This research is funded by NSF, CMMI and CIEG 0521953:
Exploiting Cyberinfrastructure to Solve Real-time Integer Programs

A Parallel Macro Partitioning Framework for Solving Mixed Integer Programs

Mahdi Namazifar
Industrial and Systems Engineering Department
University of Wisconsin-Madison

Andrew J. Miller
RealOpt, INRIA Bordeaux Sud-Ouest
Université de Bordeaux 1

(with thanks to Michael C. Ferris)

ARS08 Workshop, LIX, Ecole Polytechnique, Paris

Outline

- Background
- Parallel branch-and-bound: current state of the art
- Parallel computing architectures
  - Massively parallel
  - Grid computing
- Challenges in parallelizing MIP solvers
- MIP heuristics
- A Macro Partitioning Approach
  - Brancher
  - Assigner
  - Workers
- Early computational results
- To-do lists

Background

We consider a general 0-1 mixed integer programming (**MIP**) problem

$$\begin{aligned} \min_{(x,y)\in \mathbb{R}^n \times \mathbb{R}^p} \quad & cx + fy \\ \text{s.t.} \quad & Ax + Gy \geq b \\ & x \geq 0, y \in \{0,1\}^p \end{aligned}$$

THE DIFFICULTY: Problems of realistic size are often hard to solve…and even harder to solve *quickly.*

THE OPPURTUNITY: Increasing availability of multiple CPUs in parallel architectures.

THE CHALLENGE: How can we fully exploit available computational resources in the solution of large MIPs?

Example: MIP2003

- A library of test problems, available at http://miplib.zib.de/
- These problems come from a variety of applications and many remained *unsolved* for years
  - ***atlanta-ip*** (21732 constraints 106 general integer variables 46667 binary variables, 1965 continuous variables): unsolvable before 2006, Xpress-MP can now solve this in about five hours with specified settings
  - ***protfold*** (2112 constraints,1835 binary variables): Xpress-MP can solve this in several days on a dual core machine with optimized settings
  - ***dano3mip*** (3202 constraints, 13321 continuous variables, 552 binary variables): still unsolved
  - Numerous other problems are still unsolved, or take many hours or even days of computation time to solve. These instances are ***not*** particularly large!

NSF-CMMI 0521953: Real-Time Mixed Integer Programming

- Premise: MIP has proven to be a powerful methodolgy for solving design and strategic problems; less so for real-time operational problems
- Can we use all the computational power at our disposal to turn MIP into a technology that can provide decision support in real time?
  - Optimization: either the true problem or a "pre-computing" stage
  - Re-optimization: sensitivity analysis, warm starts, etc.

A Great Unsolved Problem

Until recently, most applications of integer programming have been to planning models where solution time is not an issue. Significant improvements in methodology, high-speed computing and data availability have made it possible to apply integer programming at the operational level for instances of modest size, where solution time may take minutes…The next challenge is real-time mixed-integer programming (RTMIP). While such problems are prevalent in numerous application areas, the technology available for their solution is still at the research level…We believe that this pioneering use of cyberinfrastructure will open up new possibilities for the operations research community to exploit the computational resources, data storage capabilities and communication bandwidth that are now available for use in real-time decision-making.

-George Nemhauser, "Need and Potential for Real-Time Mixed-Integer Programming", Great Unsolved Problems in OR feature, *ORMS Today*, February 2007.

http://www.lionhrtpub.com/orms/orms-2-07/frinside.html

Parallel Computing: Massively Parallel Computers

- many, many dedicated processors
- very centralized: if one processor crashes, the whole system may be affected
- emphasis on defining subproblems ***quickly*** *(*ramp-up process); otherwise, many dedicated processors are doing nothing at the beginning
- strong emphasis on load balancing (otherwise, many dedicated processors are doing nothing at the end)
- little emphasis on reducing the amount of information passed…but little is not 0!
- This is framework that we will focus on in this talk…but we will keep the other one in mind.

Parallel Computing: Grid Computing

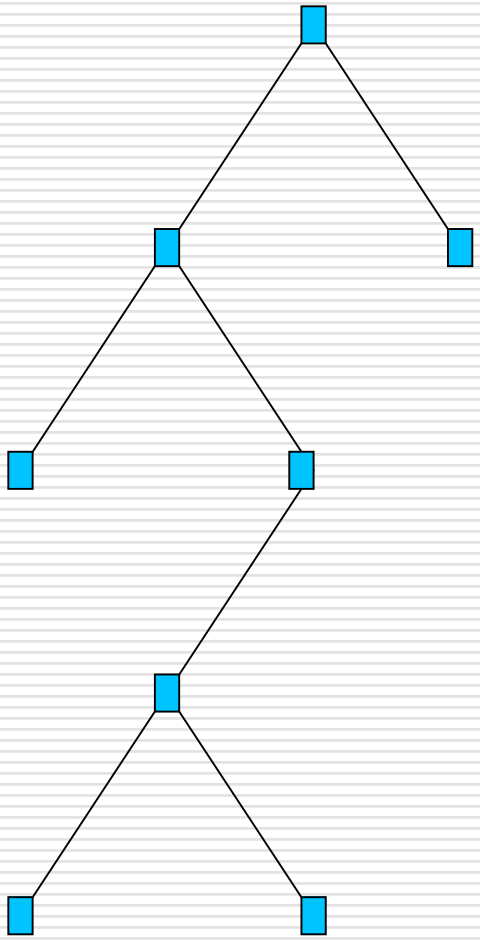
- many spare processors
- very decentralized: if one processor crashes, the work will be re-started, and the rest keep going without noticing
- large emphasis on reducing the amount of information passed
- significant emphasis on load balancing (defining work for many processors)
- less emphasis on efficient ramp-up

The current state of the art

- The most robust methods for solving general MIPs are LP-based branch-and-bound and branch-and-cut methods.
- A number of researchers have investigated parallelizing these methods (Linderoth, Perumalla, Savelsbergh [1997]; Ralphs (2002); Ferris, Pataki, Schmieta (2003); Eckstein (2003))
- The best commercial solvers can use up to 32 processors in their branch-and-cut codes

The current state of the art

- Considerable speedup (though not close to linear) can often be obtained through the right search strategies:
  - passing only "long, skinny" subtrees
  - sophisticated subtree allocation to processors based on regular checkpointing
- However, this approach has evident performance bottlenecks:
  - Generating enough "interesting" subtrees (not too large, not too trivial)
  - Passing all this information
- Hence the lack of implementation for more than ~32 processors



Research Issues

If we want to be able to use hundreds or thousands of processors to solve MIPs, we need to re-think the framework that we use. In particular, we need to address at least the following questions.

Question: What should each processor do? How can we effectively use many at once?

Answer: *We need to partition the problem into many non-overlapping, tractable, nontrivial sub-problems very quickly (so that each can be assigned to a different processor).*

Question: How can we define these subproblems? (We have seen that we need alternatives to single variable branching.)

Answer: *We use LP-and-FIX, RINS, Local Branching, and Solution Crossing cuts.*

Primal Heuristics

Two main classes:

- ***Construction heuristics***: These produce a feasible solution from scratch (Example: LP-and-FIX).
- ***Improvement heuristics***: These try to improve a given feasible solution (Example: RINS, Local Branching, and Solution Crossing).

LP-and-FIX

IDEA: Explore a sub-space defined by the current LP relaxation solution.

HOW: Fix the variables with integral value in LP relaxation solution $(\hat{x}, \hat{y})$ and solve the resulting problem:

$$\begin{aligned} \min_{(x,y)\in \mathbb{R}^n \times \mathbb{R}^p} \quad & cx + fy \\ \text{s.t.} \quad & Ax + Gy \geq b \\ & x \geq 0, y \in \{0,1\}^p \\ & y_j = \hat{y}_j \text{ for all } j \in Q \text{ with } \hat{y}_j \in \{0,1\} \end{aligned}$$

RINS

(Relaxation Induced Neighborhood Search)
[E. Danna, E. Rothberg, and C. Le Pape 2005]

IDEA: Explore the sub-space defined by the intersection of the LP relaxation solution $(\hat{x}, \hat{y})$ and an MIP feasible solution $(\bar{x}, \bar{y})$.

HOW: If a binary variable has the same value in both solutions, fix the its value:

$$\begin{aligned} \min_{(x,y)\in \mathbb{R}^n \times \mathbb{R}^p} \quad & cx + fy \\ \text{s.t.} \quad & Ax + Gy \geq b \\ & x \geq 0, y \in \{0,1\}^p \\ & y_j = \bar{y}_j \text{ for all } j \in Q \text{ with } \bar{y}_j = \hat{y}_j \end{aligned}$$

Local Branching

IDEA: The same as RINS (Explore the neighborhood around an MIP feasible solution $(\bar{x}, \bar{y})$) [Fischetti and Lodi 2003]

HOW: The neighborhood consists of y vectors that do not differ from $\bar{y}$ in more than k indices

$$\begin{aligned} \min_{(x,y)\in \mathbb{R}^n \times \mathbb{R}^p} \quad & cx + fy \\ \text{s.t.} \quad & Ax + Gy \geq b \\ & x \geq 0, y \in \{0,1\}^p \\ & \sum_{j\in Q:\bar{y}_j=0} y_j + \sum_{j\in Q:\bar{y}_j=1} (1 - y_j) \leq k \end{aligned}$$

Note that this strategy is "orthogonal" to that defined by RINS.

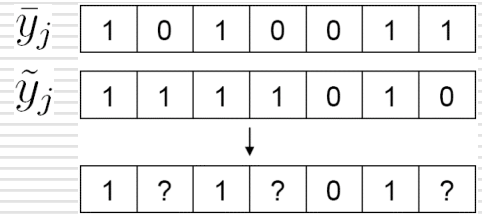
Solution Crossing

IDEA: Using concepts of Evolutionary Methods in improving the existing feasible solutions ([E. Rothberg 2007])

HOW:

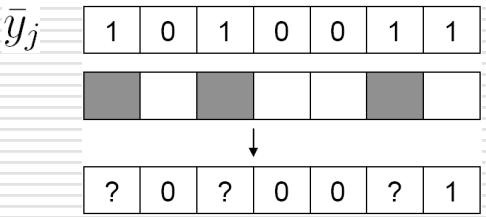
- Population: A set of feasible solutions
- Combination: (Similar to RINS)

$$\begin{aligned} \min_{(x,y)\in \mathbb{R}^n \times \mathbb{R}^p} \quad & cx + fy \\ \text{s.t.} \quad & Ax + Gy \geq b \\ & x \geq 0, y \in \{0,1\}^p \\ & y_j = \bar{y}_j \text{ for all } j \in Q \text{ with } \bar{y}_j = \tilde{y}_j \end{aligned}$$



- Mutation:

$$\begin{aligned} \min_{(x,y)\in \mathbb{R}^n \times \mathbb{R}^p} \quad & cx + fy \\ \text{s.t.} \quad & Ax + Gy \geq b \\ & x \geq 0, y \in \{0,1\}^p \\ & y_j = \bar{y}_j \text{ for all } j \in \bar{Q} \subset Q, \text{ for some } \bar{Q} \end{aligned}$$



A Parallel Macro Partitioning Framework (PMaP)

- Brancher: Generates sub-problems (single processor)
- Workers: Solve sub-problems (Many processors)
- Assigner: Assigns the generated sub-problems by Brancher to Slaves (single processor)

Brancher

- Starts solving the main problem using Branch and Bound
- At each node of Branch and Bound tree if there exist any feasible solution, for each one generates a RINS problem, puts this problem in the sub-problem pool, and adds the following cut to the problem it is solving:

$$\sum_{j \in S^0} y_j + \sum_{j \in S^1} (1 - y_j) \geq 1$$

The complement of the RINS cut

$$\sum_{j \in S^0} y_j + \sum_{j \in S^1} (1 - y_j) = 0$$

S^0: Set of variables with value 0 in the feasible solution
S^1: Set of variables with value 1 in the feasible solution

- At each node of Branch and Bound tree if there is no feasible solution, generates a LP-and-FIX problem, puts it in the sub-problem pool, and adds the following cut to the problem it is solving:

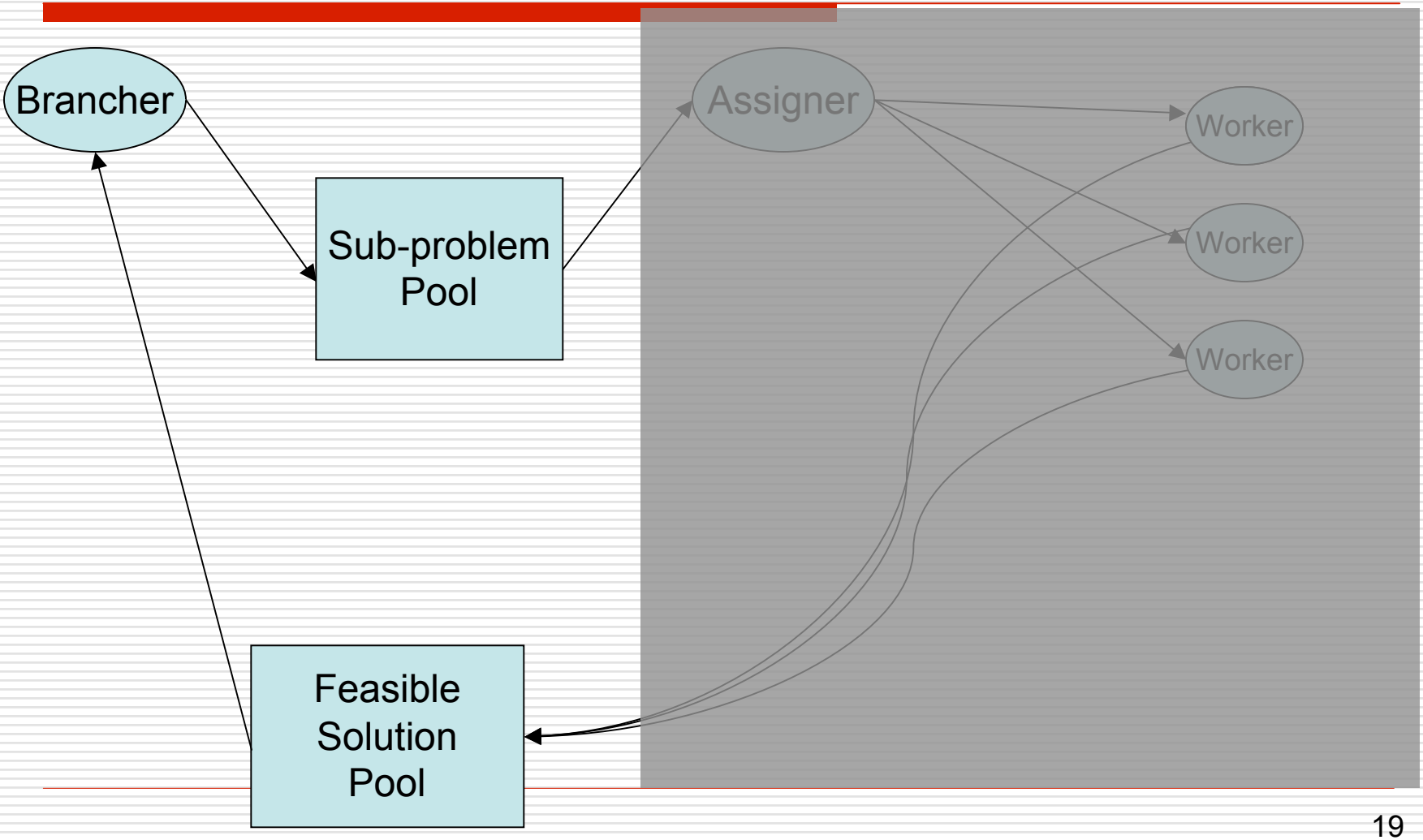
$$\sum_{j \in S^0} y_j + \sum_{j \in S^1} (1 - y_j) \geq 1$$

The complement of the LP-and-FIX cut

$$\sum_{j \in S^0} y_j + \sum_{j \in S^1} (1 - y_j) = 0$$

S^0: Set of variables with value 0 in the LP relaxation solution
S^1: Set of variables with value 1 in the LP relaxation solution

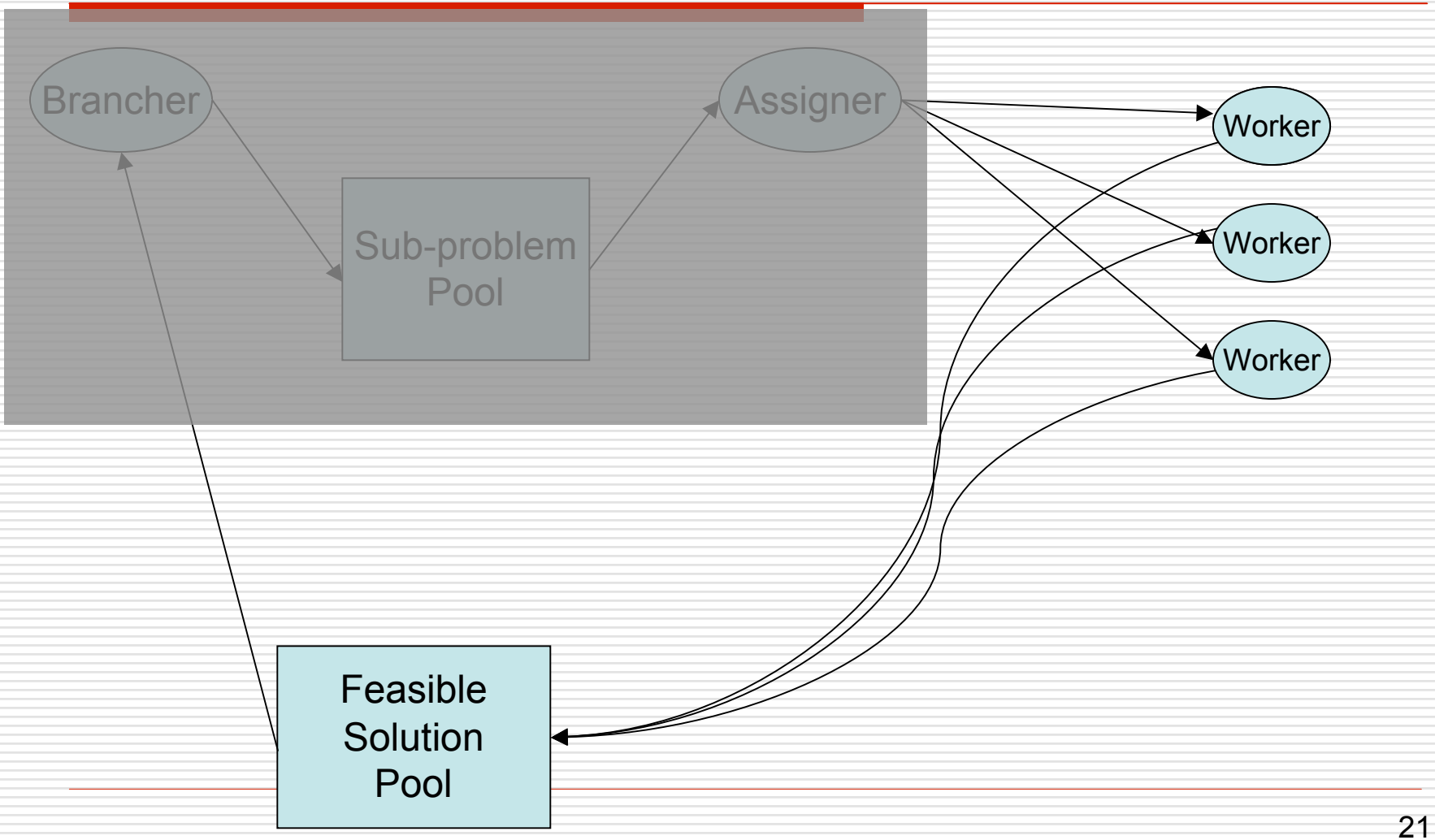
Brancher


Brancher
Sub-problem Pool
Assigner
Worker
Worker
Worker
Feasible Solution Pool

Worker Processor

- Waits until one sub-problem is assigned to it
- Starts solving the sub-problem using Branch and Bound
- Whenever finds a feasible solution, writes that into the Feasible Solution Pool
- When the solution process is over, sends a message to assigner
- Waits until the next sub-problem is assigned to it and does the same procedure

Worker

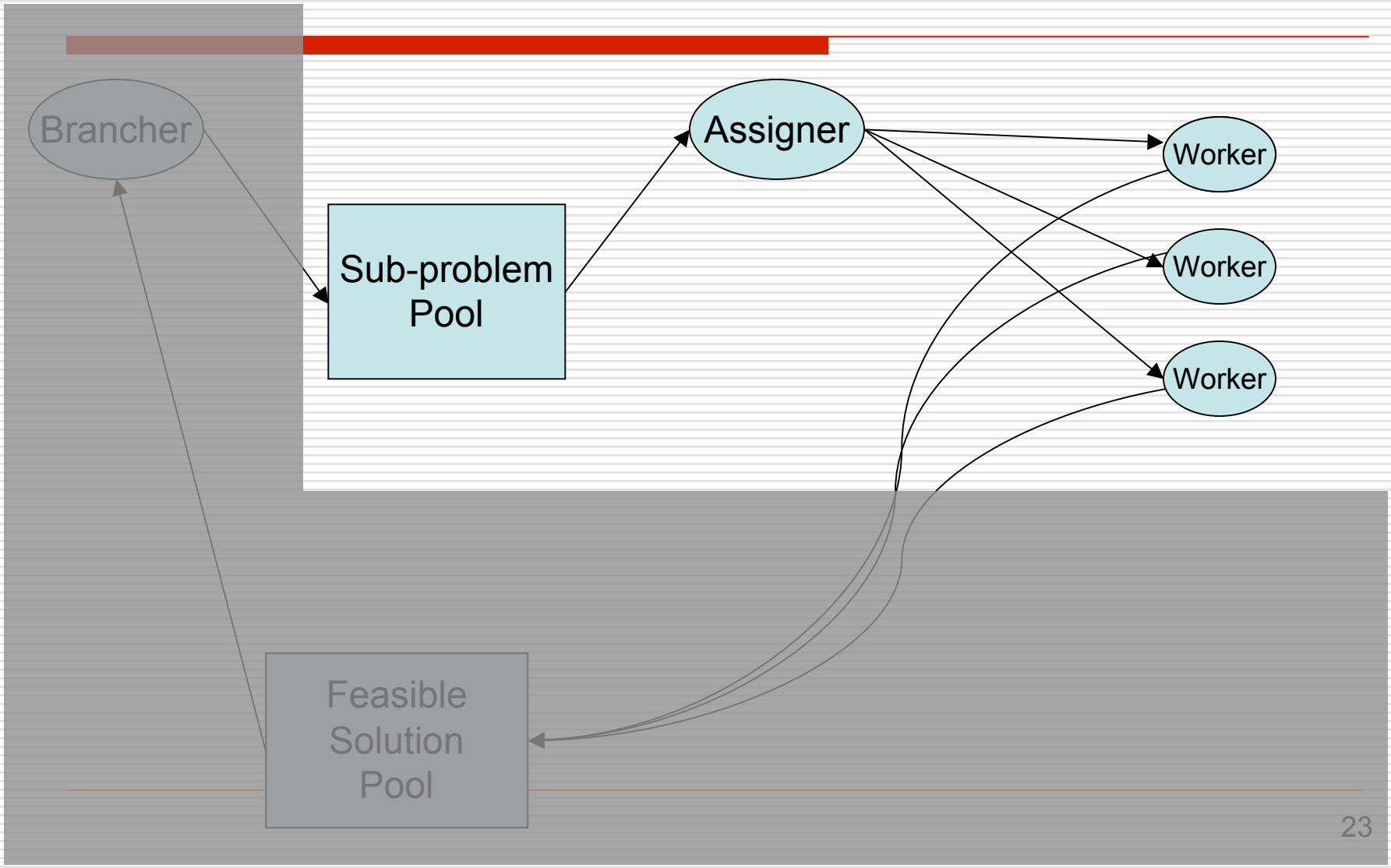

Brancher
Assigner
Sub-problem Pool
Worker
Worker
Worker
Feasible Solution Pool

Assigner

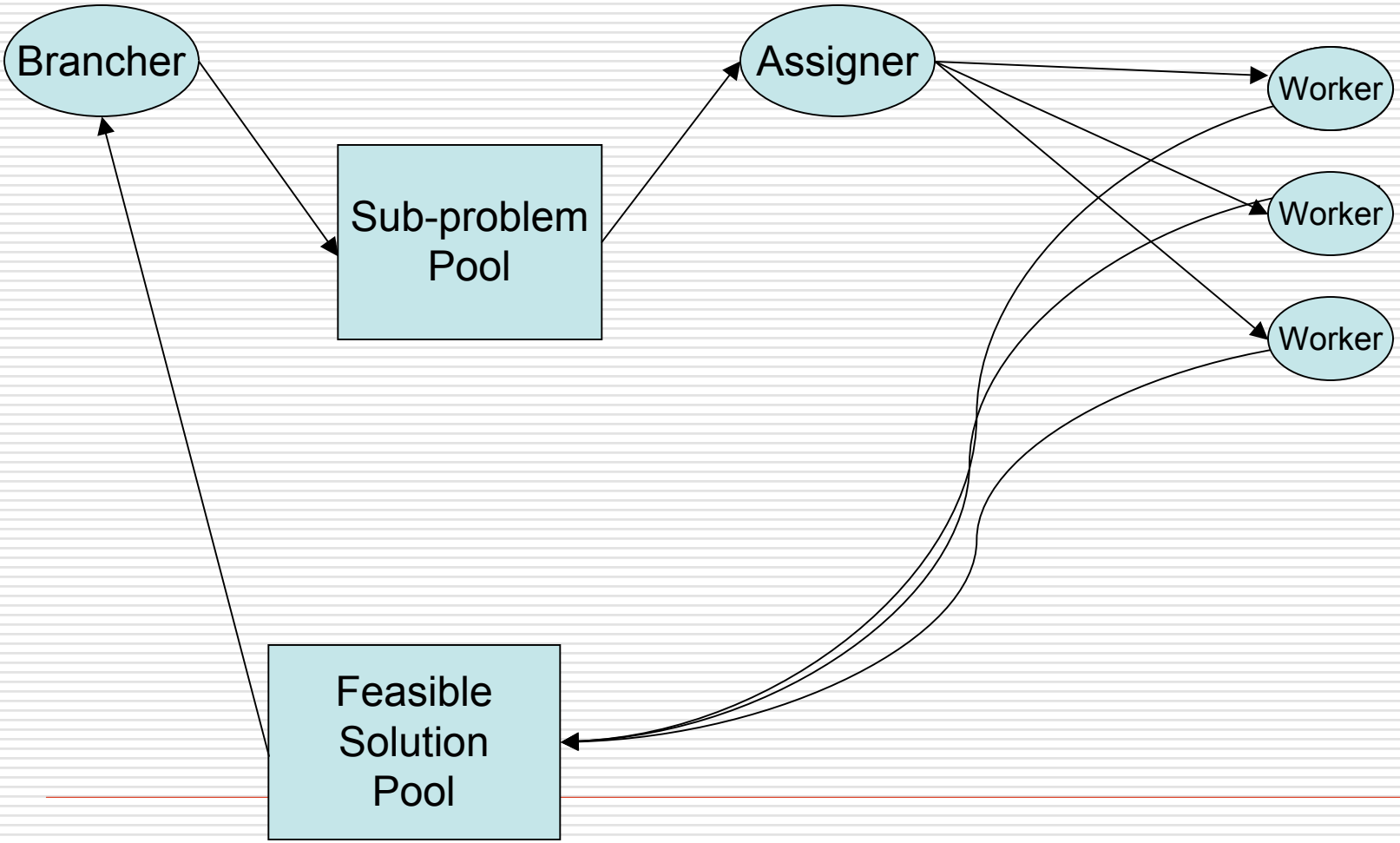
While the program is running

- Checks the sub-problem pool.
- If there exists one or more sub-problems in the sub-problem pool, gets one of them; Otherwise waits until one appears.
- Checks the status of worker processors.
- If there is any idle worker processor, assigns the problem in hand to that; otherwise waits until one becomes free and then assigns the problem.
- Updates the status of worker processors

Assigner


Brancher
Assigner
Worker
Worker
Worker
Sub-problem Pool
Feasible Solution Pool

The Framework



Some Notes!

- Implemented using Cbc-COIN
- Communication between processors is done through two channels:
  - MPI (Message Passing Interface)
  - Text Files
- We run the program on the Datastar machine in the San Diego Supercomputer Center (SDSC)
  - DataStar consists of nodes of two types: 8-way p655+ and 32-way p690+ nodes
  - For test runs we usually use p690+ nodes with 128GB memory per node and 1.7GHz CPU's
- PMaP is at a preliminary stage. There is still a long way to go!

Results

Problem	COIN-Cbc	PMaP (using Cbc)	Optimal Solution
aflow40b	1274	1168	1168
seymour	435	425	423
harp2	-7.29769e+7	-7.38998e+7	-7.38998e+7
markshare1	14	4	1
markshare2	33	16	1
mas74	12886	11801.2	11801.2
mas76	40935.1	40005.05	40005.05
nsrand-ipx	54560	54080	54080

- The experiments ran for 30 minutes.
- We used 35 processors for each run of PMaP.

Newer Results

Problem	COIN-Cbc	PMaP (using Cbc)	Optimal Solution
dano3mip	791.385	719.782	?
protfold	-	-21	-31
sp97ar	-	685860294.1	?
glass4	2.20002e+9	1.90002e+9	1.20001e+9

- Again each problem ran for 30 minutes, and PMaP used 35 processors.

The Most Recent Results

Problem	CPLEX	PMaP (with Cbc)	Optimal Solution
glass4	1.60001e+09	1.60001e+09	1.20001e+09
markshare1	7	4	1
markshare2	25	16	1
protfold	-20	-21	-31
atlanta-ip	-	95.0098	90.0099
sp97ar	6.62541e+08	6.8753e+08	?
seymour	425	425	423
Danoint	65.6667	65.6667	65.6667
Dano3mip	698.6296	719.782	?
A1c1s1	11790.1684	12054.8	11503.4
swath	730.1	577	467.407
liu	1236	1870	?

- We compare the results with parallel CPLEX 10 installed on Datastar.
- Each problem ran for 30 minutes.
- We used 32 processors for each run of PMaP and CPLEX.
- CPLEX can be run on the limited number of processors which share the same memory (on Datastar at most 32), but PMaP can be run on as many processors as the machine has.

Preliminary Conclusions

- PMaP is capable of using many different processors to considerable advantage (it improve both COIN-CBC and MINTO enormously).
- PMaP is already competitive with the best commercial solvers on the most powerful parallel frameworks that these solvers can use.

To-Do List: Ongoing

- Explore re-optimization possibilities (we are excited about the potential here).

To-Do List: Immediate

- Extract lower bounds from PMaP.
- Test PMaP on lots of processors (>1000).
- Implement local branching cuts.
- Implement solution crossing cuts.
- Compare the performance of PMaP with other parallel solvers.

To-Do List: Immediately After That

- Incorporate multiple brancher capability.
- Fine-tune the number of variables fixed, and other parameterers) to perform better dynamic load balancing.
- Optimize pre-processing at worker nodes.