Combinatorial Scientific Computing:
Algorithms for Enabling Computational Science

(Computing Sparse Jacobians and Hessians with Graph Coloring)

Alex Pothen
Purdue University
CSCAPES Institute
www.cs.purdue.edu/homes/apothen/

Woudschoten Conference
Oct. 2010

Outline

- Context: Constrained Nonlinear Optimization
- Sparse derivative computation
  - Modeling Framework: Structural Orthogonality
  - Distance-2 coloring for Jacobians
  - Star and Acyclic coloring for Hessians
  - New Algorithms and Results (Acyclic coloring)
  - Orderings and Optimality
- ColPack Software
- A case study on sparse Jacobian computation
  - Simulated Moving Bed chromatography
- CSCAPES Institute

Industrial Simulated Moving Bed Chromatography

- Petrochemicals (Xylene isomers)
- Sugars (Fructose/glucose separation) ➔ High fructose corn syrup
- Pharmaceuticals (Enantiomeric separation)

  Separate 'good' from 'bad' compounds based on chirality

*http://www.pharmaceutical-technology.com

**http://www.organo.co.jp

Optimization Problem

- Task: $\min_x f(x)$ s.t. $c(x) = 0$.

- Consider Lagrangian $\mathcal{L}(x, \lambda) = f(x) + \lambda^T c(x)$

  Solve

  $$[g(x, \lambda), c(x)] \equiv \left[\nabla f(x) + \lambda^T \nabla c(x),\ c(x)\right] = 0.$$

1

Optimization Problem

PURDUE UNIVERSITY

- Solve

$$[g(x,\lambda), c(x)] \equiv \left[\nabla f(x) + \lambda^T \nabla c(x),\ c(x)\right] = 0.$$

- Apply SQP method, i.e., apply iteration

$$\begin{aligned} &\nabla^2_{x,\lambda}\mathcal{L}(x_k,\lambda_k)\ p_k^N \\ &= \begin{bmatrix} B(x_k,\lambda_k) & A(x_k)^T \\ A(x_k) & 0 \end{bmatrix} p_k^N \\ &= -\nabla_{x,\lambda}\mathcal{L}(x_k,\lambda_k). \end{aligned}$$

2

cscapes

Why compute derivatives?

- Fundamental numerical methods require derivatives:
  - Nonlinear optimization
    - Unconstrained optimization
      (require gradients and Hessians)
    - Constrained optimization
      (require Jacobians, Hessians, or Hessian-vector products)
  - Parameter estimation
  - Solution of discritized nonlinear PDEs
    (require Jacobians or Jacobian-vector products)

- *Simulations* in science, engineering, and economics present additional needs for derivative evaluation:
  - Uncertainty quantification
  - Sensitivity analysis

How *could* derivatives be computed?

- Hand coding
  - Tedious and error-prone
  - Coding time grows with program size and complexity
  - No natural way to compute derivative matrix-vector products
- Divided (Finite) Differencing
  - Incurs truncation errors (is only an approximation)
  - Cost grows with number of independents
  - No natural way to compute transposed-Jacobian-vec products
- Symbolic Differentiation
  - Takes up lots of memory since it relies on first generating symbolic expressions explicitly
  - Does not exploit common sub-expressions directly

Automatic Differentiation overcomes all of these drawbacks.

What is Automatic Differentiation?

- A technique for computing analytic derivatives of a function specified as a computer program
- Key ingredients: analytic differentiation of elementary functions plus propagation by the chain rule of calculus
  - A programming language provides a set of elementary mathematical functions
  - A function computed by a program is a composition of these intrinsic functions
  - Derivatives of intrinsic functions are obtained by table-lookup, and combined using the chain rule

AD: Decomposition of function evaluation and its graph representation

Code list

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

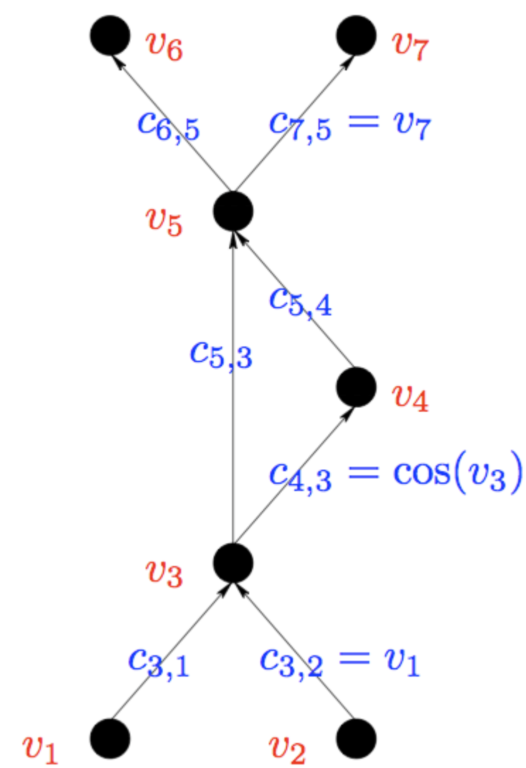
for $j = 1, \ldots, n+p+m$.

Local partial derivatives

$$c_{j,i} = \frac{\partial \varphi_j}{\partial v_i}$$

for $j = 1, \ldots, n+p+m$ and $i \prec j$.

```
v3=v1*v2
v5=v3*sin(v3)  ... v4=sin(v3); v5=v3*v4
v6=cos(v5)
v7=exp(v5)
```



n: independents   m: dependents   p: intermediates

More on AD

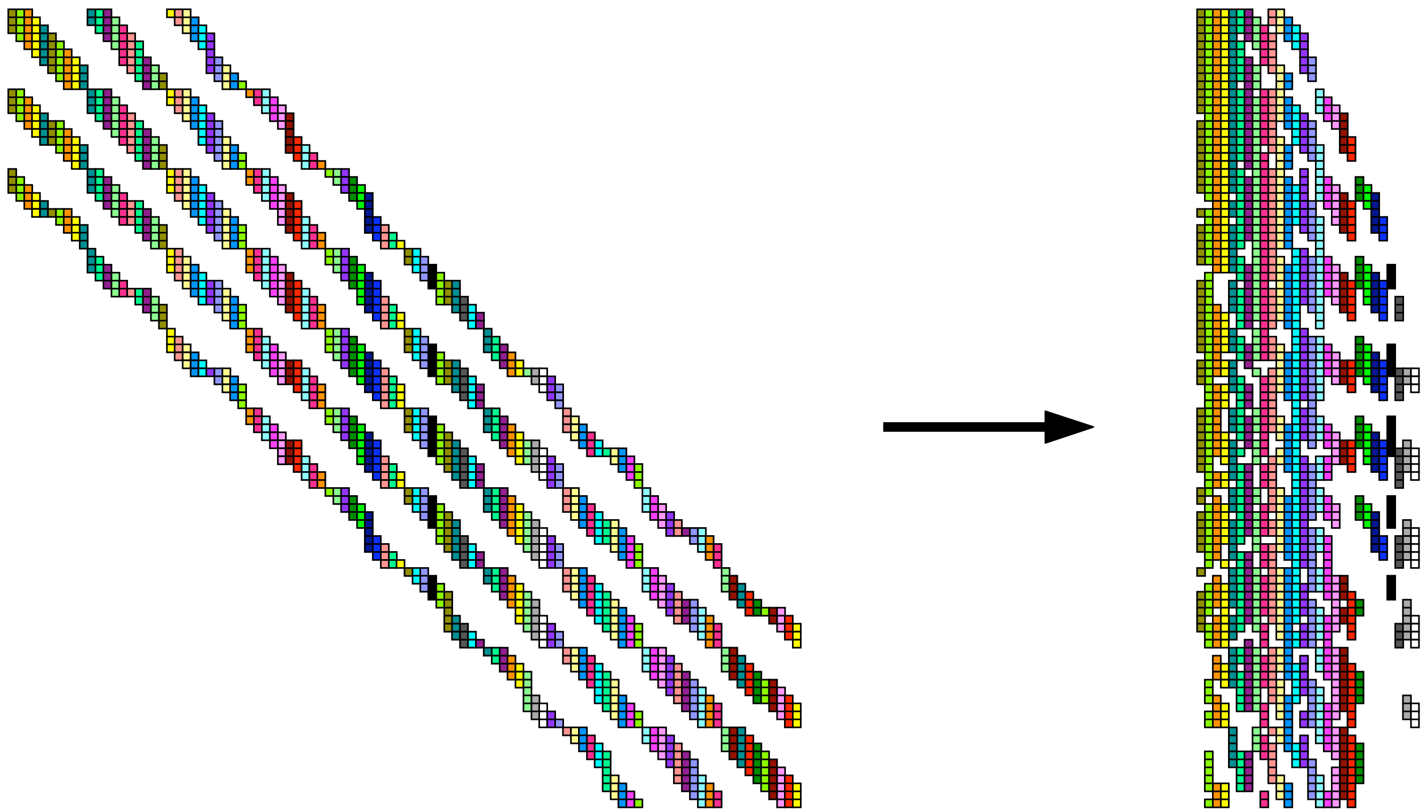
- Has two main modes (due to associativity of chain rule):
  - Forward (Tangent) Mode
  - Reverse (Adjoint) Mode
- Can be implemented in one of two ways:
  - Operator overloading
  - Source transformation
- Modern resurgence in AD spurred by Speelpenning's thesis (1980, Illinois)

Sparse derivative computation

Coloring: An abstraction for grouping a set of related objects into a few ``independent'' sets

Applications in parallel computing, Automatic Differentiation, preconditioning, etc.

Derivative Computation via Compression



Sources of model variation in derivative computation via compression

Three orthogonal axes, each with two possibilities:

Type of Derivative Matrix	Recovery Method	Dimension of Partitioning*
• Jacobian (nonsymmetric)	• Direct	• Unidirectional
• Hessian (symmetric)	• Substitution	• Bidirectional

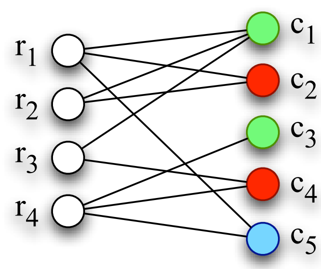
* for the Jacobian case only

Distance-2 coloring: a Model for Structural Orthogonality

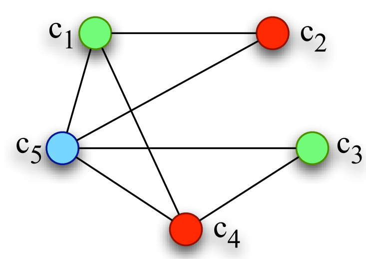
structurally orthogonal partition | distance-2 coloring | distance-1 coloring

unsymmetric case

$$A = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 & a_{15} \\ a_{21} & a_{22} & 0 & 0 & 0 \\ a_{31} & 0 & 0 & a_{34} & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} \end{bmatrix}$$



$\mathcal{G}_b$



$\mathcal{G}_b^2[\mathcal{V}_2]$

Distance-2 coloring: a Model for Structural Orthogonality

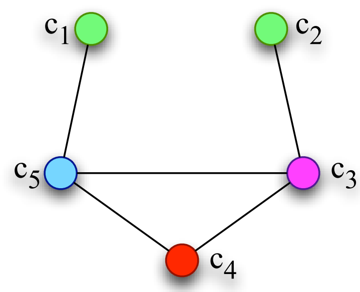
symmetric case

structurally orthogonal partition

$$\begin{bmatrix} a_{11} & 0 & 0 & 0 & a_{15} \\ 0 & a_{22} & a_{23} & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & a_{35} \\ 0 & 0 & a_{43} & a_{44} & a_{45} \\ a_{51} & 0 & a_{53} & a_{54} & a_{55} \end{bmatrix}$$

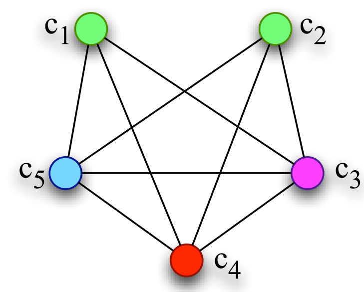
A

distance-2 coloring



$\mathcal{G}_a$

distance-1 coloring



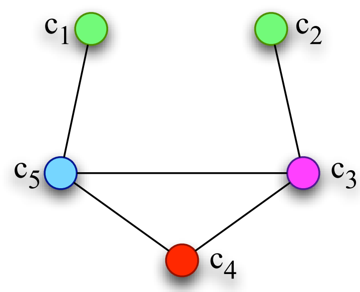
$\mathcal{G}_a^2$

Distance-2 coloring: a Model for Structural Orthogonality

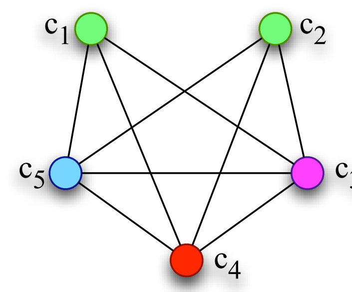
structurally orthogonal partition | distance-2 coloring | distance-1 coloring

symmetric case

$$A = \begin{bmatrix} a_{11} & 0 & 0 & 0 & a_{15} \\ 0 & a_{22} & a_{23} & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & a_{35} \\ 0 & 0 & a_{43} & a_{44} & a_{45} \\ a_{51} & 0 & a_{53} & a_{54} & a_{55} \end{bmatrix}$$



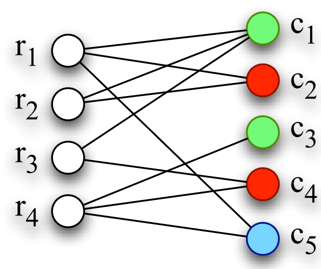
$\mathcal{G}_a$



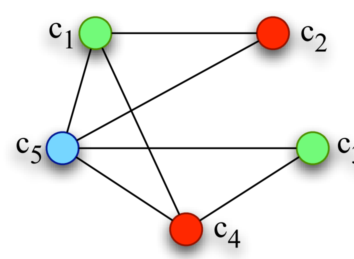
$\mathcal{G}_a^2$

unsymmetric case

$$A = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 & a_{15} \\ a_{21} & a_{22} & 0 & 0 & 0 \\ a_{31} & 0 & 0 & a_{34} & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} \end{bmatrix}$$



$\mathcal{G}_b$



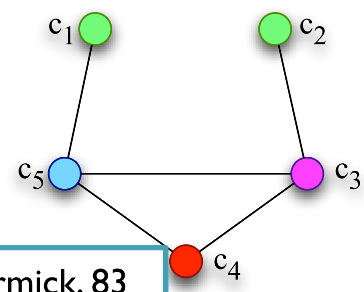
$\mathcal{G}_b^2[\mathcal{V}_2]$

Distance-2 coloring: a Model for Structural Orthogonality

	structurally orthogonal partition	distance-2 coloring	distance-1 coloring

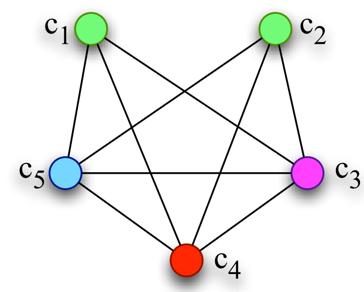
symmetric case

$$A = \begin{bmatrix} a_{11} & 0 & 0 & 0 & a_{15} \\ 0 & a_{22} & a_{23} & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & a_{35} \\ 0 & 0 & a_{43} & a_{44} & a_{45} \\ a_{51} & 0 & a_{53} & a_{54} & a_{55} \end{bmatrix}$$



McCormick, 83

$\mathcal{G}_a$

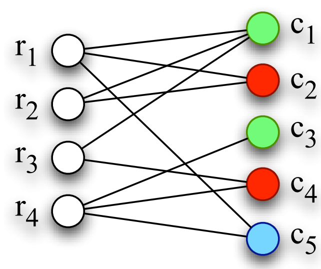


$\mathcal{G}_a^2$

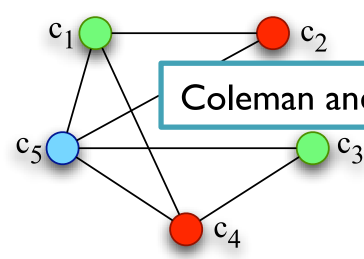
nonsymmetric case

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & a_{15} \\ a_{21} & a_{22} & 0 & 0 & 0 \\ a_{31} & 0 & 0 & a_{34} & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} \end{bmatrix}$$

Curtis, Powell and Reid, 74



$\mathcal{G}_b$

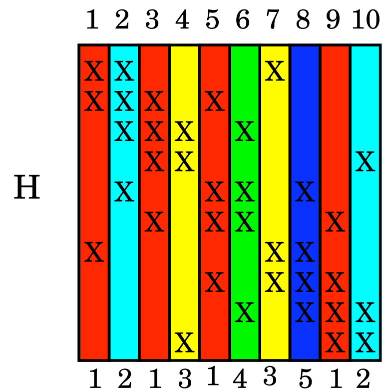


Coleman and More, 83

$\mathcal{G}_b^2[\mathcal{V}_2]$

Model for Hessian computation via a direct method: Star Coloring

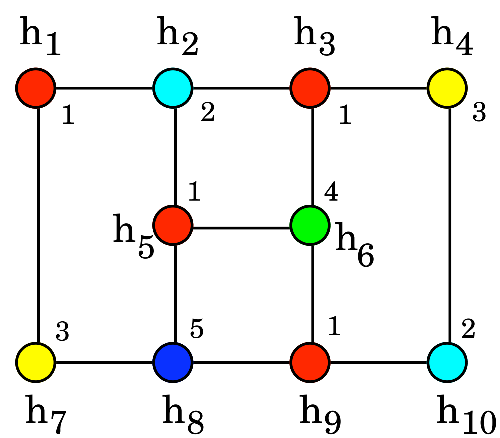
symmetrically orthogonal partition



$$\begin{pmatrix} h_{11} & h_{12} & h_{17} & 0 & 0 \\ h_{21}+h_{23}+h_{25} & h_{22} & 0 & 0 & 0 \\ h_{33} & h_{32} & h_{34} & h_{36} & 0 \\ h_{43} & h_{4,10} & h_{44} & 0 & 0 \\ h_{55} & h_{52} & 0 & h_{56} & h_{58} \\ h_{63}+h_{65}+h_{69} & 0 & 0 & h_{66} & 0 \\ h_{71} & 0 & h_{77} & 0 & h_{78} \\ h_{85}+h_{89} & 0 & 0 & 0 & h_{88} \\ h_{99} & h_{9,10} & 0 & h_{96} & h_{98} \\ h_{10,9} & h_{10,10} & h_{10,4} & 0 & 0 \end{pmatrix}$$

star coloring:

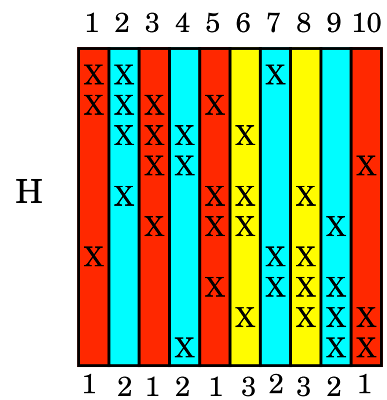
- distance-1 coloring +
- every path on 4 vertices uses at least 3 colors



compressed Hessian
B = HS

Model for Hessian computation via substitution: Acyclic coloring

substitutable partition

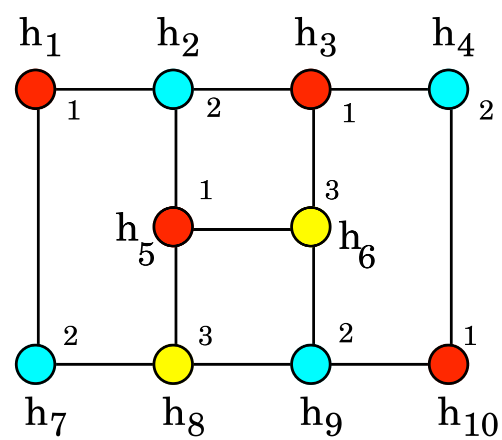


$$B = \begin{pmatrix} h_{11} & h_{12}+h_{17} & 0 \\ h_{21}+h_{23}+h_{25} & h_{22} & 0 \\ h_{33} & h_{32}+h_{34} & h_{36} \\ h_{43}+h_{4,10} & h_{44} & 0 \\ h_{55} & h_{52} & h_{56}+h_{58} \\ h_{63}+h_{65} & h_{69} & h_{66} \\ h_{71} & h_{77} & h_{78} \\ h_{85} & h_{87}+h_{89} & h_{88} \\ h_{9,10} & h_{99} & h_{96}+h_{98} \\ h_{10,10} & h_{10,4}+h_{10,9} & 0 \end{pmatrix}$$

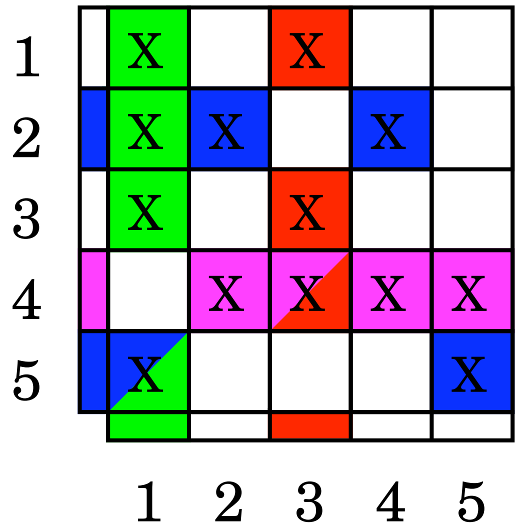
compressed Hessian
B = HS

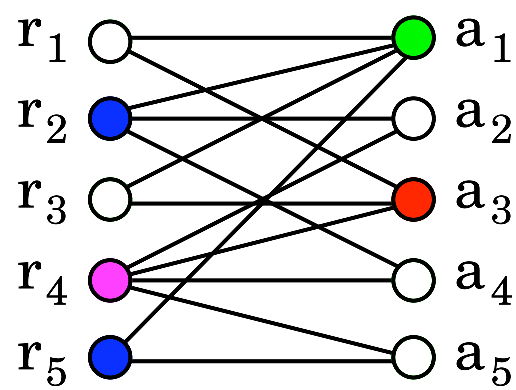
acyclic coloring:

- distance-1 coloring +
- every cycle uses at least 3 colors



Bidirectional Jacobian computation





- Direct method:
  - star bicoloring (shown in the picture) (Coleman & Verma'98 and Hossain & Steihaug'98)
- Substitution method:
  - acyclic bicoloring (Coleman & Verma'98)

Overview of coloring models in derivative computation

	Unidirectional partition	Bidirectional partition	
Jacobian	distance-2 coloring	star bicoloring	**Direct**
Hessian	star coloring restricted star coloring	NA	**Direct**
Jacobian	NA	acyclic bicoloring	**Substitution**
Hessian	acyclic coloring triangular coloring	NA	**Substitution**

Jacobian: bipartite graph
Hessian: adjacency graph

Further reading: Gebremedhin, Manne and Pothen, SIAM Review 47(4):629—705, 2005.

Coloring Algorithms

Complexity of coloring

New Algorithms

Results

Complexity and algorithms

- Minimizing colors for Distance-k, star, and acyclic coloring are NP-hard
- Approximating coloring to within $O(n^{1-e})$ is NP-hard too

GREEDY$(G=(V,E))$

Order the vertices in V

for i = 1 to $|V|$ **do**

Determine forbidden colors to v_i

Assign v_i the smallest permissible color

end-for

- A greedy heuristic usually gives a good, often optimal, solution
- The key is to find good orderings for coloring, and many have been developed

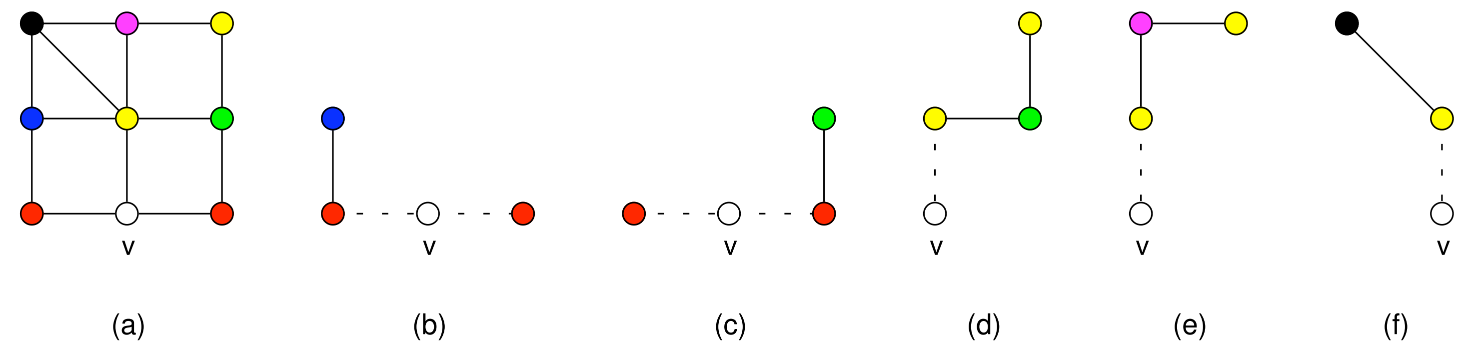
Ref: Gebremedhin, Tarafdar, Manne, Pothen, SIAM J. Sci. Compt. 29:1042--1072, 2007.

New Algorithms

- For Jacobians, partial distance-2 colorings of bipartite graphs with GREEDY can be computed with orders of magnitude less storage and time than distance-1 colorings of the square of the graph, as had been done in earlier work
- For distance-k coloring, GREEDY can be implemented to run in $O(|V|d_k)$ time, where d_k is average degree-k
- We have developed $O(|V|d_2)$-time heuristic algorithms for star and acyclic coloring

  Key idea: exploit structure of two-colored induced subgraphs

A star coloring algorithm



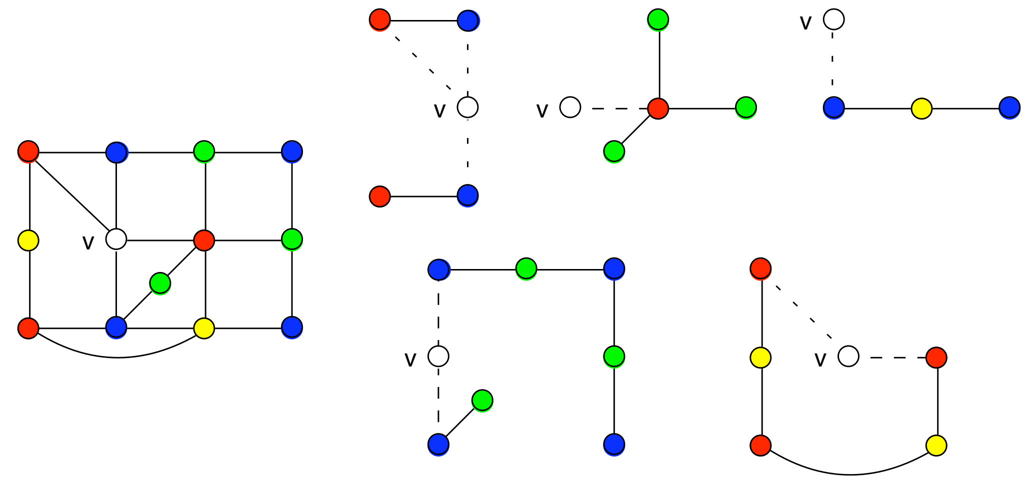
Algorithm (Input: $G=(V,E)$):

for each v in V

1. Choose color for v:
   - Forbid colors used by neighbors $N(v)$ of v
   - Forbid colors leading to two-colored paths on 4 vertices:
     - For every pair of same-colored vertices w and x in $N(v)$, forbid colors used by $N(w)$ and $N(x)$
     - For every non-single-edge star S incident on v, forbid color of hub of S
2. Update collection of two-colored stars

Time: $O(|V|d_2)$ Space: $O(|E|)$

An acyclic coloring algorithm



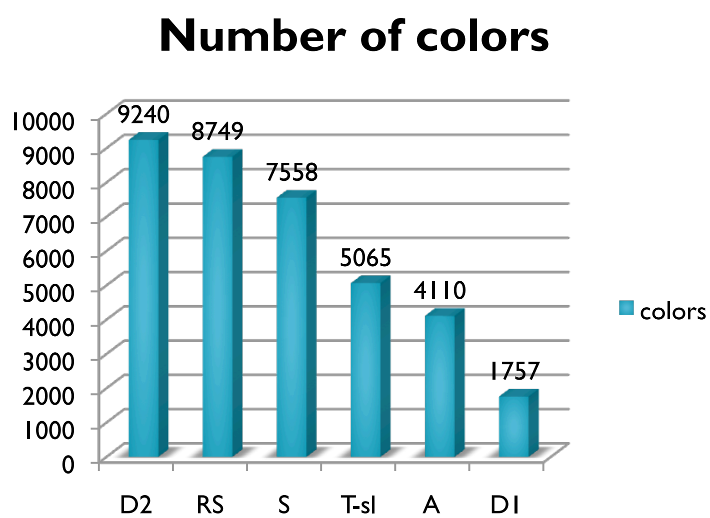
Algorithm (Input: $G=(V,E)$):

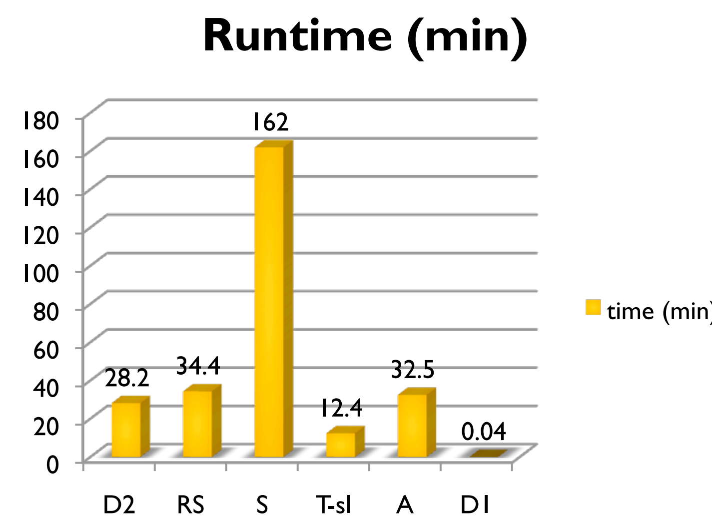
for each v in V

1. Choose color for v
   - Forbid colors used by neighbors of v
   - Forbid colors leading to two-colored cycles
     - For every tree T incident on v, if v adjacent to at least two same-color vertices, forbid the other color in T
2. Update collection of two-colored trees (merge if necessary)

Time: $O(|V|d_2\,\alpha)$ Space: $O(|E|)$

Performance



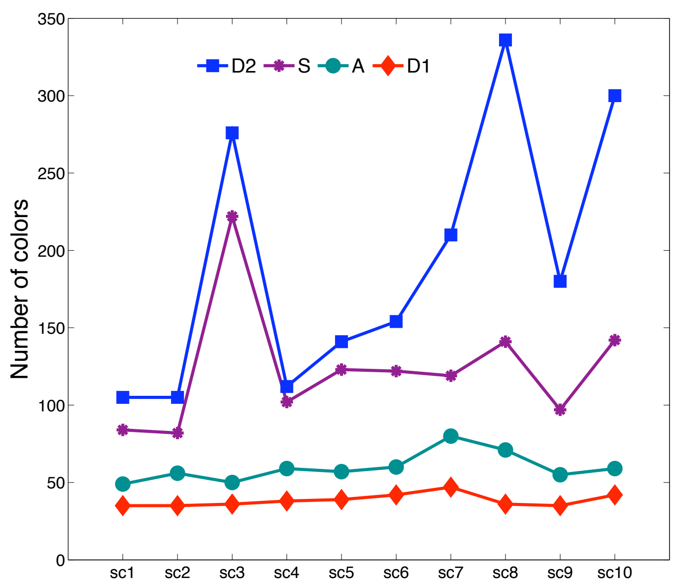


29 test graphs, aggregate data

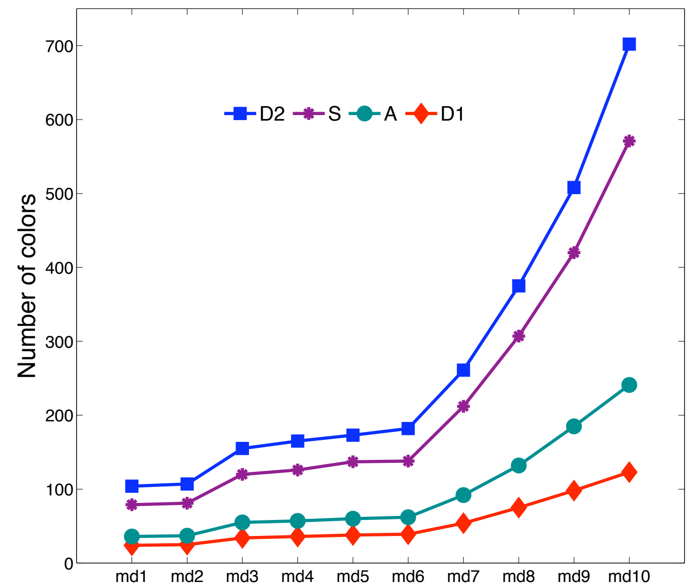
|V|	|E|	Max Degree	Min Degree	Avg Degree
1.5M	88M	6.4K	800	4.2K

Further reading:
Gebremedhin, Tarafdar, Manne and Pothen, New Acyclic and Star Coloring Algorithms with Applications to Computing Hessians. *SIAM J. Sci. Comput. 29:1042–1072, 2007.*

Number of Colors: Star, Acyclic

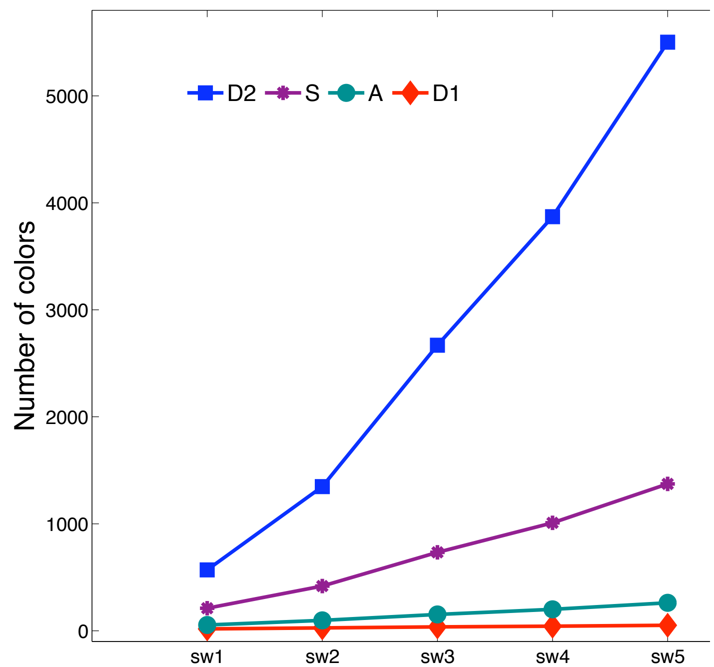


Scientific Computing
1 -23 Million edges

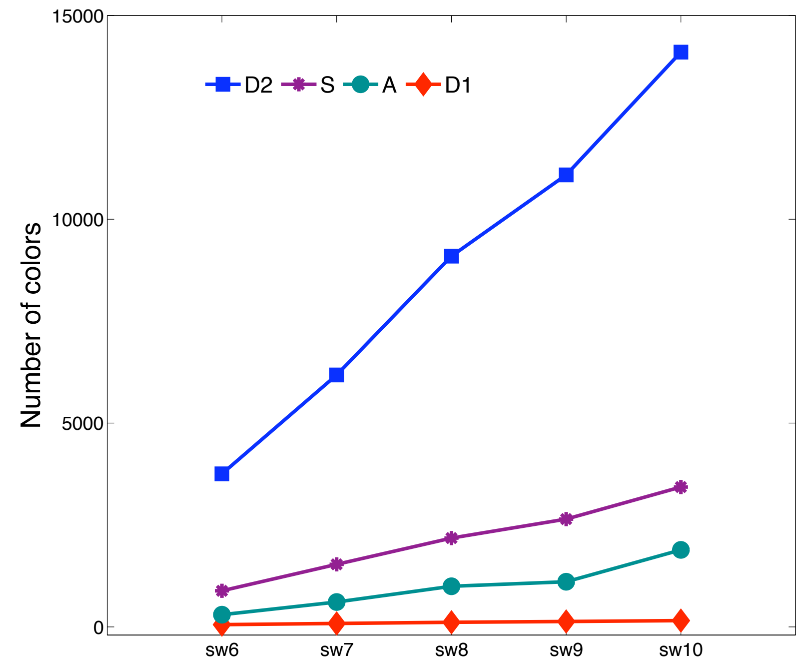


Molecular Dynamics
0.4 – 6.5 Million edges

Number of Colors: Star, Acyclic



Small World I random graphs
2 - 11 Million edges



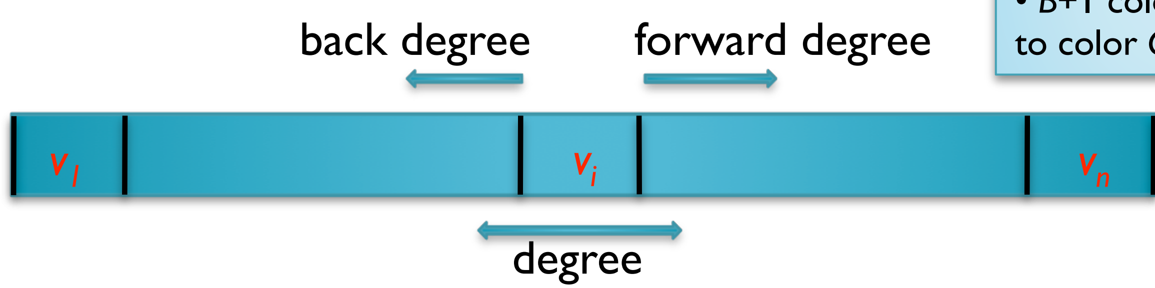
Small World II random graphs
2 - 11 Million edges
Higher degrees, dense subgraphs

Ordering techniques

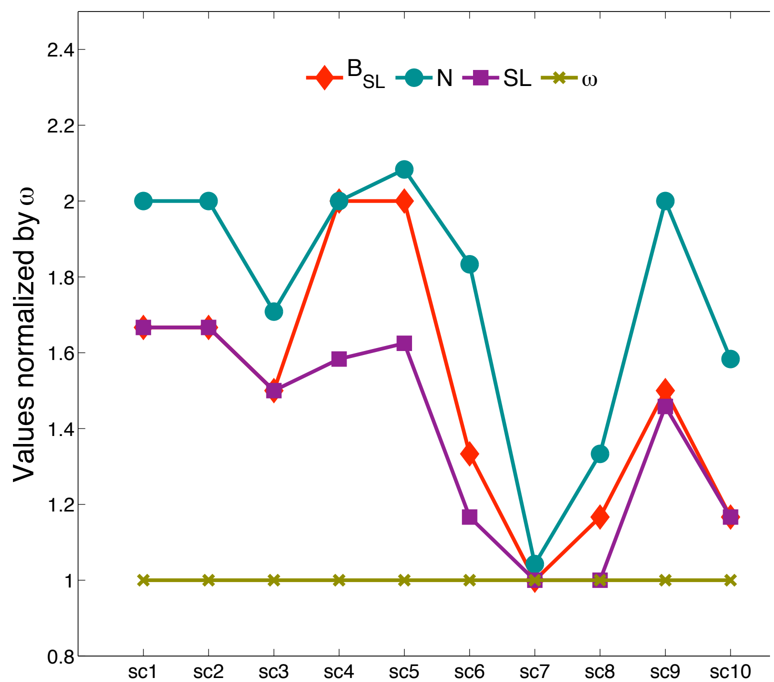
Ordering	Description	Remark
Largest First	v_i has largest degree in sequence $v_i, v_{i+1}, \ldots, v_n$	sorted in non-increasing order of degrees in input graph *G*
Incidence Degree	v_i has largest back degree in sequence $v_i, v_{i+1}, \ldots, v_n$	
Smallest Last	v_i has smallest back degree in sequence $v_1, v_2, \ldots, v_i$	• minimizes *B* over *all* orderings • $B_{SL} + 1 = col(G)$
Dynamic Largest First	v_i has largest forward degree in sequence $v_i, v_{i+1}, \ldots, v_n$	

O(|*E*|)-time implementations possible for all four

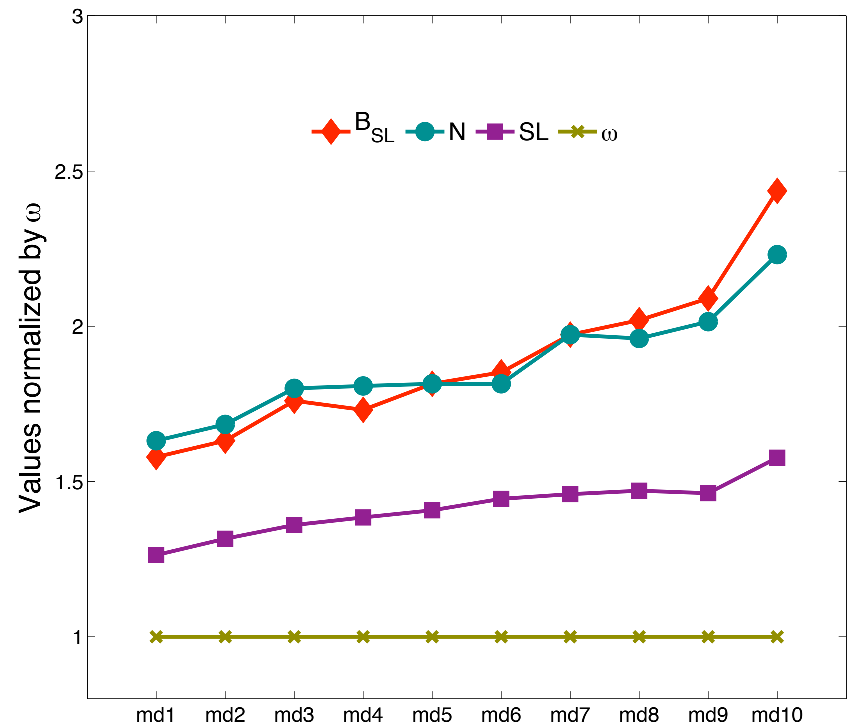
- *B* = max back degree over entire seq.
- *B*+1 colors suffice to color *G*.



Nearly Optimal Distance-*1* coloring



Scientific Computing
1 -23 Million edges

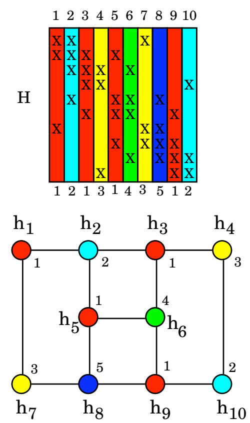


Molecular Dynamics
0.4 – 6.5 Million edges

Hessian recovery algorithms

Direct (star coloring)

$$B \quad \begin{pmatrix} h_{11} & h_{12} & h_{17} & 0 & 0 \\ h_{21}+h_{23}+h_{25} & h_{22} & 0 & 0 & 0 \\ h_{33} & h_{32} & h_{34} & h_{36} & 0 \\ h_{43} & h_{4,10} & h_{44} & 0 & 0 \\ h_{55} & h_{52} & 0 & h_{56} & h_{58} \\ h_{63}+h_{65}+h_{69} & 0 & 0 & h_{66} & 0 \\ h_{71} & 0 & h_{77} & 0 & h_{78} \\ h_{85}+h_{89} & 0 & 0 & 0 & h_{88} \\ h_{99} & h_{9,10} & 0 & h_{96} & h_{98} \\ h_{10,9} & h_{10,10} & h_{10,4} & 0 & 0 \end{pmatrix}$$



$H[i, i] \leftarrow B[i, color[h_i]]$

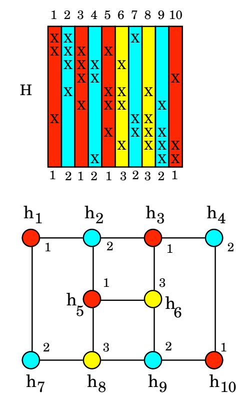
for each two-colored star

for each spoke-hub pair (h_s, h_u)

$H[s, u] \leftarrow B[s, color[h_u]]$

Substitution (acyclic coloring)

$$B \quad \begin{pmatrix} h_{11} & h_{12}+h_{17} & 0 \\ h_{21}+h_{23}+h_{25} & h_{22} & 0 \\ h_{33} & h_{32}+h_{34} & h_{36} \\ h_{43}+h_{4,10} & h_{44} & 0 \\ h_{55} & h_{52} & h_{56}+h_{58} \\ h_{63}+h_{65} & h_{69} & h_{66} \\ h_{71} & h_{77} & h_{78} \\ h_{85} & h_{87}+h_{89} & h_{88} \\ h_{9,10} & h_{99} & h_{96}+h_{98} \\ h_{10,10} & h_{10,4}+h_{10,9} & 0 \end{pmatrix}$$



$H[i, i] \leftarrow B[i, color[h_i]]$

for each two-colored tree *T*

while *T* is non-empty

evaluate and delete "leaf" edges

Ref: Gebremedhin, Pothen, Tarafdar and Walther, INFORMS JOC, 2009.

Sparse derivative computation

Software – ColPack

Coloring, Ordering, Functionalities for Derivatives

ColPack: coloring capabilities

General graph $G = (V, E)$	Bipartite graph, One sided coloring $G_b = (V_1, V_2, E)$	Bipartite graph, Bicoloring $G_b = (V_1, V_2, E)$
• Distance-1 coloring $O(\|V\|d_1) = O(\|E\|)$	• Distance-2 coloring on V_2 $O(\|E\|\,\Delta(V_1))$	• Star bicoloring* $O((\|V_1\|+\|V_2\|)d_2)$
• Distance-2 coloring $O(\|V\|d_2)$	• Distance-2 coloring on V_1 $O(\|E\|\,\Delta(V_2))$	
• Star coloring* $O(\|V\|d_2)$		
• Acyclic coloring $O(\|V\|d_2\,\alpha)$		
• Restricted star coloring $O(\|V\|d_2)$		
• Triangular coloring* $O(\|V\|d_2)$		

* more than one algorithm available; complexity of fastest algorithm shown

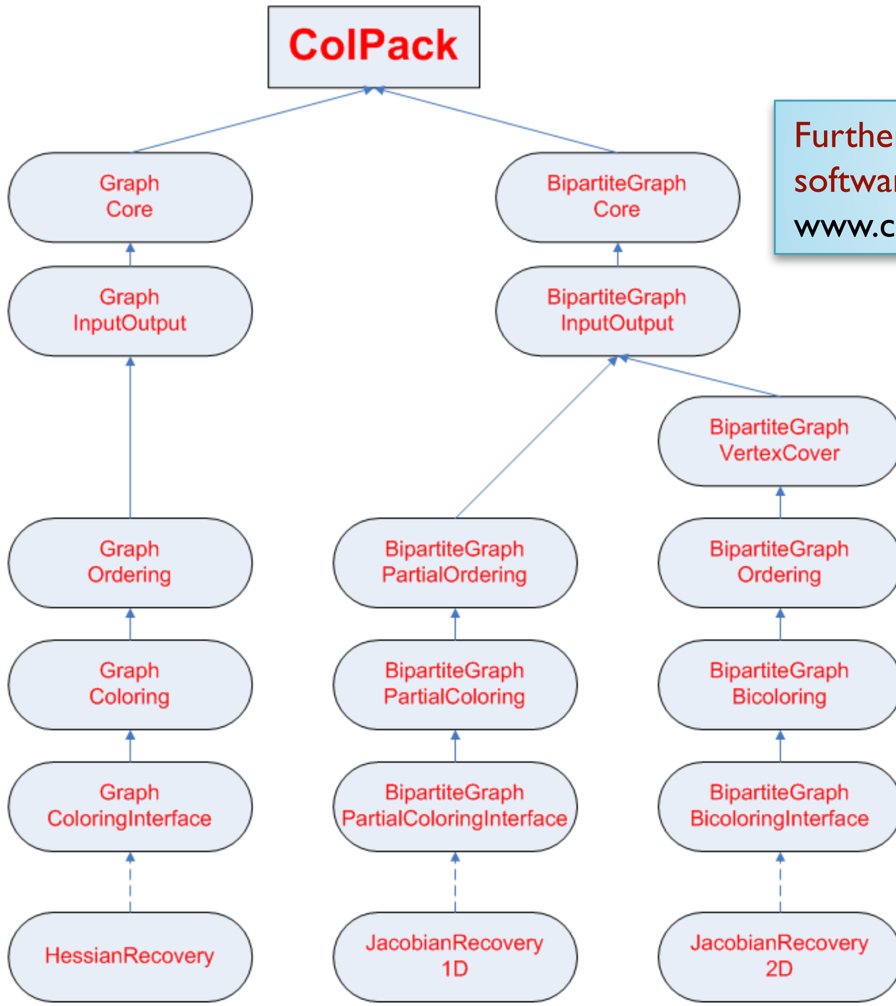
ColPack: ordering capabilities

General graph	Bipartite graph, One sided coloring	Bipartite graph, Bicoloring
• Natural	• Column Natural	• Natural
• Random	• Column Random	• Random
• Largest First	• Column LF	• LF
• Smallest Last	• Column SL	• SL
• Incidence Degree	• Column ID	• ID
• Dynamic LF	• Row Natural	• Dynamic LF
• Distance-2 LF	• Row Random	• Selective LF
• Distance-2 SL	• Row LF	• Selective SL
• Distance-2 ID	• Row SL	• Selective ID
	• Row ID	

ColPack: other functionalities

- Recovery routines
  - Jacobians
    - Unidirectional, Direct (via distance-2 coloring)
      - columnwise and rowwise
    - Bidirectional, Direct (via star bicoloring)
  - Hessians
    - Direct (via star coloring)
    - Substitution (via acyclic coloring)
- Graph construction routines
  - Various file formats supported

ColPack: organization



Further information & software download: www.cscapes.org/coloringpage

A case study on sparse Jacobian computation

Simulated Moving Bed chromatography

Principle of Chromatography

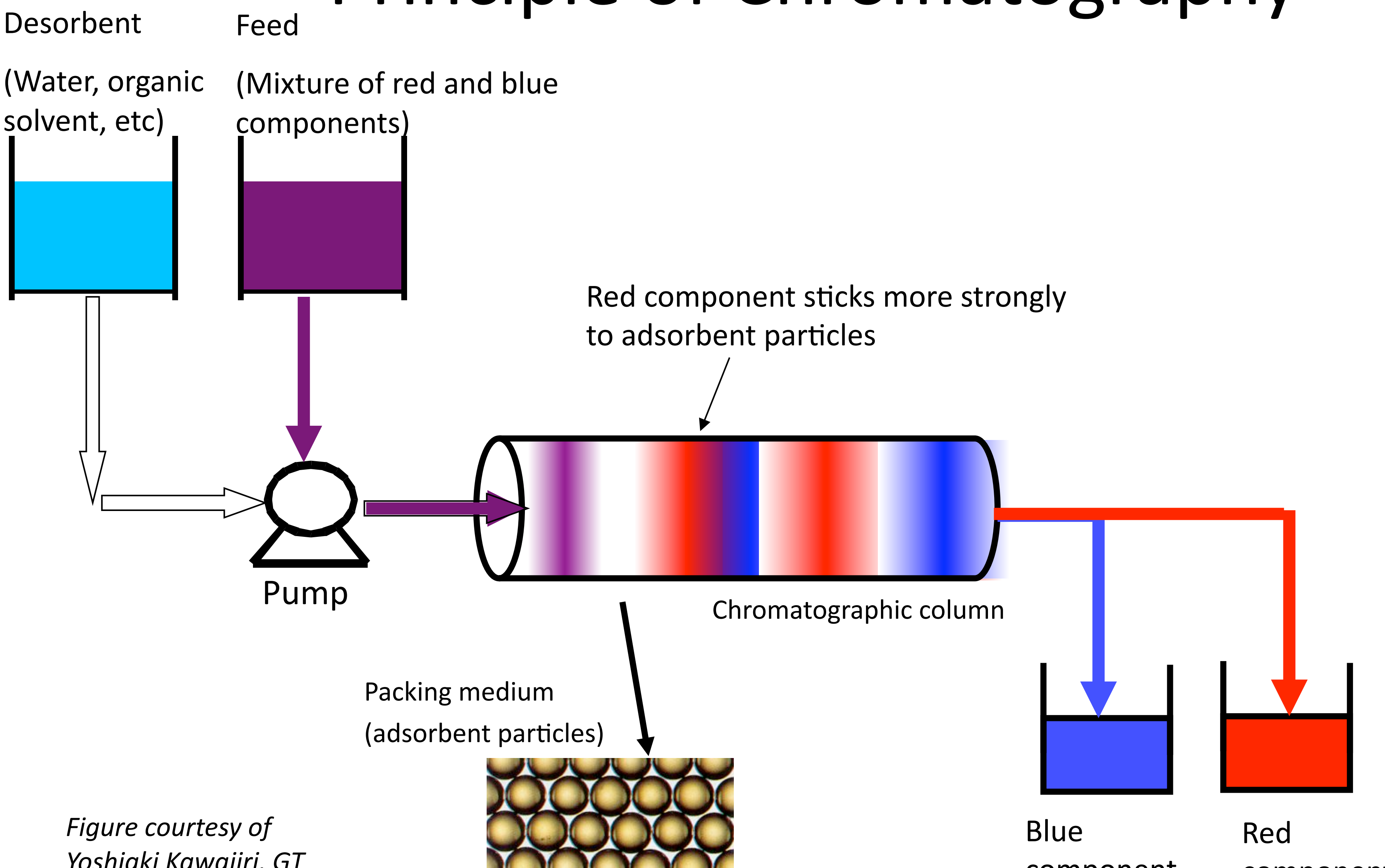
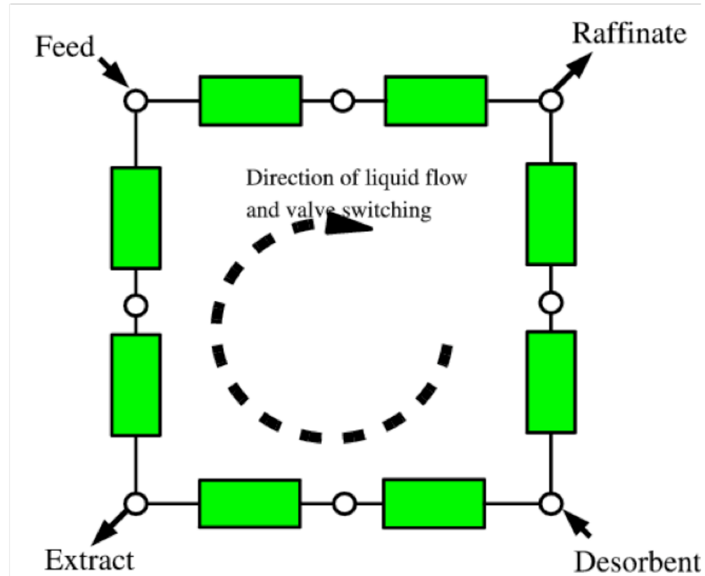


Figure courtesy of Yoshiaki Kawajiri, GT

Simulated Moving Bed process

- A psuedo counter-current process that mimics operation of Moving Beds
- Reaches only Cyclic Steady State
- Various objectives to be maximized:
  E.g: product purity, product recovery, desorbent consumption, throughput
- We considered throughput maximization
- Objective modeled as an optimization problem with PDAEs as constraints
- Full discretization was used to solve the PDAEs ➔ sparse Jacobians



Framework for sparse derivative computation

Procedure SPARSECOMPUTE($F : R^n \to R^m$ or $f : R^n \to R$)

1. Determine the sparsity structure of derivative $F' \equiv A \in R^{m \times n}$ or $f'' \equiv A \in R^{n \times n}$
2. Obtain a seed matrix $S \in \{0,1\}^{n \times q}$ with the smallest q
3. Compute elements of compressed matrix $B = AS \in R^{m \times q}$
4. Recover the numerical values of the entries of *A* from *B*

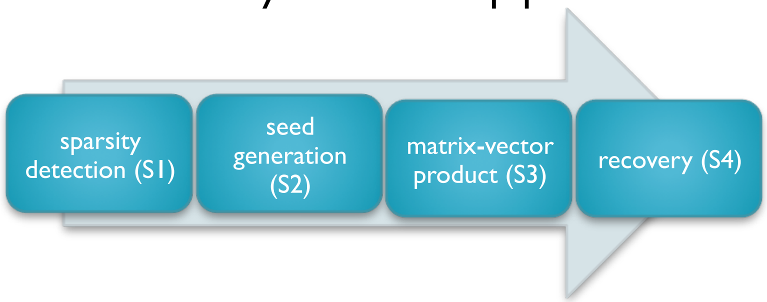
Seed matrix *S* partitions columns of *A*:

$$s_{jk} = \begin{cases} 1 & \text{Iff column } j \text{ of } A \text{ belongs to group } k \\ 0 & \text{otherwise} \end{cases}$$

S computed by coloring a graph of A; B computed using AD

Results on Jacobian computation on SMB problem

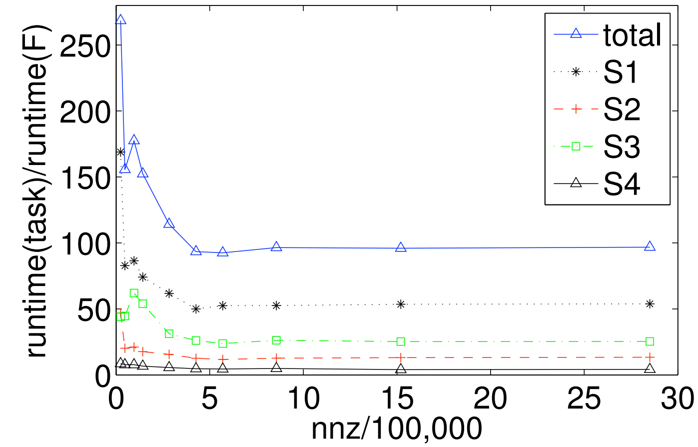
- Tested efficacy of the 4-step procedure:

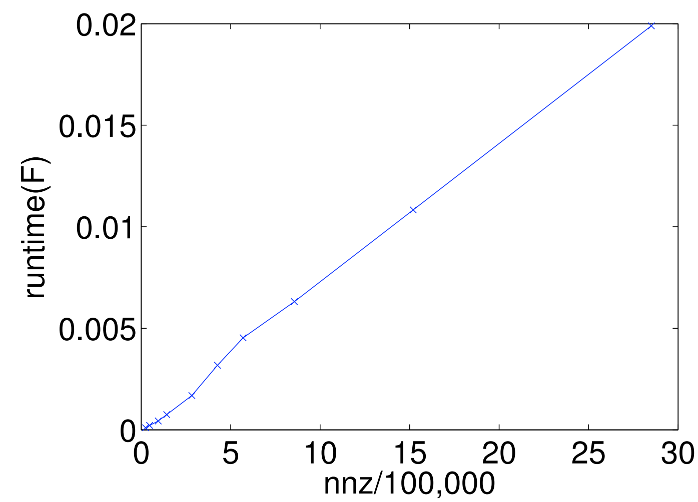


- Used ADOL-C for steps S1and S3, and ColPack for steps S2 and S4
- Observed results for each step matched analytical results
- Techniques enabled huge savings in runtime

  Time(Jacobian eval) ≈ 100×Time(function eval)

- Dense computation (without exploiting sparsity) was infeasible





Ref: Gebremedhin, Pothen and Walther, AD2008, LNCSE 64, 339—349, 2008.

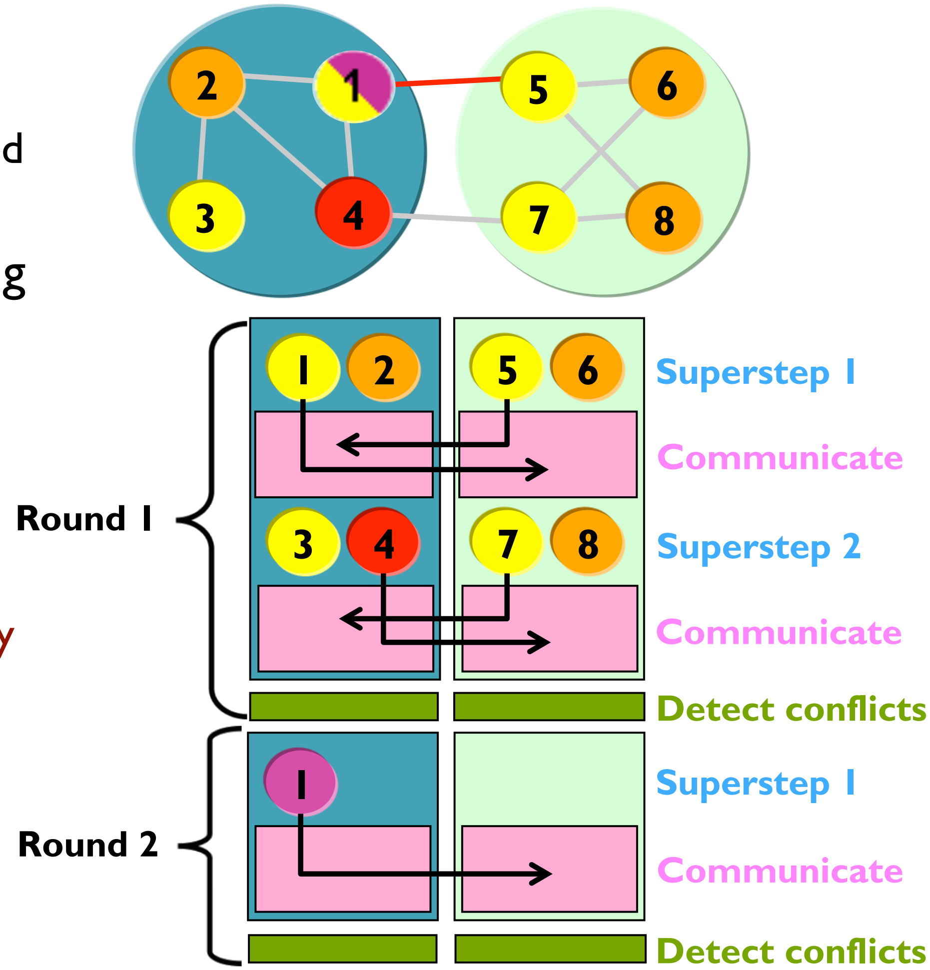
Parallel Coloring

Parallelizing greedy coloring

- Goal: Given a distributed graph, parallelize greedy coloring such that
  - Speedup is attained
  - Number of colors used is roughly same as in serial
- Difficult task since greedy is inherently sequential, computation small relative to communication, and data accesses are irregular
- D1 coloring: approaches based on Luby's parallel algorithm for *maximal independent set* had very limited success
- D2 coloring: no practical parallel algorithms existed
- We developed a framework for effective parallelization of greedy coloring on distributed memory architectures
- Using the framework, we designed various specialized parallel algorithms for D1 and D2 coloring
  - First MPI implementations to yield speedup

Framework for parallel greedy coloring

- Exploit features of initial data distribution
  - Distinguish between interior and boundary vertices
- Proceed in rounds, each having two phases:
  - Tentative coloring
  - Conflict detection
- Coloring phase organized in supersteps
  - A processor communicates only after coloring a subset of its assigned vertices
    ➔ infrequent, coarse-grain communication
- Randomization used in resolving conflicts



Specializations of the framework

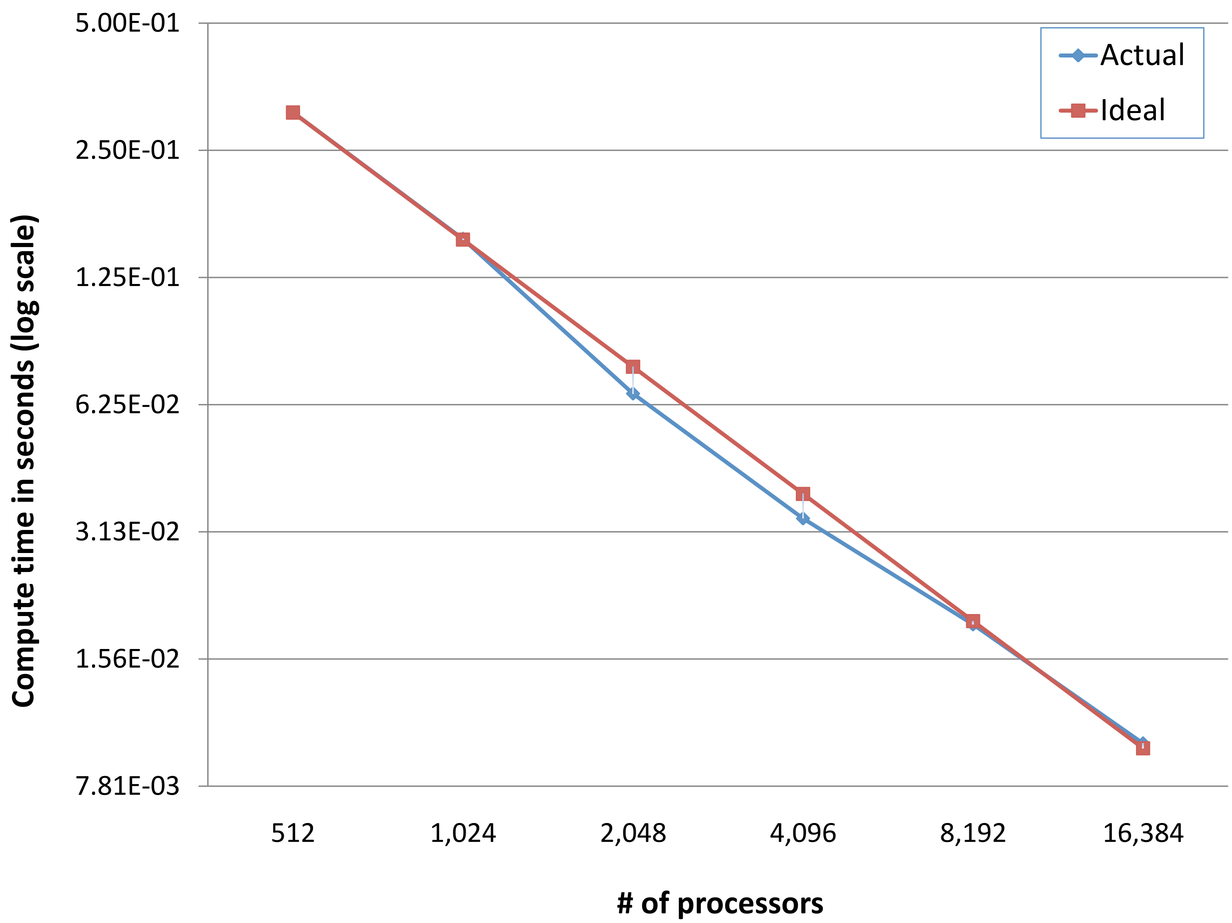
Color selection strategies	• First Fit • Staggered First Fit
Coloring order	• Interior before boundary • Interior after boundary • Interior interleaved with boundary
Local vertex ordering	• Various degree-based techniques
Supersteps	• Synchronous • Asynchronous
Inter-processor communication	• Customized • Broadcast-based

Implementation and experimentation

- Using the framework (JPDC, 2008)
  - Designed specialized parallel algorithms for distance-1 coloring
  - Experimentally studied how to tune "parameters" according to
    - size, density, and distribution of input graph
    - number of processors
    - computational platform
- Extending the framework (SISC, under review)
  - Designed parallel algorithms for D2 and restricted star coloring (to support Hessian computation)
  - Designed parallel algorithms for D2 coloring of bipartite graphs (to support Jacobian computation)

  New Challenge: efficient mechanism for information exchange between processors hosting D2 neighboring vertices needs to be devised
- Software
  - MPI implementations of D1 and D2 coloring made available in Zoltan

D1-coloring, IBM Blue Gene/P



Overview of our contributions

- Serial algorithms and software
  - Jacobian computation via distance-2 coloring algorithms on bipartite graphs
  - Hessians: Developed novel algorithms for acyclic, star, distance-k (k = 1,2) and other coloring problems; developed associated matrix recovery algorithms
  - Several ordering algorithms for reducing the number of colors
  - Delivered implementations via the software package ColPack (released Oct. 2008; Oct 2010)
  - Interfaced ColPack with the AD tool ADOL-C
- Application Highlights
  - Enabled Jacobian computation in Simulated Moving Beds
  - Enabled Hessian computation in optimizing electric power flow
- Parallel algorithms and software
  - Developed a parallelization framework for distributed-memory greedy coloring
  - Deployed implementations via the Zoltan toolkit
  - Designed massively multi-threaded and multi-core parallel algorithms for D1 and D2 coloring

Conclusion

- For large, sparse derivative matrices, computation via compression (coloring) renders big savings in runtime and memory usage
- A unifying framework is structural orthogonality and relaxations, which leads to efficient algorithms for coloring.
- Integrated ColPack into an AD tool and interfaced with optimization software.

For more information

- AD algorithms and community
  - Griewank and Walther, Evaluating Derivatives, SIAM, 2008
  - http://www.autodiff.org
- ADOL-C: http://www.math.tu-dresden.de/~adol-c
- OpenAD: http://www.mcs.anl.gov/openad/
- ColPack: http://www.cscapes.org/coloringpage

- Contact: apothen@purdue.edu

Thanks

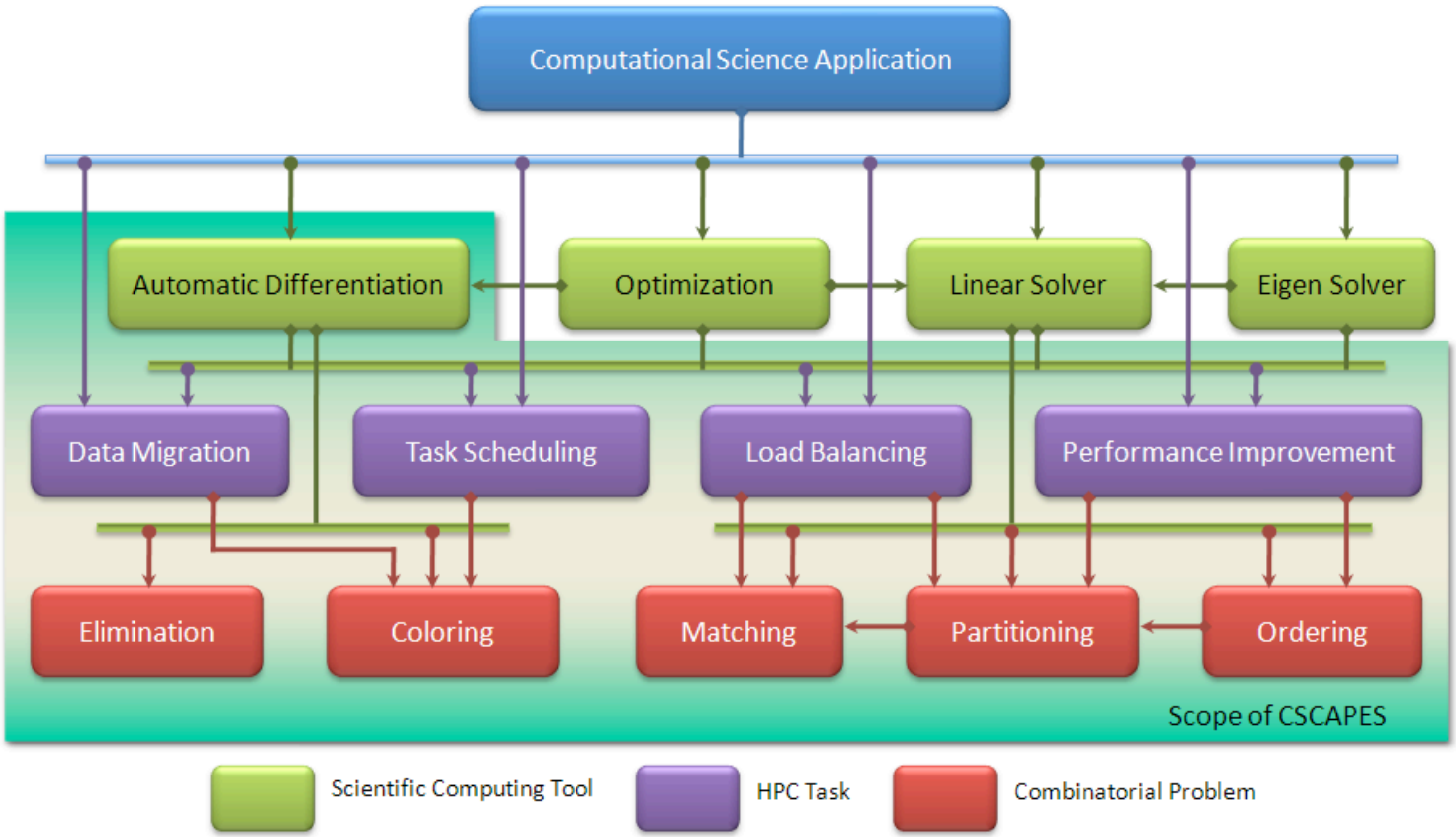
- Coloring
  - Assefaw Gebremedhin, Fredrik Manne, Mostofa Patwary, Duc Nguyen, Arijit Tarafdar
- AD
  - Paul Hovland, Uwe Naumann, Andrea Walther
- SMB
  - Larry Biegler (CMU), Yoshiaki Kawajiri (GaTech)
- CSCAPES
  - Erik Boman, Paul Hovland, Umit Catalyurek, Karen Devine, Jean Utke, Boyana Norris, Bruce Hendrickson, Florin Dobrian, Mahantesh Halappanavar, several others

- Financial support : DOE, NSF

Further reading
www.cscapes.org

- Gebremedhin, Manne and Pothen. What color is your Jacobian? Graph coloring for computing derivatives. *SIAM Review 47(4):627—705, 2005.*
- Gebremedhin, Tarafdar, Manne and Pothen. New acyclic and star coloring algorithms with applications to computing Hessians. *SIAM J. Sci. Comput. 29:1042—1072, 2007.*
- Gebremedhin, Pothen and Walther. Exploiting sparsity in Jacobian computation via coloring and automatic differentiation: a case study in a Simulated Moving Bed process. *AD2008, LNCSE 64:339---349, 2008.*
- Gebremedhin, Pothen, Tarafdar and Walther. Efficient computation of sparse Hessians using coloring and Automatic Differentiation. *INFORMS Journal on Computing, 21:209---223, 2009.*
- Bozdag, Gebremedhin, Manne, Boman and Catalyurek. A framework for scalable greedy coloring on distributed-memory parallel computers. *J. Parallel Distrib. Comput. 68(4):515—535, 2008.*
- Gebremedhin, Nguyen, Patwary and Pothen. COLPACK: Graph Coloring Software for Derivative Computation and Beyond, *Submitted, Oct. 2010.*

CSCAPES Institute



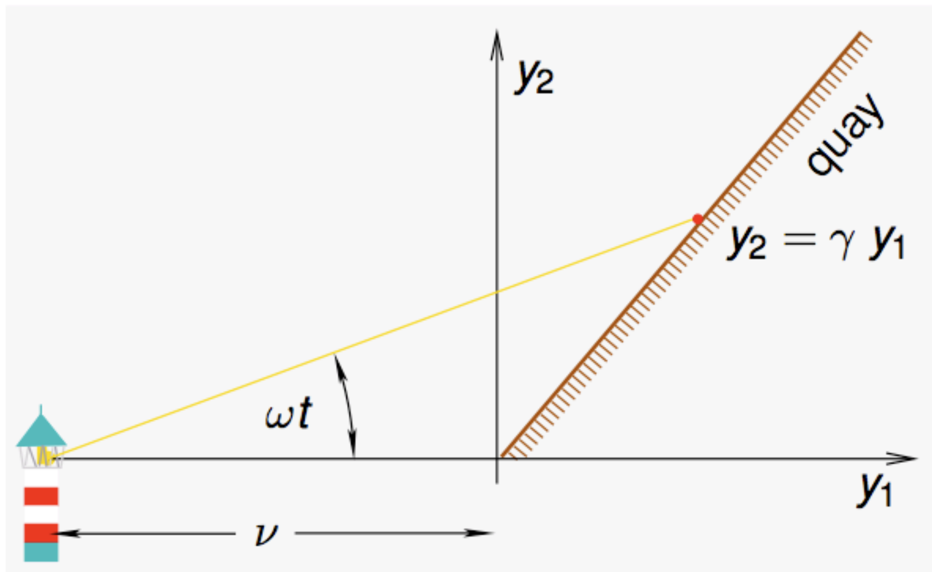
Combinatorial Scientific Computing and Petascale Simulations (CSCAPES) Institute

- One of four DOE Scientific Discovery thru Advanced Computing (SciDAC) Institutes; only one in Appl. Math
  - Excellence in research, education and training
  - Collaborations with science projects in SciDAC
- Focus not on specific application, but on algorithms and software for combinatorial problems
- Participants from Purdue, Sandia, Argonne, Ohio State, Colorado State
- CSCAPES workshops with talks, tutorials on software, discussions on collaborations
- www.cscapes.org

Extra slides

Illustrating Forward and Reverse

Example: Lighthouse



$$y_1 = \frac{\nu \tan(\omega\, t)}{\gamma - \tan(\omega\, t)} \quad \text{and} \quad y_2 = \frac{\gamma\, \nu \tan(\omega\, t)}{\gamma - \tan(\omega\, t)}$$

Evaluation Procedure (Lighthouse)

$$y_1 = \frac{\nu \tan(\omega t)}{\gamma - \tan(\omega t)}$$

$$y_2 = \frac{\gamma \nu \tan(\omega t)}{\gamma - \tan(\omega t)}$$

$$\rightarrow$$

$$\begin{array}{lll}
v_{-3} & = x_1 = \nu & \\
v_{-2} & = x_2 = \gamma & \\
v_{-1} & = x_3 = \omega & \\
v_0 & = x_4 = t & \\
\hline
v_1 & = v_{-1} * v_0 & \equiv \varphi_1(v_{-1}, v_0) \\
v_2 & = \tan(v_1) & \equiv \varphi_2(v_1) \\
v_3 & = v_{-2} - v_2 & \equiv \varphi_3(v_{-2}, v_2) \\
v_4 & = v_{-3} * v_2 & \equiv \varphi_4(v_{-3}, v_2) \\
v_5 & = v_4 / v_3 & \equiv \varphi_5(v_4, v_3) \\
v_6 & = v_5 * v_{-2} & \equiv \varphi_6(v_5, v_{-2}) \\
\hline
y_1 & = v_5 & \\
y_2 & = v_6 &
\end{array}$$

Function: $y = F(x)$

Derivatives: $F'(x)\dot{x}, \ \bar{y}^{\top} F'(x)$

Forward differentiation of lighthouse example

$$
\begin{array}{rcl|rcl}
v_{-3} & = & x_1 = \nu & \dot{v}_{-3} & \equiv & \dot{x}_1 = 0 \\
v_{-2} & = & x_2 = \gamma & \dot{v}_{-2} & \equiv & \dot{x}_2 = 0 \\
v_{-1} & = & x_3 = \omega & \dot{v}_{-1} & \equiv & \dot{x}_3 = 0 \\
v_0 & = & x_4 = t & \dot{v}_0 & \equiv & \dot{x}_4 = 1 \\
\hline
v_1 & = & v_{-1} * v_0 & \dot{v}_1 & = & \dot{v}_{-1} * v_0 + v_{-1} * \dot{v}_0 \\
v_2 & = & \tan(v_1) & \dot{v}_2 & = & \dot{v}_1 / \cos(v_1)^2 \\
v_3 & = & v_{-2} - v_2 & \dot{v}_3 & = & \dot{v}_{-2} - \dot{v}_2 \\
v_4 & = & v_{-3} * v_2 & \dot{v}_4 & = & \dot{v}_{-3} * v_2 + v_{-3} * \dot{v}_2 \\
v_5 & = & v_4 / v_3 & \dot{v}_5 & = & (\dot{v}_4 - \dot{v}_3 * v_5) * (1/v_3) \\
v_6 & = & v_5 * v_{-2} & \dot{v}_6 & = & \dot{v}_5 * v_{-2} + v_5 * \dot{v}_{-2} \\
\hline
y_1 & = & v_5 & \dot{y}_1 & = & \dot{v}_5 \\
y_2 & = & v_6 & \dot{y}_2 & = & \dot{v}_6
\end{array}
$$

General Tangent Procedure

$$
\begin{array}{lcll}
[v_{i-n}, \dot{v}_{i-n}] & = & [x_i, \dot{x}_i] & i = 1, \ldots, n \\
\hline
[v_i, \dot{v}_i] & = & [\varphi_i(u_i), \dot{\varphi}_i(u_i)] & i = 1, \ldots, l \\
\hline
[y_m, \dot{y}_m] & = & [v_{l-i}, \dot{v}_{l-i}]; & i = m-1, \ldots, 0
\end{array}
$$

with $u_i \equiv (v_j)_{j \prec i}$ and

$$\dot{\varphi}_i(u_i, \dot{u}_i) \equiv \varphi_i'(u_i)\dot{u}_i$$

Adjoint recursion of lighthouse-example

$$v_{-3} = x_1; \quad v_{-2} = x_2; \quad v_{-1} = x_3; \quad v_0 = x_4;$$
$$v_1 = v_{-1} * v_0;$$
$$v_2 = \tan(v_1);$$
$$v_3 = v_{-2} - v_2;$$
$$v_4 = v_{-3} * v_2;$$
$$v_5 = v_4 / v_3;$$
$$v_6 = v_5 * v_{-2};$$
$$y_1 = v_5; \quad y_2 = v_6;$$

$$\bar{v}_5 = \bar{y}_1; \quad \bar{v}_6 = \bar{y}_2;$$
$$\bar{v}_5 \mathrel{+}= \bar{v}_6 * v_{-2}; \quad \bar{v}_{-2} \mathrel{+}= \bar{v}_6 * v_5;$$
$$\bar{v}_4 \mathrel{+}= \bar{v}_5 / v_3; \quad \bar{v}_3 \mathrel{+}= \bar{v}_5 * v_5 / v_3;$$
$$\bar{v}_{-3} \mathrel{+}= \bar{v}_4 * v_2; \quad \bar{v}_2 \mathrel{+}= \bar{v}_4 * v_{-3};$$
$$\bar{v}_{-2} \mathrel{+}= \bar{v}_3; \quad \bar{v}_2 \mathrel{-}= \bar{v}_3;$$
$$\bar{v}_1 \mathrel{+}= \bar{v}_2 / \cos^2(v_1);$$
$$\bar{v}_{-1} \mathrel{+}= \bar{v}_1 * v_0; \quad \bar{v}_0 \mathrel{+}= \bar{v}_1 * v_{-1};$$
$$\bar{x}_4 = \bar{v}_0; \quad \bar{x}_3 = \bar{v}_{-1}; \quad \bar{x}_2 = \bar{v}_{-2}; \quad \bar{x}_1 = \bar{v}_{-3};$$

Incremental adjoint recursion

$\bar{v}_i$	$=$	0	$i = 1-n, \ldots, l$
v_{i-n}	$=$	x_i	$i = 1, \ldots, n$
v_i	$=$	$\varphi_i(v_j)_{j \prec i}$	$i = 1, \ldots, l$
y_{m-i}	$=$	v_{l-i}	$i = m-1, \ldots, 0$
$\bar{v}_{l-i}$	$=$	$\bar{y}_{m-i}$	$i = 0, \ldots, m-1$
$\bar{v}_j$	$+=$	$\bar{v}_i \frac{\partial}{\partial v_j} \varphi_i(u_i)$ for $j \prec i$	$i = l, \ldots, 1$
$\bar{x}_i$	$=$	$\bar{v}_{i-n}$	$i = n, \ldots, 1$

Forward mode of AD

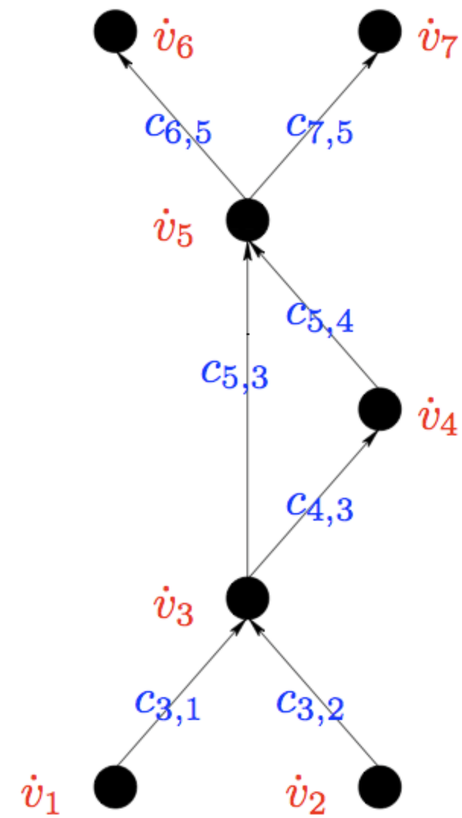
Propagation of directional derivatives

$$\dot{v}_j \; = \; \sum_{i \prec j} c_{j,i} \cdot \dot{v}_i \; = \; (f_j')^T \cdot (\dot{v}_i)_{i \prec j}$$

for $j = 1, \ldots, n + p + m$.

For example,

$$\dot{v}_5 \; = \; \dot{v}_3 \cdot c_{5,3} + \dot{v}_4 \cdot c_{5,4} \quad .$$



Reverse mode of AD

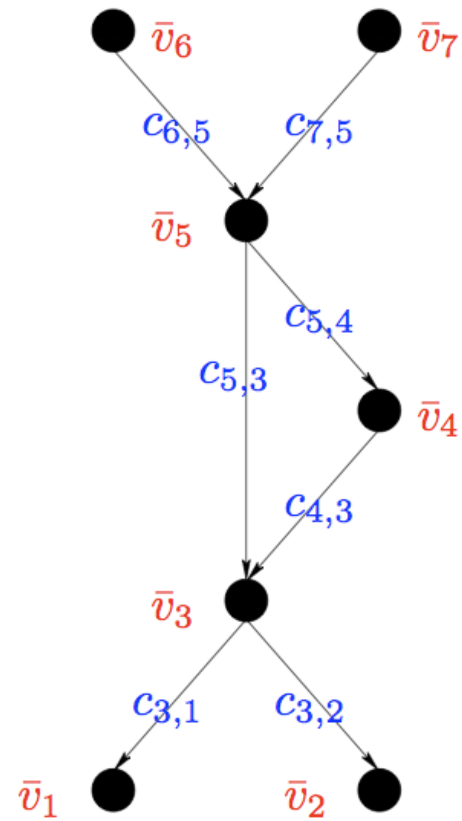
Propagation of adjoints

$$\bar{v}_j \;=\; \sum_{k:j\prec k} \bar{v}_k \cdot c_{k,j}$$

for $j = n + p + m, \ldots, 1.$

For example,

$$\bar{v}_3 \;=\; \bar{v}_5 \cdot c_{5,3} + \bar{v}_4 \cdot c_{4,3} \quad .$$



Theoretical complexity

forward mode:	$\mathrm{OPS}(F'(x)\dot{x})$	$\leq c_1\,\mathrm{OPS}(F), c_1$	$\in [2, 5/2]$
reverse mode:	$\mathrm{OPS}(\bar{y}^\top F'(x))$	$\leq c_2\,\mathrm{OPS}(F), c_2$	$\in [3, 4]$
	$\mathrm{MEM}(\bar{y}^\top F'(x))$	$\sim \mathrm{OPS}(F)$	
combination:	$\mathrm{OPS}(\bar{y}^\top F''(x)\dot{x})$	$\leq c_3\,\mathrm{OPS}(F), c_3$	$\in [7, 10]$

Ref: Griewank and Walther, Evaluating Derivatives, Second Edition, SIAM, 2008

Practical implications

- Jacobians of functions with small number of independent variables (Forward mode)
- Jacobians of functions with small number of dependent variables (Reverse mode)
- Jacobian-vector products (Forward mode)
- Transposed-Jacobian-vector products (Reverse mode)

- Hessian-vector product (Forward+Reverse mode)

- Large, sparse Jacobians and Hessians (Forward mode plus "compression")

PURDUE UNIVERSITY

Implementation

Overloading	Source Transformation
Introduction of *active* floating point type $(v, \dot{v})$ and overloading of elemental functions $v = \varphi(\mathbf{u})$ such that $$(v, \dot{v}) = \dot{\varphi}(\mathbf{u}, \dot{\mathbf{u}}).$$ For example, $$v = \cos(u)$$ becomes $$(v, \dot{v}) = (\cos(u), -\sin(u) \cdot \dot{u}).$$	$F \rightarrow$ 1. Lexical analysis 2. Syntax analysis 3. Semantic analysis 4. Static data flow analyses 5. AD 6. Code optimization 7. Unparsing $\rightarrow \dot{F}, \bar{F}, etc.$

cscapes

Implementation (cont'd)

Operator overloading

- Relatively easy to implement
- Robustness easy to achieve
- No compiler analysis/ optimization
- Adjoint computation requires "tape" interpretation
- Disadvantages fade away when computing higher derivatives

Source transformation

- Relatively hard to implement
- Robustness hard to achieve
- Static analyses and compiler optimization

Select AD tools

Tool	Language	Type	Mode	Organization	Remark
TAF	Fortran 95	ST	F and R	FastOpt	Commercial tool
Tapenade	Fortran 95	ST	F and R	INRIA	
OpenAD/F	Fortran 95	ST	F and R	Argonne/UC/Rice	Development driven by climate model and astrophysics code
ADIFOR	Fortran 77	ST	F	Rice/Argonne	• Mature tool • Hundreds of users
ADOL-C	C/C++	OL	F and R	Dresden	• Mature tool • Widely used • Supports higher order derivatives
ADIC	C	ST	F	Argonne/UC	• Shares infrastructure with OpenAD/F • RM under devt
TAC++	C and some C++	ST	F and R	FastOpt	Commercial tool (under development)
Adimat	Matlab	ST	F	Aachen	

For more info: http://www.autodiff.org

Select AD tools

Tool	Language	Type	Mode	Organization	Remark
TAF	Fortran 95	ST	F and R	FastOpt	Commercial tool
Tapenade	Fortran 95	ST	F and R	INRIA	
OpenAD/F	Fortran 95	ST	F and R	Argonne/UC/Rice	Development driven by climate model and astrophysics code
ADIFOR	Fortran 77	ST	F	Rice/Argonne	• Mature tool • Hundreds of users
ADOL-C	C/C++	OL	F and R	Dresden	• Mature tool • Widely used • Supports higher order derivatives
ADIC	C	ST	F	Argonne/UC	• Shares infrastructure with OpenAD/F • RM under devt
TAC++	C and some C++	ST	F and R	FastOpt	Commercial tool (under development)
Adimat	Matlab	ST	F	Aachen	

For more info: http://www.autodiff.org

ADOL-C: source code modification

- Include needed header-files
  easiest way: #include “adolc.h”
- Define region that has to be differentiated:

trace_on(tag,*keep*);	Start of
...	active section
trace_off(*file*);	and its end

- Mark independents and dependents in active section:

xa <<= xp;	mark and initialize independents
...	calculations
ya >>= yp;	mark dependents

- Declare all active variables of type adouble
- Calculate derivative objects after trace_off(*file*)

ADOL-C: easy-to-use routines

int gradient(tag,n,x[n],g[n]):

tag	= tape number
n	= # indeps
x[n]	= values of indeps
g[n]	= $\nabla f(x)$

int jacobian(tag,m,n,x[n],J[m][n]):

tag	= tape number
m	= # deps
n	= # indeps
x[n]	= values of indeps
J[m][n]	= $F'(x)$

int hessian(tag,n,x[n],H[n][n])

tag	= tape number
n	= # indeps
x[n]	= values of indeps
H[n][n]	= $\nabla^2 f(x)$

ADOL-C: additional drivers for nonlinear optimization

- vec_jac(tag, m, n, repeat, x[n], u[m], z[n])
  Computes $z = u^T F'(x)$
- jac_vec(tag, m, n, x[n], v[n], z[n])
  Computes $z = F'(x)\, v$
- hess_vec(tag, n, x[n], v[n], z[n])
  Computes $z = \nabla^2 f(x)\, v$
- lagra_hess_vec(tag, n, m, x[n], v[n], u[m], h[n])
  Computes $h = u^T F''(x)\, v$
  Extension to $u^T F''(x)\, V$ available
- jac_solv(tag, n, x[n], b[n], sparse, mode)
  Computes w with $F'(x)\, w = b$ and store result in b

...

ADOL-C: routines for sparse derivative computation

Jacobian:

```
sparse_jac(tag, m, n, repeat, x, &nnz, &row_ind, &col_ind, &values,
           options);
           jac_pat(tag, m, n, x, JP, options);
 generate_seed_jac(m, n, JP, &seed, &p, option);
```

Hessian:

```
sparse_hess(tag, n, repeat, x, &nnz, &row_ind, &col_ind, &values,
            options);
            hess_pat(tag, n, x, HP, options);
 generate_seed_hess(n, HP, &seed, &p, option);
```

For more info: http://www.math.tu-dresden.de/~adol-c