COMPLEXITY OF BIDIRECTIONAL DATA FLOW ANALYSIS

Dhananjay M. Dhamdhere
dmd@cse.iitb.ernet.in

Uday P. Khedker
uday@cse.iitb.ernet.in

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay, 400 076 .

Abstract

The concept of an *information flow path* arising from the generalized theory of data flow analysis [21] is used to analyze the complexity of data flow analysis. The *width* (w) of a program flow graph with respect to a class of data flow problems is introduced as a measure of the complexity of round-robin iterative analysis. This provides the first known complexity result for round robin iterative analysis of bidirectional data flows commonly used in algorithms based on the suppression of partial redundancies [6, 7, 8, 9, 17, 18, 25]. We also show that width provides a better bound on the complexity of unidirectional data flows than the classical notion of *depth*.

The paper presents ways to reduce the width, and thereby the complexity of flow analysis, for several interesting problems. Complexity analysis using the notion of width is also shown to motivate efficient solution methods for various bidirectional problems, *viz.* the *alternating iterations* method, and an interval analysis based elimination method for the partial redundancy elimination problems. The paper also presents a condition for the decomposability of a bidirectional problem into a sequence of unidirectional problems.

1 Introduction

The data flows used in classical code optimization mostly involve *unidirectional* dependencies, i.e. the data flow information at a node of the program flow graph is influenced either by its predecessors, or its successors. Such data flows can be readily classified into *forward* and *backward* data flows [1]. *Bidirectional* problems, wherein the information at a node depends on its predecessors as well as its successors, arise in several unifications of optimizing transformations. The Morel and Renvoise Algorithm (MRA) for suppression of partial redundancies [25], which unifies the traditional optimizations of code movement, common subexpression elimination and loop optimization; and the Composite Hoisting and Strength reduction Algorithm (CHSA) [8, 17, 18], are examples of such unifications. Their appeal lies in the reductions in optimizer sizes and running times offered by them.

ACM-20th PoPl-1/93-S.C.,USA

Classical work in complexity analysis of data flow problems uses the characteristics of control flow graphs, *viz.* the number of nodes, edges or back edges in a flow graph, as the basis of the complexity study. Though bidirectional data flow problems have been known for over a decade, it has not been possible to explain their intricacies, or determine their complexities, using the traditional theory of data flow analysis. A generalized theory offering a uniform treatment of the unidirectional and bidirectional problems has shown that the bidirectional problems are no more complex than the unidirectional problems as far as the order of the work involved is concerned [21]. However certain complexity issues, *viz.* the bound on the number of iterations in round robin iterative data flow analysis of bidirectional flows, have remained un-addressed.

In this paper, we use the key concept from the generalized theory of data flow analysis [21], namely the concept of an *information flow path*, to analyze the complexity of data flow analysis. We use the nature of an information flow path as the basis for a classification of data flow problems. We then combine the concept of an information flow path with the properties of graph traversals to evolve the notion of the *width* (w) of a program flow graph with respect to a class of data flow problems. The width is shown to be a measure of the complexity of round robin iterative data flow analysis for a class of problems. This is the first time a complexity result has been provided for the round robin iterative solution of bidirectional data flows. Another significant outcome is the demonstration that the classical bound of $d + 1$ iterations for the solution of unidirectional problems, where d is the depth of the flow graph, is a loose bound. A tighter bound using the notion of width is provided in section 3.

We present ways to reduce the width, and thereby the complexity of flow analysis, for a set of interesting problems. The study of information flow paths also yields valuable insight into the behaviour of data flows, which motivates more efficient techniques for the solution of data flow problems, including bidirectional problems. We use this insight to devise efficient methods for the solution of several bidirectional data flow problems, *viz.* the method of *alternating iterations*, which is applicable to useful variants of the partial redundancy elimination algorithm, and an interval analysis based elimination method applicable to the Morel Renvoise Algorithm [25], and the Composite Hoisting and Strength reduction Algorithm [8, 17, 18].

The feasibility of decomposing a bidirectional data flow problem into a sequence of cascaded unidirectional problems has been the subject of some discussion in the literature [6]. We show that the nature of information flow paths is useful in this context as well. We prove a general condition for the feasibility of decomposition and demonstrate its applicability to some interesting bidirectional problems.

A brief description of the bidirectional data flow problems referred in this paper is contained in the appendix. Section 2 defines the concept of an *information flow path*, based on which section 3 defines the notion of width. Section 4 relates width to the efficiency of data flow analysis and presents several methods for efficient data flow analysis of bidirectional data flow problems. Section 5 discusses the feasibility of decomposing a bidirectional data flow problem into cascaded unidirectional problems. The concluding section presents experimental results and analyzes their implications.

2 Information flow paths

2.1 Traversals

Given a depth first spanning tree, we differentiate between *back edges* and non-back edges; we term the latter as *forward edges* in this paper.[1] Thus the term forward edges, as used in this paper, includes the conventional notions of forward as well as cross edges [1].

Information flow is realized during a *traversal.* We define the notions of traversal as follows :

(i) *Edge Traversals* : Forward traversal along an edge is a tail-to-head traversal of the edge, while the backward traversal is a head-to-tail traversal.

(ii) *Graph Traversals* : A forward traversal over a graph visits the nodes of the graph in a depth first order. A backward traversal visits the nodes in a reverse depth first order.[2]

F_e	:	Forward traversal along an edge
B_e	:	Backward traversal along an edge
F_f	:	Forward traversal along a forward edge
B_f	:	Backward traversal along a forward edge
F_b	:	Forward traversal along a back edge
B_b	:	Backward traversal along a back edge
F_G	:	Forward traversal over the graph
B_G	:	Backward traversal over the graph

Figure 1: Notations for traversals

A graph traversal depends on the ordering imposed by a DFS tree while an edge traversal does not. Figure 1 summarizes the notations used in this paper. The direction of traversal is indicated by B or F with subscript e for the edge traversal and G for the graph traversal. For edge traversal, subscripts f and b are used instead of e if forward and back edges need to be distinguished.

An iteration in a round robin iterative analysis realizes one graph traversal.

2.2 The notion of information flow [21]

We restrict our discussion to *bit vector problems* [15] wherein each bit can be treated independently, i.e. the value of a bit i in a bit vector is influenced only by the corresponding bits in other bit vectors of the program flow graph.

Let TOP and BOT refer to the values of a bit in $\top$ and $\bot$ bit vectors, respectively. A BOT value for a data flow property implies a useful item of information from the viewpoint of data flow analysis (*viz.* a reaching definition), whereas a TOP value implies that such information can not be concluded. For iterative data flow analysis, the data flow properties are initialized to TOP for all nodes (except for the graph entry/exit nodes, which may have other values). Some properties change to BOT due to the local effect of computations in a node, *viz.* when a definition is generated, or an expression is killed. In turn, these change the properties of the neighbouring nodes to BOT.

Definition 1 : Information Flow

Information is said to flow *from node i to some neighbouring node j when a property of i, on becoming* BOT, *causes the corresponding property of j to become* BOT.

Note that the information flow is transitive. Incorporation of the information flows due to all BOT prop-

[1] We use the terms *forward edges* and *back edges* as synonyms of *advancing edges* and *retreating edges* respectively. We prefer the former because it expresses the intuitive notion of direction more clearly. For non-reducible flow graphs, a back edge in this paper means a retreating edge.

[2] We assume that a visit to a node leaves the IN and OUT properties of the node in mutually consistent states.

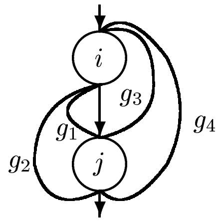


Figure 2: Edge flow functions

erties in the program flow graph leads to a fixed point of the data flow equations [21].

2.3 Flow functions

There are two fundamental kinds of flows in a data flow problem :

(i) Information flows *within* a node, i.e. between the entry and exit of node i : Represented by *node flow* functions $f(i)$.

(ii) Information flows along edge (i,j) : Represented by *edge flow* functions $g(i,j)$. (Figure 2)

$g_1(i,j)$: Information flow between the exit of i and the entry of j.

$g_2(i,j)$: Information flow between the exit of i and the exit of j.

$g_3(i,j)$: Information flow between the entry of i and the entry of j.

$g_4(i,j)$: Information flow between the entry of i and the exit of j.

The flow functions are superscripted by f or b to indicate whether the flow is in the forward or the backward direction. The flow functions are determined directly from the data flow equations governing a problem. If a flow is not present, the corresponding flow function is $\top$. We refer to the functions by their type names f, g_1, g_2, g_3 and g_4 respectively with an appropriate superscript f or b. Table 1 lists the function types for the data flow problems defined in the appendix.

The edge flow functions of the type g^f (g^b) are said to be *clustered* if the information flow is identical for all out-edges (in-edges) of a node.

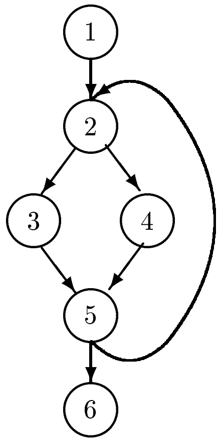
2.4 Information flow paths

Definition 2 : **Information flow path**

An information flow path (ifp) is an allowable sequence of edge traversals $\bar{e}_1, \bar{e}_2, \ldots, \bar{e}_k$ *along which information can flow during data flow analysis.*

Note that a traversal over an edge $\bar{e}_i$ could be a forward or a backward traversal over the graph edge e_i. Thus an information flow path may not be a graph theoretic path. We represent an *ifp* ρ from program point p to a program point q by $< p, q, \rho >$ where a program point refers to the entry/exit of a basic block. Where convenient, we will represent an *ifp* simply as a sequence of nodes, leaving the traversal of the edges connecting these nodes implicit.

Example 1 : Consider an F_G traversal over the graph



Some of the possible *ifp*'s are :

(i) $(1,2,4,5,6) = F_f\ F_f\ F_f\ F_f$

(ii) $(1,2,5,6) = F_f\ B_b\ F_f$

(iii) $(6,5,3,2,1) = B_f\ B_f\ B_f\ B_f$

(iv) $(6,5,2,1) = B_f\ F_b\ B_f$

(v) $(3,2,4) = B_f\ F_f$

(vi) $(3,5,4) = F_f\ B_f$

□

Sequences of edge traversals along which information may not flow can be determined by *syntactic* criteria, i.e. knowing which node and edge functions are absent (i.e. $= \top$), as well as *semantic criteria*, i.e. knowing the nature of the actual data flow.

Example 2 : Syntactic criteria indicate that the *ifp*'s for the problem of available expressions cannot contain a backward edge traversal (since all the backward edge flow functions are $\top$) and the *ifp*'s for MRA cannot contain two successive forward edge traversals (since $f^f = \top$). Thus *ifp* (i) of example 1 cannot be an *ifp* for MRA. Example 4 shows how the *ifp*'s of EPA cannot contain two backward edge traversals following a forward edge traversal due to the semantic criteria. □

For monotone data flows, an *ifp* is necessarily acyclic. Note, however, that the underlying graph theoretic path may be cyclic since a node may appear in the path once for its entry point and once for its exit point.

Problem	Function types	Class $\mathcal{C}$
Reaching Def.	f^f, g_1^f	$< \{\mathrm{OUT}\}, \{f^f, g^f(1000)\} >$
Live Variables	f^b, g_1^b	$< \{\mathrm{IN}\}, \{f^b, g^b(1000)\} >$
MRA	f^b, g_1^b, g_1^f	$< \{\mathrm{IN}\}, \{f^b, g^f(1000), g^b(1000)\} >$
LSIA	f^f, g_1^f, g_1^b	$< \{\mathrm{OUT}\}, \{f^f, g^b(1000), g^f(1000)\} >$
MMRA	f^b, g_1^b, g_1^f, g_3^f	$< \{\mathrm{IN}\}, \{f^b, g_\Pi^f(1000), g_\Sigma^f(0010), g_\Pi^b(1000)\} >$
EPA	f^b, g_1^b, g_3^f	$< \{\mathrm{IN}\}, \{f^b, g_\Sigma^f(0010), g_\Pi^b(1000)\} >$
CHSA	f^f, f^b, g_1^b, g_1^f	$< \{\mathrm{OUT}\}, \{f^f, f^b, g^f(1000), g^b(1000)\} >$

Table 1: Classification of data flow problems considered in this paper

Example 3 : An *ifp* $(2, 5, 3, 2) = B_b B_f B_f$ in the graph of example 1 is a valid information flow path for MRA as PPIN_2 may affect PPOUT_2 along such a path. □

For bit vector problems, the most general form of the flow functions is : $h(X) = A + \neg B \cdot X$. The function h is called a *preserve* function if $h(\mathrm{BOT})$ = BOT. A preserve function guarantees the propagation of information under all conditions. Preserve functions have the form $h(X) = X$ and $h(X) = A + X$ for the union problems (here, BOT = 1) and the form $h(X) = X$ and $h(X) = \neg B \cdot X$ for the intersection problems (BOT = 0).

Definition 3 : **Information preserving path**
An ifp is an information preserving path (ipp) if all flow functions in the ifp are preserve functions.

2.5 Classification of data flow problems

A data flow problem is said to be *non-singular* if it involves more than one distinct confluence operator. Thus the EPA and MMRA data flow problems of the appendix are non-singular, while all other problems are singular (refer to the appendix for the data flow equations).

Data flow problems are classified according to the nature of the information flow paths. A class $\mathcal{C}$ is a tuple <{*path origin*}, {*flow function types*}> where :

(i) Path origin is a program point represented as IN/OUT.

(ii) The node flow function types are represented by f^f/f^b. Edge flow functions are represented by $g(b_1 b_2 b_3 b_4)$ where b_i is 1 if the corresponding flow function is $\neq \top$. For non-singular data flows, the function name is subscripted by the operator.

Table 1 contains the classification of the problems referred in the paper. Note the dualism among the classes with respect to the path origin and the function types. For simplicity, we will name the classes by their representative data flow problems. Hence we will use the names appearing in the first column of Table 1 as the class names.

The significance of classes lies in the fact that for a given graph, all problems in a class possess similar *ifp*'s and hence obey the same complexity bound. However, using the knowledge of a data flow problem it may be possible to develop a more specific definition of its *ifp*'s, and hence a more specific bound for it.

We define the information flow paths for a data flow problem (or a class of data flow problems) by a regular expression over the set of edge traversals. Table 2 contains the regular expressions for the data flow problems defined in the appendix. Example 4 below shows how a more specific regular expression for the *ifp*'s of the EPA data flow problem is obtained from that for the EPA class of data flow problems.

Example 4 : Consider the MMRA and EPA problems. MMRA, and other members of its class, have *ifp*'s which can be represented by the regular expression $(B_e \mid F_e)^+$. The *ifp*'s of the EPA class of problems are represented by the same regular expression. However, the *ifp*'s of the EPA data flow problem can be more precisely represented by $({B_e}^+ \mid F_e)({F_e}^* B_e \mid {F_e}^*)$. This is due to the nature of the EPA data flow (eqs. 7, 8 of figure 6), which restricts the backward flow resulting from the PPIN property turning BOT due to the g_Σ^f function merely to the OUT property of the predecessors (since the IN property of the predecessors is already BOT). □

3 The width of a graph

To simplify the presentation, we represent the forward and backward directions generically by δ. For example, if δ is the forward direction, then an F is replaced by δ while a B is replaced by δ^-. Figure 3 summarizes the generic notation. Thus $\delta_G = F_G$ if the graph traversal visits the nodes of a graph in a depth first order.

Definition 4 : **Conforming and Non-conforming edge traversals**

Problem	Function types	Information flow paths
Reaching Def.	f^f, g_1^f	${F_e}^+$
Live Variables	f^b, g_1^b	${B_e}^+$
MRA	f^b, g_1^b, g_1^f	$({B_e}^+(F_e \mid \epsilon))^+{B_e}^*$
LSIA	f^f, g_1^f, g_1^b	${F_e}^+((B_e{F_e}^+)^* \mid B_e)$
MMRA	f^b, g_1^b, g_1^f, g_3^f	$(B_e \mid F_e)^+$
EPA	f^b, g_1^b, g_3^f	$({B_e}^+ \mid F_e){F_e}^*(B_e \mid \epsilon)$
CHSA	f^f, f^b, g_1^b, g_1^f	$B_e(B_e \mid F_e)^*$

Table 2: Information flow paths of the data flow problems referred in this paper

δ_f	:	Traversal along a forward edge in direction δ
δ_b	:	Traversal along a back edge in direction δ
δ_f^-	:	Traversal along a forward edge in direction δ^-
δ_b^-	:	Traversal along a back edge in direction δ^-
δ_G	:	Traversal over the graph in direction δ
δ_G^-	:	Traversal over the graph in direction δ^-

Figure 3: Generic notation for various traversals.

For a δ_G traversal, δ_f and δ_b^- edge traversals are conforming edge traversals while δ_f^- and δ_b are non-conforming edge traversals.

It is readily seen that the δ_f and δ_b^- traversals are compatible with δ_G traversal whereas the δ_f^- and δ_b traversals are in the opposite direction.

Definition 5 : Span

A span is a maximal sequence of conforming edge traversals in an ifp.

Spans are separated by a non-conforming edge traversal and vice-versa. Thus, two successive non-conforming edge traversals have a *null span* between them. Further, an information flow path may begin and/or end with a null span.

The information along a span can be propagated in one δ_G traversal; the same graph traversal also realizes the information flow along the preceding non-conforming edge traversal.

Definition 6 : Segment

A segment is a maximal sequence of successive edge traversals in the same direction.

Successive F_e's constitute a *forward* segment while successive B_e's constitute a *backward* segment. A segment may be *bounded* or *unbounded.*

Example 5 : Consider an F_G graph traversal and an *ifp*

$$\overbrace{\underbrace{F_f\ F_f\ F_f}\ \underline{F_b}}\ \underbrace{\overbrace{B_b}\ \overbrace{F_f}}\ \overbrace{\underline{B_f}\ \underline{B_f}}$$

The underbraces denote the (non-null) spans; overbraces, segments; and underscores denote the non-conforming edge traversals. Note that there is a null span between two successive B_f's. □

Example 6 : The *ifp*'s of unidirectional data flow problems consist of a single unbounded segment. From table 2, MRA, MMRA, EPA and CHSA have *unbounded* backward segments. MMRA, EPA and CHSA have unbounded forward segments too, while MRA has a bounded forward segment. □

For an *ifp* $< p, q, \rho >$, let $length(\rho)$ and $width(\rho)$ denote the total number of edge flow functions and the number of edge flow functions along non-conforming edge traversals, respectively.

Definition 7 : Bypassed information flow path

An ifp $< p, q, \rho_1 >$ is said to be bypassed by $< p, q, \rho_2 >$ if the edge functions of p are clustered, and

(i) either ρ_2 is an ipp or $length(\rho_2) = 1$, and

(ii) $width(\rho_2) < width(\rho_1)$.

Intuitively, $< p, q, \rho_1 >$ is bypassed by $< p, q, \rho_2 >$ if the same information is guaranteed to flow along ρ_2 and $length(\rho_2) < length(\rho_1)$. Such a guarantee can be given only if ρ_2 is an *ipp* or $length(\rho_2) = 1$ and the edge functions are clustered.

In practical data flow problems, bypassing usually occurs due to $length(\rho_2) = 1$.

Definition 8 : Width

The width w of a graph G for a class of data flow problems with respect to a traversal δ_G is the maximum number of non-conforming edge traversals along an ifp, no part of which is bypassed.

If we represent the number of spans by s then $s = w + 1$ for the width determining path.

Theorem 1 : *$w + 1$ iterations are sufficient for the round robin algorithm to converge on a fixed point.*

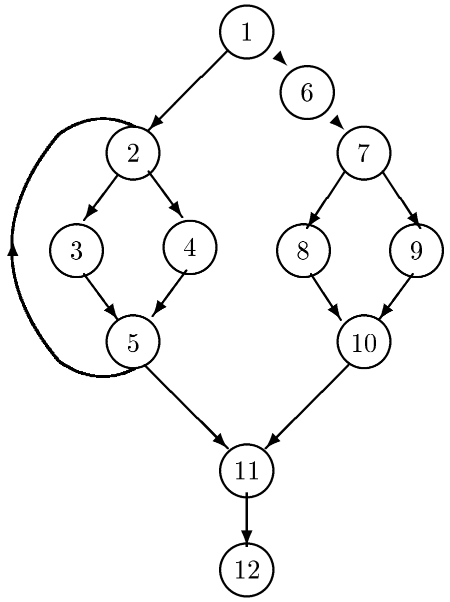
Proof : The information flow can be initiated only after the IN and OUT properties of all nodes are computed to determine the information originating within the nodes. This can be achieved in the first iteration.

The same iteration also realizes the propagation of information along a non-null span (if any) at the beginning of an *ifp*. However, every non-conforming edge traversal, and the span that follows it, requires a separate iteration. Thus, $w+1$ iterations are sufficient for information propagation along the width determining path.

Now consider an *ifp* $< p, q, \rho_1 >$ such that $width(\rho_1) > w$. This is possible only if a section ρ' of ρ_1 is bypassed by another *ifp* ρ''. Let $width(\rho')$ be w', $width(\rho'')$ be w'' and $width(\rho_1)$ be w_1. Then $(w_1 - w') + w'' \leq w$. Again $w + 1$ iterations suffice for information to propagate from p to q. □

Note that the bound $w+1$ is a static prediction. For a particular instance of a data flow problem, the number of iterations could be less as the behaviour of the non-preserving *ifp*'s (i.e. *ifp*'s containing functions of the form $h(X) = A + \neg B \cdot X$) is governed by the constants A and B for that instance.

Example 7 : Consider the following graph



If we choose $\delta_G = B_G$, B_b and F_f become the non-conforming edge traversals. The width determining paths for MRA, EPA and MMRA are :

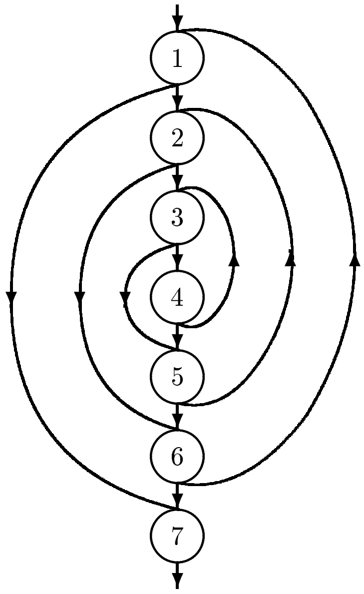
- $(6, 1, 2, 5, 11, 10, 9, 7, 8) = B_f F_f B_b F_f B_f B_f B_f F_f$
- $(1, 6, 7, 8, 10, 11, 12) = F_f F_f F_f F_f F_f F_f$
- $(1, 6, 7, 8, 10, 11, 12) = F_f F_f F_f F_f F_f F_f$

The widths for the three problems are 4, 6, and 6, respectively. □

3.1 The width and the depth

Depth (d) is defined as the maximum number of back edges along any acyclic path [1]. To use the notion of width for unidirectional flows, choose δ as the natural direction of the flow in the problem. Now there are no flows along δ_f^- and δ_b^- edge traversals, and the only non-conforming edge traversal is δ_b. Since the width considers only those paths which do not have bypassed fragments, $w \leq d$. Thus, width provides a tighter bound on the number of iterations.

Example 8 : Consider a *spiral graph* [3] whose depth increases linearly with the nesting depth.[3]



Here $d = 3$, while $w = 1$ for a unidirectional problem since every part of an *ifp* beginning on a back edge is bypassed, *viz.* path $(5, 2, 6)$ is bypassed by the path $(5, 6)$. This explains the observation that the number of iterations remains constant even as the size of the spiral graph grows. □

Further, the notion of depth assumes a fixed pattern for information flow governed by the directed paths in the flow graph, hence it is only applicable to unidirectional data flow problems.

4 Efficiency of data flow analysis

This section discusses the issues raised by the presence of unbounded segments in information flow paths, and the importance of bounded segments in reducing the complexity of data flow analysis.

4.1 Choice of direction in graph traversal

Consider a data flow problem whose *ifp*'s have unbounded segments in one direction. Recall that $w = \#\delta_b + \#\delta_f^-$, i.e. width has contributions from the

[3]Spiral structures result from **repeat** ... **until** loops with premature exits.

back edges in the δ segments and forward edges in the δ^- segments. $\#\delta_f^-$ is likely to be smaller when the unbounded segments are in the δ direction, rather than the δ^- direction. Hence the appropriate direction for graph traversal is the one that makes the unbounded segments lie in the δ direction.

For example, MRA has unbounded backward segments but bounded forward segments, hence backward graph traversal would require fewer iterations than forward traversal. For unidirectional problems, the favoured direction of traversal is trivially the direction of the data flow.

Alternating iterations

For problems with unbounded segments in both directions, alternating the direction of graph traversal between successive iterations can effectively reduce the solution complexity. This can be explained as follows : A segment in the δ direction may consist of a number of spans separated by non-conforming edges, which may themselves form sizable spans in the δ^- direction. Since the effect of a span in the δ direction is incorporated by a single iteration in the δ direction, alternating the direction of graph traversal between successive iterations would yield better results. Thus, the alternating iterations approach is clearly warranted in the case of EPA and MMRA.

Let s_ρ be the number of non-null spans along an *ifp* ρ. The width of ρ for alternating iterations is defined as follows : $width_a(\rho) = 2s_\rho + c$, where $c = 1$ if ρ ends with a non-null span, else $c = 0$. The number of iterations for the method of alternating iterations is then $w_a + 1$ where w_a is defined analogous to w, *viz.* $w_a = \max(width_a(\rho))\ \forall\rho$, where ρ is an *ifp*, no part of which is bypassed.

4.2 Handling bounded segments

When a data flow problem has bounded segments in one direction, and unbounded segments in the other direction, complexity of the data flow analysis can be reduced by attempting to *truncate* the information flow paths. Consider an *ifp* $\rho = (\cdots, \bar{e}_i, \bar{e}_{i+1}, \bar{e}_{i+2})$, where edge $\bar{e}_{i+1}$ constitutes a bounded segment of length 1, i.e. $\bar{e}_i$ and $\bar{e}_{i+2}$ belong to the segments in the opposite direction. Truncation can be effected by transforming the program flow graph or the data flow equations so as to terminate each *ifp* analogous to ρ *before* it reaches the edge $\bar{e}_{i+2}$. Effectively, ρ is split into two *ifp*'s ρ_1 and ρ_2. Since $width(\rho_1), width(\rho_2) \leq width(\rho)$, this could reduce the width. In this section, we present three transformations based on this approach.

4.2.1 Edge splitting

An edge which runs from a *branch node* (i.e. a node with more than one successor) to a *join node* (i.e. a node with more than one predecessor) is called a *critical*

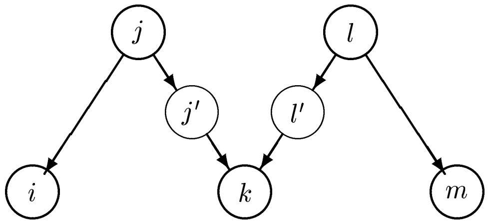


Figure 4: Edge splitting

edge. It has been reported [11] that when such an edge is split by inserting a new node, the solution complexity of MRA is reduced. The following lemma captures the influence of the edge splitting graph transformation on the solution complexity of MRA.

Lemma 1 : *Following edge splitting, MRA can be solved with the complexity of a unidirectional problem.*
Proof : Following section 4.1, we choose as $\delta_G = B_G$ for MRA. Consider figure 4. Before edge splitting, the information flow path from node i to node m ($\rho = (i, j, k, l, m) = B_f F_f B_f F_f$) would have contained two δ_f^- (i.e. F_f) edge traversals. With the insertion of j', ρ is split into two *ifp*'s $\rho_1 = (i, j, j')$ and $\rho_2 = (k, l', l, m)$, which contain a single δ_f^- traversal each. (This is because a $\rho' = (i, j, j', k, l', l, m)$ is not an *ifp* for MRA due to two successive δ_f^- traversals (j, j') and (j', k)). In general, edge splitting restricts the number of δ_f^- edge traversals along any information path to ≤ 1 in the transformed graph.[4] Thus $w \leq w' + 1$, where w' is the width of the graph for a unidirectional data flow problem. □

As a result of edge splitting the MRA *ifp*'s become ${B_e}^+(F_e \mid \epsilon)$ for the transformed graph, instead of the original $({B_e}^+(F_e \mid \epsilon))^+{B_e}^*$.

4.2.2 Edge placement

The technique of *edge placement* eliminates a partial redundancy of an expression e in node i, which cannot be safely hoisted into a predecessor j, by creating a synthetic node along the edge (j, i) and hoisting e into it [6]. Unlike edge splitting, however, a synthetic node is conceptual; it does not participate in data flow analysis. It becomes real only when a computation is inserted in it during optimization phase following the data flow analysis.

Use of edge placement in MRA results in elimination of the Π term from the PPIN_i equation (eq. 1 of figure 6). It thus transforms the data flow, rather than the flow graph, to achieve the same effect as edge splitting, *viz.* restricting the number of δ_f^- traversals along any *ifp* to < 1. Solution efficiency vis-a-vis MRA is guaranteed by the fact that the resulting data flow is simply (backwards) unidirectional in nature.

[4] < 1 in the original graph.

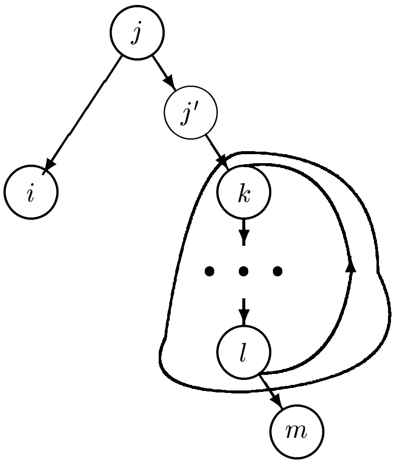


Figure 5: Interval based solution for MRA

4.2.3 Interval pre-headers

[10] mentions that an interval based elimination method can not be extended to MRA. The notion of width can be used to justify this, and to motivate an edge splitting graph transformation which enables an elimination approach to MRA solution. Consider a forward edge (j, k) terminating on the interval header (figure 5). The information flowing forward along this edge can flow (backwards) along the latching edge (k, l) of the interval to reach the exit edge (l, m) of the interval. Thus, information can flow forward 'through' an interval, as well as back through it due to the mostly backward nature of MRA.

However, if we introduce an interval pre-header by splitting every inter-interval edge (i.e. every edge in a reduced graph), we will effectively truncate an information flow path before it reaches an interval-header along a forward edge, *viz.* the path $\rho = (i, j, k, l, m)$ would be truncated to $\rho_1 = (i, j, j')$. This restricts the information flow between two intervals to a unidirectional flow. Thus the bidirectional problem is partitioned into smaller sub-problems related to each other by a unidirectional data flow. This makes the classical interval analysis approach feasible for MRA.

A detailed discussion of elimination algorithms for bidirectional data flow problems can be found in [20].

5 Decomposing bidirectional flows into unidirectional flows

Decomposition of a bidirectional data flow problem into a sequence of unidirectional problems (i.e. solving a bidirectional problem as a sequence of cascaded unidirectional problems) is motivated by the desire to reduce the amount of work or to improve the understandability of the data flow involved. Prior work on decomposition has been *ad hoc* and/or directed at specific bidirectional data flow problems [6, 11, 23]. In this section we provide a condition for the decomposability of a bidirectional data flow problem.

Lemma 2 : *It is feasible to decompose a bidirectional data flow problem into a sequence of unidirectional data flow problems if and only if the number of segments in every information flow path for the data flow problem is bounded by a constant.*

Proof : Let δ be the direction of the first segment in an information flow path. Information propagation along the segment can be realized by a unidirectional data flow problem which has δ as its natural direction of flow. Information flow along the following segment would require a unidirectional problem in the opposite direction, etc. Thus, the number of unidirectional problems required will equal the number of segments, which should be bounded by a constant for the decomposition to be feasible. Further, the order of solving the unidirectional problems will have to be the same as the order of segments in the information flow paths. □

Corollary 2.1 : *MRA cannot be solved by cascaded unidirectional problems.*[5]

The *ifp*'s of MRA have the form $({B_e}^+(F_e \mid \epsilon))^+ {B_e}^*$, thus they can have an unbounded number of forward and backward segments. Similar statements hold for the LSIA and CHSA problems.□

In the following, we explain two decompositions reported in literature, and motivate a third one.

Corollary 2.2 : *It is possible to decompose MRA if edge splitting is performed.*

Following section 4.2.1, the information flow paths in the resulting program flow graph can be characterized by the regular expression ${B_e}^+(F_e \mid \epsilon)$. Since the number of segments can at most be 2, it is possible to solve MRA by cascaded unidirectional problems. Further, since the second (i.e. the forward) segment has a length ≤ 1, it is possible to solve MRA on a graph in which critical edges have been split as a backward problem followed by a *forward correction* [11].□

Corollary 2.3 : *EPA can be decomposed into cascaded unidirectional problems.*

The decomposition of the Edge Placement Algorithm(EPA) [6] is feasible since its information flow paths are characterized by $({B_e}^+ \mid F_e){F_e}^*(B_e \mid \epsilon)$ (refer table 2). The number of segments in an *ifp* can at most be 3, hence the condition of lemma 2 is satisfied. A decomposition for EPA is presented in [6, 10].□

Corollary 2.4 : *It is possible to decompose MMRA if edge splitting is performed.*

The *ifp*'s of MMRA incorporate two distinct effects :

[5] unless edge splitting is performed.

(i) The mostly backward propagation of MRA which leads to *ifp* sections described by the regular expression $(B_e{}^+(F_e \mid \epsilon))^+ B_e{}^*$, and

(ii) The forward propagation of EPA which leads to *ifp* sections described by the regular expression $F_e{}^*(B_e \mid \epsilon)$.

Edge splitting truncates the *ifp* sections of the first kind to simply $B_e{}^+(F_e \mid \epsilon)$ in the transformed flow graph, and $B_e{}^+$ in the original graph, respectively (refer lemma 1). The resulting *ifp*'s are described by the regular expression $(B_e{}^+ \mid F_e)F_e{}^*(B_e \mid \epsilon)$ in both the transformed as well as the original graphs. Since this is the same as the regular expression for EPA in corollary 2.3, the decomposability of MMRA follows. The resulting decomposition is described in [12].□

This result in not surprising since EPA differs from MMRA only in the use of the edge placement technique (refer appendix) which has already been shown to achieve an effect equivalent to edge splitting in the original program flow graph (refer section 4.2.1).

6 Results and Conclusions

The results of analyzing a suit of scientific programs written in Fortran-77 are presented in table 3. The following observations can be made concerning these results :

(i) Very good correlation is seen between the width and the number of iterations for the problems which have bounded segments in the δ^- direction (*viz.* MRA).

(ii) For EPA and MMRA, the predicted and observed widths for backward iterations differed by very large margins. This can be attributed to the following :

  (a) Width is a static prediction. Many of the long *ifp*'s traced during width computation may not be realized during data flow analysis. This fact is well accepted by practitioners of data flow analysis.

  (b) The *ifp*'s of EPA and MMRA have unbounded δ^- segments. Hence, the computed widths are very large. This accentuates the effect mentioned in (a) above. A similar effect is observed in the case of unidirectional problems having unbounded δ^- segments. For example, the problem of available expressions has unbounded δ^- segments when solved using backward iterations. Its width is therefore the number of edges along the longest acyclic forward path in the program. In practice, information does not propagate so much.

(iii) For problems with unbounded segments in both directions (*viz.* EPA and MMRA), the method of alternating iterations scored over the backward direction of graph traversal by a large margin in a total of 11 cases. In 10 cases, it performed only marginally worse than backward traversal. On the whole, alternating iterations should be the favoured solution method for EPA and MMRA. When compared to MRA, the additional complexities of data flow in EPA and MMRA do not appear to make them very expensive in practice. (Note that MRA is also known to take upto 5 iterations in practice [25]).

(iv) Despite observation (ii), observation (iii) above demonstrates the applicability of the concept of an information flow path to non-singular data flows as well, thereby vindicating the use of *ifp*'s as the basis of a study of the complexity of data flow analysis.

Acknowledgements

The authors wish to thank the referees for their critical comments. Thanks are also due to Amitabha Sanyal, Ajit Diwan and Vikram Dhaneshwar for useful feedback from time to time, and to Viral Acharya and Moses Charikar for the implementation support.

References

[1] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers – Principles, Techniques, and Tools.* Addison-Wesley, 1986.

[2] F. E. Allen and J. Cocke. A program data flow analysis procedure. *Communications of ACM*, 19(3):137–147, 1977.

[3] S. Biswas, G. P. Bhattacharjee, and P. Dhar. A comparison of some algorithms for live variable analysis. *International Journal of Computer Mathematics*, 8:121–134, 1980.

[4] F. C. Chow. Minimizing register usage penalty at procedure calls. In *Proceedings of SIGPLAN'88 Symposium. on Compiler Construction*, pages 85–94, 1988. Also Published as *SIGPLAN Notices*, 23(7).

[5] D. M. Dhamdhere. An elimination algorithm for bidirectional data flow analysis. Technical report CSE-TR-88-31, Department of Computer Science and Engineering, The University of Connecticut, Storrs, 1988.

program	$\|N\|$	$\|E\|$	$\|X\|$	w'	MRA		EPA				MMRA			
					Backward		Backward		Alternate		Backward		Alternate	
					w	$\#i$	w	$\#i$	w_a	$\#i$	w	$\#i$	w_a	$\#i$
pgm1	36	62	168	1	2	4	13	4	5	5	26	6	20	5
pgm2	36	64	63	1	3	4	12	3	5	4	27	5	22	5
pgm3	48	68	111	1	3	3	23	10	7	5	39	10	31	5
pgm4	60	84	143	1	3	3	27	9	7	5	50	18	39	5
pgm5	45	50	241	1	2	2	37	3	5	3	41	2	9	3
pgm6	30	47	160	1	2	4	13	4	5	5	23	4	17	5
pgm7	34	46	140	1	4	3	22	5	7	5	27	4	15	5
pgm8	37	52	204	1	2	4	18	7	5	5	28	7	19	5
pgm9	61	84	224	1	3	4	29	3	5	4	47	4	29	4
pgm10	40	53	343	2	3	4	24	6	8	5	37	4	25	6
pgm11	46	78	207	1	2	4	14	5	5	5	26	7	20	6
pgm12	33	48	305	1	2	4	14	3	4	3	28	7	23	5
pgm13	34	48	54	1	2	4	24	3	3	3	33	4	11	4
pgm14	37	43	98	1	2	4	35	3	3	3	36	7	3	3
pgm15	62	83	365	1	2	4	30	3	5	4	41	4	21	5

$|N|$: Number of nodes
$|E|$: Number of edges
$|X|$: Number of expressions
w' : Width for unidirectional problems
w : Width for bidirectional problems
w_a : Width for alternating iterations
$\#i$: Number of iterations[a]

[a] For a width w, though $w+1$ iterations are sufficient for converging on a fixed point, an additional iteration is required to ascertain that the fixed point is reached. Thus, the actual bound on the number iterations for a round robin algorithm is $w+2$.

Table 3: Experimental results

[6] D. M. Dhamdhere. A fast algorithm for code movement optimization. *ACM SIGPLAN Notices*, 23(10):172–180, 1988.

[7] D. M. Dhamdhere. Register assignment using code placement techniques. *Computer Languages*, 13(2):75–93, 1988.

[8] D. M. Dhamdhere. A new algorithm for composite hoisting and strength reduction optimization. *International Journal of Computer Mathematics*, 27(1):1–14, 1989.

[9] D. M. Dhamdhere. Comments on practical adaptation of the global optimization algorithm by Morel & Renvoise. *ACM Transactions on Programming Languages and Systems*, 13(2):291–294, 1991.

[10] D. M. Dhamdhere and Harish Patil. An elimination algorithm for bidirectional data flow analysis using edge placement technique. *ACM Transactions on Programming Languages and Systems*, 15(2):312–336, 1993.

[11] D. M. Dhamdhere, B. K. Rosen, and F. K. Zadeck. How to analyze large programs efficiently and informatively. In *ACM SIGPLAN '92 Conference on Programming Language Design and Implementation*, 1992.

[12] Vikram Dhaneshwar. M. Tech. dissertation (stage 1). Department of Computer Science and Engineering, Indian Institute of Technology, Bombay, 1992.

[13] A. C. Fong, J. B. Kam, and J. D. Ullman. Applications of lattice algebra to loop optimization. In *Proceedings of the 2^{nd} Annual ACM Symposium on Principles of Programming Languages*, pages 1–9, 1975.

[14] S. Graham and M. Wegman. A fast and usually linear algorithm for global data flow analysis. *Journal of ACM*, 23(1):172–202, 1976.

[15] M. S. Hecht. *Flow Analysis of Computer Programs*. Elsevier North-Holland Inc., 1977.

[16] M. S. Hecht and J. D. Ullman. A simple algorithm for global data flow analysis problems. *SIAM Journal of Computing*, 4(4):519–532, 1977.

[17] S. M. Joshi and D. M. Dhamdhere. A composite algorithm for strength reduction and code move-

ment : part I. *International Journal of Computer Mathematics*, 11(1):21–44, 1982.

[18] S. M. Joshi and D. M. Dhamdhere. A composite algorithm for strength reduction and code movement : part II. *International Journal of Computer Mathematics*, 11(2):111–126, 1982.

[19] J. B. Kam and J. D. Ullman. Monotone data flow analysis frameworks. *Acta Informatica*, 7(3):305–318, 1977.

[20] Uday P. Khedker and D. M. Dhamdhere. Elimination algorithms for bidirectional data flow problems. Department of Computer Science and Engineering, Indian Institute of Technology, Bombay, 1992. (In preparation).

[21] Uday P. Khedker and D. M. Dhamdhere. A generalized theory of data flow analysis. Technical report TR-070-92, Department of Computer Science and Engineering, Indian Institute of Technology, Bombay, 1992.

[22] G. Kildall. A unified approach to global program optimization. In *Proceedings of the 1st Annual ACM Symposium on Principles of Programming Languages*, pages 194–206, 1973.

[23] J. Knoop, O. Ruthing, and B. Steffen. Lazy code motion. In *ACM SIGPLAN '92 Conference on Programming Language Design and Implementation*, 1992. Also Published as *SIGPLAN Notices*, 27(7).

[24] T. J. Marlowe and B. G. Ryder. Properties of data flow frameworks. *Acta Informatica*, 28:121–163, 1990.

[25] E. Morel and C. Renvoise. Global optimization by suppression of partial redundancies. *Communications of ACM*, 22(2):96–103, 1979.

[26] R. E. Tarjan. Fast algorithms for solving path problems. *Journal of ACM*, 28(3):594–614, 1981.

Appendix

Bidirectional Data Flow Algorithms

The Morel and Renvoise algorithm (MRA) [25] performs global program optimization by the suppression of partial redundancies. The basic approach is one of frequency reduction by *code hoisting*. Partial redundancy of an expression is used to motivate its hoisting. Since common subexpression elimination and loop invariant movement are special cases of redundancy elimination, the algorithm unifies several traditional optimizations within a single framework. Equations 1 and 2 represent the data flows of MRA.

Local property ANTLOC_i represents information concerning *upwards exposed* occurrence(s) of an expression e in node i of the program flow graph, while TRANSP_i reflects the absence of definition(s) of e's operands in the node. Partial redundancy is represented by the data flow property of partial availability $\text{PAVIN}_i/\text{PAVOUT}_i$ defined using the classical forward data flow problem of available expressions [1] with the confluence operator changed to Σ. Safety of hoisting is incorporated by considering only those expressions which are very busy at the entry/exit of a node. This is incorporated by the Π term of the PPOUT_i equation (eq. 2). A property INSERT_i identifies nodes where occurrences of expression e should be inserted. $\text{PPIN}_i \cdot \neg\text{ANTLOC}_i$ identifies occurrences which are redundant following the insertion.

In the Load-Store Insertion Algorithm [7], the problem of placing Load and Store instructions of a variable to characterize its live range for register assignment is modelled as redundancy elimination of the Load and Store instructions. Here, we consider the problem of the placement of Store instructions, which is a dual of MRA (eqs. 3 and 4). The local and global properties used in the equations are self-explanatory.

In the Modified MRA algorithm (MMRA) [9], certain simplifications are introduced in the MRA equations. An additional term, with Σ as the confluence, is added to the PPIN_i equation (eq. 5) to suppress redundant hoisting of an expression. Other terms in the equations are the same as in MRA.

The technique of *edge placement* [6] aims at total elimination of partial redundancies by placing an expression which is partially redundant in node i, but can not be safely placed in a predecessor node j, into a synthetic node placed along edge (j, i). This simplifies the data flow dependencies by eliminating the Π term of MRA's PPIN_i equation. The Edge Placement Algorithm (EPA) [6] uses the technique of edge placement, and also incorporates the suppression of redundant hoisting by adding a Σ term in the PPIN_i equation (eq. 7).

As mentioned in section 2.5, MMRA and EPA are non-singular data flows since they involve two distinct confluence operators.

The Composite Hoisting and Strength reduction Algorithm (CHSA) [17, 18] unifies strength reduction and redundancy elimination into the same framework. *Additive* computations are used to update the value of a high strength expression following an update of the induction variable's value. Recomputations of the high strength expression are placed at other strategic places in the program. The $\text{NOCOMIN}_i/\text{NOCOMOUT}_i$ problem is used to limit the number of additive computations along a path following a recomputation of the high strength expression. CHSA is the only known singular data flow problem to possess the node and edge flow functions in both directions — forwards as well as backwards.

1. Morel Renvoise Algorithm (MRA) [25] :

$$\mathrm{PPIN}_i = \mathrm{CONST}_i \cdot (\mathrm{ANTLOC}_i + \mathrm{TRANSP}_i \cdot \mathrm{PPOUT}_i) \cdot \prod_{p \,\in\, pred(i)} (\mathrm{AVOUT}_p + \mathrm{PPOUT}_p) \tag{1}$$

$$\mathrm{PPOUT}_i = \prod_{s \,\in\, succ(i)} (\mathrm{PPIN}_s) \tag{2}$$

2. Load Store Insertion Algorithm (LSIA) [7] :

$$\mathrm{SPPIN}_i = \prod_{p \,\in\, pred(i)} (\mathrm{SPPOUT}_p) \tag{3}$$

$$\mathrm{SPPOUT}_i = \mathrm{DPANTOUT}_i \cdot (\mathrm{DCOMP}_i + \mathrm{DTRANSP}_i \cdot \mathrm{SPPIN}_i) \cdot \prod_{s \,\in\, succ(i)} (\mathrm{DANTIN}_s + \mathrm{SPPIN}_s) \tag{4}$$

3. Modified MRA (MMRA) [9] :

$$\mathrm{PPIN}_i = \mathrm{PAVIN}_i \cdot (\mathrm{ANTLOC}_i + \mathrm{TRANSP}_i \cdot \mathrm{PPOUT}_i) \cdot \prod_{p \,\in\, pred(i)} (\mathrm{AVOUT}_p + \mathrm{PPOUT}_p) \cdot \sum_{p \,\in\, pred(i)} (\mathrm{PPIN}_p \cdot \neg\mathrm{ANTLOC}_p + \mathrm{AVOUT}_p) \tag{5}$$

$$\mathrm{PPOUT}_i = \prod_{s \,\in\, succ(i)} (\mathrm{PPIN}_s) \tag{6}$$

4. Edge Placement Algorithm (EPA) [6] :

$$\mathrm{PPIN}_i = \mathrm{PAVIN}_i \cdot (\mathrm{ANTLOC}_i + \mathrm{TRANSP}_i \cdot \mathrm{PPOUT}_i) \cdot \sum_{p \,\in\, pred(i)} (\mathrm{PPIN}_p \cdot \neg\mathrm{ANTLOC}_p + \mathrm{AVOUT}_p) \tag{7}$$

$$\mathrm{PPOUT}_i = \prod_{s \,\in\, succ(i)} (\mathrm{PPIN}_s) \tag{8}$$

5. Composite Hoisting and Strength reduction Algorithm (CHSA) [17, 18] :

$$\mathrm{NOCOMIN}_i = \mathrm{CONSTA}_i \cdot (\sum_{p \,\in\, pred(i)} \mathrm{NOCOMOUT}_p + \mathrm{CONSTB}_i \cdot \mathrm{NOCOMOUT}_i) \tag{9}$$

$$\mathrm{NOCOMOUT}_i = \mathrm{CONSTC}_i + \mathrm{CONSTD}_i \cdot (\sum_{s \,\in\, succ(i)} \mathrm{NOCOMIN}_s + \mathrm{CONSTE}_i \cdot \mathrm{NOCOMIN}_i) \tag{10}$$

Figure 6: Data flow equations of the problems referred in this paper