Data Flow Reversal I

Computational Complexity

Uwe Naumann

Informatik 12:
Software and Tools for Computational Engineering (STCE)

RWTH Aachen University

Contents

Outline

Objective and Learning Outcomes

Data Flow Reversal

Computational Complexity of Data Flow Reversal

- DAG Reversal
- Vertex Cover
- Proof of NP-Completeness (Step 1)
- Proof of NP-Completeness (Step 2)

Summary and Next Steps

Software and Tools for Computational Engineering
RWTH AACHEN UNIVERSITY

Objective

- Formulation of Data Flow Reversal problem as DAG REVERSAL and proof of NP-completeness

Learning Outcomes

- You will understand
  - DAG REVERSAL
  - MINIMUM MEMORY DATA FLOW REVERSAL
- You will be able to
  - reproduce the proof of NP completeness.

Outline

Objective and Learning Outcomes

Data Flow Reversal

Computational Complexity of Data Flow Reversal
- DAG REVERSAL
- VERTEX COVER
- Proof of NP-Completeness (Step 1)
- Proof of NP-Completeness (Step 2)

Summary and Next Steps

We consider implementations of multivariate vector functions

$$F : \boldsymbol{R}^n \rightarrow \boldsymbol{R}^m : \mathbf{y} = F(\mathbf{x})$$

as (numerical computer) programs.

Such programs decompose into sequences of $q = p + m$ elemental functions φ_j evaluated as a single assignment code[1]

$$v_j = \varphi_j(v_k)_{k \prec j} \quad \text{for } j = n, \ldots, n + q - 1$$

and where $v_i = x_i$ for $i = 0, \ldots, n - 1$, w.l.o.g, $y_k = v_{n+p+k}$ for $k = 0, \ldots, m - 1$ and $k \prec j$ if v_k is an argument of φ_j.

A directed acyclic graph (DAG) $G = (V = X \cup Z \cup Y, E)$ is induced such that $|X| = n$, $|Z| = p$ and $|Y| = m$.

[1] Variables are written once.

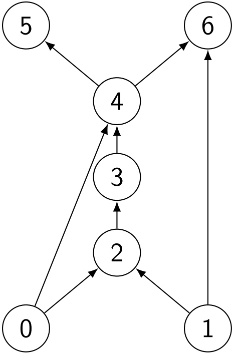
Data Flow Reversal

Example

$$t = x_0 \cdot \sin(x_0 \cdot x_1)$$
$$x_0 = \cos(t)$$
$$x_1 = t/x_1$$

$$v_0 = x_0$$
$$v_1 = x_1$$
$$v_2 = v_0 \cdot v_1$$
$$v_3 = \sin(v_2)$$
$$v_4 = v_0 \cdot v_3$$
$$v_5 = \cos(v_4)$$
$$v_6 = v_4/v_1$$
$$x_0 = v_5$$
$$x_1 = v_6$$

$G = (V, E)$:



Wanted: $v_6, v_5, v_4, v_3, v_2, v_1, v_0$

A data flow reversal recovers the results of the elemental functions evaluated by a program in reverse order. Relevant applications include debugging and adjoint algorithmic differentiation.

The data flow reversal problem aims for recovery of the results of the elemental functions evaluated by a program in reverse order such that for a given upper bound $\overline{MEM}$ on the available persistent memory the computational cost is minimized.

The computational cost ($COST$) is defined as the sum of the number of elemental function evaluations (OPS) to be performed in addition to a single evaluation of the program (requiring $|Z \cup Y|$ elemental function evaluations) and the number of write accesses to persistent memory.

We assume vanishing cost for the strictly sequential read accesses to memory ($\rightarrow$ prefetching).

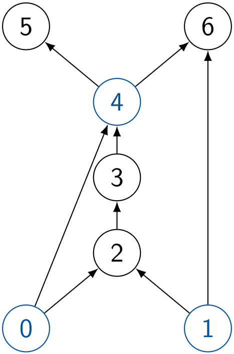
W.l.o.g, we assume only persistently stored values to be available for data flow reversal, i.e, even v_{n+q-1} is not automatically available following the initial evaluation of the program as the data flow reversal might not follow immediately.

Borderline cases are

- store-all; Results of all elemental functions are pushed onto a stack. Recovery implies reversal. $MEM = |V|$ is maximized while $OPS = |Z \cup Y|$ is minimized and $COST = MEM$, e.g, requiring $\overline{MEM} \geq 7$ in the previous example.
- recompute-all: Results of all elemental functions are recomputed in reverse order as functions of the persistent inputs to the program, respectively. $MEM = |X|$ is minimized while $OPS = O(|Z \cup Y|^2)$ is maximized and $COST = MEM + OPS$. e.g, requiring $\overline{MEM} = 2$ in the previous example.

A data flow reversal needs to recompute nonpersistent values from persistent values at locally quadratic (in the length of the longest path connecting the corresponding vertices in the DAG) OPS, e.g, for $\overline{MEM} = 2$ we get $OPS = 4 + 4 + 3 + 2 + 1 = 14$.

Let $\overline{MEM} = 3$ allowing for persistent storage of one value in addition to the two inputs, e.g, v_0, v_1, v_4.



- compute all and store $v_0, v_1, v_4 \Rightarrow COST = 3$
- compute v_6 from v_1 and $v_4 \Rightarrow COST = 4$
- compute v_5 from $v_4 \Rightarrow COST = 5$
- v_4 is available
- EITHER:
  - compute v_3 from v_0 and $v_1 \Rightarrow COST = 7$
  - compute v_2 from v_0 and $v_1 \Rightarrow COST = 8$
  - v_1 and v_0 are available
- OR:
  - compute v_2 from v_0 and v_1 and store it $\Rightarrow COST = 7$
  - compute v_3 from $v_2 \Rightarrow COST = 8$
  - v_2, v_1 and v_0 are available

Outline

The data flow reversal problem is also known as DAG Reversal:

Given a DAG and two integers $C, \overline{MEM} > 0$, is there a data flow reversal that uses at most $MEM \leq \overline{MEM}$ memory units and yields a computational cost of $COST \leq C$?

DAG Reversal is NP-complete.

- U. Naumann: *DAG Reversal is NP-Complete*. Journal of Discrete Algorithms, Elsevier 2010.

Part of the proof is by reduction from Vertex Cover.

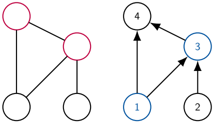
VERTEX COVER problem: Given a graph $G = (V, E)$ is there a subset $W \subseteq V$ of size $\omega \leq \Omega$, s.t. each edge in E is incident with at least one vertex from W?

VERTEX COVER is NP-complete.

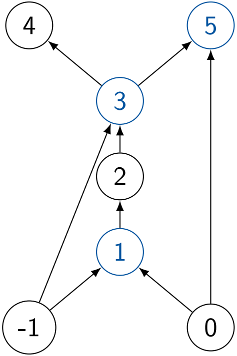
Proof: [Garey/Johnson (1979)]

VERTEX COVER for DAGs is NP-complete.

Proof: Enumerate vertices and make edges directed s.t. $(i, j) \in E \Leftrightarrow i < j$.



q.e.d.



The proof proceeds in two stages.

1. We show that asking for minimal *MEM* while keeping minimal $COST = |V|$ is NP-complete.
2. We show that an algorithm for DAG REVERSAL solves the above efficiently. Hence, DAG REVERSAL cannot be easier than 1.

MINIMUM MEMORY DATA FLOW REVERSAL (MMDFR):

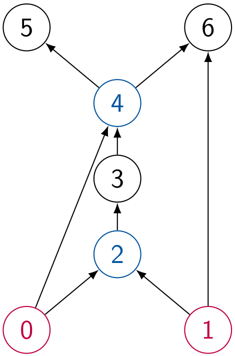
Given a DAG $G = (V, E)$ and an integer $\overline{MEM} > 0$, is there a data flow reversal with $COST = |V|$ and $MEM \leq \overline{MEM}$?

MMDFR is NP-complete.

An algorithm for this decision version of MMDFR implies an algorithm for the corresponding optimization version.

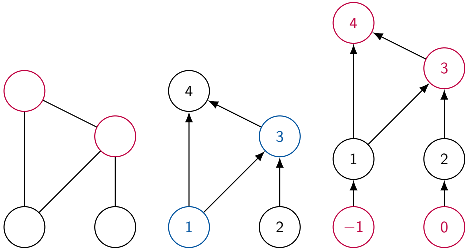
A maximum of $|V|$ solutions of the decision problem solves the optimization problem.

Example



- $\overline{MEM} = 7 \rightarrow$ store-all
- $\overline{MEM} = 6 \rightarrow$ (e.g.) recompute v_3
- $\overline{MEM} = 5 \rightarrow$ (e.g.) recompute v_2 and v_5
- $\overline{MEM} = 4 \rightarrow$ (e.g.) recompute v_3, v_5, and v_6
- $\overline{MEM} \leq 3 \rightarrow$ no solution

(Polynomial) Reduction from VERTEX COVER to MMDFR is by enumeration of vertices ($\Rightarrow$ DAG) and horizontal split of minimal vertices ($\Rightarrow G'$).



We claim that there is a solution for MMDFR on G' with $M\bar{E}M = \Omega + |X|$ if and only if there is a solution for VERTEX COVER with Ω on G.

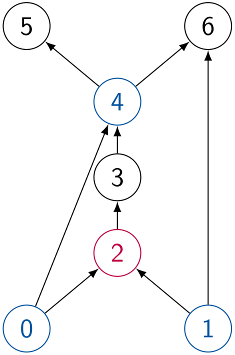
Obviously, $|V|$ is a sharp lower bound for $COST$ as the recovery of each value (either by persistent storage during the initial evaluation of the program or by recomputation using at least a single elemental function evaluation) has at least unit cost; Store-all reaches the bound.

"$\Leftarrow$" Consider a solution for VERTEX COVER ($|W| \leq \Omega$). Each edge is incident with at least one vertex in W. Hence, predecessors of vertices from $V \setminus W$ are in W. Values corresponding to vertices in W can be stored persistently at unit cost and they can be recovered for free. Nonpersistent values corresponding to vertices in $V \setminus W$ can be recomputed at unit cost. Values of inputs need to be stored persistently in any case yielding a data flow reversal with $COST = |V|$ and $MEM \leq \bar{MEM} = \Omega + |X|$.

"$\Rightarrow$" Consider a solution for MMDFR ($MEM \leq \bar{MEM} = \Omega + |X|$). Nonpersistent values need to be recomputed at unit cost. Hence, their predecessors of the corresponding vertices need to be stored. The set W of vertices corresponding to persistent values is a vertex cover in G with $|W| \leq \Omega$.

q.e.d.

Reuse of persistent memory implies recomputation and thus breaks the fixed *COST* assumption of MMDFR, e.g,



- compute all and store $v_0, v_1, v_4 \Rightarrow COST = 3$
- compute v_6 from v_1 and $v_4 \Rightarrow COST = 4$
- compute v_5 from $v_4 \Rightarrow COST = 5$
- v_4 is available
- compute v_2 from v_0 and v_1 and overwrite v_4 $\Rightarrow COST = 7$
- compute v_3 from $v_2 \Rightarrow COST = 8 > 7$
- v_2, v_1 and v_0 are available

An algorithm for DAG REVERSAL can be used to solve MMDFR as follows:

For $\overline{MEM} = |V|$ there is a solution of DAG REVERSAL with $COST = |V|$ (e.g, store-all).

Decrease $\overline{MEM}$ by one at a time for as long as there is a solution with $COST = |V|$. The smallest $\overline{MEM}$ for which such a solution exists is the solution of the minimization version of MMDFR.

Hence, we need to solve at most $|V|$ instances of DAG REVERSAL to solve MMDFR.

MMDFR cannot be intractable while DAG REVERSAL is not (or P=NP and all NP-complete problems become tractable).

q.e.d.

Outline

Summary

- Formulation of Data Flow Reversal problem as DAG REVERSAL and proof of NP-completeness including
  - MINIMUM MEMORY DATA FLOW REVERSAL
  - Reduction from VERTEX COVER

Next Steps

- Reproduce the proof of NP completeness.
- Continue the course to find out more ...