DISCRETE OPTIMIZATION PROBLEMS

(S, f) $\quad f: S \rightarrow R$. Objective: find $x_{opt} \mid f(x_{opt}) \leq f(x)$

S → feasible solution set

$g(x)$ = cost of reaching state x from initial state s along current path

$h(x)$ = heuristic estimate of reaching goal state from state x

$l(x) = g(x) + h(x)$

$$\sum_{tile=1}^{n} (\text{Manhattan distance between initial and final position})_{tile}$$

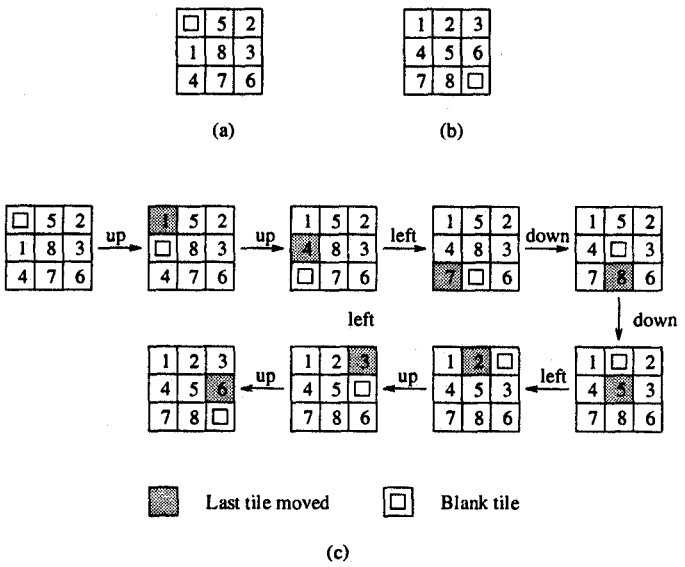


Figure 8.1 An 8-puzzle problem instance: (a) initial configuration; (b) final configuration; and (c) a sequence of moves leading from the initial to the final configuration.

Discrete optimization problems: NP-hard

Avg. time complexity of heuristic search algorithms: often polynomial

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

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

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

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

minimize

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

$A\bar{x} \geq b$

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

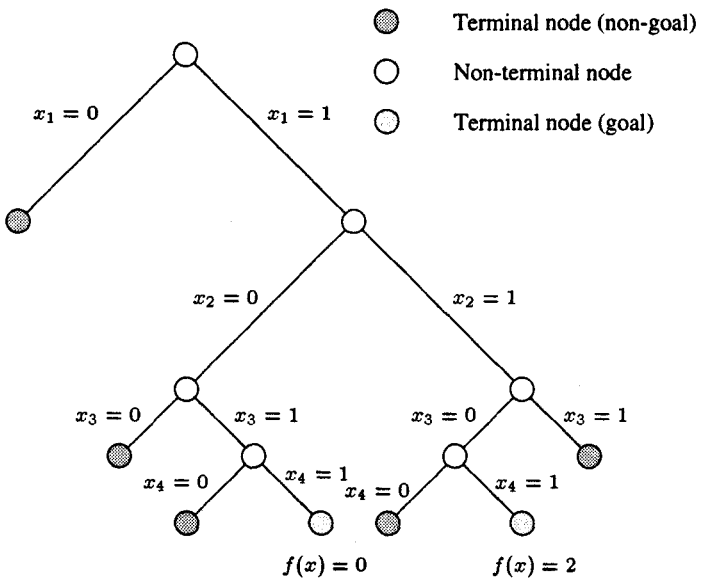


Figure 8.2 The graph corresponding to the 0/1 integer-linear-programming problem.

— 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

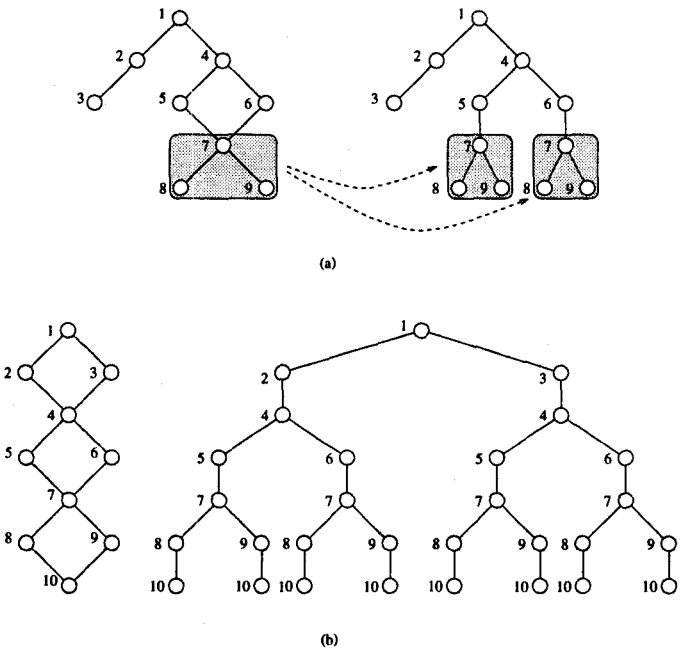
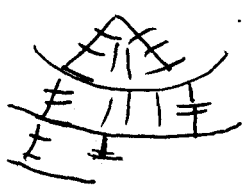


Figure 8.3 Two examples of unfolding a graph into a tree.

(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
  eg in 8-puzzle, up to 3 untried alternatives are stored at each step
  → $\Theta(nd)$, where d = max depth & n = storage

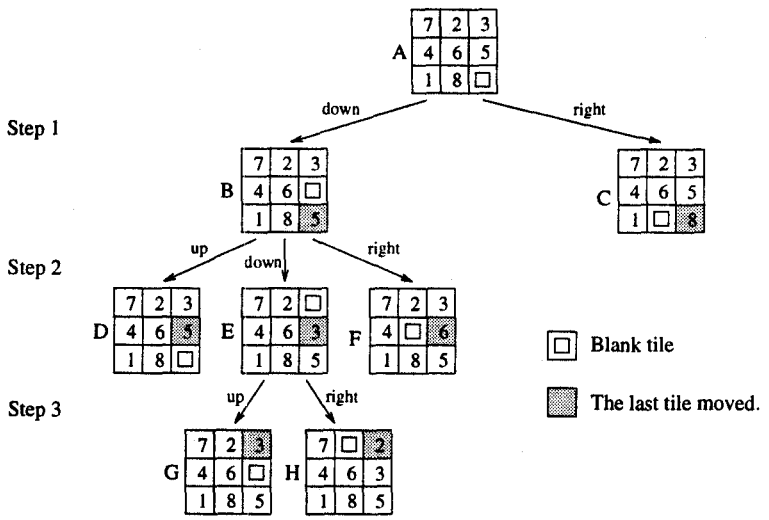


Figure 8.4 States resulting from the first three steps of depth-first search applied to an instance of the 8-puzzle.

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)

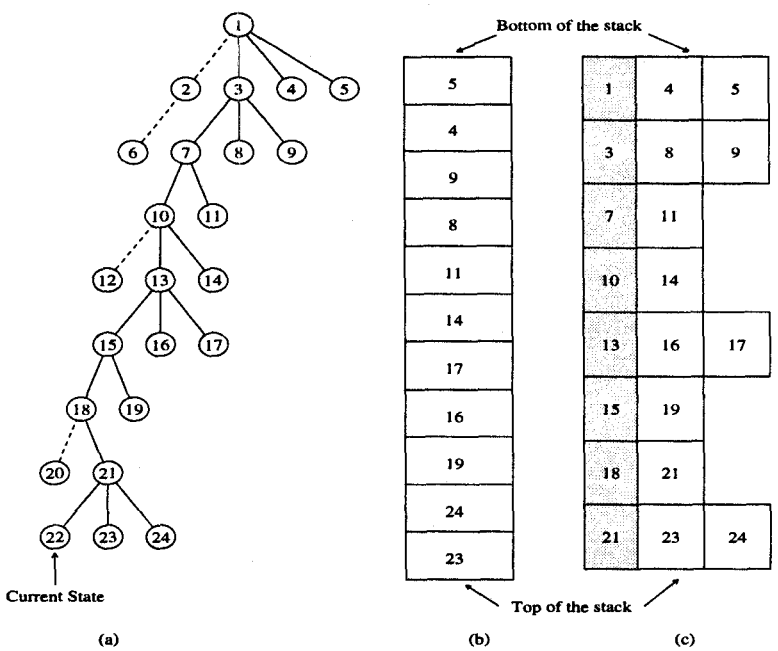


Figure 8.5 Representing a DFS tree: (a) the DFS tree; Successor nodes shown with dashed lines have already been explored; (b) the stack storing untried alternatives only; and (c) the stack storing untried alternatives along with their parent. The shaded blocks represent the parent state and the block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

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

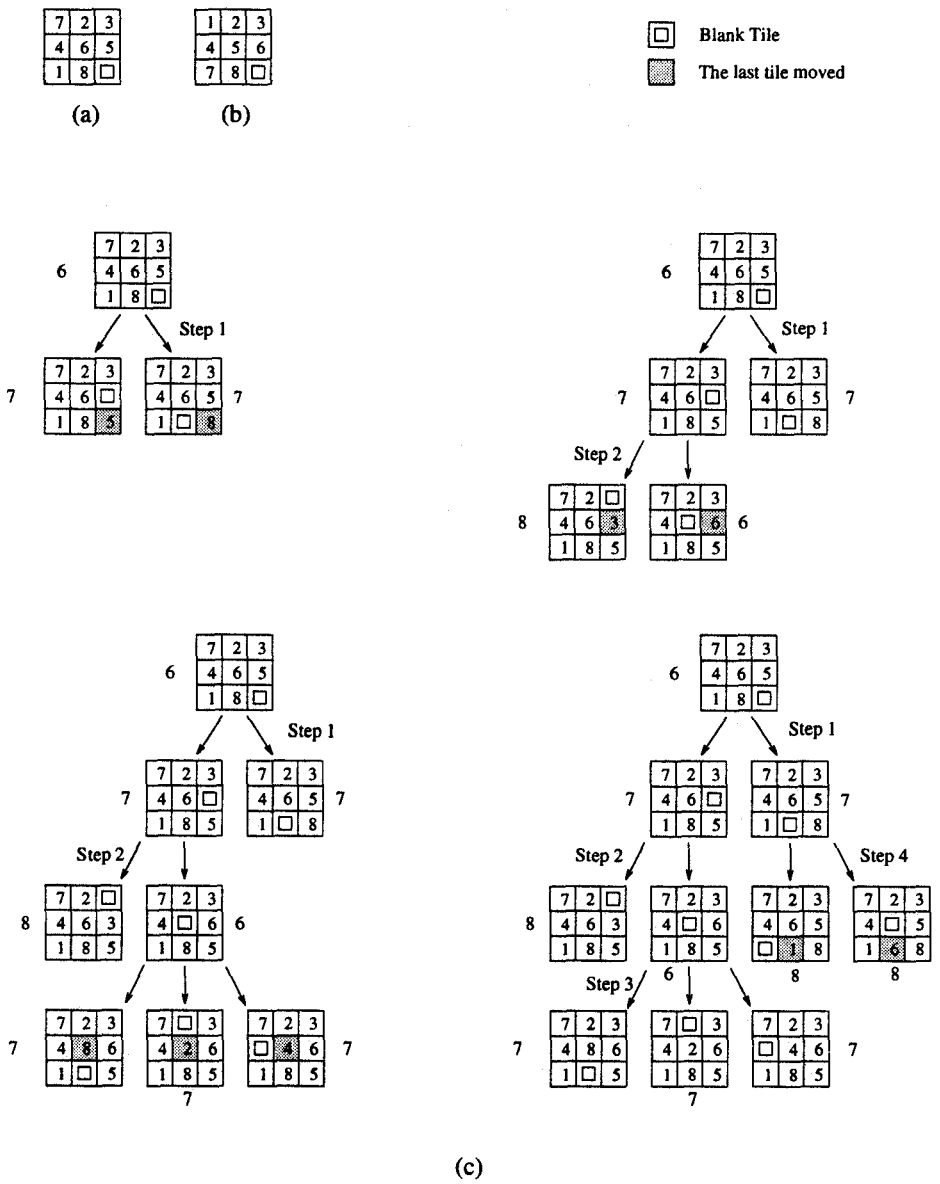


Figure 8.6 Applying best-first search to the 8-puzzle: (a) initial configuration; (b) final configuration; and (c) states resulting from the first four steps of best-first search. Each state is labeled with its h-value (that is, the Manhattan distance from the state to the final state).

PARALLEL DEPTH-FIRST SEARCH

- Assume time for node expansion = 1 $\Rightarrow$ $T_s = W$ (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 donor processor to ask when I become idle

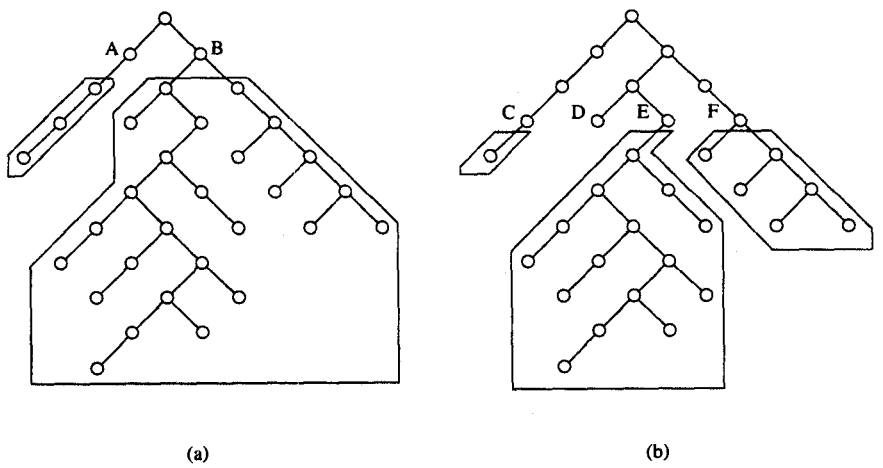


Figure 8.7 The unstructured nature of tree search and the imbalance resulting from static partitioning.

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

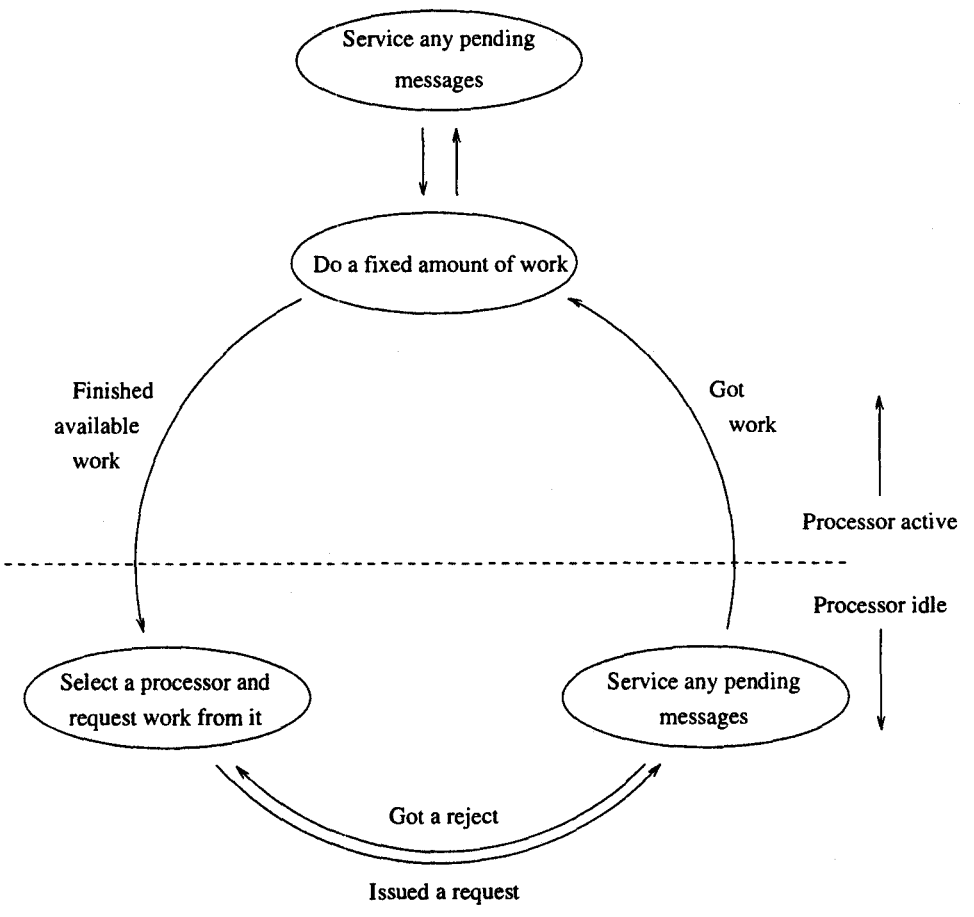


Figure 8.8 A generic scheme for dynamic load balancing.

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_o

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

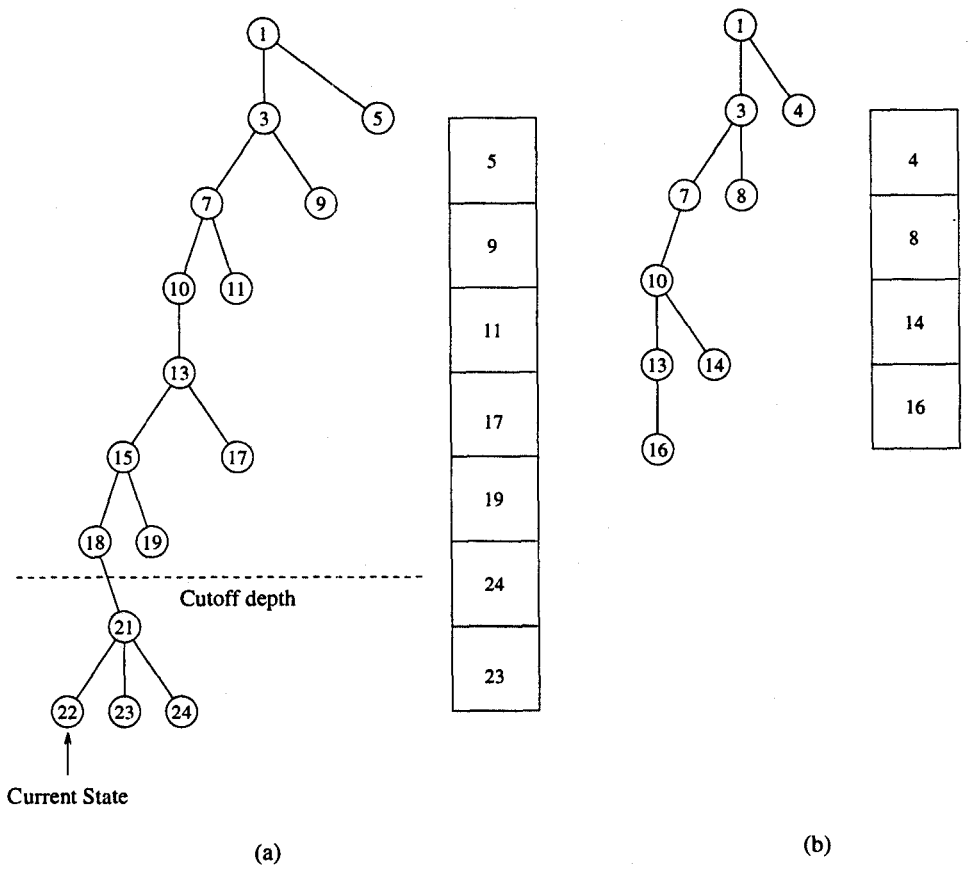


Figure 8.9 Splitting the DFS tree in Figure 8.5. The two subtrees along with their stack representations are shown in (a) and (b).

Technique to get upper bound on communication overhead assumes

1) Work can be partitioned as long as its size > threshold ϵ
2) <u>α-splitting</u>. α is a l.b. on the load imbalance that results from work splitting. Both partitions of w have at least αw work, i.e., $\psi w > \alpha w$ & $(1-\psi)w > \alpha w$

$V(P) \equiv$ 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, ——— $< \epsilon$

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

$T_o = t_{comm}\, V(P) \log W$ & $E = \frac{1}{1+T_o/W} = \frac{1}{1+\left(t_{comm}\, V(P) \log W\right)/W}$

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

→ 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 ∴ 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 $F(i,P) \longrightarrow F(P,P)$

$V(P) = f(0,P)$

$f(i,P) = \frac{i}{P}\left(1 + f(i,P)\right) + \frac{P-i}{P}\left(1 + f(i+1,P)\right)$

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

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

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

$= P \sum_{i=1}^{P} \frac{1}{i} = P \times H_P$, where H_P is a harmonic number

As P becomes large, $H_P \simeq 1.69 \ln P$ ∴ $V(P) = \Theta(P \log P)$

Analysis of Load-balancing Schemes for HCs

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

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

1) Async RR

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

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

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

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

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

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

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 ↑W so as to keep ratio between W/p and $O(p \log W)$ constant

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

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

Equating T_o 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 $t_{comm} = \Theta(\log p)$ was assumed on HC, in $t_s + m t_w + t_h \log p$, $(t_s + m t_w)$ constant dominates $t_h \log p$ ∴ $t_{comm} = \Theta(1)$

⇒ For RP & ARR, isoefficiency fun (due to comm) goes ↓ by $(\log p)$

∴ 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 $\Rightarrow t_{comm} = \Theta(1)$

→ $T_o = O(V(P) \log W)$

1) Asynch RR $\quad T_o = (P^2 \log W)$

→ Equating with problem size W & simplify as before,
communication isoefficiency $= O(P^2 \log P)$

→ For $\text{isoefficiency}_{contention}$, 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 $\text{isoefficiency}_{contention} = O(P^3 \log P)$

2) Global RR : $T_o = O(P \log W)$ as $V(P) = \Theta(P)$

→ Equating with problem size W & simplify as before,
communication isoefficiency $= O(P \log P)$

→ $\text{Isoefficiency}_{contention-target}$: $\frac{O(P \log W)}{\Theta(W/P)}$ is const $\Rightarrow O(P^2 \log P)$

→ $\text{Isoefficiency}_{bus\_contention}$: $O(P^2 \log P)$

3) Random polling : $T_o = O(P \log P \log W)$

→ communication isoefficiency $= O(P \log^2 P)$

→ bus contention isoefficiency $= O(P^2 \log^2 P)$

• 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 = black: send black token to P_{i+1}

→ P_i = white: pass token unchanged.

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

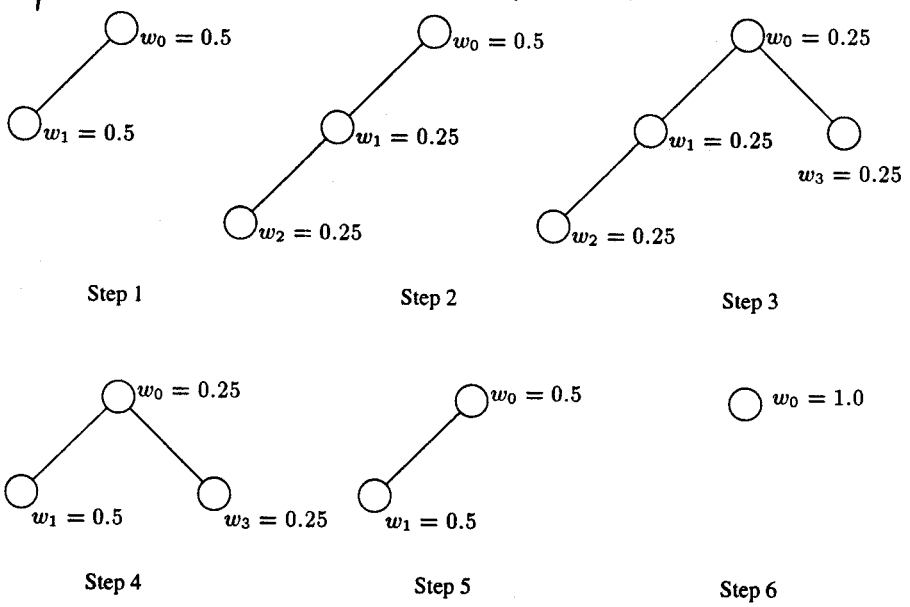


Figure 8.10 Tree based terminations detection. Steps 1–6 illustrate the weights at various processors after each work transfer.

For a large # processors, overall isoefficiency fn ↑

Tree based termination detection

→ does not change overall isoefficiency function

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

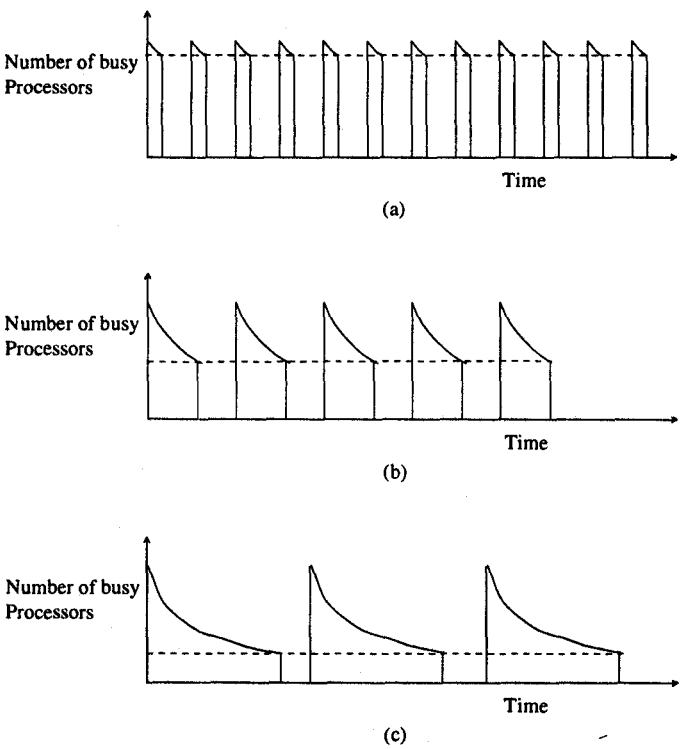


Figure 8.14 Three different triggering mechanisms: (a) a high triggering frequency leads to high load-balancing cost, (b) the optimal frequency yields good performance, and (c) a low frequency leads to high idle times.

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 $w_{idle} \geq t_L \times p$

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

Wait until cost of solns being searched by other procs is at least c

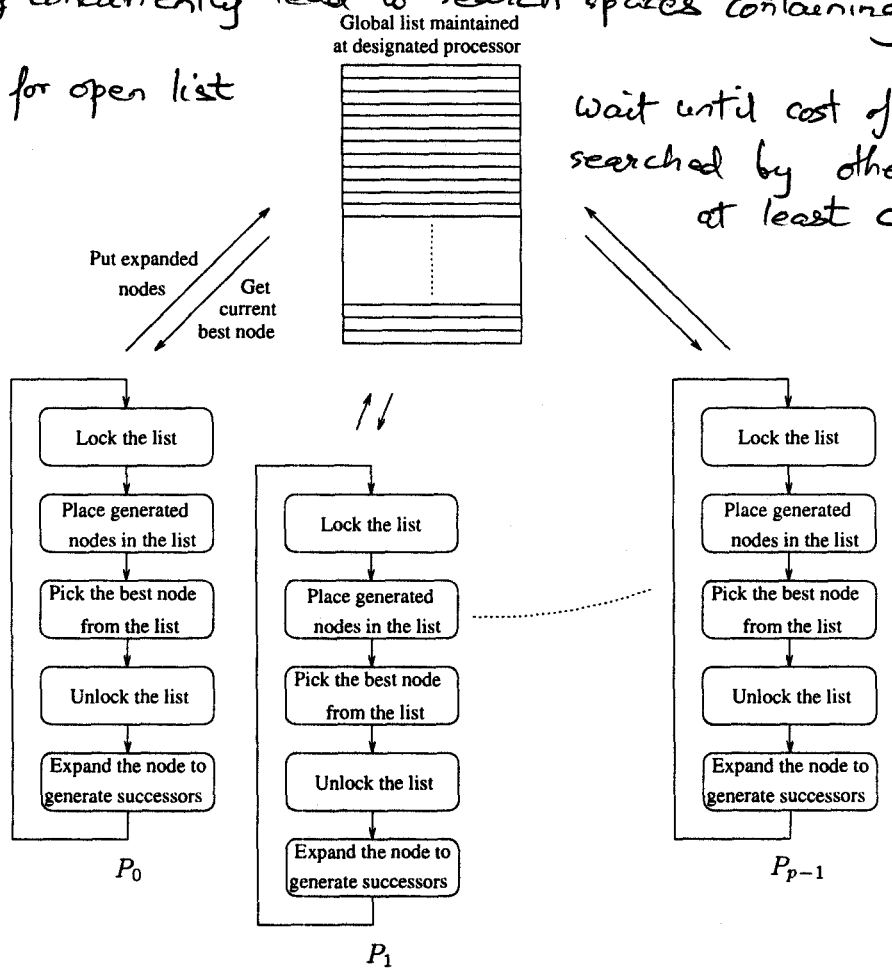


Figure 8.16 A general schematic for parallel best-first search using a centralized strategy. The locking operation is used here to serialize queue access by various processors.

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

hash(node) → 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

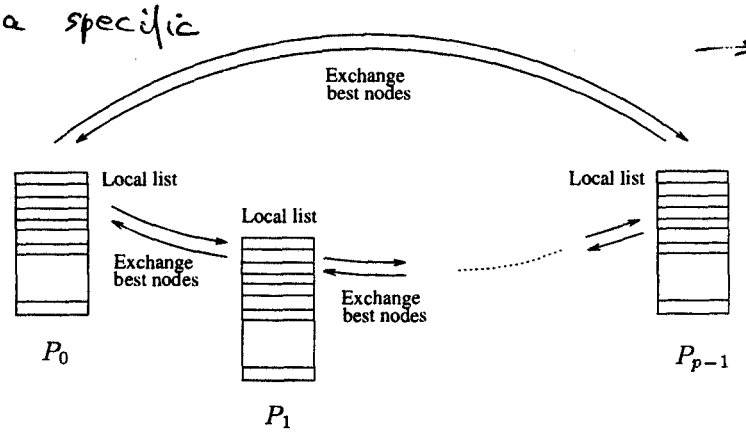


Figure 8.17 A message-passing implementation of parallel best-first search using the ring communication strategy.

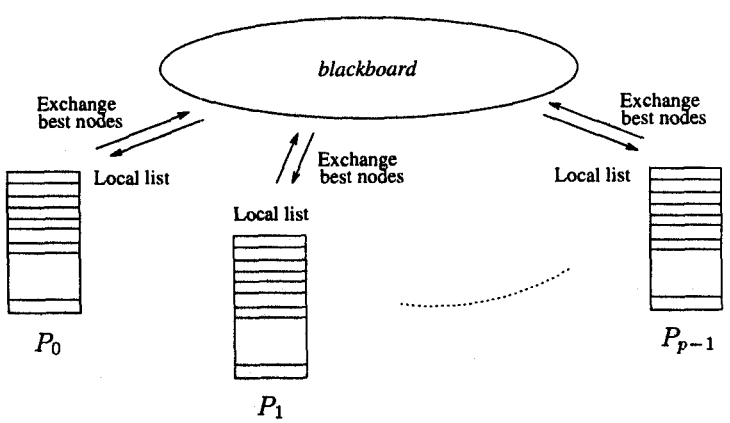


Figure 8.18 An implementation of parallel best-first search using the blackboard communication strategy.

Speedup Anamolies in Parallel Search Algorithms

→ portion 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

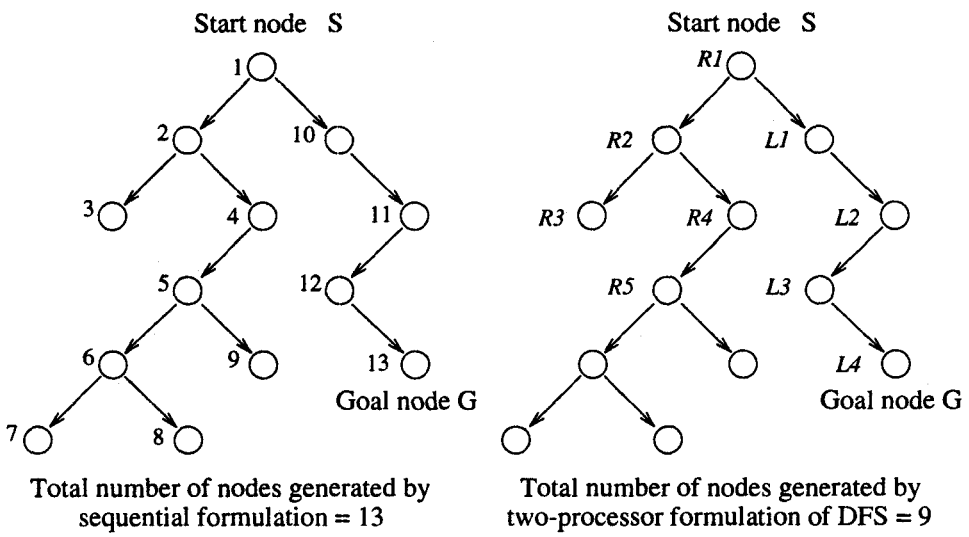


Figure 8.19 The difference in number of nodes searched by sequential and parallel formulations of DFS. For this example, parallel DFS reaches a goal node after searching fewer nodes than sequential DFS.

eg: search ovhd factor > 1, ⇒ sublinear speedup

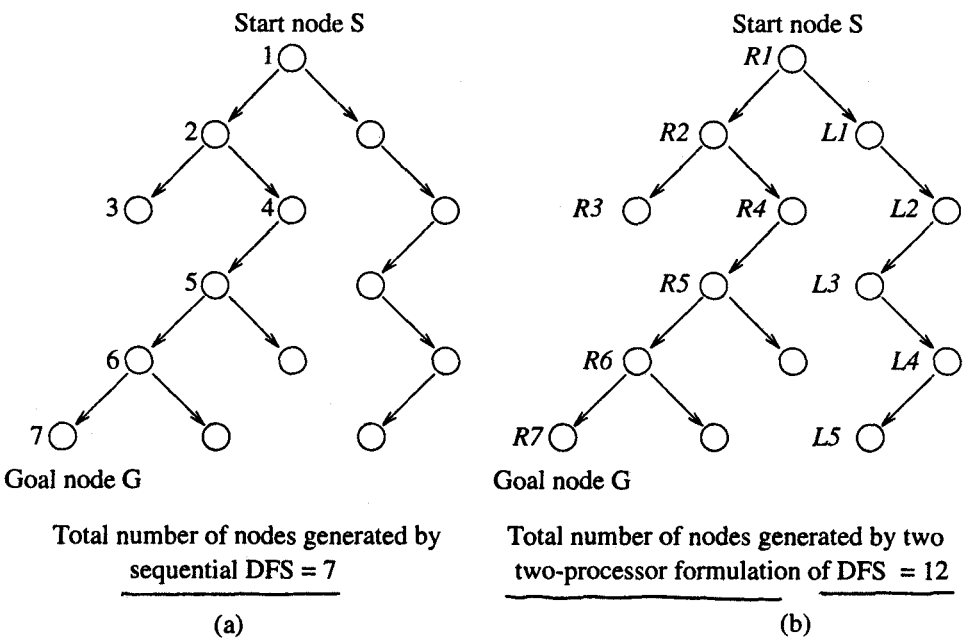


Figure 8.20 A parallel DFS formulation that searches more nodes than its sequential counterpart.

Speedup anamolies also manifest themselves in best-first search algorithms

→ 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: If node A chosen first ⇒ parallel BFS has sublinear speedup
  If node B chosen first ⇒ parallel BFS has superlinear speedup

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 $\rho \equiv$ 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. [$\rho \ll 1$]
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

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

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

For a Bernoulli distribution, mean # trials is

$$\rho + 2\rho(1-\rho) + \cdots\cdots + K\rho(1-\rho)^{K-1} = \frac{1}{\rho} - (1-\rho)^K\left(\frac{1}{\rho} + K\right) \approx \frac{1}{\rho}$$ ∎

- Avg # leaf nodes expanded by sequential DFS $= W \approx \frac{1}{p}\left(\frac{1}{\rho_1} + \frac{1}{\rho_2} + \cdots\cdots + \frac{1}{\rho_p}\right) = \frac{1}{HM}$

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

- In each step of parallel DFS, 1 node from each region explored simultaneously.
  $P[\text{success in a step}] = 1 - \prod_{i=1}^{p}(1-\rho_i) \approx \rho_1 + \rho_2 + \cdots + \rho_p$, neglecting 2nd order terms

  $$W_p \approx \frac{p}{\rho_1 + \rho_2 + \cdots\cdots + \rho_p} = \frac{1}{AM}$$

- $AM \geq HM$