IE 495 – Lecture 12

Advanced Features in The LShaped Method

"izing" the LShaped Method

Prof. Jeff Linderoth

February 24, 2003

Outline

- Review
- The LShaped algorithm
  - Regular—izing
  - Bunching and Trickling Down. Similar—izing
  - Parallel—izing

Please Don't Call On Me!

- True or False: The LShaped method works only on problems with complete or relatively complete recourse.
- How many cuts might be added during a "major" iteration of the LShaped method
- How many cuts might be added during a "major" iteration of the multicut-LShaped method

Multicut Review

- A key idea in the LShaped method is to underestimate $\mathcal{Q}(x)$ by an auxiliary variable θ.
- We get the underestimate by the subgradient inequality.
- $\mathcal{Q}(x) = \sum_{s \in S} p_s Q(x, \omega_s)$
- For any scenario $s \in S$, $-T_s^T \lambda_s^* \in \partial Q(x, \omega_s)$, and some "fancy" convex analysis can show that

$$-\sum_{s \in S} p_s T_s^T \lambda_s^* \in \partial \mathcal{Q}(x)$$

$\Rightarrow$ We can equally well approximate (or underestimate) *each* $Q(x, \omega_s)$ by the auxilary variable(s) $\theta_s, s \in S$.

A Whole Spectrum

- LShaped—One cut per master iteration
- Multicut—$|S|$ cuts per master iteration
- We can do anything in between...
- Partition the scenarios into C "clusters" $\mathcal{S}_1, \mathcal{S}_2, \ldots \mathcal{S}_C$.

$$\mathcal{Q}_{[\mathcal{S}_k]}(x) = \sum_{s \in S_k} p_s Q(x, \omega_s)$$

The “Chunked” multicut method

$$\mathcal{Q}(x) = \sum_{k=1}^{C} \mathcal{Q}_{[\mathcal{S}_k]}(x).$$

$$\eta = \sum_{s \in S_k} p_s T_s^T \lambda_s^* \in \partial \mathcal{Q}_{[\mathcal{S}_k]}(x)$$

- We can do the same thing, just approximating $\mathcal{Q}_{[\mathcal{S}_k]}(x)$ by the subgradient inequalities.

Chunked-Multicut-LShaped Method – Step 0

- With θ_k^0 a lower bound for $\mathcal{Q}_{[\mathcal{S}_k]}(x)$.
- Let $\mathcal{B}_0 = \{\Re_n^+ \times \{\theta_1, \theta_2, \ldots, \theta_C\} | Ax = b\}$
- Let $\mathcal{B}_1 = \{\Re_n^+ \times \{\theta_1, \theta_2, \ldots, \theta_C\} | \theta_k \geq \theta_k^0 \quad \forall k = 1, 2, \ldots C\}$

Multicut-LShaped Method – Step 1

- Solve the *master problem*:

$$\min\{c^T x + \sum_{k=1}^{C} \theta_k | (x, \theta_1, \theta_2, \ldots \theta_C) \in \mathcal{B}_0 \cap \mathcal{B}_1\}$$

- yielding a solution $(\hat{x}, \hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_C)$.

Lshaped Method – Step 2

- Evaluate $\mathcal{Q}(\hat{x}) = \sum_{s \in S} p_s Q(\hat{x}, \omega_s)$.
- If $\mathcal{Q}(\hat{x}) = \infty$, which means that there is some $\hat{\omega}$ such that $Q(\hat{x}, \hat{\omega}) = \infty$, we add a *feasibility cut*:
  - ◇ $\mathcal{B}_1 = \mathcal{B}_1 \cap \{(x, \theta) | \sigma^T(h(\hat{\omega}) - T(\hat{\omega})x) \leq 0\}$
  - ◇ (Note that the inequality has no terms in θ_s – it is the same inequality as the LShaped method
- Go to 1.

Step 2 (cont.)

- If $\mathcal{Q}(\hat{x}) < \infty$, then you were able to solve all s scenario LP's (with corresponding dual optimal solution λ_s^*), and you get subgradients for each of the chunks...

$$u_k = \sum_{s \in S_k} p_s T_s^T \lambda_s^* \in \partial \mathcal{Q}_{[\mathcal{S}_k]}(x)$$

- If $\mathcal{Q}_{[\mathcal{S}_k]}(\hat{x}) \leq \theta_k \forall k = 1, 2, \ldots C$, Stop. $\hat{x}$ is optimal.
- If $\mathcal{Q}_{[\mathcal{S}_k]}(\hat{x}) > \theta_k$
  - ◇ $\mathcal{B}_1 = \mathcal{B}_1 \cap \{(x, \theta_1, \theta_2, \ldots \theta_C) : \theta_k \geq \mathcal{Q}_{[\mathcal{S}_k]}(\hat{x}) + u_k^T(x - \hat{x})\}$.
- Go to 1.

A Dumb Algorithm?

$$\min_{x \in \Re^n_+} \{c^T x + \mathcal{Q}(x) | Ax = b\}$$

- What happens if you start the (multi-cut) LShaped procedure with the optimal solution x^*?

? Are you done?

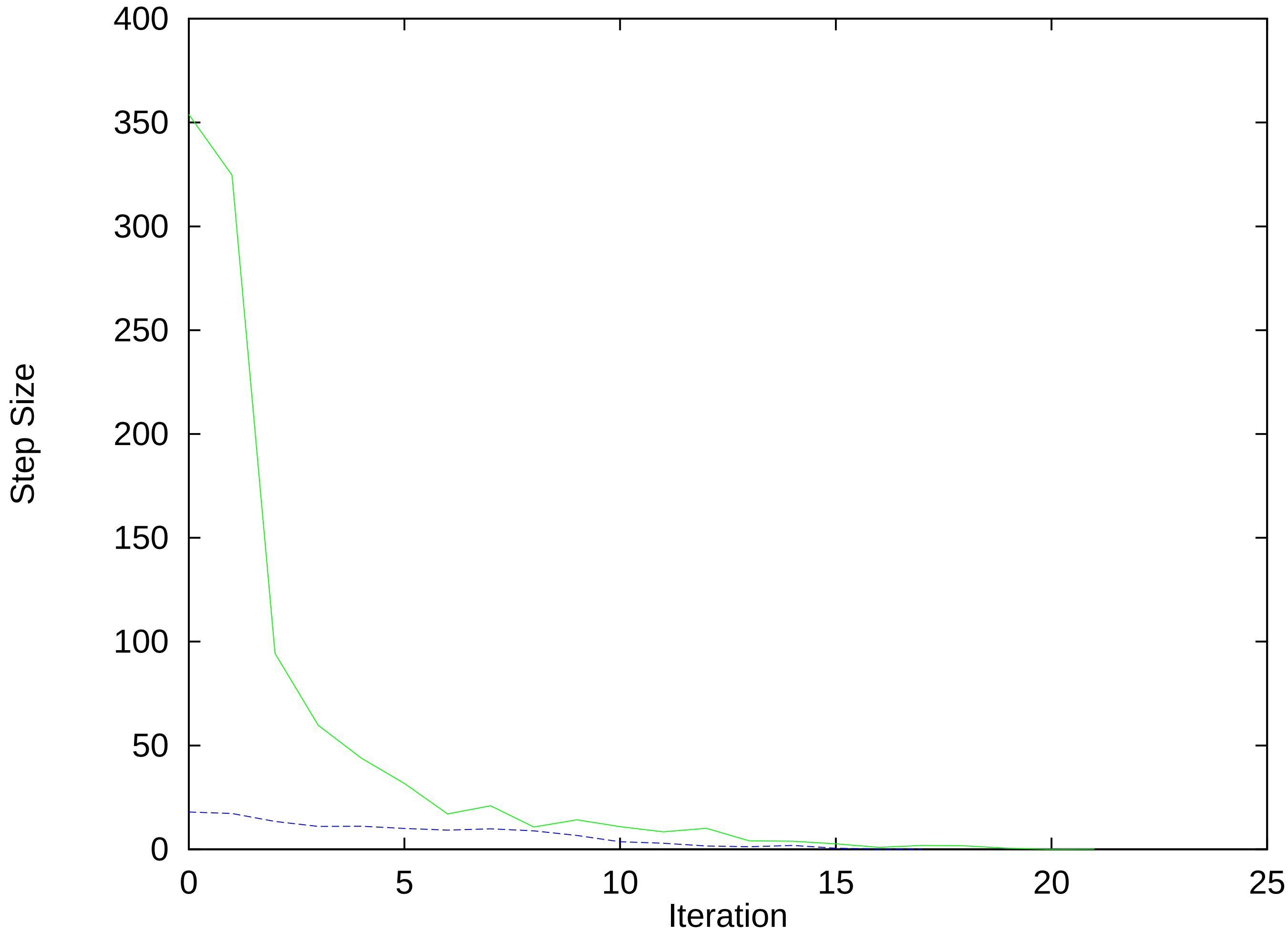
A Trust Region Method

- 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 $\|x - x^k\|_\infty \leq \Delta_k$
- Δ_k large $\Rightarrow$ like LShaped
- Δ_k small $\Rightarrow$ “stay very close”.
- This is often called *Regularizing* the method.

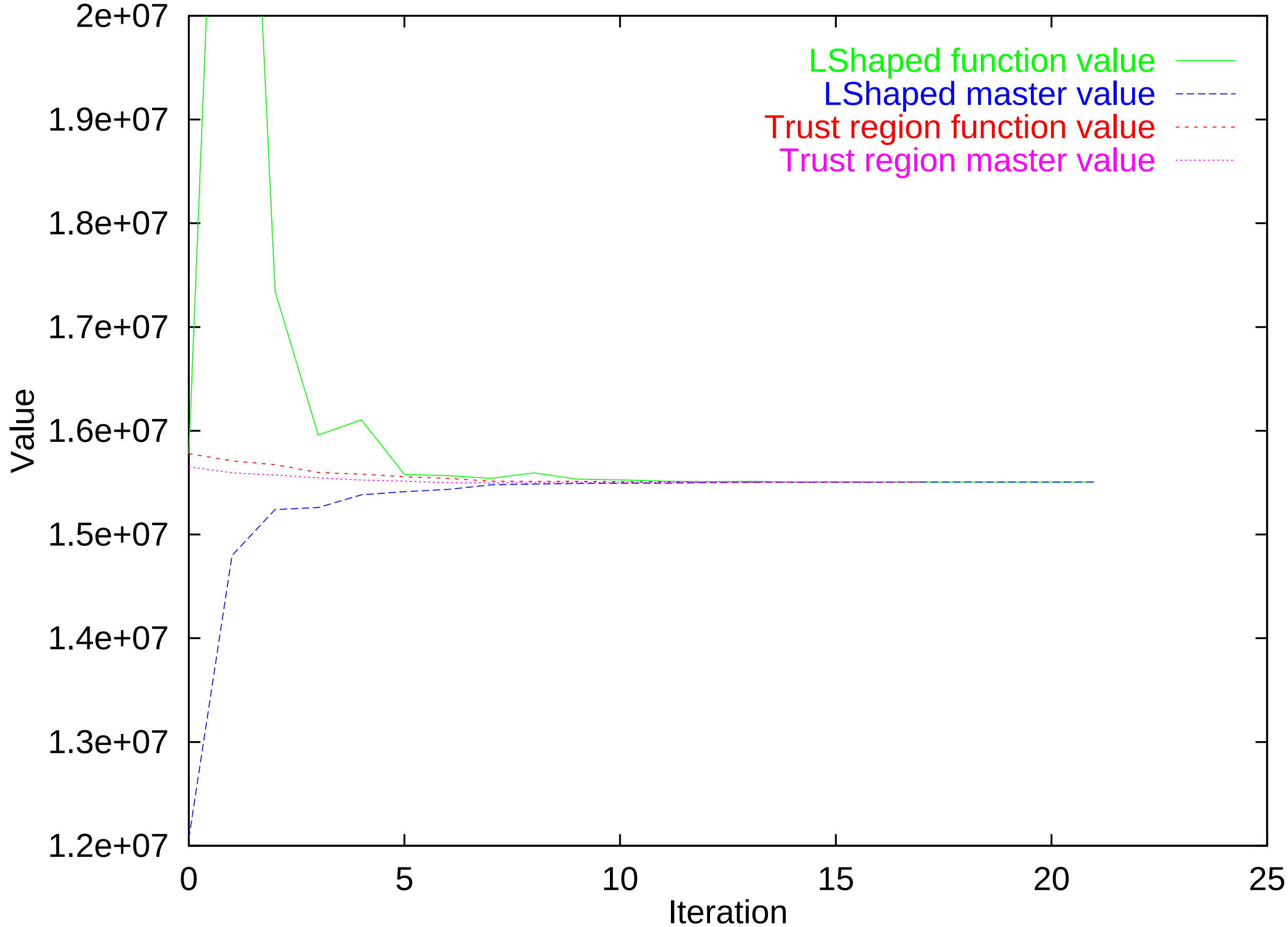
Another Idea

- Alternatively (dual-ly) “penalize” the length of the step you will take.
  - $\min c^T x + \sum_{j \in C} \theta_j + 1/(2\rho)\|x - x^k\|^2$
  - ρ large $\Rightarrow$ like LShaped
  - ρ small $\Rightarrow$ “stay very close”.
- This is known as the *regularized decomposition method*.

Trust Region Effect



Trust Region Effect



Regularizing

- Penalized Step Length Approach:
  - ◇ A. Ruszczyński, "A Regularized Decomposition for Minimizing a Sum of Polyhedral Functions", *Mathematical Programming*, **35**, pp. 309-333, 1986.
- $\|\cdot\|_\infty$ Approach:
  - ◇ J. T. Linderoth and S. J. Wright, "Implementing a Decomposition Algorithm for Stochastic Programming on a Computational Grid," *Computational Optimization and Applications*, special issue on Stochastic Programming, 24:207-250, 2003.
  - This "seminal" `:-)` paper is available on my web site.

Stop Thief!

"Lesser artists borrow, great artists steal."

– Igor Stravinsky

- Stabilizing Bender's decomposition is hardly a new idea
  - ◇ Marsten, Hogan, Blankenship ('75)
  - ◇ Lemaréchal ('75, . . .)
  - ◇ Kiwiel ('83, . . .)
  - ◇ Ruszczyński ('86)
  - ◇ Neame, Boland, and Ralph ('98)

A Little More Detail...

- Let $m(x)$ be the "model" function that you get my solving the master problem.
- Let a be your current "incumbent" solution
- Multicut-LShaped
  - $m(x) = \min\{c^T x + \sum_{k=1}^{C} \theta_k | (x, \theta_1, \theta_2, \ldots \theta_C) \in \mathcal{B}_0 \cap \mathcal{B}_1\}$
- Trust-region
  - $m(x) = \min\{c^T x + \sum_{k=1}^{C} \theta_k | (x, \theta_1, \theta_2, \ldots \theta_C) \in \mathcal{B}_0 \cap \mathcal{B}_1, \|x - a\|_\infty \leq \Delta_l\}$
- Regularized-Decomposition.
  - $m(x) = \min\{c^T x + \sum_{k=1}^{C} \theta_k + 1/(2\rho_l)\|x - a\|_2^2 | (x, \theta_1, \theta_2, \ldots \theta_C) \in \mathcal{B}_0 \cap \mathcal{B}_1\}$

What Can Happen?— Great Model

- Suppose you solve the master problem, getting a solution x. What can happen?
- $\mathcal{Q}_{[\mathcal{S}_k]}(x) = m(k)$
- In this case, if $x = a$, you are done.
- Otherwise, Let $a = x$ and do another iteration.
  - ◇ Maybe increase Δ or ρ

What Can Happen?— Good Enough Model

- $\mathcal{Q}_{[\mathcal{S}_k]}(x) > m(k)$. (Your "model" of the function was not entirely accurate).
- How good was it? Compare the true decrease to the predicted decrease.

$$\sigma = \frac{\mathcal{Q}_{[\mathcal{S}_k]}(a) - \mathcal{Q}_{[\mathcal{S}_k]}(x)}{\mathcal{Q}_{[\mathcal{S}_k]}(a) - m(x)}$$

- If $\sigma > \mu$ (Say $\mu = 0.8$). Our model was "good enough". Let $a = x$. Also add the cuts.
  - ◇ Maybe decrease Δ or ρ — Probably leave alone.

What Can Happen—Bad Model

- If $\sigma < \mu$,
  - Our model was not good enough
  - σ can even be < 0
- Just stay put $a = a$
- Add the "improved" model information gained from the subgradient ineequalities, and continue.

Bundle-Trust

- These ideas are known in the nondifferntiable optimization community as "Bundle-Trust-Region" methods.
  - ◇ *Bundle* — Build up a bundle of subgradients to better approximate your function
  - ◇ *Trust region* — Stay close (in a region you trust), until you build up a good enough bundle to model your function accurately
- Accept new iterate if it improves the objective by a "sufficient" amount. Potentially increase Δ_k. (*Serious Step*)
- Otherwise, improve the estimation of $\mathcal{Q}(x^k)$, resolve master problem, and potentially reduce Δ_k (*Null Step*)

⋆ These methods can be shown to converge, *EVEN IF YOU DELETE CUTS*.

Bunching

- A (if not *the*) major component of the work that must be done in any of the methods we have learned is evaluating $\mathcal{Q}(x)$.
  - We must solve $|S|$ linear programs that differ only in their right hand side.
  - The dual LPs differ only the objective function.
  - (Assuming – like we usually have – that the objective function coefficients are deterministic).

$$\lambda_s^* = \arg\max_{\lambda}\{\lambda^T(h_s - T_s\hat{x}) : \lambda^T W \leq q\}.$$

Bunching

- Just to simplify notation a bit, let's assume that we have a collection of right hand sides $\mathcal{R}$ for which we must solve $\forall r \in \mathcal{R}$:

$$\max_{\lambda} \{\lambda^T r | \lambda^T W \leq q\}$$

⋆ If λ is feasible for one r (one $h_s - T_s \hat{x}$), then it is feasible for *all* $h_s - T_s \hat{x}$.

  ◇ (r only appears in the objective function)

Basic Bunching Idea

- Choose a RHS $\hat{r} \in \mathcal{R}$.
- Solve the scenario subproblem, getting a basis $B_{\hat{r}}$.
- (Again) $B_{\hat{r}}$ is a dual feasible basis for all $r \in \mathcal{R}$.
- If $\forall\ r \in \mathcal{R}$, $B_{\hat{r}}^{-1} r \geq 0$, then $B_{\hat{r}}$ is also an optimal basis for r
  - $\lambda_r^* = q_{B_{\hat{r}}} B_{\hat{r}}^{-1}$
- You don't need to solve the LP.

Simple Bunching

- Denote by $\mathcal{T} \subseteq \mathcal{R}$ the set of right hand sides that you must solve
- Denote by $\mathcal{U}_k$ the kth "bunch" of right hand sides.

1. Let $\mathcal{T} = \mathcal{R}$. Let $k = 0$
2. Choose a "representative" $r \in \mathcal{T}$. If $\mathcal{T} = \emptyset$, let $b = k$, be the total number of bunches. **Stop.**
3. Solve $\max_\lambda \{\lambda^T r | \lambda^T W \leq q\}$, obtaining a basis B_r, and optimal dual solution λ_k.
4. Forall $t \in \mathcal{T}$, check if $B_r^{-1} t \geq 0$. If so, $\mathcal{U}_k = \mathcal{U}_k \cup t$. Let $\mathcal{T} = \mathcal{T} \setminus \mathcal{U}_k$.
5. **Go To** 2.

Simple Bunching

- Let U_k be the scenario indices of the right-hand-sides in bunch $\mathcal{U}_k$.
- Subgradients:

$$\eta = (\sum_{k=1}^{b} \lambda_k)^T \sum_{s \in U_k} p_s T_s$$

- Main idea to just exploit similarity to solve the LP's faster (or not at all). Use as appropriate for the particular algorithm.

Trickling Down

- Key idea: Keep a *tree* of pivot elements, starting from a "representative" basis $\hat{B}$

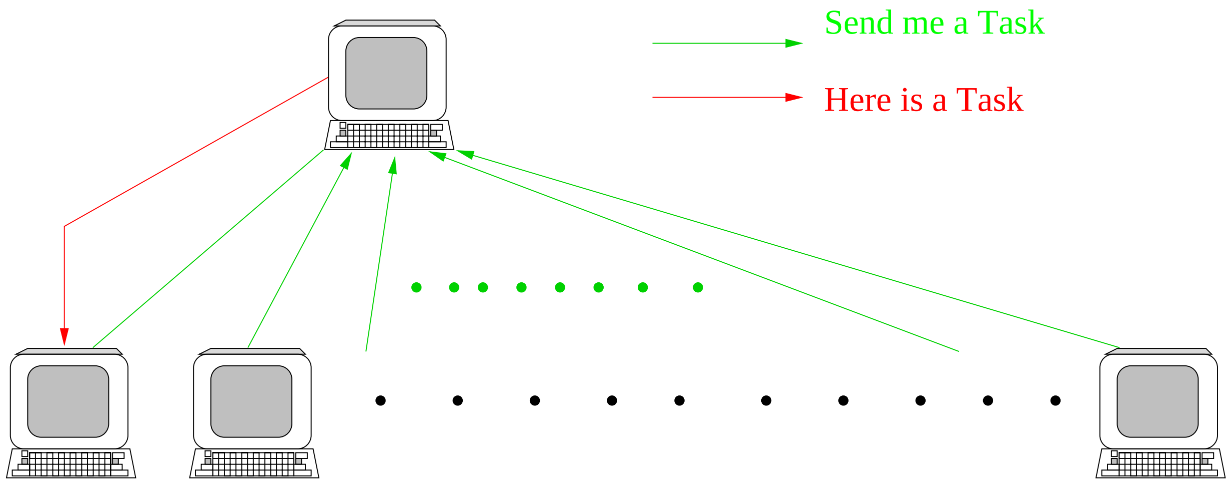
Picture

- I don't know if it's worth it
- *Definitely* should use dual simplex to solve scenario LP's. Intelligent ordering of scenarios can help.

Parallelizing

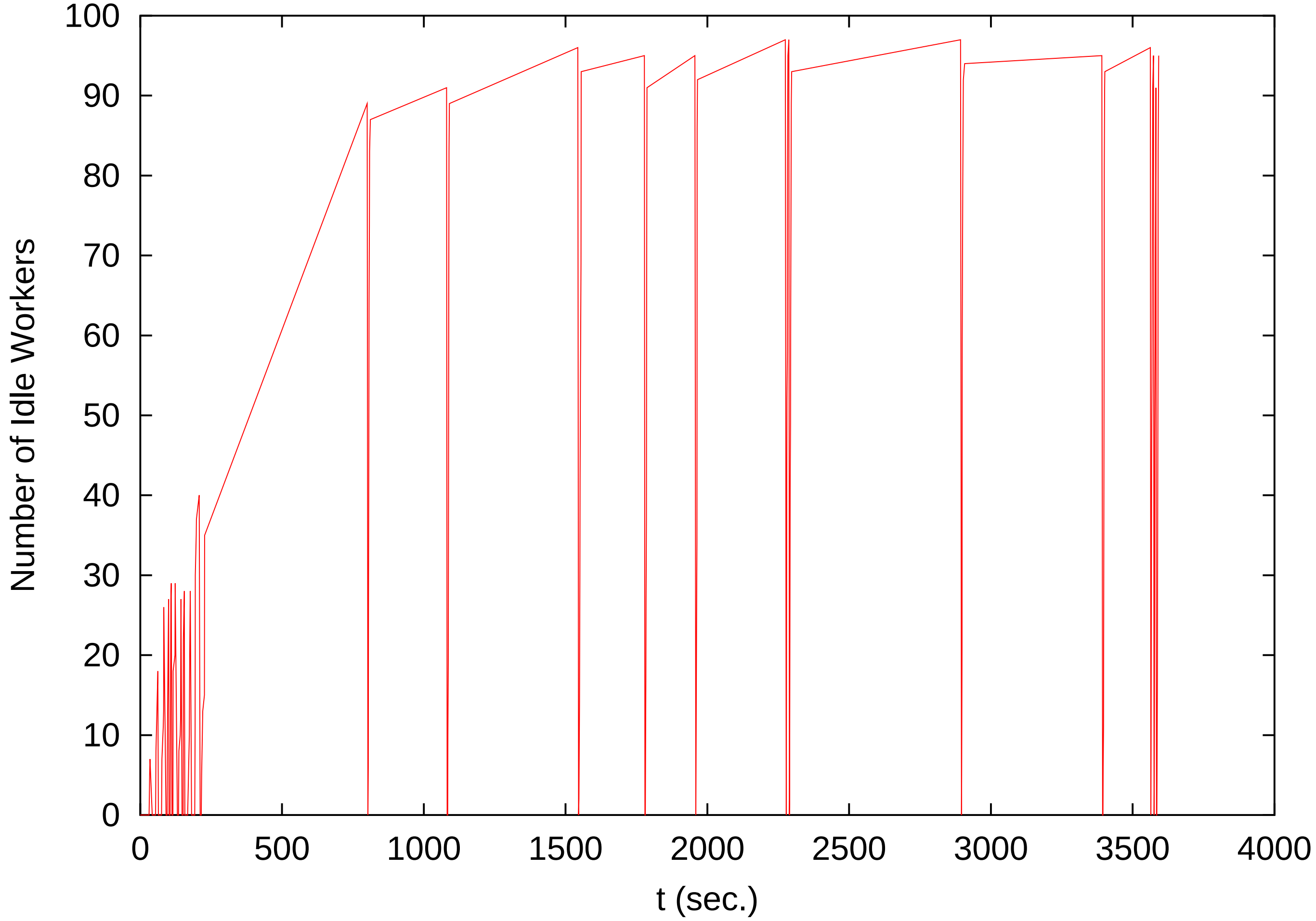
- The evaluation of $\mathcal{Q}(x)$ — solving the different LP's, can be done independently.
  - ◇ If you have K computers, send them each one of K chunks, and your evaluation of $\mathcal{Q}(x)$ will be completed K times faster.
- Work
  - ◇ One or more scenario chunks $\mathcal{S}_{j_1}, \ldots \mathcal{S}_{j_C}$ and point $(\hat{x})$
- Result
  - ◇ A subgradient of each of the $\mathcal{Q}_{[S_k]}(\hat{x})$.
- How many chunks to send?

Contention



- If task size is too small, the master is overwhelmed with requests, reducing overall efficiency
- What else can impact the parallel efficiency?
  - Solving the master problem. (This is why we should be able to delete cuts).

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 C^{-1} machines

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

ATR – An Asynchronous Trust Region 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
- Convergence theory and cut deletion theory is similar to the synchronous algorithm
- 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

The World's Largest LP

- Storm – A cargo flight scheduling problem (Mulvey and Ruszczyński)
- We aim to solve an instance with 10,000,000 scenarios
- $x \in \Re^{121}, y(\omega_i) \in \Re^{1259}$
- The deterministic equivalent is of size

$$A \in \Re^{985{,}032{,}889 \times 12{,}590{,}000{,}121}$$

- Cuts/iteration 1024, # Chunks 1024, $|\mathcal{B}| = 4$
- Started from an $N = 20000$ solution, $\Delta_0 = 1$
- CPLEX used to solve the master LP. Soplex used to solve (most of) the worker LPs.

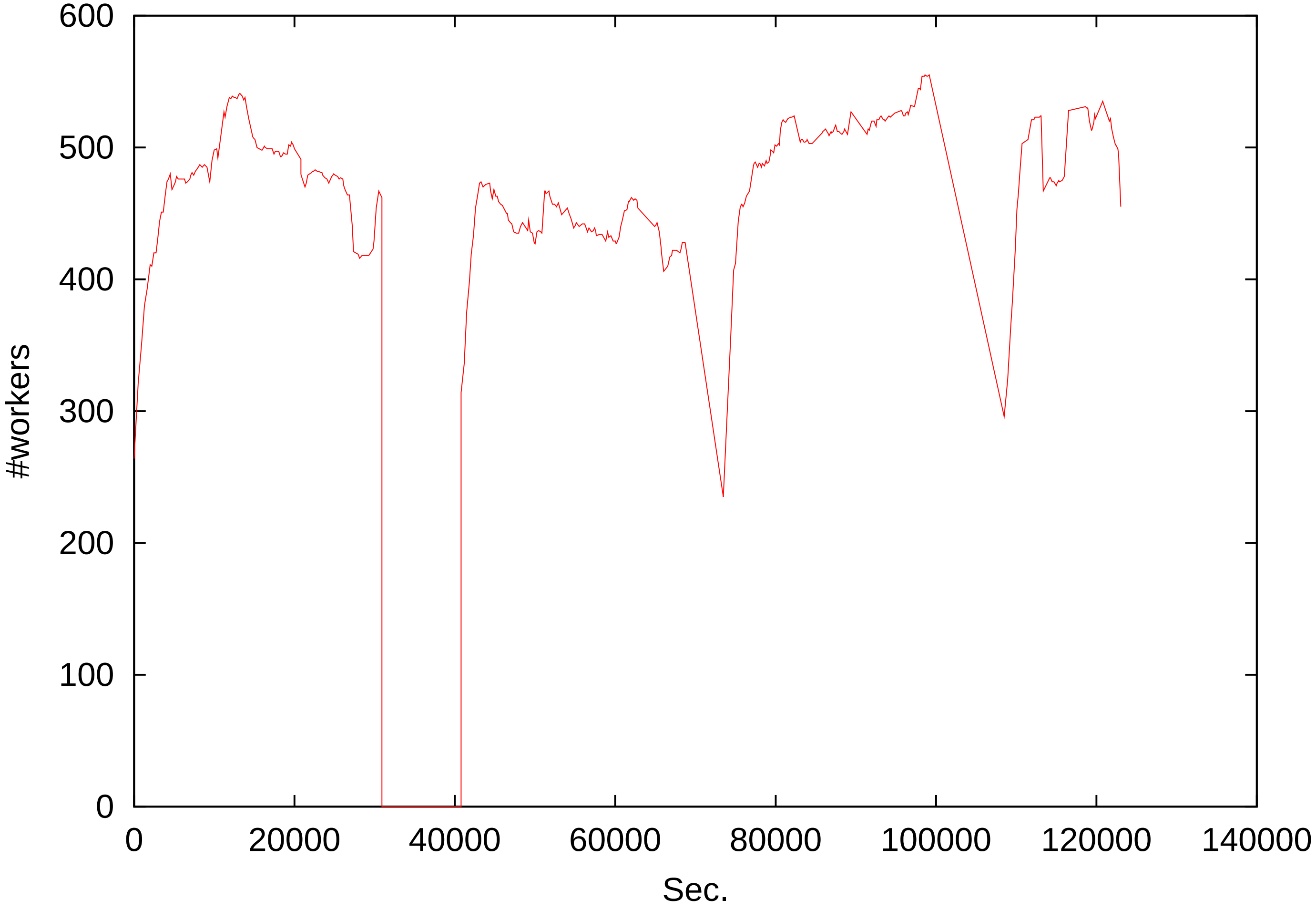
The Super Storm Metacomputer

Number	Type	Location
184	Intel/Linux	Argonne
254	Intel/Linux	New Mexico
36	Intel/Linux	NCSA
265	Intel/Linux	Wisconsin
88	Intel/Solaris	Wisconsin
239	Sun/Solaris	Wisconsin
124	Intel/Linux	Georgia Tech
90	Intel/Solaris	Georgia Tech
13	Sun/Solaris	Georgia Tech
9	Intel/Linux	Columbia U.
10	Sun/Solaris	Columbia U.
33	Intel/Linux	Italy (INFN)
1345		

TA-DA!!!!!

Wall clock time	31:53:37
CPU time	1.03 Years
Avg. # machines	433
Max # machines	556
Parallel Efficiency	67%
Master iterations	199
CPU Time solving the master problem	1:54:37
Maximum number of rows in master problem	39647

Number of Workers



Next Time

- Bounds
- Algorithms Based on Bounds

⋆ New Assignment. Due Monday.

- Prepare a one or two page description of their project.
  - ◇ Background
  - ◇ Technique
  - ◇ Tools they will use
  - ◇ Any assistance or additional background they need to succesfully complete their project
  - ◇ Desired Conclusion of the project