The Complexity of Decision Problems in Automata Theory and Logic," MAC TR-133, Project MAC, 1974, MIT, Cambrige, Mass.

[S91] Sundar, R. "A Lower Bound for the Dictionary Problem under a Hashing Model," *Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science*, 1991.

[TY79] Tarjan, R. E. and Yao, A. "Storing a Sparse Table," *Communications of the ACM* **22**(11), 1979, 606-611.

[U] Ullman, J. D. *Principles of Database and Knowledge-Base Systems, Vol. II*, 1989, Computer Science Press, Rockville, MD.

[UY90] Ullman, J. D. and Yannakakis, M. "The Input/Output Complexity of Transitive Closure," *Proceedings of the 9th ACM SIGACT-SIGART-SIGMOD Symposium on PODS*, 1990.

[Var] Vardi, M. Y. "Complexity of Relational Query Languages," *14th Symposium on Theory of Computation*, 1982, 137-146.

[VVV93] Van den Bussche, J., Van Gucht, D., and Vossen, G. "Reflective Programming in the Relational Algebra," *Arbeitsgruppe Informatik, Universitat Giessen, Bericht Nr. 9210*, 1992. Appeared in the *Proceedings of the 12th ACM SIGACT-SIGMOD-SIGART Symposium on PODS*, 1993.

[VKZ77] Van Emde Boas, P., Kaas, R. and Zijlstra, E. "Design and Implementation of an Efficient Priority Queue," *Mathematical Systems Theory* **10**, 1977, 99-127.

[W84] Willard, D. E. "New Trie Data Structures which Support Very Fast Search Operations," *Journal of Computer and System Sciences* **28**, 1984, 379-394.

[Y92] Yannakakis, M. "On the Approximation of Maximum Satisfiability," *Proceedings of the 3rd ACM-SIAM Symposium on Discrete Algorithms*, 1992, 1-9.

[Y81] Yao, A. C. "Should Tables Be Sorted," *Journal of the ACM* **28**(3), 1981, 615-628.

[Y82] Yao, A. C. "Space-time Tradeoff for Answering Range Queries," *Proceedings of the 14th ACM Symposium on Theory of Computing*, 1982, 128-136.