On the Use of Second Order Information in GAMS/CONOPT3

by

Arne Stolbjerg Drud

ARKI Consulting & Development A/S
Bagsvaerd
Denmark

Presented at

SIAM Optimization Meeting
May 2002
Toronto

Overview

Introduction:

- My Personal Interest in Nonlinear Optimization
- The Universe of Models and Methods
- On Multi-Method Solvers

The GRG Method

- The Basic Method
- Variations on the GRG Method
- The Methods used in CONOPT2
- The new SQP Method in CONOPT3

Some Results

Improving a Multi-Method Solver

My Personal Interest in Optimization

I am trying to convert theory and methods developed by many of my colleagues into software, that is useful to practical modelers.

Many of my user have neither knowledge nor appreciation of the many aspects of the theory of nonlinear optimization.

However, they are usually experts in their own field (economics, chemistry, engineering etc) and they can appreciate the results they get.

My interest is therefore not in proofs of global convergence or of superlinear or quadratic convergence, but in robust implementations of large scale model components and in their practical use.

In other words: My interest is to push the frontiers for practical use of NLP using many small pushes.

This talk with describe one of these small pushes.

Note: Although global optimality is desirable, I will only talk about locally optimal solutions.

The Universe of Models

The models users attempt to solve (or dream of solving) can be characterized by many partly independent attributes:

- Size (from a few variables and constraints into the millions)
- Degree of nonlinearity (from linear constraints and nonlinear objective over a few nonlinear constraints to most constraints being very nonlinear)
- Degrees of freedom (or number of superbasics) (from none in the optimal solution to hundreds and thousands)
- Source of difficulty (nonlinearity of the constraints, combinatorial difficulties related to active inequalities, size).

Existing methods are often designed and appropriate for only a subset of this universe. SLP is for example usually excellent for models with few degrees of freedom, many inequalities, and fairly mild nonlinearities, independent of the size.

Although SQP can solve these models and have superior theoretical properties, it is often an overkill and it can be much slower than simpler approaches.

Multi-Method Solvers

Solver = Software package
Method = Algorithm / Theory

We know from experience that to cover a large part of the Universe of Model we need several methods.

Some Solvers have therefore implementations of several Methods. The selection of the appropriate method for a particular method can be done according to (at least) three paradigms:

1. User selection.
2. Automatic A Priori Selection based on model characteristics.
3. Automatic Dynamic Selection based on model characteristics and information on the local behavior of the optimization process.

Dynamic Selection assumes that switching between methods is relatively cheap, preferably that the methods use the same or similar data structures.

CONOPT is a Multi-Method Solver with Automatic Dynamic Selection of Methods.

The GRG Method

The GRG Method is a feasible path method based on the following steps:

1. Start with a Feasible Point.
2. Compute the Jacobian of the constraints. It defines the tangent space of the linearized model.
3. Select a Basis from the columns of the Jacobian and invert (factorize) it.
4. Use the inverse Basis to compute a Reduced Gradient.
5. Test for optimality.
6. Use the Reduced Gradient to find an improving directions in the tangent space.
7. Perform a linesearch to find the next solution. During the linesearch, use the inverse Basis to restore feasibility using Newton-like iterations.
8. Go to 2 and repeat.

Variations on the GRG Method (included in CONOPT2)

Steepest Descend Method: The search direction in the non-basic space is selected as the (negative) reduced gradient. It is usually slow, but it contributes a robust component and it is the basis of many proofs of convergence.

Quasi-Newton Method: The search direction in the non-basic space is the reduced gradient multiplied by an approximated Hessian. Has usually better convergence than Steepest Descend but needs more memory.

Linear-Mode Iterations: When bounds become active after very small movements (a combinatorial problem that slows down both Steepest Descend and Quasi-Newton), we use the same Jacobian for several iterations. The inverse Basis is updated using LP-like methods. Each iteration is much cheaper.

SLP: The computation of the search direction takes into account both the tangent space and its intersection with the box defined by bounds by solving an LP approximately. Both search direction and basis is changed in the process. Works well for models with few nonlinearities and a combinatorial nature.

Common Properties of all Variations

Each iteration start with a feasible point and ends with a feasible point.

At the start of an iteration, a basis is used to find a reduced gradient (and to check for optimality).

After the search direction has been computed, a basis (and an inverse) is available for use in the Newton-like iterations of the linesearch.

The main difference between the variations is the method used to find the search direction and the amount of work and information each variation use.

It is therefore relatively easy and cheap to switch between the variations. When switching to the Quasi-Newton variation it is necessary to initialize the estimate of the Hessian. There is a little direct cost, but progress may be slowed down for some iterations.

A Common Problem for all Variations

No matter how good the search direction, it is sometimes not possible to take the optimal step because the Newton-like iterations used to restore feasibility do not converge.

There is no reason to spend effort on a good search direction if only a small part of it can be used.

CONOPT2 has some heuristics intended to overcome this problem:

- Change the basis to make it "better conditioned"
- Use Steepest Descend
- Use methods based on Steepest Edge to scale the search direction.

NOTE: Although the feasibility restoration does give some problems for the GRG method it is also of great help. No Merrit Function, SLP and SQP models are always feasible etc.

The SQP Component of CONOPT3

Background:

CONOPT2 works very poorly on models with many degrees of freedom (500 and up). The Quasi-Newton method will only work if we have space for a (reduced) Hessian and the automatically selected alternative is Steepest Descend. If the Quasi-Newton method is forced in use it is often very slow for these large models.

CONOPT2 works but it converges very slowly on models with more than 100-200 degrees of freedom and a combinatorial nature. Each time the set of active and inactive inequalities changes the reduced Hessian changes as well. This stalls the progress, and several iterations are often needed to build up new second order information before good progress is re-established.

The New SQP Component of CONOPT3

Input Requirement – Second Order Information:
EITHER a routine that can compute the Hessian of the Lagrangian as a sparse matrix OR a routine that can compute the Hessian of the Lagrangian multiplied by a vector or BOTH.

The standard SQP model (with Lagrange Multipliers implied by the GRG method and the input basis) is solved approximately, starting from a feasible point.

When the number of Superbasic variables is large (default > 500) then a Conjugate Gradient (CG) or Scaled Conjugate Gradient (SCG) method is combined with a Reduced Gradient method. The CG method is restarted each time a variable reaches a bound.

When the number of Superbasic variables is small a Reduced Hessian approximation is used as a pre-conditioner to the CG method. The Reduced Hessian approximation is maintained and updated using methods from Quasi Newton, and it is adjusted when a variable reaches a bound and when the basis is changed.

The SQP Component of CONOPT3 (cont)

Stopping criteria:

The amount of resources allocated to the SQP is determined based on past performance. Initially, only a few iterations and a short move is allowed. If progress is good, more iterations and a larger move is allowed. If progress is poor, fewer iterations and a smaller move is allowed.

The Progress measure depends on:

- Could the predicted step be made or was the linesearch limited by problems with feasibility in the Newton iterations.
- Was the change in objective as expected.
- Was the change in reduced gradient as expected.

Missing Component:

What should be done when the steplength is less than one due to higher order terms. A proximity term that bends towards steepest descend / steepest edge should be added.

The SQP Component of CONOPT3 (cont)

Implementation of the CG methods:

If the expected number of iterations is small (default < 10), then the CG iterations are based on the Hessian-Vector product routine (if available).

If the expected number of iterations is large, then the CG iterations are based on the Hessian as a sparse matrix (if available).

The CG iterations take place in the space of superbasic variables and the Hessian is defined in the space of all variables. It is therefore necessary with repeated transformations forth and back. The transformation from the superbasic space takes place with a tangent computing routine, and the transformation back takes place with a routine similar to pricing.

When the Hessian of the Lagrangian is available as a sparse matrix CONOPT3 can compute a scaling pre-conditioner based on the diagonal of the Reduced Hessian. This pre-conditioner can be updated in connection with basis changes inside the SQP.

Other New Components of CONOPT3

- An Automatic Scaling Algorithm, that seems to work, i.e. it is in general better than no scaling. CONOPT2 has one, but it is not good.

- A new factorization engine that work much faster for models with more than 10000 constraints. It has been used on models with over 1 millions constraints.

CONOPT2 vs. CONOPT3 for some NLP models

Example 1: A version of Alan Manne's RUNS model

The model has 4145 variables and 3428 constraints
The Jacobian has 14905 Jacobian elements, 263 of which are nonlinear.
The Hessian of the Lagrangian has 263 elements on the diagonal, 135 elements below the diagonal, and there are 263 nonlinear variables.

MIXED type model	CONOPT2	CONOPT3
Easy case: 1. Solve (cold start)	7590 iterations 284 seconds	514 iterations 63 seconds
Easy case 2. Solve (hot start)	930 iterations 41 seconds	54 iterations 6 seconds
Hard case: 1. Solve (cold start)	41532 iterations 3109 seconds	997 iterations 414 seconds
Hard case: 2. Solve (hot start)	1813 seconds 176 seconds	79 iterations 44 seconds

CONOPT2 vs. CONOPT3 for some NLP models

Example 2: An entropy maximization model of Sherman Robinson (SAM Balancing)

The model has 5048 variables and 2947 constraints
The Jacobian has 19275 elements, 6631 of which are nonlinear.
The Hessian of the Lagrangian has 2657 elements on the diagonal, 1987 elements below the diagonal, and there are 2790 nonlinear variables.

The model has around 2100 superbasic variables and CONOPT2 could not solve it properly (terminated on slow convergence).
CONOPT3 used CG, scaled with the diagonal of the reduced Hessian.

MIXED/CURVED type model	CONOPT2	CONOPT3
Base case	19527 iterations 13758 seconds	183 iterations 56 seconds

Results using Solver Comparison Utilities from http://www.gamsworld.org/performance/index.htm

Based on the COPS models in GAMS Model Library

Solver Square Comparison: Considers all models.

Compares all solver return outcomes (for example optimal, infeasible, unbounded, interrupt, fail) of one solver with all return outcomes of another solver. Interrupt denotes resource or iteration limit has been reached. Solver CONOPT2 is represented on the left (rows) and solver CONOPT3 on top (columns). See the solver return definitions for return codes.

Tracefile 1 :	copstest.co2
Tracefile 2 :	copstest.co3
Solvers used :	CONOPT2
	CONOPT3

	optimal	**infeasible**	**unbounded**	**interrupt (time or iter.)**	**fail**	**total CONOPT2**
Optimal	53	-	-	-	-	53
Infeasible	-	-	-	-	-	-
Unbounded	-	-	-	-	-	-
Interrupt (time or iter.)	2	-	-	-	-	2
Fail	13	-	-	-	-	13
total CONOPT3	**68**	-	-	-	-	**68**

Results using Solver Comparison Utilities from http://www.gamsworld.org/performance/index.htm

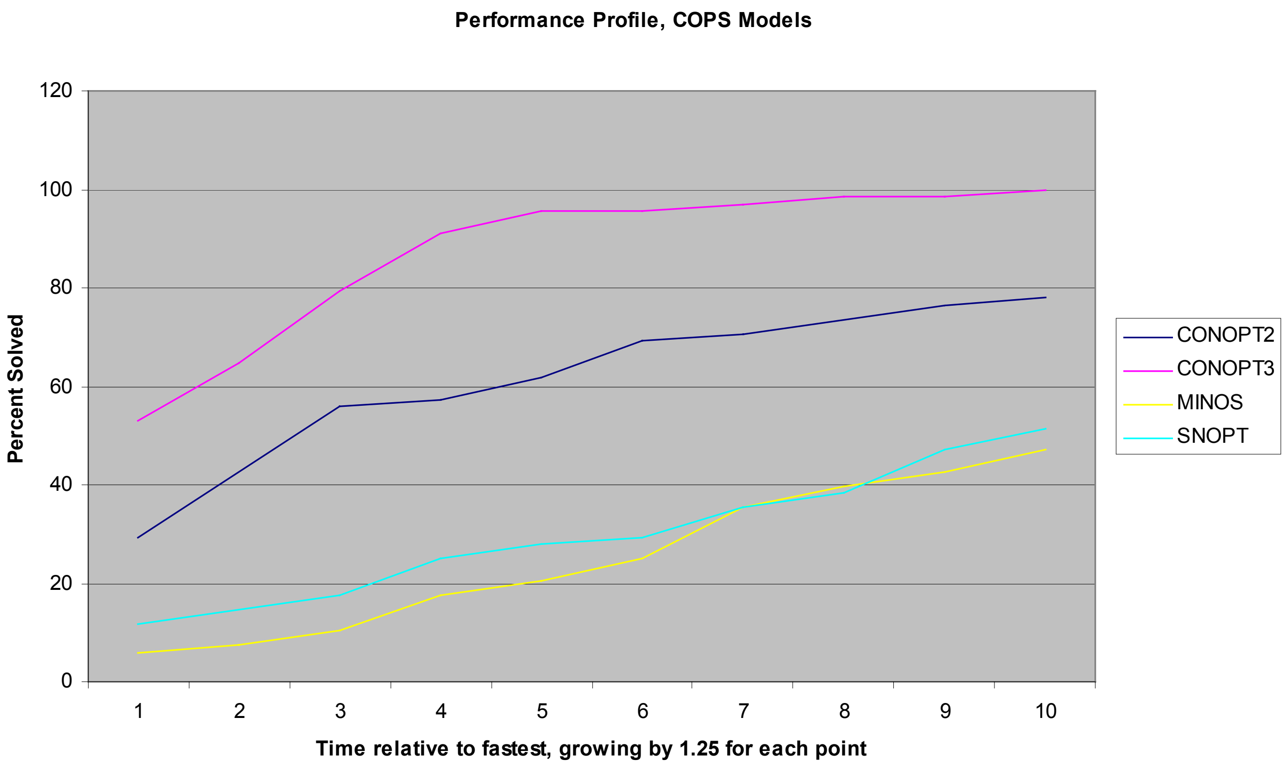
Based on the COPS models in GAMS Model Library

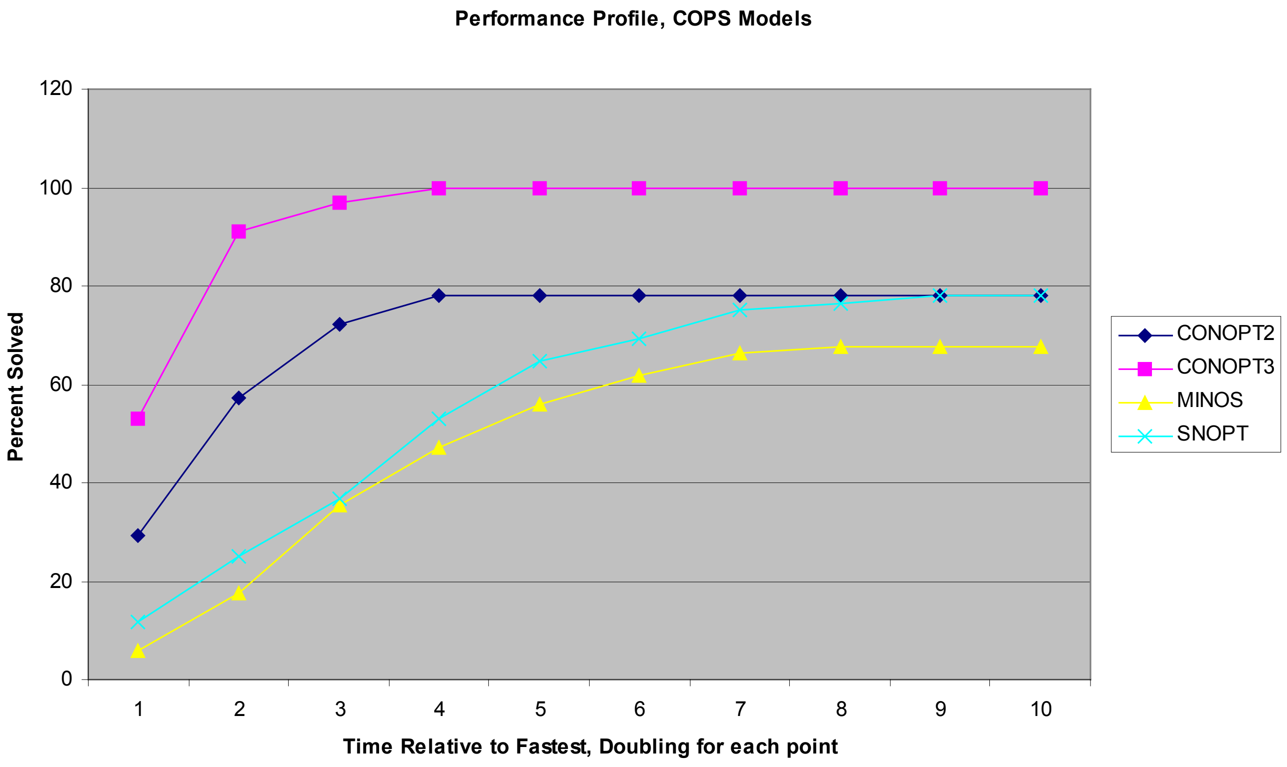
Total resource time for all models.

Tracefile 1 :	copstest.co2
Tracefile 2 :	copstest.co3
Solvers used :	CONOPT2
	CONOPT3

	Total	Obj CONOPT2 better	Obj same	Obj CONOPT3 better
Solver CONOPT2 infinitely faster :	-	-	-	-
Solver CONOPT2 much faster :	1	-	1	-
Solver CONOPT2 faster :	17	-	17	-
Solvers perform the same :	11	-	9	2
Solver CONOPT3 faster :	17	-	14	3
Solver CONOPT3 much faster :	7	-	7	-
Solver CONOPT3 infinitely faster :	15	-	-	15
Both solvers failed to solve optimally :	-	-	-	-
Total models: :	**68**	**0**	**48**	**20**

Solver Comparison Criteria: See the Performance web-page mentioned above.


Performance Profile, COPS Models
Percent Solved
0
20
40
60
80
100
120
1
2
3
4
5
6
7
8
9
10
Time relative to fastest, growing by 1.25 for each point
CONOPT2
CONOPT3
MINOS
SNOPT


Performance Profile, COPS Models
Percent Solved
0
20
40
60
80
100
120
1
2
3
4
5
6
7
8
9
10
Time Relative to Fastest, Doubling for each point
CONOPT2
CONOPT3
MINOS
SNOPT

Improving a Multi-Method Solver

There are a number of ways to improve a Multi-Method Solver like CONOPT3:

- Improve the common computational engines such as linear algebra / factorizations. All components and the overall solver become faster.
- Improve some of the individual methods. Will help for models that use this sub-method heavily.
- Add a new Method that concentrates on uncovered models from the Model Universe. This may increase the set of models that the solver can attack successfully.
- Add a component for difficult models. This may increase the overall reliability of the solver even if the new method only used to get past a bad spot.
- Improve the logic for dynamic switching between methods. The challenge is to avoid expensive methods when cheap ones will do.

Improving a Multi-Method Solver (cont.)

I see great potential in the lasts two points.

However, there is a lack of a theoretical basis for improving the dynamic switching. It is necessary to develop models that describe the solution progress as a function of various methods and decision parameters, and it is necessary to work on how observations of the solution process can be used to calibrate such models.

If this is done well, I am convinced that for a given set of methods we can improve the overall speed and reliability considerably.