

# DISCRETE OPTIMIZATION PROBLEMS

(S, f)  $f: S \rightarrow R$ . Objective: find  $x_{opt} \mid f(x_{opt}) \leq f(x)$   
 ↘ feasible solution set  $x, x_{opt} \in S$

$g(x)$  = cost of reaching state  $x$  from initial state  $s$  along current path  
 $h(x)$  = heuristic estimate of reaching goal state from state  $x$   
 $l(x) = g(x) + \underline{h(x)}$

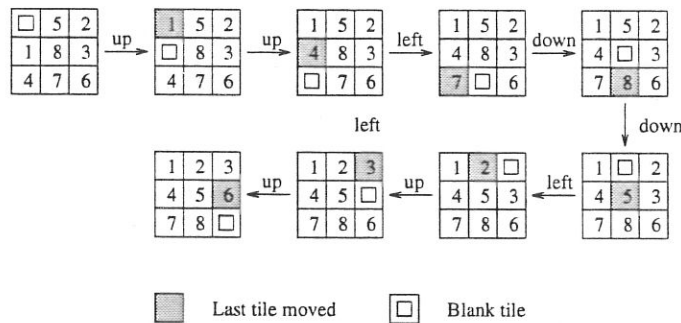
$\sum_{tile=1}^n$  (Manhattan distance between initial and final position) <sub>tile</sub>

5	2	
1	8	3
4	7	6

(a)

1	2	3
4	5	6
7	8	□

(b)



(c)

**Figure 8.1** An 8-puzzle problem instance: (a) initial configuration; (b) final configuration; and (c) a sequence of moves leading from the initial to the final configuration.  
 Copyright (r) 1994 Benjamin/Cummings Publishing Co.

Discrete optimization problems: NP-hard

Avg. time complexity of heuristic search algorithms: often polynomial

$$\begin{aligned} \text{eg } 5x_1 + 2x_2 + x_3 + 2x_4 &\geq 8 \\ x_1 - x_2 - x_3 + 2x_4 &\geq 2 \\ 3x_1 + x_2 + x_3 + 3x_4 &\geq 5 \end{aligned}$$

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

minimize

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

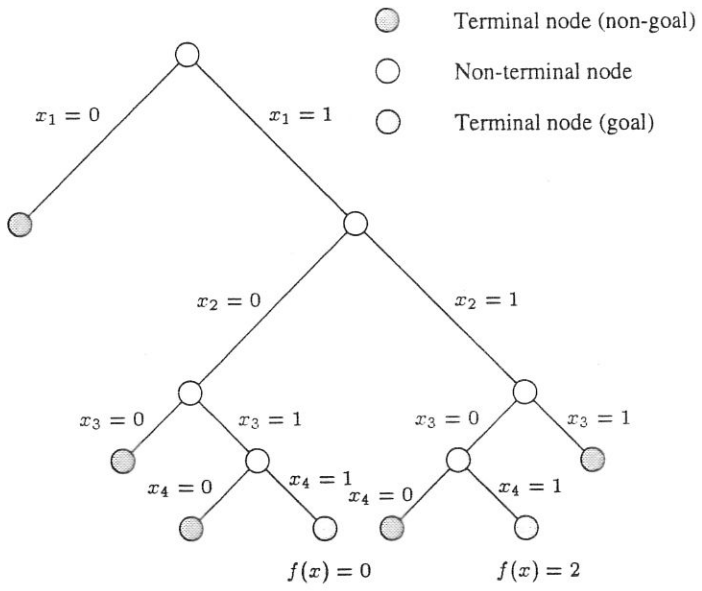
$$A\bar{x} \geq b$$

minimize  $f(\bar{x}) = c^T \bar{x}$

(find  $n \times 1$  vector  $\bar{x}$ )

• free variables, fixed variables

→ unnecessary to generate the entire feasible set — can restrict search



**Figure 8.2** The graph corresponding to the 0/1 integer-linear-programming problem.  
Copyright (r) 1994 Benjamin/Cummings Publishing Co.

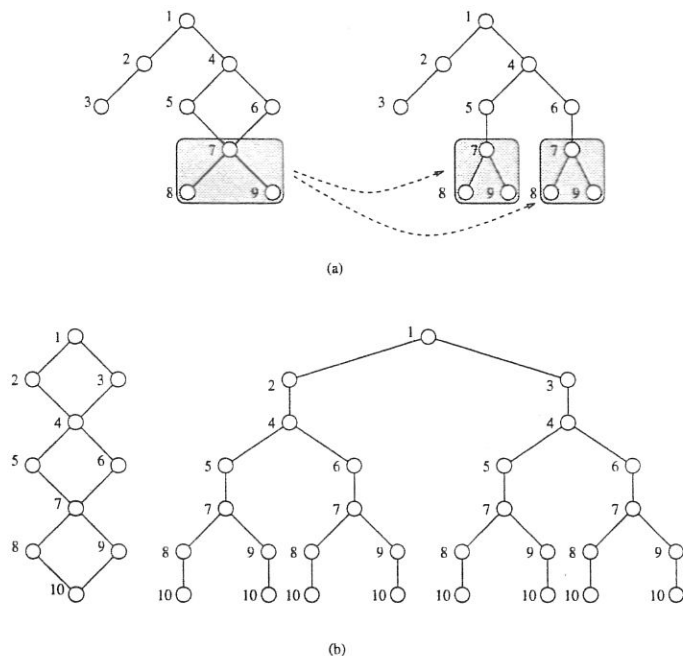
After partial instantiation, check for feasibility:

$$\sum_{x_j \text{ is free}} \max\{A[i,j], 0\} + \sum_{x_j \text{ is fixed}} A[i,j]x_j \geq b_i \quad (i=1,2,\dots,m)$$

— search space can be a graph (eg 8-puzzle) or a tree (eg 0/1 LP)  
 if graph, must check if the state has already been generated

## DEPTH-FIRST SEARCH ALGORITHMS (for tree-search formulations)

(+) storage requirement is linear in the depth of search space



**Figure 8.3** Two examples of unfolding a graph into a tree.  
 Copyright (r) 1994 Benjamin/Cummings Publishing Co.

(1) simple backtracking — finds first soln, may not be optimal  
 no heuristics to order successors of an expanded node

(2) Depth-first branch & bound (DFBB)

- searches state space to find optimal solution
- keeps updated current best solution path
- discard inferior partial solution paths

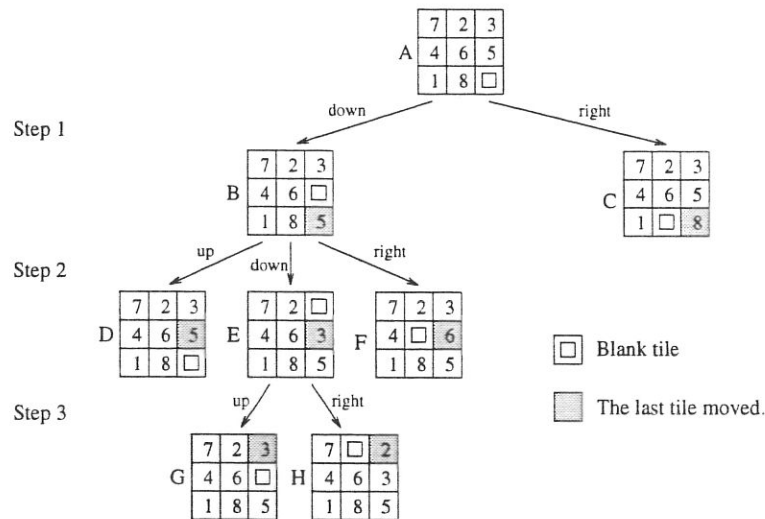
(3) Iterative deepening depth-first search (ID-DFS)

→ finds soln with fewest edges, may not be least cost

(3') Iterative deepening A\* (IDA\*) uses f-values to bound depth; repeatedly performs cost-bounded DFS



- In each step untried alternatives must be stored
  - In 8-puzzle, up to 3 untried alternatives are stored at each step
  - $\Theta(md)$ , where  $d$  = max depth &  $m$  = storage



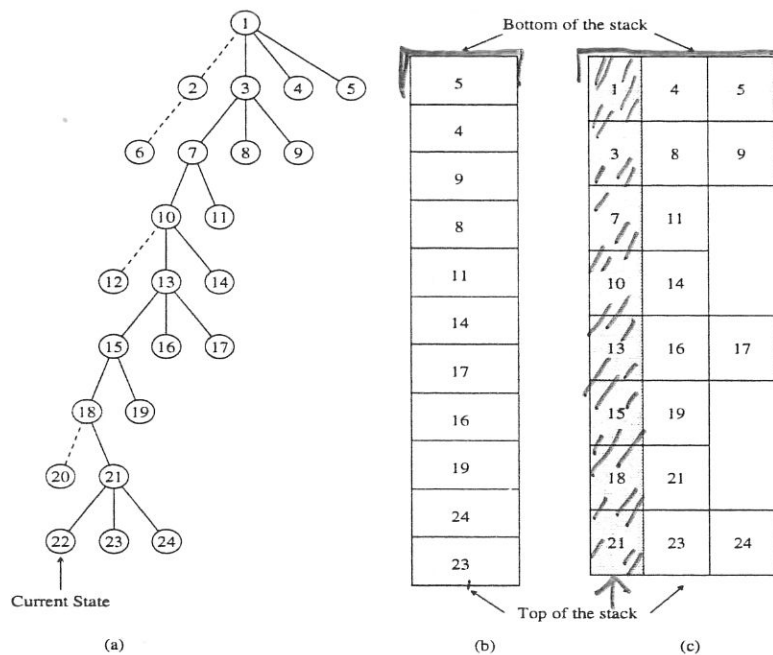
**Figure 8.4** States resulting from the first three steps of depth-first search applied to an instance of the 8-puzzle.  
Copyright (r) 1994 Benjamin/Cummings Publishing Co.

storing untried alternatives

→ without parent state (b), or with parent state (c)

→ representation (c) necessary if

- seq of Xformations is needed as part of solution, or
- state space is a graph & ancestor states could be regenerated (need to check for duplication & remove cycles)



**Figure 8.5** Representing a DFS tree: (a) the DFS tree; Successor nodes shown with dashed lines have already been explored; (b) the stack storing untried alternatives only; and (c) the stack storing untried alternatives along with their parent. The shaded blocks represent the parent state and the block to the right represents successor states that have not been explored.

Copyright (r) 1994 Benjamin/Cummings Publishing Co.

## BEST-FIRST SEARCH ALGORITHMS

→ for both trees & graphs

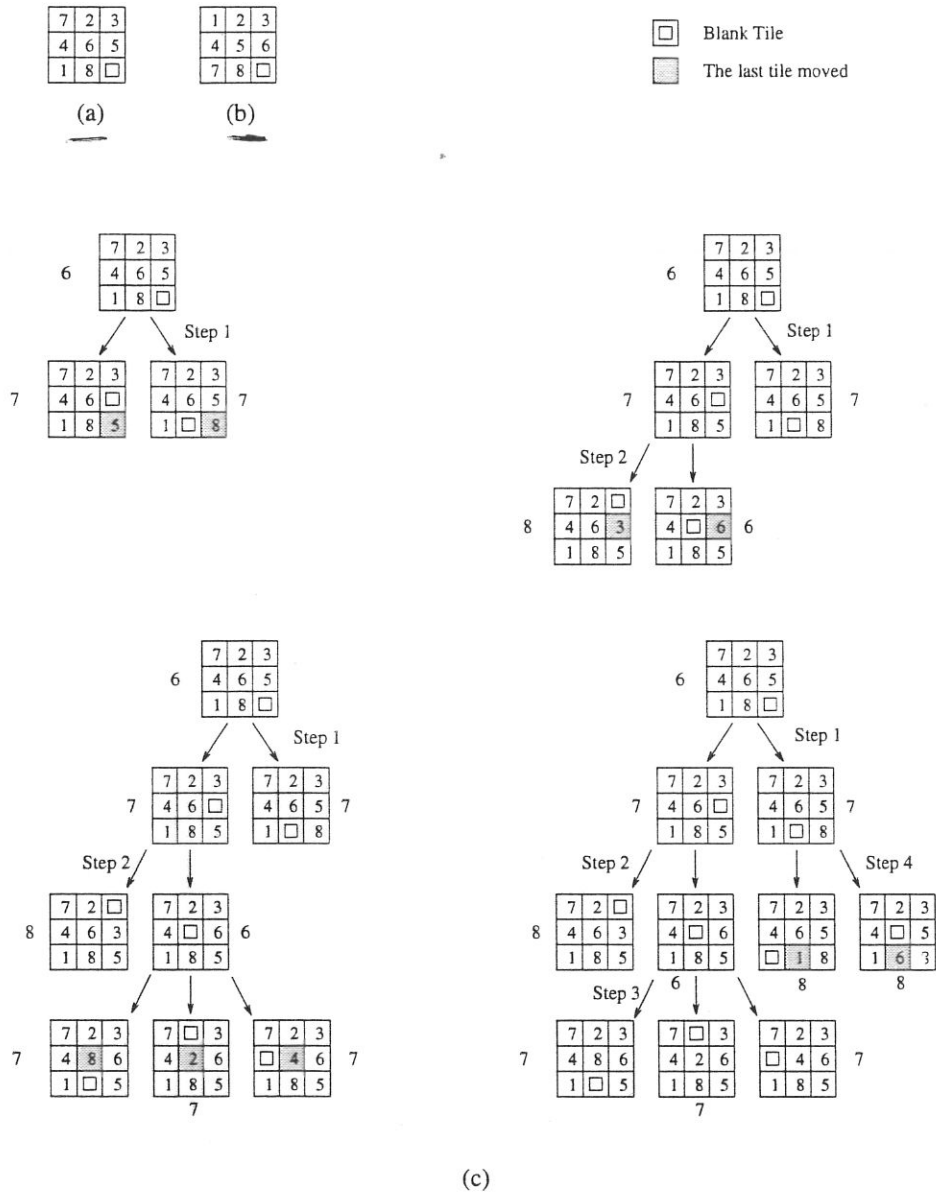
→ use heuristics to direct search to portions of search space likely to yield solutions

→ smaller heuristic values to more promising nodes

→ open & closed lists; open list is kept sorted as per heuristic values;

→ in each step, most promising node from the open list is removed & explored

→ The BFS technique A\* uses the lower bound func  $l$  as heuristic  
 → drawback of BFS: memory requirement is linear in size of the search space explored

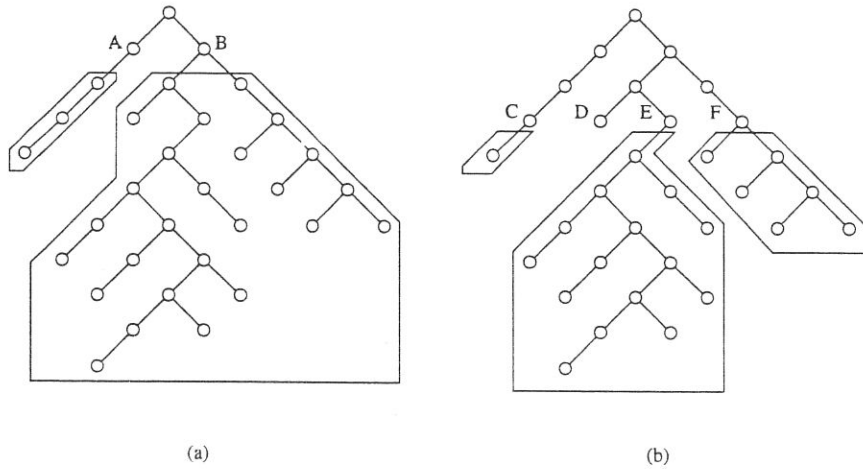


**Figure 8.6** Applying best-first search to the 8-puzzle: (a) initial configuration; (b) final configuration; and (c) states resulting from the first four steps of best-first search. Each state is labeled with its  $h$ -value (that is, the Manhattan distance from the state to the final state).

Copyright (r) 1994 Benjamin/Cummings Publishing Co.

## PARALLEL DEPTH-FIRST SEARCH

- Assume time for node expansion = 1  $\Rightarrow T_s \approx W$  (problem size)
- distribute search space dynamically  $\Rightarrow$  dynamic load balancing needed
- each processor has local stack for DFS
- active/idle state
- find solution & broadcast
- termination detection: detect whether processors have become idle without finding a solution
- 2 main issues: how to split work, & which donez processors to ask when I become idle



**Figure 8.7** The unstructured nature of tree search and the imbalance resulting from static partitioning.  
Copyright (r) 1994 Benjamin/Cummings Publishing Co.

## Work-splitting strategies

- cutoff depth

- 1) send nodes near bottom of stack
- 2) send nodes near cutoff depth → if strong heuristic, helps to cut search space
- 3) send half the nodes between bottom of stack & cutoff depth

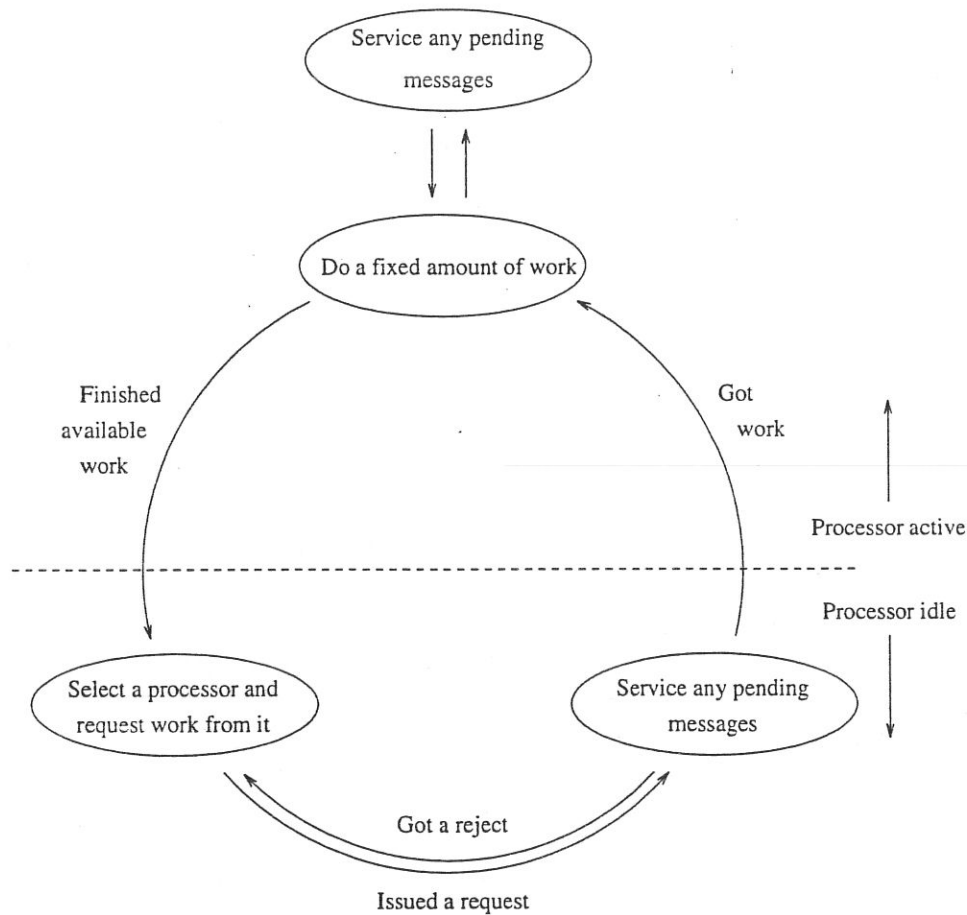


Figure 8.8 A generic scheme for dynamic load balancing.  
Copyright (r) 1994 Benjamin/Cummings Publishing Co.

## Dynamic Load-balancing schemes

1) Asynchronous RR:  $target_i := (myid + 1) \bmod n$

-  $target_i$  increased (mod  $n$ ) for each request

- Each processor generates requests independently

2) Global RR: global var 'target' at  $P_0$

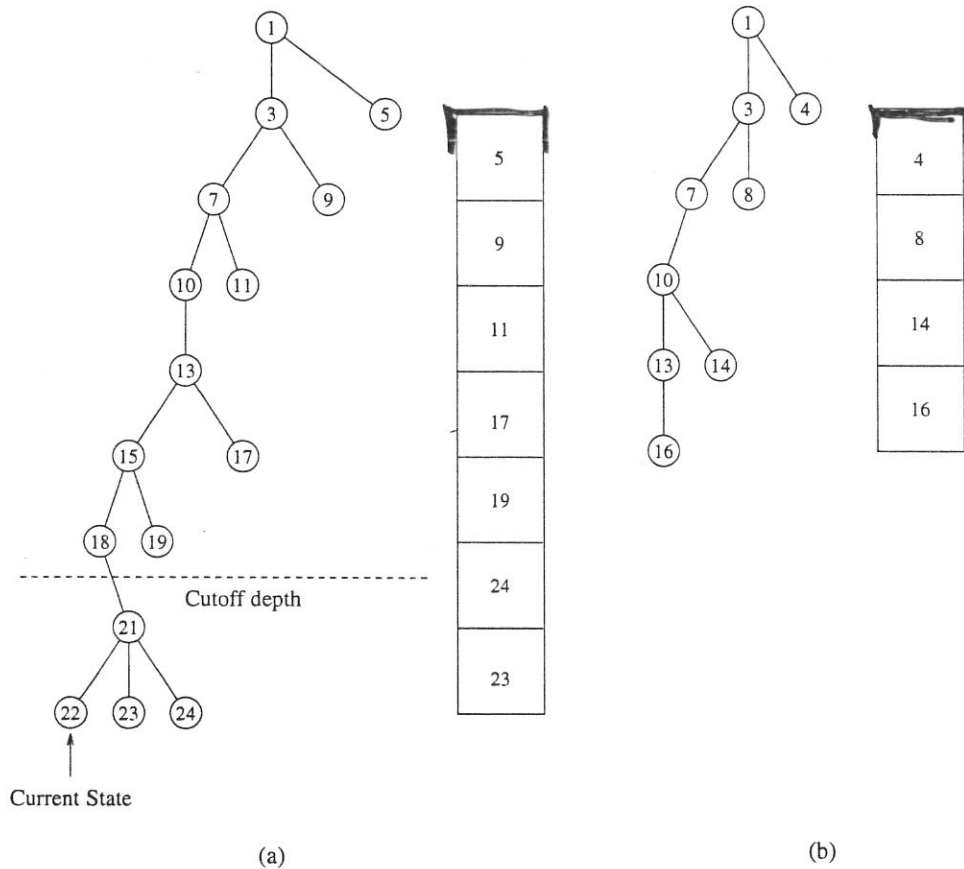
(+) successive work requests are distributed evenly (-) contention at  $P_0$

3) Random polling: randomly poll/select a donor



To analyse performance & scalability of parallel DFS, compute overhead  $T_0$

- 1) communication
- 2) idle time (subsumed by (1))
- 3) termination detection,
- 4) contention for shared resources



**Figure 8.9** Splitting the DFS tree in Figure 8.5. The two subtrees along with their stack representations are shown in (a) and (b).  
Copyright (r) 1994 Benjamin/Cummings Publishing Co.

Technique to get upper bound on communication overhead assumes

- 1) work can be partitioned as long as its size  $>$  threshold  $\in$
- 2)  $\alpha$ -splitting.  $\alpha$  is a l.b. on the load imbalance that results from work splitting. Both partitions of  $w$  have at least  $\alpha w$  work, i.e.,  

$$\psi w > \alpha w \quad \& \quad (1 - \psi) w > \alpha w$$

$V(p) \equiv$  after every  $V(p)$  work requests, each processor has received at least one work request

After  $V(p)$  req, max work at any processor  $< (1-\alpha)W$   
 $2V(p) < (1-\alpha)^2 W$   
 $kV(p) < (1-\alpha)^k W$

$$(1-\alpha)^k W = \epsilon \Rightarrow \left(\frac{W}{\epsilon}\right) = \left(\frac{1}{1-\alpha}\right)^k$$

After  $\left(\log_{1/(1-\alpha)} (W/\epsilon)\right) V(p)$  requests,  $\text{---} < \epsilon$

$\Rightarrow$  total number of work requests is  $O(V(p) \log W)$

$$T_0 = t_{\text{comm}} V(p) \log W \quad \& \quad \epsilon = \frac{1}{1 + T_0/W} = \frac{1}{1 + (t_{\text{comm}} V(p) \log W)/W}$$

Obj: derive isoefficiency func. for each load-balancing scheme & on different parallel archs determines  $V(p)$   
 determines  $t_{\text{comm}}$

$\rightarrow$  computing  $V(p)$  for various load-balancing schemes

1) Asynch RR:  $V(p) = O(p^2)$

2) Global RR:  $V(p) = p$

3) Random polling: worst-case of  $V(p)$  is unbounded  $\therefore$  find avg.-case

•  $F(i, p) \equiv$  state in which  $i$  of the  $p$  boxes have been marked

•  $f(i, p) \equiv$  avg. # trials needed to go  $F(i, p) \rightarrow F(p, p)$

$$V(p) = f(0, p) \text{ i.e. } F(0, p) \xrightarrow{\# \text{ hops}} F(p, p)$$

$$* f(i, p) = \frac{i}{p} (1 + f(i, p)) + \frac{p-i}{p} (1 + f(i+1, p))$$

$$\frac{p-i}{p} f(i, p) = 1 + \frac{p-i}{p} f(i+1, p)$$

$$\rightarrow f(i, p) = \frac{p}{p-i} + f(i+1, p)$$

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

$$= p \sum_{i=1}^p \frac{1}{i} = p \times H_p, \text{ where } H_p \text{ is a harmonic number}$$

As  $p$  becomes large,  $H_p \approx 1.69 \ln p \therefore V(p) = \Theta(p \log p)$

## Analysis of Load-balancing Schemes for HCs

→ asymptotic value of  $t_{\text{comm}} = \Theta(\log p)$

→  $T_0 = O(V(p) \log p \log W)$ . Balance this ovhd. with problem size  $W$  to derive isoefficiency function

### 1) Async RR

$$T_0 = O(p^2 \log p \log W)$$

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

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

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

$$\approx O(p^2 \log^2 p) \text{ isoefficiency function due to communication}$$

### 2) Global RR: $T_0 = O(p \log p \log W)$

isoefficiency func due to communication is  $O(p \log^2 p)$

isoefficiency func due to contention = ?

— # times global var 'target' is accessed = # work requests  $O(p \log W)$

— Execution time =  $\Theta(W/p)$

— crossover point requires  $\uparrow W$  so as to keep ratio between  $W/p$  and  $O(p \log W)$  constant

$$\frac{W}{p} = O(p \log W). \text{ On simplification, isoefficiency} = \underline{O(p^2 \log p)}$$

### 3) Random polling: $T_0 = O(p \log^2 p \log W)$

Equating  $T_0$  with problem size  $W$  and simplifying as before, isoefficiency function is  $O(p \log^3 p)$ .

## EFFECTS OF M/C characteristics on the Isoefficiency Fn.

although  $t_{\text{comm}} = \Theta(\log p)$  was assumed on HC, in  $t_s + mt_w + t_h \log p$ ,  $(t_s + mt_w)$  constant dominates  $t_h \log p \therefore t_{\text{comm}} = \Theta(1)$

⇒ For RP & ARR, isoefficiency fun (due to comm) goes  $\downarrow$  by  $(\log p)$   
 $\therefore RP \equiv O(p \log^2 p)$  &  $ARR = O(p^2 \log p)$

For GRR, dominant isoefficiency term is due to contention  
⇒ not affected by the M/C characteristic

## Analysis of Load-balancing Schemes for NOW (Ethernet)

→ delivery time is constant  $\Rightarrow t_{\text{comm}} = \Theta(1)$

→  $T_0 = O(V(p) \log W)$

1) Asynch RR  $T_0 = (p^2 \log W)$

→ Equating with problem size  $W$  & simplify as before,

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

→ For isoefficiency<sub>contention</sub>, use analysis used for analyzing contention to 'target' in GRR on HC i.e.  $\frac{O(V(p) \log W)}{\Theta(W/p)}$  is constant,

giving isoefficiency<sub>contention</sub> =  $O(p^3 \log p)$

2) Global RR:  $T_0 = O(p \log W)$  as  $V(p) = \Theta(p)$

→ Equating with problem size  $W$  & simplify as before,

communication isoefficiency =  $O(p \log p)$

→ Isoefficiency<sub>contention-target</sub>:  $\frac{O(p \log p)}{\Theta(W/p)}$  is const  $\Rightarrow \underline{O(p^2 \log p)}$

→ Isoefficiency<sub>bus-contention</sub>:  $\underline{O(p^2 \log p)}$

3) Random polling:  $T_0 = O(p \log p \log W)$

→ communication isoefficiency =  $\underline{O(p \log^2 p)}$

→ bus contention isoefficiency =  $\underline{O(p^2 \log^2 p)}$

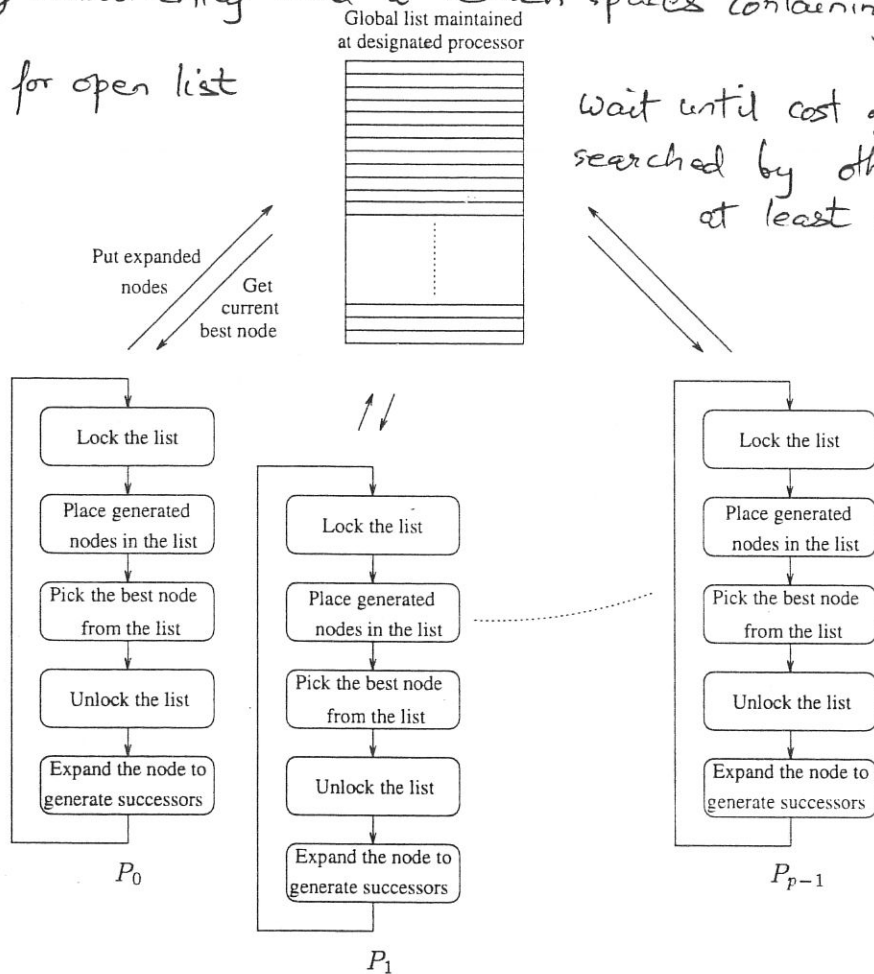
# Parallel Best-First Search

- open/closed list, each proc works on 1 of the current best nodes
- parallel BFS expands more than 1 node at a time ⇒ more work?
- if different procs concurrently expand different nodes from open list

2 problems

- should not terminate when a solution is found; as remaining  $(p-1)$  nodes may concurrently lead to search spaces containing better goal nodes

- contention for open list



**Figure 8.16** 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.

Copyright (r) 1994 Benjamin/Cummings Publishing Co.

Use a distributed open list; each processor has local open list & search space is statically divided initially

- communication needed to minimize unnecessary search
- trade-off communication & computation

Choice of communication strategy; depends on whether search space is tree or graph

# Strategies for Parallel BF Tree Search

Objective: ensure nodes with good  $f$ -values are distributed among processors

- 1) random: periodically send some best nodes to randomly selected processor
- 2) ring
- 3) blackboard: expand local best node only if it is within some tolerance limit of the best node on blackboard  
 If local best node is much better, some local nodes sent to blackboard  
 If worse, get some nodes from blackboard

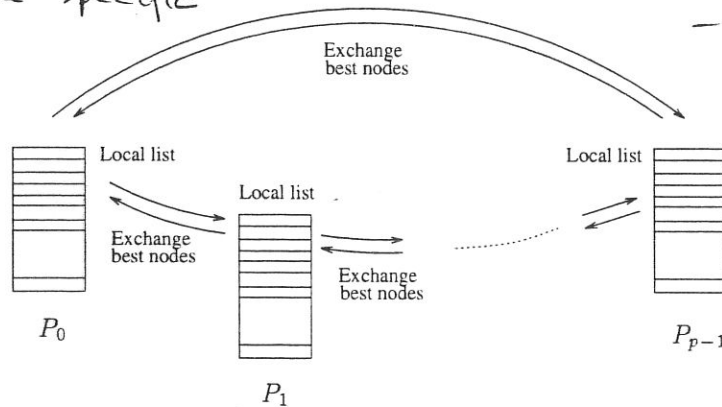
→ need shared-address space

# Strategies for Parallel BF Graph Search: check for node replication.

map each node to a specific processor

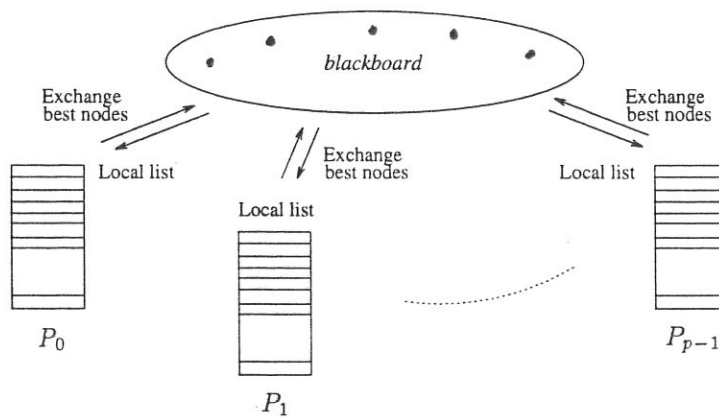
$\text{hash}(\text{node}) \rightarrow \text{proc.}$

→ each node generation results in communication



→ random hash fn:  
 each proc equally likely to be assigned a part of the search space that would also be explored by a sequential formulation

**Figure 8.17** A message-passing implementation of parallel best-first search using the ring communication strategy.  
 Copyright (r) 1994 Benjamin/Cummings Publishing Co.



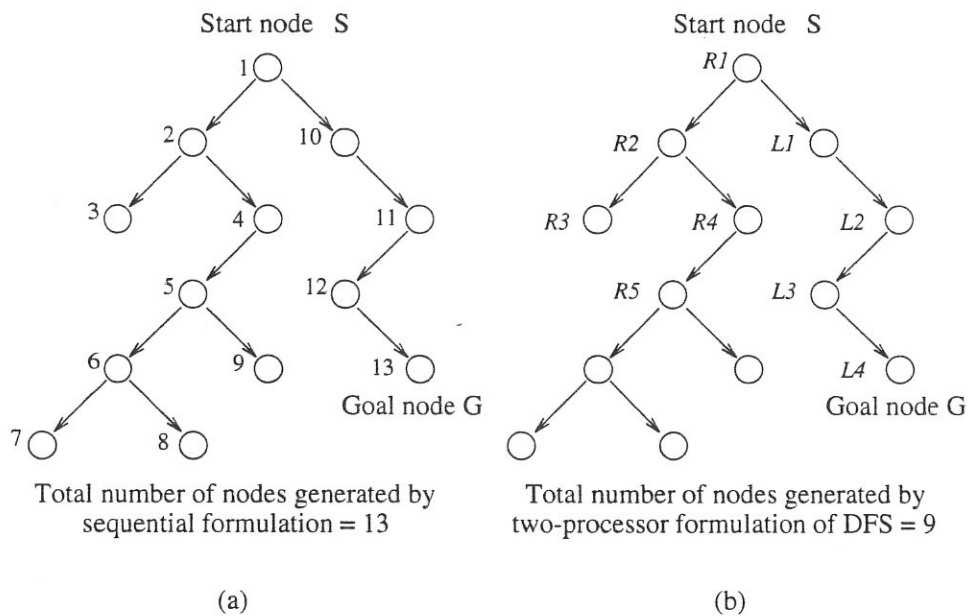
**Figure 8.18** An implementation of parallel best-first search using the blackboard communication strategy.  
 Copyright (r) 1994 Benjamin/Cummings Publishing Co.

# Speedup Anomalies in Parallel Search Algorithms

→ position of search space examined by various procs. determined dynamically and can differ for each execution

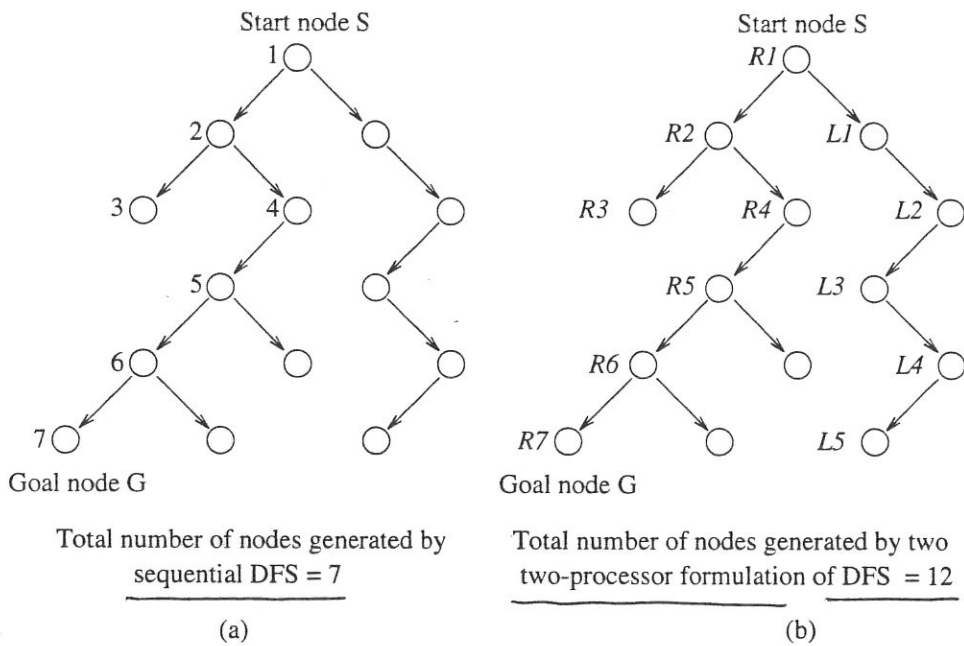
Eg: search overhead factor =  $\frac{9}{13}$

if communication ovhd is not too large, speedup will be superlinear



**Figure 8.19** 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.  
Copyright (r) 1994 Benjamin/Cummings Publishing Co.

eg: search overhead factor  $> 1$ ,  $\Rightarrow$  sublinear speedup



**Figure 8.20** A parallel DFS formulation that searches more nodes than its sequential counterpart.

Copyright (r) 1994 Benjamin/Cummings Publishing Co.

Speedup anomalies also manifest themselves in best-first search algorithms

$\rightarrow$  nodes on open list have identical heuristic values but require vastly different amounts of search to detect a solution

eg node A leads rapidly to goal node; node B leads nowhere after extensive work

- In parallel BFS, A & B chosen for expansion by different processors
- Seq:  $\#$  node A chosen first  $\Rightarrow$  parallel BFS has sublinear speedup
- $\#$  node B chosen first  $\Rightarrow$  parallel BFS has superlinear speedup



• Dijkstra's token termination detection algorithm

→ ring; black & white states; black & white token colors

→ [if work transferred from  $P_i$  to  $P_j$  &  $i > j$ , state = black token becomes black

(1) On becoming idle,  $P_0$  becomes white & sends white token to  $P_1$

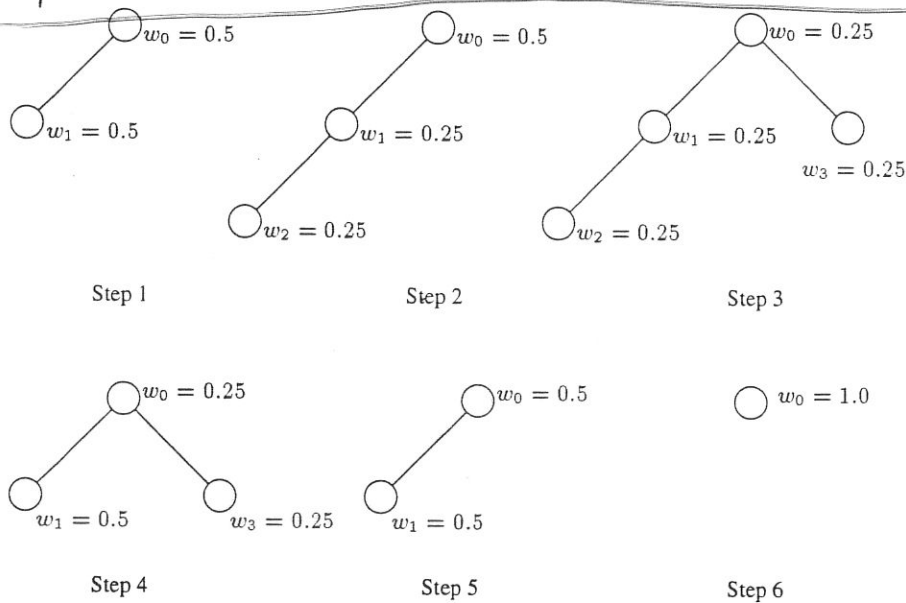
(2)  $P_i$  sends work to  $P_j$  &  $i > j \Rightarrow P_i$  becomes black

(3) If  $P_i$  has token & is idle, pass token to  $P_{i+1}$

→  $P_i = \text{black}$ : send black token to  $P_{i+1}$

→  $P_i = \text{white}$ : pass token unchanged.

(4) After  $P_i$  passes token to  $P_{i+1}$ ,  $P_i$  becomes white



**Figure 8.10** Tree based terminations detection. Steps 1–6 illustrate the weights at various processors after each work transfer.

Copyright (r) 1994 Benjamin/Cummings Publishing Co.

For a large # processors, overall isoefficiency  $\ln \uparrow$

Tree based termination detection

→ does not change overall isoefficiency function

Init: root has  $w = 1$

When a new proc joins: split  $w$  with new processor.

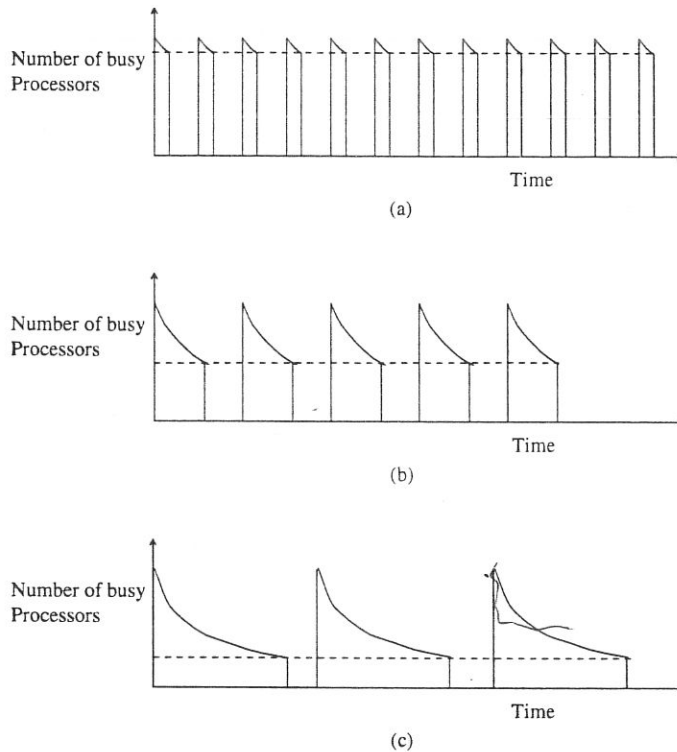
When a proc. finishes work: return its  $w$  to the loaner

# Parallel DFS on SIMD

2 problems

- 1) all procs must be in same stage of expansion
  - expanding different nodes requires different amounts of work
  - what if backtracking?
- 2) Load balancing must be performed globally

∴ All processors are in either search or load-balancing phase, at any time



**Figure 8.14** Three different triggering mechanisms: (a) a high triggering frequency leads to high load-balancing cost, (b) the optimal frequency yields good performance, and (c) a low frequency leads to high idle times.

Copyright (r) 1994 Benjamin/Cummings Publishing Co.

Critical components:—

- 1) load-balancing triggering mechanism;
- 2) load-balancing algorithm

eg trigger.

$w_{idle}$  = total idle time of all procs since beginning of current search phase

$t_L$  = total time spent by each processor in next load-balancing phase

trigger LB when  $w_{idle} \geq t_L \times p$

# Analysis of Average Speedup in Parallel DFS

## Assumptions

- 1) state space tree has  $M$  leaves. Solns occur only at leaf nodes
- 2) seq & parallel DFS stop after finding one solution
- 3) Parallel DFS: space tree equally partitioned among  $p$  procs.
- 4) At least one solution
- 5) No information to order search  $\Rightarrow$  density of solutions across unexplored nodes is independent of order of search
- 6) Solution density  $p \equiv$  probability (leaf node is soln).  
Assume Bernoulli distribution of solutions  $\Rightarrow$  event of a leaf node being a solution is independent of any other leaf node being a soln. [ $p \ll 1$ ]
- 7) Total # nodes generated by  $p$  procs before finding a soln:  $W_p$   
Avg. # leaf nodes generated by sequential DFS before finding a soln:  $W$   
 $W, W_p \leq M$

## Analysis

$p_i \equiv$  density of solns in region  $i$

Th: if  $p > 0$ , mean # leaves generated by a single processor searching a region is  $1/p$  (assuming  $K = \#$  leaves in region is large).

For a Bernoulli distribution, mean # trials is

$$p + 2p(1-p) + \dots + Kp(1-p)^{K-1} = \frac{1}{p} - (1-p)^K \left( \frac{1}{p} + K \right) \approx \frac{1}{p} \quad \square$$

• Avg # leaf nodes expanded by sequential DFS =  $W \approx \frac{1}{p} \left( \frac{1}{p_1} + \frac{1}{p_2} + \dots + \frac{1}{p_p} \right) = \frac{1}{HM}$

(assumes soln is always found in selected region. Probability that region  $i$  does not have any solns is  $(1-p_i)^K$  ..... overall result unaltered)

• In each step of parallel DFS, 1 node from each region explored simultaneously

$$P[\text{success in a step}] = 1 - \prod_{i=1}^p (1-p_i) \approx p_1 + p_2 + \dots + p_p, \text{ neglecting 2nd order terms}$$

$$W_p \approx \frac{p}{p_1 + p_2 + \dots + p_p} = \frac{1}{AM}$$

•  $AM \geq HM$