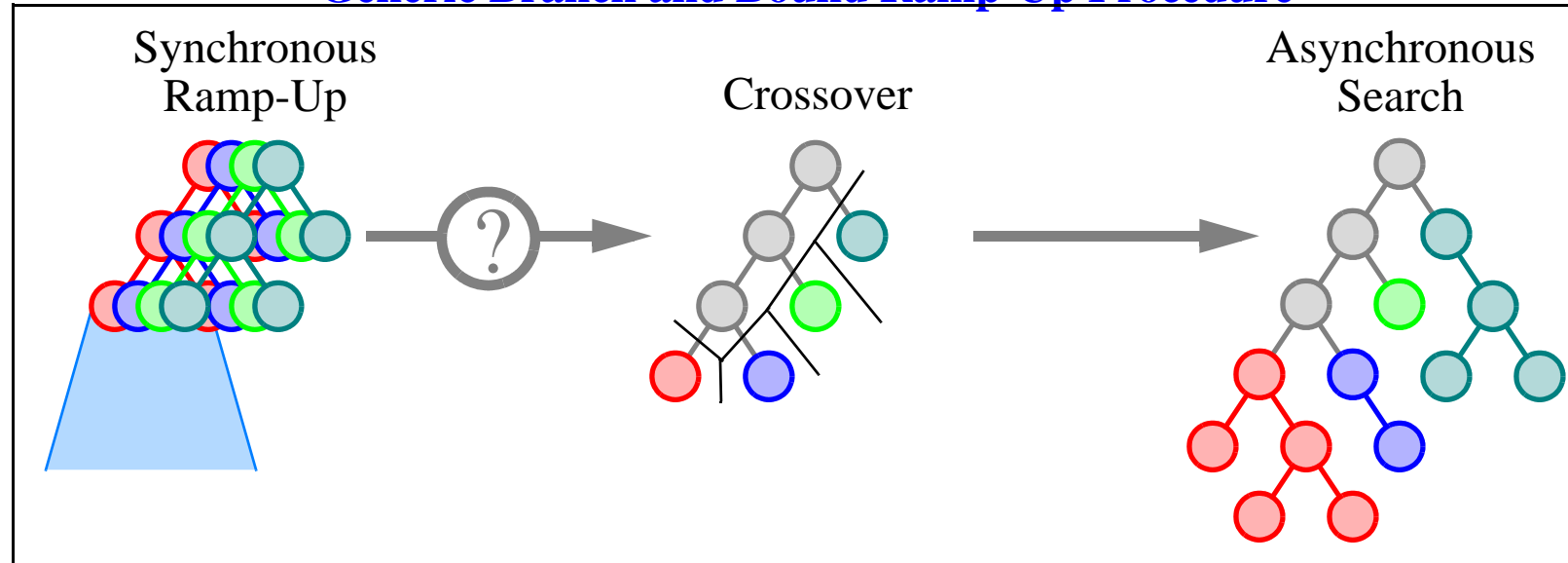


# Rounding Heuristics and Ramp-Up Procedures for Parallel MIP

Jonathan Eckstein, Business School and RUTCOR, Rutgers University

Various portions joint with Cindy Phillips (Sandia National Labs), Bill Hart (Sandia National Labs), Mikhail Nediak (Queens University), Konrad Borys (RUTCOR)  
July 2005; Supported in part by USA NSF Grant CCR-9902092

## Generic Branch and Bound Ramp-Up Procedure



- *Ramp-Up:* all processors redundantly develop top of tree, synchronously parallelizing some of each subproblem's work
- Application-specified virtual function decides when tree parallelism is likely to be better
- *Crossover:* partition tree evenly (no commucation!)
- Then start usual *asynchronous search*

### Specialization to MIP: during ramp-up, use synchronous parallelism for

- Pseudocost initialization (“Lazy strong branching”):
  - The *first* time you encounter  $x_j$  being fractional, “probe” subproblem bounds
  - Subsequently, just track average behavior from actually branching on  $x_j$
- Heuristic

## Crossover:

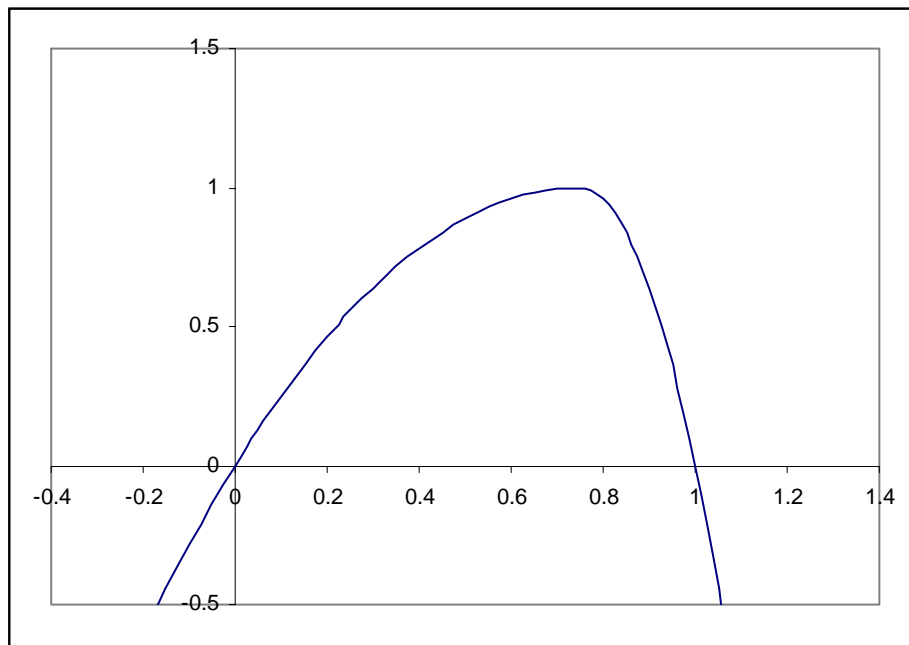
- Let  $P =$  pool size,  $p =$  number of processors
- Use exponential smoothing to track average number of new fractional variables:

$$\tilde{k} \leftarrow \lambda \tilde{k} + (1 - \lambda)k$$

- End ramp-up if  $P \geq \beta p$  or  $P \geq \gamma \tilde{k}$

## Merit-Function-Based MIP Heuristic

Use *merit function* to measure integrality for binary variables: differentiable and strictly concave



$\mathcal{C}^1$  quadratic spline defined by

- $\phi(0) = 0$
- $\phi(\alpha) = 1$
- $\phi'(\alpha) = 0$
- $\phi(1) = 0$

$\alpha = 0.75$  shown

## “Lite” version of heuristic for use with branch and bound

- Couple closely to branch-and-bound process
- Start from solution to current subproblem, with same bounds
- Try not to run for very long at a time

*Sum/Frank-Wolfe approach: use weighted combination of merit function and original objective*

$$\widehat{\psi}(x) = \psi(x) + w c^T x$$

- Use Frank-Wolfe method to find local min of  $\widehat{\psi}$  (no line search needed)
- If  $\psi(x) = 0$ , stop with an integer point.
- Otherwise, add Gomory cuts and repeat (use globalized version of COIN's [CglGomory](#))
- Handle general integer variables by converting to a small number of binary variables representing a range around the current value

### Where's the Parallelism for Ramp-Up?

#### 1. Randomize $\psi$

- For each  $x_j$  that's supposed to be integer, use the merit function  $\phi_\alpha$ , where  $\alpha$  is random
- $\alpha$ 's vary randomly with both variables and processors
- Processors simultaneously execute heuristic with different merit functions

#### 2. "Pre-splitting"

- Fix some (small) subset of the currently fractional variables (Note: can count SOS sets as "variables" in this procedure)
- On different processors,
  - Fix different subsets of variables and/or...
  - Fix same subset to different value combination

... and combinations of these two

## Pre-Splitting Configuration

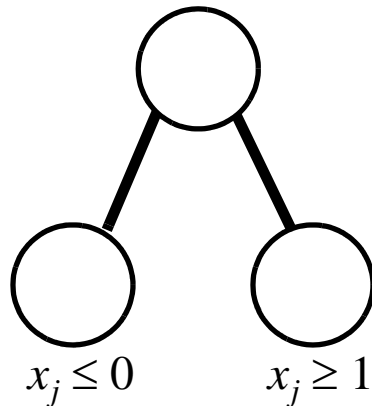
$\tilde{m}$  is the “preferred” number of different merit functions;  $p$  is the number of processors. Compute the desired number of pre-splitting configurations,

$$\tilde{\lambda} = \max \left\{ \left\lfloor \frac{p}{\tilde{m}} \right\rfloor, 1 \right\}$$

Use pseudocost probing to gather information on which variables affect integrality the most

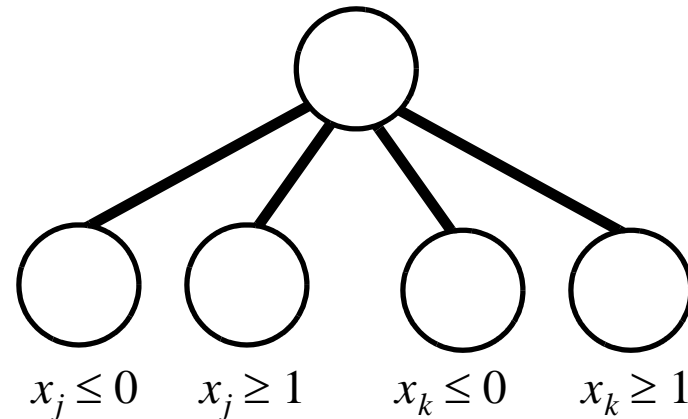
- When you pseudocost-probe  $x_j$ , record how much its integrality increases in the up and down branches; call changes  $\delta_j^+$ ,  $\delta_j^-$
- Sort the currently fractional variables by  $\max\{\delta_j^+, \delta_j^-\}$
- Highest variables are most attractive for pre-splitting
- Also gather data on which variables stay fractional when you branch up/down on each  $x_j$

### Build Pre-Splitting Tree



*Overlapping* splits allowed!

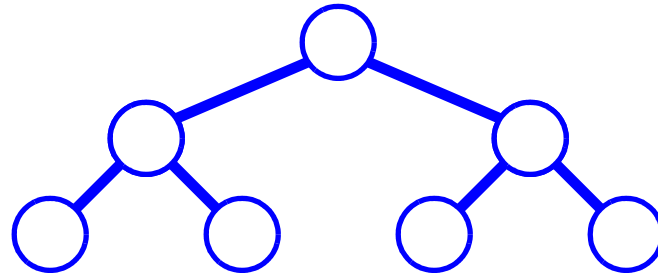
or...



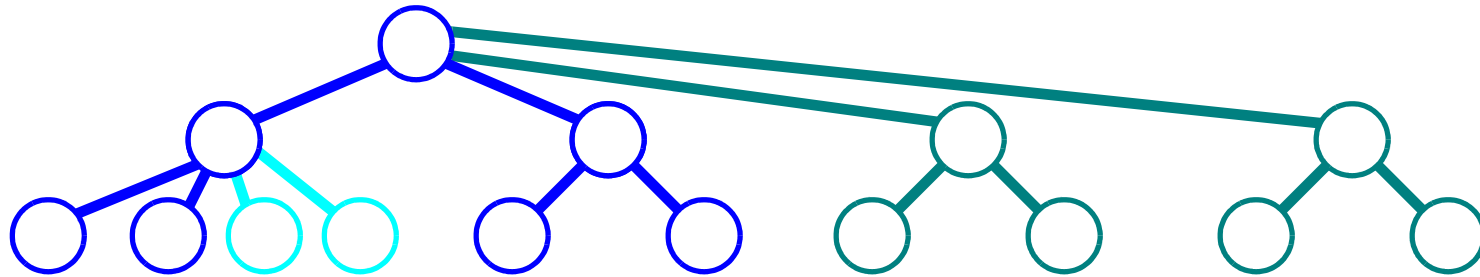
- Each node has a list of further possible splitting variables.
- Start with root, with list = {all fractional variables in subproblem}
- We specify a *depth limit*  $\tilde{d}$

While possible, until there would be more than  $\tilde{\lambda}$  leaves, pick node

- That is above the depth limit, and has a nonempty list
- Has already been split as little as possible
- Is as high as possible
- Add a (possibly overlapping) split on highest-ranked variable  $x_j$ 
  - Delete variables from children's lists if they are "in conflict" with  $x_j$
- Effect: build tree as symmetrically as possible down to depth  $d$ .



- Then sweep down from root again, adding overlapping splits:



### Allocating Pre-Splitting Configurations to Processors

- When the loop ends, there will be  $1 \leq \lambda \leq \tilde{\lambda}$  leaf nodes.
- Apportion an approximately equal number of processors to each leaf node (about  $p/\lambda$ ).
- Each processor selects a random merit function.
- Optionally, force one processor per leaf node to use the canonical  $\alpha = 0.5$  merit function.
- Each processor fixes variables according to its leaf node, and then runs the heuristic.

## Computational Results

Hardware platform: 32 dual Xeon 2.8 GHz = 64 processors, two dual Xeon 3 GHz control nodes, gigabit ethernet w/switch, RH 9 Linux on all nodes

LP solver is CLP (old version for technical reasons)

### Test problems:

- 52 problems from MIPLIB 3.0
- Remaining problems have very slow root LP's or crash in CLP (very likely due to old CLP version; should change soon)

End ramp-up based on  $\beta = \gamma = 1$  criterion

Abort each problem after ramp-up, and check gap from known optimum

Caveats: Cuts used in heuristic, but not bounding (should change soon)

Target merit functions:  $\tilde{m} = 1, 2, 4, 8, 16, 32, 64$

Depth limit:  $\tilde{d} = 0, 1, 2, 3, 4, 5, 6$

Force one processor per group to use canonical merit function, or don't bother

All sensible combinations of  $\tilde{m}, \tilde{d}$  for 64 processors

“Conflict” between variables defined statically -- two variables are in conflict if they appear in any common constraint

## Results: Big Picture

- 10 problems (19%): no solution found for any parameter setting
- 31 problems (60%): solution found for *every* parameter setting
- 11 problems (21%): solution found for some but not all settings
- Between 4 and 18 calls to heuristic, with average of 9.5

### Baseline comparison to runs without heuristic:

- With best-first search: no feasible solutions at all
- Diving on integrality: solutions found to only 5 problems (10%)

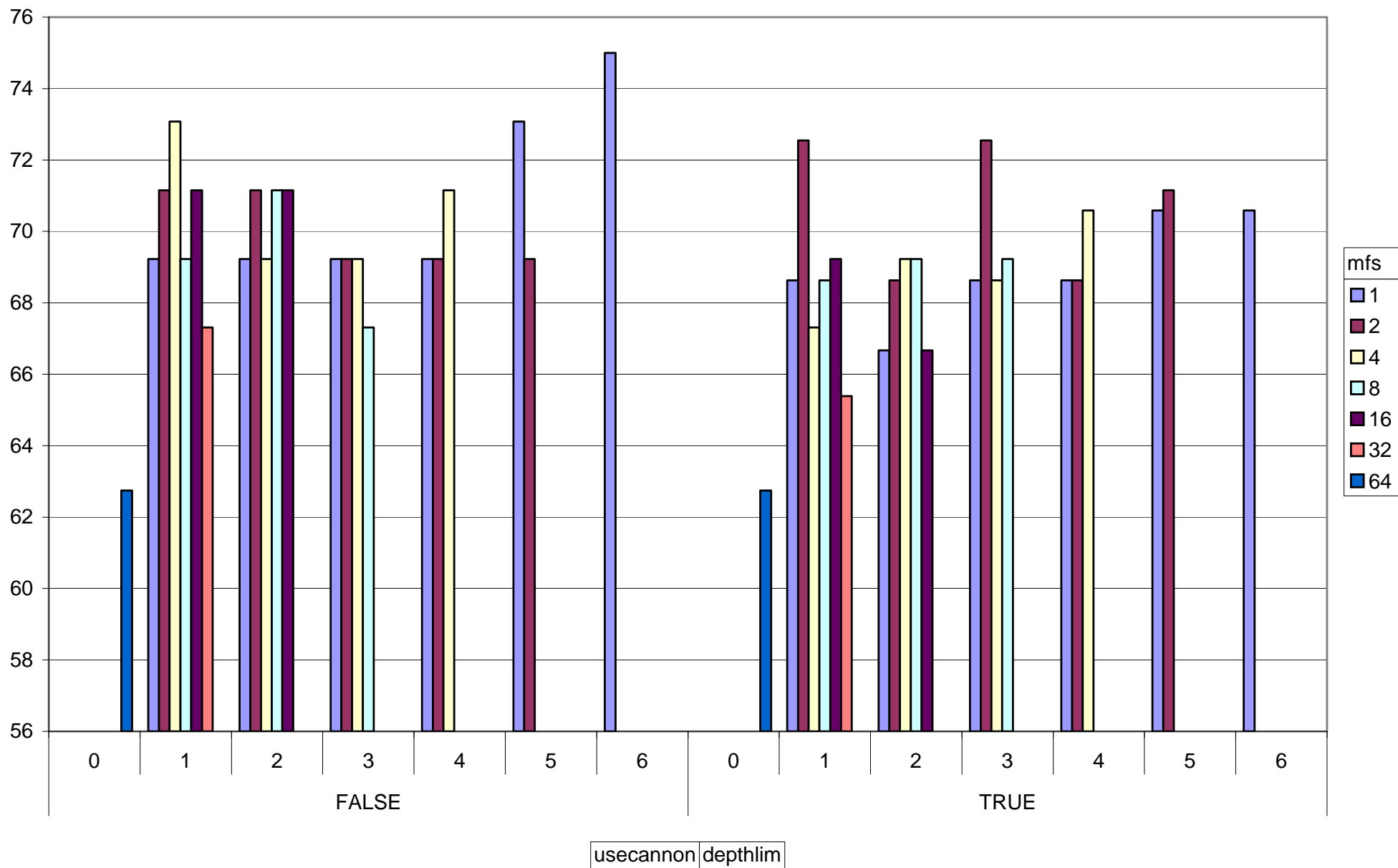
### Run time (in seconds) of ramp-up phase

Method	Average	SD
Heuristic (all settings)	9.8	19.0
No heuristic, best-first	2.9	10.1
No heuristic, diving	2.2	7.5

### Aside: experiments with empirical conflict graph:

- Define conflict dynamically from pseudocost probing: if probing  $x_j$  leaves  $x_l$  fractional, they are not in conflict
- Requires significant extra communication between processors after pseudocost probing
- Experiments suggest there is no significant benefit over a static conflict graph

# Percent Success in Finding Feasible Solutions



# Median Percent Gap From Integer Optimum

