Simulated Annealing for Optimal Pivot Selection in Jacobian Accumulation

Uwe Naumann[1] and Peter Gottschling[2]

[1] Mathematics and Computer Science Division, Argonne National Laboratory
naumann@mcs.anl.gov,
[2] Department of Computer Science, University of Hertfordshire, College Lane, Hatfield, UK
P.1.Gottschling@herts.ac.uk

Abstract. We report on new results in logarithmic simulated annealing applied to the optimal Jacobian accumulation problem. This is a continuation of work that was presented at the SAGA'01 conference in Berlin, Germany [16]. We discuss the optimal edge elimination problem in linearized computational graphs [15] in the context of linear algebra. We introduce row and column pivoting on the extended Jacobian as analogs to front and back edge elimination in linearized computational graphs. Neighborhood relations for simulated annealing are defined on a metagraph that is derived from the computational graph. All prerequisites for logarithmic simulated annealing are fulfilled for dyadic pivoting, which is equivalent to vertex elimination in linearized computational graphs [7]. For row and column pivoting we cannot yet give a proof that the corresponding elimination sequences are polynomial in size. In practice, however, the likelihood for an exponential elimination sequence to occur is negligible. Numerical results are presented for algorithms based on both homogeneous and inhomogeneous Markov chains for all pivoting techniques. The superiority of row and column pivoting over dyadic pivoting can be observed when applying these techniques to Roe's numerical flux [17].

Keywords. Jacobian matrices, dyadic pivoting, row and column pivoting, (logarithmic) simulated annealing

1 Background

Derivatives of vector functions that represent mathematical models of scientific, engineering, or economic problems play an increasingly important role in modern numerical computing. They can be regarded as the enabling key factor allowing for a transition from the pure simulation of the real world process to the optimization of some specific objective with respect to a set of model parameters. For a given computer program that implements an underlying numerical model automatic differentiation (AD) [3–6] provides a set of techniques for transforming the program into one that computes not only the function value for a set of inputs but also the corresponding first and higher derivatives.

A large portion of the ongoing research in this field is aimed towards the improvement of the efficiency of the derivative code that is generated. Successful methods are often built on a combination of classical compiler algorithms and the exploitation of mathematical properties of the code. In this paper we consider the problem of minimizing the number of floating point operations performed by those parts of the derivative code that do not depend on the flow of control, for example, *basic blocks* [1]. Approximate solutions of the corresponding combinatorial optimization problem are obtained by exploiting the associativity of the *chain rule*. Before stating the problem formally as an elimination method in a system of linear equations we present a brief introduction to the principles of AD. We provide background information that we consider essential for the further understanding of the material covered by subsequent sections.

Consider the following Fortran subroutine:

```
SUBROUTINE EX (n,x)
  INTEGER :: n
  REAL, DIMENSION(n), INTENT(INOUT) :: x

  DO i=1,n-1
    x(i)=x(i)*sin(x(i)*x(i+1))
    x(i+1)=x(i)*x(i+1)
    x(i)=cos(x(i))
  END DO
END SUBROUTINE EX
```

The final value of $\mathbf{x} = (\mathtt{x(1)}, \ldots, \mathtt{x(n)})^T$ is computed from its start value by `n-1` executions of the statements that form the body of the loop. The code implements a vector function $F : I\!R^n \to I\!R^n$ of the form $\mathbf{y} = F(\mathbf{x})$. After running the example code the values of $\mathbf{y}$ happen to be stored in $\mathbf{x}$ in our case. An example similar to the body of the loop was used in [16] to introduce vertex elimination techniques in linearized computational graphs and the resulting combinatorial optimization problem.

Suppose that one is interested in the Jacobian matrix F' of F, that is the matrix of the partial derivatives of the outputs with respect to the inputs. It is defined as

$$F' = F'(\mathbf{x}) = \left(\frac{\partial y_j}{\partial x_i}\right)_{j,i=1,\ldots,n}$$

where x_i denotes the input value of `x(i)` and y_j the corresponding output value. The *forward mode* of AD [6][Chapter 3] transforms the code semantically such that in addition to the function value the product of the Jacobian with some direction $\dot{\mathbf{x}}$ in the input space is computed. The result of this transformation is an implementation of the function

$$(\mathbf{y}, \dot{\mathbf{y}}) = \dot{F}(\mathbf{x}, \dot{\mathbf{x}}) \equiv (F(\mathbf{x}), F'(\mathbf{x}) \cdot \dot{\mathbf{x}}) \quad .$$

Given such a program the Jacobian itself can be computed by letting the directions $\dot{\mathbf{x}}$ range over the Cartesian basis vectors in $I\!R^n$ since, obviously, $F' = F' \cdot I_n$,

where I_n denotes the identity matrix in $\mathbb{R}^n$. How is this transformation performed?

The statements of the loop body can be decomposed into a sequence of scalar assignments of the results of all elemental arithmetic operators and intrinsic functions to locally unique intermediate variables `z1, z2, z3`. This *code list* can be augmented by statements that compute for all elemental assignments the local partial derivatives `a`, ..., `h` of the left-hand side with respect to the arguments on the right-hand side. [1]

```
DO i=1,n-1
  a=x(i+1); b=x(i)
    z1=x(i)*x(i+1)
  c=cos(z1)
    z2=sin(z1)
  d=z2; e=x(i)
    z3=x(i)*z2
  f=x(i+1); g=z3
    x(i+1)=z3*x(i+1)
  h=-sin(z3)
    x(i)=cos(z3)
END DO
```

The resulting code is often referred to as the *linearized code list*. The next transformation step associates directional derivative components with all input, intermediate, and output variables, and it combines the well-know differentiation rules with the chain rule to generate a *tangent-linear* version of the code.

```
SUBROUTINE D_EX (n,x,l,d_x)
  INTEGER :: n,l
  REAL, DIMENSION(n), INTENT(INOUT) :: x
  REAL, DIMENSION(n,l), INTENT(INOUT) :: d_x
  REAL :: z1,z2,z3,a,b,c,d,e,f,g,h
  REAL, DIMENSION(l) :: d_z1, d_z2, d_z3

  DO i=1,n-1
    a=x(i+1); b=x(i)
    d_z1=a*d_x(i,:)+b*d_x(i+1,:)
    z1=x(i)*x(i+1)
    c=cos(z1)
    d_z2=c*d_z1
    z2=sin(z1)
    d=z2; e=x(i)
    d_z3=d*d_x(i,:)+e*d_z2
```

[1] Software tools for AD usually avoid the generation of trivial assignments of the form `a=x(i+1)`. We decided to leave them in the code to facilitate an easier understanding of the concepts that forward mode AD is built on.

```
    z3=x(i)*z2
    f=x(i+1); g=z3
    d_x(i+1,:)=f*d_z3+g*d_x(i+1,:)
    x(i+1)=z3*x(i+1)
    h=-sin(z3)
    d_x(i,:)=h*d_z3
    x(i)=cos(z3)
  END DO

END SUBROUTINE D_EX
```

The directional derivative components are denoted by the `d_` prefix. They are vectors in $\mathbb{R}^n$ as in the *forward vector mode* of AD [6][Chapter 3]. All derivative propagation statements must be interpreted as vector operations of the form $\dot{\mathbf{z}} = a \cdot \dot{\mathbf{x}} + b \cdot \dot{\mathbf{y}}$ where $\dot{\mathbf{z}}, \dot{\mathbf{x}}, \dot{\mathbf{y}} \in \mathbb{R}^n$ and $a, b \in \mathbb{R}$. The above code uses Fortran array operation syntax. Our measure of complexity of the tangent-linear code is the number of scalar floating-point multiplications performed in addition to the arithmetic operations that are required to built the linearized code list. Hence, we count only multiplications that are performed during the propagation of the directional derivatives. It is straight-forward to verify that the corresponding value for our example is equal to $8n(n-1)$. The derivative propagation part of a single execution of the loop body performs $8n$ multiplications, n per local partial derivative `a`, ..., `h`. The number of iterations is $n-1$.

Preaccumulation techniques are based on the observation that the local partial derivatives can be combined according to the chain rule [7] which may result in a decreased number of potential factors in the directional derivative propagation part of the corresponding tangent-linear code. This transformation is equivalent to a transformation of the linearized computational graph into a bipartite form as shown, for example, in [16]. For our example the preaccumulation of the local Jacobian reduces the number of multiplications in the directional derivative propagation to four. [2]

```
SUBROUTINE D_EX_BB (n,x,l,d_x)

  INTEGER :: n,l
  REAL, DIMENSION(n), INTENT(INOUT) :: x
  REAL, DIMENSION(n,l), INTENT(INOUT) :: d_x
  REAL :: z1,z2,z3,a,b,c,d,e,f,g,h,ec,eca,ecb,j11,j12,j21,j22
  REAL, DIMENSION(l) :: d_tmp

  DO i=1,n-1
    a=x(i+1); b=x(i)
    z1=x(i)*x(i+1)
```

[2] There are two inputs and two outputs, and the local Jacobian is dense. By inspection, we know that `x(i)` and `x(i+1)` are always distinct. In general, this information needs to be provided by alias / array section analysis [1].

```
    c=cos(z1)
    z2=sin(z1)
    d=z2; e=x(i)
    z3=x(i)*z2
    f=x(i+1); g=z3
    x(i+1)=z3*x(i+1)
    h=-sin(z3)
    x(i)=cos(z3)
    ec=e*c
    eca=d+ec*a; ecb=ec*b
    j11=h*eca
    j12=h*ecb
    j21=f*eca
    j22=g+f*ecb
    d_tmp=j11*d_x(i,:)+j12*d_x(i+1,:)
    d_x(i+1,:)=j21*d_x(i,:)+j22*d_x(i+1,:)
    d_x(i,:)=d_tmp
  END DO

END SUBROUTINE D_EX_BB
```

The preaccumulation of the Jacobian entries `j11`, `j12`, `j21`, `j22` can be performed using vertex elimination techniques at a cost of seven multiplications as shown in [16]. The corresponding code computes intermediate local partial derivatives `ec`,`eca`, and `ecb`. The following derivative propagation requires $4n$ multiplications. [3] Consequently, the number of multiplications performed by the tangent-linear code that uses preaccumulation is $(7+4n)(n-1)$. For large n this represents an improvement by a factor that is close to two. Moreover, the preaccumulation itself can be done in different ways by exploiting the associativity of the chain rule. The improvements are not very large for small codes. However, they can become more signifcant if large parts of the program are subject to preaccumulation. An example is discussed in Section 5.2.

The paper is structured as follows: In Section 2 we state the preaccumulation problem as a combinatorial optimization problem on the tangent-linear system of equations. The elimination techniques that are used to solve this system are introduced in Section 3 together with an example. The elimination problem is discussed in the context of simulated annealing in Section 4. Numerical test are presented in Section 5. The paper resumes with conclusions drawn in Section 6.

2 Problem Description

The optimal preaccumulation of local Jacobians of basic blocks in tangent-linear and adjoint models of numerical simulation programs is a highly desirable feature

[3] The assignment to `d_tmp` is required to assure correctness of the derivative code. This is due to `x` being both input and output of the basic block that represents the loop body.

of modern software tools for automatic differentiation [6]. Hard combinatorial optimization problems must be solved to determine near-optimal elimination sequences on the underlying linearized computational graphs [14] or, equivalently, on the extended Jacobian matrix (see below). Local heuristics have been developed to obtain good approximations to the solutions of these problems [2, 7, 12]. If the resulting derivative code is to be used extensively over a long period of time, then more expensive techniques, such as simulated annealing, can be employed to, possibly, achieve further improvements for crucial parts of the computation, for example, for CFD-kernels [17].

We consider numerical simulation programs that implement non-linear vector functions

$$F : I\!R^n \supseteq D \to I\!R^m : \mathbf{x} \mapsto \mathbf{y} = F(\mathbf{x}) \quad .$$

The Jacobian matrix of F is denoted by

$$F' = F'(\mathbf{x}_0) = \left(\frac{\partial y_i}{\partial x_j}(\mathbf{x}_0) \right)_{j=1,\ldots,n}^{i=1,\ldots,m} \quad .$$

AD provides a set of techniques for generating derivative code for F, such that, for example, F' can be computed with machine accuracy.

The code list of F ensures that each assignment creates a different variable name. A given input $\mathbf{x}$ determines the flow of control uniquely and the corresponding code list becomes a sequences of, in general, non-linear assignments

$$v_j = \varphi_j(v_i)_{i \prec j} \quad ,$$

where $j = 1, \ldots, q$ and $q = p + m$. The number of intermediate variables is denoted by p. Following the notation in [6], the set of arguments of φ_j is denoted as $\{v_i : i \prec j\}$, that is $i \prec j$ if v_i is an argument of φ_j. In AD the elemental functions φ_j are assumed to be the elementary arithmetic operators and intrinsic functions provided by the programming language that is used to implement F. Furthermore, we set $x_i \equiv v_{i-n}$, $i = 1, \ldots, n$, $z_k \equiv v_k$, $k = 1, \ldots, p$, and $y_j \equiv v_{p+j}$, $j = 1, \ldots, m$. For clarity of the following argument we assume that the dependent variables are mutually independent. Obviously, this must be the case for all independent variables too.

Source transformation tools for AD can be used to generate tangent-linear and adjoint models automatically for a given program for F. As before, a given argument $\mathbf{x}$ of F determines the flow of control uniquely. Tangent-linear models associate directional derivatives $\dot{v}_i$ with every code list variable for $i = 1-n, \ldots, q$ and they compute

$$\dot{v}_j = \sum_{i \prec j} c_{j,i} \cdot \dot{v}_i \quad ,$$

for $j = 1, \ldots, q$ and where

$$c_{j,i} \equiv \frac{\partial \varphi_j(v_i)_{i \prec j}}{\partial v_i} \quad .$$

For the purpose of this paper it is sufficient to introduce the formalism for tangent-linear models. Analogous results hold for adjoint models. Both tangent-linear and adjoint models rely on the existence of jointly continuous partial derivatives for all elemental functions φ_j, $j = 1, \ldots, q$, on open neighborhoods $\mathcal{D}_j \subset \mathbb{R}^{|\{i:i \prec j\}|}$ of their respective domains.

A tangent-linear model represents a system of linear equations that can be written in matrix form as follows.

$$\begin{pmatrix} \dot{\mathbf{z}} \\ \dot{\mathbf{y}} \end{pmatrix} = C \cdot \begin{pmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{z}} \end{pmatrix} \quad , \tag{1}$$

where $C \in \mathbb{R}^{q \times (n+p)}$ is the extended Jacobian that is defined as

$$C = (c_{j,i})_{i=1-n,\ldots,p}^{j=1,\ldots,q} \tag{2}$$

with local partial derivatives $c_{j,i}$. The computation of $\dot{\mathbf{y}} = F' \cdot \dot{\mathbf{x}}$ can be interpreted as the solution of Equation (1) for $\dot{\mathbf{y}}$ in terms of $\dot{\mathbf{x}}$. To do so we consider elimination techniques on C. Following the notation in [6] the extended Jacobian C can be partitioned as follows.

$$C = \begin{pmatrix} B & L \\ R & T \end{pmatrix} \quad , \tag{3}$$

where $B \in \mathbb{R}^{p \times n}$, $L \in \mathbb{R}^{p \times p}$, $R \in \mathbb{R}^{m \times n}$, and $T \in \mathbb{R}^{m \times p}$. Since the structure of C is induced by a code list the matrix $L = (l_{j,i})_{j,i=1,\ldots,p}$ must be strictly lower diagonal, that is $l_{j,i} = 0$ if $i \geq j$. Solving Equation (1) for $\dot{\mathbf{y}}$ in terms of $\dot{\mathbf{x}}$ is regarded as the elimination of all non-zero elements in B, L, and T.

3 Elimination Techniques

3.1 Dyadic Pivoting

Dyadic pivoting (DP) is equivalent to vertex elimination in linearized computational graphs as introduced in [7]. In C some $j \in \{1, \ldots, p\}$ is picked and the outer product of the jth row with the jth column is added to the corresponding submatrix of C that is spanned by the $(j+1)$th, $\ldots$, qth rows and by the $(1-n)$th, $\ldots$, $(j-1)$th columns. All elements in both the jth row and the jth column are set to zero. This simultaneous row and column elimination is expressed by the following equation.

$$C_j = C + Ce'_j e_j^T C - Ce'_j - Ce_j^T \quad , \tag{4}$$

where e_j is the jth Cartesian basis vector in $\mathbb{R}^q$ and e'_j is the $(n+j)$th Cartesian basis vector in $\mathbb{R}^{n+p}$. The proof for the correctness of Equation (4) can be found in [7]. It comes as an immediate consequence of the chain rule.

The problem of finding an order of the pivots that minimizes the fill-in has been shown to be NP-complete in [9]. The closely related optimal Jacobian

accumulation problem (OJA) for vertex elimination in linearized computational graphs is also conjectured to be intractable. If this conjecture is true, then the same applies to the dyadic pivoting problem in extended Jacobians.
The two special orderings that pick j according to $j = 1, \ldots, p$ and $j = p, \ldots, 1$ are essentially equivalent to the sparse forward and reverse mode of AD, respectively, as pointed out in [6].

Logarithmic simulated annealing has been applied successfully to the vertex elimination problem in linearized computational graphs in [16]. The two pivoting techniques to be introduced next represent refinements of dyadic pivoting. Their use in simulated annealing algorithms and the definition of appropriate neighborhood relations are discussed in Section 4.

3.2 Row Pivoting

Row pivoting is equivalent to front edge elimination in linearized computational graphs as introduced in [14]. The name is motivated by the choice of a particular element inside a given row. In C a pair of pivots (j, i), such that $j \in \{1, \ldots, p\}$, $i \in \{1 - n, \ldots, p\}$, $j > i$, is picked, that is the ith element in the jth row of C, and the product of $c_{j,i}$ and the jth column is added to the ith column of C. The entry $c_{j,i}$ is set to zero. If the elimination of $c_{j,i}$ results in the jth row becoming equal to the zero vector in $\mathbb{R}^{n+p}$, then the entire jth column is also set to zero. This procedure is expressed by the following equation.

$$C_{(j,i)} = C + e_j^T C e_i' C e_j' - e_j^T C e_i' \quad . \tag{5}$$

In addition,

$$C_{(j,i)} = C_{(j,i)} - C_{(j,i)} e_j' \quad \text{if} \quad e_j^T C_{(j,i)} = \mathbf{0} \quad .$$

The latter is required to avoid unnecessary floating-point operations during the elimination procedure. Otherwise, the solution of Equation (1) for $\dot{\mathbf{y}}$ would also give a solution for $\dot{\mathbf{z}}$ in terms of $\dot{\mathbf{x}}$. Although this might be useful derivative information in some cases we are not interested in its computation in the current context. The proof for the correctness of Equation (5) is an immediate consequence of a front edge elimination sequence being a special *face elimination* sequence as described in detail in [15]. The proof is based on the ideas presented in [7].

3.3 Column Pivoting

In column pivoting we choose an element inside a given column. This technique is equivalent to back edge elimination in linearized computational graphs [14]. In C a pair of pivots (i, j) such that $i \in \{1, \ldots, p\}$, $j \in \{1, \ldots, q\}$, and $j > i$, is picked, that is the jth element in the ith column of C, and the product of $c_{j,i}$ and the ith row is added to the jth row of C. The entry $c_{j,i}$ is set to zero. If the elimination of $c_{j,i}$ results in the ith column becoming equal to the zero vector in $\mathbb{R}^q$, then the entire ith row is also set to zero. Formally,

$$C_{(i,j)} = C + e_j^T C e_i' e_i^T C - e_j^T C e_i' \quad , \tag{6}$$

where the notation is the same as in Section 3.2. Again,

$$C_{(i,j)} = C_{(i,j)} - e_i^T C_{(i,j)} \quad \text{if} \;\; C_{(i,j)} e_i' = \mathbf{0} \quad .$$

Remark 1 *Dyadic pivoting is a special case of both row and column pivoting. Picking j as a dyadic pivot is equivalent to choosing row pivots (j,i) such that $i = 1-n, \ldots, j-1$. Similarly, the elimination resulting from making j the pivot can be performed by choosing column pivots (j,i) such that $i = j+1, \ldots, q$.*

Notation 1 *Mixed sequences of row and column pivots are denoted by RCP. The extended Jacobian that is obtained by applying some RCP $[(i_1,j_1),\ldots,(i_k,j_k)]$ to C is denoted by $C_{[(i_1,j_1),\ldots,(i_k,j_k)]}$. An RC-pivot (i,j) is a row pivot if $i > j$, and it is a column pivot if $i < j$. The application of some dyadic pivot sequence (DP) $[i_1,\ldots,i_k]$ to C is denoted by $C_[i_1,\ldots,i_k]$. RCP's and DP's can be mixed. For example, $C_{[(i,j),i]}$, where $i < j$ is the extended Jacobian that is derived from C by choosing (i,j) as a column pivot followed by the elimination resulting from making i the dyadic pivot.*

Remark 2 *The elemental arithmetic operations in Equation (4)–(6) are fused multiply adds (fma) of the form $c_{k,i} = c_{k,i} + c_{k,j}c_{j,i}$. Our objective is to compute a sequence of pivots that minimizes the number of fma's required to compute the Jacobian starting from a given extended Jacobian. For DP's this problem is equivalent to the optimal vertex elimination problem in linearized computational graphs [7]. When considering RCP's one is solving the optimal edge elimination problem in linearized computational graphs [14].*

3.4 Example

We consider the extended Jacobian C of the lion graph [14] representing the linearized computational graph of a vector function $F : \mathbb{R}^2 \to \mathbb{R}^4$. C is given as

$$C = \begin{pmatrix} c_{1,-1} & c_{1,0} & 0 & 0 \\ 0 & 0 & c_{2,1} & 0 \\ 0 & 0 & 0 & c_{3,2} \\ 0 & 0 & 0 & c_{4,2} \\ 0 & 0 & 0 & c_{5,2} \\ 0 & 0 & c_{6,1} & c_{6,2} \end{pmatrix} \quad .$$

The Jacobian F' can be accumulated based on the dyadic pivot sequence $[1,2]$ by applying Equation (4) to get

$$C_{[1,2]} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ c_{2,-1} & c_{2,0} & 0 & 0 \\ c_{3,2}c_{2,-1} & c_{3,2}c_{2,0} & 0 & 0 \\ c_{4,2}c_{2,-1} & c_{4,2}c_{2,0} & 0 & 0 \\ c_{5,2}c_{2,-1} & c_{5,2}c_{2,0} & 0 & 0 \\ c_{6,-1} + c_{6,2}c_{2,-1} & c_{6,0} + c_{6,2}c_{2,0} & 0 & 0 \end{pmatrix},$$

where $c_{2,-1} = c_{2,1}c_{1,-1}$, $c_{2,0} = c_{2,1}c_{1,0}$, $c_{6,-1} = c_{6,1}c_{1,-1}$, and $c_{6,0} = c_{6,1}c_{1,0}$. The Jacobian F' appears as the (4×2)-matrix in the lower left corner of $C_{[1,2]}$. Its accumulation based on the given dyadic pivot sequence requires twelve `fma`'s. Alternatively, F' can be computed as $C_{[2,1]}$ at exactly the same cost.[4]

Choosing $(j, i) = (1, 0)$ as a row pivot and performing the corresponding elimination transforms C into

$$C_{[(2,0)]} = \begin{pmatrix} c_{1,-1} & 0 & 0 & 0 \\ 0 & c_{2,1}c_{1,0} & c_{2,1} & 0 \\ 0 & 0 & 0 & c_{3,2} \\ 0 & 0 & 0 & c_{4,2} \\ 0 & 0 & 0 & c_{5,2} \\ 0 & c_{6,1}c_{1,0} & c_{6,1} & c_{6,2} \end{pmatrix}$$

at the cost of two `fma`'s. With $(i, j) = (2, 6)$ as a column pivot we get

$$C_{[(2,6)]} = \begin{pmatrix} c_{1,-1} & c_{1,0} & 0 & 0 \\ 0 & 0 & c_{2,1} & 0 \\ 0 & 0 & 0 & c_{3,2} \\ 0 & 0 & 0 & c_{4,2} \\ 0 & 0 & 0 & c_{5,2} \\ 0 & 0 & c_{6,1} + c_{6,2}c_{2,1} & 0 \end{pmatrix}$$

by performing one `fma`. While any of the two possible dyadic pivoting sequences results in an overall cost of twelve `fma`'s the accumulation of F' as $C_{[(2,6),1,2]}$ requires only eleven `fma`'s. This example was used in [14] to prove the superiority of mixed row and column pivoting over dyadic pivoting. Little is known about the theoretical discrepancy that may result from a comparison of the optimal RCP sequence with the optimal DP sequence. Practical tests suggest that this discrepancy is likely to be small. One of the motivations for the research presented in this paper is the desired ability to compare RCP and DP strategies for various test problems.

4 Simulated Annealing

Simulated annealing has been applied to the computation of optimal dyadic pivot sequences first in [13]. In [16] we proposed new ideas on how to make logarithmic cooling schedules for inhomogeneous Markov chains work on this problem.

[4] Note that the number of non-zero entries in both C and in $C_{[1,2]}$ is equal to eight. The preaccumulation of this local Jacobian would therefore not lead to a decrease in the number of `fma`'s performed by the directional derivative propagation. Consequently, preaccumulation would not be a good idea if the lion graph represented some basic block in a larger program. The accumulation of the local Jacobian itself is, of cause, cheaper when using elimination techniques. Classical AD requires at least 16 `fma`'s to compute F' as $F' \cdot I_2$ in forward mode. The advantages of elimination techniques result from the implicit exploitation of the structural sparsity of the extended Jacobian and the ability to exploit the associativity of the chain rule.

As in [16] neighborhood relations on the configuration space $\mathcal{V}$ are defined by rearrangements transforming a sequence of pivots $\sigma \in \mathcal{V}$ into $\sigma' \in \mathcal{N}_\sigma \subseteq \mathcal{V}$ where $\mathcal{N}_\sigma$ denotes the neighborhood of σ. These *transitions* are denoted by $[\sigma, \sigma']$ Acceptance probabilities are associated with all feasible transitions $[\sigma, \sigma']$. They are defined by

$$A[\sigma, \sigma'] = \begin{cases} 1, & \text{if } \mathcal{C}(\sigma') \leq \mathcal{C}(\sigma), \\ e^{-\frac{\mathcal{C}(\sigma') - \mathcal{C}(\sigma)}{T}}, & \text{otherwise,} \end{cases} \tag{7}$$

where $\mathcal{C}(\sigma)$ denotes the cost, that is the number of `fma`'s, associated with the sequence of pivots σ. The control parameter T can be interpreted as the *temperature* in the annealing process [10, 11].

We consider two neighborhoods, one for DP and RCP, respectively.

Neighborhood 1 *Let $[j_1, \ldots, j_k]$ be a DP. Do one of the following with probability* 0.5 :

1. *Choose a feasible $j_{k'}$ and remove it from the sequence. Feasibility of $j_{k'}$ with respect to this* backward step *is guaranteed if $[j_{k'+1}, \ldots, j_k]$ can be applied to $C_{[j_1, \ldots, j_{k'-1}]}$.*
2. *Select the next dyadic pivot j_{k+1} in $C_{[j_1, \ldots, j_k]}$ at random* (forward step).

Formally,

$$[j_1, \ldots, j_k] \mapsto \begin{cases} [j_1, \ldots, j_{k'-1}, j_{k'+1}, \ldots, j_k] \\ [j_1, \ldots, j_k, j_{k+1}] \end{cases} .$$

The neighborhood for RCP is defined analogous.

Neighborhood 2

$$[(i_1, j_1), \ldots, (i_k, j_k)] \mapsto \begin{cases} [(i_1, j_1), \ldots, (i_{k'-1}, j_{k'-1}), (i_{k'+1}, j_{k'+1}), \ldots, (i_k, j_k)] \\ [(i_1, j_1), \ldots, (i_k, j_k), (i_{k+1}, j_{k+1})] \end{cases} .$$

The cooling schedule is defined by

$$T = T(k) = \frac{\Gamma}{\ln(k+2)}, \quad \text{for } k = 0, 1, \ldots, \bar{n} \quad ,$$

in Equation (7) for a maximal number of $\bar{n}$ cooling steps. The choice of Γ is based on experimental results. It is discussed in Section 5.2.

For Neighborhood 1 the conditions for asymptotic convergence of the logarithmic simulated annealing algorithm [8] can be proven by taking an approach similar to the one in [16]. For RCP the necessary polynomiality of the computation of the objective function could not be verified so far. Restrictions on the feasibility of RC-pivot choices that preserve overall optimality are the subject of ongoing research. In practice, the design of the simulated annealing algorithm based on the logarithmic cooling schedule makes the occurrence of exponential sequences of pivots highly unlikely.

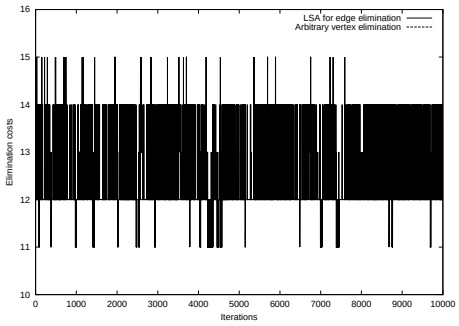

Fig. 1. Evolution of Cost function for Lion Graph

5 Computational Experiments

The tests results presented in this section were obtained using our template library ANGEL (`angellib.sourceforge.net`). It provides various strategies for computing DP's and RCP's, including heuristics and several simulated annealing algorithms.

5.1 Lion

The lion graph is used in [14] to prove the superiority of RC-pivoting over dyadic pivoting. As shown in Section 3.4 both sequences of dyadic pivots result in an overall cost of twelve `fma`'s. The use of RCP's can reduce this cost to eleven `fma`'s. Although being an academic example, the lion graph and its extended Jacobian represent an obvious test case for methods that claim to compute Jacobians at an optimal or near-optimal cost.

So far, no technique, besides exhaustive search, is known that automatically finds an optimal RC-pivot sequence. Moreover, there is not a lot of experience in algorithms for computing near-optimal RC-pivot sequences. The extension of Markowitz-type heuristics for DP to RCP in [2] resulted in an observable superiority of RCP over DP in only a few examples.

To find an optimal pivot sequence, simulated annealing combined with a logarithmic cooling schedule ($\Gamma = 5$) is applied. Figure 1 shows the development of the objective function. Optimal RCP's are found repeatedly even in the high temperature phase. Obviously, straight-forward random search would probably have found the minimum too. Nevertheless, any method for optimizing RCP sequences must be able to solve the lion problem as a basic feasibility test.

5.2 Roe's Flux

In this application from computational fluid dynamics fluxes between two cells of a finite-volume flow solver are calculated using Roe's flux difference scheme

[17]. The extended Jacobian is derived from the computational graph that was generated by the software tool EliAD [18]. The best known sequence of pivots so far results in an overall cost of 962 `fma`'s for accumulating the (10×5)-Jacobian.

	FT		SA	
T/Γ	DP	RCP	DP	RCP
0.1	904	885	899	881
0.2	904	887	904	887
0.5	908	887	908	887
1	894	886	879	885
2	897	984	880	878
5	936	1204	888	882
10	1039	1258	887	902
20	1148	1286	912	993

Table 1. Values of the objective function for different Γ (simulated annealing – SA) and T (fixed temperature Metropolis – FT)

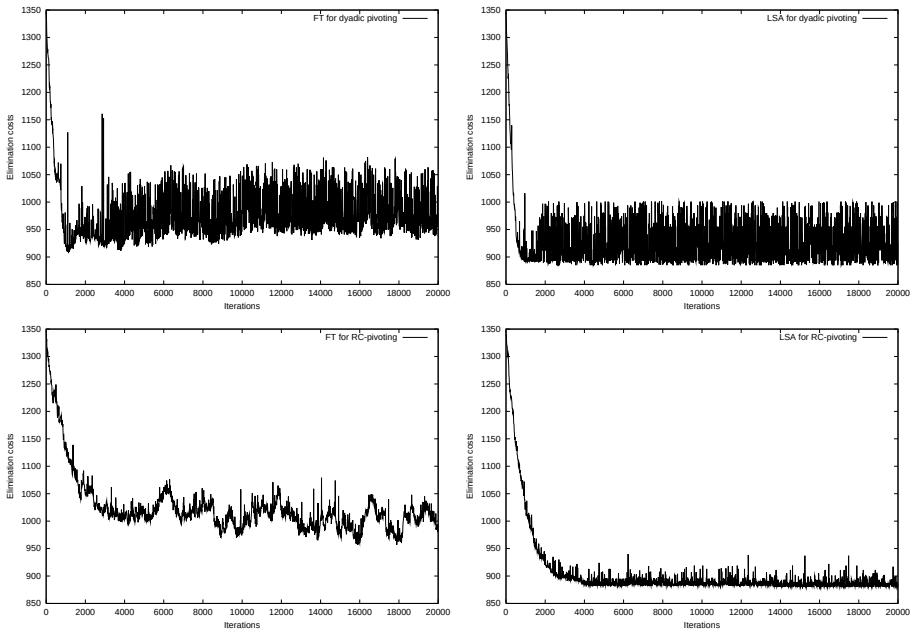

Fig. 2. Evolution of Cost Function for Roe's flux

Table 1 compares the values of the objective function for a Metropolis algorithm with fixed temperature (value in left column should be interpreted as T) and simulated annealing (value in left column should be interpreted as Γ). Both algorithms were applied to DP and RCP, and 10,000 iterations were performed. Both algorithms are strongly influenced by the choice of Γ/T. Low values reduce the ability to escape from local minima. High values lead to a random-search-like behavior. Values for Γ/T in the range between 1 and 2 yield the lowest costs.

The evolution of the objective function over 20,000 iterations is displayed in Figure 2. Further asymptotic improvements can be observed for the logarithmic cooling schedule. DP's converge faster initially and their cost varies stronger during the runtime of the algorithm. This is the result of one dyadic pivot constituting multiple row or column pivots. After 1,000,000 iterations of our simulated annealing algorithm with logarithmic cooling ($\Gamma = 5$) the best RCP was able to compute the Jacobian by using 861 `fma`'s which represents an improvement by over ten per cent.

6 Conclusion

The use of RCP in simulated annealing results in a considerably widened search space compared to DP. Hence, even for graphs where the optimal RCP has a lower cost than the optimal DP it is not clear whether the result of running a simulated annealing algorithm for RCP results in an improvement of the best elimination sequence know so far. Furthermore, switching from DP to RCP may increase the runtime of the simulated annealing algorithm significantly. Nevertheless the use of such algorithms is justified if the corresponding Jacobian code is likely to account for a large portion of the computational effort of a given numerical algorithm. Moreover, the generation of optimal Jacobian code is expected to be a trade-off between a low number of arithmetic operations and the efficiency of the corresponding memory access pattern. Further investigations are required to built the optimal Jacobian code in general.

From the theoretical point of view, the difficulty to reduce accumulation costs by switching from DP to RCP can be interpreted as an indication for the cost discrepancy between optimal DP and optimal RCP not being very large. This conjecture is supported by our numerical results.

Acknowledgments

This work was supported by U.K. Engineering and Physical Sciences Research Council under Grant GR/R38101/01.

This work was also supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38.

References

1. A. Aho, R. Sethi, and J. Ullman. *Compilers. Principles, Techniques, and Tools.* Addison-Wesley, Reading, MA, 1986.
2. A. Albrecht, P. Gottschling, and U. Naumann. Markowitz-type heuristics for computing Jacobian matrices efficiently. In *Proceedings of International Conference on Computational Science*. Springer, LNCS, 2003. To appear.
3. M. Berz, C. Bischof, G. Corliss, and A. Griewank, editors. *Computational Differentiation: Techniques, Applications, and Tools*, Proceedings Series, Philadelphia, 1996. SIAM.
4. G. Corliss, C. Faure, A. Griewank, L. Hascoet, and U. Naumann, editors. *Automatic Differentiation of Algorithms – From Simulation to Optimization*, New York, 2002. Springer.
5. G. Corliss and A. Griewank, editors. *Automatic Differentiation: Theory, Implementation, and Application*, Proceedings Series, Philadelphia, 1991. SIAM.
6. A. Griewank. *Evaluating Derivatives. Principles and Techniques of Algorithmic Differentiation.* Number 19 in Frontiers in Applied Mathematics. SIAM, Philadelphia, 2000.
7. A. Griewank and S. Reese. On the calculation of Jacobian matrices by the Markovitz rule. In *[5]*, pages 126–135, 1991.
8. B. Hajek. Cooling schedules for optimal annealing. *Mathem. of Operations Research*, 13:311–329, 1988.
9. K. Herley. A note on the NP-completeness of optimum Jacobian accumulation by vertex elimination. Presentation at: Theory Institute on Combinatorial Challenges in Computational Differentiation, 1993.
10. P. Van Laarhoven and E. Aarts. *Simulated Annealing: Theory and Applications.* Reidel, 1988.
11. N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 21:1087–92, 1953.
12. U. Naumann. An enhanced Markowitz rule for accumulating Jacobians efficiently. In K. Mikula, editor, *ALGORITHMY'2000 Conference on Scientific Computing*, pages 320–329. Slovak University of Technology, Bratislava, Slovakia, September 2000.
13. U. Naumann. Cheaper Jacobians by Simulated Annealing. *SIAM J. Opt.*, 13(3):660–674, 2002.
14. U. Naumann. Elimination techniques for cheap Jacobians. In *[4]*, pages 247–253, 2002.
15. U. Naumann. Optimal accumulation of Jacobian matrices by elimination methods on the dual computational graph. *Mathematical Programming*, 2003. To appear.
16. U. Naumann and P. Gottschling. Prospects for Simulated Annealing in Automatic Differentiation. In K. Steinhöfel, editor, *SAGA 2002 - Stochastic Algorithms, Foundations and Applications*, volume 2264 of *LNCS*. Springer, Berlin, 2001.
17. P.L. Roe. Approximate Riemann solvers, parameter vectors, and difference schemes. *Journnal of Computational Physics*, 43:357–372, 1981.
18. M. Tadjouddine, S. Forth, J. Pryce, and J. Reid. Performance issues for vertex elimination methods in computing Jacobians using Automatic Differentiation. In *Proceedings of the ICCS 2000 Conference*, volume 2330 of *Springer LNCS*, pages 1077–1086, 2002.

The submitted manuscript has been created by the University of Chicago as Operator of Argonne National Laboratory ("Argonne") under Contract No. W-31-109-ENG-38 with the U.S. Department of Energy. The U.S. Government retains for itself, and others acting on its behalf, a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.