A Triangular-Based Branch and Bound Method for Nonconvex Quadratic Programming and the Computational Grid

JEFF LINDEROTH

Industrial and Systems Engineering
Lehigh University
jtl3@lehigh.edu

How This Talk Came to Be...

January 9, 2003

‘‘We’re thinking of writing an NSF proposal. What are you working on these days, Jeff?’’

‘‘I have begun preliminary work on a branch-and-bound method for a global optimization problem that relies on (convex) quadratic relaxations. Having a simple API to be able to build the nonlinear relaxations on the fly during the branch-and-bound process would be something very useful for this problem’’

The Fundamental Theorem of `email`

Theorem 1.

Mentioning a topic off-handedly in email about a subject you are planning on pursuing in research does *not* make you an expert in the field.

Theorem 2.

Mentioning by email that you “have begun preliminary work” on a subject doesn’t mean that you will have anything useful to say about that subject in nine months

Proofs. (By picture)

Q.E.D.

Jeff’s Main Summer Activities

Golf

Feeding New Son Jacob

⋆ Parallel B&B for (non)convex QCQP *not* a top summer priority

Outline

- Nonconvex Quadratic Programming
  - ◇ Relaxations with convex/concave envelopes of bilinear functions
  - ◇ Formulae for envelopes over triangles
  - ◇ Why triangles are good
  - ◇ How *not* to solve the resulting relaxations
- The Computational Grid
  - ◇ Brief Introduction
  - ◇ Branch-and-bound on the Computational Grid
    - ◇ The Quadratic Assignment Problem
  - ◇ Special challenges for branch-and-bound methods for non-discrete problems on the computational grid

(Nonconvex) QCQP

$$\min_{x \in \Re^n} q_0(x)$$

subject to

$$\begin{array}{rcll} q_k(x) & \geq & b_k & \forall k \in \mathcal{I} \\ q_k(x) & = & b_k & \forall k \in \mathcal{E} \\ x & \leq & u & \\ x & \geq & l & \end{array}$$

where

$$q_k(x) = (c^k)^T x + x^T Q^k x \qquad \forall k \in \{0 \cup \mathcal{I} \cup \mathcal{E}\}$$

- l and u are finite
- $q_k(x)$ could be convex, concave, or nonconvex

Caution

- I'm certainly not an expert in this area.
- I *do* know that these problems are *very hard* from a computational standpoint
  - ◇ QCQP generalizes integer programming and lots of other hard problems.
  - ◇ Problems with a tens of variables (or tens of quadratic terms) are about the limit of what can be solved

⋆ Solving these problems may require a large amount of computing resources—The computational grid!

Solving QCQP

- Popular (best?) method is to use convex and concave envelopes.
- Consider quadratic term $x_i x_j$, for $(x_i, x_j) \in \Omega \equiv [l_i, u_i] \times [l_j, u_j]$.
  - ◇ $x_i x_j \geq \max\{l_i x_j + l_j x_i - l_i l_j, u_i x_j + u_j x_i - u_i u_j\}$
  - ◇ $x_i x_j \leq \min\{l_i x_j + u_j x_i - l_i u_j, u_i x_j + l_j x_i - u_i l_j\}$
- These functions are (resp.) the convex and concave envelope of the function $x_i x_j$ over $[l_i, u_i] \times [l_j, u_j]$. (McCormick ’76, Al-Khayyal and Falk, ’83)
- $\text{vex}_\Omega(f)$—*Convex Envelope* of f over Ω—Pointwise supremum of convex underestimators of f over Ω.
- $\text{cav}_\Omega(f)$—*Concave Envelope* of f over Ω—Pointwise infimum of concave overestimators of f over Ω.

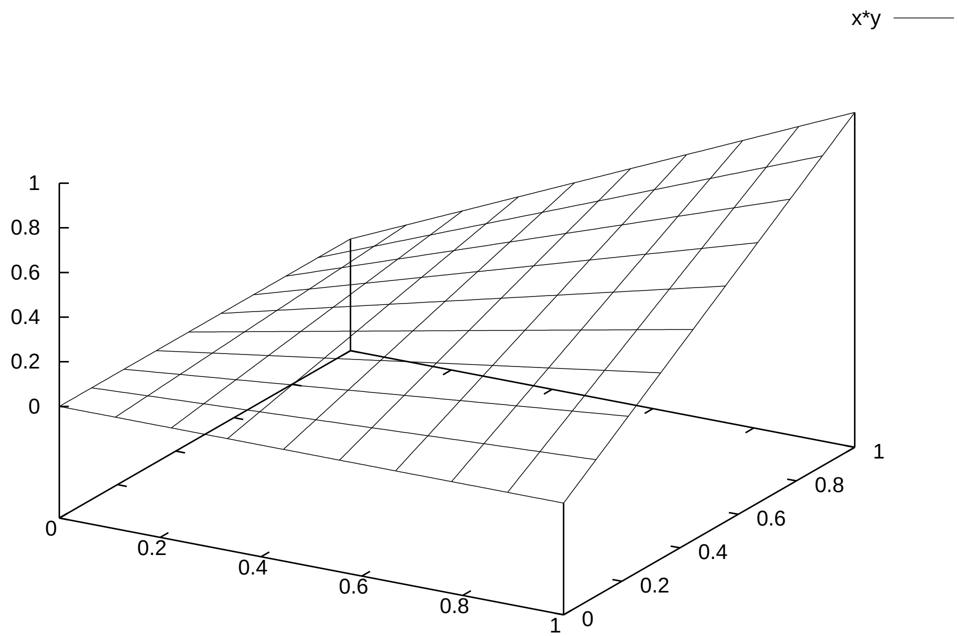
(LP) Relaxation of QCQP

$$\min_{l \le x \le u} \sum_{i=1}^{n} c_i^0 x_i + \sum_{i=1}^{n} \sum_{j=1}^{n} Q_{ij}^0 z_{ij}$$

subject to

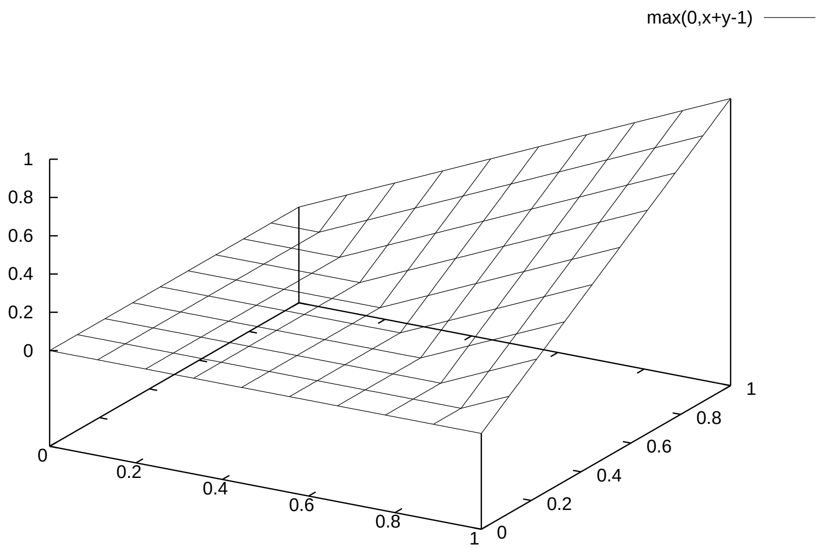
$$\begin{array}{rcll}
\sum_{i=1}^{n} c_i^k x_i + \sum_{i=1}^{n} \sum_{j=1}^{n} Q_{ij}^k z_{ij} & \ge & b_k & \forall k \in \mathcal{I} \\
\sum_{i=1}^{n} c_i^k x_i + \sum_{i=1}^{n} \sum_{j=1}^{n} Q_{ij}^k z_{ij} & = & b_k & \forall k \in \mathcal{E} \\
z_{ij} - l_i x_j - l_j x_i + l_i l_j & \ge & 0 & \forall i = 1, \ldots, n, j = 1, \ldots, n \\
z_{ij} - u_i x_j - u_j x_i + u_i u_j & \ge & 0 & \forall i = 1, \ldots, n, j = 1, \ldots, n \\
z_{ij} - l_i x_j - u_j x_i + l_i u_j & \le & 0 & \forall i = 1, \ldots, n, j = 1, \ldots, n \\
z_{ij} - u_i x_j - l_j x_i + u_i l_j & \le & 0 & \forall i = 1, \ldots, n, j = 1, \ldots, n
\end{array}$$

Worth 1000 Words?—Part I

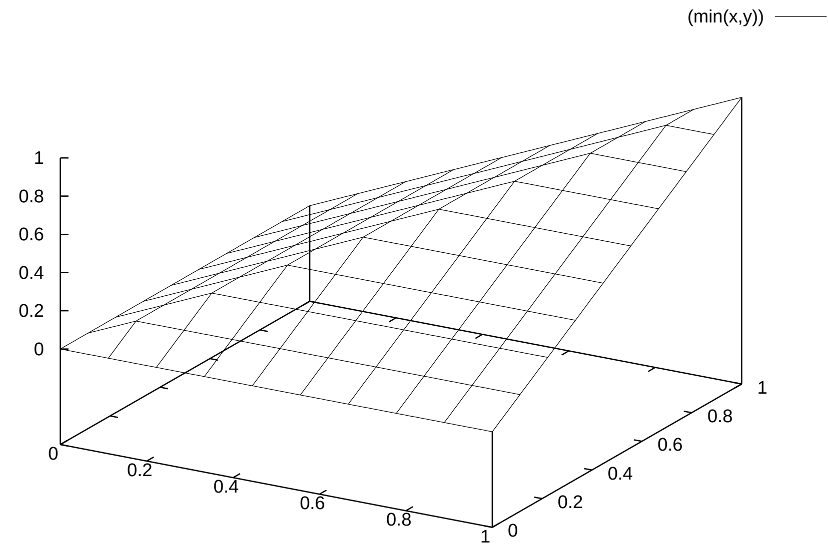


$$x_i x_j$$

Worth 1000 Words?—Part II



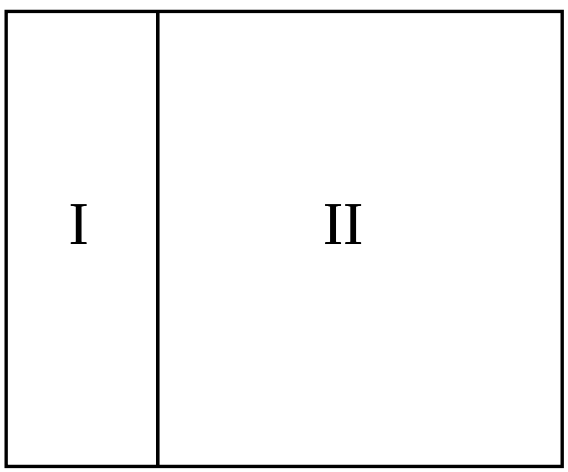
$\text{vex}(x_i x_j)$

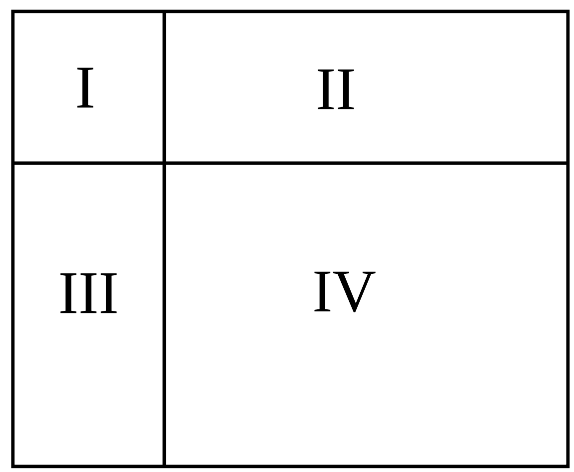


$\text{cav}(x_i x_j)$

Branching

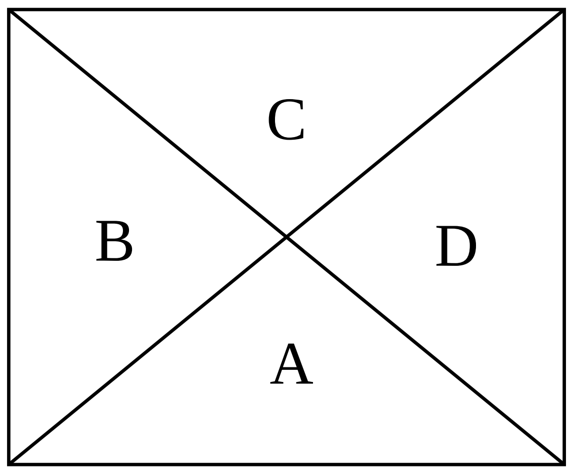
- In LP relaxation, $z_{ij} = x_i x_j \;\; \forall x_i, x_j \in \partial\Omega$.
- If $z_{ij} \neq x_i x_j$, we branch. Two suggested branching schemes





Triangle-Based Branching

- I'd like to propose a *triangular-based* branching scheme...



- In order to do this, we need formulae for $\text{cav}_{A,B,C,D}(x_i x_j)$ and $\text{vex}_{A,B,C,D}(x_i x_j)$

Concave Envelope Formulae

$$\begin{aligned}
\text{Let } AB &= \Omega \cap \{(x_i, x_j) | x_j - u_j \le \frac{l_j - u_j}{u_i - l_i}(x_i + l_i)\} \\
\text{Let } CD &= \Omega \cap \{(x_i, x_j) | x_j - u_j \ge \frac{l_j - u_j}{u_i - l_i}(x_i + l_i)\}
\end{aligned}$$

$$\text{Cav}_{AB}(x_i x_j) = \begin{cases} l_i l_j & \text{if } x_i = l_i, x_j = l_j \\ \frac{c_0 + c_i x_i + c_j x_j + c_{ij} x_i x_j + c_{i2} x_i^2 + c_{j2} x_j^2}{d_0 + d_i x_i + d_j x_j} & \text{Otherwise} \end{cases}$$

$$\text{Cav}_{CD}(x_i x_j) = \begin{cases} u_i u_j & \text{if } x_i = u_i, x_j = u_j \\ \frac{c_0 + c_i x_i + c_j x_j + c_{ij} x_i x_j + c_{i2} x_i^2 + c_{j2} x_j^2}{d_0 + d_i x_i + d_j x_j} & \text{Otherwise} \end{cases}$$

Messy Definitions for Completeness

Coef.	cav_{AB}	cav_{CD}
c_0	$-l_i^2 l_j^2 + l_i l_j u_i u_j$	$u_i^2 u_j^2 - l_i l_j u_i u_j$
c_i	$-l_i l_j u_j - l_j u_i u_j + 2 l_j^2 l_i$	$-2 u_j^2 u_i + l_j u_i u_j + l_i l_j u_j$
c_j	$-l_i l_j u_i - l_i u_i u_j + 2 l_i^2 l_j$	$-2 u_i^2 u_j + l_i u_i u_j + l_i l_j u_i$
c_{ij}	$u_i u_j - l_i l_j$	$u_i u_j - l_i l_j$
c_{i^2}	$l_j u_j - l_j^2$	$u_j^2 - l_j u_j$
c_{j^2}	$l_i u_i - l_i^2$	$u_i^2 - l_i u_i$
d_0	$-l_i u_j - u_i l_j + 2 l_i l_j$	$-2 u_i u_j + l_i u_j + l_j u_i$
d_i	$u_j - l_j$	$u_j - l_j$
d_j	$u_i - l_i$	$u_i - l_i$

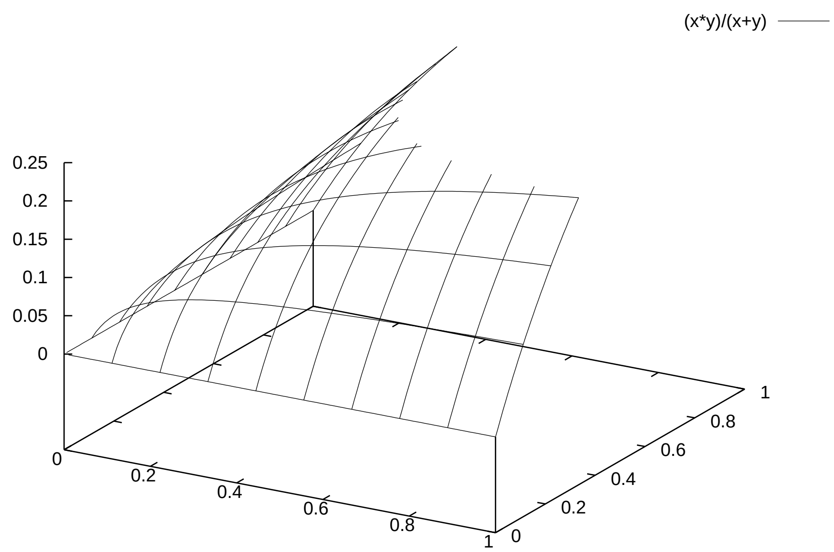
Now Vex

- You can likewise derive formulae for $\text{vex}_{BC}(x_i x_j)$ and $\text{vex}_{AD}(x_i x_j)$
- I won't bore you with the formulae. For $\Omega = [0,1] \times [0,1]$,

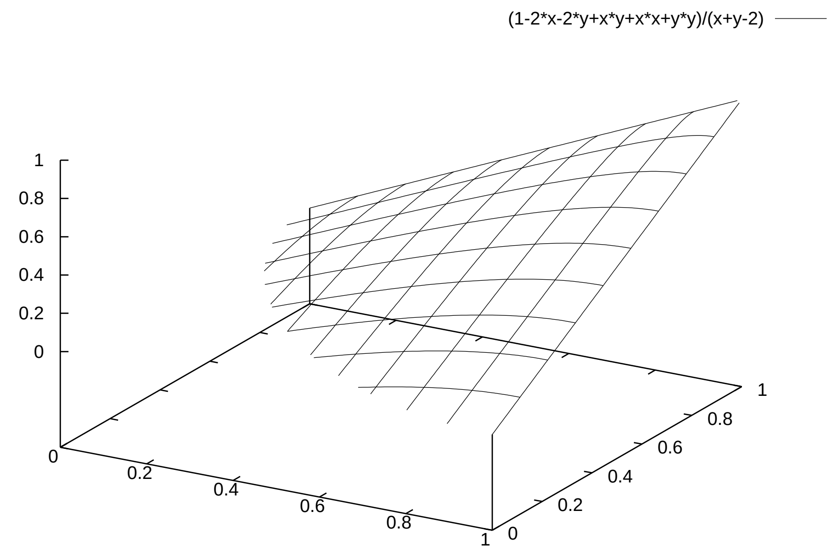
$$\text{vex}_{BC}(x_i x_j) = \frac{x_i^2}{x_i - x_j + 1}$$

$$\text{vex}_{AD}(x_i x_j) = \frac{x_j^2}{-x_i + x_j + 1}$$

cav Pics



$\mathrm{cav}_{AB}(x_i x_j)$



$\mathrm{cav}_{CD}(x_i x_j)$

This Just In...

- Recall, I said I was not an expert...
- The convex envelope formulae appeared implictly in [Sherali and Alameddine '90].
- They said they were planning on developing an algorithm using these results, but I don't think they ever did.

⋆ I claim that this would be a very good idea.

Why Triangles Are Good

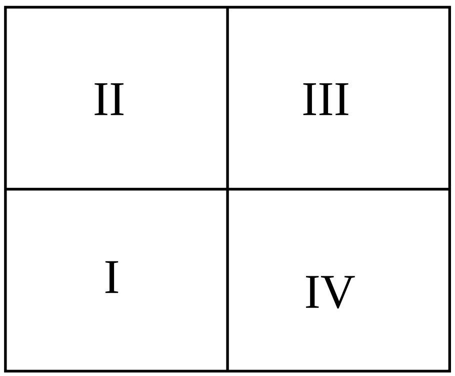
- Just like integer programming (and maybe even more so), a relaxation is good if it is tight.

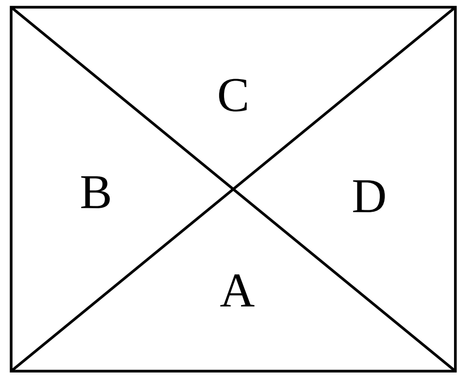
⋆ In this case, we can explicity calculate a meaningful measure of *relaxation goodness* (η) over an arbitrary region Γ.

$$\eta_\Gamma = \int_\Gamma (\text{cav}_\Gamma(x_i x_j) - \text{vex}_\Gamma(x_i x_j)) dx_i dx_j.$$

Branching Schemes

- For Example: $(x_i, x_j) \in [0,2] \times [0,2]$. Consider two branching schemes...





$$\begin{aligned} \eta_{[0,2]\times[0,2]} &= 8/3 \\ \eta_{\text{Rectangle}} = \eta_I + \eta_{II} + \eta_{III} + \eta_{IV} &= 2/3 \\ \eta_{\text{Triangle}} = \eta_A + \eta_B + \eta_C + \eta_D &= 4/9 \end{aligned}$$

- A branch-and-bound algorithm based on triangular subdivisions may be quite good!

Barriers to Triangular B&B Algorithm

- How to (easily, at least for prototyping purposes) interface B&B C++ driver code with existing NLP software to solve relaxations?

⋆ COIN to the rescue!

  ◇ NLPAPI (a very recent addition to COIN) is a C API to NLP software.

    - Lancelot
    - IPOPT—Very, very, very recently (like three days ago)

- This is great, but there is a more fundamental barrier to using NLP in a B&B algorithm...

NLP Stinks!

- NLP is quite slow.
  - ◇ This is largely a function of NLPAPI/Lancelot
  - ◇ The entire problem is built from scratch every time, writing out SIF files, before calling Lancelot
- NLP is sometimes wrong(!?!?!)
  - The envelope functions are not differentiable everywhere on the boundary.
  - They have the “wrong” curvature outside of the region of interest
  - NLP sometime says, “I don’t think your problem has a feasible solution, but I’m not too sure.”

It's Probably My Fault

- NLP doesn't stink. I just couldn't resist putting up that slide.
- It's the wrong hammer for the job.
- The envelope functions I presented have a second-order cone representation.
  - ◇ Thanks go to Kurt Anstreicher for making me believe that there really was a SOC representation of the envelope functions
  - ◇ Thanks go to Masakazu Muramatsu for showing me how these things work.

Ice Cream Cone (Symmetric Cone) Programming

$$\min\{c^T x | Ax = b, x \in \mathcal{K}\}$$

- $\mathcal{K} \subset \Re^n$ is a symmetric cone
- Quadratic cone in $\Re^n$:

$$\mathcal{K}_q^n = \left\{ x \in \Re^n : x_1 \geq \sqrt{\sum_{i=2}^{n} x_i^2} \right\}$$

- SOCP has a nice duality theory — It can tell me (with confidence) that a problem is infeasible
- SOCP solvers are robust
- I think it should reasonable to embed a SOCP (or even an SDP) solver into a branch and bound algorithm.

SOC Representation (Example)

- Imagine $\Omega = [0,1] \times [0,1]$
- Restrict $(x_i, x_j) \in B \equiv \{(x_i, x_j) | x_i \leq x_j, x_i + x_j \leq 1\}$

  $\Rightarrow$ $z_{ij} \geq \frac{x_i^2}{x_i - x_j + 1}$, $z_{ij} \leq \frac{xy}{x+y}$

$$z_{ij} \geq \frac{x_i^2}{x_i - x_j + 1}, (x_i, x_j) \in B \quad \Leftrightarrow \quad \begin{bmatrix} z_{ij} + 1 - x_j + x_i \\ 2x_i \\ z_{ij} - 1 + x_j - x_i \end{bmatrix} \in \mathcal{K}_q^3$$

$$z_{ij} \leq \frac{xy}{x+y}, (x_i, x_j) \in B \quad \Leftrightarrow \quad \begin{bmatrix} 2x_i + x_j - z_{ij} \\ 2x_i \\ -x_j - z_{ij} \end{bmatrix} \in \mathcal{K}_q^3$$

Wake Up!

- I am going to start talking about “The Grid”—Probably a more interesting topic

The Computational Grid

``A Grid is a hardware and software infrastructure that provides dependable, consistent, and pervasive access to resources to enable sharing of computational resources''

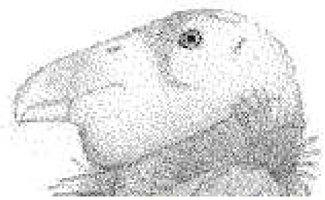
- Analogy is to power grid
  - Computational resources are ubiquitous
  - Their use could/should be transparent to the user

Building a Grid

- There have been *lots* of software tools that provide necessary grid services...
  - ◇ Resource scheduling
  - ◇ Fault-detection
  - ◇ Remote execution
- One problem remains: GREED!
  - ◇ Most people don't want to contribute "their" machine!

⋆ Condor is used to build the Grid!

What is Condor?



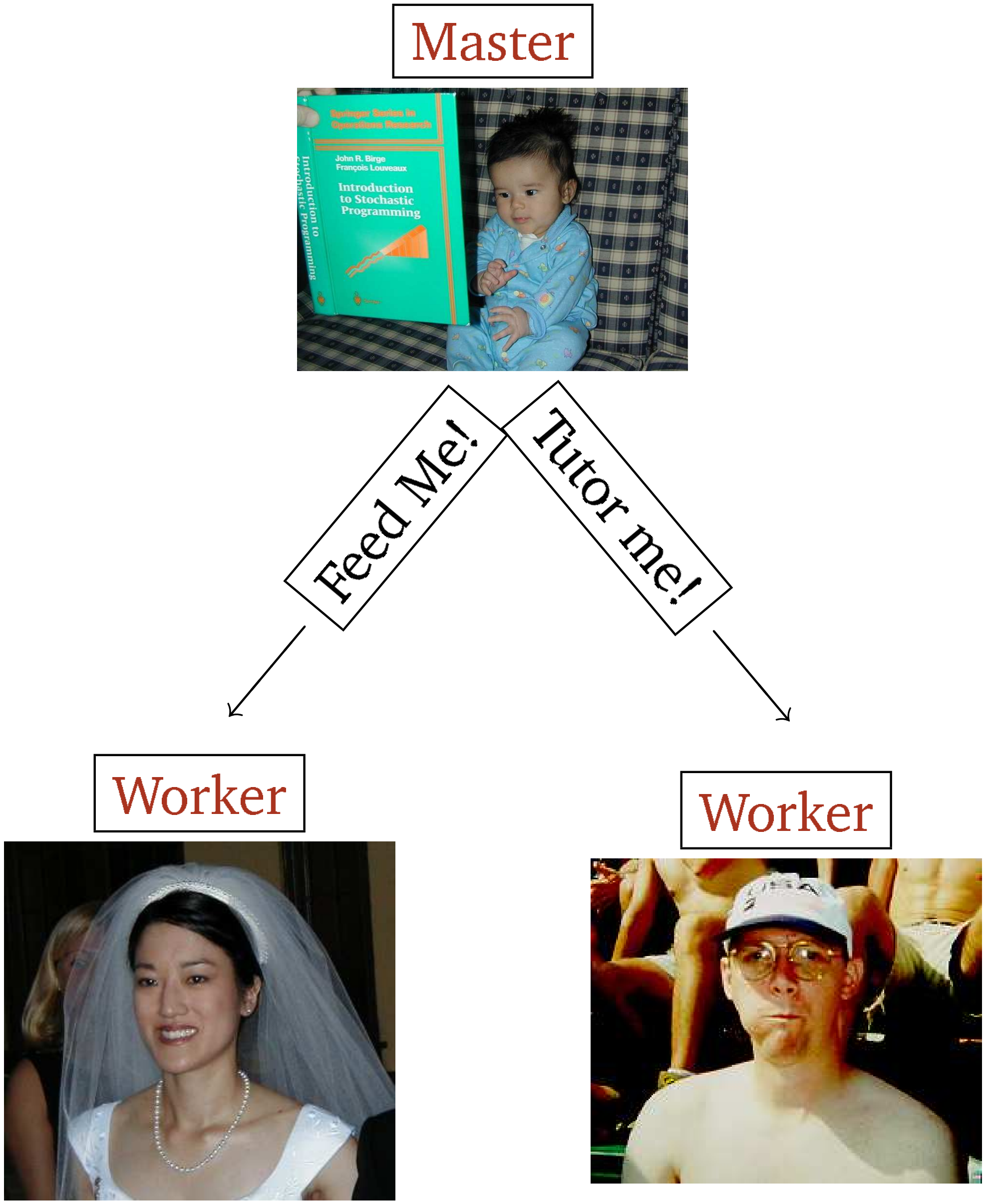
- Manages collections of “distributively owned” workstations
  - User need not have an account or access to the machine
  - Workstation owner specifies conditions under which jobs are allowed to run—Jobs must vacate when user claims machine!
  - All jobs are scheduled and “fairly” allocated among the pool
- How does it do this?
  - Scheduling/Matchmaking
  - Jobs can be checkpointed and migrated
  - Remote system calls provide the originating machines environment

Grid-Enabled B&B

- Condor gives us the infrastructure from which to build a grid (the spare CPU cycles),
- We still need a mechanism for controlling the branch-and-bound process on the Grid
- *Don't* lose a portion of the branch-and-bound tree when a process vacates
- *Do* make use of additional resources as they come online

⋆ To make parallel branch-and-bound fault-tolerant, we could (should?) use the master-worker paradigm

- What is the master-worker paradigm, you ask?



- Master assigns tasks to the workers
- Workers perform tasks, and report results back to master
- Workers do not communicate (except through the master)

MW

– Goux, Kulkarni, Linderoth, Yoder

- A set of abstract C++ classes
- User writes 10 functions
- `MW`...
  - ◇ Interacts with resource management software (Condor)
  - ◇ Interacts with message passing software (PVM, Files)
  - ◇ Ensures that all tasks are scheduled and completed
  - ◇ All these complexities are hidden from the user

⋆ I'm actively looking for new users and suggestions for additional functionality

MWInterface

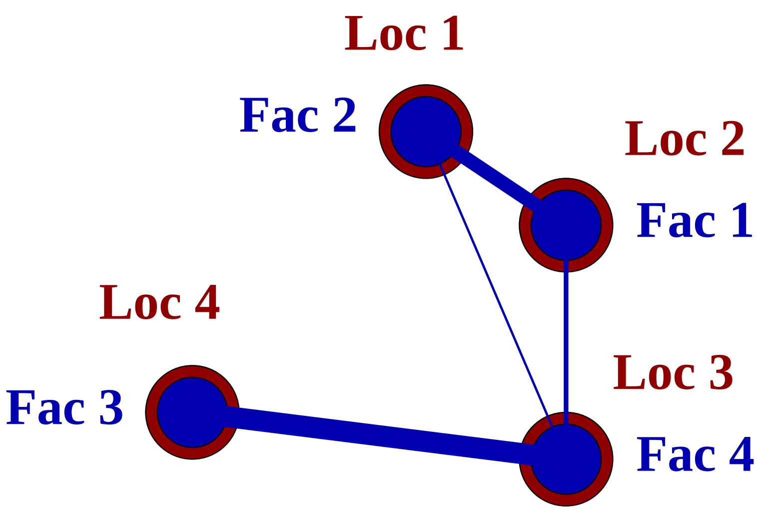
- MWMaster
  - `get_userinfo()`
  - `setup_initial_tasks()`
  - `pack_worker_init_data()`
  - `act_on_completed_task()`
- MWTask
  - `(un)pack_work`
  - `(un)pack_result`
- MWWorker
  - `unpack_worker_init_data()`
  - `execute_task()`

MWApplications

- MWMINLP (Goux, Leyffer, Nocedal) – A branch and bound code for nonlinear integer programming
- MWLShaped (Linderoth, Shapiro, Wright) – A cutting plane and verification code for linear stochastic programming
- FATCOP (Chen, Ferris, Linderoth) – A branch and cut code for linear integer programming
- MWQAP (Anstreicher, Brixius, Goux, Linderoth) – A branch and bound code for solving the quadratic assignment problem
- MWQPBB (Linderoth) – The rudimentary, incomplete, nonsensical code I currently working on
- . . . (Your application here) . . .

The Quadratic Assignment Problem

$$\min_{\pi \in \Pi} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{\pi(i)\pi(j)} + \sum_{i=1}^{n} c_{i\pi(i)}$$



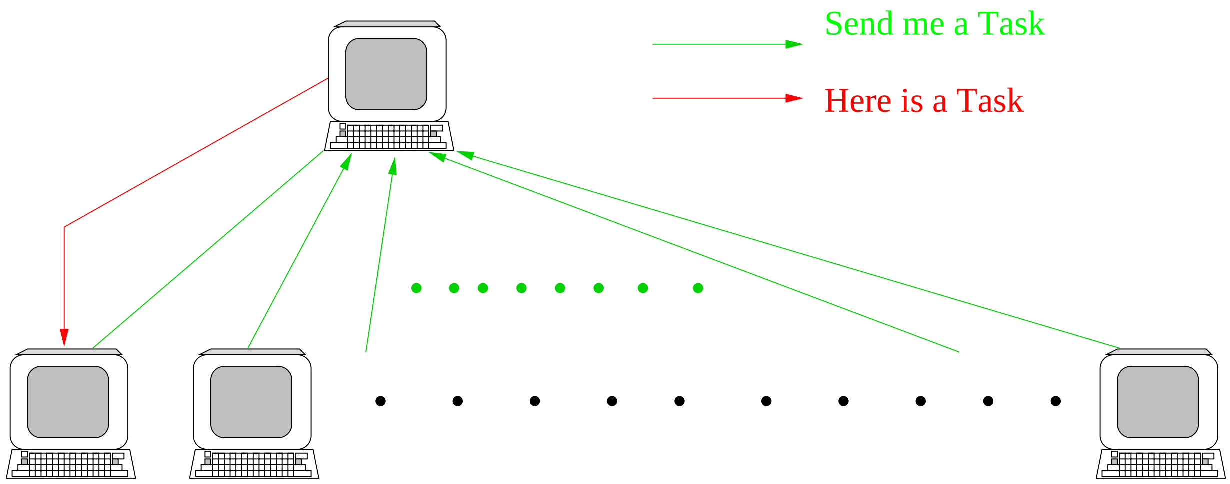
- QAP is NP-“Super”-hard.
  - ◇ TSP : $n > 16,000$
  - ◇ QAP : $n = 25$
- Branch and Bound is the method of choice, but very few tight, computable, bounds exist.

Features of QAP B&B Algorithm

- Convex quadratic programming relaxation.
  - ◇ Solved using Frank-Wolfe algorithm.
- Use "polytomic" branching, based on one facility or one location.
- Exploit symmetry in branching
- Uses (extensively) *strong branching*:
  - ◇ Tentatively branch on each facility/location to see which branching choice will be best
- Implement using MW to run on the Computational Grid

MW Implementation

- Fitting the B & B algorithm into the master-worker paradigm is not groundbreaking research
- We must avoid "contention" at the master



All The Queueing Theory I Know

- We can reduce contention in two ways
  1. Increase the service rate
  2. Reduce the arrival rate

⋆ A parallel depth-first oriented strategy achieves these goals.

- ◇ Available worker is given "deepest" node by master
- ◇ Worker examines the subtree rooted at this node in a depth-first fashion for t seconds.

The Holy Grail!

- (NUG30) $(n = 30)$ has been the "holy-grail" of computational QAP research for > 30 years
- Using an old idea of Knuth, we estimated the CPU time required to solve NUG30 to be 5-10 *years* on a fast workstation

$\Rightarrow$ We'd better get a pretty big Grid!

Our Computational Grid

Number	Type	Location
414	Intel/Linux	Argonne
96	SGI/Irix	Argonne
1024	SGI/Irix	NCSA
16	Intel/Linux	NCSA
45	SGI/Irix	NCSA
246	Intel/Linux	Wisconsin
146	Intel/Solaris	Wisconsin
133	Sun/Solaris	Wisconsin
190	Intel/Linux	Georgia Tech
94	Intel/Solaris	Georgia Tech
54	Intel/Linux	Italy (INFN)
25	Intel/Linux	New Mexico
5	Intel/Linux	Columbia U.
10	Sun/Solaris	Columbia U.
12	Sun/Solaris	Northwestern
2510		

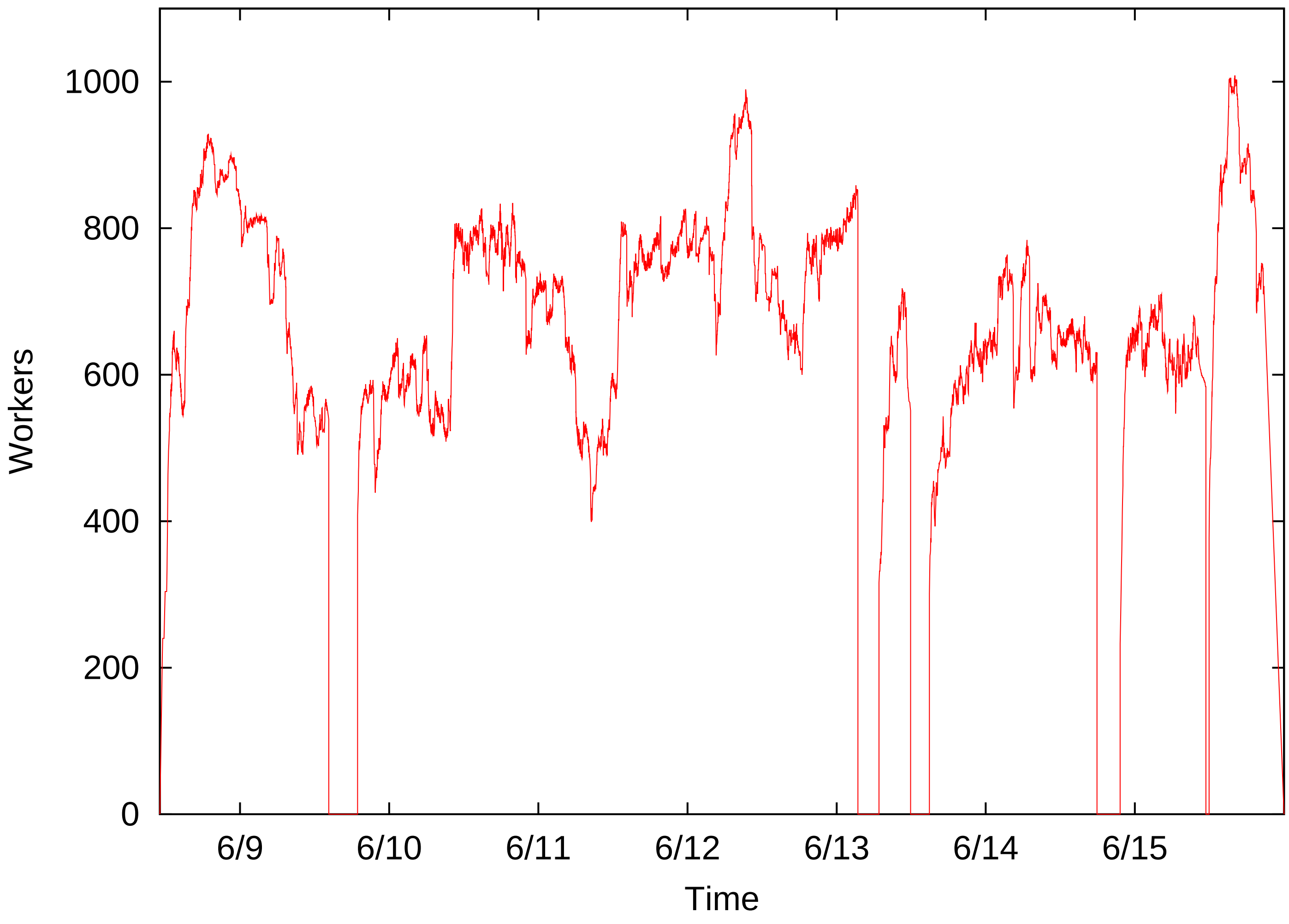
NUG30 is solved!

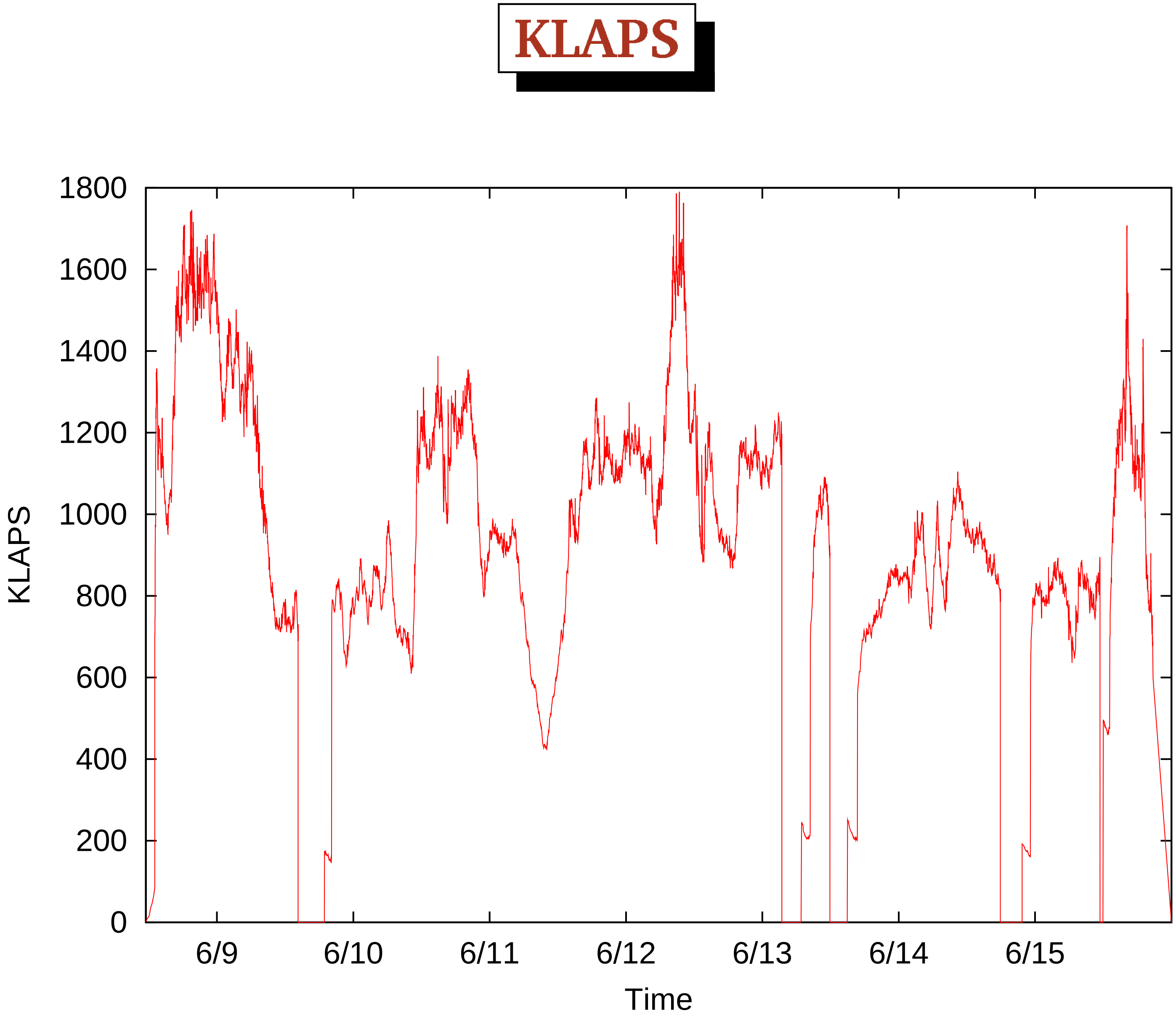
14, 5, 28, 24, 1, 3, 16, 15, 10, 9, 21, 2, 4, 29, 25, 22, 13, 26, 17, 30, 6, 20, 19, 8, 18, 7, 27, 12, 11, 23

"My father used 3.46×10^8 CPU seconds, and all I got was this lousy permutation"

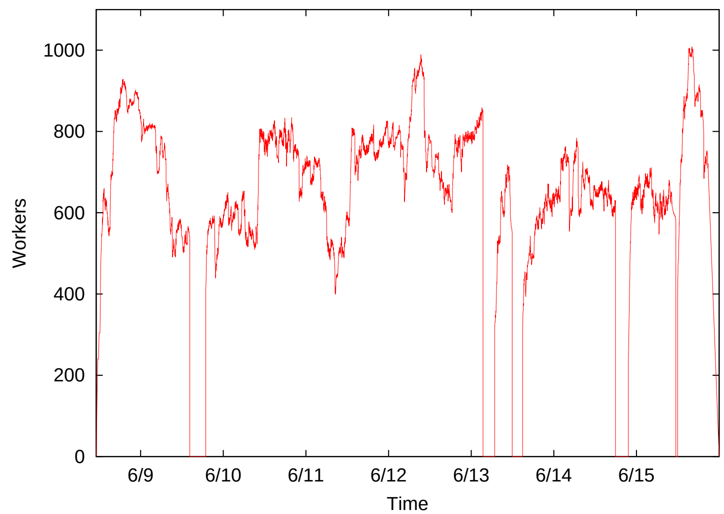
Wall Clock Time:	6:22:04:31
Avg. # Machines:	653
CPU Time:	$\approx$ 11 years
Nodes:	11,892,208,412
LAPs:	574,254,156,532
Parallel Efficiency:	92%

Workers




KLAPS
1800
1600
1400
1200
1000
800
600
400
200
0
KLAPS
6/9
6/10
6/11
6/12
6/13
6/14
6/15
Time

Parallel DFS worked *Great* for QAP



- Kept up to 1000 workers busy $> 90\%$ of the time in a very dynamic grid environment

---

- We knew *a priori* a very good solution
- Tree depth was bounded

Problems with DFS for Global Optimization

- Tree depth not bounded!
  - B&B algorithms may not converge unless you search nodes in a best first fashion (or at least you have to branch on the node with the best lower bound "every once in a while").
- We may not know a good solution
  - ⋆ Use NLP solvers to try and find feasible (locally optimal) solution

How Bad Can Depth-First Search Be?

Ex: Nonconvex quadratic programming formulation of max clique problem on ten nodes.

- ◇ Naive implementation
- ◇ Two-way rectangular branching

- Depth-First Search—$> 3,000,000$ nodes
- Best-First Search— $\approx 30,000$ nodes

How Bad Can Best-First Search Be?

Ex: Nonconvex quadratic programming formulation of max clique problem on 200 nodes.

- ◇ Naive `MW` (Parallel) Implementation running on a Computational Grid of around 100 nodes

- Master processes crashes, since the number of nodes in the list exhausts the computer memory (1GB).

- *Huge* unexplored subtree messages passed from Workers to Master

Conclusions

This page intentionally left blank

The Future of Global Optimization

Disclaimer: This really comes from the perspective of an integer programmer – not someone intimately in touch with the field!

- I think that many of the great advances in deterministic global optimization have come by including more IP technology into the solvers
- But I think maybe more could be done!
  - ◇ Cutting Planes
    - ◇ Nonlinear inequalities?
    - ◇ Can one use RLT (Sherali *et. al*) cuts in a "separate-when-needed" manner
  - ◇ Strong Branching
  - ◇ Strong~er~ Preprocessing
- Run it on the Grid!

(My) Future Work

- Implement SOCP relaxations.
- Add obvious (but *very important*) bells-and-whistles to current code.
  - Strong Preprocessing
  - Strong Branching
- How to balance depth-first with best-first search on the Grid?
- Try to solve some big instances!
  - I'm here looking for big, unsolved, interesting problems!