Monte Carlo Methods; Combinatorial Search

Parallel and Distributed Computing

Department of Computer Science and Engineering (DEI)
Instituto Superior Técnico

November 22, 2012

Outline

- Monte Carlo methods
- Parallel Search
  - Backtrack Search
  - Branch and Bound

Monte Carlo Method

Solve a problem using statistical sampling.

Monte Carlo Method

Solve a problem using statistical sampling.

Applications of the Monte Carlo method:

- integrals of arbitrary functions of 6+ dimensions
- predicting future values of stocks
- solving partial differential equations
- sharpening satellite images
- modeling cell populations
- finding approximate solutions to NP-hard problems

Monte Carlo Method

Solve a problem using statistical sampling.

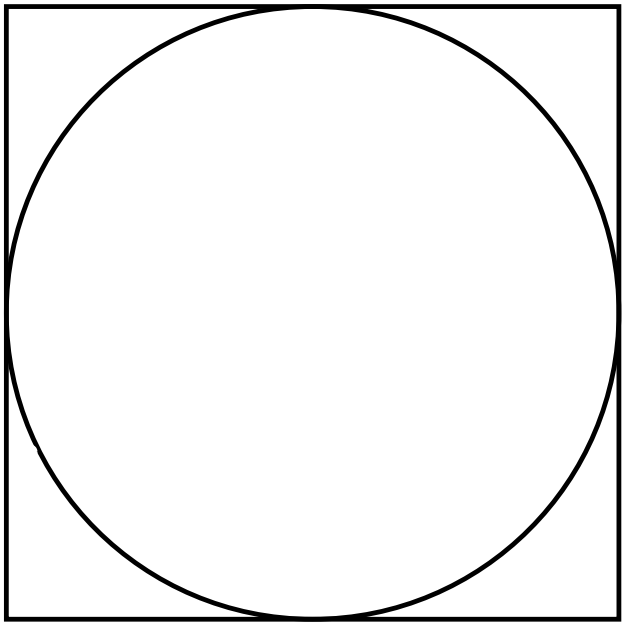
Applications of the Monte Carlo method:

- integrals of arbitrary functions of 6+ dimensions
- predicting future values of stocks
- solving partial differential equations
- sharpening satellite images
- modeling cell populations
- finding approximate solutions to NP-hard problems

It is estimated that over half the total cycles in parallel systems is spent with Monte Carlo methods!

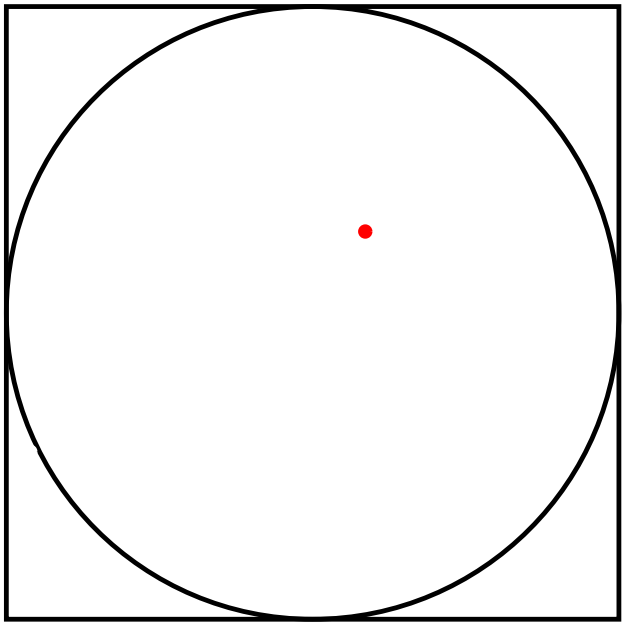
Example of Monte Carlo Method

Computation of π:



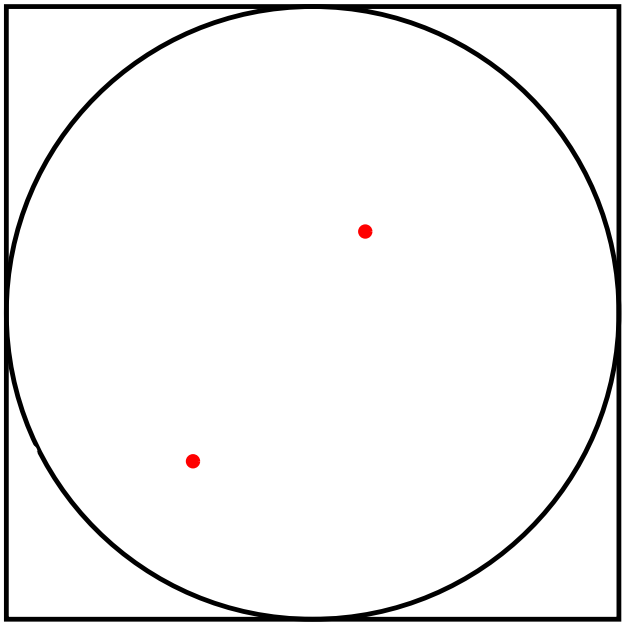
Example of Monte Carlo Method

Computation of π:



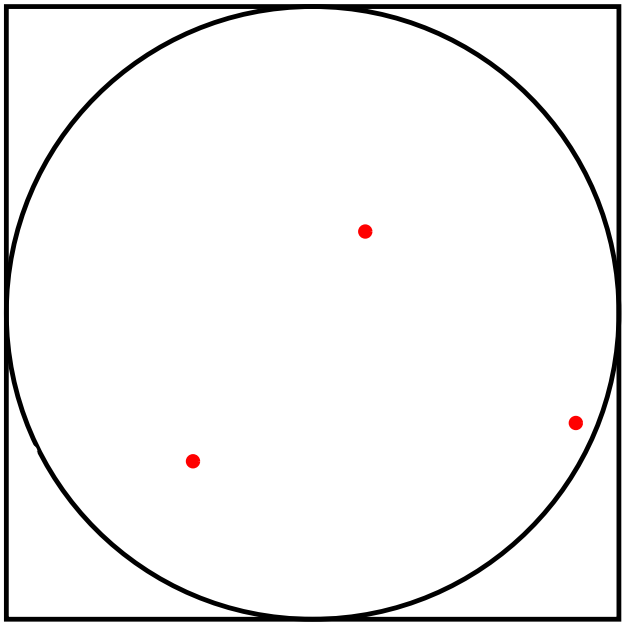
Example of Monte Carlo Method

Computation of π:



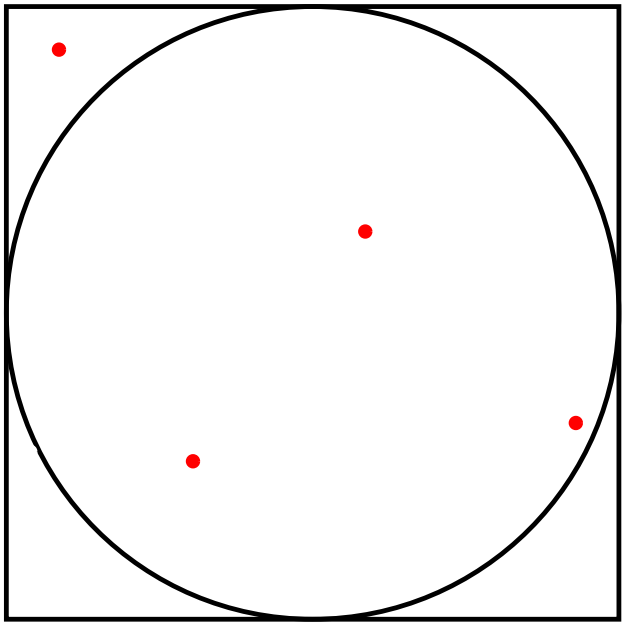
Example of Monte Carlo Method

Computation of π:



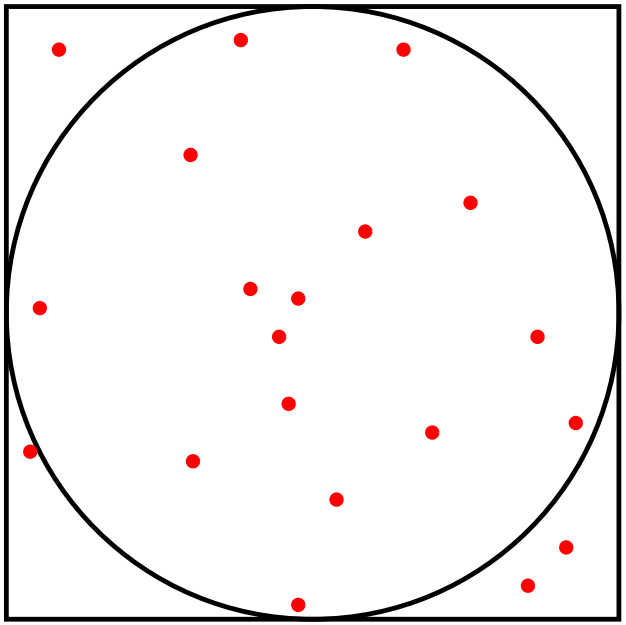
Example of Monte Carlo Method

Computation of π:



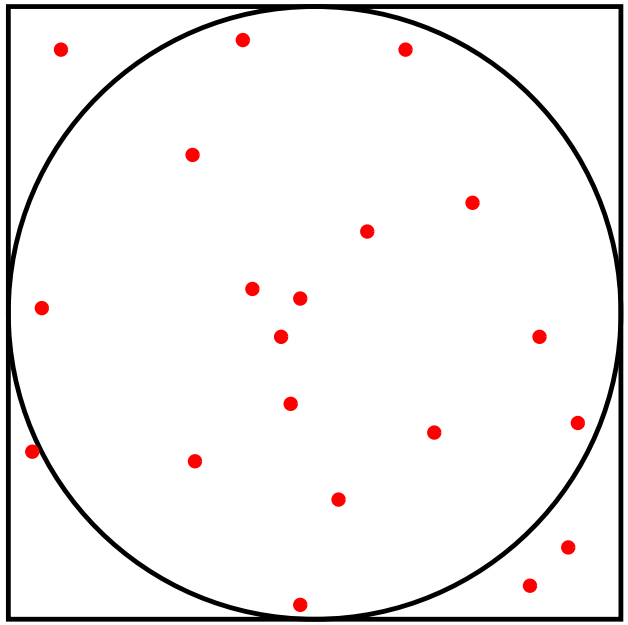
Example of Monte Carlo Method

Computation of π:



Example of Monte Carlo Method

Computation of π:



$$\frac{\text{Area}_{\bigcirc}}{\text{Area}_{\square}} = \frac{\pi D^2/4}{D^2} = \frac{\pi}{4} \qquad \frac{16}{20} = \frac{\pi}{4} \Rightarrow \pi \approx \frac{64}{20} = 3,2$$

Sample Size Defines the Error

n	Estimate	Error	$1/(2\sqrt{n})$
10	3,40000	0,23606	0,15811
100	3,36000	0,06952	0,05000
1.000	3,14400	0,00077	0,01581
10.000	3,13920	0,00076	0,00500
100.000	3,14132	0,00009	0,00158
1.000.000	3,14006	0,00049	0,00050
10.000.000	3,14136	0,00007	0,00016
100.000.000	3,14154	0,00002	0,00005
1.000.000.000	3,14155	0,00001	0,00002

Why Monte Carlo Works

Mean Value Theorem

$$\int_a^b f(x)dx = (b-a)\overline{f}$$

Monte Carlo methods obtain an estimate for $\overline{f}$:

$$\overline{f} \approx \frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

Accuracy increases at a rate of $1/\sqrt{n}$.

Why Monte Carlo is Effective

- error in Monte Carlo estimate decreases by the factor $1/\sqrt{n}$
- rate of convergence independent of integrand's dimension
- deterministic numerical integration methods do not share this property
- hence Monte Carlo superior when integrand has 6 or more dimensions

Monte Carlo methods are typically trivially amenable for parallelization.

Two possible views:

- obtain an estimate p times faster
- decrease error by $\sqrt{p}$

Another Example

Statistical Timing Analysis (STA)

Obtain the arrival times in the nodes of a circuit given that the delays are subject to variation.

Types of variations:

- Manufacturing process variations
  - no two chips are equal
  - the fabrication of a chip is a sequence of hundreds of operations, each of which will be performed a little differently on each chip
  - one of the main sources of variation in chip performances
- Environmental or operating conditions
  - e.g. changes in power supply voltage or operating temperature
- Device fatigue phenomena
  - e.g. electromigration, hot electron effects and NBTI (negative bias temperature instability)

STA: Arrival Times Accuracy

- Since delay is not given by a deterministic value, deterministic propagation of arrival times can not be done
- If we assume the average and worst-case values for the delays of each individual node we can do a worst-case analysis
  - but not accurate enough
- Or we can assume that delay variation follows a probability distribution which is independent for each node
  - better modeling of real conditions
  - much harder to propagate distributions

STA: Arrival Times Accuracy

- Since delay is not given by a deterministic value, deterministic propagation of arrival times can not be done
- If we assume the average and worst-case values for the delays of each individual node we can do a worst-case analysis
  - but not accurate enough
- Or we can assume that delay variation follows a probability distribution which is independent for each node
  - better modeling of real conditions
  - much harder to propagate distributions

Monte-Carlo comes to the rescue!

STA using Monte-Carlo

Repeat N times:

1. Randomly generate a delay value for each nodes
   - according to the probabilistic distribution at each node
2. Do a deterministic propagation of the arrival times
3. Store computed arrival times

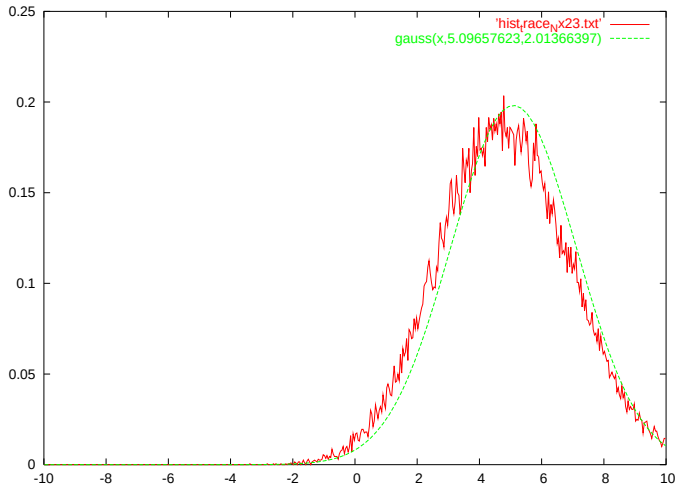
STA using Monte-Carlo

Repeat N times:

1. Randomly generate a delay value for each nodes
   - according to the probabilistic distribution at each node
2. Do a deterministic propagation of the arrival times
3. Store computed arrival times

Statistical distribution is obtained:

- Sample mean gives us the average arrival times
- Standard deviation gives us a confidence interval



STA Results

Results for some circuits using a grid with 5 machines

Benchmark	*#Blocks*	*#Nodes*	*#PIs*	*#POs*	Standalone		Grid		Speedup
					CPU	MEM	CPU	MEM	
c432	171	207	36	7	1.46	1.52	0.41	4.70	3.56
c499	218	259	41	32	1.82	1.53	0.41	4.71	4.44
c880	383	443	60	26	3.17	1.59	0.72	4.76	4.40
c1355	562	603	41	32	4.33	1.64	1.05	4.81	4.12
c1908	972	1005	33	25	7.15	1.73	1.48	5.02	4.83
c2670	1211	1445	233	140	10.05	2.83	4.11	5.11	3.55
c3540	1705	1755	50	22	13.36	1.95	2.76	5.21	4.84
c5315	2351	2529	178	123	19.61	2.15	4.04	5.44	4.85
c6288	2416	2448	32	32	20.59	2.28	4.33	5.57	4.76
c7552	3624	3832	207	108	30.19	2.52	6.19	5.75	4.88
csa.32.4	200	265	65	33	1.88	1.53	0.41	4.71	4.59
csa.64.4	400	529	129	65	3.75	1.61	0.80	4.78	4.69
csa.128.4	800	1057	257	129	7.54	1.77	1.77	5.04	4.25
csa.256.4	1600	2113	513	257	16.20	2.07	3.51	5.33	4.61
csa.1024.4	6400	8449	2049	1025	68.03	3.88	14.65	7.33	4.64
csa.8192.4	51200	67585	16385	8193	561.40	20.92	124.94	25.09	4.49

Random Number Generators

Computers are deterministic machines: pseudo-random numbers

Desirable properties of a Random Number Generator:

- uniformly distributed
- uncorrelated
- never cycles
- satisfies any statistical test for randomness
- reproducible
- machine-independent
- changing "seed" value changes sequence
- easily split into independent subsequences
- fast
- limited memory requirements

Random Number Generators

Ideal Random Number Generators are not possible.

- finite number of states $\Rightarrow$ cycles
- reproducible $\Rightarrow$ correlation

Random Number Generators

Ideal Random Number Generators are not possible.

- finite number of states $\Rightarrow$ cycles
- reproducible $\Rightarrow$ correlation

Linear Congruential Random Number Generators

$$X_i = (aX_{i-1} + c) \mod M$$

Sequence depends on the seed X_0.

Random Number Generators

Ideal Random Number Generators are not possible.

- finite number of states $\Rightarrow$ cycles
- reproducible $\Rightarrow$ correlation

Linear Congruential Random Number Generators

$$X_i = (aX_{i-1} + c) \mod M$$

Sequence depends on the seed X_0.

Lagged Fibonacci Random Number Generators

$$X_i = X_{i-p} * X_{i-q}$$

where $*$ is a binary operation, ie, exclusive-OR, addition modulo M, etc.

Parallel Random Number Generators

Centralized: master-slave approach

Parallel Random Number Generators

Centralized: master-slave approach

Decentralized:

- Leapfrog
- sequence splitting
- independent sequences

Parallel Random Number Generators

Centralized: master-slave approach

Decentralized:

- Leapfrog
- sequence splitting
- independent sequences

Independent sequences tend to work well as long as each task uses different seeds.

Combinatorial Search

Combinatorial search algorithm: find one or more optimal or suboptimal solutions in a defined problem space

Decision problems: determine if there is a solution to a given set of constraints

- Satisfiability (SAT)
- Hamiltonian path
- Eulerian path

Optimization problems: find a solution that maximizes or minimizes a cost function under a set of constraints

- Shortest path
- Traveling salesman (TSP)
- Maximum satisfiability (MAX-SAT)

Combinatorial Problems

"Easy" combinatorial problems have known algorithms that run in polynomial-time.

- Shortest path
- Eulerian path

Combinatorial Problems

"Easy" combinatorial problems have known algorithms that run in polynomial-time.

- Shortest path
- Eulerian path

The only known solutions to "hard" problems run in exponential-time.

- Satisfiability (SAT)
- Hamiltonian path
- Traveling salesman (TSP)
- Maximum satisfiability (MAX-SAT)

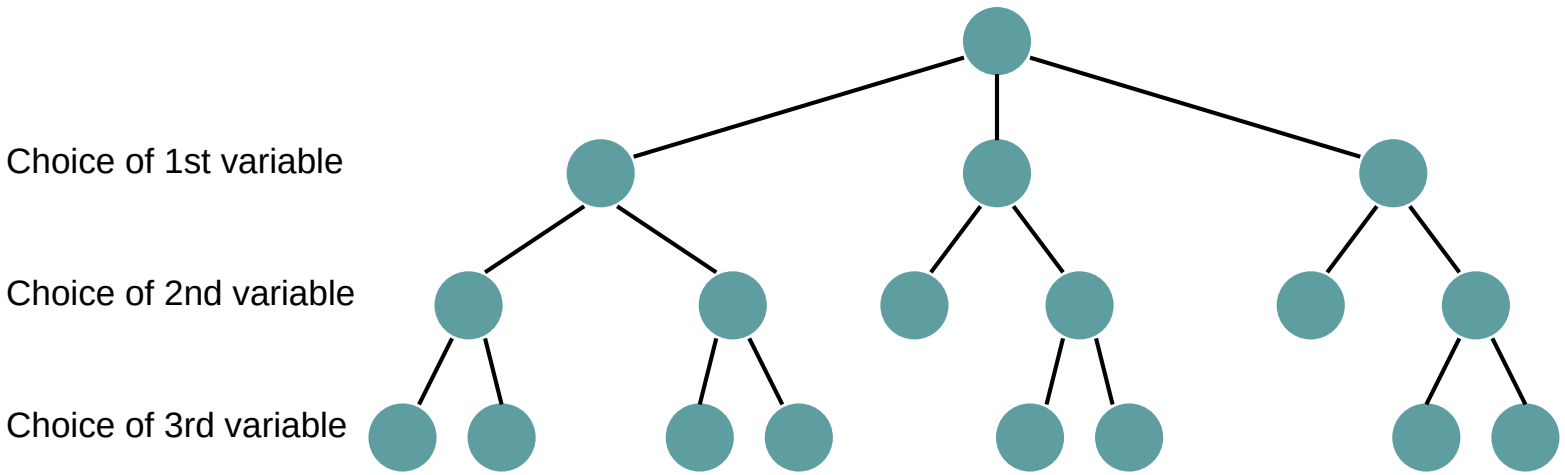
Backtrack Search

Backtrack search uses a depth-first search (typically recursive) to consider alternative solutions to a combinatorial search problem.

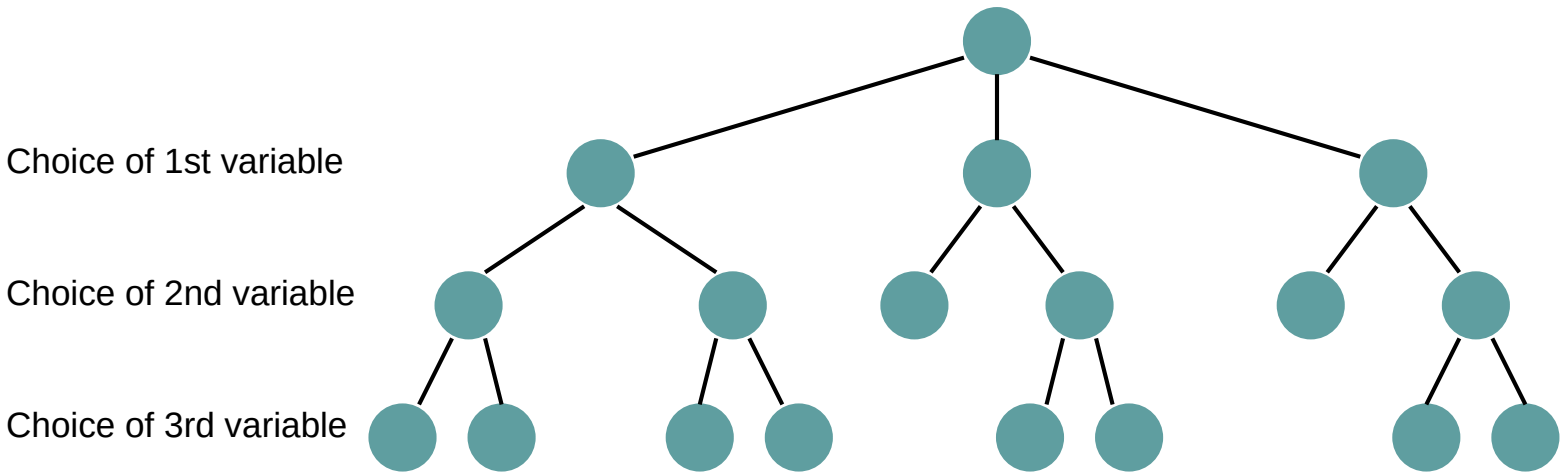
Backtrack occurs when:

- a node has no children
- all of a node's children have been explored
- current solution does not verify constraints

Backtrack Search



Backtrack Search



If b is the branching factor,

- at level k of the tree we have b^k nodes.
- up to level k of the tree we have $\frac{b^{k+1}-b}{b-1} + 1$ nodes.

Parallel Backtrack Search

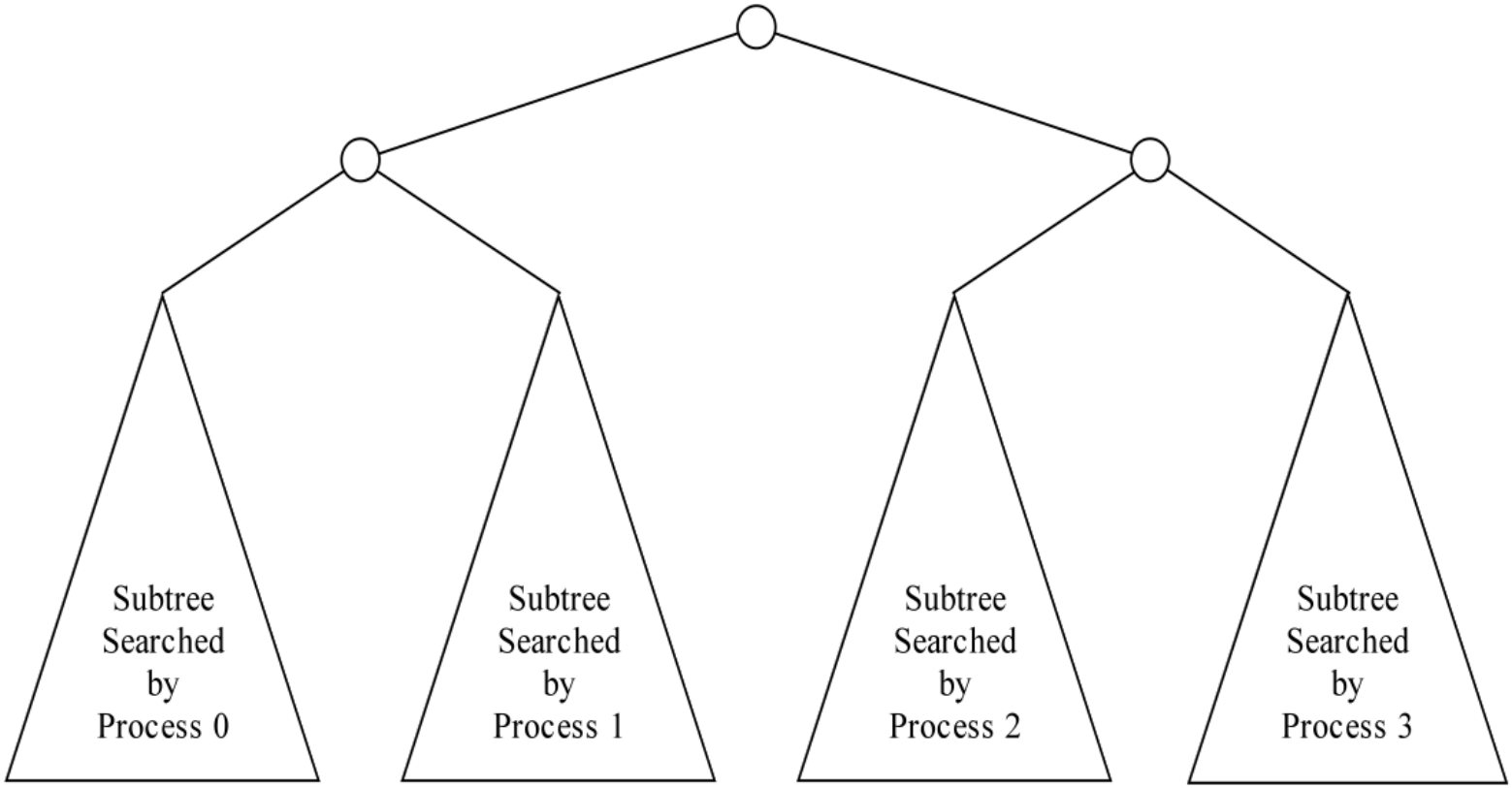
First strategy: suppose $p = b^k$ and give each processor a subtree.

- each process searches all nodes to depth k
- but then explores only one of subtrees rooted at level k

Typically, $k = \log_b p$ is much smaller than the depth of the tree.

Time required by each process to traverse first k levels of state space tree inconsequential.

Parallel Backtrack Search



Parallel Backtrack Search

What if $p \neq b^k$?

- define a level m up to which all processes execute redundant search in parallel.
- at level m, assign a subset of the b^m subtrees to the p processes

Parallel Backtrack Search

What if $p \neq b^k$?

- define a level m up to which all processes execute redundant search in parallel.
- at level m, assign a subset of the b^m subtrees to the p processes

What value of m to use?

Parallel Backtrack Search

What if $p \neq b^k$?

- define a level m up to which all processes execute redundant search in parallel.
- at level m, assign a subset of the b^m subtrees to the p processes

What value of m to use?

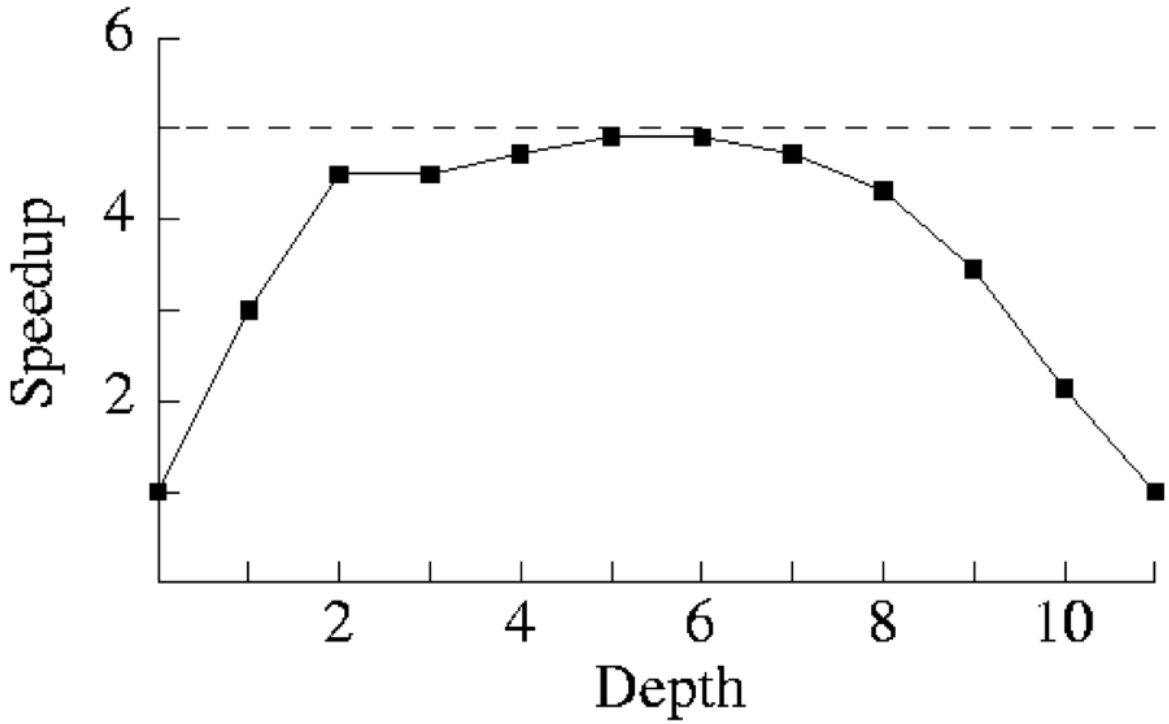
The larger the value of m

the larger the redundant computation

but better load balancing

Parallel Backtrack Search

Example: $p = 5$ processors exploring a state space tree with branching factor $b = 3$ and maximum depth 10.

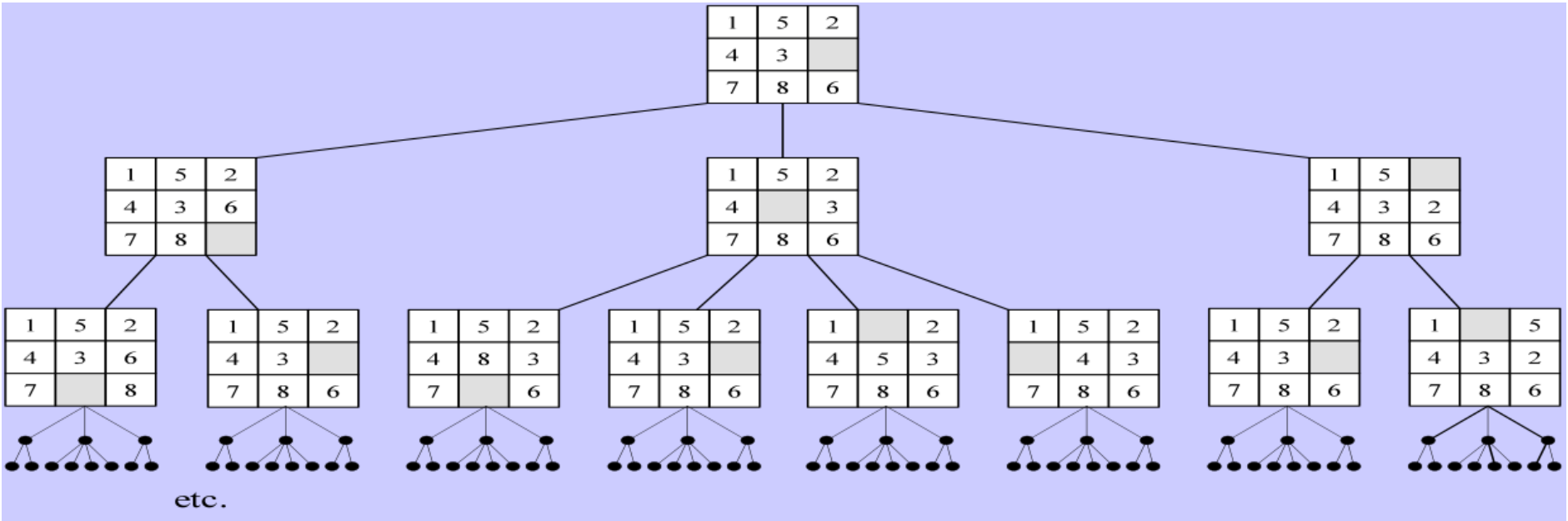


Branch and Bound

Branch and Bound, BB, is a general algorithm for optimization problems that also performs a depth-first search, but is able to prune its search by using upper and lower estimated bounds of the cost function.

Example: find the minimum number of moves to solve the 8-puzzle.

Branch and Bound



Branch and Bound

- could solve puzzle by pursuing breadth-first search of state space tree
- we want to examine as few nodes as possible
- can speed search if we associate with each node an estimate of minimum number of tile moves needed to solve the puzzle, given moves made so far

Branch and Bound

- could solve puzzle by pursuing breadth-first search of state space tree
- we want to examine as few nodes as possible
- can speed search if we associate with each node an estimate of minimum number of tile moves needed to solve the puzzle, given moves made so far

Compute Manhattan distance of each tile to each final position
$\Rightarrow$ overall sum is a lower bound on number of moves still required

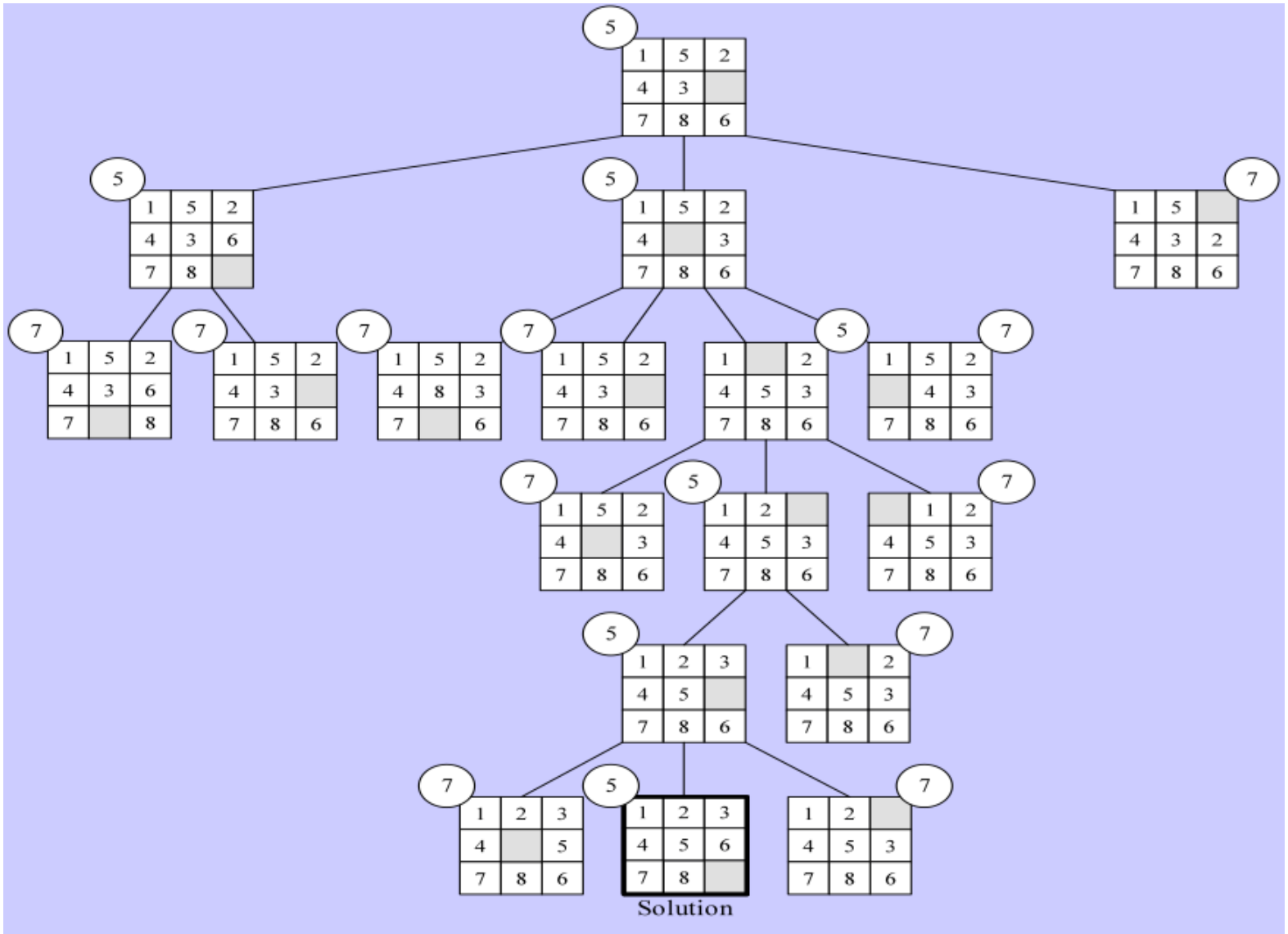
Branch and Bound

- could solve puzzle by pursuing breadth-first search of state space tree
- we want to examine as few nodes as possible
- can speed search if we associate with each node an estimate of minimum number of tile moves needed to solve the puzzle, given moves made so far

Compute Manhattan distance of each tile to each final position
$\Rightarrow$ overall sum is a lower bound on number of moves still required

If current depth + lower bound higher than an existing solution, discard subtree.

Branch and Bound



Parallel Branch and Bound

Sequential algorithm uses a priority queue.

Single priority queue too expensive to maintain in a distributed parallel system.
$\Rightarrow$ 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

Parallel Branch and Bound

- 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
- Conditions for solution found be guaranteed optimal: all nodes in state space tree with smaller lower bounds must be explored

Review

- Monte Carlo methods
- Parallel Search
  - Backtrack Search
  - Branch and Bound

Next Class

- Graph algorithms