Using a Computational Grid for Optimization

JEFF LINDEROTH
ISE Department
COR@L Lab
Lehigh University
jtl3@lehigh.edu

Department of Industrial & Systems Engineering
University of Wisconsin-Madison
February 20, 2006

The Perfect Marriage

While my wife really likes this slide, I believe that Optimization and the Computational Grid really form the perfect marriage.

Ingredients of the Perfect Marriage

Something Old
- Branch-and-Bound
- Benders' Decomposition

Something New
- Parallel Tree Search Techniques
- Nonlinear Relaxations
- Asynchronous Algorithms
- Symmetry Handling

Something Borrowed
- Your Computer
- Cycles on Supercomputer Sites

Something Blue

You can work until you’re blue in the face, and still not solve some of these problems:
- Quadratic Assignment Problem
- Multistage Stochastic Programming
- Symmetric Integer Programs
What’s All the Hype?

**Computational Grid**

A hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.

**Problems With Using Supercomputers**

- Nearly 10,000 pending jobs!
- You can queue your job and wait (literally) days until it will run.
- This hardly seems consistent or pervasive.

**High End Computational Capabilities**

**The TeraGrid**

- > 11,000 Processors
- > 45 TFlops

**NCSA — National Center for Supercomputing Applications**

1774 CPU, Itanium-2 processors

**On a Positive Note**

- How Many Processors Are Available?

```bash
[linderot@tunb ~] bhosts | grep 'ok' | wc -l
231
```

- There are 231 processors that are currently available!
- They are being saved to run the parallel job at the head of the queue
- “Backfill”: We could use those processors for our computation, but we have to schedule them for a short time period
- Use the processors as part of a larger computation that can handle processors going away
- The idle cycles are pervasive!
Something Borrowed—Your CPU Cycles

- There are lots of CPU cycles going unused (or wasted) right now.
- Can I use your machine?
- I promise to give it back as soon as you want it
- CPU Cycles are a ubiquitous and nearly endless resource, if only you can harness them
- But how should they be harnessed?

Building Grids with 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!
- How does it do this?
  - Scheduling/Matchmaking
  - Jobs can be checkpointed and migrated
  - Remote system calls provide the originating machines environment
- Flocking: Jobs in one Condor Pool can negotiate to run in other Condor pools
- Glide-in: Nodes can “temporarily” join an existing Condor pool.
Grid-Enabling Algorithms

- Condor, with flocking and glide-in, gives us the infrastructure from which to build a grid (the spare CPU cycles).
- We still need a mechanism for controlling the algorithm on a computational grid.
- No guarantee about how long a processor will be available.
- No guarantee about when new processors will become available.

- To make parallel algorithms dynamically adjustable and fault-tolerant, we could (should?) use the master-worker paradigm.
- What is the master-worker paradigm, you ask?

Master-Worker!

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

Simple!
- Fault-tolerant
- Dynamic

Making the Marriage Work

- There are three abstraction in the master-worker paradigm: Master, Worker, and Task.
- MW is a software package that encapsulates these abstractions.
  - API: C++ abstract classes
  - User writes 10 methods
  - The MWized code will transparently adapt to the dynamic and heterogeneous computing environment.
- MW also has abstract layer to resource management and communications packages (an Infrastructure Programming Interface).
  - Condor/PVM, Condor/Socket, Condor/Files, Single processor
- It's Free!: http://www.cs.wisc.edu/condor/mw

MW API

- MWMaster
  - get_userinfo()
  - setup_initial_tasks()
  - pack_worker_init_data()
  - act_on_completed_task()
- MWTask
  - pack_work(), unpack_work()
  - pack_result(), unpack_result()
- MWWorker
  - unpack_worker_init_data()
  - execute_task()
MW Applications

- **MWFACTCOP** (Chen, Ferris, L) – A branch and cut code for linear integer programming
- **MWATR** (L, Shapiro, Wright) – A trust-region-enhanced cutting plane code for two-stage linear stochastic programming and statistical verification of solution quality.
- **MWKNAK** (Glankwamdee, L) – A simple branch-and-bound knapsack solver
- **MWQAP** (Anstreicher, Brixius, Goux, L) – A branch-and-bound code for solving the quadratic assignment problem
- **MWAND** (L, Shen) – A nested decomposition-based solver for multistage stochastic linear programming
- **MWSYMCOP** (L, Margot, Thain) – An LP-based branch-and-bound solver for symmetric integer programs

MWKnap

- Simple (self-contained) branch-and-bound solver included in MW distribution

Knapsack Problem

\[ z^* = \max\{c^T x \mid a^T x \leq b, x \in \mathbb{B}^n\} \]

- Use to demonstrate how not to implement branch-and-bound with MW
- MW: Task consists of work and result

Stoopid Ideas

**Stoopid Idea #1**

Let the “work” portion of task consist of evaluating one node

- Parallel Efficiency:
  \[ \eta = \frac{\Sigma \text{(Time workers execute tasks)}}{\Sigma \text{(Time workers available)}} \]
  \[ \eta < 1\% \]

More Stoopid Ideas

**Stoopid Idea #2**

Let the workers search the trees in a best-first fashion

- The messages you pass back to the master get huge
- The master quickly exhausts all available memory

Memory Consumption
Something New

Parallel MW Tree Management Strategy

- **Master:**
  - Order list best-first
  - Switch to worst-first if the size of the list gets too big
- **Worker:**
  - Search subtree rooted at task’s node in a depth-first fashion for t seconds.
  - Pass back unevaluated nodes in stack to form new tasks.

- **In Initial Implementation, η = 0.41**
- Since there is very little synchronization required in the branch and bound algorithm, this number is shockingly low!

Deducing the Problem

- We may want the workers to examine a subtree for t seconds, but that doesn’t mean that there are t seconds of work!
- A histogram of task times. (For t = 100)

Elementary, Dear Watson

- Make sure that workers only pass back nodes that will have enough “meat”
- Allow additional time for workers to pop up the DFS stack, finishing off remaining easy nodes.
- η improved to 0.9

QAP Collaborators

- **Kurt Anstreicher**
  - University of Iowa

- **Jean-Pierre Goux**
  - Argonne, Northwestern, and Artelys

- **Nate Brixius**
  - Microsoft
### The Quadratic Assignment Problem

#### Mathematical Formulation

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

- Assign facilities to locations
- QAP is NP-Super-Hard
- Branch and Bound is the method of choice, but very few tight, computable, bounds exist.

### Odds and Ends

**Something New**

Kurt and Nate gave a tight nonlinear relaxation

\[
\min_{x} \text{vec}(X)^T ((B \otimes A) - (I \otimes S) - (T \otimes I)) \text{vec}X + C \cdot X
\]

\[\text{s.t. } Xe = X^Te = e, X \geq 0\]

- Convex quadratic relaxation
- **Something Old!** Used the Frank-Wolfe algorithm to solve the relaxation
- **Something Old and New!** Adapt “strong branching” to this context
- Engineer the algorithm for the Grid

### Our Computational Grid

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

### The Holy Grail

- (NUG30) \((n = 30)\) had been the “holy-grail” of computational QAP research for > 30 years
- In 2000\(^1\), Anstreicher, Brixius, Goux, & Linderoth set out to solve this problem
- Using an old idea of Knuth, we estimated the CPU time required to solve NUG30 to be 5-10 years on a fast workstation

\(^{1}\) Something Old!
NUG30 is solved!

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

**NUG30 Computation**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wall Clock Time</td>
<td>6:22:04:31</td>
</tr>
<tr>
<td>Avg. # Machines</td>
<td>653</td>
</tr>
<tr>
<td>CPU Time</td>
<td>(\approx 11) years</td>
</tr>
<tr>
<td>Nodes</td>
<td>11,892,208,412</td>
</tr>
<tr>
<td>LAPs</td>
<td>574,254,156,532</td>
</tr>
<tr>
<td>Parallel Efficiency</td>
<td>92%</td>
</tr>
</tbody>
</table>

**KLAPS**

**Solution of More QAP Instances**

<table>
<thead>
<tr>
<th>Instance</th>
<th>KRA30B</th>
<th>KRA32</th>
<th>THO30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wall Clock Time (Days)</td>
<td>3.79</td>
<td>12.3</td>
<td>17.2</td>
</tr>
<tr>
<td>Avg. # Machines</td>
<td>462</td>
<td>576</td>
<td>661</td>
</tr>
<tr>
<td>Max. # Machines</td>
<td>780</td>
<td>1079</td>
<td>1307</td>
</tr>
<tr>
<td>CPU Time (Years)</td>
<td>4.32</td>
<td>15.2</td>
<td>24.7</td>
</tr>
<tr>
<td>Nodes</td>
<td>(5.14 \times 10^9)</td>
<td>(16.7 \times 10^9)</td>
<td>(34.3 \times 10^9)</td>
</tr>
<tr>
<td>LAPs</td>
<td>(188 \times 10^9)</td>
<td>(681 \times 10^9)</td>
<td>(1.13 \times 10^{12})</td>
</tr>
<tr>
<td>Parallel Efficiency</td>
<td>92%</td>
<td>87%</td>
<td>89%</td>
</tr>
</tbody>
</table>
**Multistage Stochastic LP**

\[ \begin{align*}
    \mathbf{z} &= \min \left\{ \sum_{n \in \mathbb{N}} q_n c_n^T x_n \mid T_n x_{\rho(n)} + W_n x_n = h_n \quad \forall n \in \mathbb{N} \right\} \\
\end{align*} \]

**Multistage Decision Making**

- Random vectors \( \xi_1, \xi_2, \ldots, \xi_T \in \mathbb{R}^{n_T} \)
- Make sequence of decisions \( x_1, x_2, \ldots, x_T \in X_T \).
- Risk Neutral: We always aim to optimize the expected value of our current decision \( x_t \).
- Linear: Assume \( X_t \) are polyhedra
- Discrete: Assume \( \xi_t \) are drawn from a discrete distribution.

**Scenario Tree**

- \( N \): Set of nodes in the tree
- \( \rho(n) \): Unique predecessor of node \( n \) in the tree
- \( S(n) \): Set of successor nodes of \( n \)
- \( q_n \): Probability that the sequence of events leading to node \( n \) occurs
- \( x_n \): Decision taken at node \( n \)

**Multistage Stochastic Programming**

- Solving a multistage stochastic LP equivalent to solving a giant LP known as the Deterministic Equivalent
**Something Old: Nested Benders’ Decomposition**

**Value Function of node \( n \)**

\[
Q_n(x_{\rho(n)}) \overset{\text{def}}{=} \min_{x_n} \left\{ c_n^T x_n + \sum_{m \in S(n)} q_{mn} Q_m(x_n) \middle| W_n x_n = h_n - T_n x_{\rho(n)} \right\}
\]

- Nested Benders’ Decomposition works by building under-approximations to each node’s value function:

\[
Q_n(x_{\rho(n)}) \geq \min_{x_n} \left\{ c_n^T x_n + M^k_n(x_n) \mid W_n x_n = h_n - T_n x_{\rho(n)} \right\}
\]

\((\text{MLP}_n)\)

- Solution of child node’s MLP gives “cuts”, improving \( M^k_n(\cdot) \)

**MWImplementation**

**Grid Programmers Do it in Parallel!**

Nodes in nonoverlapping subtrees may be evaluated independently

- **MWTTask**—Work
  - Collection of nodes (going the same direction) from the same stage
  - The \( x_{\rho(n)} \) from these nodes

- **MWTTask**—Result
  - (Forward): \( x_n \)
  - (Backwards): Cut(s) \( F^k_n[j], f^k_n[j] \)

- \( \text{act\_on\_completed\_task()} \) is responsible for updating node state and deciding which nodes to evaluate next

**Synchronicity is Bad!**

**Number of Idle Workers**

All processors waiting for this node to finish!
MW Implementation—Asynchronous

- It is not necessary to wait for all children \(S_n\) to report in order to start a new evaluation of \(M_n^k\).
- **Something Old:** Ruszczyński ('93) also showed how to do nested decomposition in an asynchronous fashion and ensure convergence.
- **Something New:** We have “reinvented-reinterpreted” some of these results.
- Each node has state \(\{\text{Forward, Backward}\}, \{\mathbf{R}, \mathbf{Y}, \mathbf{G}\}\)  
  - **Red:** There’s nothing useful I can do with this node.  
  - **Yellow:** Node is ready to be evaluated.  
  - **Green:** Node is being evaluated, and waiting for results.
- In `act_on_completed_task()`, node states are examined.
- In this method we relax assumption that new tasks can only be created if all child tasks gave been reported.

A Grid Challenge: Cut Management

- We may require lots of memory to store the cuts
  - **Ex.** 27,000 nodes in period T-1, each node contains 20 cuts, \(x_n \in \mathbb{R}^{100} \Rightarrow \geq 400\text{MB} \text{ just to store cuts}\)
- **Grid:** Since we don’t have guarantees about worker processors, we cannot store cuts on the workers.
- **Grid:** All cuts (must) be stored on the master processor
  - Leads to memory overload of master
  - Leads to increased “service time” of the master for worker requests. (contention)
- **Grid:** We must do what we can to compress and reduce the number of cuts
  - Don’t record duplicates
  - Aggregate nodes

Duplicate Cuts

- Random variables “time-independent”
  - You can “share” cuts among nodes at a stage
  - You need only keep one copy.
- Random variables “time-dependent”
  - It may be the case that the cost-to-go functions are very similar.
  - **Example:** Four stage problem, Nodes/stage (1, 32, 1024, 32768)

<table>
<thead>
<tr>
<th>Period</th>
<th>Cuts</th>
<th>Duplicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>51</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>5055</td>
<td>5009</td>
</tr>
<tr>
<td>3</td>
<td>11264</td>
<td>11256</td>
</tr>
</tbody>
</table>

Cut Management—Aggregation

- Form the deterministic equivalent of a group of nodes, and treat this as one larger “supernode”
- Node subproblems get larger
- Fewer cuts
Telecommunication Network Design

- Set of stages $T$
- Set $J$ of links
- Sets $I_t$ of demands
- Random demand $d_t(\xi) \in \mathbb{R}^{|I_t|}$
- Budget each period
- Install capacity on links each period to minimize the total expected unserved demand

MWAND

- **MW Asynchronous Nested Decomposition**
  - Magic WAND? :-)
- Uses the COIN Osi Interface to build $\text{MLP}_n$
- Uses the COIN Clp (simplex) solver to solve $\text{MLP}_n$
- Does *not* use the COIN-Smi to manipulate stochastic program
- **SUTIL: Stochastic Programming Utility Library (Czyzyk & L)**
  - Reads SMPS
  - Samples instances
  - Creates deterministic equivalents
  - It’s Free! http://coral.ie.lehigh.edu/sutil

Some (Limited) Computational Results

- $T = 5$, Last three periods aggregated.
- Right now, we are using a “baby grid”
- **Helpful Hint:** Don’t unleash your code onto a big grid unless you are reasonable sure it is working well.

<table>
<thead>
<tr>
<th>Location</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wisconsin</td>
<td>785</td>
</tr>
<tr>
<td>NCSA</td>
<td>1280</td>
</tr>
<tr>
<td>Argonne/U of C</td>
<td>288</td>
</tr>
</tbody>
</table>

Computational Results

- **K:** Realizations/Period
- **N:** Number of scenarios
- **DE:** Size of deterministic equivalent

<table>
<thead>
<tr>
<th>K</th>
<th>N</th>
<th>DE Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>0.81M</td>
<td>18M * 31M</td>
</tr>
<tr>
<td>50</td>
<td>6.25M</td>
<td>140M * 236M</td>
</tr>
<tr>
<td>60</td>
<td>12.9M</td>
<td>290M * 488M</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>K</th>
<th>It</th>
<th>Avg Workers</th>
<th>Wall Time</th>
<th>CPU Time</th>
<th>$\eta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>9</td>
<td>62</td>
<td>2:34:21</td>
<td>6:15:15:10</td>
<td>67</td>
</tr>
<tr>
<td>50</td>
<td>7</td>
<td>75</td>
<td>1:12:49:27</td>
<td>85:20:24:15</td>
<td>77</td>
</tr>
<tr>
<td>60</td>
<td>11</td>
<td>162</td>
<td>3:16:51:00</td>
<td>431:12:15:37</td>
<td>73</td>
</tr>
</tbody>
</table>
Code Design

- \( W(\nu, \alpha) \): Set of all “words” of length \( \nu \) from alphabet \( \{0, 1, \ldots, \alpha - 1\} \).
- \( |W(\alpha, \nu)| = \alpha^\nu \)
- We will abbreviate \( W(\nu, \alpha) = W \)
- A code is a subset \( C \subseteq W \)
- Hamming distance: \( \alpha \in W, b \in W, \dist(\alpha, b) = |\{i : \alpha_i \neq b_i\}| \)

Error Correcting Code

- Find \( C \subseteq W \) such that
  \[ \alpha \in C, b \in C \implies \dist(\alpha, b) \geq 2d + 1 \]
- Maximize \( |C| \)
- Application: Words in \( C \) submit over a “noisy” channel on which at most \( d \) letters are changed can be “self-corrected.”

Covering Code

- Find a code \( C \subseteq W \) such that every word \( w \in W \) is at most a distance \( d \) away from at least one word in \( C \)
  \[ \dist(w, C) \leq d \ \forall w \in W \]
- Minimize \( |C| \)
- Application: Something far more practical
Are You Ready for Some Football!

- Predict the outcome of $v$ soccer matches
  - $\alpha = 3$
    - 0: Team A wins
    - 1: Team B wins
    - 2: Draw
- You win if you miss at most $d = 1$ games

**The Football Pool Problem**
What is the minimum number of tickets you must buy to assure yourself a win?

Solutions for $v = 3$

- These solutions are isomorphic.
  - For first component: $2 \leftrightarrow 0$
- There are Lots of isomorphic solutions:
  - “Rename” W,L,D for any subset of the matches: $(\alpha!)^v$
  - Reorder the matches: $v!$
- There are $(\alpha!)^{v!} = 1296$ equivalent solutions for $v = 3$

How Many Must I Buy?

<table>
<thead>
<tr>
<th>$v$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>C^*_v</td>
<td>$</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

- Despite significant effort on this problem for $> 40$ years, it is only known that $65 \leq C^*_6 \leq 73$

**The Football Pool Problem**
What is $|C^*_6|$?

But It’s Trivial!

- For each $j \in W$, let $x_j = 1$ iff we want $j$ is in code $C$
- Let $A \in \{0,1\}^{|W| \times |W|}$ with $a_{ij} = 1$ iff word $i \in W$ is distance $\leq d = 1$ from word $j \in W$

**IP Formulation**

$$\min e^T x$$

s.t. $Ax \geq e$

$x \in \{0,1\}^{|W|}$
CPLEX Can Solve Every IP

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Objective</th>
<th>IInf</th>
<th>Best Integer</th>
<th>Cuts/</th>
<th>ItCnt</th>
<th>Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node Left</td>
<td>56.0769</td>
<td>729</td>
<td>56.0769</td>
<td>729</td>
<td>2200</td>
<td>76.92%</td>
</tr>
<tr>
<td>* 0+ 0</td>
<td>56.0769</td>
<td>0</td>
<td>110.0000</td>
<td>56.0769</td>
<td>2200</td>
<td>49.02%</td>
</tr>
<tr>
<td>* 0+ 0</td>
<td>56.5164</td>
<td>729</td>
<td>110.0000</td>
<td>56.5164</td>
<td>2542</td>
<td>48.62%</td>
</tr>
<tr>
<td>* 0+ 0</td>
<td>56.5279</td>
<td>729</td>
<td>107.0000</td>
<td>56.5279</td>
<td>2673</td>
<td>47.17%</td>
</tr>
<tr>
<td>* 0+ 0</td>
<td>56.5279</td>
<td>0</td>
<td>94.0000</td>
<td>56.5279</td>
<td>2673</td>
<td>39.86%</td>
</tr>
<tr>
<td>* 0+ 0</td>
<td>56.5279</td>
<td>0</td>
<td>93.0000</td>
<td>56.5279</td>
<td>2673</td>
<td>39.22%</td>
</tr>
</tbody>
</table>

Isomorphism Pruning

- For some permutation $g \in G(IP)$ and set of indices $S \subset \{1, 2, \ldots, n\}$, let

$$g(S) = \{g(i) \mid i \in S\}$$

- At a node $\alpha$ of the branch-and-bound tree
  - $F^\alpha_1 = \{i \mid x_i \text{ fixed to 1 at } \alpha\}$
  - $F^\alpha_0 = \{i \mid x_i \text{ fixed to 0 at } \alpha\}$

- Nodes $\alpha$ and $b$ are isomorphic if

$$\exists g \in G(IP) \text{ with } g(F^\alpha_1) = F^b_1, g(F^\alpha_0) = F^b_0$$

- You may prune one of $\alpha$ or $b$. [Bazaraa, Kirca 83]

What’s the Problem!?: Symmetry

- $\pi: \text{ Permutation of } \{1, 2, \ldots, n\}$
  - $\pi(x) = \pi(x_1, x_2, \ldots x_n) = (x_{\pi(1)}, x_{\pi(2)}, \ldots x_{\pi(n)})$
  - $\pi$ is a symmetry of an IP if $x \text{ feasible } \iff \pi(x) \text{ feasible}$
  - $c^T x = c^T \pi(x)$

- $G(IP): \text{ Set of all symmetries of IP}$
  - For covering design, $|G(IP)| = v!(\alpha)^v$
  - $6! \times 3^6 = 524880$

Something Old: Minimum Index Branching

[Butler, Ivanov, Lam, Margot, McKay, Read, Stinson, ...]

- The set $\{g(S) \mid g \in G(IP)\}$ is an equivalence class of all equivalent “relabelings” of $S \subset \{1, 2, \ldots, n\}$

- Choose one representative for each potential set of variable fixings.

- For example, in Minimum Index Branching, A set $S$ is a representative of its equivalence class if

$$S = \text{lexmin}\{g(S) \mid g \in G(IP)\}$$

- Isomorphism Pruning:

  - If $F^\alpha_1$ is not a representative, then prune node $\alpha$. 

Results (All Thanks to François)

The Good

- For \( d = 1, v = 5, \alpha = 3 \), Isomorphism Pruning can establish \(|C^*_5| = 27\) in 1409 nodes, 82 seconds.
- CPLEX (v9.1) does not solve the problem in more than 4 hours.

The Bad and Ugly

- For \( d = 1, v = 6, \alpha = 3 \), Isomorphism Pruning gets nowhere.
- \(|C^*_6| \geq 61\) after long running time.

Subcodes

- Partition \( W \) into words that start with each letter
  - \( W(6,3) = W_0 \cup W_1 \cup W_2 \)
  - \( w \in W_0 \) covers 11 words in \( W_0 \)
  - \( w \in W_1 \) covers 1 words in \( W_0 \)
  - \( w \in W_2 \) covers 1 words in \( W_0 \)
- An optimal code has
  - \( C_0 \subset W_0, |C^*_0| = y_0 \)
  - \( C_1 \subset W_1, |C^*_1| = y_1 \)
  - \( C_2 \subset W_2, |C^*_2| = y_2 \)
- So if a code of size \(|C^*| = M\) exists, then it must satisfy

\[
\begin{align*}
11y_0 + y_1 + y_2 & \geq 243 \\
y_0 + 11y_1 + y_2 & \geq 243 \\
y_0 + y_1 + 11y_2 & \geq 243 \\
y_0 + y_1 + y_2 & = M
\end{align*}
\]

Extending the Idea

Something New!

Combine the subcode fixing with IP.

- For some (small) \( m \), and optimal code size \( M \), enumerate all non-isomorphic solutions to the covering system
- This gives a list of possible \( y \) values, e.g. for \( m = 1 \) you get a list of triples \( (y_0, y_1, y_2) \)
- For each member of the list, solve the “Sequence IP”

Subcodes

- Partition \( W \) into words that start with each letter
  - \( W(6,3) = W_0 \cup W_1 \cup W_2 \)
  - \( w \in W_0 \) covers 11 words in \( W_0 \)
  - \( w \in W_1 \) covers 1 words in \( W_0 \)
  - \( w \in W_2 \) covers 1 words in \( W_0 \)
- An optimal code has
  - \( C_0 \subset W_0, |C^*_0| = y_0 \)
  - \( C_1 \subset W_1, |C^*_1| = y_1 \)
  - \( C_2 \subset W_2, |C^*_2| = y_2 \)
- So if a code of size \(|C^*| = M\) exists, then it must satisfy

\[
\begin{align*}
11y_0 + y_1 + y_2 & \geq 243 \\
y_0 + 11y_1 + y_2 & \geq 243 \\
y_0 + y_1 + 11y_2 & \geq 243 \\
y_0 + y_1 + y_2 & = M
\end{align*}
\]

Extending the Idea

Something New!

Combine the subcode fixing with IP.

- For some (small) \( m \), and optimal code size \( M \), enumerate all non-isomorphic solutions to the covering system
- This gives a list of possible \( y \) values, e.g. for \( m = 1 \) you get a list of triples \( (y_0, y_1, y_2) \)
- For each member of the list, solve the “Sequence IP”
Sequence IP $(M, y_0, y_1, y_2)$

\[
\begin{align*}
\min & \quad e^T x \\
\text{s.t.} & \quad Ax \geq e \\
& \quad \sum_{i \in W_0} x_i = y_0 \\
& \quad \sum_{i \in W_1} x_i = y_1 \\
& \quad \sum_{i \in W_2} x_i = y_2 \\
& \quad e^T x \leq M \\
& \quad x \in \{0, 1\}^{|W|}
\end{align*}
\]

Improving the Lower Bound on $C_6^*$

If you find no solution, then $M + 1$ is a valid lower bound.

How to Solve It: SYMCOP

- There were even more “pre-pre” processing tricks that were used to reduce the number of sequence IPs to solve.
- Final total: 346 sequence IPs.
- François and Jeff implemented these ideas using the old MW-FATCOP framework for MILP.
- Many of the engineering/tuning ideas from the knapsack and QAP experience were used in this implementation.
- Greg has made numerous improvements in MW’s robustness and has been great at scavenging cycles in an effort to solve this problem.
- Our mission: establish $C_6^* \geq 70$.

Not For a Lack of Trying!

Doh!

- I really, really, really wanted to announce that $C_6^* \geq 70$ today.
- Sadly, only 284 of the 346 sequence IPs have been completed.
- Our Personal Condor master machine is lying in pieces.
- But it has been working hard!

Statistics so far...

- Wall Time: 30.7 days
- CPU Time: 36.24 years
- Avg Workers: 455.8
- Max Workers: 1253
- Total Nodes: $6.51 \times 10^8$
- Total LP Pivots: $4.46 \times 10^{11}$
- Parallel Performance: 95.6%
Conclusions

- Optimization and the Grid really are “The Perfect Marriage”
- As my wife Helen so often tells me, even the most perfect marriage requires a little bit of work
- You may need to “tune” your optimization algorithm to run nicely on a Grid platform
  - Exploit dynamic grain size
  - Re-examine search strategy
  - Make algorithm run asynchronously
- The boost in computing power that you get by making your code/algorithm Grid-enabled is worth the effort!

It’s Not All “On The Grid”

Recent Applications

- Stochastic Network Interdiction
- Stochastic Vehicle Routing

Recent Work with Companies

- Agere Systems: Product Portfolio Management
- Air Products and Chemicals: Rescheduling for Bulk-Gas Production/Distribution
- Air Products and BOC Gases: Product Swap Contract Valuation
- FCC: Implementation for Combinatorial Auction

But Wait, There’s More

Other Current Research Areas

- Computational Integer Programming
  - Georgia Teach & UW-Madison
  - SAS Institute
- Mixed Integer Nonlinear Programming
  - Argonne National Lab
- Quasi-Monte Carlo Sampling for Stochastic Programming
  - Northwestern University and UW-Madison

The End

- MW: http://www.cs.wisc.edu/condor/mw
- COR@L: http://coral.ie.lehigh.edu
- mailto:jtl3@lehigh.edu

Optimization and the Grid really are “The Perfect Marriage”
As my wife Helen so often tells me, even the most perfect marriage requires a little bit of work
You may need to “tune” your optimization algorithm to run nicely on a Grid platform
  - Exploit dynamic grain size
  - Re-examine search strategy
  - Make algorithm run asynchronously
The boost in computing power that you get by making your code/algorithm Grid-enabled is worth the effort!