Introduction to Parallel & Distributed Computing

Discrete Search & Load Balancing

Lecture 14, Spring 2014

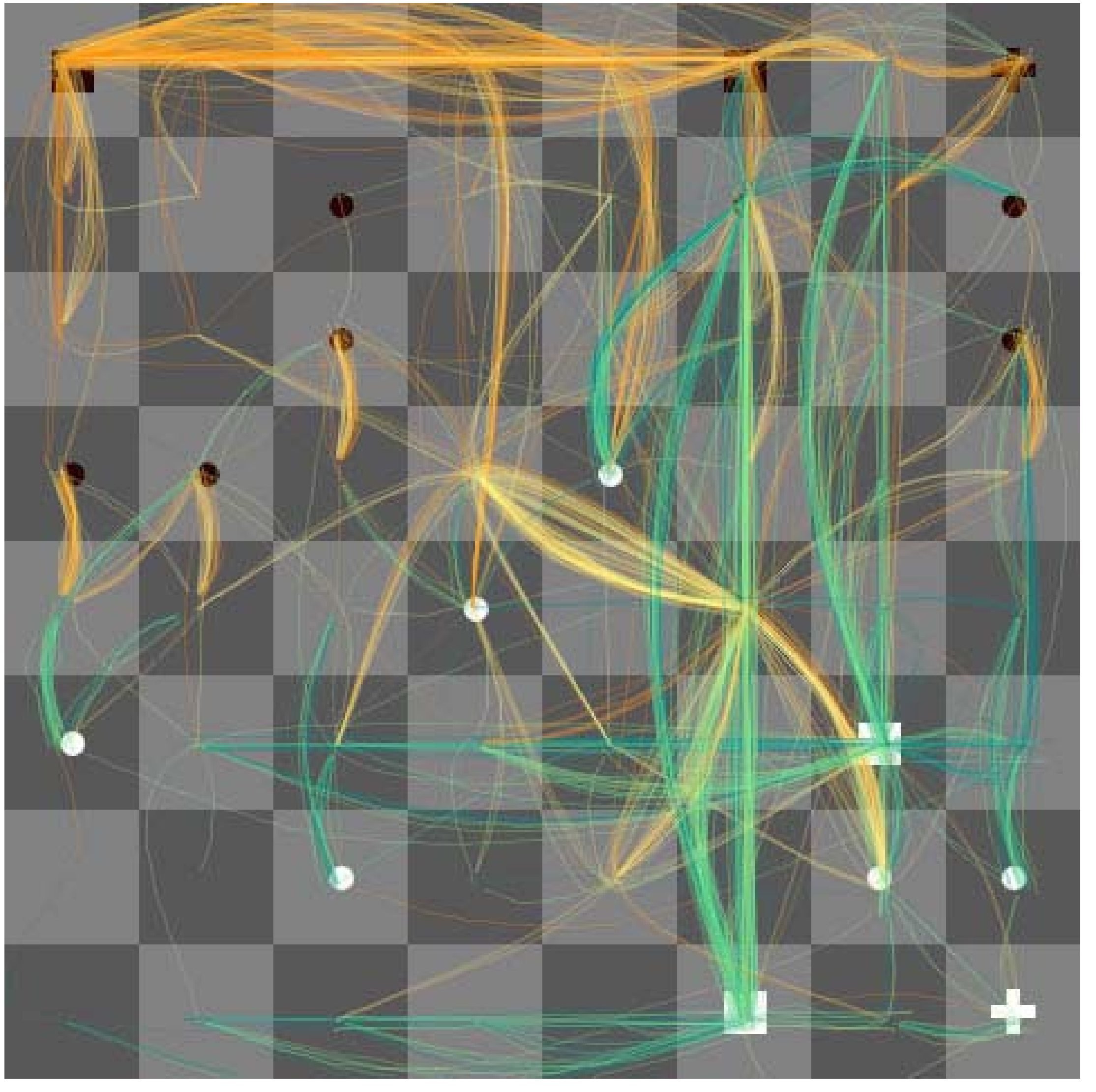
Instructor: 罗国杰

gluo@pku.edu.cn

In this Lecture …

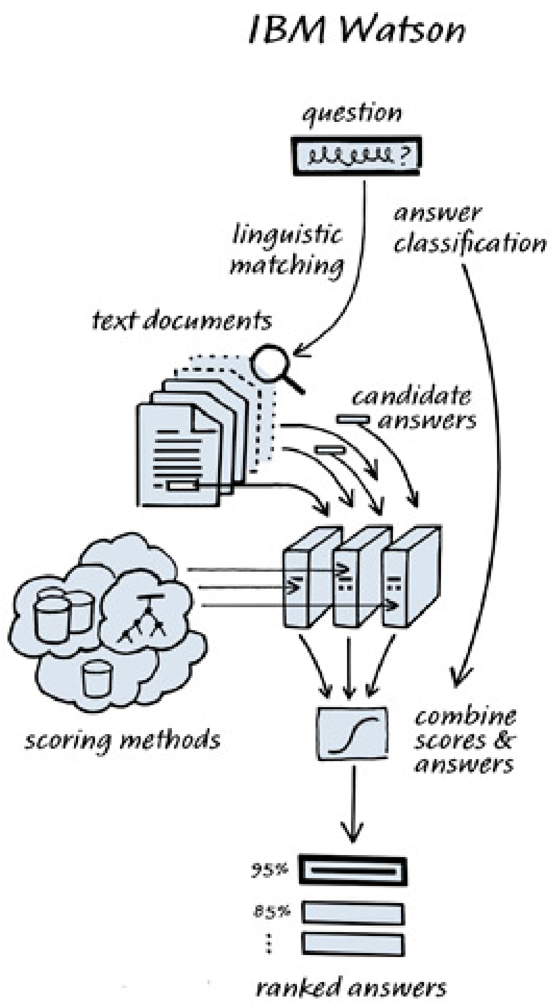
- **Parallelization & load balancing schemes**
  - **in depth-first search**
  - **in best-first search**
- **Speedup anomalies**

Example of Discrete Search



http://vizclass.csc.ncsu.edu/2011/08/viz-thinking-machine-4-chess.html

IBM Watson



http://blog.stephenwolfram.com/2011/01/jeopardy-ibm-and-wolframalpha/

Depth-First Search

- **Uses depth-first search to consider alternative solutions to a combinatorial search problem**
- **Recursive algorithm**
- **Backtrack occurs when**
  - **A node has no children (“dead end”)**
  - **All of a node’s children have been explored**

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Example: Crossword Puzzle Creation

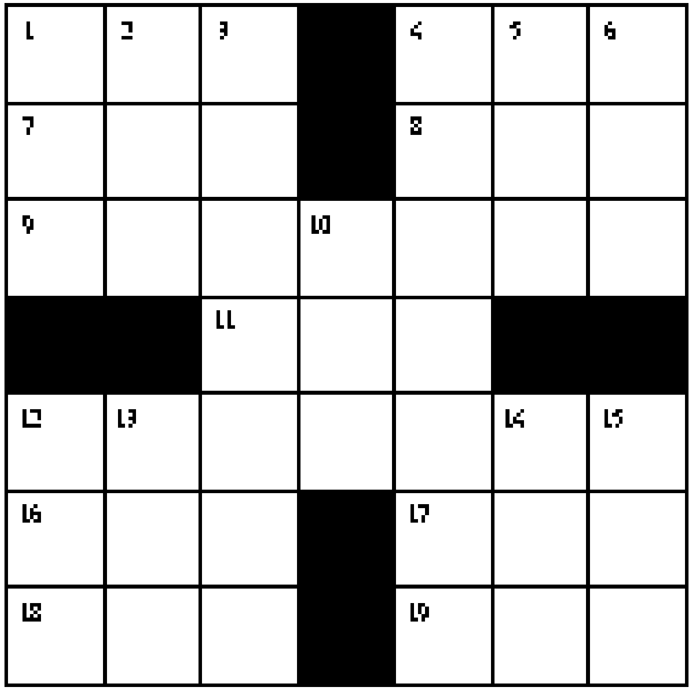
- **Given**
  - **Blank crossword puzzle**
  - **Dictionary of words and phrases**
- **Assign letters to blank spaces so that all puzzle’s horizontal and vertical “words” are from the dictionary**
- **Halt as soon as a solution is found**

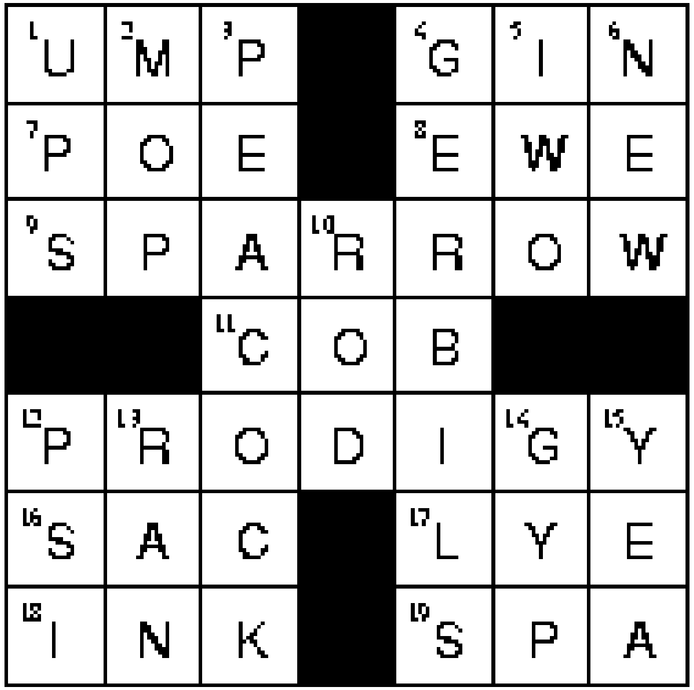
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Crossword Puzzle Problem

Given a blank crossword puzzle and a dictionary

find a way to fill in the puzzle.





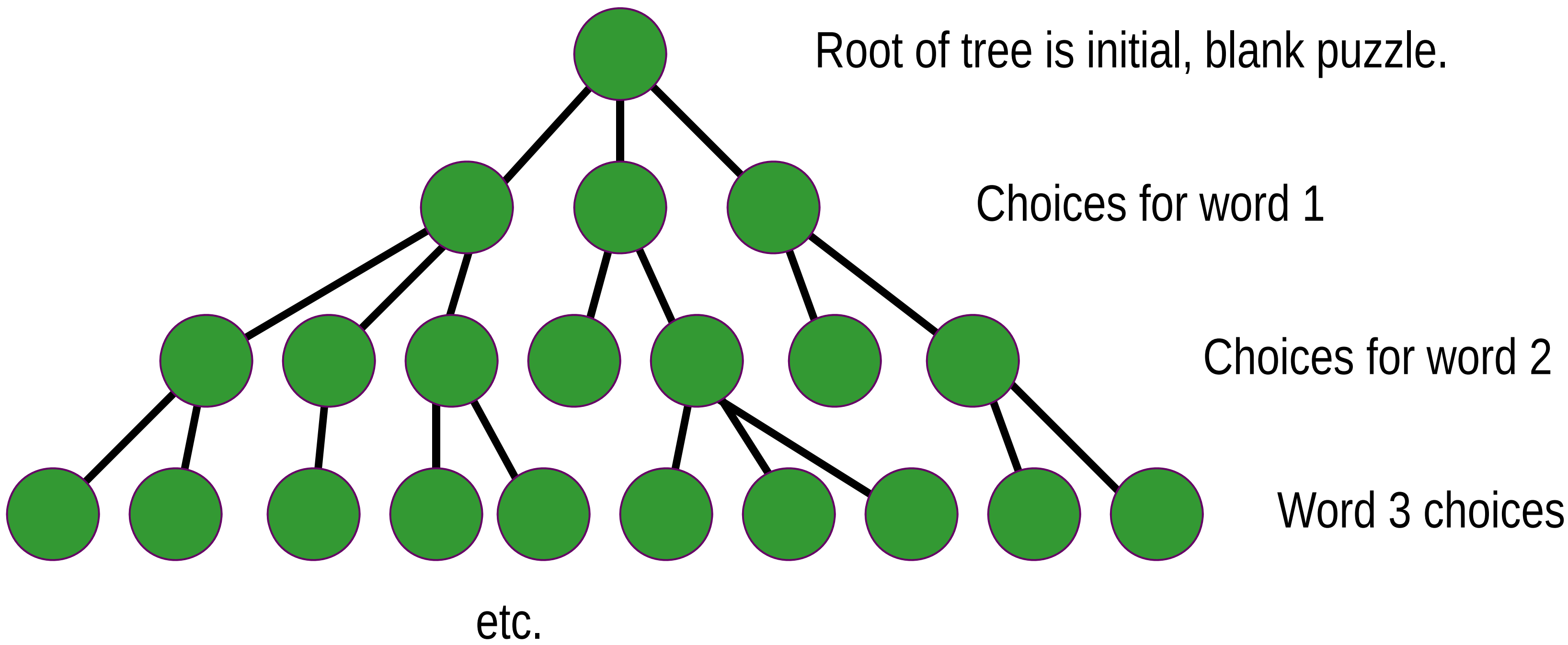
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

A Search Strategy

- **Identify longest incomplete word in puzzle (break ties arbitrarily)**
- **Look for a word of that length**
- **If cannot find such a word, backtrack; otherwise,**
  - **Find longest incomplete word that has at least one letter assigned (break ties arbitrarily)**
  - **Look for a word of that length**
  - **If cannot find such a word, backtrack**
- **Recurs until a solution is found or all possibilities have been attempted**

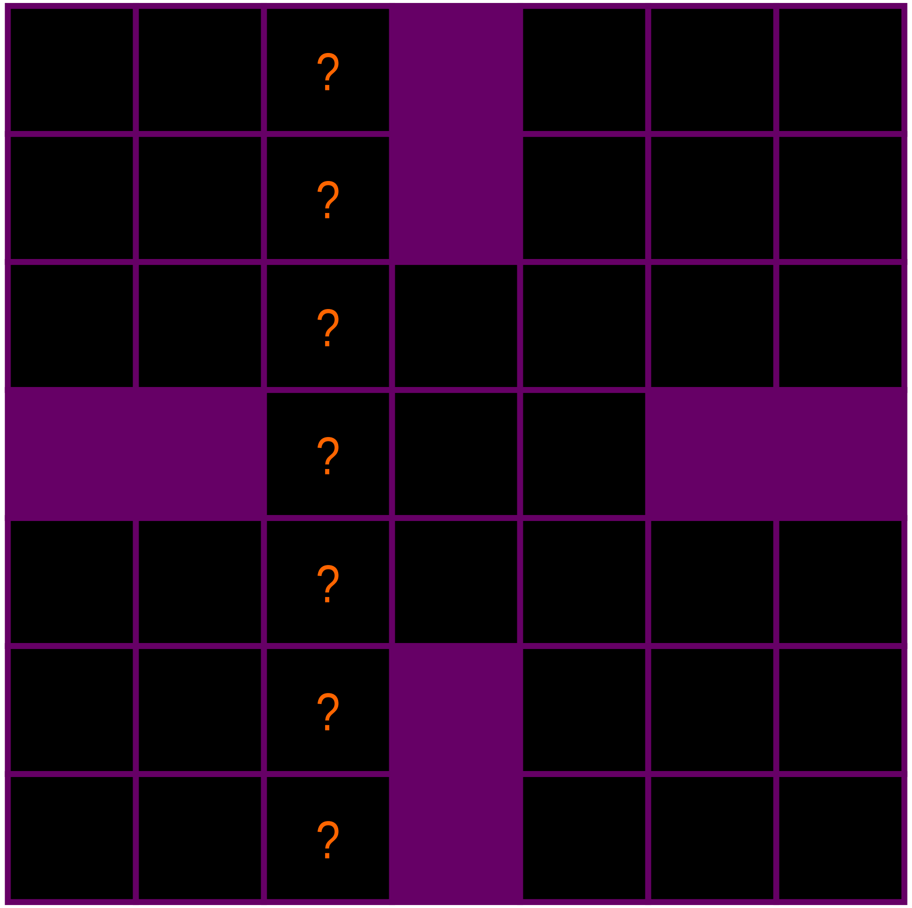
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

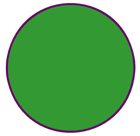
State Space Tree



Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

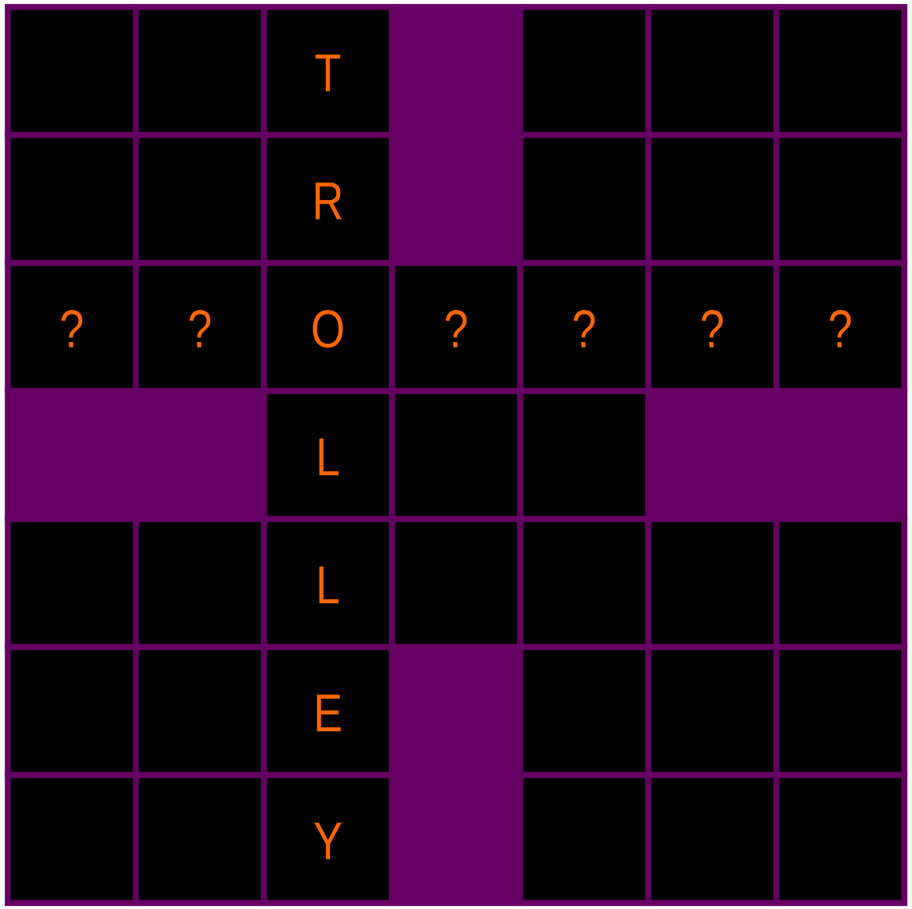
Depth-First Search

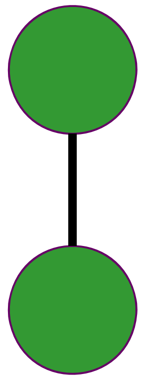




Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

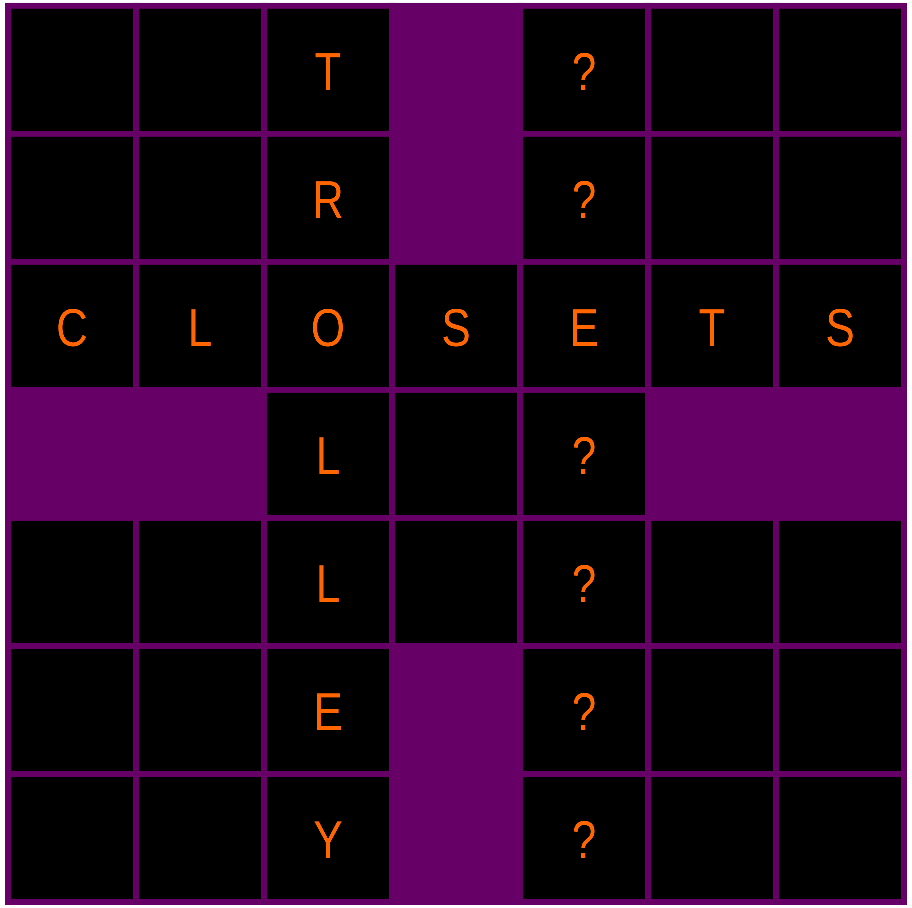
Depth-First Search

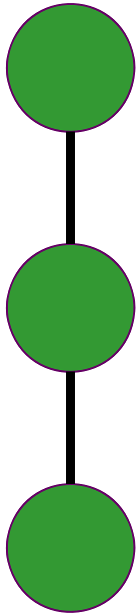




Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

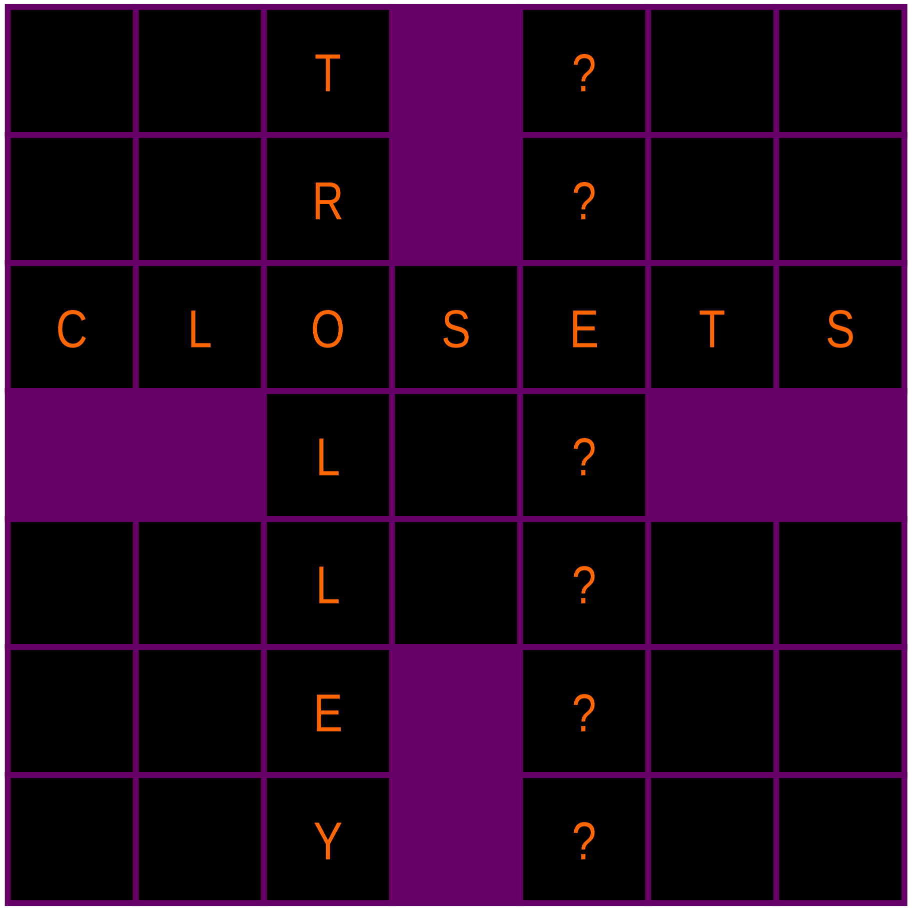
Depth-First Search

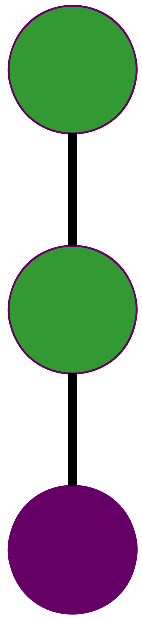




Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Depth-First Search



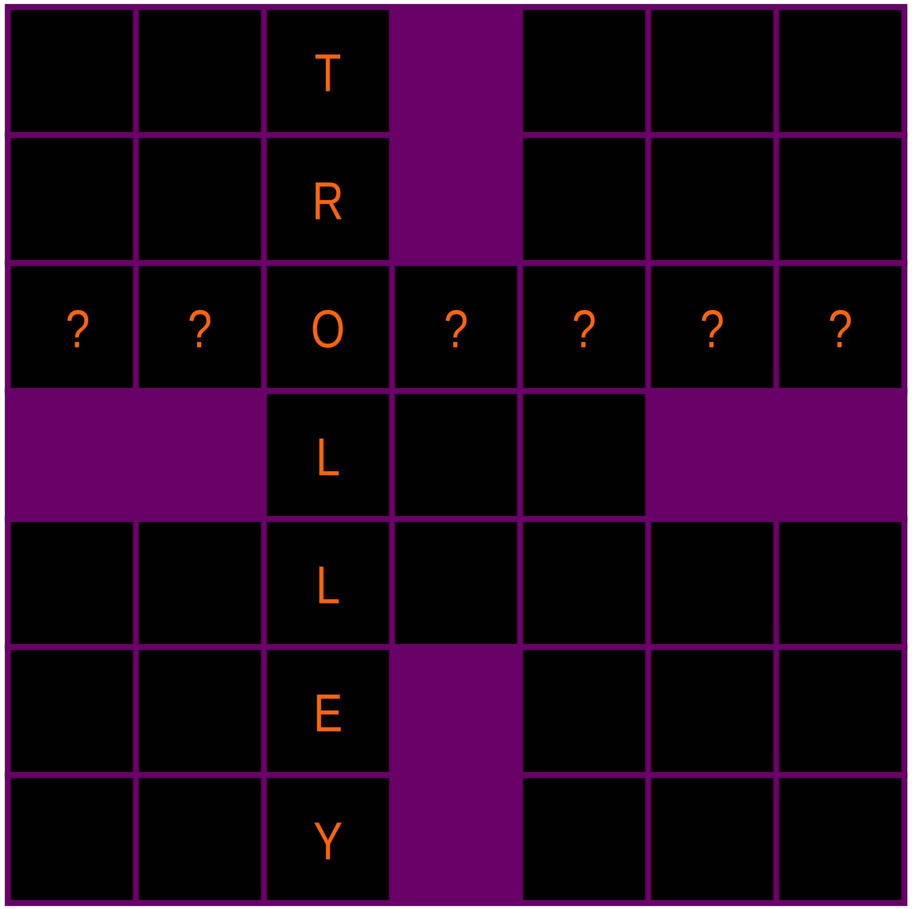


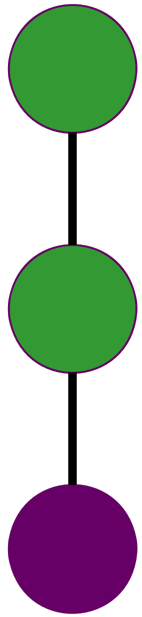
Cannot find word.

Must backtrack.

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Depth-First Search



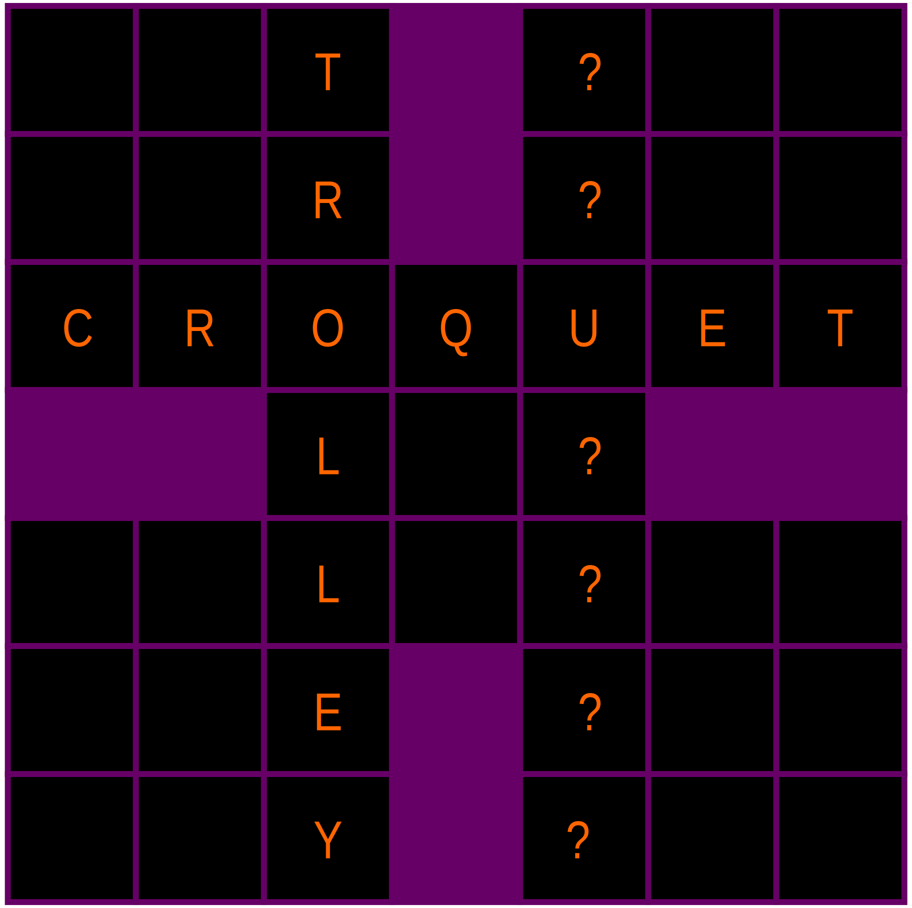


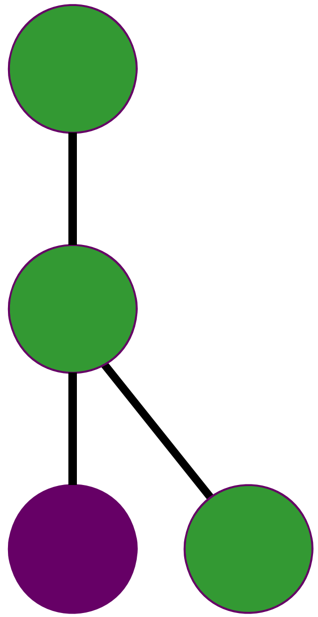
Cannot find word.

Must backtrack.

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

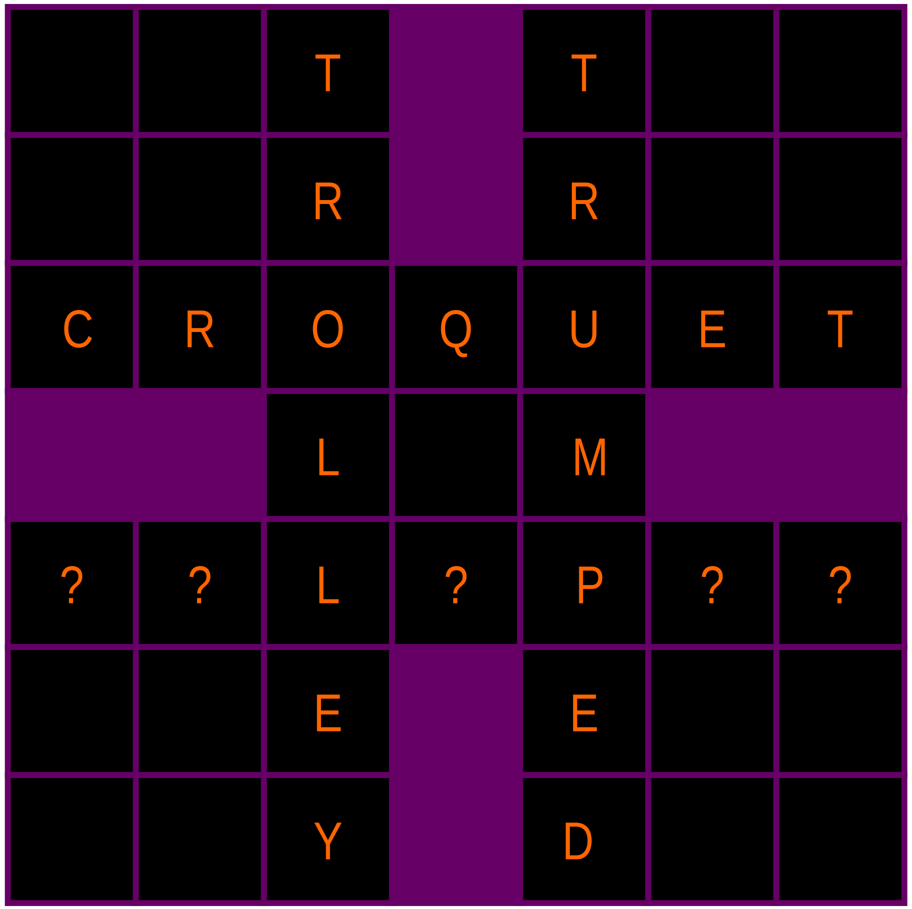
Depth-First Search

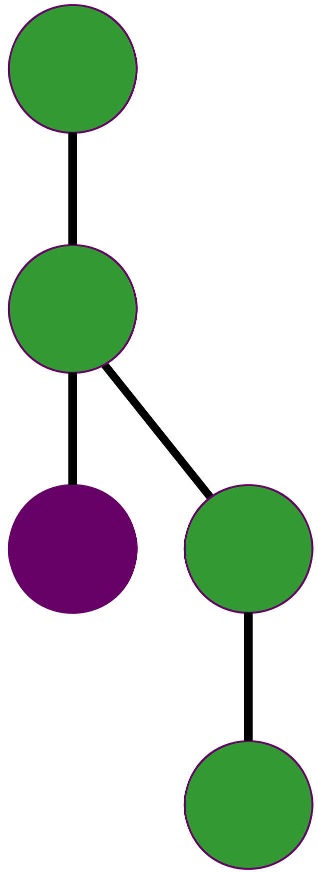




Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Depth-First Search





Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Time and Space Complexity

- **Suppose average branching factor in state space tree is b**
  - **Branching factor: the (average) number of children at each node**
- **Searching a tree of depth k requires examining**

$$1+b+b^2+\cdots+b^k=\frac{b^{k+1}-b}{b-1}+1=\theta(b^k)$$

  nodes in the worst case (exponential time)
- **Amount of memory required is $\Theta(k)$**

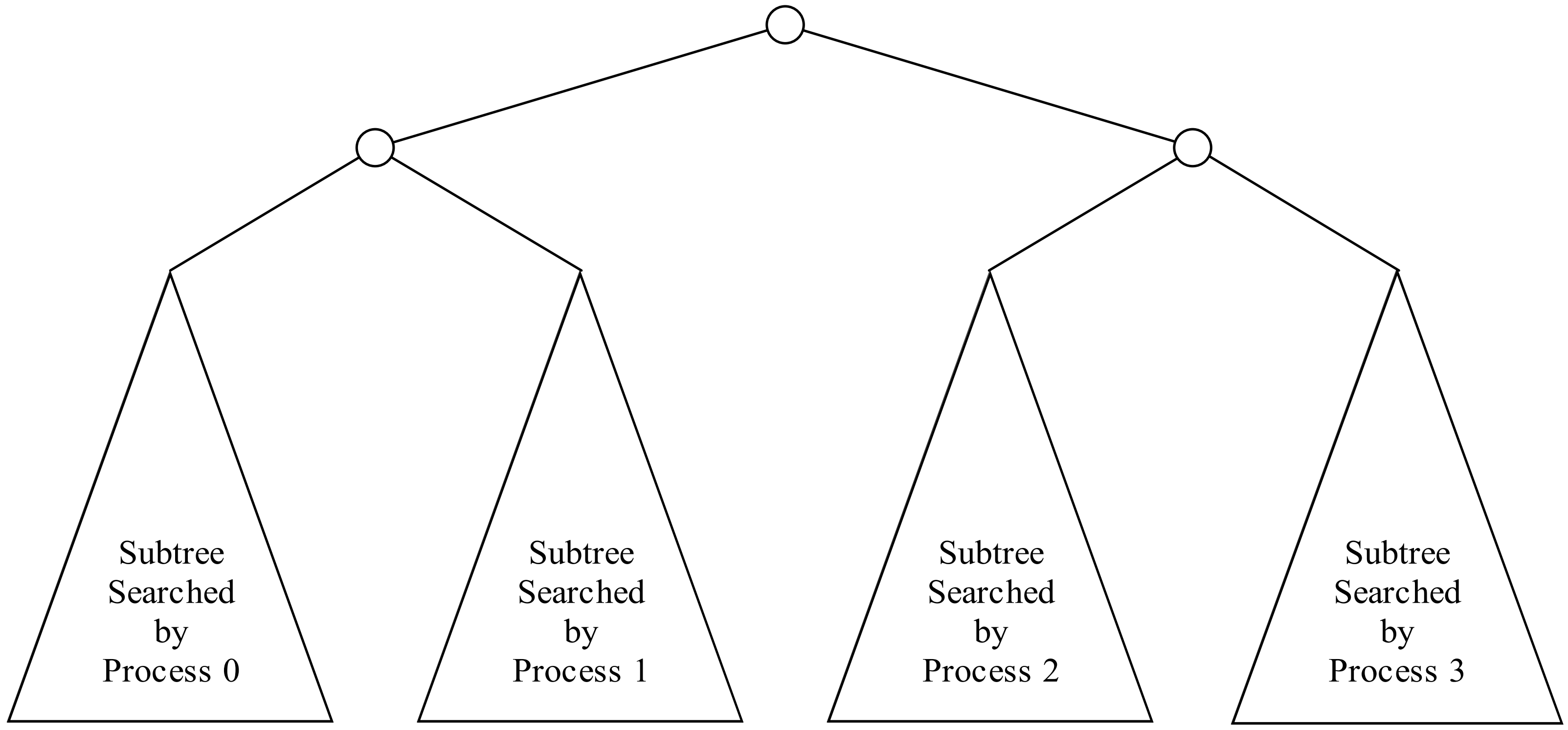
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Parallel Depth-First Search

- **First strategy: give each processor a subtree**
- **Suppose** $p = b^k$
  - **A process searches all nodes to depth** k
  - **It then explores only one of subtrees rooted at level** k
  - **If** d **(depth of search) >** $2k$**, time required by each process to traverse first** k **levels of state space tree inconsequential**

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Parallel Backtrack when $p = b^k$



Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

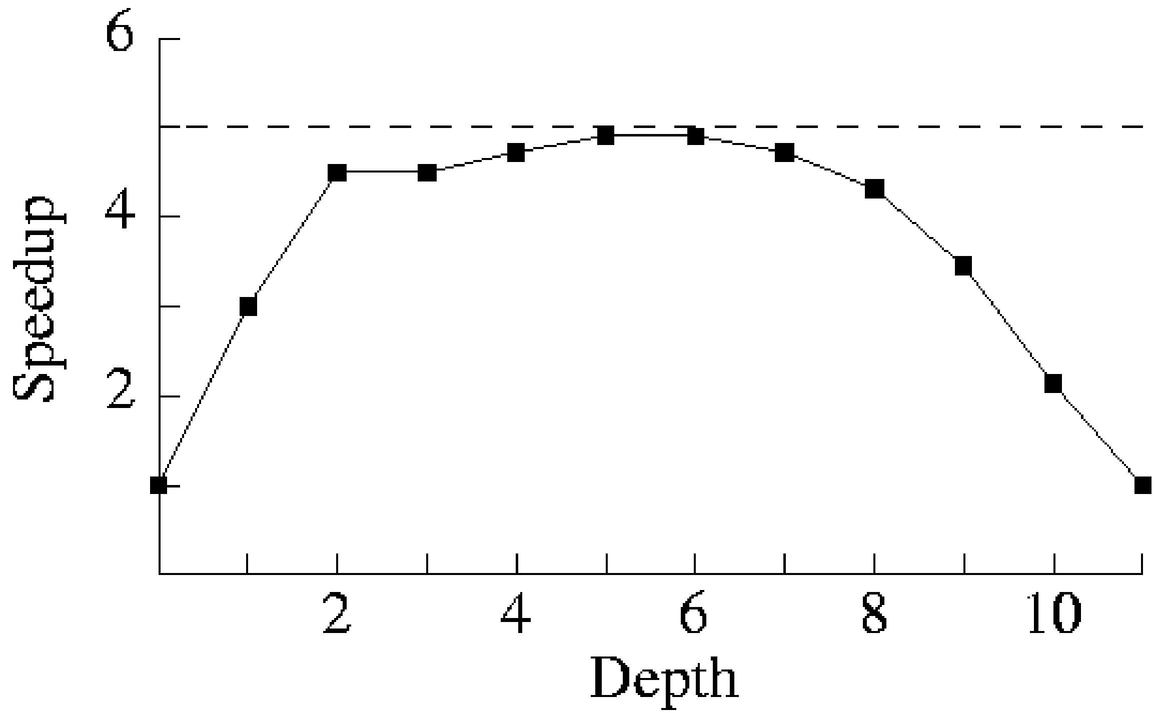
What If $p \neq b^k$?

- **A process can perform sequential search to level *m* of state space tree**
- **Each process explores its share of the subtrees rooted by nodes at level *m***
  - **As *m* increases, there are more subtrees to divide among processes, which can make workloads more balanced**
  - **Increasing *m* also increases number of redundant computations**

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Maximum Speedup when $p \neq b^k$

In this example 5 processors are exploring a state space tree with branching factor 3 and depth 10.



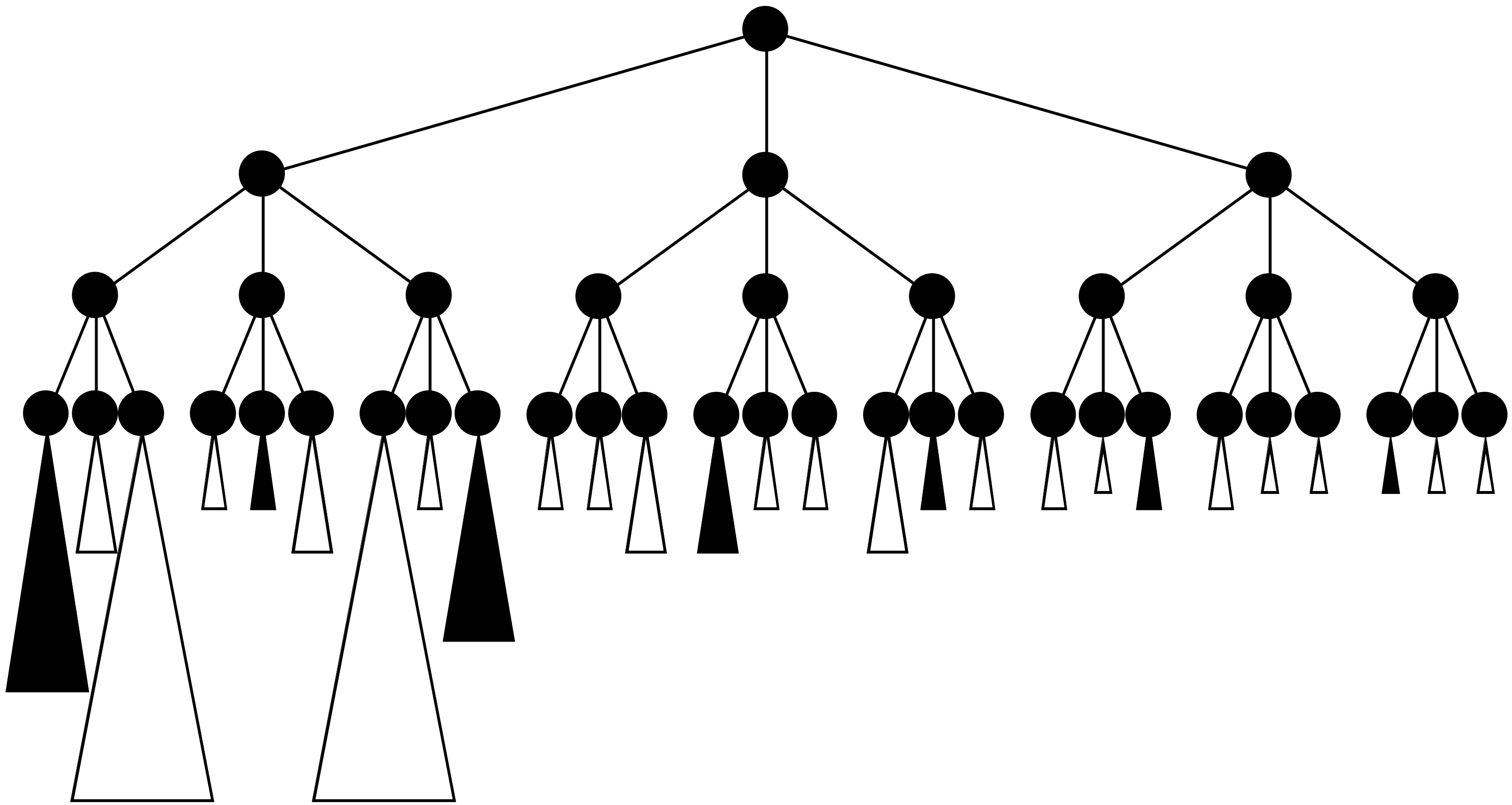
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Disadvantage of Allocating One Subtree per Process

- **In most cases state space tree is not balanced**
- **Example: in crossword puzzle problem, some word choices lead to dead ends quicker than others**
- **Alternative: make sequential search go deeper, so that each process handles many subtrees (cyclic allocation)**

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Allocating Many Subtrees per Process



Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Parallel DFS: Motivation

- **Discrete optimizations are usually NP-hard problems. Does parallelism really help much?**
  - **For many problems, the average-case runtime is polynomial.**
  - **Often, we can find suboptimal solutions in polynomial time.**
  - **Many problems have smaller state spaces but require real-time solutions.**
  - **For some other problems, an improvement in objective function is highly desirable, irrespective of time.**

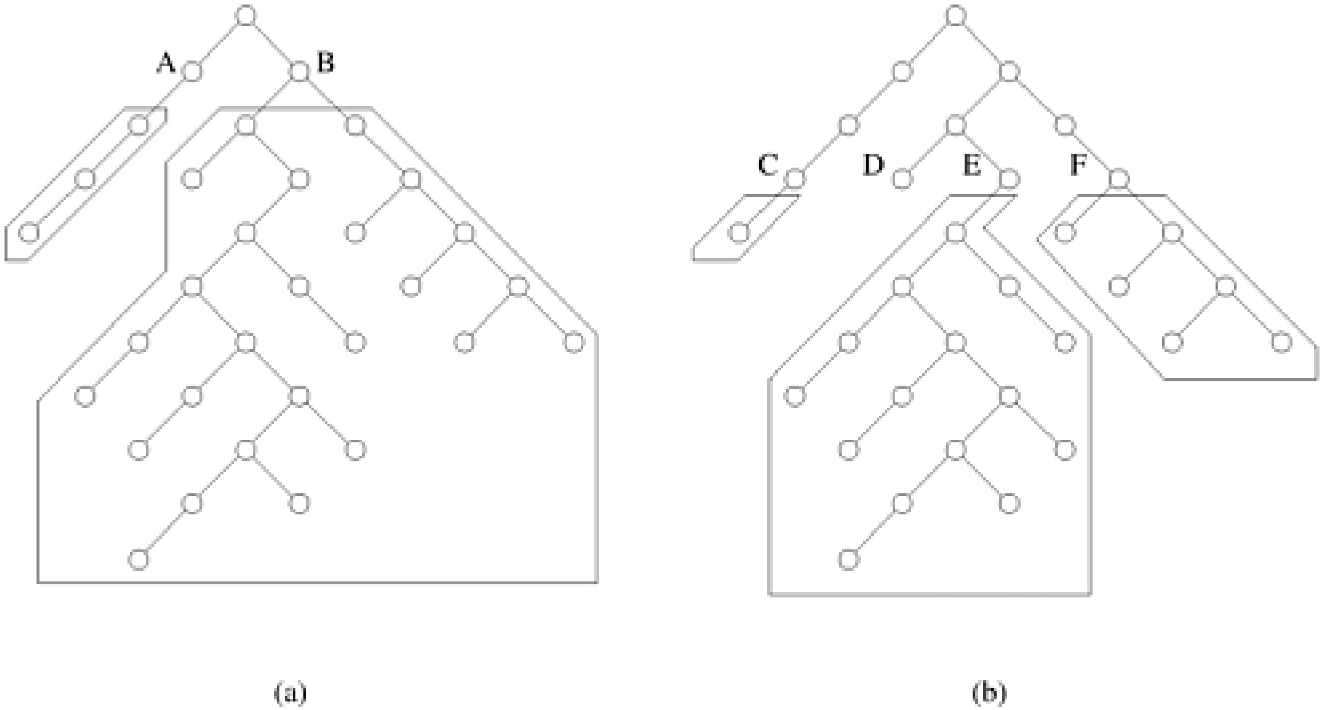
A. Grama et al., “Introduction to Parallel Computing,” Addison Wesley, 2003

Parallel Depth-First Search

- **How is the search space partitioned across processors?**
  - **Different subtrees can be searched concurrently.**
  - **However, subtrees can be very different in size.**
  - **It is difficult to estimate the size of a subtree rooted at a node.**
- **Dynamic load balancing is required.**

A. Grama et al., “Introduction to Parallel Computing,” Addison Wesley, 2003

Parallel Depth-First Search



The unstructured nature of tree search and the imbalance resulting from static partitioning.

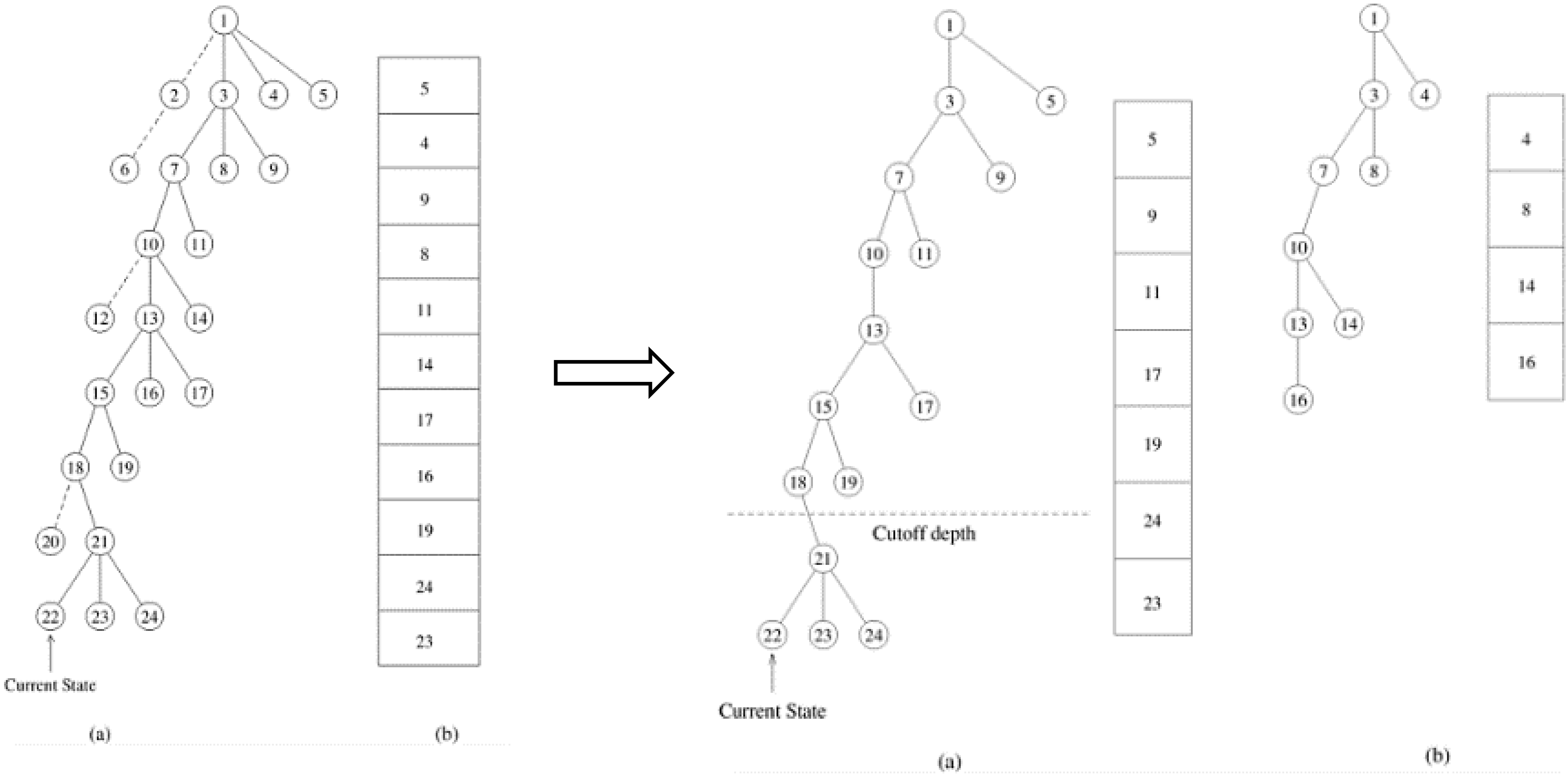
A. Grama et al., “Introduction to Parallel Computing,” Addison Wesley, 2003

Parameters in Parallel DFS: Work Splitting

- **Terminologies**
  - ***Donor*** **process: the process that sends work**
  - ***Recipient*** **process: the process that requests/receives work**
  - ***Half-split*****: ideally, the stack is split into two equal pieces such that the search space of each stack is the same**
  - ***Cutoff depth*****: to avoid sending very small amounts of work, nodes beyond a specified stack depth are not given away**

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Parameters in Parallel DFS: Work Splitting



Splitting the DFS tree:
the two subtrees along with their stack representations.

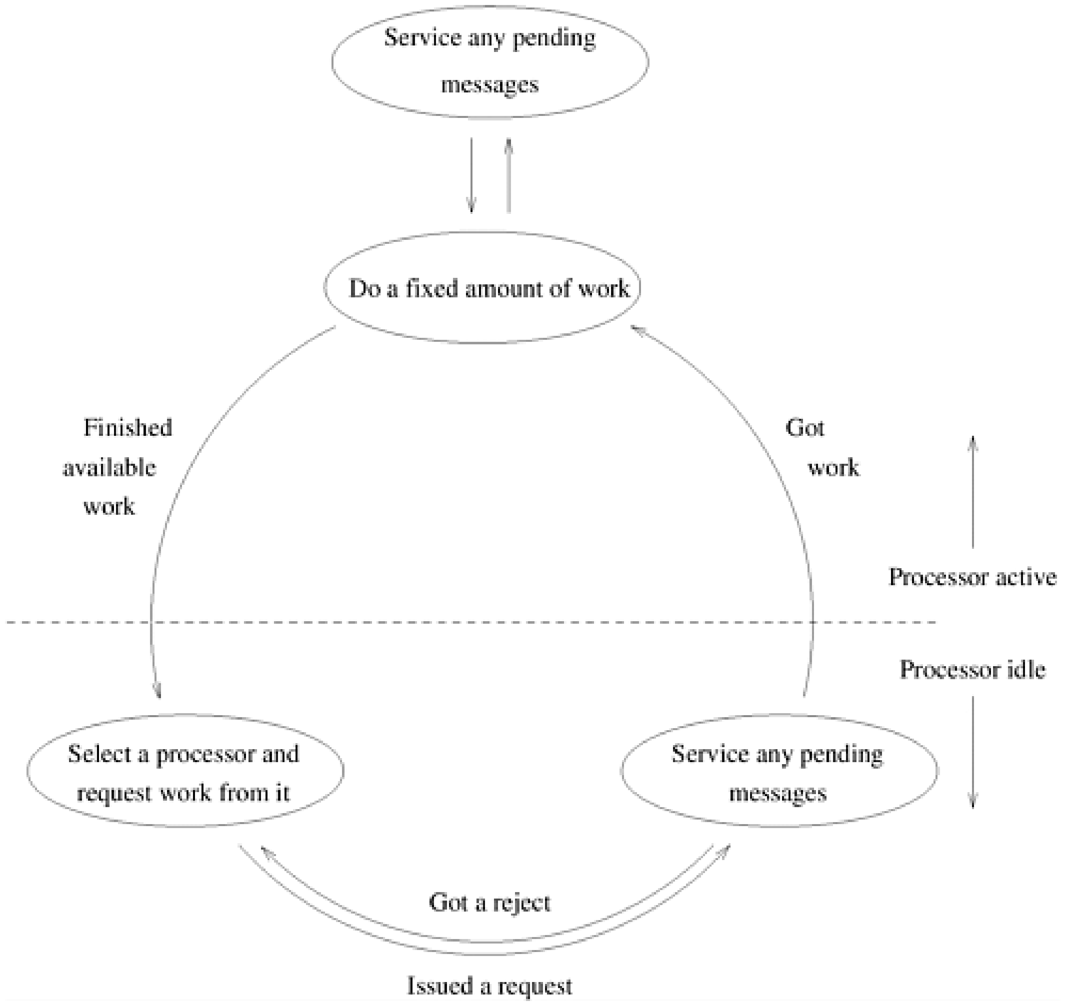
A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Parameters in Parallel DFS: Work Splitting

- **Some possible strategies**

1. **Send nodes near the bottom of the stack**
   - Works well with uniform search space; has low splitting cost
2. **Send nodes near the cutoff depth**
   - Performs better with a strong heuristic (tries to distribute the parts of the search space likely to contain a solution)
3. **Send half the nodes between the bottom and the cutoff depth**
   - Works well with uniform and irregular search space

Parallel DFS: Dynamic Load Balancing



A generic scheme for dynamic load balancing.

A. Grama et al., “Introduction to Parallel Computing,” Addison Wesley, 2003

Parallel DFS: Dynamic Load Balancing

- The entire space is assigned to one processor to begin with.
- When a processor runs out of work, it gets more work from another processor.
  - Message passing machines: work requests and responses
  - Shared address space machines: locking and extracting work
- Unexplored states can be conveniently stored as local stacks at processors.
- On reaching final state at a processor, all processors terminate.

A. Grama et al., “Introduction to Parallel Computing,” Addison Wesley, 2003

Load-Balancing Schemes

- **Who do you request work from? Note that we would like to distribute work requests evenly, in a global sense.**
- **Asynchronous round robin (ARR)**
  - **Each process maintains a counter and makes requests in a round-robin fashion.**
- **Global round robin (GRR)**
  - **The system maintains a global counter and requests are made in a round-robin fashion, globally.**
- **Random polling (RP)**
  - **Request a randomly selected process for work.**

A. Grama et al., “Introduction to Parallel Computing,” Addison Wesley, 2003

Analyzing DFS

- **We can't compute, analytically, the serial work W or parallel time W_p in terms of input size n**
- **Instead, we quantify total overhead T_o in terms of W to compute scalability.**
  - $T_O = pW_p - W$
  - **Overhead is due to**
    - **Communication (requesting and sending work)**
    - **Idle time (waiting for work)**
    - **Termination detection**
    - **Contention for shared resources (e.g., the global counter in GRR)**
    - **Search overhead factor**
  - **For dynamic load balancing, idling time is subsumed by communication.**
- **We must quantify the total number of requests in the system.**

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Search Overhead Factor

- **The amount of work done by serial and parallel formulations of search algorithms is often different.**
- **Let W be serial work and pW_P be parallel work. Search overhead factor s is defined as pW_P/W.**
- **Upper bound on speedup is $p \times 1/s$.**
  - **ATTENTION: $W/W_P < 1$ is possible (speedup anomalies)**

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Analyzing DFS: Assumptions

- **Search overhead factor = one**
- **Work at any processor can be partitioned into independent pieces as long as its size exceeds a threshold ε.**
- **A reasonable work-splitting mechanism is available.**
  - **If work w at a processor is split into two parts ψw and $(1-\psi)w$, there exists an arbitrarily small constant α $(0 < \alpha \leq 0.5)$, such that $\psi w > \alpha w$ and $(1-\psi)w > \alpha w$.**
  - **The constant α sets a lower bound on the load imbalance from work splitting.**

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Analyzing DFS

- **If processor P_i initially had work w_i, after a single request by processor P_j and split, neither P_i nor P_j have more than $(1–\alpha)w_i$ work.**
- **For each load balancing strategy, we define $V(P)$ as the total number of work requests after which each processor receives at least one work request (note that $V(p) \geq p$).**
- **Assume that the largest piece of work at any point is W.**
- **After $V(p)$ requests, the maximum work remaining at any processor is less than $(1–\alpha)W$; after $2V(p)$ requests, it is less than $(1–\alpha)^2W$; …**
- **After $(\log_{1/1(1-\alpha)}(W/\varepsilon))V(p)$ requests, the maximum work remaining at any processor is below a threshold value ε.**
- **The total number of work requests is $O(V(p) \log W)$.**

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Analyzing DFS

- **If t_{comm} is the time required to communicate a piece of work, then the communication overhead T_O is**

$$T_O = t_{comm} V(p) log\ W$$

The corresponding efficiency E is given by

$$E = \frac{1}{1 + T_o/W} = \frac{1}{1 + (t_{comm} V(p) \log W)/W}$$

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Analyzing DFS: for Various Schemes

- **Asynchronous Round Robin**
  - $V(p) = O(p^2)$ **in the *worst case*.**
- **Global Round Robin**
  - $V(p) = p$**.**
- **Random Polling**
  - **Worst case** $V(p)$ **is unbounded.**
  - **We do *average case* analysis.**

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

for Random Polling

- Let $F(i,p)$ represent a state in which i of the processors have been requested, and $p–i$ have not.
- Let $f(i,p)$ denote the average number of trials needed to change from state $F(i,p)$ to $F(p,p)$ ($V(p) = f(0,p)$).

$$
\begin{aligned}
f(i,p) &= \frac{i}{p}(1 + f(i,p)) + \frac{p-i}{p}(1 + f(i+1,p)), \\
\frac{p-i}{p} f(i,p) &= 1 + \frac{p-i}{p} f(i+1,p), \\
f(i,p) &= \frac{p}{p-i} + f(i+1,p).
\end{aligned}
$$

A. Grama et al., “Introduction to Parallel Computing,” Addison Wesley, 2003

for Random Polling

- **We have:**

$$f(0,p) = p \times \sum_{i=0}^{p-1} \frac{1}{p-i},$$

$$= p \times \sum_{i=1}^{p} \frac{1}{i},$$

$$= p \times H_p,$$

harmonic number

As p **becomes large,** $H_p \simeq 1.69 \ln p$**. Thus,** $V(p) = O(p \log p)$**.**

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Analysis of Load-Balancing Schemes

- **If** $t_{comm} = O(1)$**, we have** $T_0 = O(V(p) \log W)$**.**
- **Asynchronous Round Robin: Since** $V(p) = O(p^2)$**,** $T_0 = O(p^2 \log w)$**. It follows that:**

$$W = O(p^2 \log(p^2 \log W)),$$

$$= O(p^2 \log p + p^2 \log \log W)$$

$$= O(p^2 \log p)$$

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Analysis of Load-Balancing Schemes

- **Global Round Robin: Since** $V(p) = O(p)$, $T_0 = O(p \log W)$**.** **It follows that** $W = O(p \log p)$**.**

  However, there is contention here! The global counter must be incremented $O(p \log W)$ **times in** $O(W/p)$ **time.**

  From this, we have: $$\frac{W}{p} = O(p \log W)$$

  and $W = O(p^2 \log p)$**.**

  The worse of these two expressions, $W = O(p^2 \log p)$ **is the isoefficiency.**

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Analysis of Load-Balancing Schemes

- **Random Polling: We have** $V(p) = O(p \log p)$**,**

  $T_o = O(p \log p \log W)$

  Therefore $W = O(p \log^2 p)$**.**

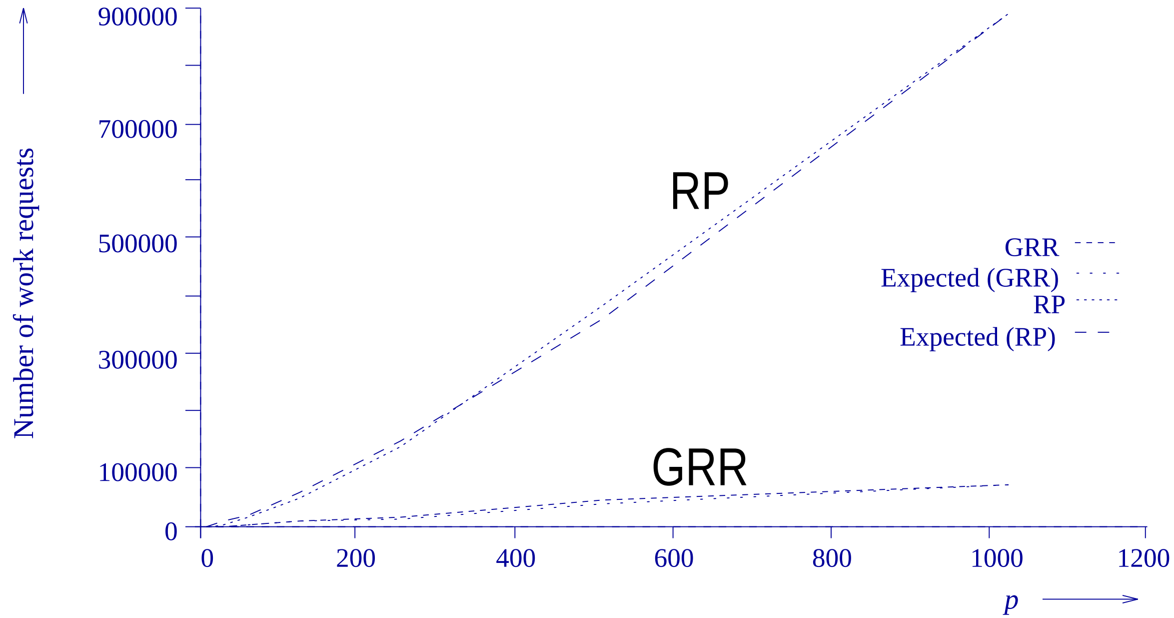
A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Analysis of Load-Balancing Schemes: Conclusions

- **Asynchronous round robin has poor performance because it makes a large number of work requests.**
- **Global round robin has poor performance because of contention at counter, although it makes the least number of requests.**
- **Random polling strikes a desirable compromise.**

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

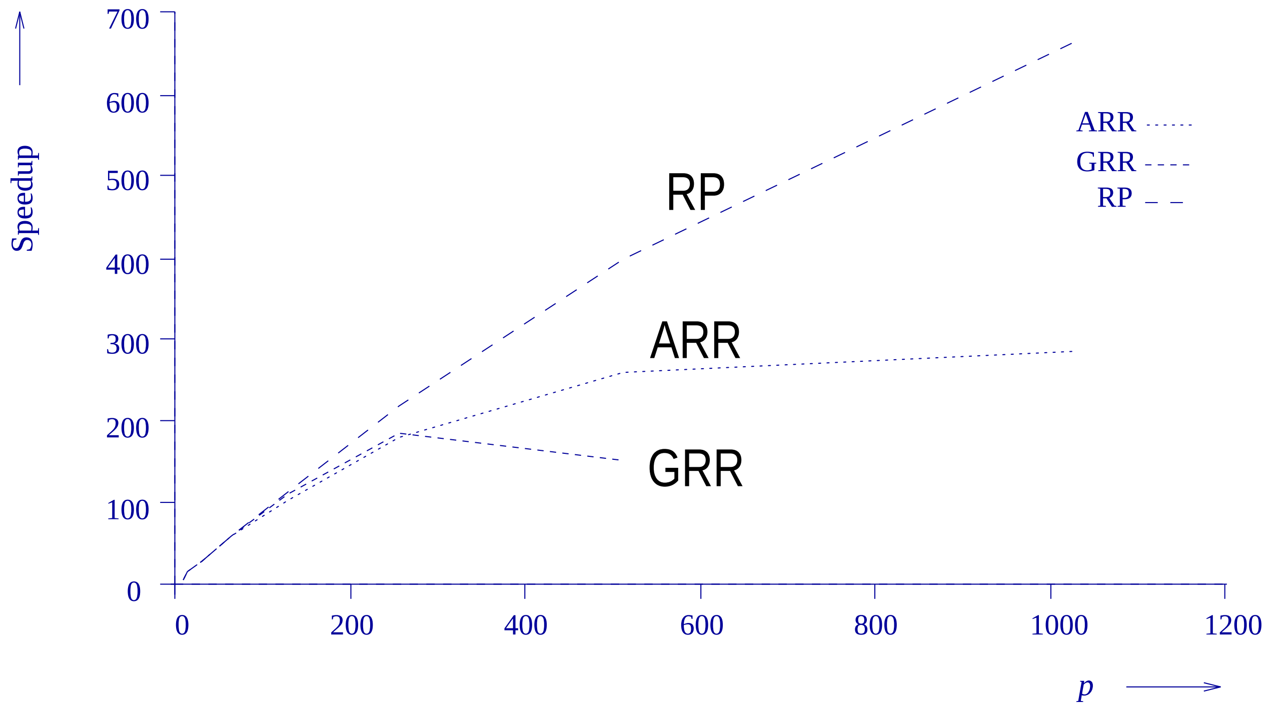
Experimental Validation: Satisfiability Problem



Number of work requests generated for RP and GRR and their expected values ($O(p \log^2 p)$ and $O(p \log p)$ respectively).

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

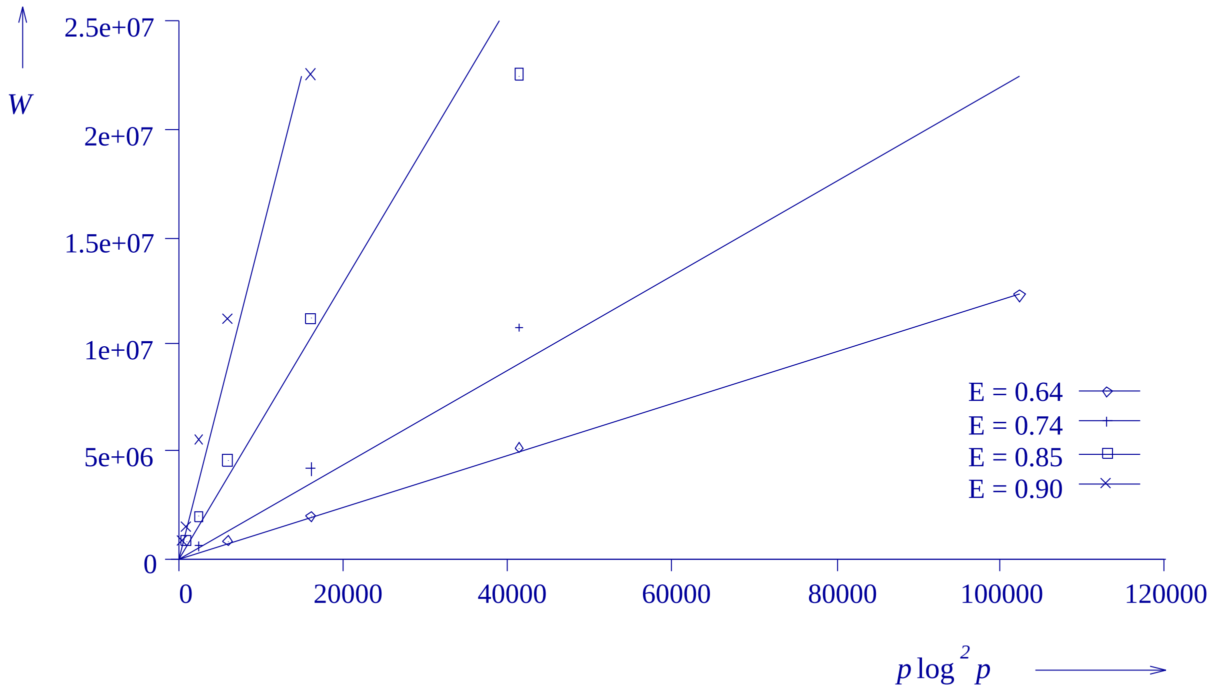
Experimental Validation: Satisfiability Problem



Speedups of parallel DFS using
ARR, GRR and RP load-balancing schemes.

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Experimental Validation: Satisfiability Problem



Experimental isoefficiency curves for RP for different efficiencies.

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

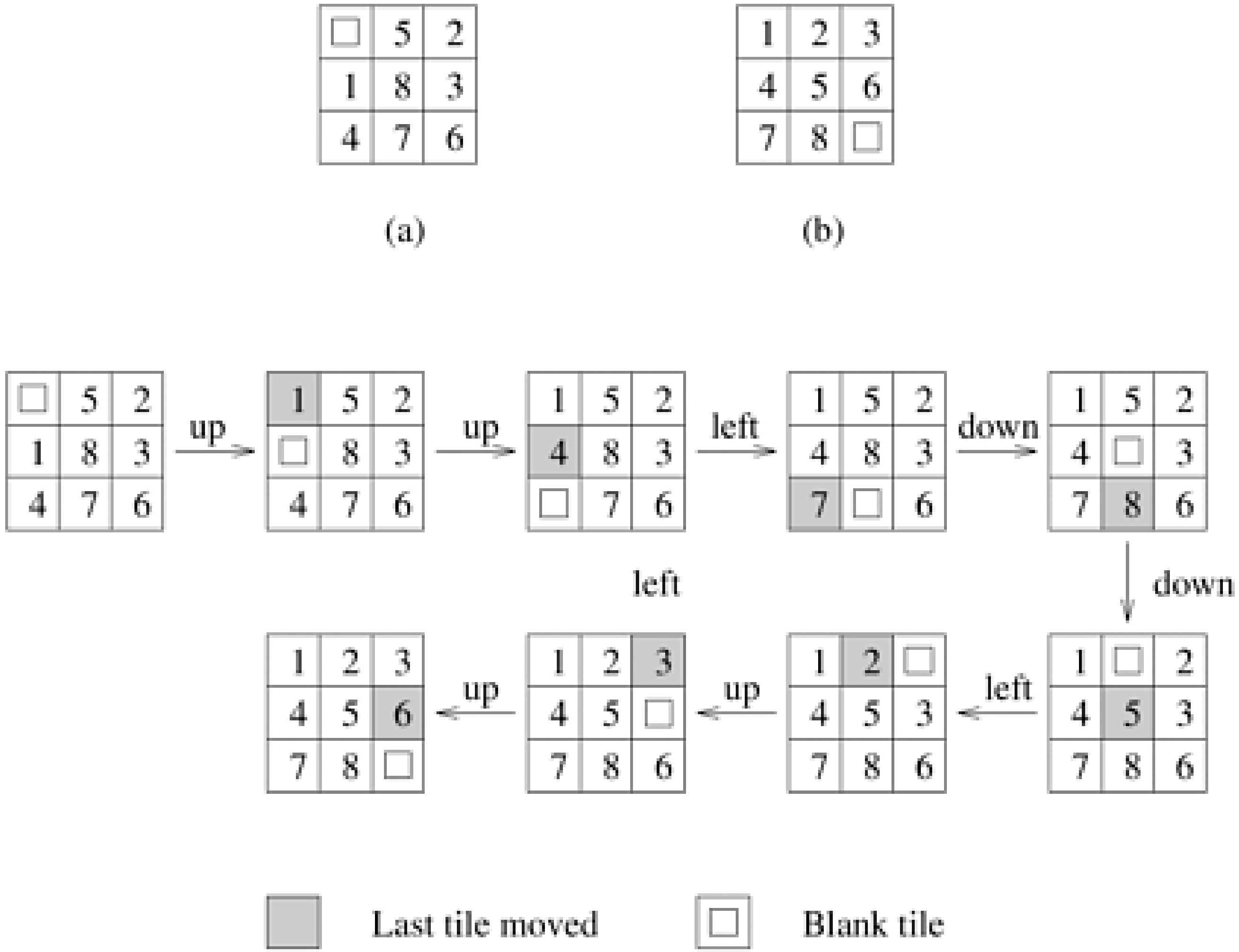
Best-First Search: 8-puzzle

1	2	3
4	5	6
7	8	Hole

This is the solution state.
Tiles slide up, down, or
sideways into hole.

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

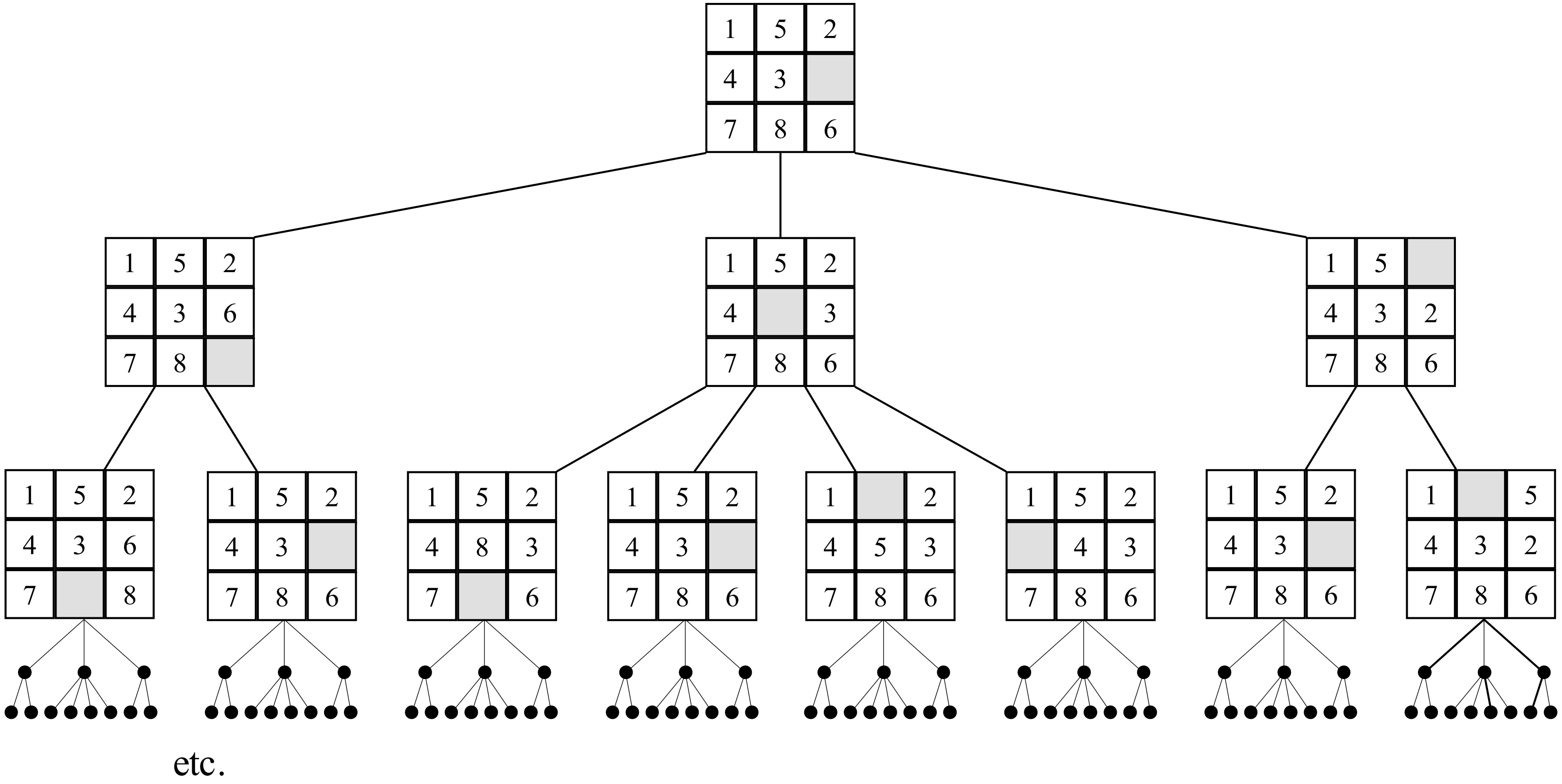
Example: Solve 8-Puzzle Problem



(c)

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

State Space Tree Represents Possible Moves



Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Example: 8-Puzzle Problem

- The 8-puzzle problem consists of a 3×3 grid containing eight tiles, numbered one through eight.
- One of the grid segments (called the "blank") is empty. A tile can be moved into the blank position from a position adjacent to it, thus creating a blank in the tile's original position.
- The goal is to move from a given initial position to the final position in a minimum number of moves.

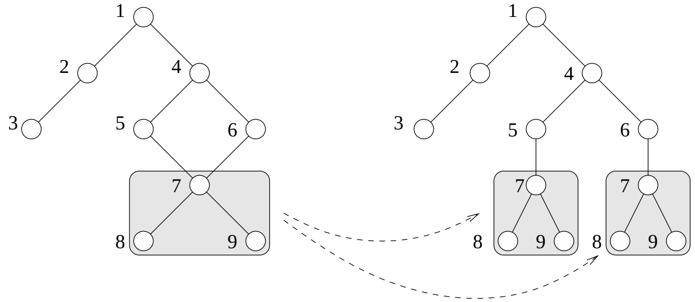
A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Search Space

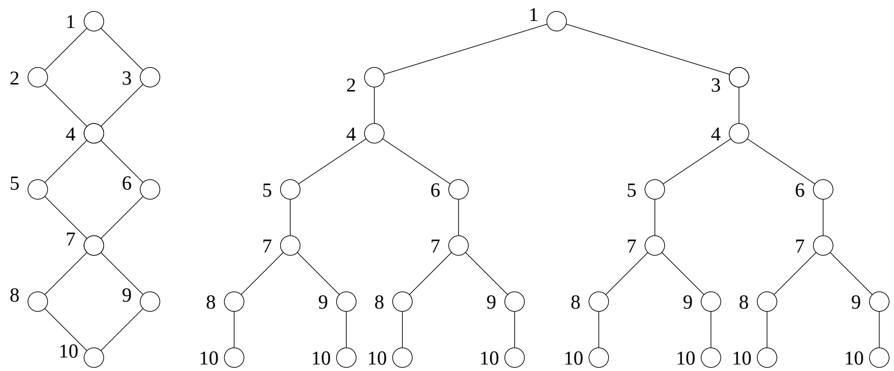
- **Is the search space a tree or a graph?**
  - **The space of an 8-puzzle is a graph.**
- **This has important implications for search since unfolding a graph into a tree can have significant overheads.**

A. Grama et al., “Introduction to Parallel Computing,” Addison Wesley, 2003

Unfolded Search Tree



(a)

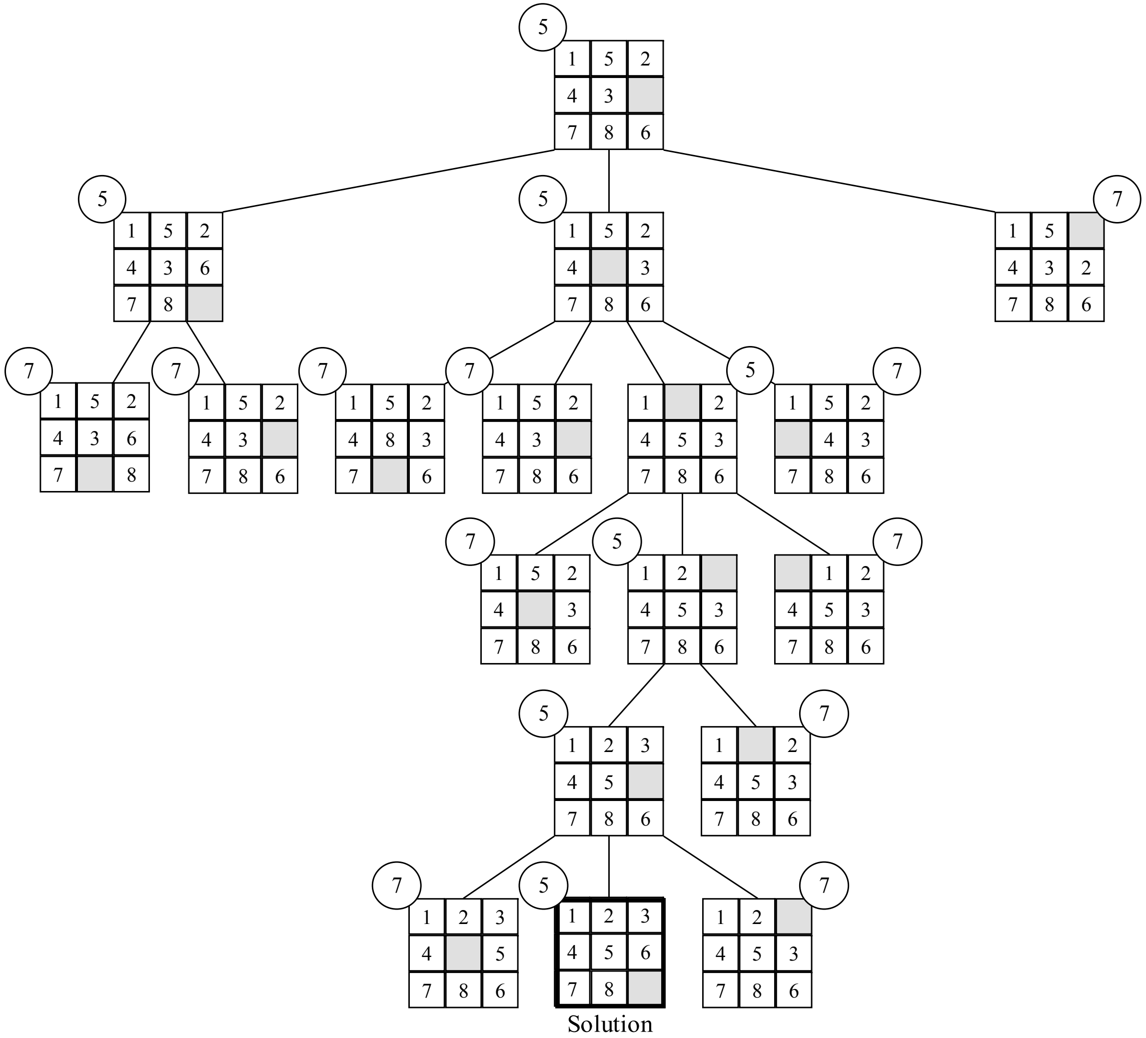


(b)

Two examples of unfolding a graph into a tree.

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Best-First Search of 8-puzzle



Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Manhattan Distance

4	3	2	3	4
3	2	1	2	3
2	1	0	1	2
3	2	1	2	3
4	3	2	3	4

Manhattan distance from the yellow intersection.

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

A Lower Bound Function

- A lower bound on number of moves needed to solve puzzle is sum of Manhattan distance of each tile's current position from its correct position
- Depth of node in state space tree indicates number of moves made so far
- Adding two values gives lower bound on number of moves needed for any solution, given moves made so far
- We always search from node having smallest value of this function (best-first search)

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Pseudocode: Sequential Algorithm

```
Intialize (q)
Insert (q, initial)
repeat
    u ← Delete_Min (q)
    if u is a solution then
            Print_solution (u)
            Halt
    else
            for i ← 1 to Possible_Constraints (u) do
                    Add constraint i to u, creating v
                    Insert (q, v)
            endfor
    endif
forever
```

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

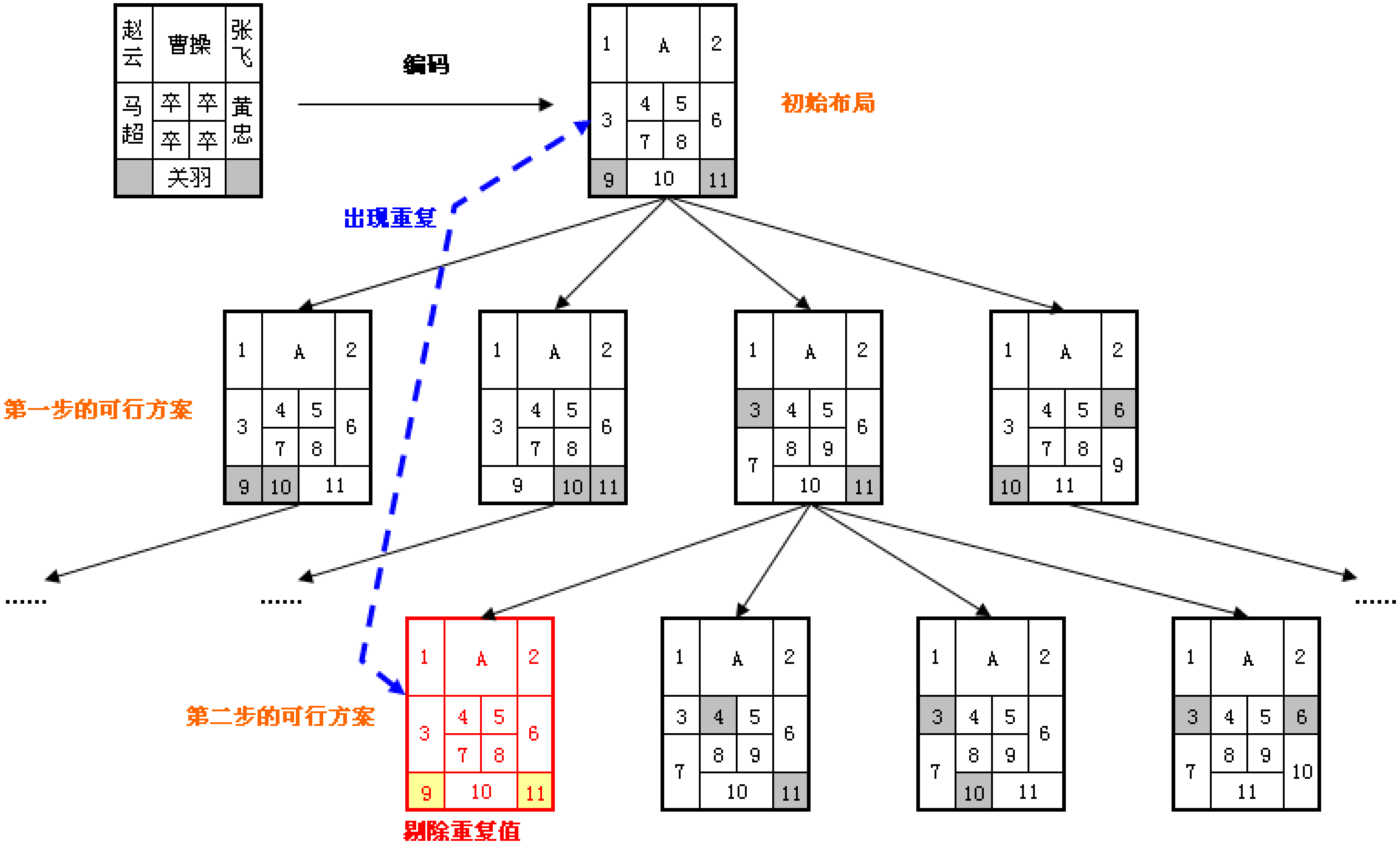
Time and Space Complexity

- **In worst case, lower bound function causes function to perform breadth-first search**
- **Suppose branching factor is b and optimum solution is at depth k of state space tree**
- **Worst-case time complexity is $\Theta(b^k)$**
- **On average, b nodes inserted into priority queue every time a node is deleted**
- **Worst-case space complexity is $\Theta(b^k)$**
- **Memory limitations often put an upper bound on the size of the problem that can be solved**

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Similar Example: Chinese Klotski

Similar Example: Chinese Klotski



http://www.cnblogs.com/zhenyulu/articles/75399.html

Parallel Best-First Search

- **We will develop a parallel algorithm suitable for implementation on a multicomputer or distributed multiprocessor**
- **Conflicting goals**
  - **Want to maximize ratio of local to non-local memory references**
  - **Want to ensure processors searching worthwhile portions of state space tree**

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Single Priority Queue

- Maintaining a single priority queue not a good idea
- Communication overhead too great
- Accessing queue is a performance bottleneck
- Does not allow problem size to scale with number of processors

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Multiple Priority Queues

- **Each process maintains separate priority queue of unexamined subproblems**
- **Each process retrieves subproblem with smallest lower bound to continue search**
- **Occasionally processes send unexamined subproblems to other processes**

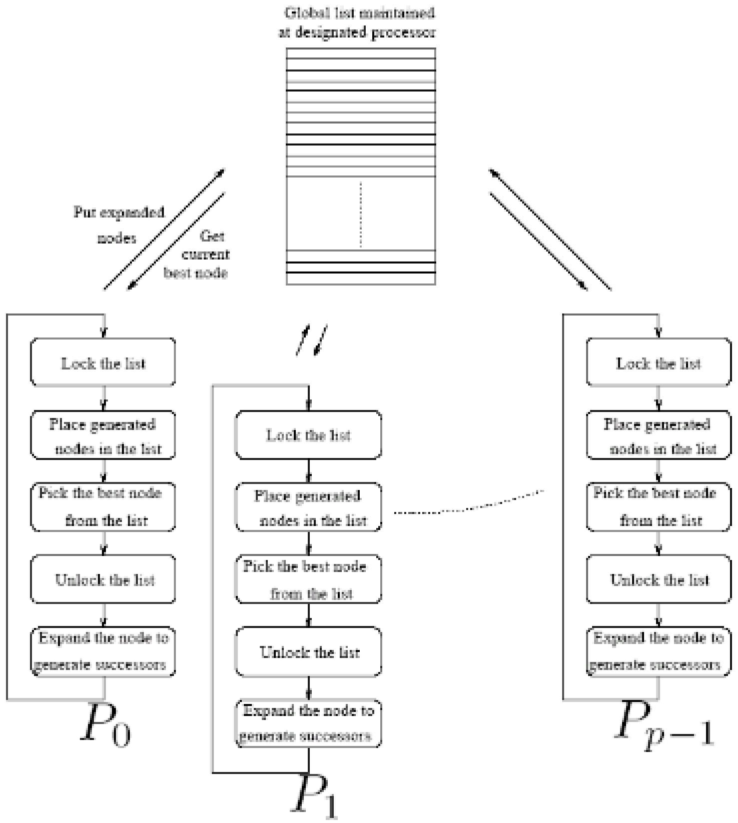
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Parallel Best-First Search

- **The core data structure is the open list (typically implemented as a priority queue).**
- **Each processor locks this queue, extracts the best node, unlocks it.**
- **Successors of the node are generated, their heuristic functions estimated, and the nodes inserted into the open list as necessary after appropriate locking.**
- **Termination signaled when we find a solution whose cost is better than the best heuristic value in the open list.**
- **Since we expand more than one node at a time, we may expand nodes that would not be expanded by a sequential algorithm.**

A. Grama et al., “Introduction to Parallel Computing,” Addison Wesley, 2003

Parallel Best-First Search: Centralized Strategy



A general schematic for parallel best-first search using a centralized strategy. The locking operation is used here to serialize queue access by various processors.

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Parallel Best-First Search

- **The open list is a point of contention.**
- **Let t_{exp} be the average time to expand a single node, and t_{access} be the average time to access the *open* list for a single-node expansion.**
- **If there are n nodes to be expanded by both the sequential and parallel formulations (assuming that they do an equal amount of work), then the sequential run time is given by $n(t_{access} + t_{exp})$.**
- **The parallel run time will be at least nt_{access}.**
- **Upper bound on the speedup is $(t_{access} + t_{exp})/t_{access}$**

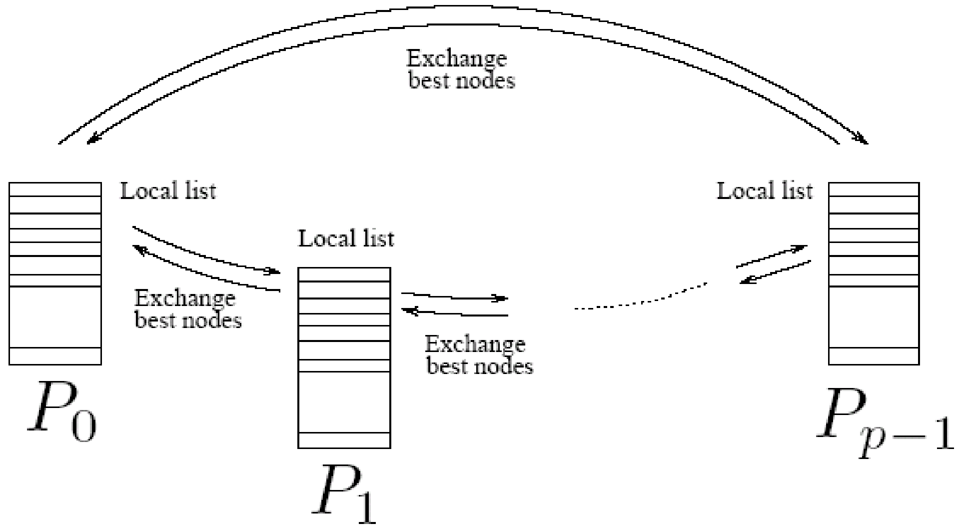
A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Parallel Best-First Search

- **Avoid contention by having multiple open lists.**
- **Initially, the search space is statically divided across these open lists.**
- **Processors concurrently operate on these open lists.**
- **Since the heuristic values of nodes in these lists may diverge significantly, we must periodically balance the quality of nodes in each list.**
- **A number of balancing strategies based on (i) *random*, (ii) *ring*, or (iii) *blackboard* communications are possible.**

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

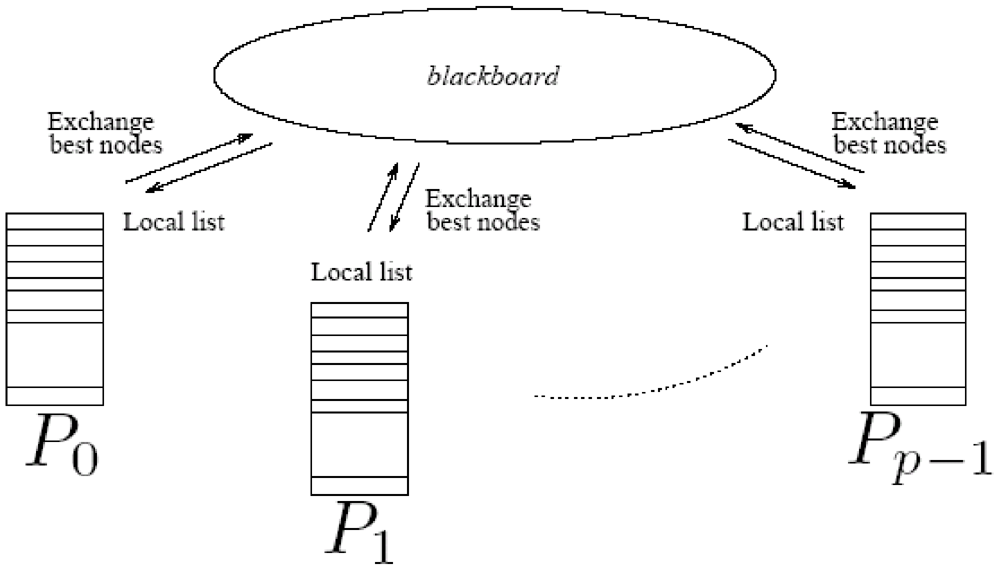
Parallel Best-First Search: Ring Communication Strategy



A message-passing implementation of parallel best-first search using the ring communication strategy.

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Parallel Best-First Search: Blackboard Communication Strategy



An implementation of parallel best-first search using the blackboard communication strategy.

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Parallel Best-First Graph Search

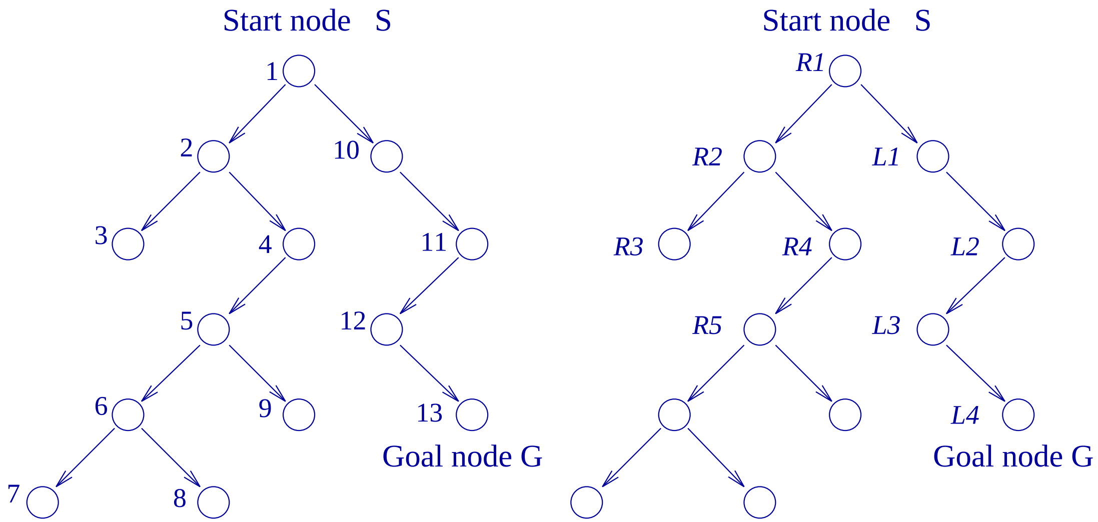
- **Graph search involves a closed list, where the major operation is a lookup (on a key corresponding to the state).**
- **The classic data structure is a hash.**
- **Hashing can be parallelized by using two functions - the first one hashes each node to a processor, and the second one hashes within the processor.**
- **This strategy can be combined with the idea of multiple open lists.**
- **If a node does not exist in a closed list, it is inserted into the open list at the target of the first hash function.**
- **In addition to facilitating lookup, randomization also equalizes quality of nodes in various open lists.**

Speedup Anomalies in Parallel Search

- **Since the search space explored by processors is determined dynamically at runtime, the actual work might vary significantly.**
- **Executions yielding speedups greater than p by using p processors are referred to as *acceleration anomalies*. Speedups of less than p using p processors are called *deceleration anomalies*.**
- **Speedup anomalies also manifest themselves in best-first search algorithms.**
- **If the heuristic function is good, the work done in parallel best-first search is typically more than that in its serial counterpart.**

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Speedup Anomalies in Parallel Search



Total number of nodes generated by sequential formulation = 13

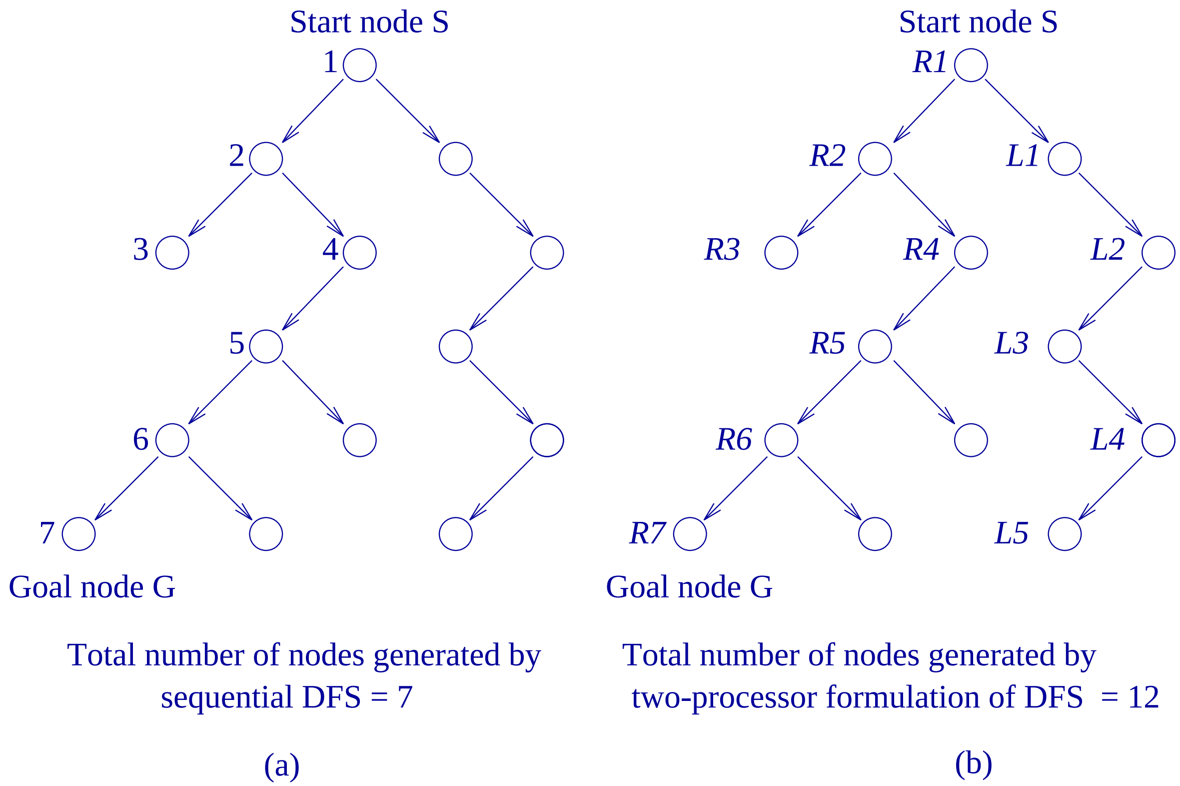
Total number of nodes generated by two-processor formulation of DFS = 9

(a) (b)

The difference in number of nodes searched by sequential and parallel formulations of DFS. For this example, parallel DFS reaches a goal node after searching fewer nodes than sequential DFS.

A. Grama et al., “Introduction to Parallel Computing,” Addison Wesley, 2003

Speedup Anomalies in Parallel Search



A parallel DFS formulation that searches more nodes than its sequential counterpart

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Summary

- **Parallel depth-first search**
  - **Load balancing schemes: ARR, GRR, RP**
  - **Scalability analysis**
- **Parallel best-first search**
  - **Centralized strategy**
  - **Communication strategies: random, ring, blackboard**
- **Speedup anomalies**

Thank You!

In This Lecture …

- **Terminology in combinatorial search**
  - **Divide and conquer**
  - **Backtrack search**
  - **Branch and bound**

Terminology

- Combinatorial algorithm: computation performed on discrete structure
- Combinatorial search: finding one or more optimal or suboptimal solutions in a defined problem space
- Kinds of combinatorial search problem
  - Decision problem
  - Optimization problem

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Combinatorial Search: Examples

- Laying out circuits in VLSI
- Planning motion of robot arms
- Assigning crews to airline flights
- Proving theorems
- Playing games

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

We'll Review Several Combinatorial Search Methods

- **Divide and conquer**
- **Backtrack search**
- **Branch and bound**

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Search Tree

- **Each node represents a problem or sub-problem**
- **Root of tree: initial problem to be solved**
- **Children of a node created by adding constraints**
- **AND node: to find solution, must solve problems represented by all children nodes**
- **OR node: to find solution, solve any of problems represented by children nodes**

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Search Tree (cont.)

- AND tree
  - Contains only AND nodes
  - Divide-and-conquer algorithms
- OR tree
  - Contains only OR nodes
  - Backtrack search and branch and bound
- AND/OR tree
  - Contains both AND and OR nodes
  - Game trees

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Divide and Conquer

- **Divide-and-conquer methodology**
  - **Partition a problem into subproblems**
  - **Solve the subproblems**
  - **Combine solutions to subproblems**
- **Recursive: subproblems may be solved using the divide-and-conquer methodology**
- **Examples**
  - **quicksort**
  - **closest pair of points problem**
  - **…**

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Centralized Multiprocessor Divide and Conquer

- **Unsolved subproblems kept in one stack**
- **Processors needing work can access stack**
- **Processors with extra work can put it on the stack**
- **Effective workload balancing mechanism**
- **Stack can become a bottleneck as number of processors increases**

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Multicomputer Divide and Conquer

- Subproblems must be distributed among memories of individual processors
- Two designs
  - Original problem and final solution stored in memory of a single processor
  - Both original problem and final solution distributed among memories of all processors

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Design 1

Original problem and final solution stored in memory of a single processor

- Algorithm has three phases
  - Phase 1: problems divided and propagated throughout the parallel computer
  - Phase 2: processors compute solutions to their subproblems
  - Phase 3: partial results are combined
- Maximum speedup limited by propagation and combining overhead

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Design 2

- Both original problem and final solution are distributed among processors' memories
- Eliminates starting up and winding down phases of design 1
- Allows maximum problem size to increase with number of processors
- Used this approach for parallel quicksort algorithms
- Challenge: keeping workloads balanced among processors

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Distributed Termination Detection

- Suppose we only want to print one solution
- We want all processes to halt as soon as one process finds a solution
- This means processes must periodically check for messages
  - Every process calls MPI_Iprobe every time search reaches a particular level (such as the cutoff depth)
  - A process sends a message after it has found a solution

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

MPI_Iprobe

- **Nonblocking test for a message**

```
int MPI_Iprobe(
   int source,
   int tag,
   MPI_Comm comm,
   int *flag,
   MPI_Status *status);
```

- **flag: true if a message with the specified source, tag, and communicator is available**
- **It is not necessary to receive a message immediately after it has been probed for, and the same message may be probed for several times before it is received.**

Simple (Incorrect) Algorithm

- A process halts after one of the following events has happened:
  - It has found a solution and sent a message to all of the other processes
  - It has received a message from another process
  - It has completely searched its portion of the state space tree

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Review: Discrete Optimization

- **A *discrete optimization problem* (DOP) can be expressed as a tuple (S, f).**
  - **The set S is a finite or countably infinite set of all solutions that satisfy specified constraints.**
  - **The function f is the cost function that maps each element in set S onto the set of real numbers R.**
- **The objective of a DOP is to find a feasible solution x_{opt}, such that $f(x_{opt}) \leq f(x)$ for all $x \in S$.**
- **A number of diverse problems such as VLSI layouts, robot motion planning, test pattern generation, and facility location can be formulated as DOPs.**

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

DOP Example: Integer Programming

- In the 0/1 *integer-linear-programming (ILP)* problem
  - We are given an $m \times n$ matrix A, an $m \times 1$ vector b, and an $n \times 1$ vector c.
  - The objective is to determine an $n \times 1$ vector x, whose elements can take on only the value 0 or 1.
- The vector x must satisfy the constraints

  $Ax \leq b$ and $x \in \{0,1\}^n$

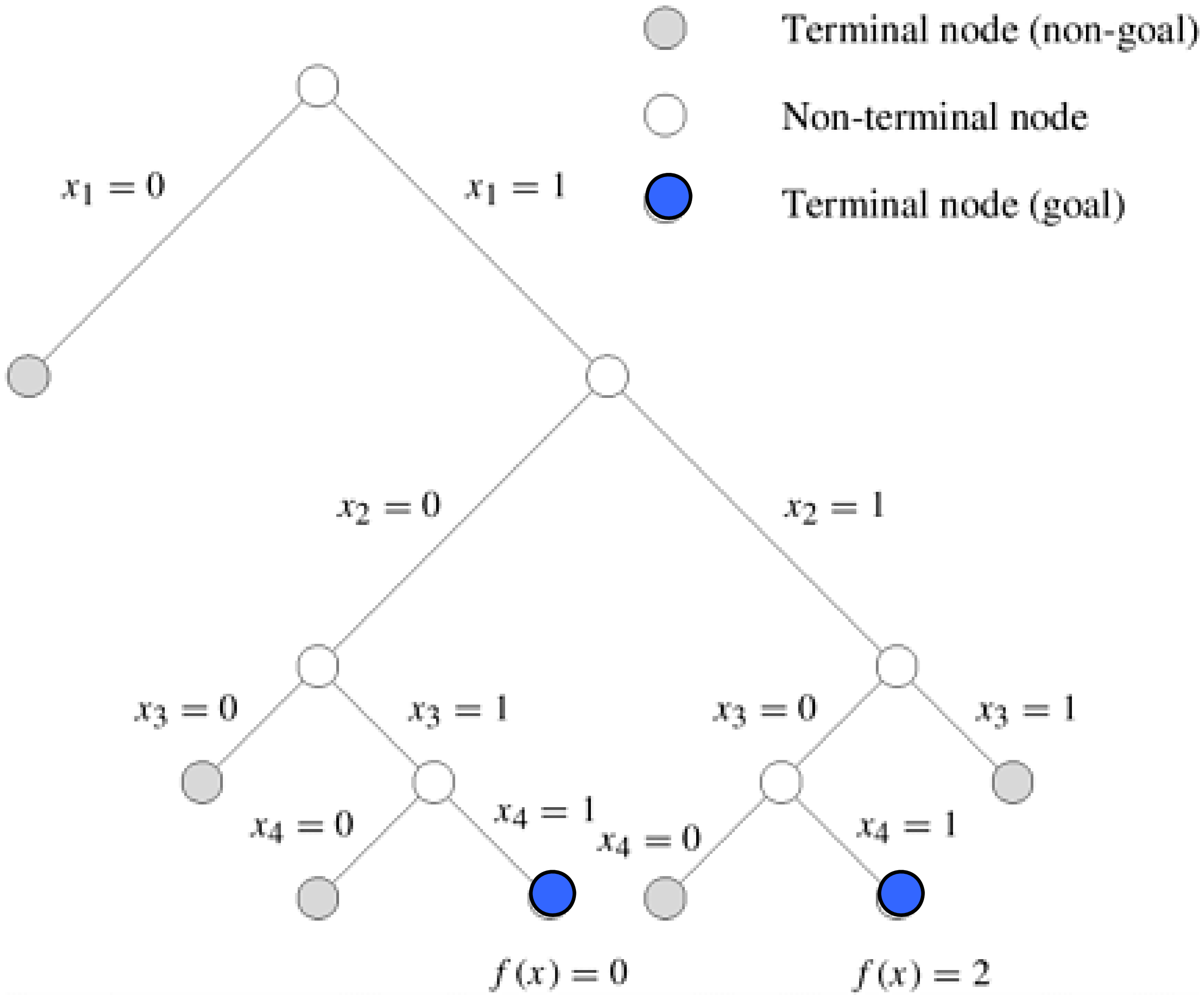
  and the function

  $$f(x) = c^T x$$

  must be minimized.

A. Grama et al., “Introduction to Parallel Computing,” Addison Wesley, 2003

Search Tree of an ILP



A. Grama et al., “Introduction to Parallel Computing,” Addison Wesley, 2003

In this Lecture …

- **Branch and bound**
  - **Variant of backtrack search**
- **Alpha-beta search**
  - **AND-OR-tree search**

Start-up Mode

- **Process *0* contains original problem in its priority queue**
- **Other processes have no work**
- **After process *0* distributes an unexamined subproblem, *2* processes have work**
- **A logarithmic number of distribution steps are sufficient to get all processes engaged**

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Efficiency

- **Conditions for solution to be found an guaranteed optimal**
  - **At least one solution node must be found**
  - **All nodes in state space tree with smaller lower bounds must be explored**
- **Execution time dictated by which of these events occurs last**
- **This depends on number of processes, shape of state space tree, communication pattern**

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Efficiency (cont.)

- **Sequential algorithm searches minimum number of nodes (never explores nodes with lower bounds greater than cost of optimal solution)**
- **Parallel algorithm may examine unnecessary nodes because each process searching *locally best* nodes**
- **Exchanging subproblems**
  - **Promotes distribute of subproblems with good lower bounds, reducing amount of wasted work**
  - **Increases communication overhead**

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Summary (1/5)

- Combinatorial search used to find solutions to a variety of discrete decision and optimization problems
- Can categorize problems by type of state space tree they traverse
- Divide-and-conquer algorithms traverse AND trees
- Backtrack search and branch-and-bound search traverse OR trees

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Summary (2/5)

- **Parallel divide and conquer**
  - **If problem starts on a single process and solution resides on a single process, then speedup limited by propagation and combining overhead**
  - **If problem and solution distributed among processors, efficiency can be much higher, but balancing workloads can still be a challenge**

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Summary (3/5)

- **Backtrack search**
  - Depth-first search applied to state space trees
  - Can be used to find a single solution or every solution
  - Does not take advantage of knowledge about the problem to avoid exploring subtrees that cannot lead to a solution
  - Requires space linear in depth of search (good)
  - Challenge: balancing work of exploring subtrees among processors
  - Need to implement distributed termination detection

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Summary (4/5)

- **Branch-and-bound search**
  - **Able to use lower bound information to avoid exploration of subtrees that cannot lead to optimal solution**
  - **Need to avoid search overhead without introducing too much communication overhead**
  - **Also need distributed termination detection**

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.