# Metacomputing for Stochastic Optimization

Jeff Linderoth
Steve Wright

Math and Computer Science Division
Argonne National Laboratory

Alexander Shapiro

Industrial and Systems Engineering
Georgia Institute of Technology

ISMP, August 8, 2000

# Outline

- Two-stage stochastic linear programming
  - The multicut LShaped method
  - A trust region version
- What in the world is a metacomputer?
- Implementing the (trust region) multicut LShaped method on a metacomputing platform
  - Computational Results
- Monte Carlo methods – An even better use of a metacomputer
  - Computational Results

# Two-Stage Linear Recourse Program

$$\min\ c^T x + \sum_{i=1}^{N} p_i q(\omega_i)^T y(\omega_i), \quad \text{subject to}$$

$$Ax = b, \;\; x \geq 0,$$

$$T(\omega_i)x + W y(\omega_i) = h(\omega_i), \;\; y(\omega_i) \geq 0, \quad i = 1, 2, \ldots, N.$$

- Minimizes expected value of objective in the presence of uncertainty
- The scenario $\omega_i$ occurs with probability $p_i$ and is characterized by the data $(q(\omega_i), T(\omega_i), h(\omega_i))$

## Some definitions

- Let $\mathcal{Q}_i(x)$ be the $i$th scenario's objective value for a fixed $x$:

$$\mathcal{Q}_i(x) \equiv \min \; q(\omega_i)^T y(\omega_i) \;\; \text{subject to}$$
$$W(\omega_i)y(\omega_i) = h(\omega_i) - T(\omega_i)x, \;\; y(\omega_i) \geq 0,$$

- For a partition of the $N$ scenarios into chunks $\mathcal{N}_1, \mathcal{N}_2, \ldots \mathcal{N}_t$, let $\mathcal{Q}_{[j]}(x)$ be the contribution of the $j$th chunk to $\mathcal{Q}(x)$:

$$\mathcal{Q}_{[j]}(x) \equiv \sum_{i \in \mathcal{N}_j} p_i \mathcal{Q}_i(x)$$

- Let $\mathcal{Q}(x) \equiv \sum_{i=1}^{N} p_i \mathcal{Q}_i(x) = \sum_{j=1}^{t} \mathcal{Q}_{[j]}$

$\Rightarrow$ The original problem can be written as

$$\min_x \; c^T x + \mathcal{Q}(x), \;\; \text{subject to } Ax = b, \; x \geq 0.$$

## Important (and well-known) Facts

- $\mathcal{Q}_i(x)$, $\mathcal{Q}_{[j]}(x)$, and $\mathcal{Q}(x)$ are piecewise linear convex functions of $x$.
- If $\pi(\omega_i)$ is an optimal dual solution to the linear program corresponding to $\mathcal{Q}_i(\hat{x})$, then $T(\omega_i)^T \pi(\omega_i) \in \partial \mathcal{Q}(\hat{x})$
- $g_j(\hat{x}) \equiv \sum_{i \in \mathcal{N}_{[j]}} p_i T(\omega_i)^T \pi(\omega_i) \in \partial \mathcal{Q}_{[j]}(\hat{x})$.
- Evaluation of $\mathcal{Q}(\hat{x})$ is separable
  - ◇ We can solve linear programs corresponding to each $\mathcal{Q}_i(\hat{x})$ independently

# Multicut L-shaped method

- Represent $\mathcal{Q}_{[j]}(x)$ by an artificial variable $\theta_j$ and find supporting planes for $\theta_j$ (from subgradients of $\mathcal{Q}_{[j]}(x^k)$).
  $\Rightarrow \theta_j \geq g_j(x^k)^T x + (\mathcal{Q}_{[j]}(x^k) - g_j^T x^k) \qquad (*)$

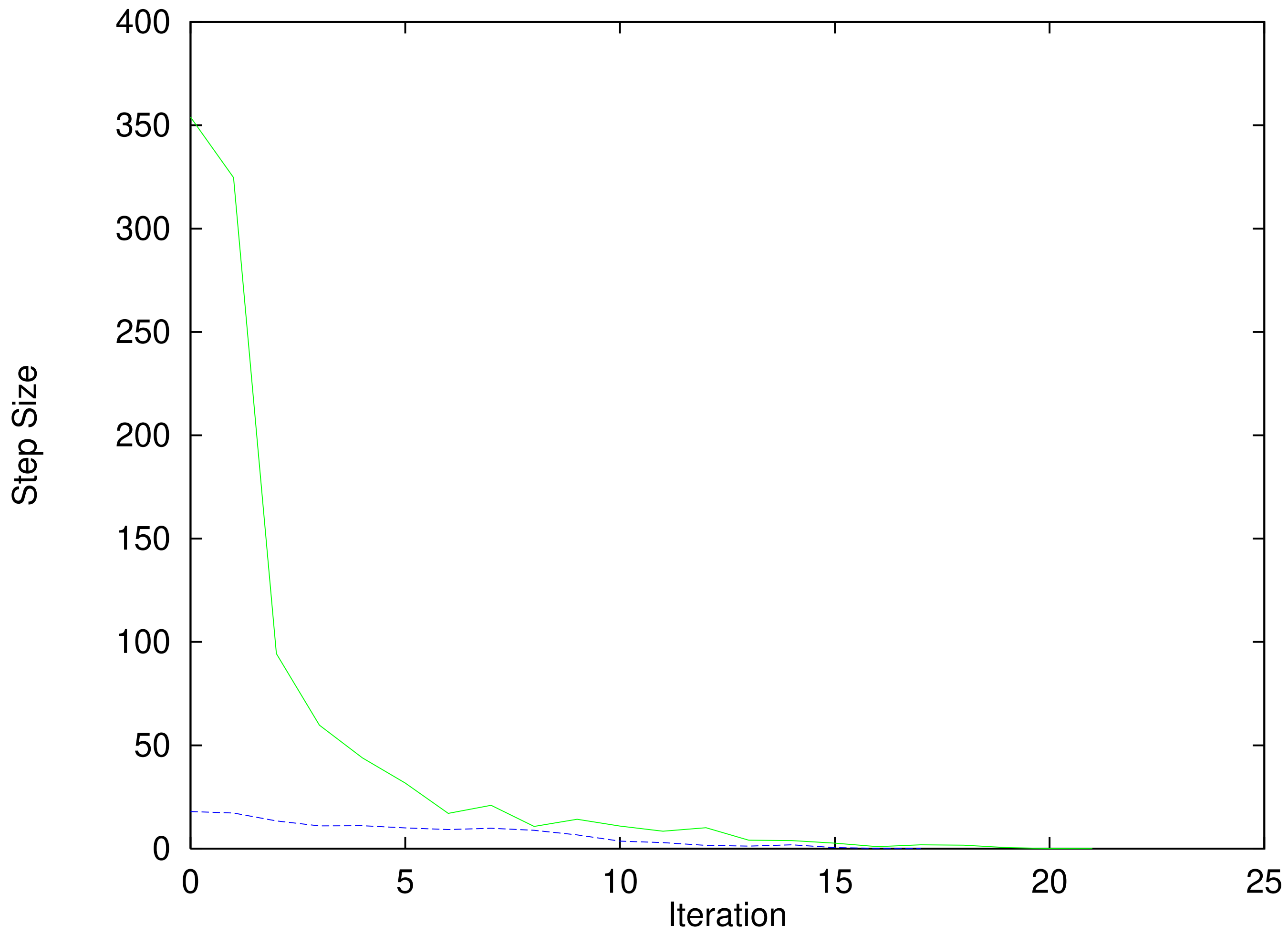
---

1. Solve the **master problem** with the current $\theta_j$-approximations to $\mathcal{Q}_{[j]}(x)$ for $x^k$.
2. Solve the **subproblems**, evaluating $\mathcal{Q}_{[j]}(x^k)$ and obtaining a subgradient $g_j(x^k)$. Add inequalities (*) to the master problem
3. k = k+1. Goto 1.

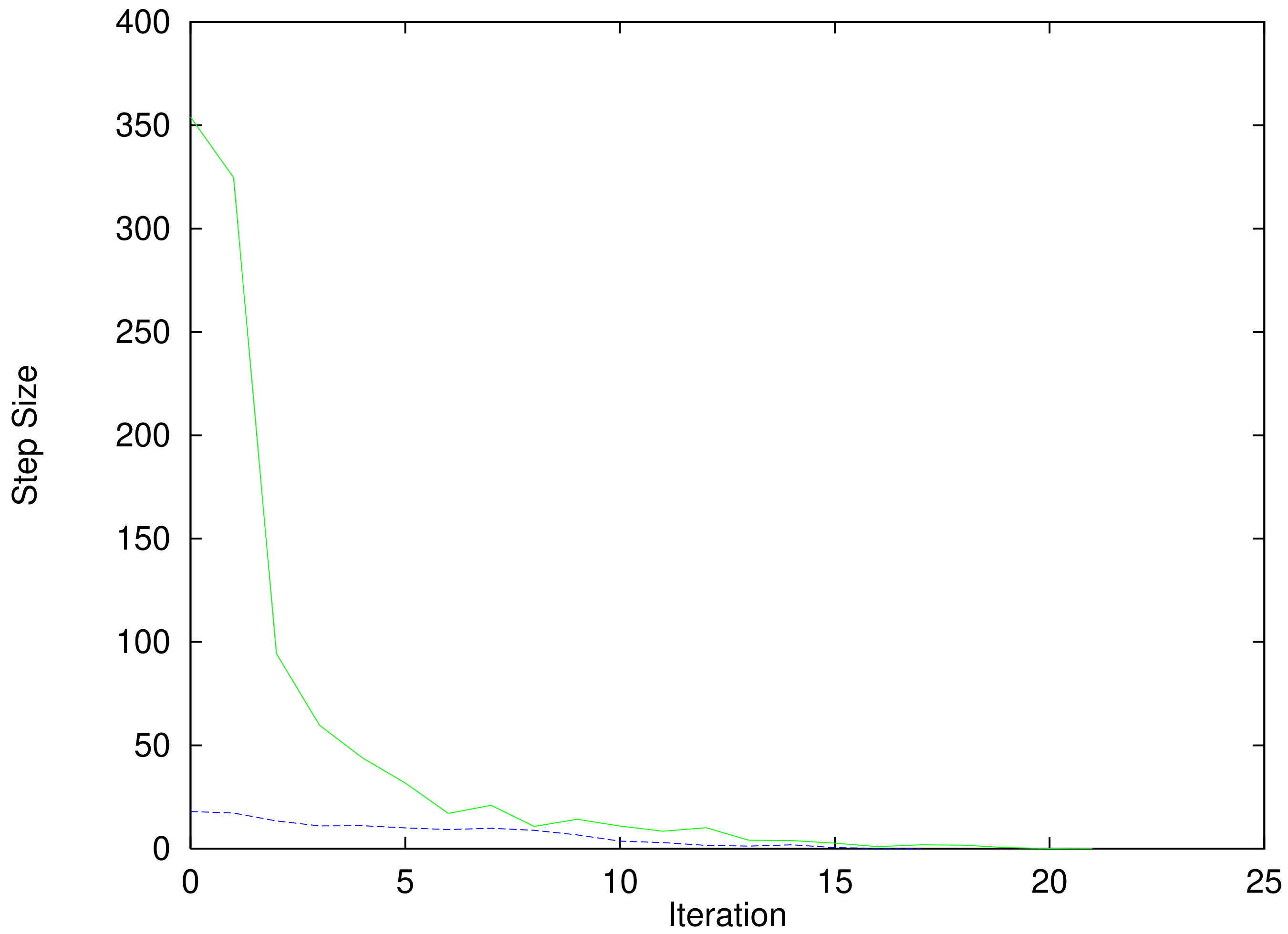
# A Trust Region Method

- The LShaped method suffers from "stability" problems,
  - ◇ Especially in early iterations when a "good enough" model of $\mathcal{Q}(x)$ is not known
  - ◇ Especially bad if started from a good guess at the solution
- Borrow the trust region concept from NLP
  - ◇ At iteration $k$, impose constraints $\|x - x^k\|_\infty \leq \Delta_k$
  - ◇ Only accept new iterate if it improves the objective by a "sufficient" amount
  - ◇ Allows for the systematic deletion of cuts

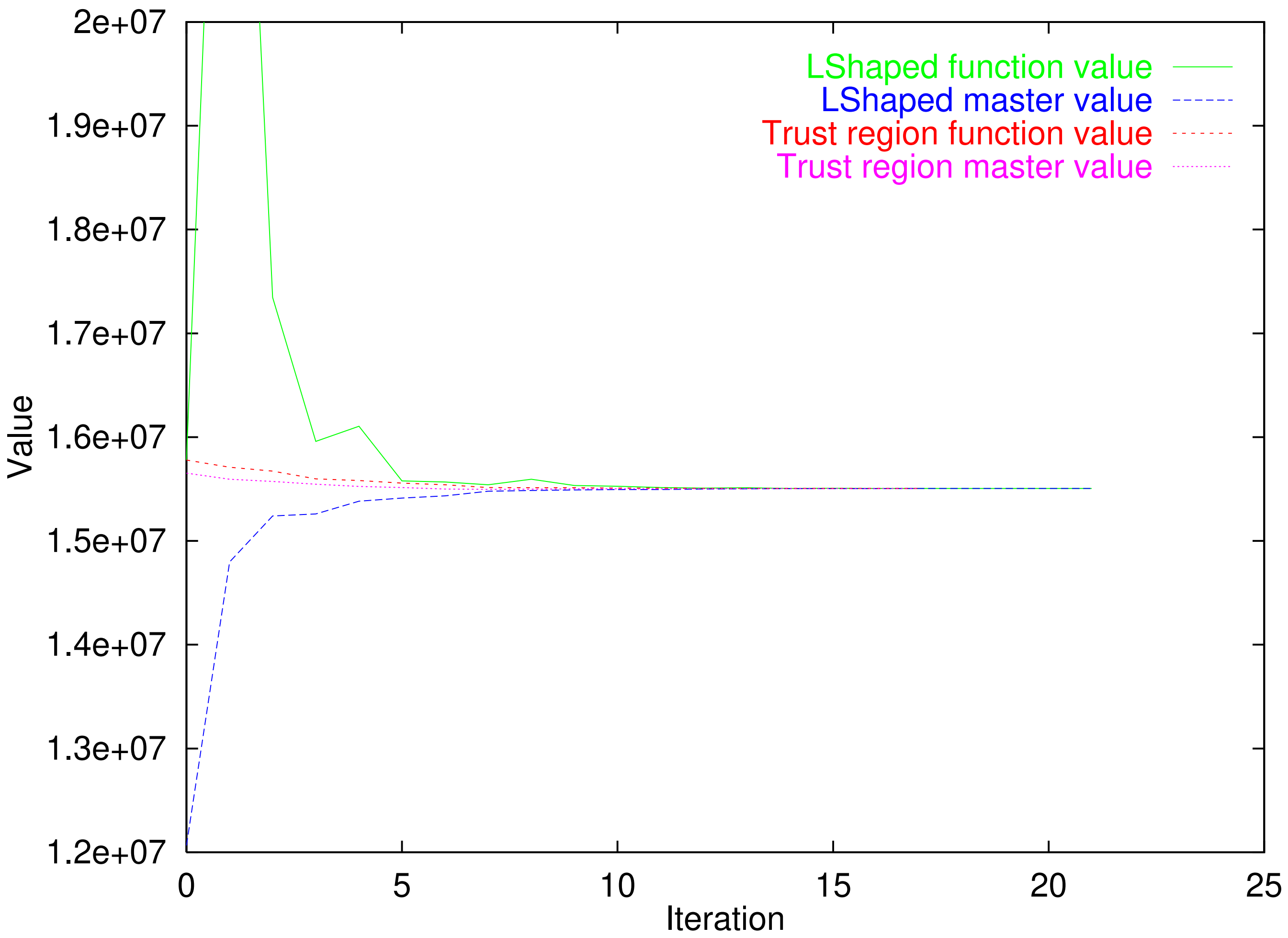
# Trust Region Effect



# Trust Region Effect



# Trust Region Effect



# Stop Thief!

- Stabilizing Bender's decomposition is hardly a new idea
  - Marsten, Hogan, Blankenship ('75)
  - Lemaréchal ('75, . . . )
  - Kiwiel ('83, . . . )
  - Ruszczyński ('86)

---

- We like $\|\cdot\|_\infty$ trust region
  - The master problem is just an LP
  - Degeneracy has not reared its ugly head
- Convergence for arbitrary subgradients – not necessarily extreme point dual solutions $\pi(\omega_i)$.

# Metacomputing

- Some people think of the Internet as a metacomputer
  + Valuable information-sharing device
  - Less valuable as a computational device
- Notable exceptions
  - ◇ SETI@home
  - ◇ Code cracking (`distributed.net`, etc.)
- What about optimization?
  - ◇ metaNEOS project
- Metacomputing is often now called the "Computational Grid"

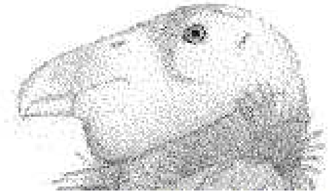
## Metacomputing != Parallel Computing

- Dynamic
  - Resources may come and go during the course of the computation
- Larger scale
  - More resources are available
- Communicationally challenged
  - Machines may be very far apart $\Rightarrow$ slow communication channels between them
- Heterogenous
  - Different hardware, operating systems, and software.

# Creating a Computational Grid

- How is a user supposed to "tap into" all this computational power?
  - Screensaver download?
  - Will they need accounts on all machines?
- How to match computing task requirements with machine characteristics?
- How will the software recognize when machines become available or go away?
- Greed!
  - Most people don't want to contribute "their" machine!

# Condor

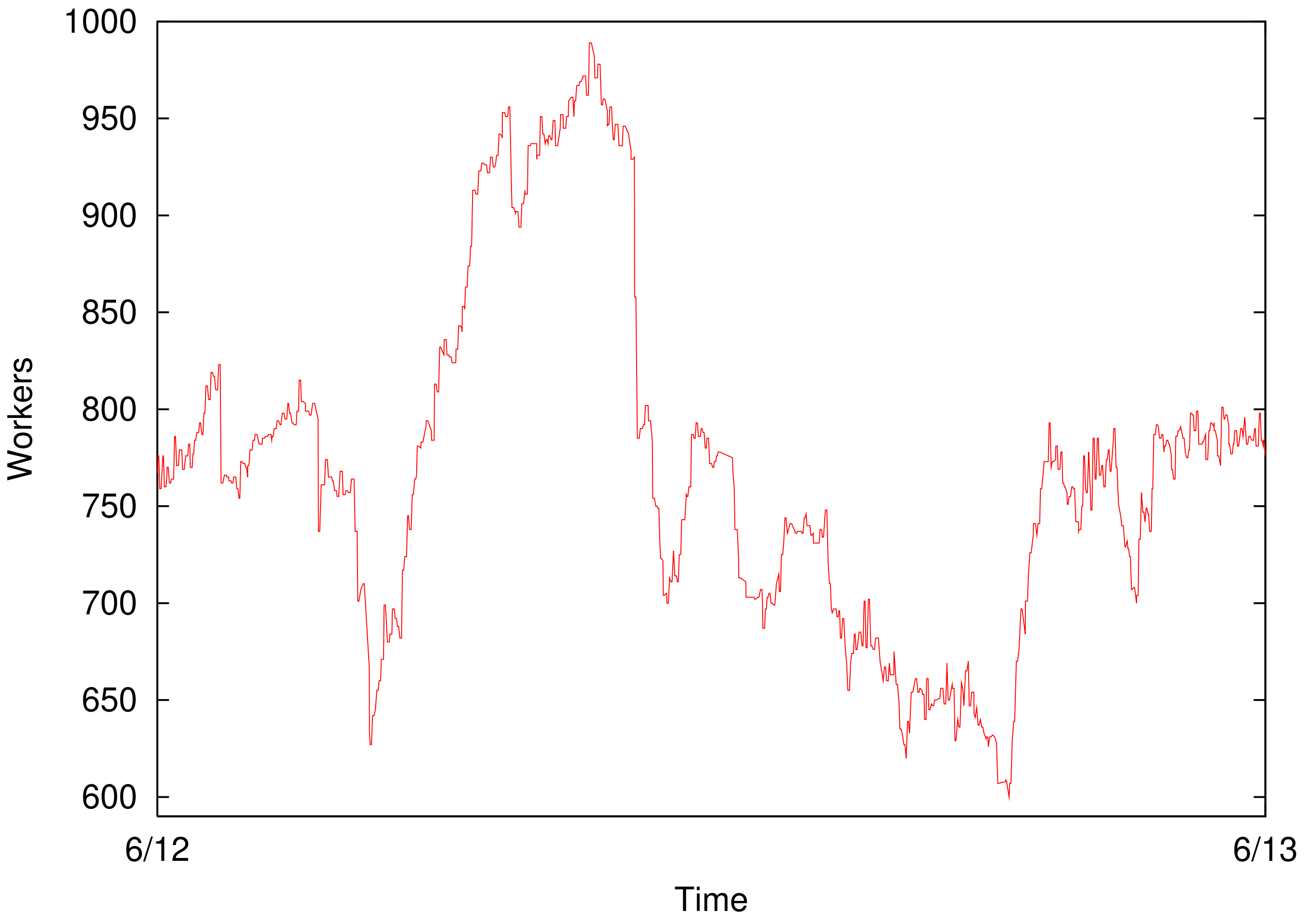


- Manages collections of "distributively owned" workstations
  - ◇ User need not have an account or access to the machine
  - ◇ Workstation owner specifies conditions under which jobs are allowed to run
  - ◇ All jobs are scheduled and "fairly" allocated among the pool
- How does it do this?
  - ◇ Scheduling/Matchmaking
  - ◇ Jobs can be checkpointed and migrated
  - ◇ Remote system calls provide the originating machines environment
  - ◇ "Flocking" allows jobs from one Condor pool to run in other Condor pools

# MW

- There are three abstractions in the master-worker paradigm: Master, Worker, and Task.
- `MW` is a software package that encapsulates these abstractions
  - ◇ C++ abstract classes
  - ◇ User writes 10 functions
  - ◇ The `MWized` code will transparently adapt to the dynamic and heterogeneous environment
- `MW` also has abstract layer to resource management and communications packages (Condor, PVM, etc.)
  - ◇ This complexity is hidden from the user

# Number of available machines



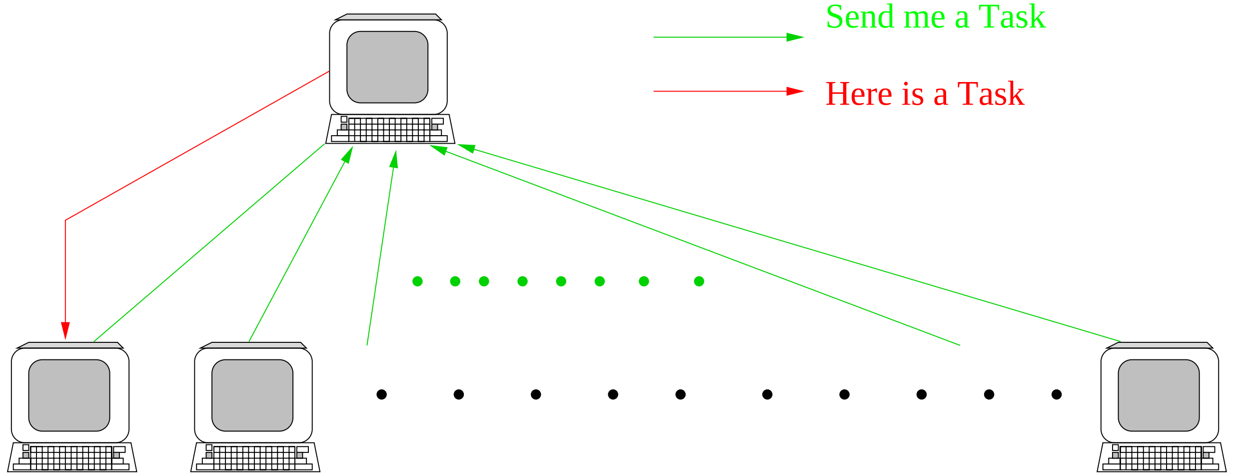
# MWApplications

- MWFATCOP (Chen, Ferris, Linderoth) – A branch and cut code for linear integer programming
- MWMINLP (Goux, Leyffer, Nocedal) – A branch and bound code for nonlinear integer programming
- MWSVM (Ferris, Voelker) – A column generation code for solving support vector machine problems
- MWQAP (Anstreicher, Brixius, Goux, Linderoth) – A branch and bound code for solving the quadratic assignment problem
- MWLShaped (Linderoth, Shapiro, Wright) – A cutting plane and verification code for linear stochastic programming

# Metacomputerizing LShaped

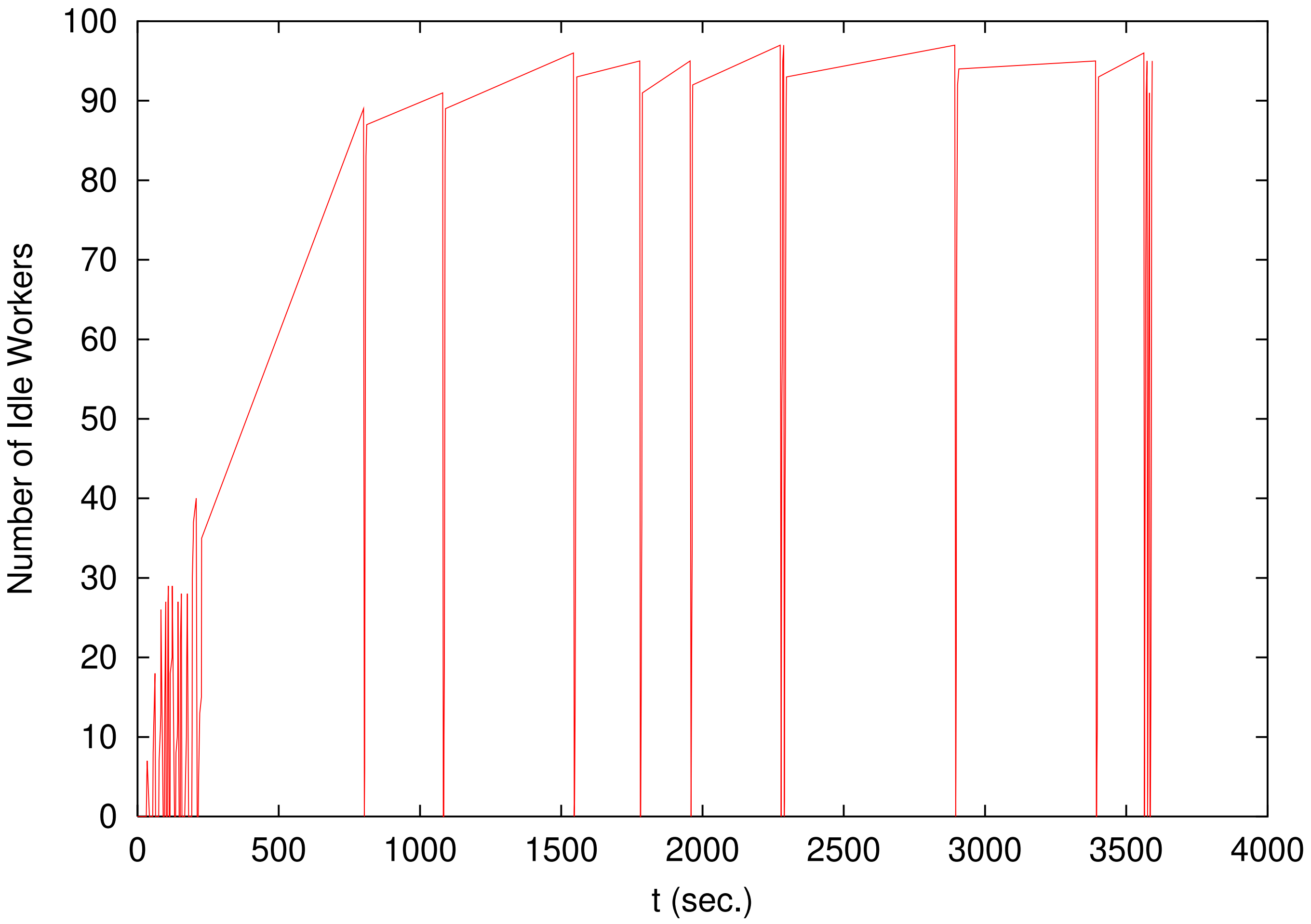
- Work
  - Scenario chunk $\mathcal{N}_j$ and point $(\hat{x})$
- Result
  - A subgradient of $\mathcal{Q}_{[j]}(\hat{x})$.
- What should the cluster size $|\mathcal{N}_j|$ be?
  - We form one cut per cluster (though we don't have to)
  - Little experimental data on how many cuts to add per iteration
  - We have the further complication of selecting the right "grain size" to avoid contention

# Contention




- If $|\mathcal{N}_j|$ is too small, the master is overwhelmed with requests
- Choosing the right $|\mathcal{N}_j|$ is (of course) very problem dependent
- We are still doing computational testing
- We could even choose $|\mathcal{N}_j|$ dynamically
- Usually, I set $|\mathcal{N}_j| = 0.01N$

# Worker Usage



## Stamp Out Synchronicity!

- We start a new iteration only after all "chunks" have been evaluated
  - ◇ In a metacomputer, different processors act at different speeds, so many may wait idle for the "slowpoke"
  - ◇ Even worse, metacomputing tools can fail to inform the user that their worker has failed!
  - ◇ We can never efficiently use more than $|\mathcal{N}_j|^{-1}$ machines

⋆ Asynchronous methods are preferred for traditional parallel computing environments. They are nearly *required* for metacomputing environments!

## An Asynchronous (trust-region) LShaped method

- Keep a "basket" $\mathcal{B}$ of trial points for which we are evaluating the objective function
- Make decision on whether or accept new iterate $x^{k+1}$ after entire $\mathcal{Q}(x^k)$ is computed
- Populate the basket quickly by initially solving the master problem after only $\alpha\%$ of the scenario LPs have been solved

\+ *Greatly* reduces the synchronicity requirements

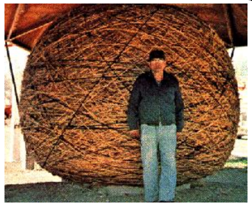
– Might be doing some "unnecessary" work – the candidiate points might be better if you waited for complete information from the preceeding iterations

# Effect of Basket Size

| $\|\mathcal{B}\|$ | # Master It | Parallel Eff | Avg # workers | Time (sec.) |
|---|---|---|---|---|
| 2 | 20 | 3% | 139 | 1720 |
| 3 | 45 | 11% | 78 | 1234 |
| 5 | 57 | 12% | 76 | 1426 |
| 10 | 94 | 34% | 52 | 1628 |
| 15 | 220 | 35% | 108 | 1351 |

- One instance does not compelling evidence make
- Why is the parallel efficiency so low?
  - Short run time and overhead in using a metacomputer
  - $|\mathcal{N}_j|$ is too small – The tasks are only about 4 seconds

# The World's Largest LP



- Storm – A cargo flight scheduling problem (Mulvey and Ruszczyński)
- We aim to solve an instance with 250,000 scenario
- $x \in \Re^{121}, y(\omega_i) \in \Re^{1259}$
- The deterministic equivalent is of size

$$A \in \Re^{132,000,185 \times 314,750,121}$$

---

- $|\mathcal{N}_j| = 1000$, $|\mathcal{B}| = 10$.
- Started from an $N = 3000$ solution, $\Delta_k = 1$
- CPLEX used to solve the master LP. Soplex used to solve (most of) the worker LPs.

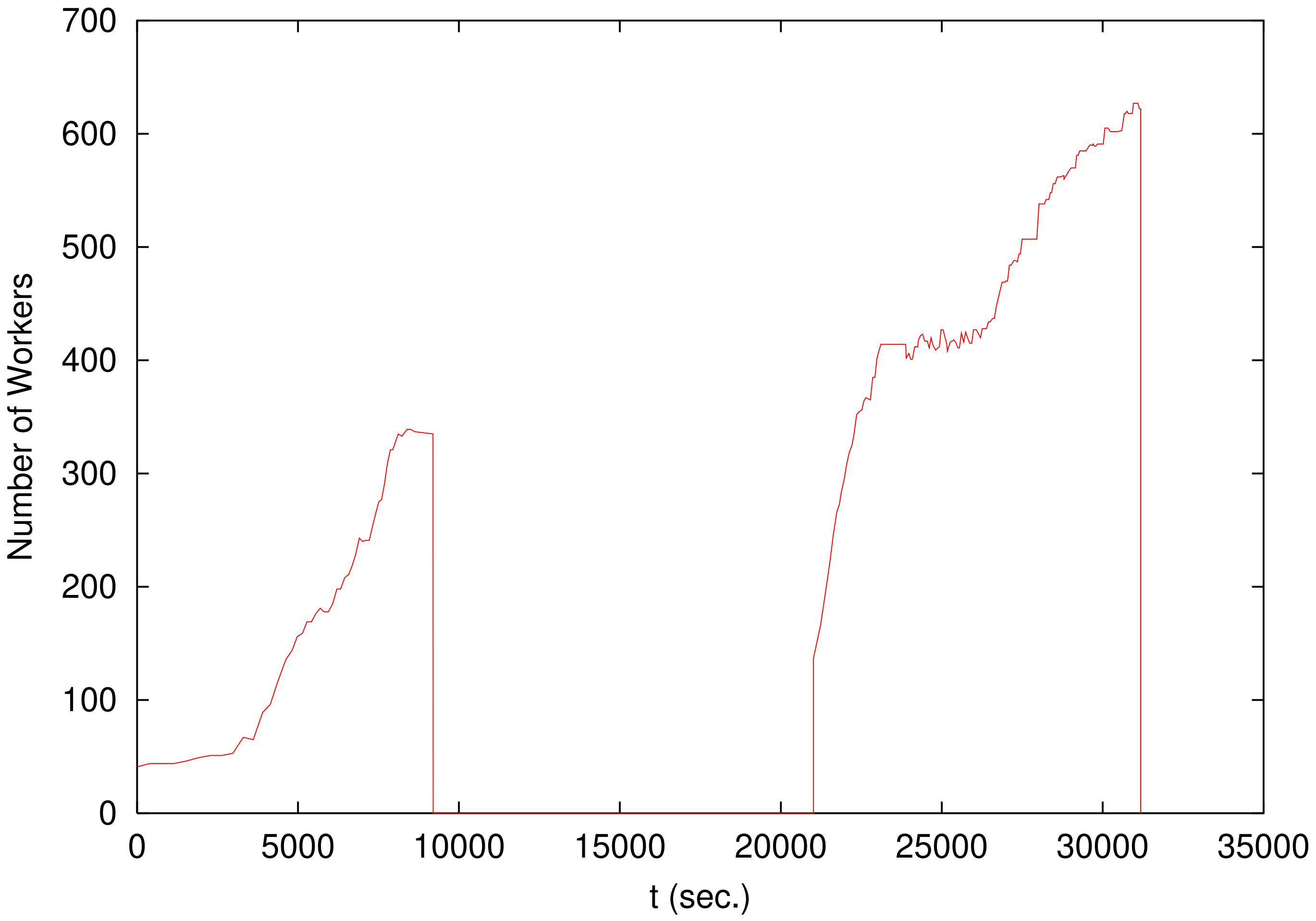
# The Super Storm Metacomputer

| Number | Type | Location |
|---|---|---|
| 494 | Intel/Linux | Argonne |
| 16 | Intel/Linux | NCSA |
| 288 | Intel/Linux | Wisconsin |
| 148 | Intel/Solaris | Wisconsin |
| 137 | Sun/Solaris | Wisconsin |
| 192 | Intel/Linux | Georgia Tech |
| 96 | Intel/Solaris | Georgia Tech |
| 17 | Sun/Solaris | Georgia Tech |
| 55 | Intel/Linux | Italy (INFN) |
| 147 | Intel/Linux | New Mexico |
| 12 | Sun/Solaris | Northwestern |
| 10 | Intel/Linux | Columbia U. |
| 10 | Sun/Solaris | Columbia U. |

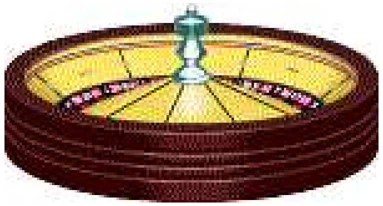
# TA-DA!!!!!

| | |
|---|---|
| Wall clock time | 7:10:15 |
| CPU time | 46:08:13:05 |
| Avg. # machines | 225 |
| Max # machines | 627 |
| Parallel Efficiency | 71% |
| Master iterations | 199 |
| CPU Time solving the master problem | 20:53 |
| Maximum number of rows in master problem | 12640 |
| LP's solved | 49,531,000 |

# Number of Workers



# Monte Carlo Methods



- Instances we can solve are sampled from "true" instances with an exorbitant number of real scenarios ( $N \approx 10^{70}$ )
  - ◇ Not even a super-duper metacomputer can solve $10^{70}$ LPs in a reasonable amount of time. :-)
- Maybe our instances are from a continuous distribution
- Just how far off are we?

⋆ Monte Carlo/Sampling methods provide the answer!!

# Even More Definitions

- $v^*$ is the optimal solution to the "true" problem
  - $v^* \equiv \min_{x \in S} \{f(x) := \mathbb{E}_P h(x, \omega)\}$
- $\widehat{v}_N$ is the optimal solution to a sampled problem of size $N$
  - $\widehat{v}_N \equiv \min_{x \in S} \left\{\widehat{f}_N(x) := N^{-1} \sum_{j=1}^{N} h(x, \omega^j)\right\}$
- Estimate $\mathbb{E}(\widehat{v}_N)$ as $\mathbb{E}(\widehat{v}_N) = M^{-1} \sum_{j=1}^{M} \widehat{v}_N^j$
- One can show...

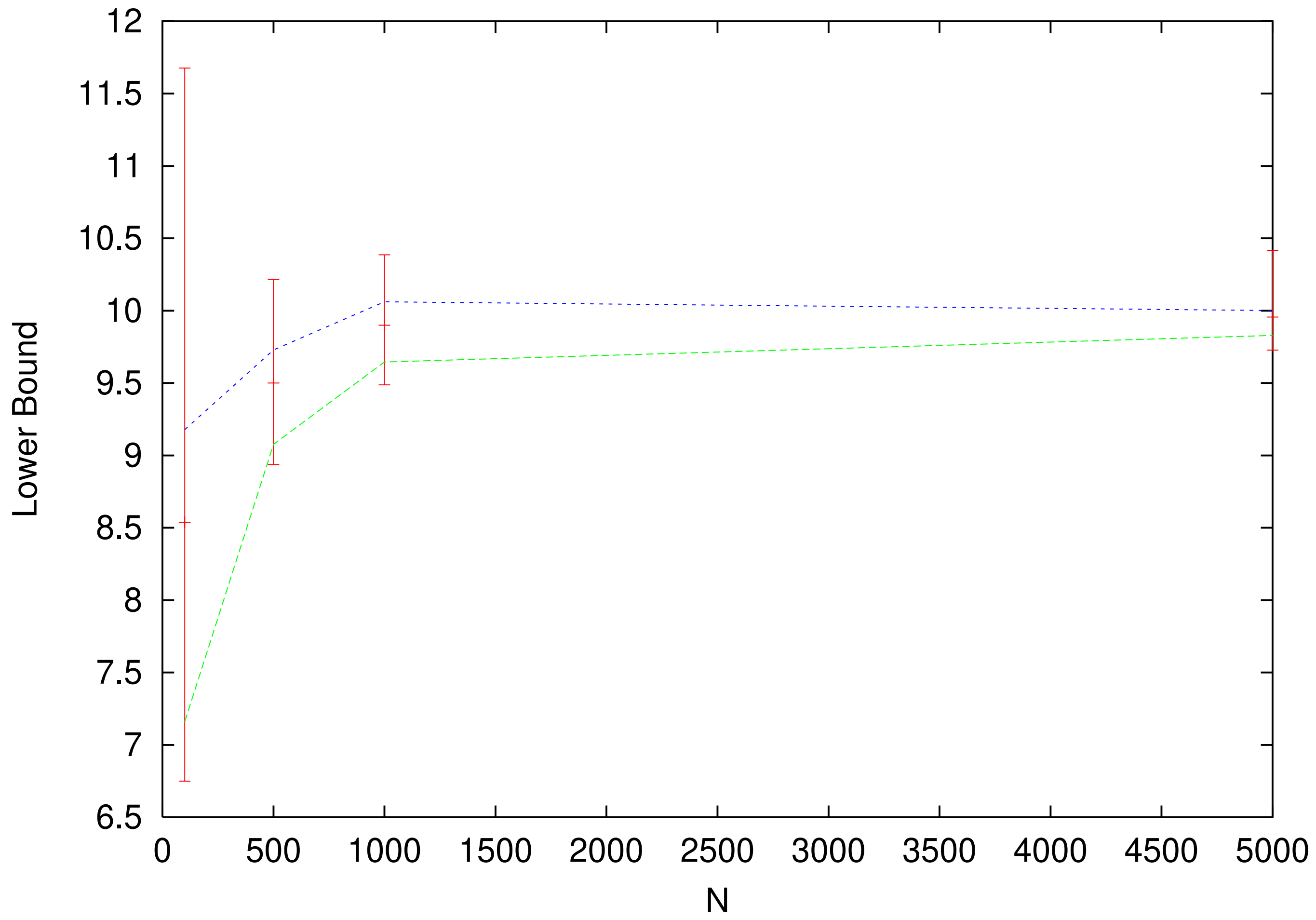
$$\mathbb{E}(\widehat{v}_N) \leq v^* \leq \widehat{f}_{N'}(x) \ \forall x$$

$$\widehat{f}_{N'}(\hat{x}) - \mathbb{E}(\widehat{v}_N) \to f(\hat{x}) - v^*, \text{ as } N, M, N' \to \infty$$

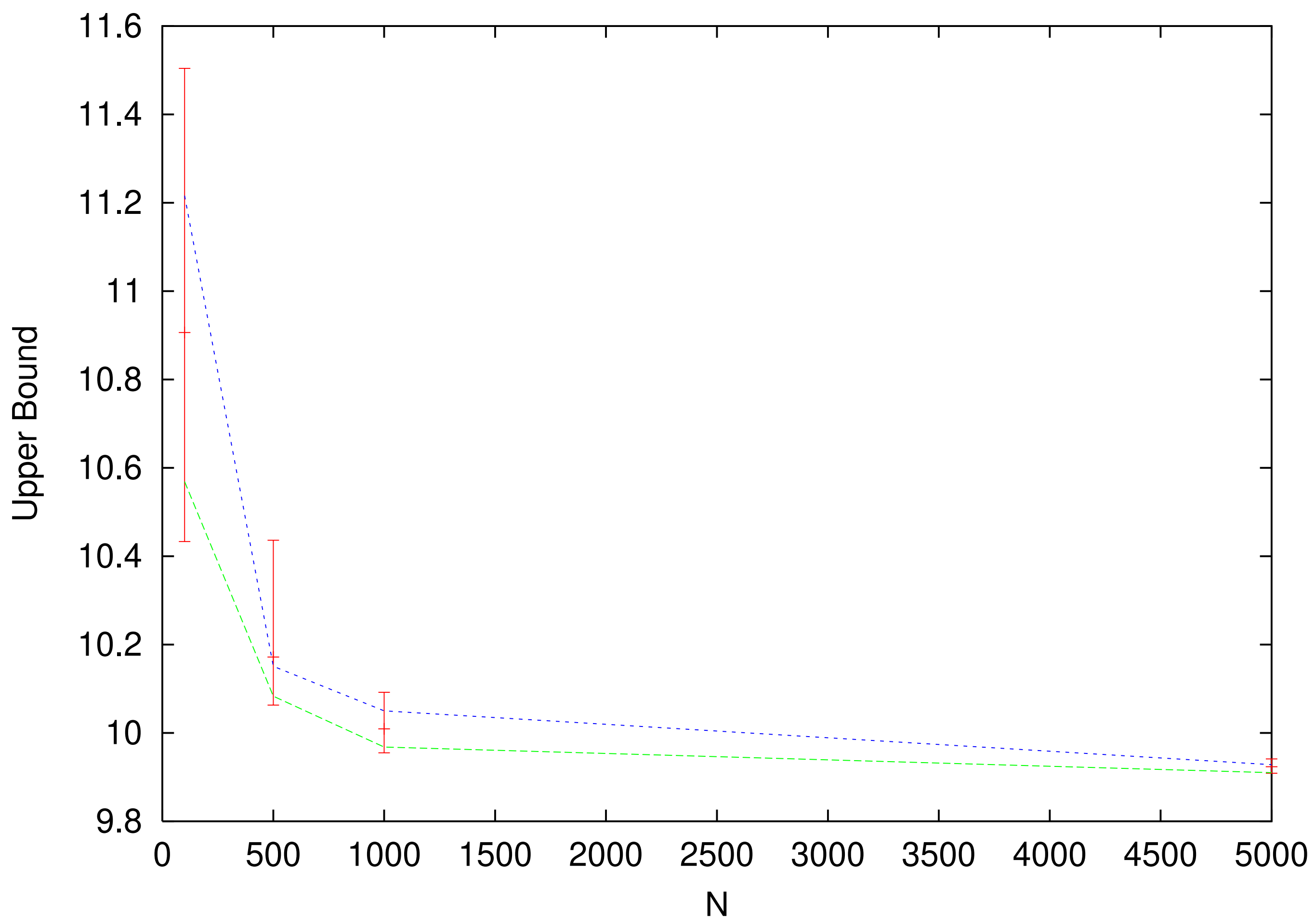
# Estimating the Gap

- $\widehat{f}_{N'}(\hat{x})$ computed by solving $N'$ independent LPs
- $\mathbb{E}(\widehat{v}_N)$ computed by solving $M$ independent stochastic LPs
- This is known as Naturally/Pleasantly/Embarrassingly parallel computing
  - All of the computing tasks are known *a priori*
  - There is *no* synchronization required
- There are not too many interesting parallel computing issues
  - The raw computational power of the metacomputer makes it possible to test convergence issues

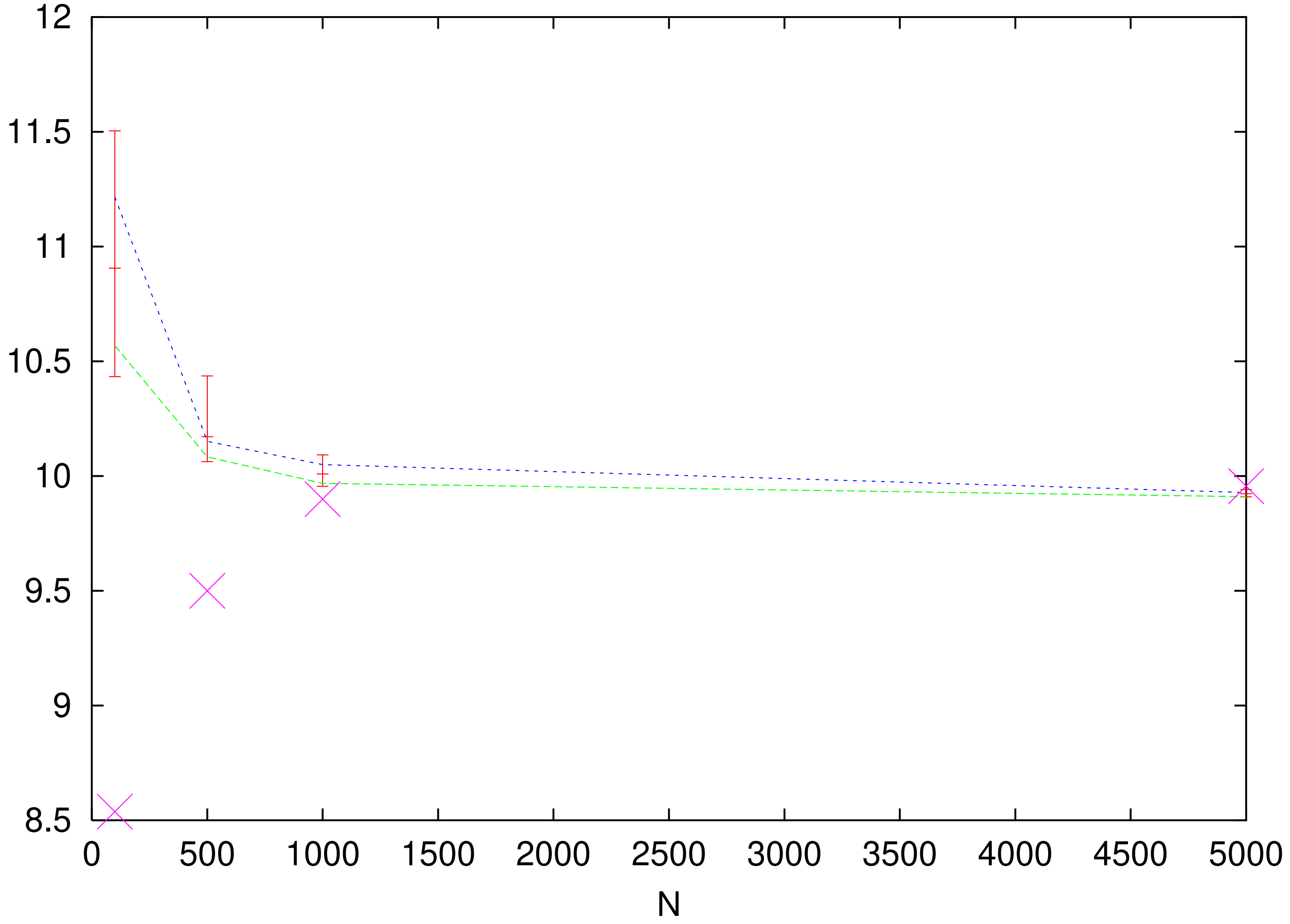
## $\widehat{v}_N$, SSN instance, $M = 12$



## $\widehat{f}_{N'}$, SSN instance, $N' = 10^6$



# SSN Convergence



# Conclusions

- We've wasted our time solving super huge LP's. `:-)`
  - Apply to multi-stage problems
- Metacomputers are extremely powerful computing platforms
  - There are tools available to make harnessing metacomputers easier
  - You may need to tune your algorithm to effectively use a metacomputing platform
- Solving stochastic optimization problems is a wonderful use for metacomputers

`http://www.mcs.anl.gov/metaneos`