Search Algorithms for Discrete Optimization Problems (Chapter 11)

Alexandre David
B2-206

Today

- Discrete optimization – basics.
- Sequential search algorithms.
- Parallel depth-first search.
- Parallel best-first search.
- Speedup anomalies.

Discrete Optimization Problems (DOP)

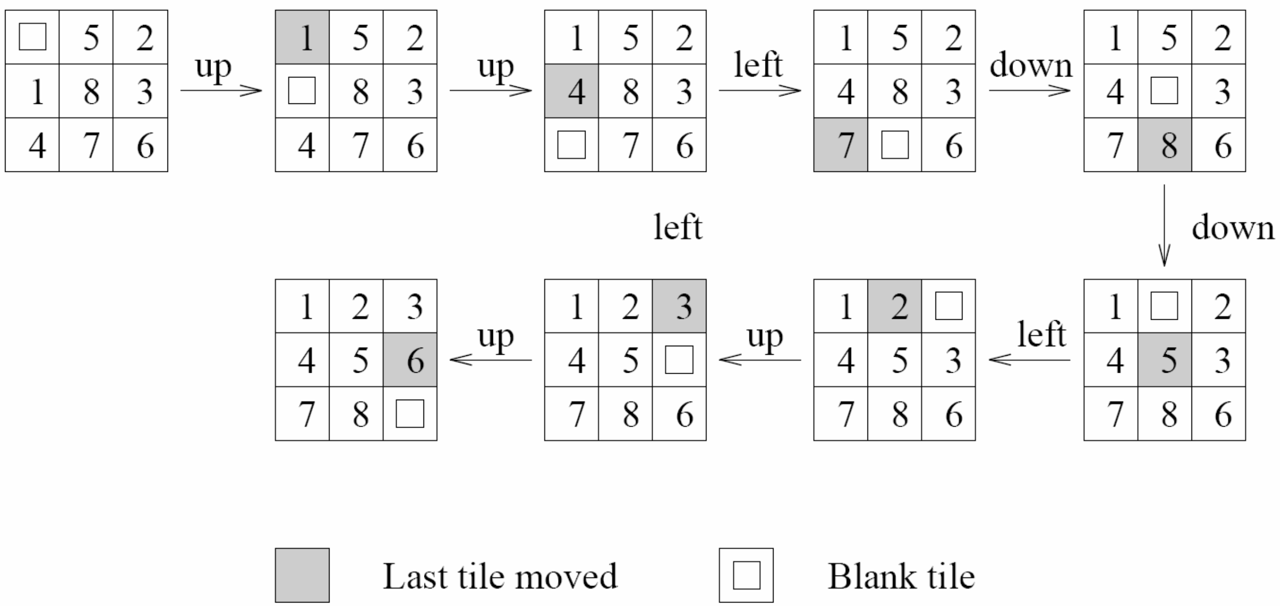
- Tuple (S, f) where
  - S is a finite (or countable) set of feasible solutions.
  - The function f is the cost $f: S \rightarrow R$.
- Objective: Find a solution $x_{opt} \in S$ s.t. $f(x_{opt}) \leq f(x)$ for all $x \in S$.
- Applications: Planning, scheduling, layout of VLSI chips, etc …

The 0/1 Integer-Linear-Programming Problem

- Input: an *m * m* matrix *A*, an *m * 1* vector *b*, and an *n * 1* vector *c*.
- Find vector $\overline{x}$ of 0/1 s.t.
  - The constraint $A\overline{x} \geq b$ is satisfied.
  - The function $f(\overline{x}) = c^T\overline{x}$ is minimized.

The 8-Puzzle Problem

S = All paths from initial
to final configurations.
Function *f* =number of moves.



DOP

- The feasible space S is typically very large.
- Reformulate a DOP as the problem of finding the minimum cost-path from an initial node to goal node(s).
- S contains paths.
- The graph is called the state-space, the nodes are called states.
- Often, f=sum of the edge costs.

0/1 Integer-Linear-Programming Problem Revisited

$$A=\begin{bmatrix} 5 & 2 & 1 & 2 \\ 1 & -1 & -1 & 2 \\ 3 & 1 & 1 & 3 \end{bmatrix} \quad b=\begin{bmatrix} 8 \\ 2 \\ 5 \end{bmatrix} \quad c=\begin{bmatrix} 2 \\ 1 \\ -1 \\ -2 \end{bmatrix}$$

→
$$5x_1 + 2x_2 + x_3 + 2x_4 \geq 8$$
$$x_1 - x_2 - x_3 + 2x_4 \geq 2$$
$$3x_1 + x_2 + x_3 + 3x_4 \geq 5$$

Constraints

→
$$f(x) = 2x_1 + x_2 - x_3 - 2x_4$$

Cost

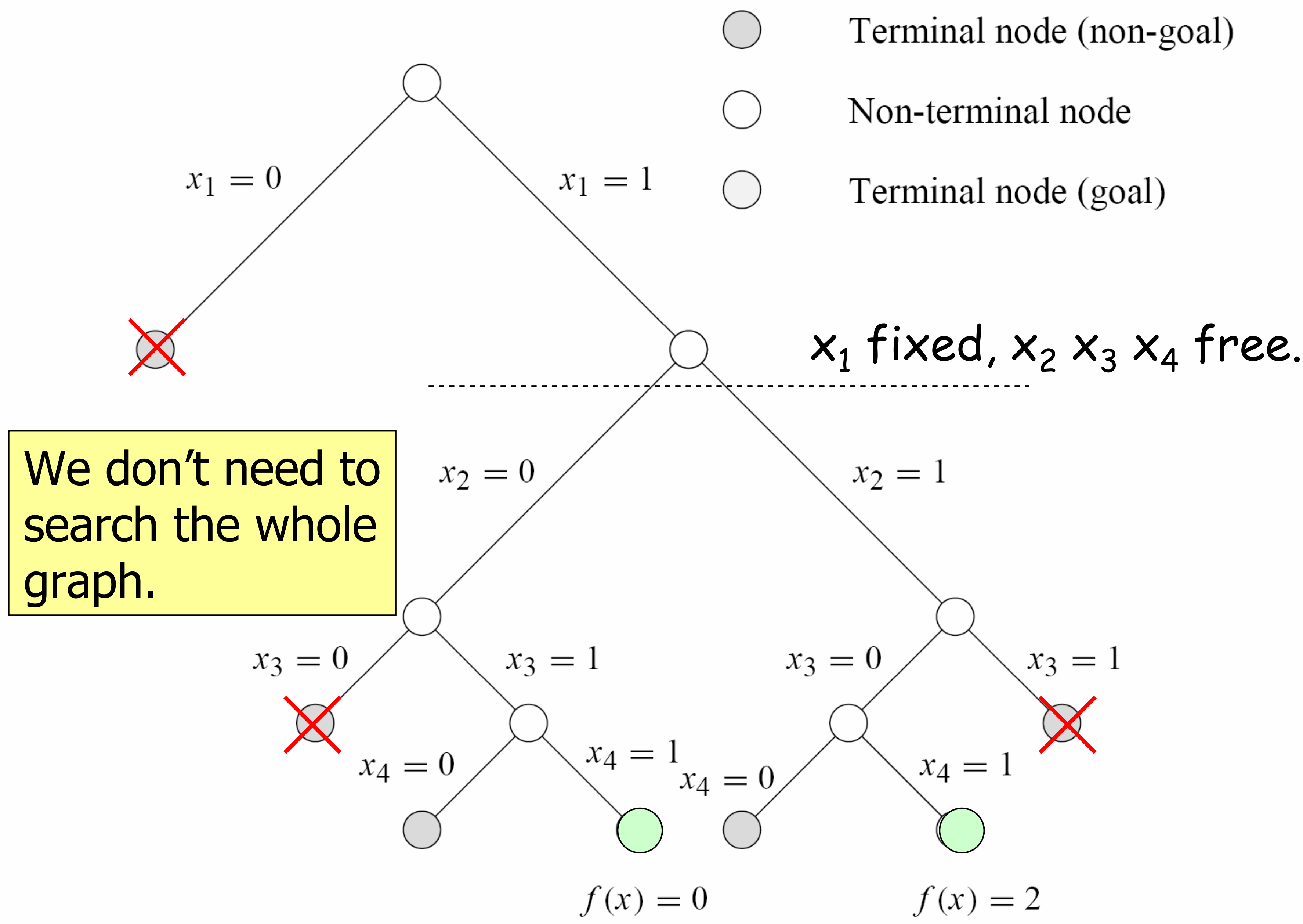


Figure 11.2 The graph corresponding to the 0/1 integer-linear-programming problem.

Heuristics

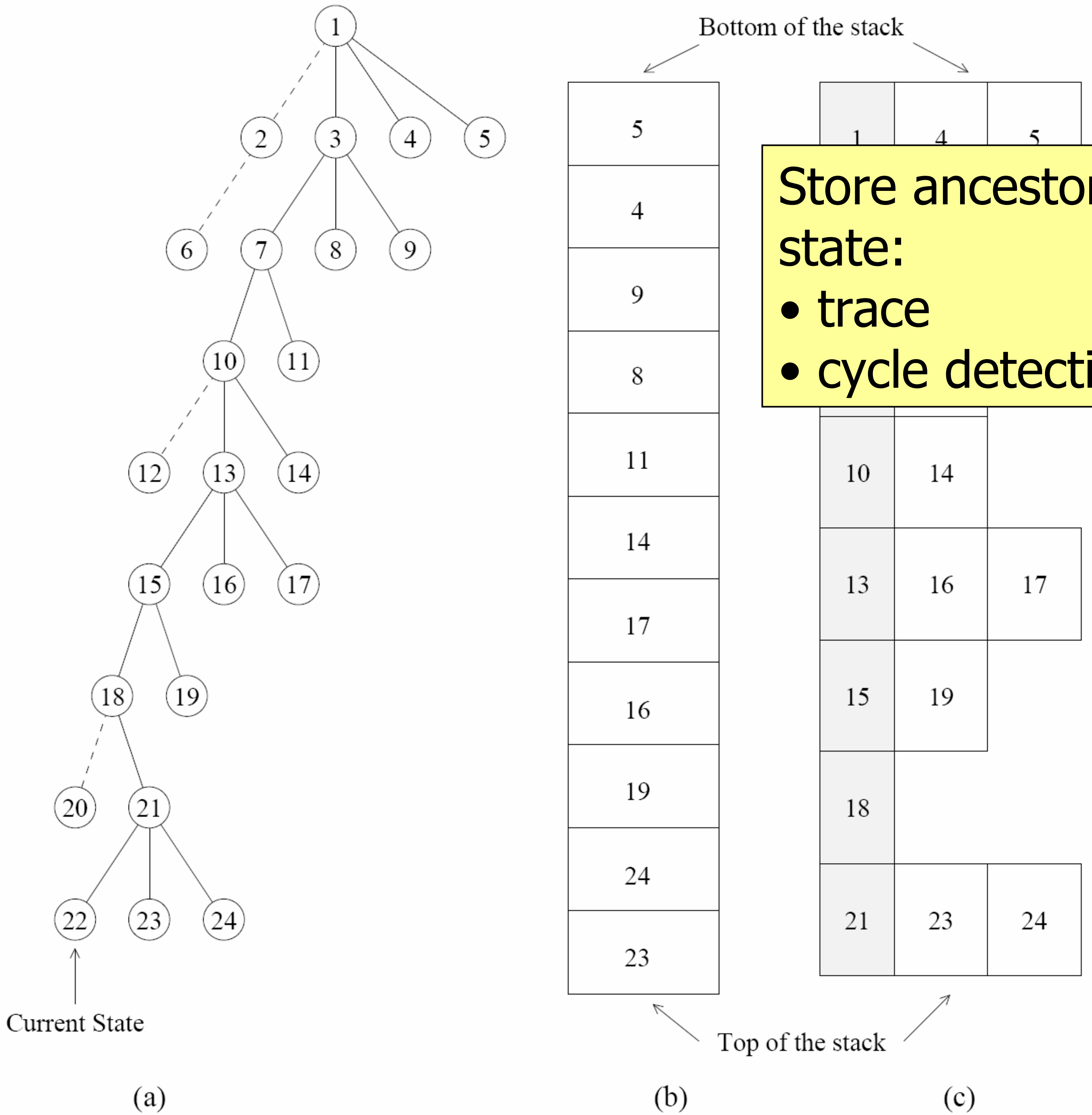
- Often possible to estimate the cost to reach goal states from an intermediate state.
  - Heuristic estimate.
  - If the heuristic is guaranteed to be a lower bound on the cost then it is an admissible heuristic.
  - Good for pruning the search.
- 8-puzzle problem: Manhattan distance.

Sequential Search Algorithms

- Trees: Each successor leads to an unexplored state.
- (General) Graphs: States reachable by several paths → check explored states.
- Depth-first search (trees) – storage linear in function of the depth.
- Depth-first branch-and-bound.
- Iterative deepening DFS, A*

Avoid being stuck in a branch.

DFS



Store ancestor state:
- trace
- cycle detection.

Best First Search

- 2 lists:
  - States to be explored on the open list. waiting
  - States explored on the closed list. passed
  - Choose best from open list, replace if find better states – more memory.
- A* algorithm:
  - l(x)=g(x)+h(x) used to order the search.
  - g(x): from init to x.
  - h(x): from x to goal.

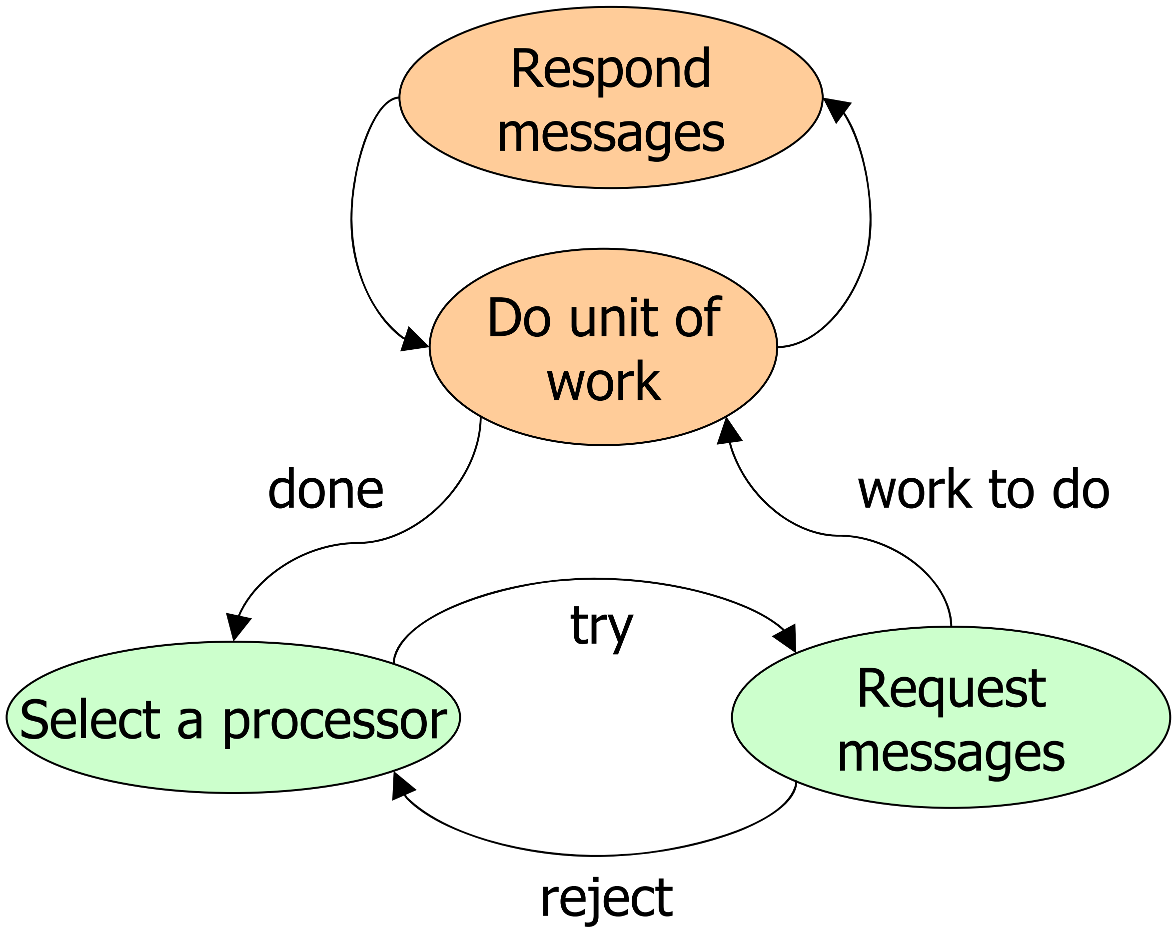
Sequential vs. Parallel Search

- Overhead for parallel search (as usual communication, contention, load imbalance).
- Big difference with other algorithms: Amount of work can be very different because different parts of the search space are explored.
  - Super-linear anomalies.
  - Critical issue: Distribution of the search space.

Parallel DFS

- Static partitioning: Assign a processor per branch from the root: Load imbalance.
- Dynamic partitioning: Idle processors request work from busy ones.
  - Assume the search is done on disjoint parts of the search space – otherwise duplicate work.
  - Local stack of states to explore.
  - Recipient/donor; see worker model.

Generic Scheme for Load Balancing



Work Splitting

- Work-splitting strategies:
  - Send nodes near bottom of the stack (root).
  - Send nodes near end.
  - Send some nodes from each level (stack splitting).
- Half-split: ½ of the stack split – difficult to estimate the size of the sub-trees.
- Do not send nodes beyond the cutoff depth. *Why?*

Load Balancing

- Which processor to ask?
  - Asynchronous Round Robin.
    - Ask to (local_target++)%p.
    - + asynchronous, - even work.
  - Global Round Robin.
    - Ask to (global_target++)%p.
    - - contention, + even work.
  - Random Polling.
    - + + ?

Analysis

- How to analyze?
- What’s W? W_P?
- Problem:
  - The execution time depends on the search primarily (and secondarily on the size of the input).

Analysis

- Compute overhead T_0 (as usual) from communication, idling, contention, and termination detection.
- In addition the search overhead may add another term (W_P/W). Assume = 1.
- Distinguish executed search and algorithm.
- Problem: Dynamic communication schemes, difficult to derive an exact expression.

Analysis

- Get an upper-bound, i.e., worst case.
- Assume
  - Work can be partitioned as long as > ε.
  - A reasonable work-splitting is available.
    α-splitting: Both partitions of a work w have at least αw work.
- Quantify the number of (work) requests.

Analysis

- Donor has $w_i \rightarrow w_j + w_k$.
- Assumption: $w_j > \alpha w_i$, $w_k > \alpha w_i$.
- After transfer, donor and recipient have $\leq (1-\alpha)w_i$.
- $w_0,\ldots,w_{p-1} \leq w$. Split all (2p pieces), largest $\leq (1-\alpha)w$.
- If every processor gets a request once, then each piece has been split once $\Rightarrow$ maximum load reduced by $(1-\alpha)$ at any processor.

Analysis

- Load balancing in the term V(p): After every V(p) requests, each processor receives at least one request.
- After every V(p) requests, the maximum work decreases by at least $(1-\alpha)$.
  - i*V(p) requests $\rightarrow$ remaining work $\leq (1-\alpha)^i W$.
  - To have remaining work $\leq \varepsilon$, the number of requests is $O(V(p)\log W)$.
  - $\Rightarrow T_0 = t_{comm} V(p) \log W$.

Computation of V(p)

- Asynchronous round robin: Worst case when p-1 processors request the same processor, but they all get it wrong.
  - 0 asks to 1, 2, 3… and finally p-1.
  - Same for all p-1 processes $\Rightarrow$ $V(p)=O(p^2)$.
- Global round robin: One sequence for all processor. V(p)=p.
- Random: Compute average in $O(p \log p)$.

Analysis (cont.)

- We want the isoefficiency function $W=KT_0$.
  - We have $T_0=O(V(p)\log W)$.
  - We have $V(p)$ for different load balancing schemes.
  - $\Rightarrow$ solve $W=f(p)$.
- Take contention into account for global round robin $\rightarrow O(p^2 \log p)$, and for random $O(p \log^2 p)$.

Analysis

- Asynchronous round robin: Poor performance because of its large number of work requests.
- Global round robin: Poor performance because of contention at counter, even with its least number of requests.
- Random polling: Desirable compromise.

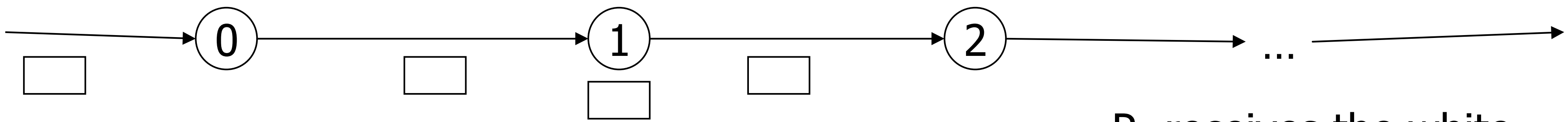
Termination Detection

- Normally simple token based algorithm works but not here. When a processor goes idle, it may receive more work later.
- Dijkstra’s token algorithm.
- Tree-based algorithm.

Dijsktra’s Token Termination Detection Algorithm

P_0 idle initiates algorithm.

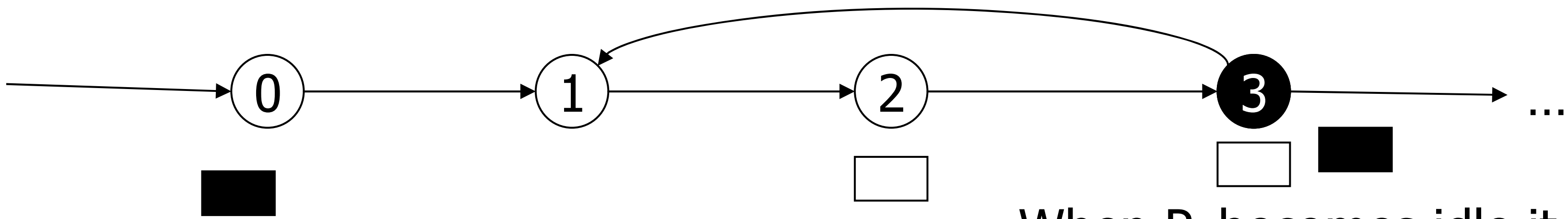
P_i idle has token: pass it.



It sends a white token.

P_0 receives the white token and is idle: stop.

P_j (not idle) sends work to P_i, j>i: P_j becomes black.



When P_j becomes idle it passes a black token and becomes white again.

P_0 receives a black token: retry.

Tree-Based Termination Detection

- Weight 1 from the root at the start.
- Weights are divided and go down the tree with the work.
- When work is done, weights are returned from the source.
- Terminate when weight is one at the root.
- Careful with precision.

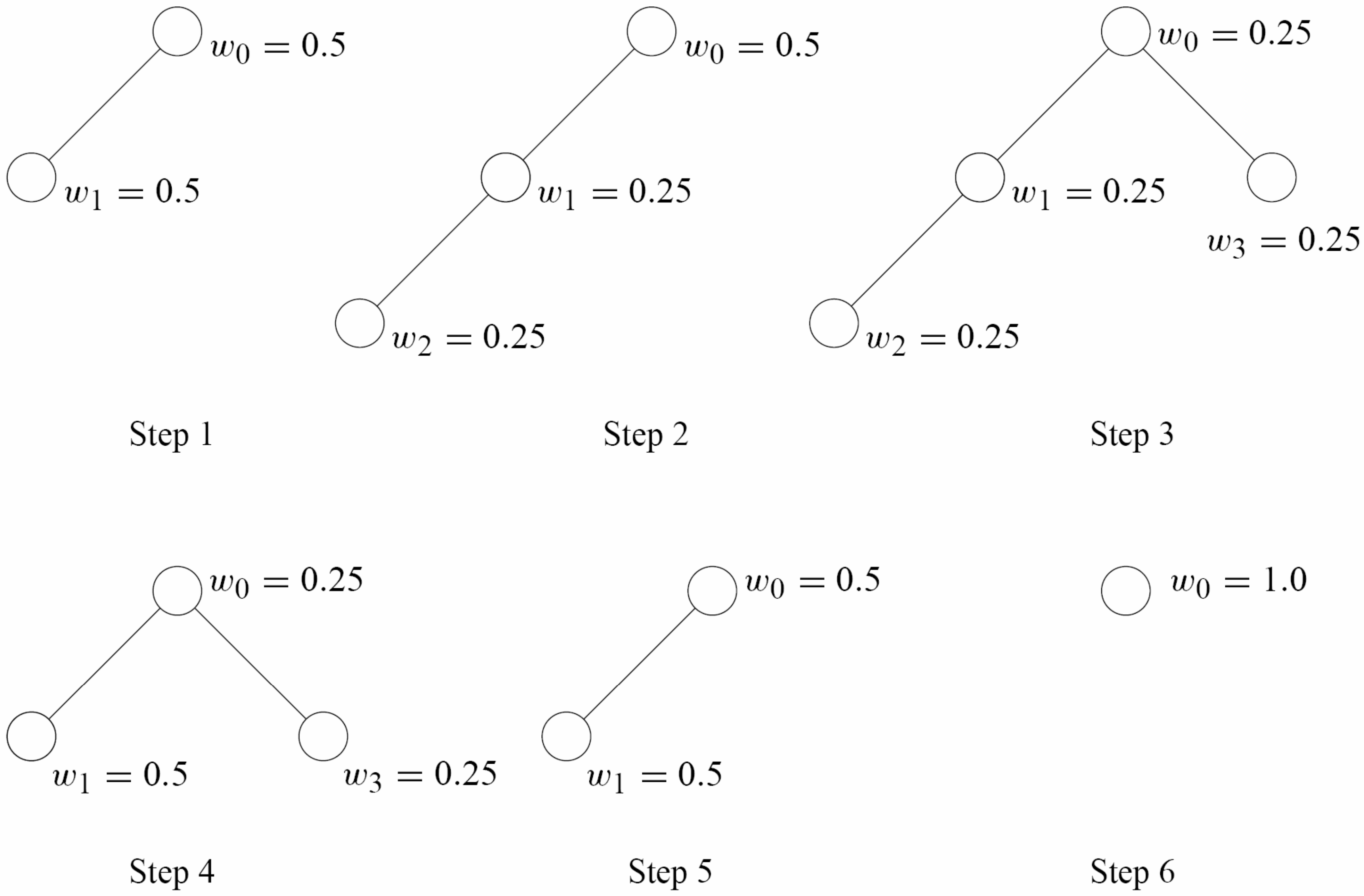


Figure 11.10 Tree-based termination detection. Steps 1–6 illustrate the weights at various processors after each work transfer.

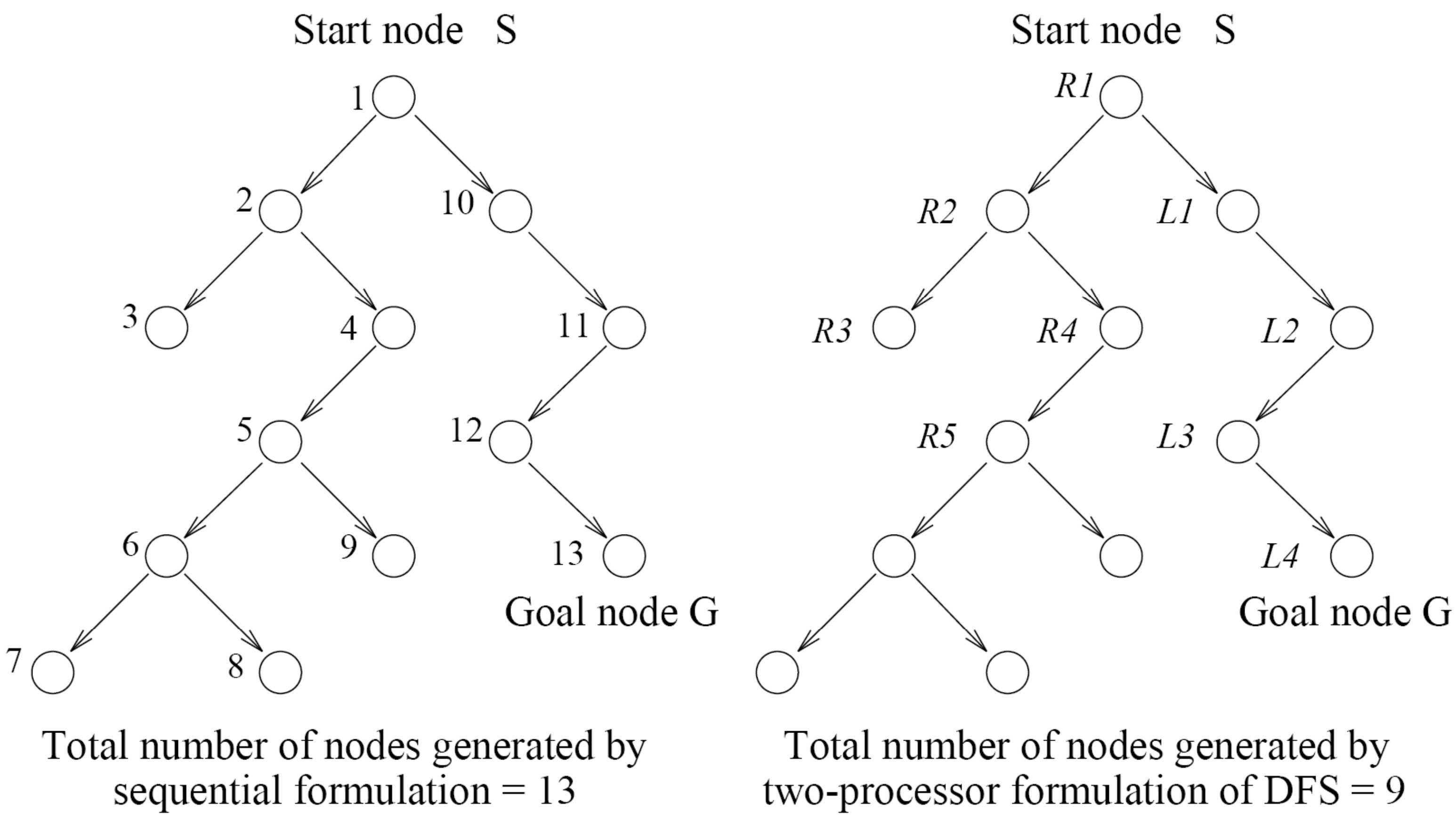
Experiments

Analysis validated by experimental results.
It works. ☺

Parallel Best-First Search

- Avoid bottleneck with one global open list.
- Local open lists must synchronize and share their best nodes.
  - Different communication schemes.
- Distributed cycle detection: Hash nodes to map them on specific processors (local check) but degrades performance.

Acceleration Anomalies



(a) (b)

Deceleration Anomalies