Electrical Engineering and Computer Science

EECS 358 - INTRODUCTION TO PARALLEL COMPUTING

Lecture 14

Parallel Search Algorithms for Optimization

Outline

- Background on Optimization Problems
- Sequential Search Algorithms
- Parallel Depth First Search
- Parallel Best First Search
- Speedup Anomalies in Parallel Search

Discrete Optimization Problems

- A discrete optimization problem can be expressed as a tuple (S, f).
- The set S is a finite or countably infinite set of all solutions that satisfy the specified constraints (feasible solutions)
- The function f is a cost function that maps each element of the set S to a real set of numbers.
- Example: 0/1 Integer Linear Programming Problem
- Example: 8-puzzle problem
- Example: VLSI CAD problems like placement, routing, floorplanning, testing

0/1 Integer Linear Programming Problem

- Given an $m \times n$ matrix A, an $m \times 1$ vector b, and an $n \times 1$ vector c.
- Objective is to determine an $n \times 1$ vector x whose elements can take values 0 or 1
- Constraint is $A.x \geq b$
- The function $f(x) = c^T x$ must be minimized.

Example of 0/1 Integer Linear Programming Problem

- Given contraints as follows:

$$5x_1 + 2x_2 + x_3 + 2x_4 \geq 8$$

$$x_1 - x_2 - x_3 + 2x_4 \geq 2$$

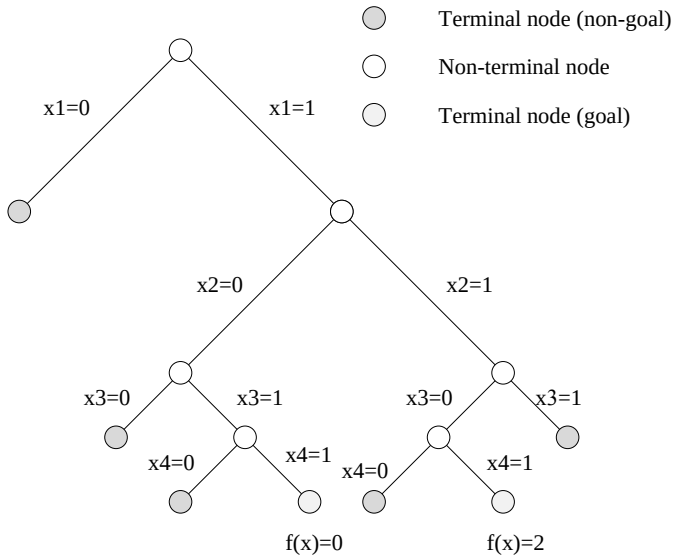
$$3x_1 + x_2 + x_3 + 3x_4 \geq 5$$

- Function f(x) to be minimized

$$f(x) = 2x_1 + x_2 - x_3 - 2x_4$$

Search Graph of ILP Problem

- Can reformulate ILP as search problem on $2^4 = 16$ possible values of $x_1, ...x_4$.

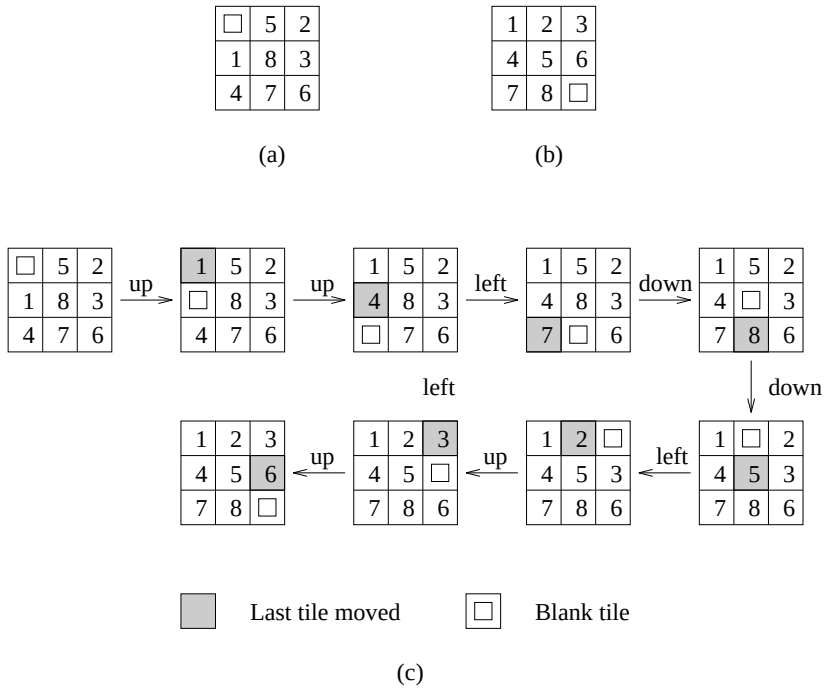


0/1 ILP Problem Search

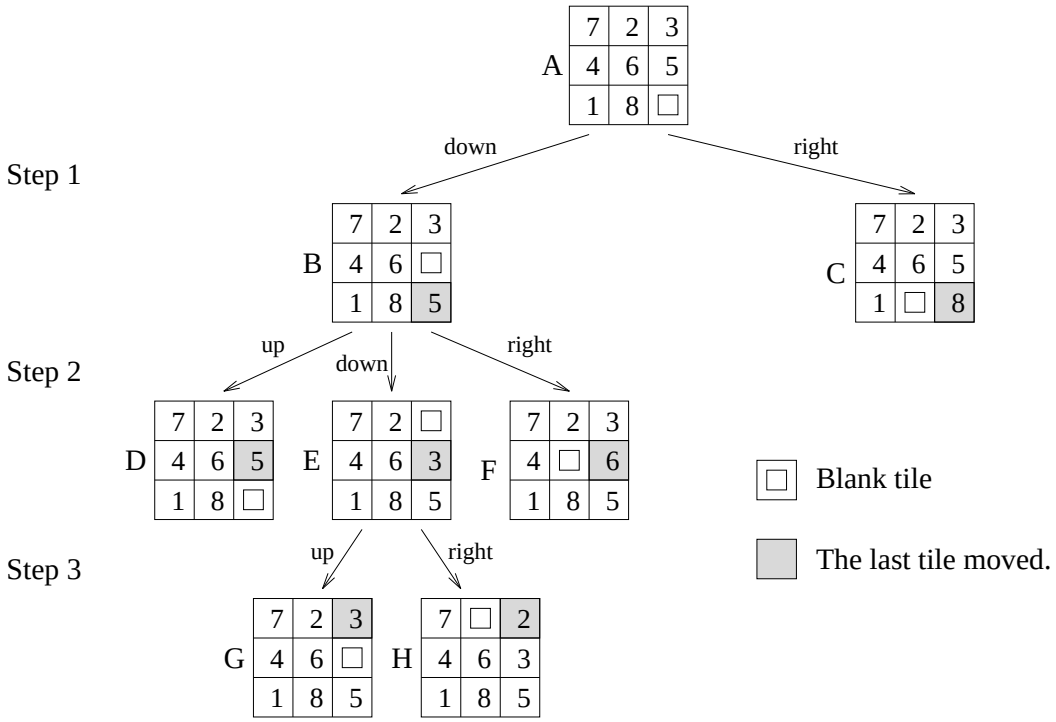
- Initial node represents the state in which none of the elements of the vector x have been assigned values.
- After a variable is assigned a value, it is called a *fixed* variable, others are called *free* variables.
- After instantiating a value (0/1) to a variable, it is possible to check if instantiation of remaining free variables can lead to a feasible solution.
- If not feasible, discard.
- If feasible, continue search.
- Function $f(x)$ is evaluated for each feasible solution
- Solution with minimum value is desired solution.

8-puzzle Problem

- Given initial configuration, and final configuration, determine minimum sequence of moves to reach final.



Example Search Graph of 8-puzzle



Heuristic Cost Functions

- For some problems, it is possible to estimate the cost to reach the goal state from an intermediate state.
- This cost is called *heuristic estimate*
- Let $h(x)$ denote the heuristics estimate of reaching goal state
- Let $g(x)$ be cost of reaching intermediate state from initial state.
- Define function $l(x) = g(x) + h(x)$ as a lower bound on cost of path to a goal state.

Example of Heuristic Function

- For 8-puzzle problem, distance between pair (i, j) and (k, l) is defined as $|i - k| + |j - l|$.
- This is called Manhattan Distance.
- Sum of Manhattan distances between initial and final configurations of all tiles is an estimate of the number of moves required.
- This is called Manhattan heuristic.

Sequential Search Algorithms

- Once a discrete optimization problem has been formulated as a graph search problem, it can be solved by branch-and-bound search and heuristic search algorithms.

- Depth First Search
  - Simple backtracking
  - Depth-first brach-and-bound
  - Iterative Deepening A*

- Best-First Search

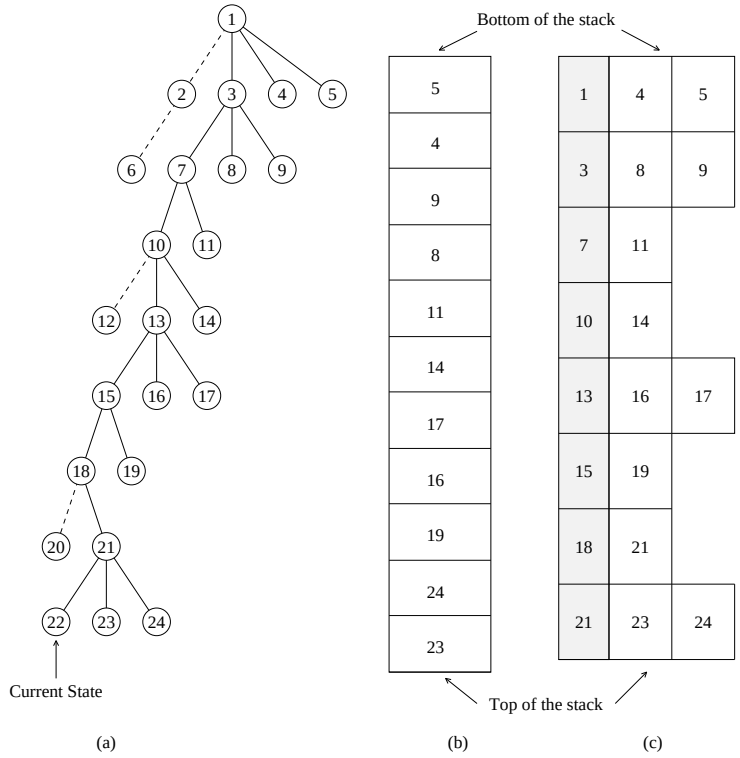
Depth First Search Algorithm

- DFS begins by expanding the initial node and generating its successors.
- In each step, DFS expands one of the most recently generated nodes.
- If this node has no successors, it backtracks and expands a different node.
- In some DFS algorithms, successors of a node are expanded in an order determined by the heuristic function.

Depth First Algorithms

- At each step of DFS algorithm, untried alternatives have to be stored.
- In general, if m is amount of storage required to store a state, and d is maximum depth, then total space requirement is $O(md)$.
- State space can be represented as a stack.
- One way is to push untried alternatives on the stack.
- Another way is to store untried alternatives along with ancestors on the stack.

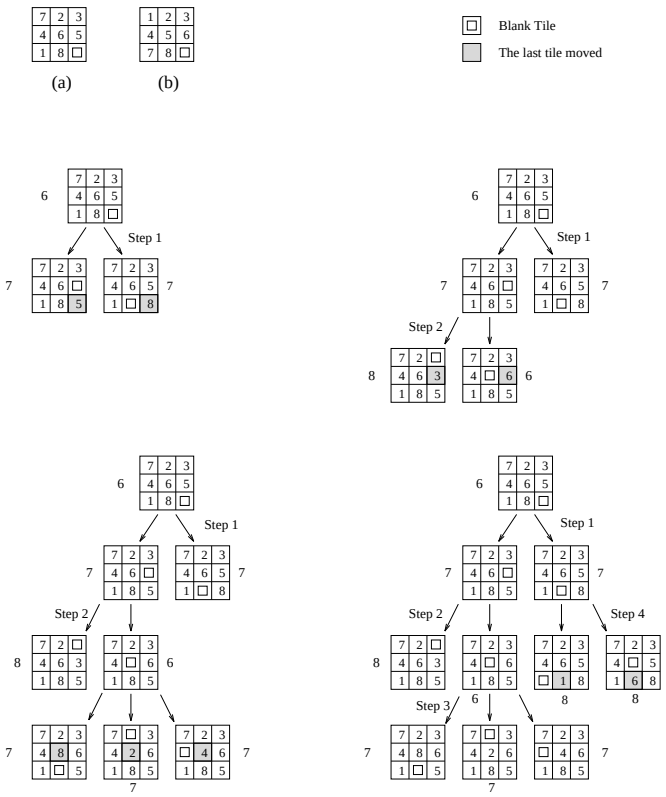
Example of DFS Tree and Stacks



Best First Search Algorithms

- These algorithms use heuristics to direct the search to portions of the search space likely to yield solutions.
- Smaller heuristic values are assigned more promising values.
- Initially, a start node is placed on *open* list.
- This list is sorted according to heuristic values.
- At each step, the most promising node from the open list is removed, placed on a *closed list*, and expanded.
- Successors of expanded node are placed on sorted *open* list.

Example of Best First Search on 8-puzzle



Parallel Search Algorithms

- Parallel search algorithms incur overhead from several sources
- Load imbalance
- Communication overhead
- Contention for shared resources
- Search overhead factor

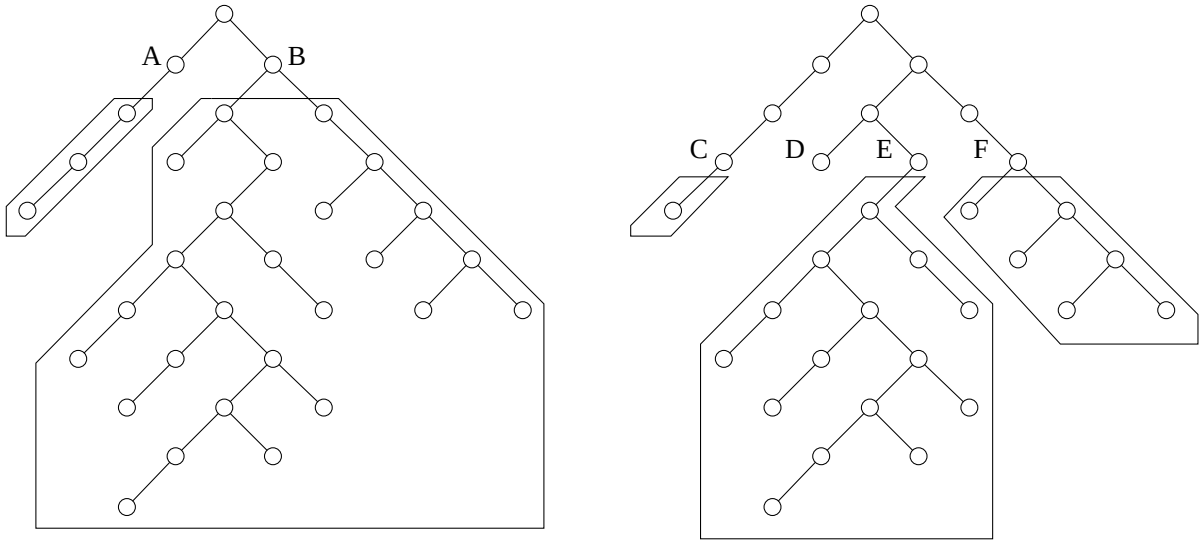
Search Overhead Factor

- Search overhead: Amount of work done by parallel formulation is often differnt from serial formulation.
- Let W be amount of work in single processor
- Let W_p be total work done by p processors
- Search overhead factor $= W_p/W$
- Upper bound on speedup is $S_p = p \times W/W_p$
- In some cases search overhead factor can be less than one, giving superlinear speedups.

Parallel Depth First Search

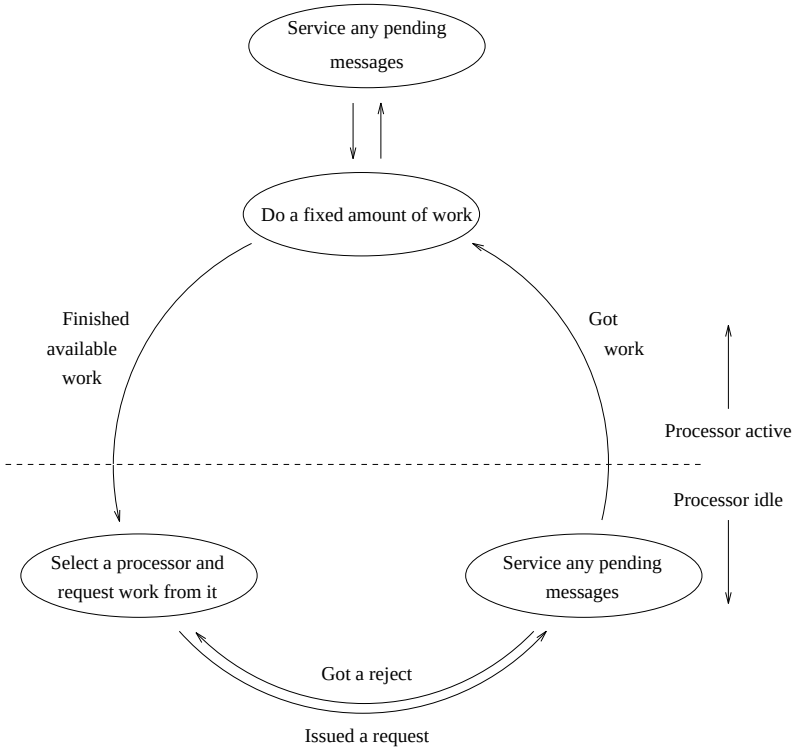
- Distribute the search space among processors
- If one statically partitions the search tree, can have load imbalance problems for many cases.
- Need dynamic load balancing for good speedups

Example of Static Partitioning of Search Space



(a) (b)

Example of Dynamic Partitioning



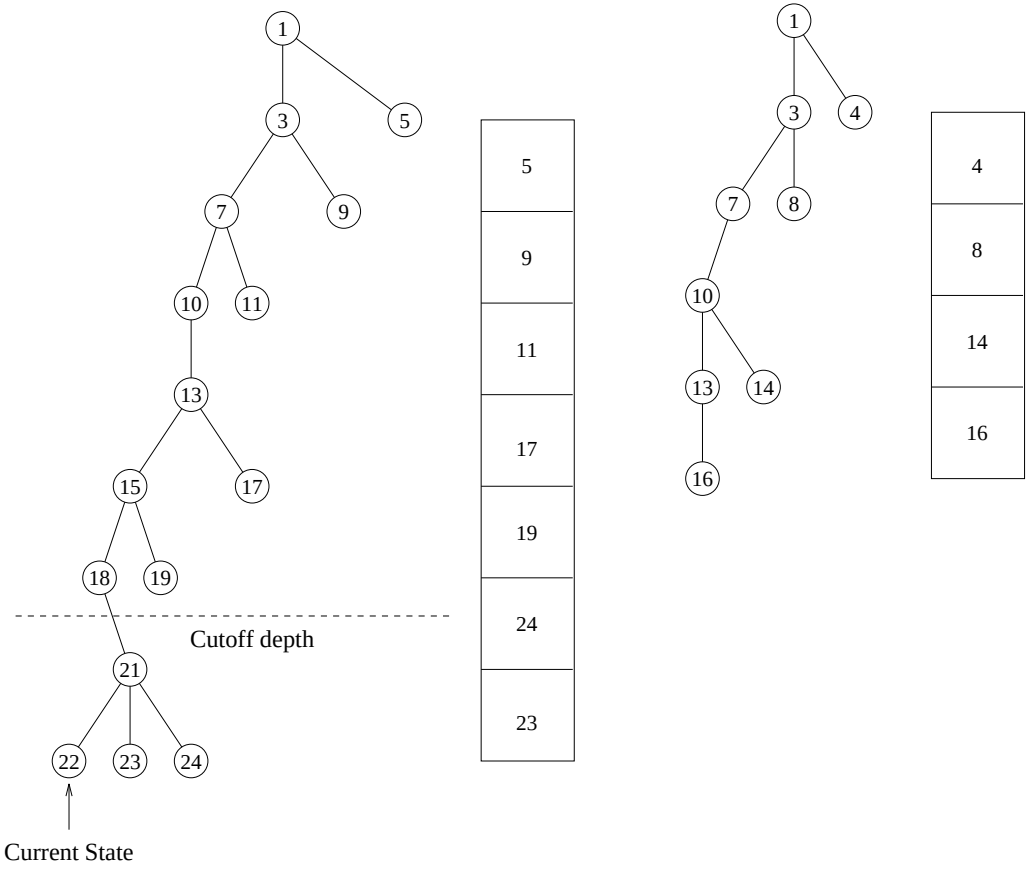
Parameters in Parallel DFS

- Work Splitting Strategies
- Load Balancing Strategies
- Termination Detection

Work Splitting Strategies

- When work is transferred, the donor's stack is split into two stacks, one of which is sent to the recipient.
- Some of the nodes (alternates) are removed from the donor's stack and added to the recipient's stack.
- If too little work is sent, recipient quickly becomes idle. Ideally, work is split into half.
- Strategies for splitting the stack
  - Send nodes near bottom of stack.
  - Send nodes near the cutoff depth.
  - Send half nodes between the bottom and cutoff depth.

Example of Work Splitting



Load Balancing Scheme

- When a processor becomes idle, and wants work, which one to contact?
- Asynchronous Round Robin
- Global Round Robin
- Random Polling: choose a random processor with equal probability

Asynchronous Round Robin

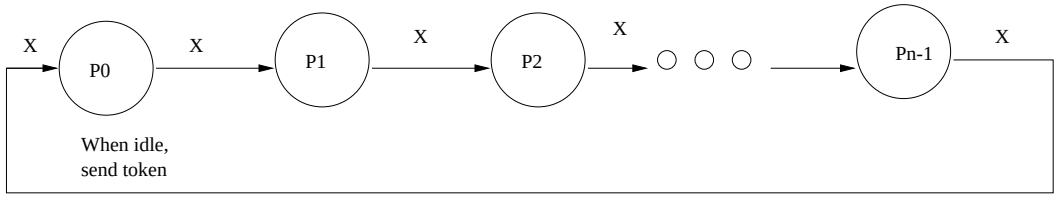
- Each processor maintains an independent variable *target*.
- When a processor runs out of work, it uses *target* as the label of a donor processor, and sends it a work request.
- The value of *target* is incremented (modulo p) each time a work request is sent.
- The initial value of *target* at each processor is set to *(label + 1) modulo p*, where *label* is local processor label.
- It is possible for two or more processors to request work from same processor.

Global Round Robin

- Use a single global variable called *target* stored at processor P_0.
- When a processor needs work, it requests and receives value of *target* from P_0.
- Then P_0 increments the value of *target* modulo p before responding to another request.
- GRR ensures that successive work request are distributed evenly across processors.
- Difficulty is in centralized bottleneck processor P_0.

Termination Detection

- Imagine the p processors to be connected in a ring, $P_0, P_1, ...P_{p-1}, P_0$.
- Processor P_0 initiates a token when it becomes idle, sends to P_1.
- Token is held in processor until that processor completes.
- Algorithm completes when P_0 receives token provided no work placed on idle processors.



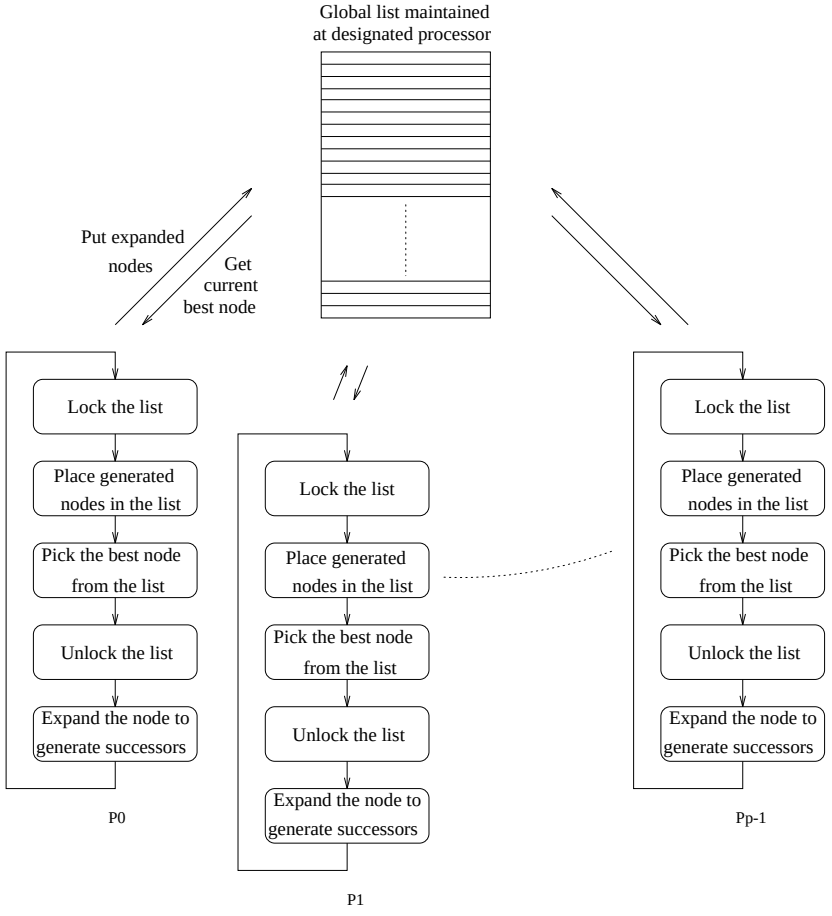
Algorithm for Termination Detection

- A processor can be in one of two states: *white* or *black*. They pass tokens among themselves of two forms: *white* or *black*.
- When processor P_0 becomes idle, it initiates termination detection by making itself white and sending a white token to P_1.
- If processor P_i sends work to processor P_j and $i > j$, then processor P_i becomes black.
- If processor P_i has the token and P_i is idle, then it passes the token to P_{i+1}. If P_i is black, then color of token sent is black. If P_i is white, token is passed unchanged.
- After P_i passes the token to P_{i+1}, P_i becomes white.
- Algorithm terminates when P_0 receives a white token.

Parallel Best First Search

- An important component of best-first search is the *open* list, which maintains the unexpanded nodes in search graph, ordered by their l-value.
- Given p processors, use a centralized strategy so that each processor can work on current best nodes in *open* list.
- Parallel formulation expands the first p nodes in the open list.

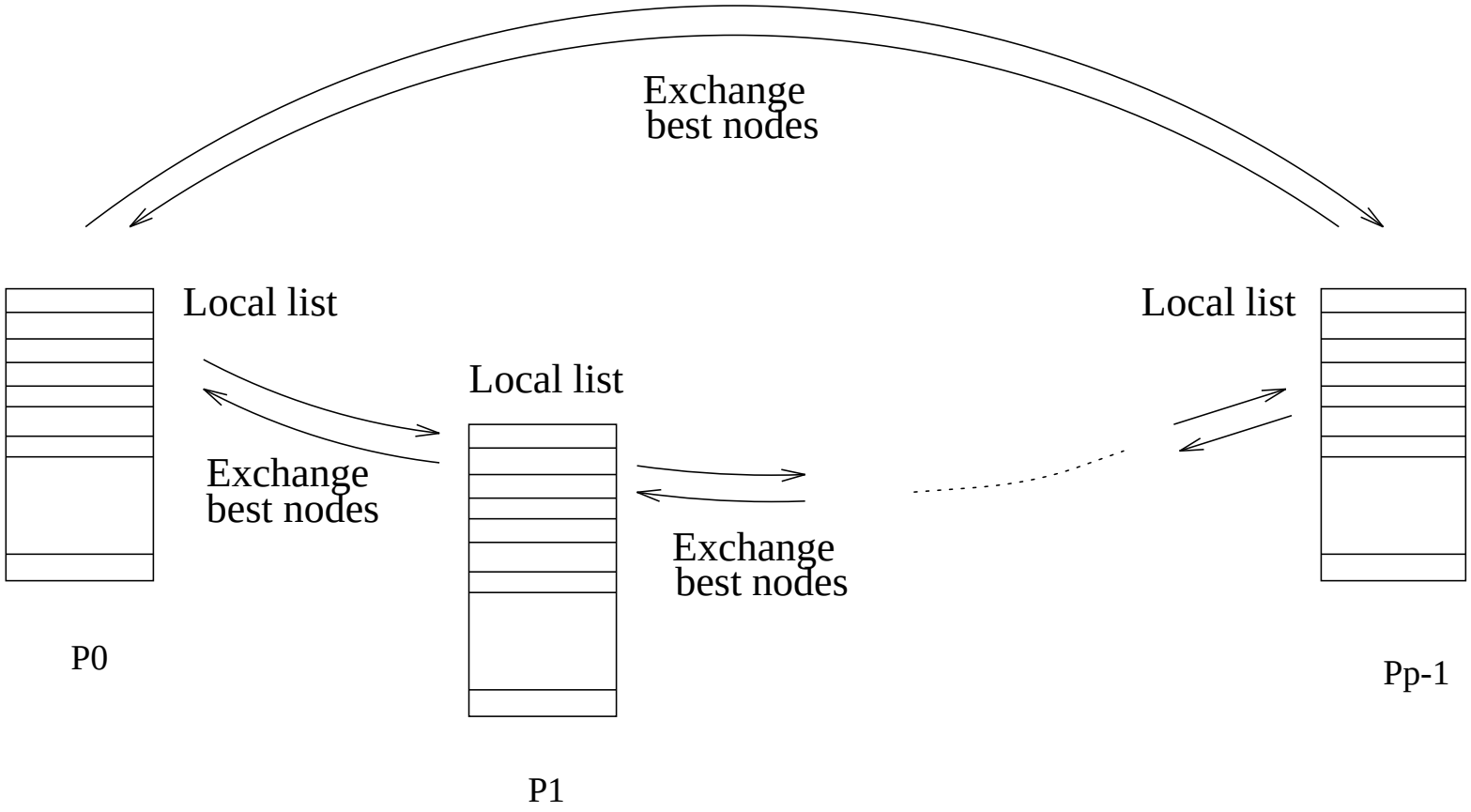
Centralized Scheme for Best First Search



Strategies for Message-Passing Machines

- Each processor has a local *open* list.
- Initially search space is statically divided among processors and expanding some nodes and distributing them to *local open* lists of various processors.
- All processors then select and expand nodes simulatenously.
- A communication strategy allows state-space nodes to be exchanged between *open* lists of different processors.

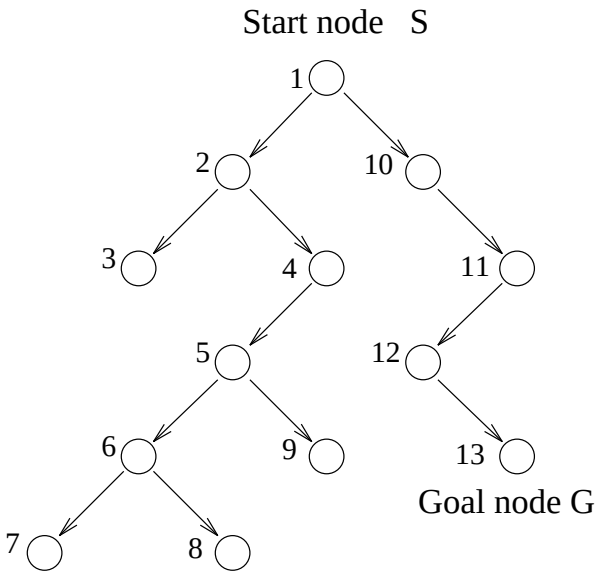
Example of Ring Communication Strategy



Speedup Anomalies in Parallel Search

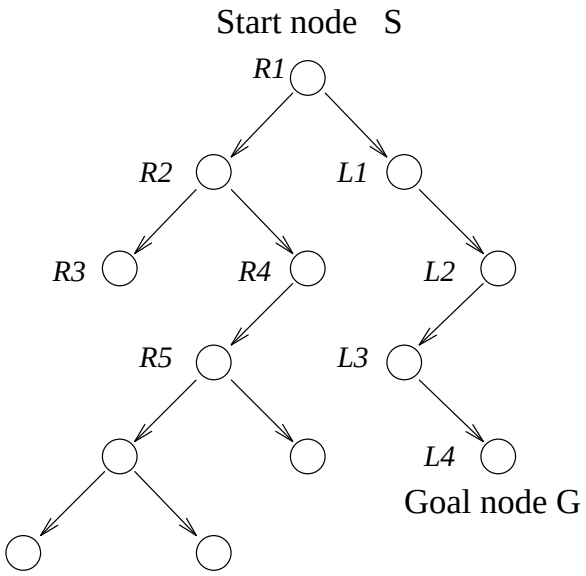
- In parallel search algorithms, speedup can greatly vary from one execution to another because the portion of search space examined by processors is determined dynamically.
- Serial search can expand a set of nodes to reach a goal node.
- Parallel search can expand more or less nodes to reach same goal node.
- Acceleration anomalies, superlinear speedups
- Deceleration anomalies, sublinear speedups.

Example of Less Nodes Searched in Parallel



Total number of nodes generated by sequential formulation = 13

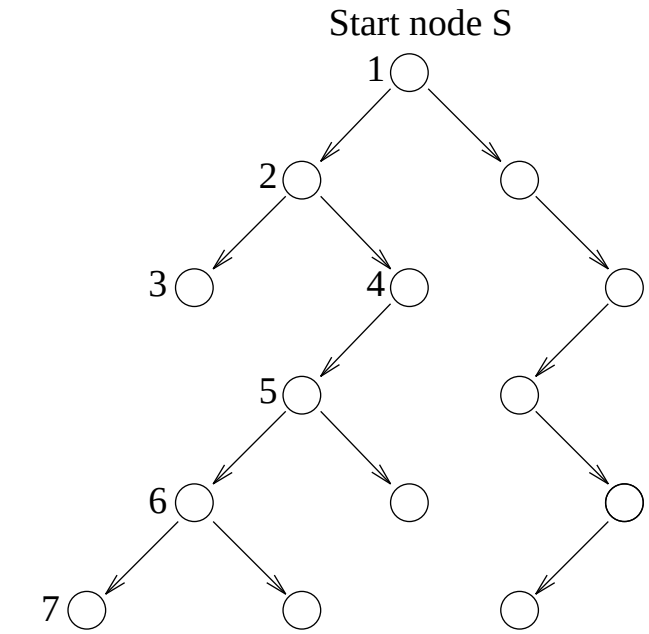
(a)



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

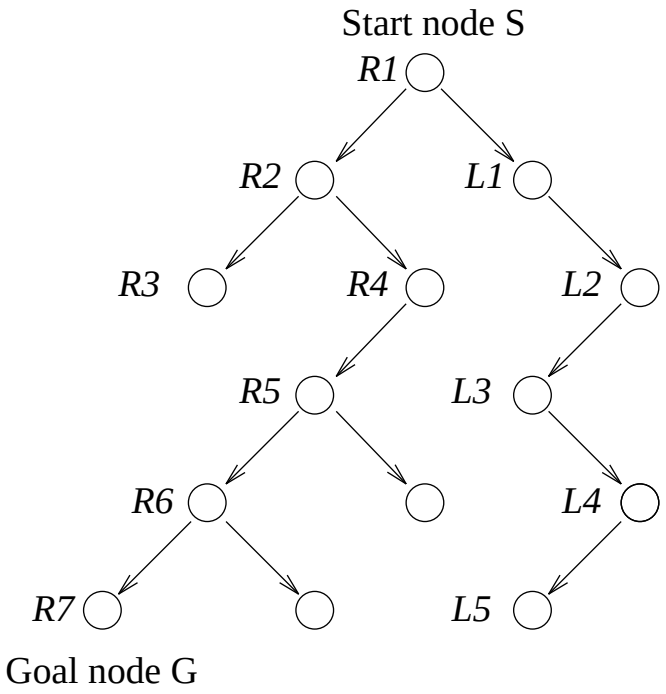
(b)

Example of More Nodes Searched in Parallel



Total number of nodes generated by sequential DFS = 7

(a)



Total number of nodes generated by two two-processor formulation of DFS = 12

(b)

Summary

- Background on Optimization Problems
- Sequential Search Algorithms
- Parallel Depth First Search
- Parallel Best First Search
- Speedup Anomalies in Parallel Search