June, 1985

# PROVING RELATIVE LOWER BOUNDS FOR INCREMENTAL ALGORITHMS

by

A.M. Berman, M.C. Paull and B.G. Ryder

DCS-TR-154

Department of Computer Science
Rutgers University
New Brunswick, NJ 08903

## ABSTRACT

A general, powerful method that permits simple proofs of **relative lower bounds** for incremental update algorithms is presented. This method is applied to derive a hierarchy of relative incremental complexity, which classifies functions by relative lower bounds. We demonstrate our technique by bounding a number of incremental algorithms drawn from various domains. The method described expands upon work by Paull, Berman, and Cheng [7] and generalizes a result of Even and Gazit [2]. Our results have interesting implications with respect to the optimality of an incremental algorithm previously developed by Ryder in [9, 10]. We also show that for certain graphs, Frederickson's update algorithm for minimum spanning tree is nearly optimal [3]. Perhaps most importantly, the proof method and hierarchy suggest which types of problems are likely to yield good incremental algorithms (i.e., of lower complexity) and which cannot be improved by an incremental approach.

## Table of Contents

1. Introduction 1
2. A Model for Proving Relative Lower Bounds 2
   2.1. Definitions 2
   2.2. The Proof Method 2
3. The IRLB Hierarchy 6
   3.1. Examples of Class 1 Functions 7
   3.2. Fast Initializations and Preprocessing: Some Observations 7
4. Examples of Incremental Relative Lower Bounds in Graph Problems 8
   4.1. A Recapitulation of All-pairs Shortest Paths 8
   4.2. Connected and Biconnected Components 9
   4.3. Transitive Closure 9
5. Solving Systems of Equations 10
   5.1. Extreme Problem I--Unconstrained Equations 11
   5.2. Constrained Problems 13
      5.2.1. Extreme Problem II---Constrained Equations 13
      5.2.2. Extreme Problem III---Further Constrained Equations 15
6. Implications 17
   6.1. A Collection of IRLB's 17
   6.2. Implications for Updating Minimum Spanning Trees and Connected Components 17
   6.3. Implications for Incremental Data Flow Analysis 18
7. Conclusion 19
REFERENCES 20

## 1. Introduction

A general, powerful method that permits simple proofs of **relative lower bounds** for incremental update algorithms is presented. We apply this method to derive a hierarchy of relative incremental complexity, which classifies functions by relative lower bounds. Our bounds are relative in the sense that they are proportional to the complexity of the problem at hand; if no lower bound is known for the initial computation, then the relative lower bound is not a lower bound *per se*. We draw on previous work by Paull, Berman and Cheng, who described the conditions under which it is possible to prove weaker lower bounds for incremental algorithms [7]. Our technique encompasses a result of Even and Gazit, who show for the particular problem of computing all shortest paths in a directed graph that an incremental algorithm can do no better, in the worst case, than recomputing the solution with the new inputs [2].

We demonstrate our technique by bounding a number of incremental algorithms drawn from various domains. Our results have interesting implications with respect to the optimality of an incremental algorithm previously developed by Ryder in [9, 10]. We also show that for certain graphs, Frederickson's update algorithm for minimum spanning tree is nearly optimal [3]. Perhaps most importantly, the proof method and hierarchy suggest which types of problems are likely to yield good (i.e., of lower complexity) incremental algorithms and which cannot be improved by an incremental approach, in the worst case.

This paper is structured as follows:

- Section 2 presents basic definitions and the proof method in its most general form.
- Section 3 describes the relative complexity hierarchy.
- Section 4 demonstrates the application of the method in problems involving undirected and directed graphs.
- Section 5 applies the method for problems formulated as systems of equations.
- Section 6 lists common problems to which the machinery has been applied, and relates the results to certain incremental algorithms in the literature.
- Section 7 presents our conclusions.

## 2. A Model for Proving Relative Lower Bounds

### 2.1. Definitions

In order to present our model with some precision, we need to be careful about our terminology. Thus we provide the following formal definitions of algorithm and incremental algorithm, modifications of those in [7].

> **Definition 2.1:** Let $\alpha: P \rightarrow Q$ be a function with domain $P$ being the set of **problem instances** or **inputs**, and range $Q$ the set of **answers** or **outputs**. Each $p \in P$ and $q \in Q$ is itself a set, with $|p|$, the length of the problem instance, represented by $\ell$.[1]
>
> Let $A$ be an **algorithm** that given a problem instance $p$ as input returns $q$ such that $q=\alpha(p)$; thus $A$ **implements** $\alpha$. The number of steps required by algorithm $A$ to compute $\alpha(p)$, in the worst case, is the **(time) complexity** of $A$, denoted $T_A(\ell)$. Let the **(time) optimal** algorithm for $\alpha$ be represented by $\mathbf{A}$; then the complexity of function $\alpha$, $T_\alpha = T_{\mathbf{A}}$.
>
> Choose $\alpha$, $p$, and $q$. Let $p'$ be another problem instance such that $|p-(p \cap p')|=1$, that is, $p'$ differs from $p$ by exactly one input element. Call this element $p_k'$. If given $p$, $p_k'$, and $q$, algorithm $\Delta A$ returns $q'$ such that $q'=\alpha(p')$, then $\Delta A$ is called an **incremental algorithm** for function $\alpha$.

When applying Definition 2.1 to any particular function, one must be careful to apply a proper mapping between the inputs of the algorithm and the elements in set $p$. In particular, the length of the problem must always be proportional to the cardinality of $p$. For example, in a typical graph problem it is consistent with our model to treat pairs of vertices as the elements in the input set; in this model, considering each vertex as an individual element would not be allowed. Hence, the removal or addition of a vertex in a graph would not be regarded as a single incremental change, but rather as a sequence of incremental changes involving each edge incident on the vertex.

### 2.2. The Proof Method

Our proof method is general, encompassing as it does a variety of lower bounds proofs. We will provide an intuitive description of the method and then lay it out more formally, first discussing the general approach and then showing specific examples. We call a bound produced by this method an **incremental relative lower bound**.

---

[1] In most presentations the problem length is assumed to be $n$, but this causes confusion when discussing graphs, where we use $n$ to represent the number of vertices.

**Definition 2.2:** Given an function $\alpha$ with time complexity $T_\alpha$, any incremental algorithm $\Delta A$ for $\alpha$, and $\rho$ some function of $\ell$, if we can show $T_{\Delta A}(\ell)\cdot\rho(\ell) \geq T_\alpha(\ell)$, then $\alpha$ has a **incremental relative lower bound (IRLB) of $1/\rho(\ell)$.**

In the particular cases where $T_\alpha$ is known, we can use an IRLB to derive immediately a (non-relative) lower bound for the incremental case. More typically, we have a "best known" algorithm, and our lower bound can be stated relative to the complexity of that algorithm. For example, Tarjan and Fredman describe a procedure for computing the Minimum Spanning Tree (MST) on an undirected graph in time $O(m\beta(n,m))$ where $\beta$ grows extremely slowly [5]. In Section 6.2 we show that the IRLB for MST is $1/n$, where *$n$ is the number of vertices*, not the length of the problem. Thus we know that we cannot beat $m\beta(n,m)/n$ for incremental MST, unless we can beat the Tarjan-Fredman bound as well. When $m$ grows proportionally to $n^2$, the IRLB is approximately $1/\sqrt{m}$.

The idea of the proof method is simple. The goal is to show that any incremental algorithm for some function $\alpha$ can be used to build an algorithm that does a complete computation of $\alpha$. We count the number of times the incremental algorithm is used. Assuming that we can disregard certain additional overhead in the constructed algorithm, the number of invocations of the incremental algorithm is the denominator of the IRLB for $\alpha$. A general procedure that illustrates this construction is shown in Figure 2-1.

```
0. input problem p_initial of length ℓ;
1. for i ← 1 to δ(ℓ) do
2.     {add some element to p, or change some element of p};
3. q ← α(p);
4. h ← initial history for p;
      /* h is additional information used by the incremental algorithm*/
5. for i ← 1 to ε(ℓ) do
6.     {p ← p with one element changed/removed;
7.     q ← ΔA(p);}          /* assume h updated as side effect */
/* if at the exit to the loop, α(p) = α(p_initial), then q = α(p_initial) */
```

**Figure 2-1:** Procedure for IRLB proof

We assume in this procedure that the steps that add, change, or remove elements in $p$, steps 2 and 6, are constant-cost operations. The exact bound that can be proved depends on the following factors in the particular instance of the procedure shown in Figure 2-1:

i. the function $\delta(\ell)$ (the number of changes made in the input);

ii. the cost of step 3 (initialization);

iii. the cost of step 4 (preprocessing);

iv. the function $\epsilon(\ell)$ (the number of changes removed);

The condition asserted at the end of the procedure does *not* require that $p = p_{initial}$ but only that $p$ and $p_{initial}$ map to the same element of the range of $\alpha$. Thus in general it is not necessary that $\delta(\ell) = \epsilon(\ell)$. In order to compute an initial answer quickly in step 3, we typically select the transformation in steps 1 and 2 such that we arrive at a problem instance that is trivial to compute. The significance of the complexity of step 4, preprocessing, is considered below in Section 3.2. We will denote the combined complexity of steps 3 and 4 by $init(\ell)$. We also must assume that the complexity of the function, $T_\alpha$, **dominates**, in the sense of Definition 2.3, the cost of steps 1 through 4.

**Definition 2.3:** Given two non-negative functions $f(n)$ and $g(n)$, if

$$g(n)-f(n) = \Theta(g(n))$$

then we say that $g(n)$ **dominates** $f(n)$.

An additional complication arises because the length of $p$, the problem instance, does not necessarily remain fixed in the construction procedure. In Step 2 of the construction we permit the procedure to add elements to $p$. Let $\ell_{\max}$ be the length of $p$ at step 3. We will assume that our complexity measures behave monotonically, and hence $T_{\Delta A}(\ell_{max}) \geq T_{\Delta A}(\ell)$. The proof of Theorem 2.1 will require that

$$T_{\Delta A}(\ell_{max})=\Omega(1/\epsilon(\ell)) \Rightarrow T_{\Delta A}(\ell)=\Omega(1/\epsilon(\ell));$$

Lemma 2.1 gives sufficient conditions for this implication to hold.

**Lemma 2.1:** If $T_{\Delta A}(\ell)$ is a polynomial function $f(\ell)$, and $\ell'=c\ell$ for some constant $c$, then:

$$T_{\Delta A}(\ell')=\Omega(g(\ell)) \Rightarrow T_{\Delta A}(\ell)=\Omega(g(\ell))$$

**Proof:** By the definition of $\Omega(g(\ell))$, there exist constants $\ell_0$ and $k_0$ such that for every $\ell'>\ell_0$, $T_{\Delta A}(\ell') \geq k_0 g(\ell)$. By assumption, $T_{\Delta A}(\ell')$ is a polynomial $f(\ell')$, and $\ell'=c\ell$, so $f(c\ell) \geq k_0 g(\ell)$. Consider $\lim_{\ell \to \infty} \frac{f(c\ell)}{f(\ell)}$; it's not hard to see that this must equal $c^k$, where $k$ is the degree of $f$. Since in the limit, the ratio $\frac{f(c\ell)}{c^k f(\ell)}$ approaches 1, $\frac{f(c\ell)}{2c^k f(\ell)}$ must approach $\frac{1}{2}$ as $\ell$ goes to infinity. Thus for large $\ell$,

$2c^k f(\ell) > f(c\ell)$. Let $k_1 = 2c^k$; then for $\ell > \ell_1$, $k_1 f(\ell) > f(c\ell) \geq k_0 g(\ell)$; thus, $f(\ell) > \frac{k_0}{k_1} g(\ell)$. Since $k_0/k_1$ is constant, we can conclude that $f(\ell) = \Omega(g(\ell))$. ■

In the case of exponential complexity, Lemma 2.1 does not hold, since $\lim_{\ell \to \infty} \frac{2^{c\ell}}{2^\ell} = \infty$ for $c > 1$. However, the implication in the lemma can be shown to hold for other common sub-exponential functions, e.g. $f(\ell) = \ell \log \ell$.

**Theorem 2.1:** Consider an incremental algorithm $\Delta A$ for function $\alpha$ and a procedure of the general form as shown in Figure 2-1 constructed using $\Delta A$. If:

i. $T_\alpha$ dominates $\delta(\ell)$;

ii. $T_\alpha$ dominates $init(\ell)$ (steps 3 and 4);

iii. $T_\alpha$ (and hence $T_{\Delta A}$) is polynomial, and $\ell_{max} = c\ell$;

iv. when the procedure is exited, $\alpha(p) = \alpha(p_{initial})$

then the function $\alpha$ has an IRLB of $1/\epsilon(\ell)$.

**Proof:** Let $|p_i|$ be the length of the problem instance at the $i$th iteration of the loop at step 5. Then the complexity of the procedure described above is approximately $\ell + \delta(\ell) + init(\ell) + \sum_{i=1}^{\epsilon(\ell)} T_{\Delta A}(|p_i|)$. We can bound the last term by $\epsilon(\ell) T_{\Delta A}(\ell_{max})$. If condition iv. is met, then the procedure described computes $\alpha(p)$. So by Definition 2.1,

$$T_\alpha \leq \ell + \delta(\ell) + init(\ell) + \epsilon(\ell) T_{\Delta A}(\ell_{max}).$$

Solving for $T_{\Delta A}(\ell_{max})$,

$$T_{\Delta A}(\ell_{max}) \geq \frac{T_\alpha - \ell - \delta(\ell) - init(\ell)}{\epsilon(\ell)}.$$

Since we assume by conditions i. and ii. that $T_\alpha$ dominates the other terms,

$$T_{\Delta A}(\ell_{max}) = \Omega\left(\frac{T_\alpha}{\epsilon(\ell)}\right).$$

Using condition iii. and Lemma 2.1, we can conclude that

$$T_{\Delta A}(\ell) = \Omega\left(\frac{T_\alpha}{\epsilon(\ell)}\right);$$

thus, $\alpha$ has an IRLB of $1/\epsilon(\ell)$. ■

Condition iii. in Theorem 2.1 is needed only when the changed problem increases in length. It is easy to prove a modified version of the theorem in which the complexity of the function is not necessarily polynomial, as long as Step 2 of the construction procedure

can only *change*, but not *add* elements. Thus certain bounds are still easily provable for functions not known to be polynomial, for example, those that are NP-complete [7]. In practice, however, most of the problems we will be interested in will have polynomial complexity.

## 3. The IRLB Hierarchy

The problems to which we have applied Theorem 2.1 fall into one of four relative complexity classes, which form a hierarchy. In this section, we present our hierarchy and give some examples of the classification of common problems. We also characterize a class of functions which are in **Class 1** of the hierarchy, using a corollary of Theorem 2.1. The IRLB hierarchy is presented in Figure 3-1.

| Class | IRLB | Example |
|---|---|---|
| 1 | $1/\ell$ | Sorting |
| 2 | $\leq 1/\ell, \geq 1/\sqrt{\ell}$ | Minimum Spanning Tree |
| 3 | 1 | All Shortest Paths |

**Figure 3-1:** A Hierarchy of IRLB's

Since Class $i$ represents functions with tighter bounds than those in Class $i-1$, a function $\alpha$ in Class $i$ is also in Class $i-1$, but the converse is not true. Unless a function is known to have an incremental algorithm which meets its IRLB, then its placement in the hierarchy is necessarily tentative, since it is always possible that a more sensitive proof technique may yield a tighter bound.

Consider the following definition which we will use to show a function is in **Class 1**.

**Definition 3.1:** Consider a function $\alpha$, with the property that for any $\ell$, there exists a problem instance $p_0^\ell$ that can be constructed and $\alpha(p_0^\ell)$ determined in time dominated by $T_\alpha$. The functions for which such a procedure exist are said to have **fast initialization values** [7].

Typically, a fast initialization value can be found by inspection. For example, if the problem is to sort $n$ elements then a fast initialization value is the set of $n$ elements $\{0, \ldots ,0\}$ which is already sorted. If the problem is to find a minimal spanning tree, a fast initialization value is to use a complete graph on $n$ nodes with all edges of weight zero; a minimum spanning tree of this graph is $\{(v_1,v_2),(v_2,v_3),\ldots,(v_{n-1},v_n)\}$. A fast initialization can be found for most problems encountered in practice; therefore, the following corollary to Theorem 2.1 is quite general.

**Corollary 3.1:** If a function $\alpha$ has a fast initialization value, then it has an IRLB of $1/\ell$ [7].

**Proof:** We wish to apply Theorem 2.1, so we construct a procedure as shown in Figure 2-1. For steps 1 and 2, we replace the entire input problem with the fast initialization value, at a cost $\ell$; thus initialization is dominated by $T_\alpha$. In the loop starting at step 5 the procedure restores the original input, one element at a time; hence $\epsilon(\ell) = \ell$. The proof follows immediately from Theorem 2.1. ■

### 3.1. Examples of Class 1 Functions

Examples of problems that can be shown to be in **Class 1** are common. For example, suppose our input is a set of integers, and the function returns the set ordered from lowest to highest. We know that sorting has an IRLB of $1/\ell$ since it has a fast initialization value $\{0, \ldots, 0\}$. Since sorting by comparisons has a proven lower bound of $\Omega(\ell \log \ell)$, incremental sorting by comparisons must have a lower bound of $\Omega(\log \ell)$.

Paull, Berman, and Cheng have shown that 3-SAT has an IRLB of $1/\ell$, and hence no NP-complete problem can have a polynomial time incremental algorithm unless P=NP [7].

### 3.2. Fast Initializations and Preprocessing: Some Observations

We now present an example which demonstrates the need to be careful about fast initializations and preprocessing time. Consider the following familiar pattern matching problem:

> **S** is a string of $n$ symbols, called the **subject**, and **P** is a string of $m$ symbols, called the **pattern**. MATCH(P,S) is *True* if one or more occurrences of P are in S, and *False* otherwise. The particular incremental variation we will consider will permit changes in S but not in P.

We might make the following argument: MATCH has a simple fast initialization; simply change the first $m$ symbols of S to P, and answer *True*. This requires $m$ steps. Then S can be restored to its original state via $m$ steps of an incremental algorithm for MATCH. Hence we conclude that MATCH has an IRLB of $1/m$. Assuming that $T_{\text{MATCH}}=\Theta(n)$, this argument suggests that no incremental algorithm for MATCH can be faster than $\Theta(\frac{n}{m})$.

On the other hand, suppose we construct an incremental algorithm which keeps track of the position of each occurrence of P within S, in a table, plus a count of the number of matches. It is easy to show that each time a change occurs in S, the algorithm need only look in the "neighborhood" of the change, to see if the table and count need to be updated. When the count goes to 0, the answer becomes *false*, and conversely when the count increases from 0, the answer becomes *true*. This is an incremental algorithm for the

problem above, requiring $\Theta(m)$ time. Since in a "typical" problem instance, $m<\sqrt{n}$, the incremental algorithm we are describing is faster than our reputed lower bound. What gives?

The flaw in the first argument is this: the initialization described does not build a table of the positions of P in S; it never even considers S at all. By ignoring the preprocessing (step 4 in the procedure) we miss the fact that the total initialization time is not dominated by the complexity of the algorithm. If the initialization did examine S, it would require time $\Theta(n+m)$, and would no longer be a fast initialization. This example illustrates the importance of the preconditions on the proof, and implies a trade-off between initialization and preprocessing on the one hand, and fast incremental updates on the other.

## 4. Examples of Incremental Relative Lower Bounds in Graph Problems

In this section we present some graph problems and argue their classification in the IRLB hierarchy.

### 4.1. A Recapitulation of All-pairs Shortest Paths

Even and Gazit show that no incremental algorithm for the All-pairs Shortest Paths (ASP) problem in a directed graph can be any faster, in the worst case, than recomputing from scratch [2]. In our terms, this places the function ASP in **Class 3**. Here we recapitulate Even and Gazit's argument to show how it fits within our framework.

Recall that the goal is to exhibit an algorithm that implements the function at hand using an incremental update algorithm for the same function. Given an input graph for the shortest path problem, $G=(V,E)$, modify it by adding two vertices, $v^*$ and $v^{**}$. For each vertex $v_i \in V$, add edges $(v_i,v^*)(v^{**},v_i)$ of weight 0; also add edge $(v^*,v^{**})$ of weight 0. This construction is illustrated in Figure 4-1.

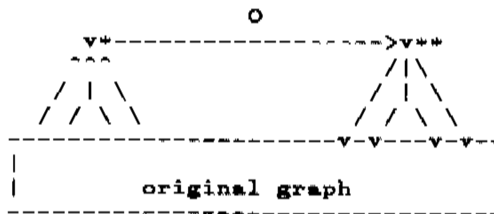


**Figure 4-1:** ASP Construction

This construction corresponds to Steps 1 and 2 in the procedure shown in Figure 2-1, and

can be done in $\Theta(n)$ steps, where $n$ is the number of vertices in $G$; since $n \leq \ell$, Steps 1 and 2 are $O(\ell)$. Step 3, the initialization. is immediate, since one shortest path between any two vertices $a$ and $b$ in the constructed graph is $av^{*}v^{**}b$ with length 0. Step 4 can be performed in parallel with 0, 1, and 2 in Even and Gazit's algorithm. To get back to a graph which has the same solution as the original input merely requires the removal of a single edge, $(v^{*},v^{**})$. The loop function in Step 5 $\epsilon(\ell)$ is 1. Thus we can conclude immediately that ASP has an IRLB of 1, and is in **Class 3**.

## 4.2. Connected and Biconnected Components

The connected components function takes as input an undirected graph $G=(V,E)$ and outputs one or more lists of vertices such that if $v_i$ and $v_j$ are in the same list, then there exists a path between them. We construct an initial trivial solution by forcing all vertices to be in a single component as follows: add an artificial vertex $v^{*}$ and for each $v_i \in V$, add an edge $(v_i,v^{*})$. In terms of Figure 2-1, $\delta(\ell) = n$ where $n = |V|$. Since in any representation of $G$, $\ell \geq n$, Steps 1 and 2 meet the conditions for the theorem. Step 3 is immediate: all vertices are in a single component. Now, to return the problem to its original solution, we must remove each of the edges added in Step 2, and run the incremental algorithm each time. Thus $\epsilon(\ell)$=n is bounded above by $\ell$ when the graph is sparse, that is, $|E| = O(n)$, and below by $\sqrt{\ell}$ when $|E|=\Omega(n^2)$. Connected components is in **Class 2**.

For a undirected graph, two vertices are in the same biconnected component if there exist two paths between the vertices with no common intermediate vertices. The proof for biconnected components uses a construction similar to that in Figure 4-1. We add two vertices $v^{*}$ and $v^{**}$ and connect each old vertex in the graph to each of these new vertices. Now each pair of old vertices $v_i$ and $v_j$ have two distinct paths between them so the entire graph is biconnected. To transform this graph to one with the same biconnected components as the original graph, we must remove all the added edges $(v_i,v^{*})$ and $(v_i,v^{**})$, 2n changes. Since 2 is a multiplicative constant, the same bound applies as in the connected components case and this function is in **Class 2**.

## 4.3. Transitive Closure

Given a partial set of binary relations of the form $a \rightarrow b$, with $\rightarrow$ representing a transitive relation between $a$ and $b$, the Transitive Closure function determines all pairs in the complete relation set. The relationships can just as well be thought of as directed edges in a graph; this is equivalent to computing reachability in a digraph, and our results apply equally well to either function.

We add new elements $\xi$ and $\phi$ to the relation. For each element $a$ in the original input set, we add three relations, $a \rightarrow \xi, \phi \rightarrow a$ and $\xi \rightarrow \phi$. This is accomplished in time proportional to the length of the input. Clearly, the initial solution is that every element is related to every other by $\rightarrow$. Then in a single step we remove the relation $\xi \rightarrow \phi$. Now it is easy to see that the resulting relation has the same transitive closure as the original. Here $\epsilon(\ell)$ is 1; hence the transitive closure function has an IRLB of 1, and is in **Class 3.**

The transitive closure bound can easily be proved either as a graph or as a set of binary relations. In the next section we will consider problems represented as systems of equations and demonstrate that the same flavor of proof can be produced. In these problems too we can use a graphical representation of the equations to formulate our IRLB proofs.

## 5. Solving Systems of Equations

In this section, we are concerned with problems which can be formulated as a system of equations; that is, the solution to the system of equations yields the problem solution [8]. We demonstrate some common problems that are amenable to this representation. We categorize the systems of equations we consider as **constrained** or **unconstrained.** The former are systems for which the constants and coefficients are functionally related; the latter are those for which they are independent. Our discussions are in terms of systems in which the equations are linear, but our work generalizes to non-linear systems as well.

A problem instance of a function $\alpha$ consists of a set of coefficients and constants in domain $P$ which define a system of equations 5.1. In this context $A$, an algorithm which implements $\alpha$, is a solution procedure for that system of equations.

$$X_i = \sum_{k \in L_i} a_{ik} * X_k + c_i \quad \text{for } 1 \leq i \leq n \quad \text{where} \quad L_i \subseteq \{1, \ldots, n\} \tag{5.1}$$

Let $\sum_{i=1}^{n} |L_i| = p$. Then there are $pn+n$ elements in the input, the $a_{ik}$ and $c_i$. The binary operators $+$ and $*$ need not be addition and multiplication but they are required to share some of the properties of those operators, namely those necessary to allow the solution of the system. Figure 5-1 states these conditions.

We could use any function of the $\{X_1, \ldots, X_n\}$ in place of the term $X_k$ on the right hand side of the equations; this would allow non-linear interdependencies among the vari-

ables in the system. All of our subsequent arguments would hold; however, the problems of interest to us can be described in the simpler equation form 5.1.

---

i. $X_i \in P$

ii. $a_{ik}, c_i$ arbitrary values in $P$

iii. There is a $0_+ \in P$, with $x + 0_+ = x \ \forall \ x \in P$ (i.e., $0_+$ is the identity element with respect to $+$)

iv. There is a $1_+ \in P$, with $x + 1_+ = 1_+ \ \forall \ x \in P$

v. There is a $B \in P$ such that $B * 0_+ = 0_+$ and $B * 1_+ = 1_+$

**Figure 5-1:** Basic Conditions I

---

For each system of equations 5.1, we can build a digraph model of the variable interdependencies. Each variable corresponds to a node in the digraph; the appearance of $X_k$ on the right hand side of the equation for $X_j$ means that an edge $(k,j)$ appears in the digraph. We can think of $a_{ik}$, the coefficient for $X_k$, as the label on the edge $(k,j)$. Thus, we can speak of changes to the system of equations as changes to this digraph model. In the discussions which follow in this section, we consider the solution of equations 5.1 in the unconstrained and constrained cases.

## 5.1. Extreme Problem I--Unconstrained Equations

Consider the following problem instance 5.2, called Extreme Problem I, constructed from a problem instance of $\alpha$ assuming Basic Conditions I.

$$X_i = \sum_{k \in L_i} a_{ik} * X_k + B * X_{n+1} + c_i \quad \text{for } 1 \leq i \leq n \text{ where } L_i \subseteq \{1, \ldots, n\} \qquad (5.2)$$

$$X_{n+1} = c_{n+1}$$

The digraph representation for the variable interrelations added to the original system 5.1 when it is transformed into equations 5.2 is given in Figure 5-2.

Assume that algorithm $A$ solves $p$, a problem instance of $\alpha$; thus, $A$ solves a system of equations 5.1. An incremental algorithm for $\alpha$ $\Delta A$ can be applied to $p'$ which differs from $p$ by a single change to any $a_{ik}$ or $c_i$. Following the procedure outlined in Figure 2-1, we show how to use the incremental algorithm for $\alpha$ to build an algorithm for $\alpha$.

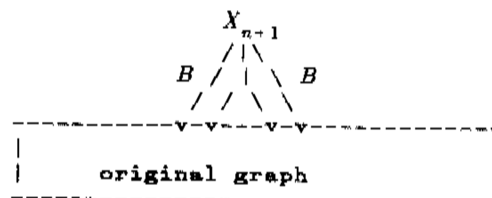


**Figure 5-2:** Additions to 5.1 Which Form Extreme Problem I

Our $p$ is the $p_{initial}$ of step 0. By adding $n$ coefficients and 1 constant we can easily transform $p$ into a related problem $s$ describable by equations 5.2; that is, we can perform steps 1.-2. with $\delta(\ell)=n+1$ which is dominated by $\ell$. If we choose $c_{n+1}=1_+$ we know that $s$ has a trivial solution, namely $X_i=1_+$, $1 \leq i \leq n+1$. Therefore steps 3.-4. are accomplished at constant cost. By changing $c_{n+1}$ to $0_+$ forming $s'$, we perform steps 5.-7. with $\epsilon(\ell)=1$. Note that a solution to $s'$ is also a solution to $p$. Thus, we can fashion an algorithm for $p$ from the incremental algorithm $\Delta A$ applied to $s'$. We summarize this procedure below, showing the corresponding steps in the procedure in Figure 2-1 in boldface. We know from Theorem 2.1 that problems describable as systems of equations 5.1 satisfying Basic Conditions I are **Class 3** problems, since $\epsilon(\ell)=1$ in this construction.

---

**Proof Construction---Extreme Problem I**

i. Given $p$, a problem instance of $\alpha$, construct a related instance $b$ of Extreme Problem I with $c_{n+1}=1_+$ (**0.-2.**). Then $b$ has the trivial solution $X_i=1_+$, $1\leq i\leq n+1$ (**3.-4.**).

ii. Form $b'$ from $b$ by changing $c_{n+1}$ to $0_+$ (**5.-6.**). Apply $\Delta A$ to $b'$ (**7.**).

---

An example of these problems is the minimum-maximum edge weight path problem [8]. Consider a collection of cities with some pairs of cities connected by roads. We want to travel from city $u$ to city $v$ in such a way that our route minimizes the maximal distance between any two cities on the route. This problem can be formulated as a set of equations 5.1 where $*$ is the maximum function and $+$ is the minimum function. For this problem the domain is the non-negative integers with $0_+ = \infty$ or any number larger than the maximum inter city distance in the particular problem instance and $1_+ = 0$. Then $X_i$ is the maximal distance on a route from city $i$ to city $v$; each $a_{ik}$ is the distance from city $i$ to city $k$ and each $c_i=0$. In the Extreme Problem I construction we add a new city $n+1$ which is a distance 0 away from every other city. Then we change the distance between $v$ and the new city to $1_+ = \infty$. The minimum-maximum edge weight path from any city to $v$ will be the same as if the new city were never added.

### 5.2. Constrained Problems

Now consider that we have problems corresponding to sets of equations 5.1 such that $a_{ik}$ and $c_i$ are functionally interdependent; that is, we are free to assign $a_{ik}$ any value in $P$, but then we cannot assign the $c_i$ independently. This is significant since it means that we cannot change an $a_{ik}$ without also changing $c_i$. Therefore, the change of an $a_{ik}$ and the resulting change of $c_i$ is considered a unit change to the input here. The same is true when we change $c_i$ and necessitate corresponding changes to $a_{ik}$.

Corresponding to these constrained problems we have two new sets of basic conditions, Basic Conditions II and III, where Basic Conditions III are a further restriction of Basic Conditions II. The problems satisfying Basic Conditions II can be shown to be in **Class 3** of our hierarchy. However, the problems satisfying Basic Conditions III are in **Class 2** and thus have a smaller IRLB than the other two categories of problems considered in this section. These discussions show that sometimes by restricting a problem, we can achieve a smaller relative worst case performance bound and thus perhaps have an efficient incremental algorithm.

#### 5.2.1. Extreme Problem II---Constrained Equations

Consider equations 5.1 with the additional constraints Basic Conditions II in Figure 5-3.

---

1. All the conditions given in Figure 5-1 hold except those on $a_{ik}, c_i$ which become $a_{ik}, c_i$ are values in $P$ constrained to be related by $c_i = f_i(a_{i1}, \ldots, a_{in})$ $1 \leq i \leq n$ for $f_i$ a function.
2. There are the following additional constraints:
   a. $a_{i1}, \ . \ . \ . \ , a_{in}$ can be chosen so that $a_{i1} * X_1 + \ . \ . \ . \ + a_{in} * X_n + f_i(a_{i1}, \ . \ . \ . \ , a_{in}) = 1_+$ when $f_i(a_{i1}, \ . \ . \ . \ , a_{in}) = 1_+$

   b. $\exists$ an element $0_* \in P$ such that $0_* * x = 0_*$ $\forall x \in P$.

   c. $a * X + c = 0_+$ if $a = 0_*$

**Figure 5-3:** Basic Conditions II

---

We define Extreme Problem II by equations 5.3 constructed from a problem instance of $\alpha$ assuming Basic Conditions II.

$$X_i = \sum_{k \in L_i} a_{ik} * X_k + B * X_{n+2} + c_i \quad \text{for } 1 \le i \le n \text{ where } L_i \subseteq \{1, \ldots, n\} \tag{5.3}$$

$$X_{n+1} = \sum_{k=1}^{n} a_{n+1,k} * X_k + c_{n+1}$$

$$X_{n+2} = a_{n+2,n+1} * X_{n+1} + c_{n+2}$$

where $a_{n+1,1}, \ldots, a_{n+1,n}$ are chosen so that

$$X_{n+1} = a_{n+1,1} * X_1 + \ldots + a_{n+1,n} * X_n + 1_+ = 1_+$$

The digraph representation for the variable interrelations added to the original system of equations 5.1 when it is transformed into equations 5.3 is shown in Figure 5-4.

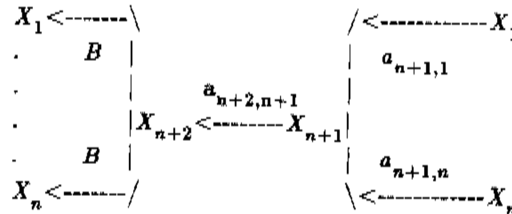


**Figure 5-4:** Additions to 5.1 to Form Extreme Problem II

We will use the same procedure as in Section 5.1 to transform $p$ a problem instance of $\alpha$ into $d$ an instance of Extreme Problem II, to change $d$ into $d'$ and finally to apply $\Delta A$ an incremental algorithm for $\alpha$ to yield a solution procedure for the original problem $p$. Corresponding steps in Figure 2-1 are noted in boldface. We know from Theorem 2.1 that problems describable as systems of equations 5.1 satisfying Basic Conditions II are **Class 3** problems, since $\epsilon(\ell)=1$ in this construction.

---

**Proof Construction---Extreme Problem II**

i. Given $p$, a problem instance of $\alpha$, construct a related instance $d$ of Extreme Problem II with $c_{n+1}=1_+$, $a_{n+2,n+1}=B$ (**0.-2.** with $\delta(\ell)=2n+3$). This problem instance has the trivial solution $X_i=1_+$, $1 \le i \le n+2$ (**3.-4.**).

ii. Form $d'$ from $d$ by changing $a_{n+2,n+1}$ to $0_*$ (**5.-6.** with $\epsilon(\ell)=1$). Apply $\Delta A$ to $d'$ (**7.**). Note that a solution to $d'$ is also a solution to $p$.

---

An example of a problem satisfying Basic Conditions II is the reaching definitions problem of data flow analysis [6, 11]. Assume we have a flow graph, a digraph represen-

tation of execution flow in a single-entry procedure. where the nodes contain straight-line code sequences and the edges represent possible transfer of control at execution time. We refer to a variable in the procedure being analyzed as a "program variable" and to a variable in the system of equations as "variable" in the discussion below. We associate a variable in our system of equations with each node in the flow graph such that $X_i$ is the set of program variable definitions in the procedure which reach the entry to node $k$ on some path from the procedure entry node. We also associate with each node a set of last definitions of program variables at that node; these are definitions which assign values to program variables that they will have upon exiting that node. Then $c_i$ is union of the sets of last definitions of program variables associated with immediate predecessors of node $i$ in the flow graph. The $a_{ik}$ are sets of definitions preserved through node $k$; that is, if $P$ is the union of all sets of last definitions of program variables in the procedure, then $a_{ik}$ is $P$ minus all definitions of program variables which have last definitions in node $k$. In the equations defining reaching definitions, $*$ is $\cap$ and $+$ is $\cup$. The Extreme Problem II construction is equivalent to adding two nodes to the flow graph. Node $n+2$ is the immediate predecessor of every other node in the flow graph. We use $a_{i,n+2} = 1_* = P$ for $1 \le i \le n$ in equations 5.3. Node $n+1$ is the immediate successor of every node in the flow graph and the immediate predecessor of node $n+2$; therefore, $X_{n+1} = P$. We use $a_{n+2,n+1} = P$ and then we change $a_{n+2,n+1}$ to $0_* = \emptyset$.

The other classical data flow problems: available expressions, very busy expressions and live uses of variables, are also examples of **Class 3** problems; however they can be represented by unconstrained equations 5.1.

### 5.2.2. Extreme Problem III---Further Constrained Equations

Consider a specialized class of problems which satisfy Basic Constraints II as well as some additional restrictions Basic Constraints III shown in Figure 5-5. We can show that these problems are actually in **Class 2** of our hierarchy. This is not a contradiction because by restricting the problems in Section 5.2.1 further, we open the possibility that a more efficient incremental algorithm can be designed for them.

Consider the following problem instance, called Extreme Problem III, constructed from a problem instance of $\alpha$ which satisfies Basic Conditions III.

1. All the conditions given in Figure 5-3 hold.
2. There are the following additional constraints:
   a. $\exists 1_* \in P$ such that $1_* * x = x \ \forall x \in P$
   b. the equations are of the form:

      $X_i = a_{i1} * X_1 + \ . \ . \ . \ + a_{in} * X_n$ where $\forall i,k \ a_{ik} = 0_*$ or $a_{ik} = 1_*$

   c. the system of equations is symmetric therefore $a_{ik} = a_{ki}$

**Figure 5-5:** Basic Conditions III

$$X_i = \sum_{k \in L_i} a_{ik} * X_k + 1_* * X_{n+1} + c_i \text{ for } 1 \leq i \leq n \text{ where } L_i \subseteq \{1, \ . \ . \ . \ , n\} \qquad (5.4)$$

$X_{n+1} = a_{n+1,1} * X_1 + \ . \ . \ . \ + a_{n+1,n} * X_n + c_{n+1}$

where $a_{n+1,1}, \ . \ . \ . \ , a_{n+1,n}$ are chosen initially so that

$X_{n+1} = a_{n+1,1} * X_1 + \ . \ . \ . \ + a_{n+1,n} * X_n + 1_+ = 1_+$

Note that 2c. in Basic Conditions III is significant because it means that a change to $a_{ik}$ implies a change to $a_{ki}$; therefore, we consider this a unit change to the input. The digraph representation for the variable interrelations added to the original system 5.1 when it is transformed into equations 5.4 is shown in Figure 5-6.

```
X_1<-------\              /<---------X_1
 .     1*  |              |  a_{n+1,1}
 .         |              |
 .         |------X_{n+1}--------|
 .     1*  |              |  a_{n+1,n}
X_n<-------/              \<---------X_n
```

**Figure 5-6:** Addition to 5.1 to Form Extreme Problem III

Following the same procedure as previously, we can construct an algorithm for $\alpha$ from an incremental algorithm for $\alpha$ applied to an instance of Extreme Problem III. We know from Theorem 2.1 that problems describable as a system of equations 5.1 satisfying Basic Conditions III are in **Class 2**, since $\epsilon(\ell) = n$ in this construction.

---

**Proof Construction---Extreme Problem III**

1. Given $p$ an instance of $\alpha$, construct $e$ a related Extreme Problem III with $c_{n+1}=1_+$ (**0.-2.** with $\delta(\ell)=2n+1$). This problem has the trivial solution $X_i=1_+$, $1 \le i \le n+1$ (**3.-4.**).
2. Form a sequence of problems $\{e_1', \ldots, e_n'\}$ from $e$ by changing each $a_{n+1,j}$ and $a_{j,n+1}$ from $1_*$ to $0_*$ in succession and applying $\Delta A$ to each problem $e_j'$ (**5.-7.** with $\epsilon(\ell)=n$).

---

Finding the connected components of a graph, a problem whose IRLB has been argued already in Section 4.2, is an example of one of these problems. It is equivalent to calculating reachability in a graph; that is, "Is node $i$ reachable from node $j$?". In this problem $X_i$ is the set of vertices in the graph reachable from node $i$. Here $*$ is $\cap$ and $+$ is $\cup$. In the equation for $X_i$, the $a_{ik}=1_*$ only for immediate neighbors of node $i$, reflecting the fact that the nodes we can reach from $i$ are either immediate neighbors of $i$ or nodes reachable from those neighbors; all other $a_{ik}=0_*$. Then $c_i$ is the set of all immediate neighbors of i. The Extreme Problem III construction is to add a new node which is a neighbor to every other node in the graph with $c_{n+1}=1_+$; therefore $X_{n+1}=1_+$. Here $1_*=1_+$ which is the set of all nodes in the graph. As we change each $a_{n+1,k}$ and $a_{k,n+1}$ to $0_*=\emptyset$, this also deletes $k$ from $c_{n+1}$ and $n+1$ from $c_k$.

## 6. Implications

### 6.1. A Collection of IRLB's

Figure 6-1 lists certain common algorithms for which we have proven IRLB's better than $1/\ell$. Some of the arguments appear in this paper; the others can be derived easily.

### 6.2. Implications for Updating Minimum Spanning Trees and Connected Components

Frederickson has presented an algorithm for incremental update of edges in a Minimum Spanning Tree (MST), and an extension that can be used to find the Connected Components in a graph [3, 4]. His algorithm runs in time $O(\sqrt{m})$, where $m$ is the number of edges in the graph. We can show that MST has an IRLB of $1/n$, where $n$ is the number of vertices in the graph. Certainly $m+n=\ell$ is a trivial lower bound for MST. For our

**Algorithms with O(1) IRLB's (Class 3)**

Shortest path in a digraph (various forms)

Transitive Closure of a binary relation

Planarity Testing

Strong Connectivity

Minimum-Maximum edge weight path

Maximum-Minimum edge weight path (dual of above)

Reaching Definitions

Domination

**Algorithms with $\leq 1/\ell, \geq 1/\sqrt{\ell}$ IRLB's (Class 2)**

Connected Components

Biconnected Components

Minimum Spanning Tree

Shortest path, undirected graph

**Figure 6-1:** Categorization of Sample Problems

purposes we will refer to a set of MST problems as **dense** if $m = kn^2, 0<k\leq 1$. For such a set of problems, the length of the problem, $\ell$, is proportional to $n^2$, and we can derive an incremental lower bound of $\ell/n = n^2/n = n = \sqrt{m}$. Thus for dense graph sets, Frederickson's algorithm is optimal to within a constant factor. This suggests that if the number of edges grows more slowly than $n^2$, there may be some improvement possible over Frederickson's result. This may occur for example if $m$ is bounded by some constant times $n$. An identical argument to this can be made for Connected Components as well.

### 6.3. Implications for Incremental Data Flow Analysis

The classical data flow analysis problems: reaching definitions, live uses of variables, available expressions, very busy expressions, all can be formulated as systems of equations (5.1) [6, 11]. By the arguments in Section 5 they can be shown to be in **Class 3**. This implies that any incremental algorithm for these problems can have no better worst case performance than the best known algorithm for them.

In previous work on incremental algorithms for data flow analysis, Ryder distinguished between two types of incremental changes to these inputs. One type of change occurs because code within a node of the flow graph changes, affecting the $a_{ik}$ or $c_i$. The other type of change also corresponds to code changes, but the structure of the flow graph is af-

fected; that is, an edge or node is added to or deleted from the graph. We refer to the latter changes as "structural changes". Ryder's incremental data flow analysis algorithms handled only non-structural changes in code; however, they seem extendible to structural changes as well [1]. Assuming the algorithms are extendible, our results show that their worst case behavior can be no better than the best known data flow analysis algorithm. This justifies Ryder's claim that worst case analysis is not an appropriate measure of the utility of these incremental algorithms. It also justifies the alternative performance analyses on a reasonable, structured programming language [10].

An interesting question under investigation is the possibility that incremental algorithms which handle only certain classes of code changes or which are applicable to a restricted set of $a_{ik}$, $c_i$ may have better worst case performance than indicated by our IRLB results. By restricting Allen/Cocke interval analysis, on which Ryder's incremental data flow algorithms are based, to programs whose loop nesting level is bounded by a constant, we can improve the worst case bound from $O(n^2)$ to $O(n)$ [10]. Similar results may be achievable with incremental data flow algorithms.

## 7. Conclusion

We have presented a general method for computing a type of lower bound on incremental computation, and applied it to a number of problems common in the literature. These results permit us to evaluate available incremental algorithms, and conjecture where better ones might be found. The bounds apply only to worst-case analysis; algorithms with better average case or amortized complexity are not ruled out.

## REFERENCES

**1.** M. Burke. An Interval Analysis Approach Toward Interprocedural Data Flow Analysis. Computer Science Technical Report RC 10640, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, July, 1984.

**2.** Shimon Even and Hillel Gazit. Updating distances in dynamic graphs.

**3.** Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees. Symposium on Theory of Computing, ACM-SIGACT, April, 1983, pp. 252-257.

**4.** Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. CSD TR 449, Purdue University, West Lafayette, IN 47907, July, 1983.

**5.** Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. Symposium on Foundations of Computer Science, IEEE Computer Society, 1984, pp. 338-346.

**6.** M. S. Hecht. *Flow Analysis of Computer Programs.* Elsevier North-Holland, 1977.

**7.** Marvin C. Paull, Charles Ching-an Cheng, and A. Michael Berman. Exploring the structure of incremental algorithms. DCS-TR-139, Rutgers University, New Brunswick, NJ 08903, May, 1984. (Preliminary Version).

**8.** Marvin C. Paull. *Introduction to Algorithm Design Principles.* Wiley-Interscience, 1985. pre-publication manuscript.

**9.** Barbara G. Ryder. Incremental data flow analysis. ACM Symposium on Principles of Programming Languages, ACM-SIGPLAN, January, 1982, pp. 167-176.

**10.** Barbara G. Ryder and Marvin C. Paull. Incremental data flow analysis algorithms. To appear in ACM Transactions on Programming Languages and Systems.

**11.** Barbara G. Ryder and Marvin C. Paull. A unified model of elimination algorithms. Being reviewed for publication in ACM Computing Surveys.