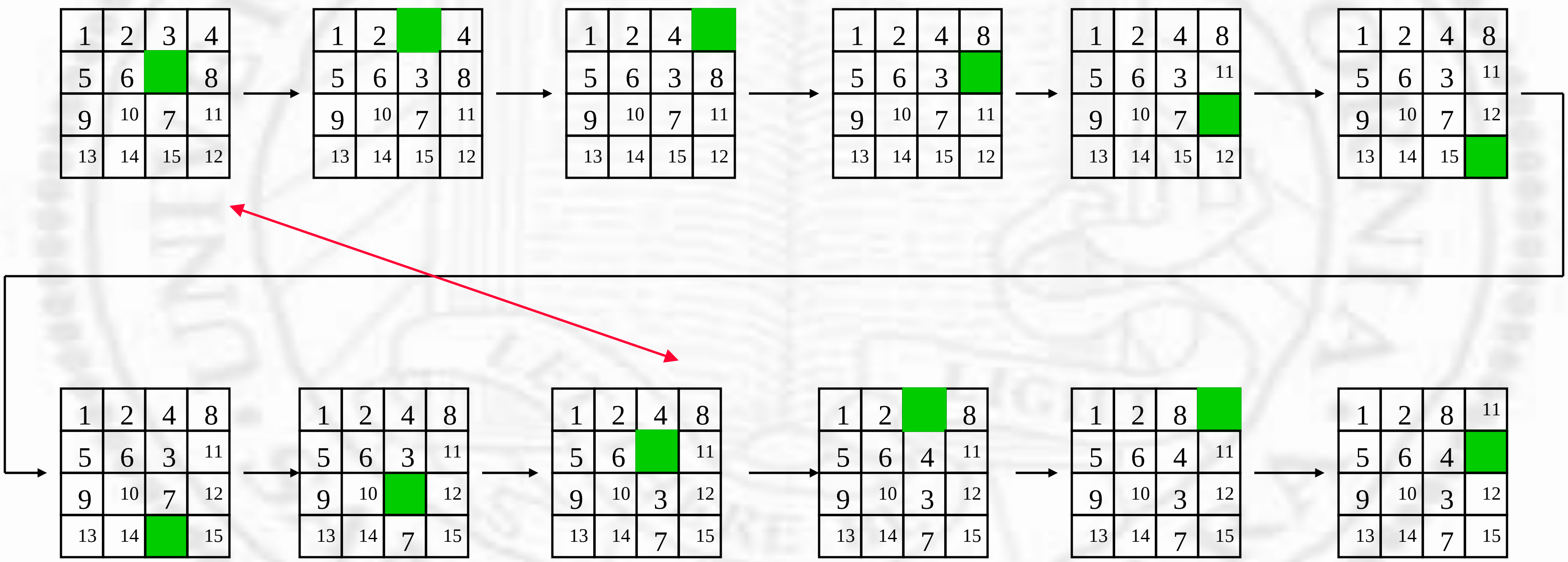
Branch and Bound

- The 15 puzzle
  - Input: 15 numbered tiles (1-15) and a blank one on a 4x4 square
  - Legal moves: a numbered title, which is a 4-neighbor of the blank title, can move into the location of the blank tile
  - Output: an ordered final configuration

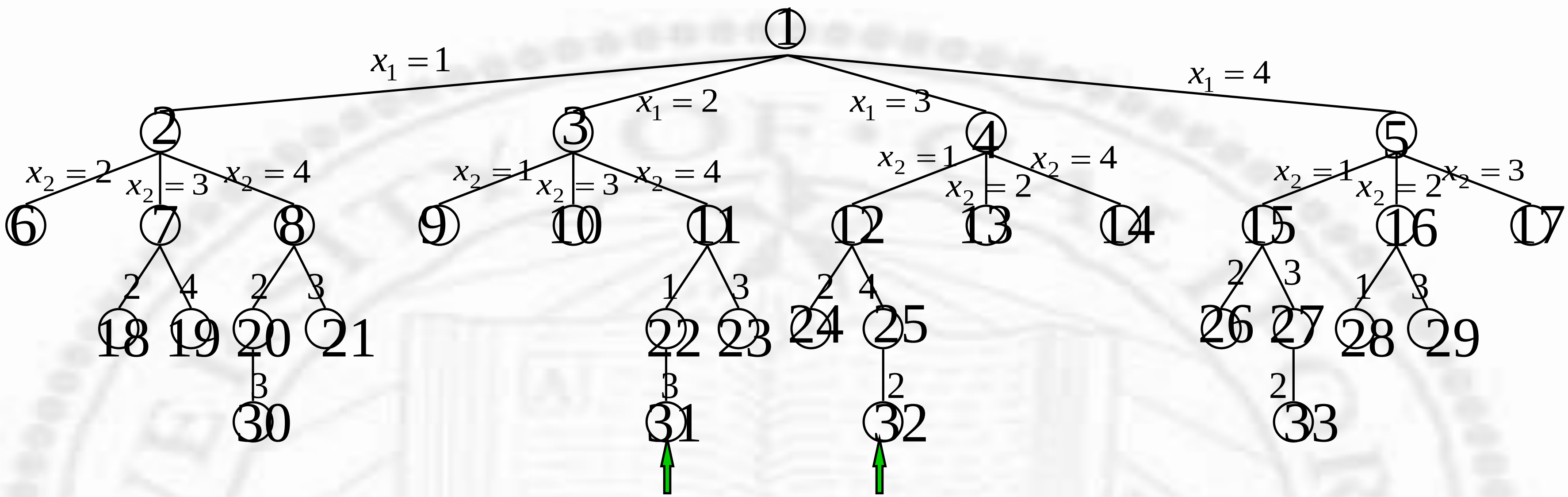
- Depth first search
  - up, right, down, left order
  - do not undo the last move



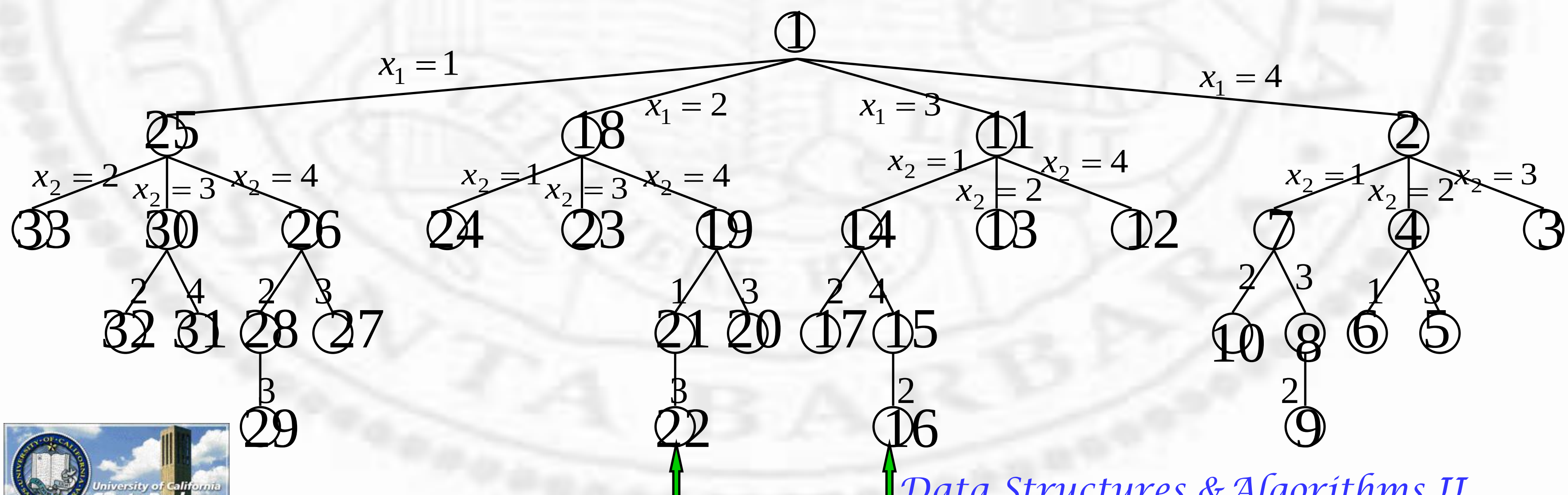
- Depth first search
  - Always go left as deep as possible
  - The left path
    - may be very long for large search problems
    - may contain a cycle and never terminate
    - the optimal solution may not be found there
    - may not be the natural way of searching
      - e.g. playing chess (many alternatives are considered simultaneously with a small number of look-ahead)

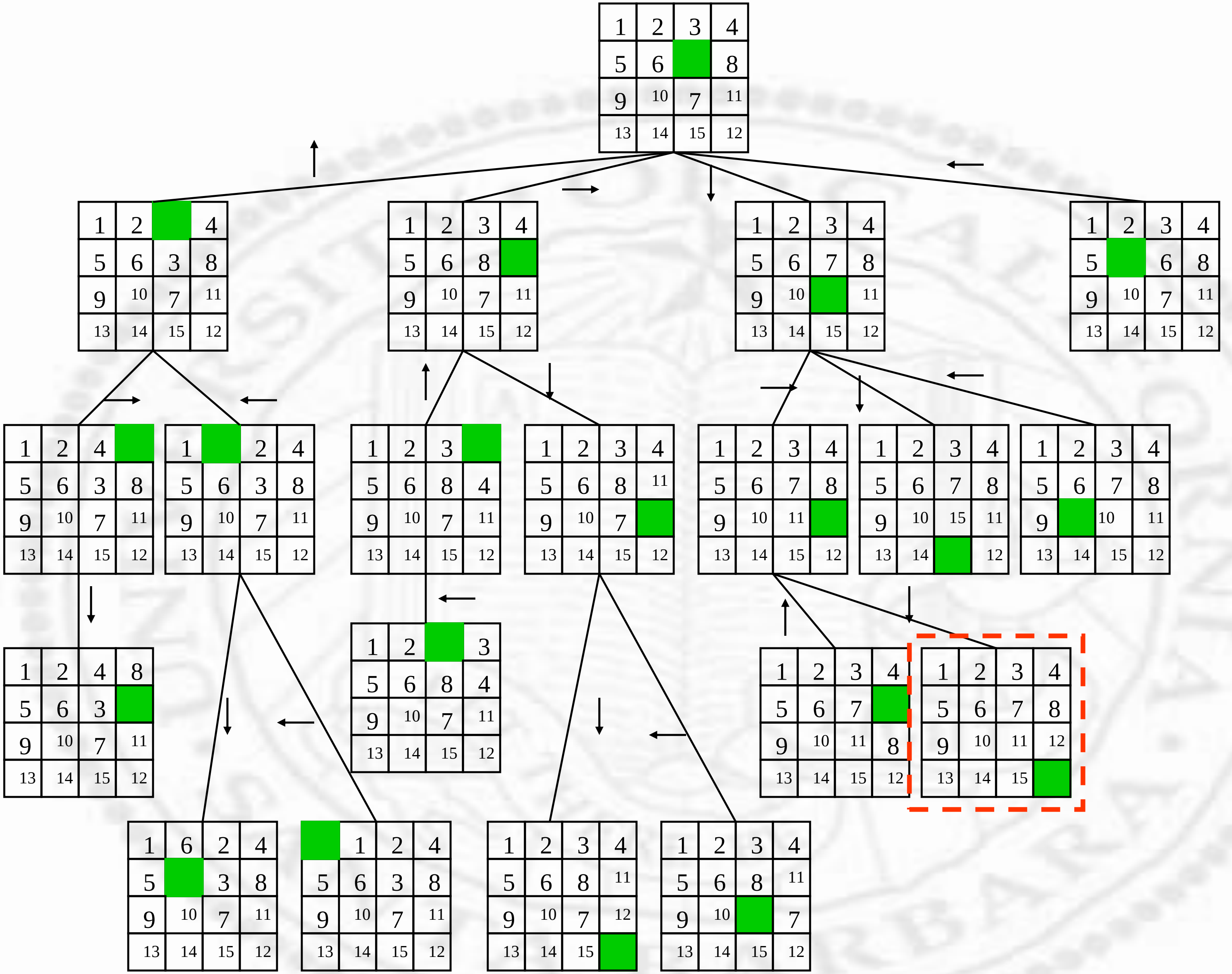
- Breadth search
  - An E-node stays an E-node until all its children are generated
  - How to organize all the children?
    - FIFO (breadth first)
    - FILO (D-search)

FIFO breadth first search for 4-queen



FILO breadth first search for 4-queen

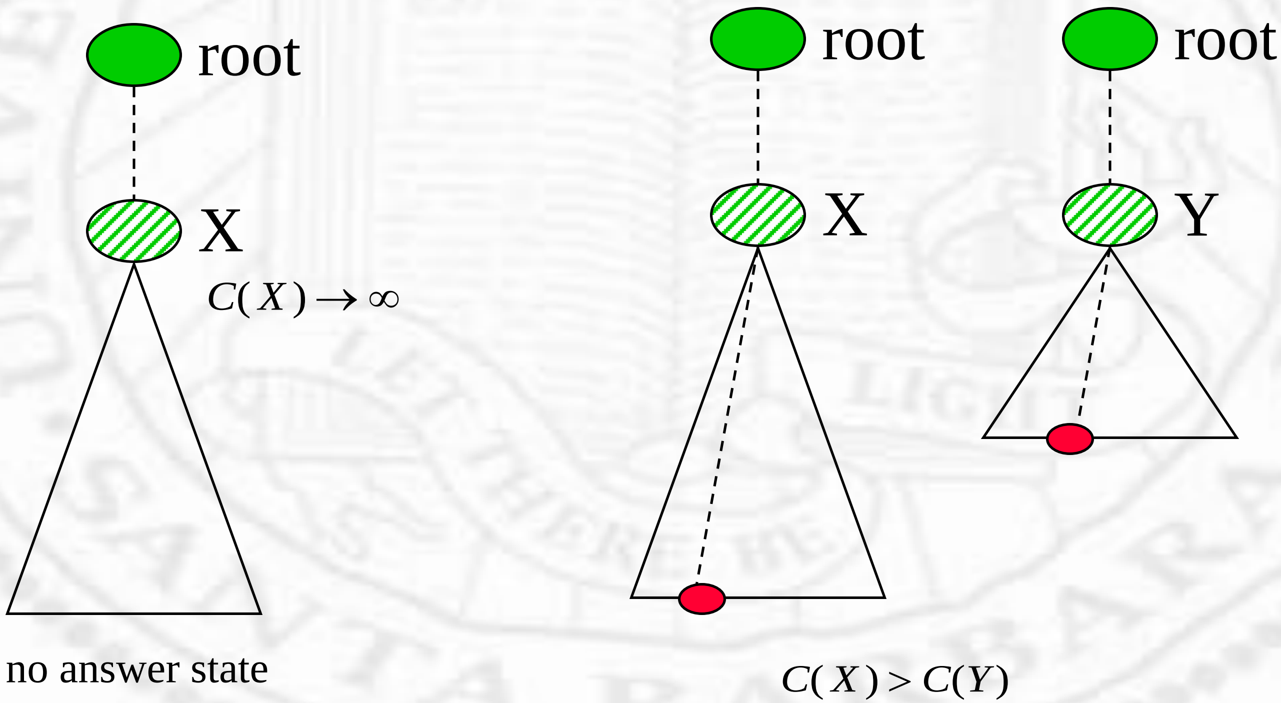




1 2 3 4 5 6 8 9 10 7 11 13 14 15 12
1 2 4 5 6 3 8 9 10 7 11 13 14 15 12
1 2 3 4 5 6 8 9 10 7 11 13 14 15 12
1 2 3 4 5 6 7 8 9 10 11 13 14 15 12
1 2 3 4 5 6 8 9 10 7 11 13 14 15 12
1 2 4 5 6 3 8 9 10 7 11 13 14 15 12
1 2 4 5 6 3 8 9 10 7 11 13 14 15 12
1 2 3 5 6 8 4 9 10 7 11 13 14 15 12
1 2 3 4 5 6 8 11 9 10 7 13 14 15 12
1 2 3 4 5 6 7 8 9 10 11 13 14 15 12
1 2 3 4 5 6 7 8 9 10 15 11 13 14 12
1 2 3 4 5 6 7 8 9 10 11 13 14 15 12
1 2 4 8 5 6 3 9 10 7 11 13 14 15 12
1 2 3 5 6 8 4 9 10 7 11 13 14 15 12
1 2 3 4 5 6 7 9 10 11 8 13 14 15 12
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
1 6 2 4 5 3 8 9 10 7 11 13 14 15 12
1 2 4 5 6 3 8 9 10 7 11 13 14 15 12
1 2 3 4 5 6 8 11 9 10 7 12 13 14 15
1 2 3 4 5 6 8 11 9 10 7 13 14 15 12

- Problems with depth, breadth searches
  - They have *rigid rules*
- If during the search it becomes apparent that a solution is near, search pattern should be altered to the path(s) that lead quickly to that solution

- Q: How to know which path is "better" and should be explored first?
- A: A cost function that is based on the cost of the solution states reachable



- Q: How do we know the cost of a partial problem state *X*?
- A: Trace out the subtree rooted at *X*
- But then the subtree has already been explored, no use of the cost function

⇒ Ideal cost function is useless, need an *estimate* without generating the subtree

- How to generate an *educated* guess?
  - current position: cost of reaching current node from root (*known*)
  - estimated distance to an answer state: cost of reaching a solution (*unknown*)

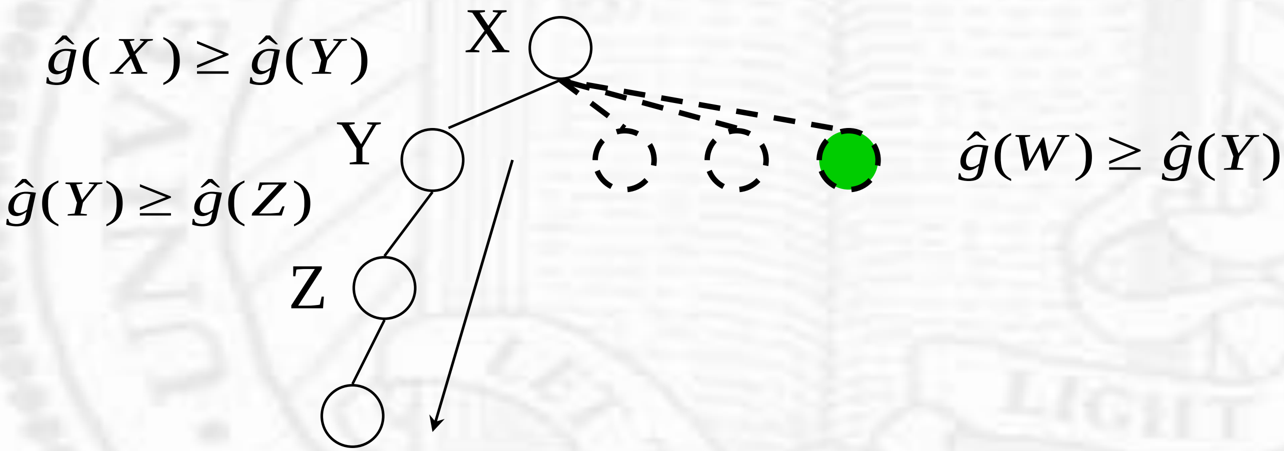
$\hat{C}(X) = f(h(X)) + \hat{g}(X)$

$h(X)$: distance from root to X

$f(.)$: nondecreasing function

$\hat{g}(X)$: *estimated* cost to reach an answer from X

- Use only $\hat{g}(X)$
  - normally, we expect search to make progress, i.e., exploring deeper into the state space tree leads us closer to an answer

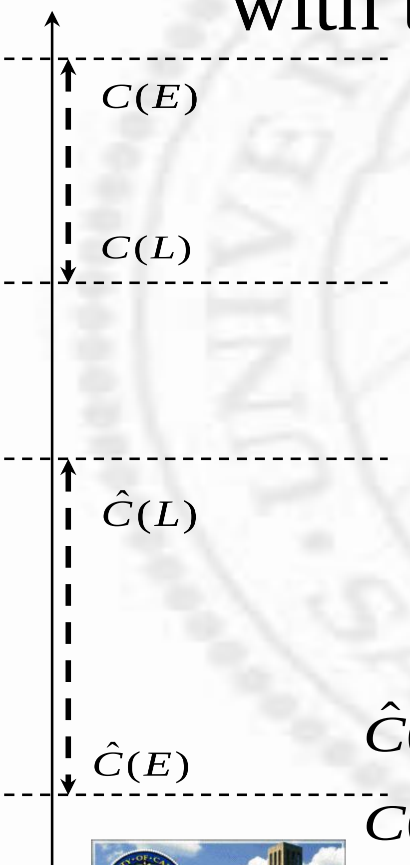


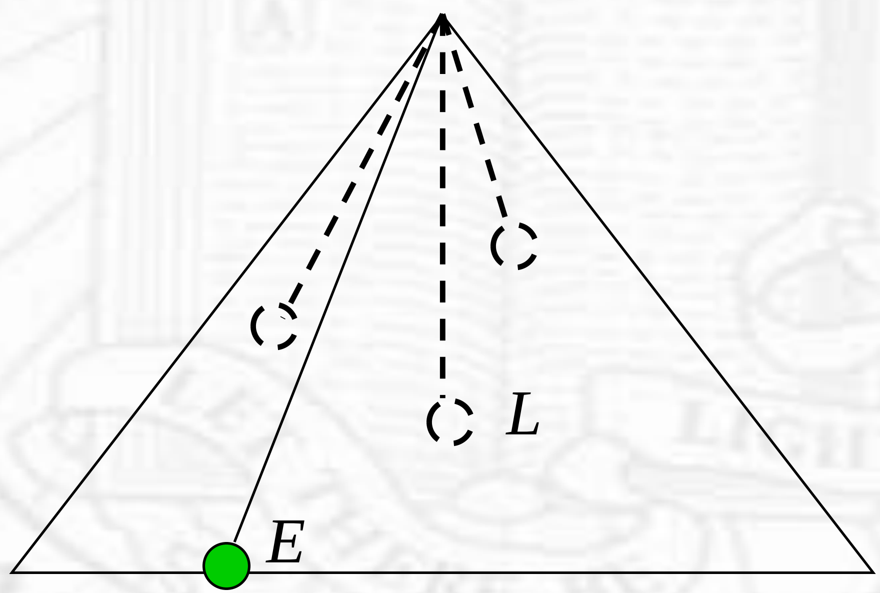
  - may turn into a depth first search

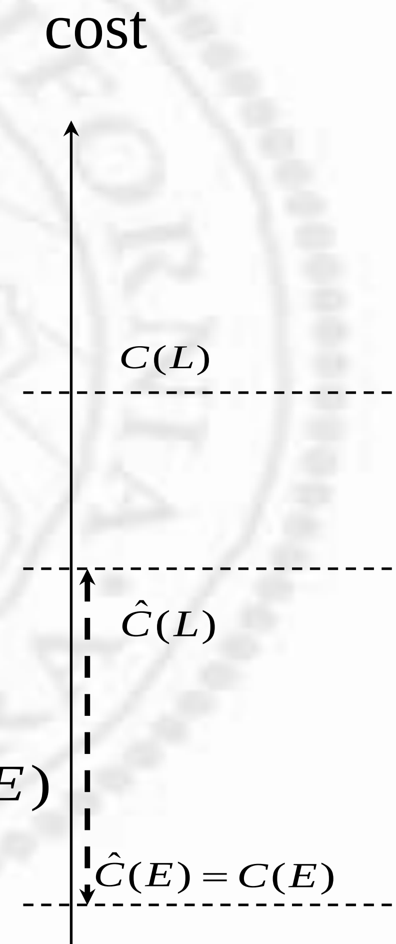
- Use only *f(h(x))*
  - – level ↓
  - – distance to root ↓
  - – effort to get there *h(x)* ↓
  - – cost *f(h(x))* ↓
  - – expand earlier
  - ⇒breadth first search
- A *balanced* cost function should have both components

- A search based on the least-cost-first principle is called *LC branch-and-bound* search
  - Can be more efficient
  - Will it find the optimal solution?

- **Proposition**: If the estimated cost function never over-estimates the cost, then LC branch-and-bound find the answer state with the minimum cost







$$\hat{C}(E) = f(h(x)) + \hat{g}(x) = f(h(x)) + 0 = C(E)$$

$$C(E) = \hat{C}(E) \le \hat{C}(L) \le C(L)$$

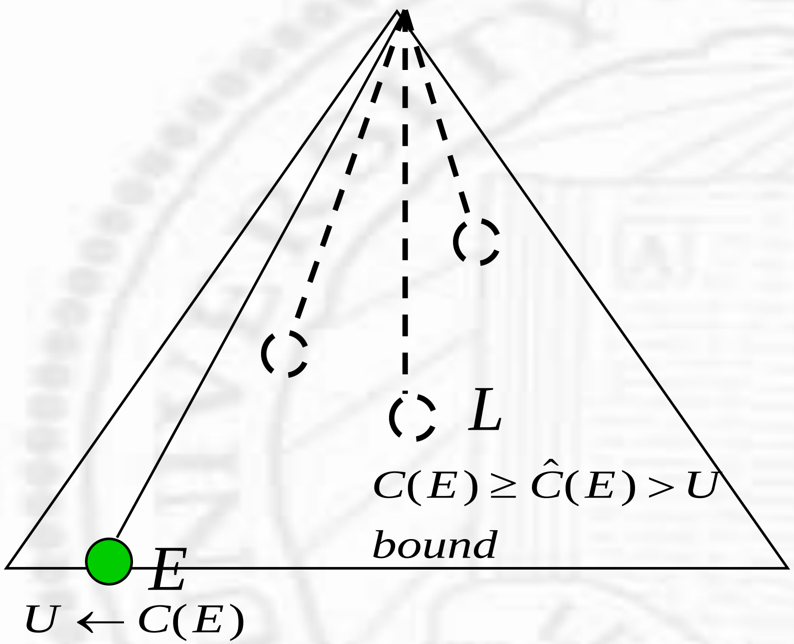
Trade-off in Search Pattern

- Depth, breadth, etc.
  - rigid ordering
  - applicable to all
  - no need for a cost function
- LC branch-and-bound
  - more flexible
  - applicable if a good cost function is available

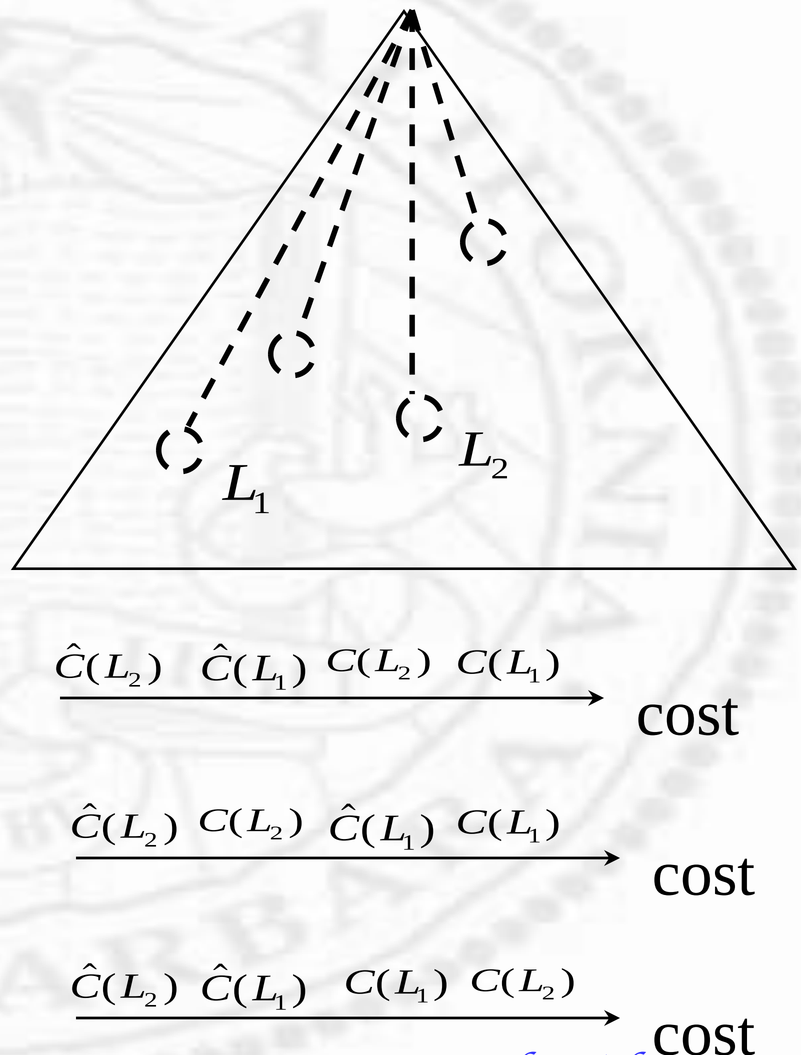
Trade-off in Bounding

- Depth first
  - feasibility
  - optimality
- Breadth and LC branch-and-bound
  - feasibility
  - optimality (?)

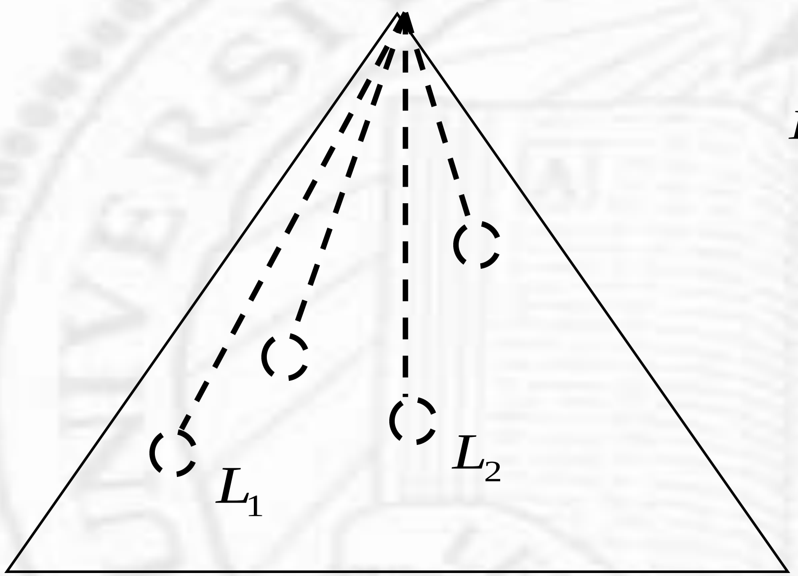
- In back tracking

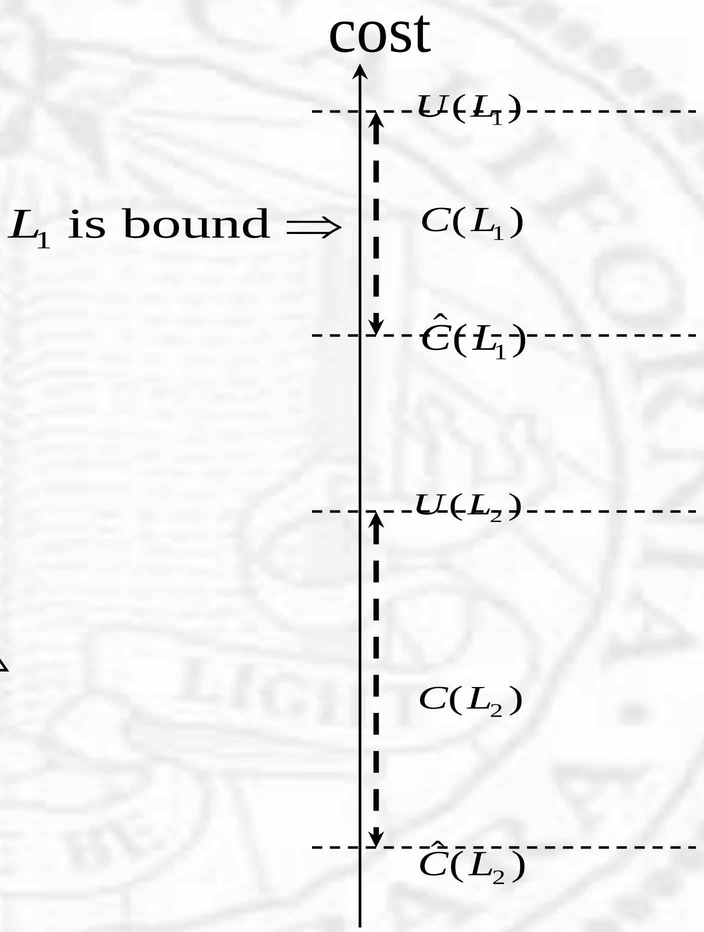


- In breadth and LC branch and bound



Bounding in Brach-and-Bound

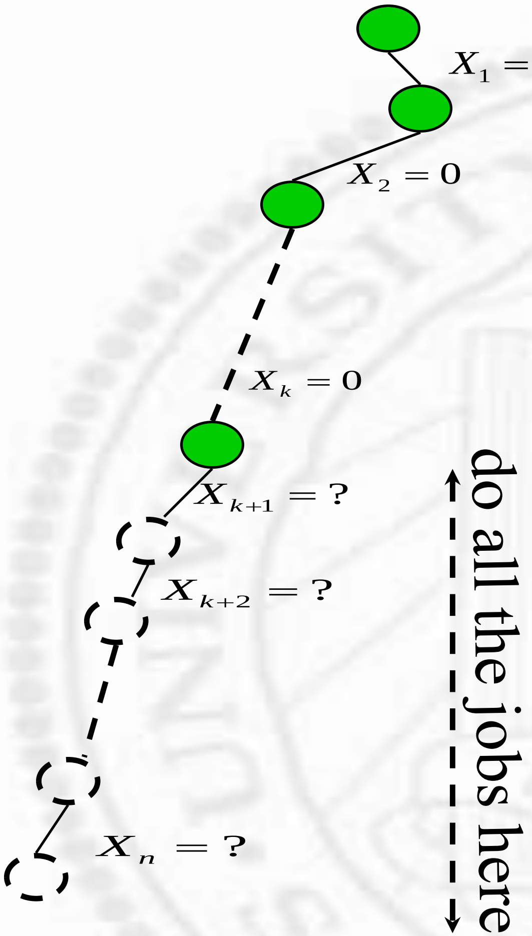




Job Sequencing with Deadline

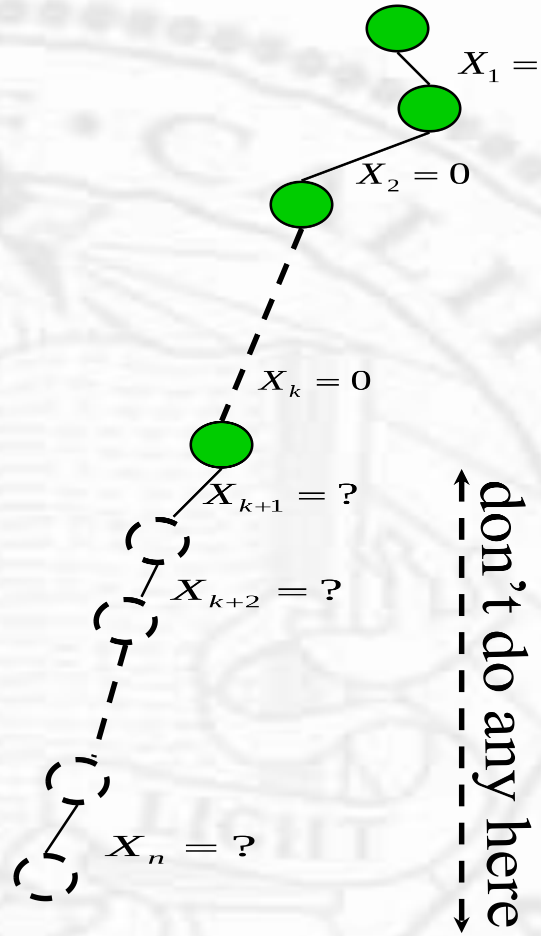
- Input:
  - a set of *n* jobs with deadlines and penalty
  - a single machine to execute all jobs
- Output: A subset of jobs, each completed before the deadline with least penalty

- Lower bound



$$\sum_{1 \le i \le k} P_i(1 - X_i) = \sum_{\substack{i \le k \\ i \notin J}} P_i$$

- Upper bound



$$\sum_{1 \le i \le k} P_i(1 - X_i) + \sum_{i > k} P_i = \sum_{\substack{i \le k \\ i \notin J}} P_i + \sum_{i > k} P_i$$

$$k,\quad \overbrace{\sum_{\substack{i<k\\ i\notin J}} P_i}^{\hat{C}},\quad \overbrace{\sum_{i\notin J} P_i}^{\hat{U}}$$

$X_k = 1$

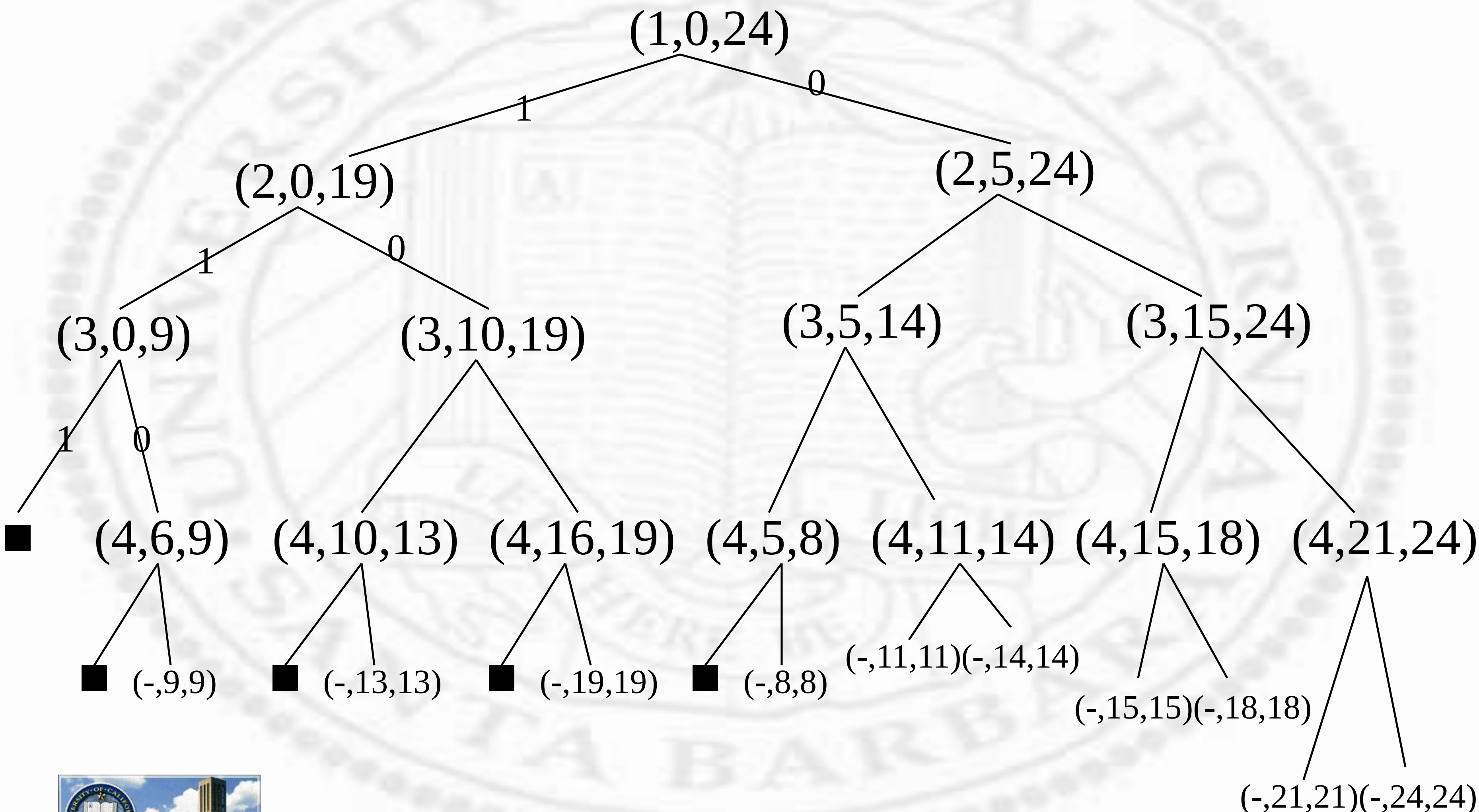
$$k+1, \hat{C}' = \hat{C}, \hat{U}' = \hat{U} - P_k$$

$X_k = 0$

$$k+1, \hat{C}' = \hat{C} + P_k, \hat{U}' = \hat{U}$$

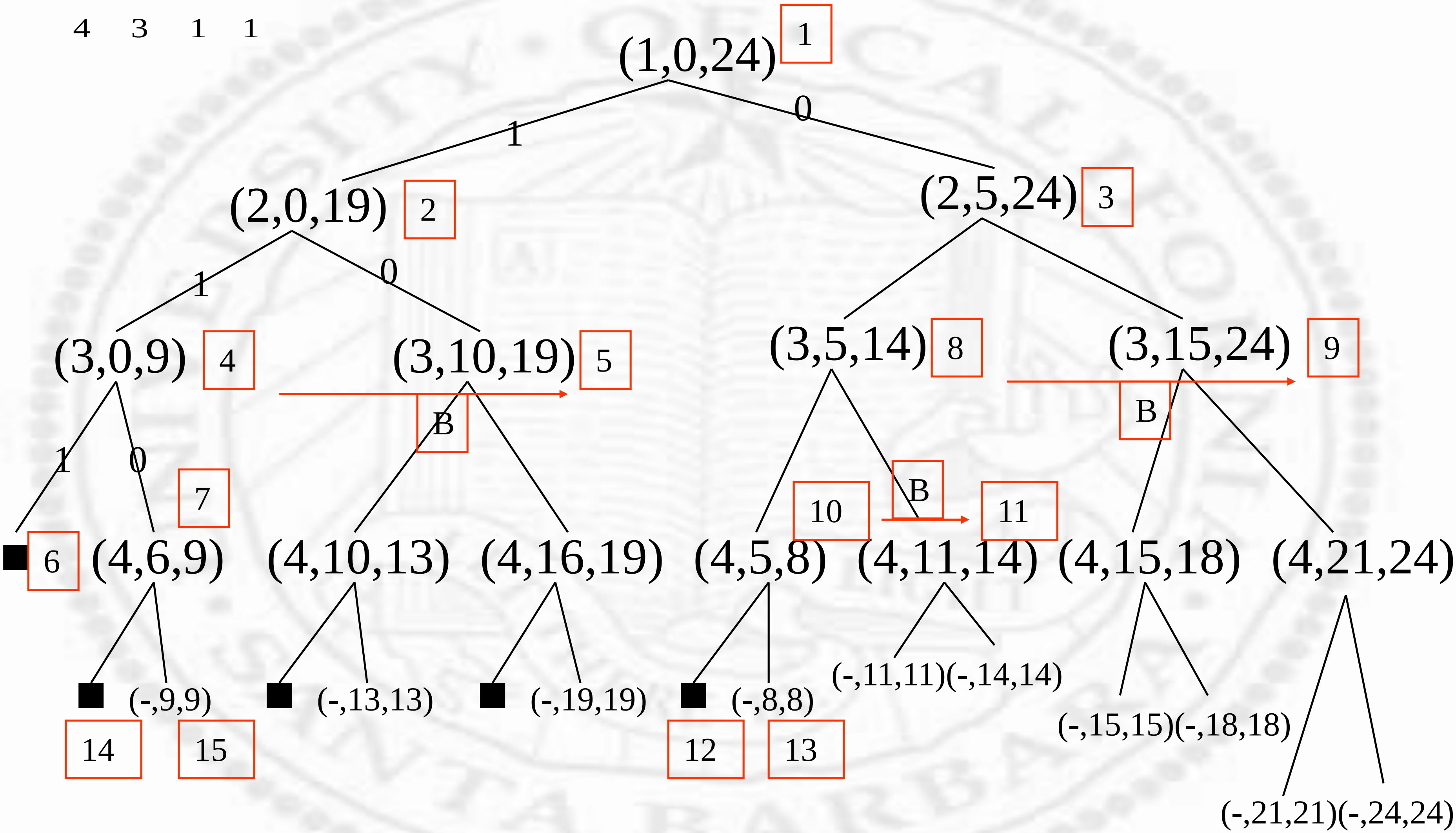
	p	d	t
1	5	1	1
2	10	3	2
3	6	2	1
4	3	1	1

$$k,\quad \overset{\hat{C}}{\sum_{\substack{i<k \\ i \notin J}} P_i},\quad \overset{\hat{U}}{\sum_{i \notin J} P_i}$$



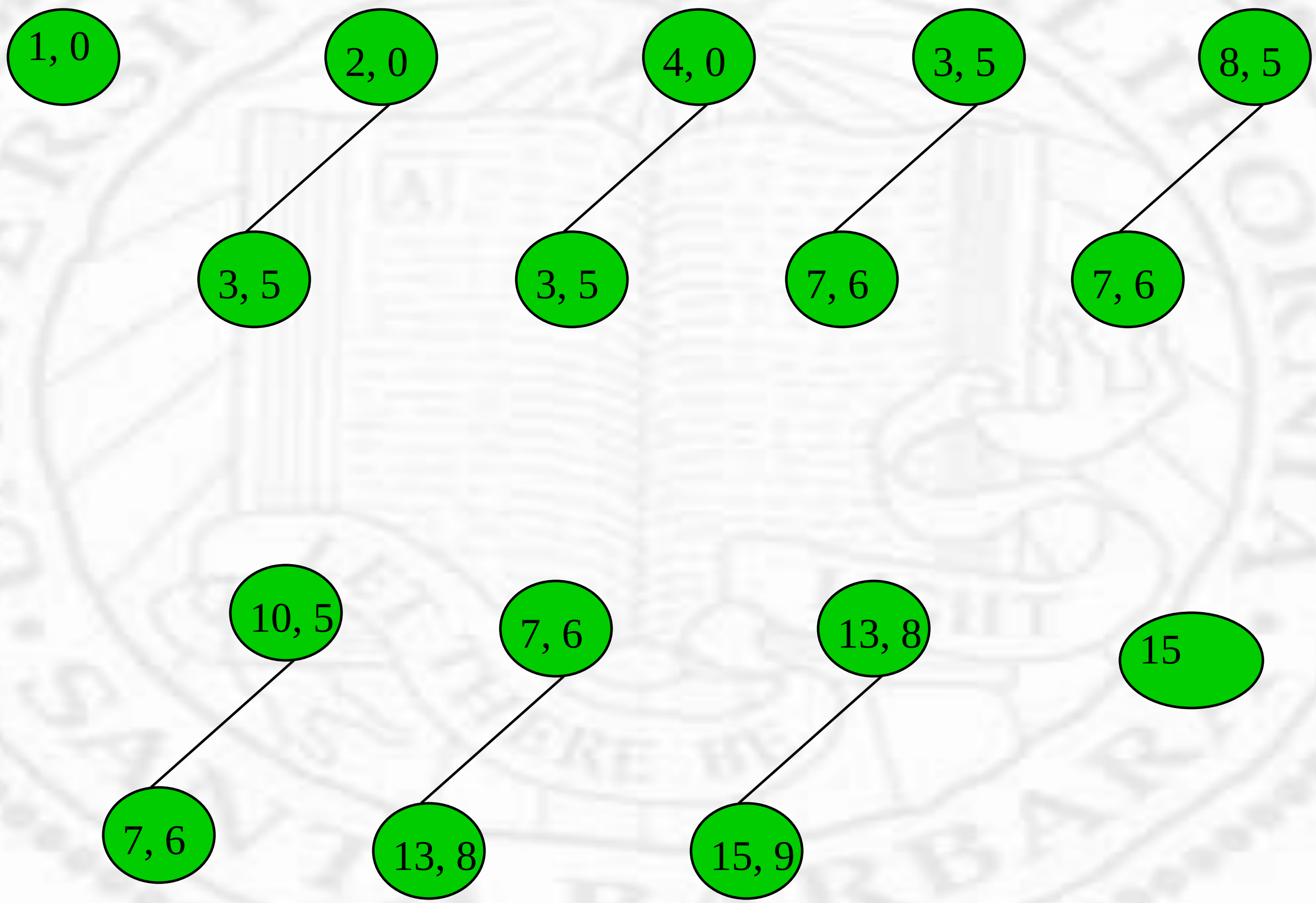
	p	d	t
1	5	1	1
2	10	3	2
3	6	2	1
4	3	1	1

$$k, \quad \overbrace{\sum_{\substack{i<k \\ i \notin J}} P_i}^{\hat{C}}, \quad \overbrace{\sum_{i \notin J} P_i}^{\hat{U}}$$

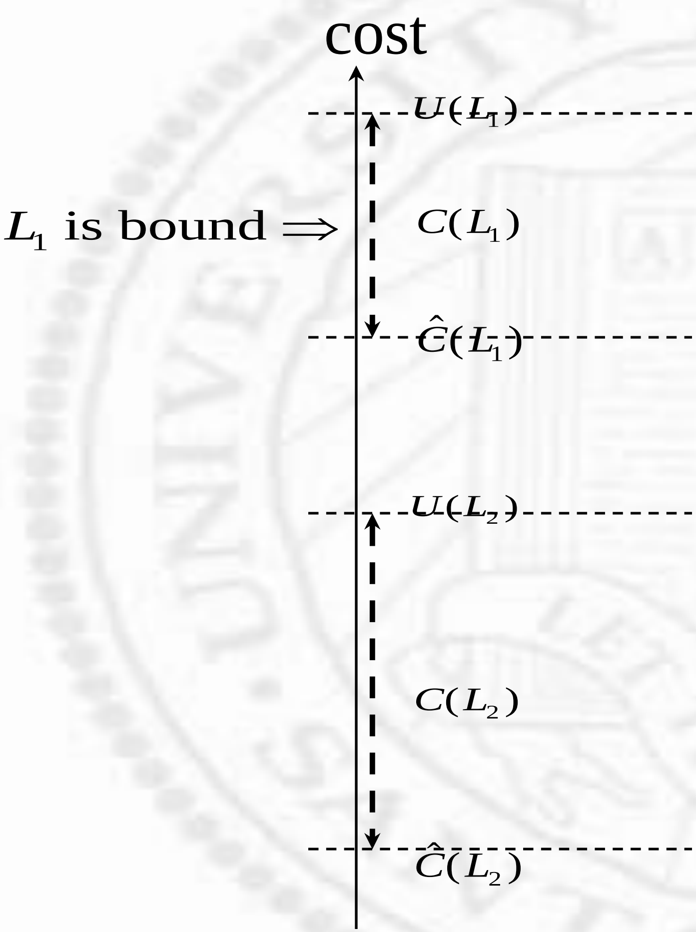


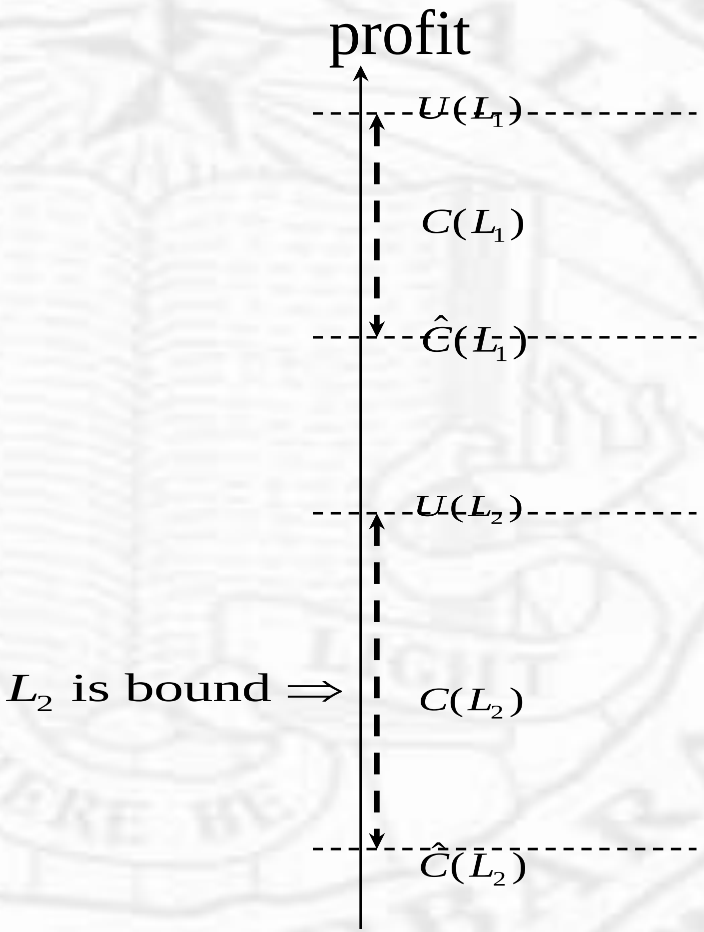
- Store candidates in a partially-ordered tree (a heap)

(node number, c)



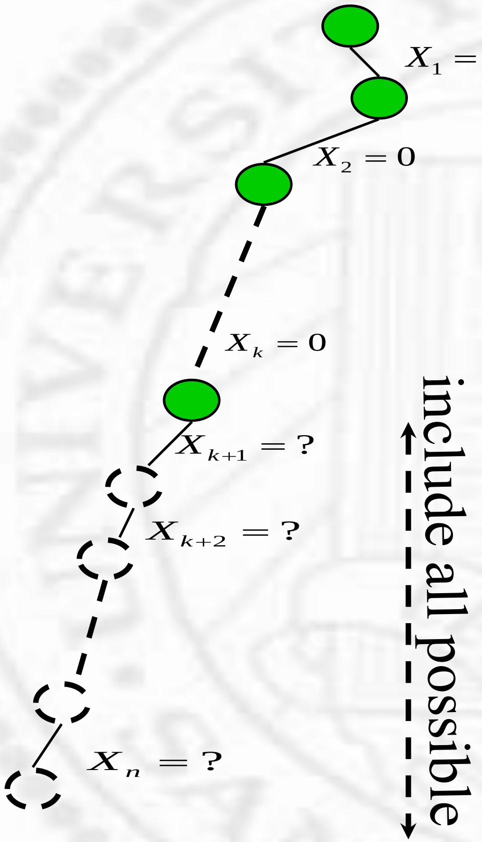
- Minimization problems
- Maximization problems



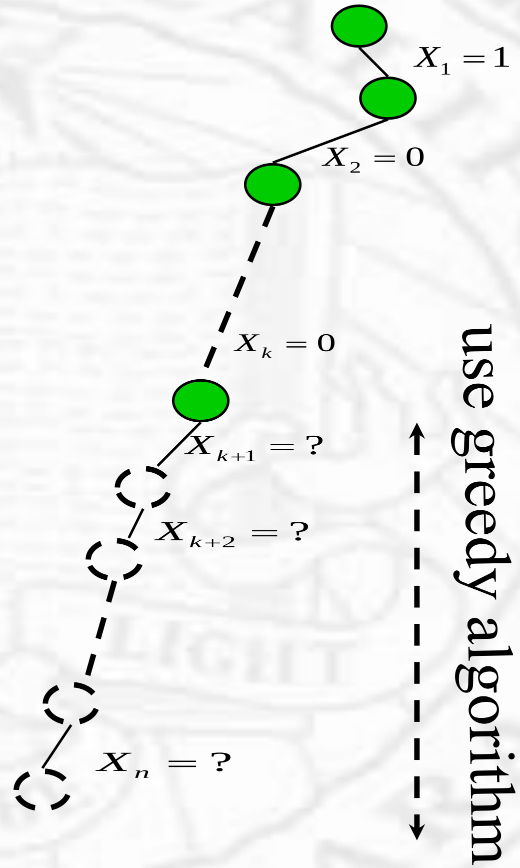


0/1-Knapsack

- Lower bound



- Upper bound



$$k, \quad \overbrace{\sum_{\substack{i<k \\ i \in sack}} P_i + \text{integer}}^{\hat{C}}, \quad \overbrace{\sum_{\substack{i<k \\ i \in sack}} P_i + \text{greedy}}^{\hat{U}}$$

$X_k = 1$

$$k+1, \hat{C}' = \hat{C}, \hat{U}' = \hat{U}$$

$X_k = 0$

$$k+1, \hat{C}' \text{ from integer}, \hat{U}' \text{ from greedy}$$

n=4 W=(2, 4, 6, 9)

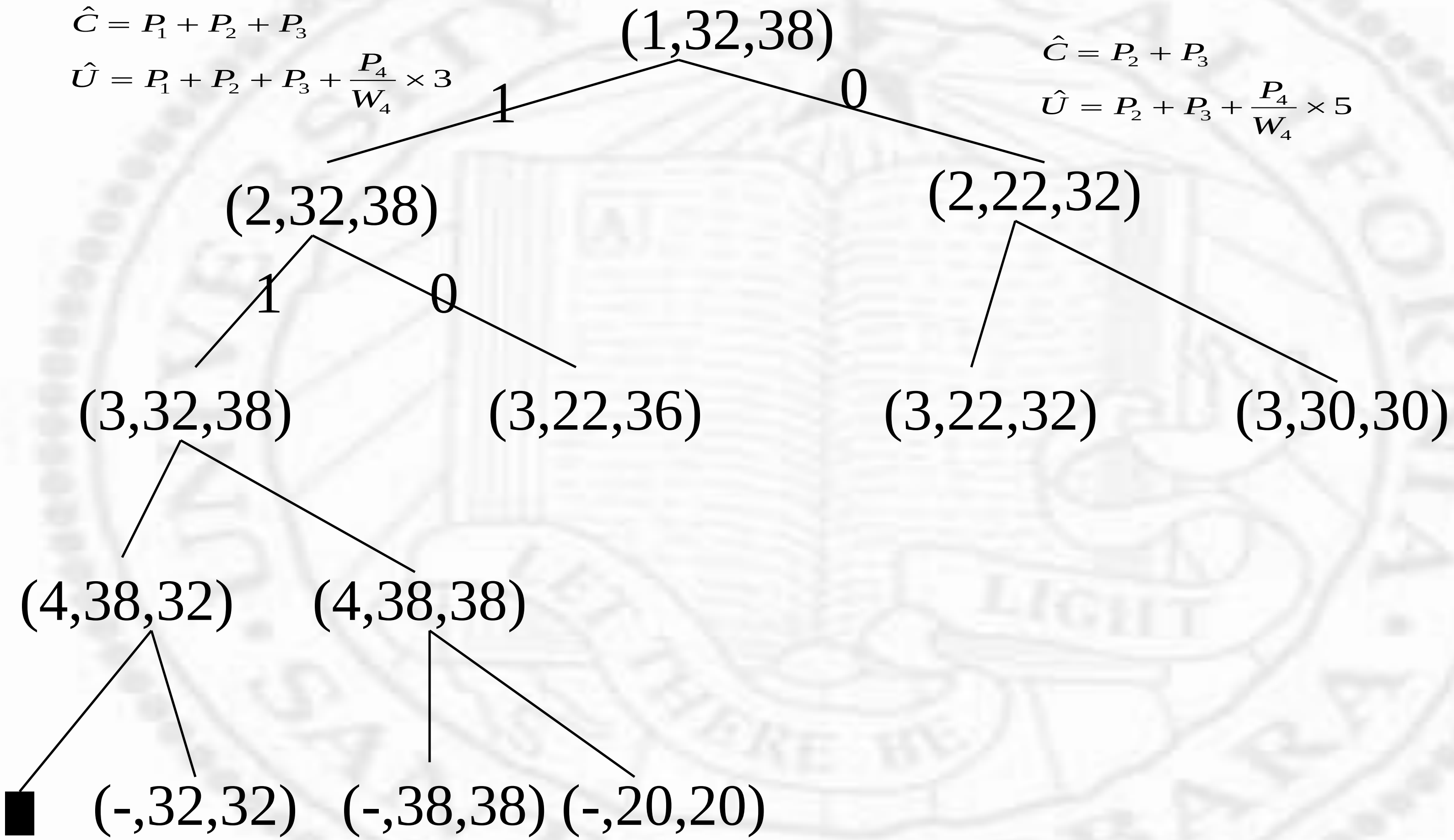
M=15 P=(10,10,12,18)

$$\hat{C} = P_1 + P_2 + P_3$$

$$\hat{U} = P_1 + P_2 + P_3 + \frac{P_4}{W_4} \times 3$$

$$\hat{C} = P_2 + P_3$$

$$\hat{U} = P_2 + P_3 + \frac{P_4}{W_4} \times 5$$



n=4 W=(2, 4, 6, 9)

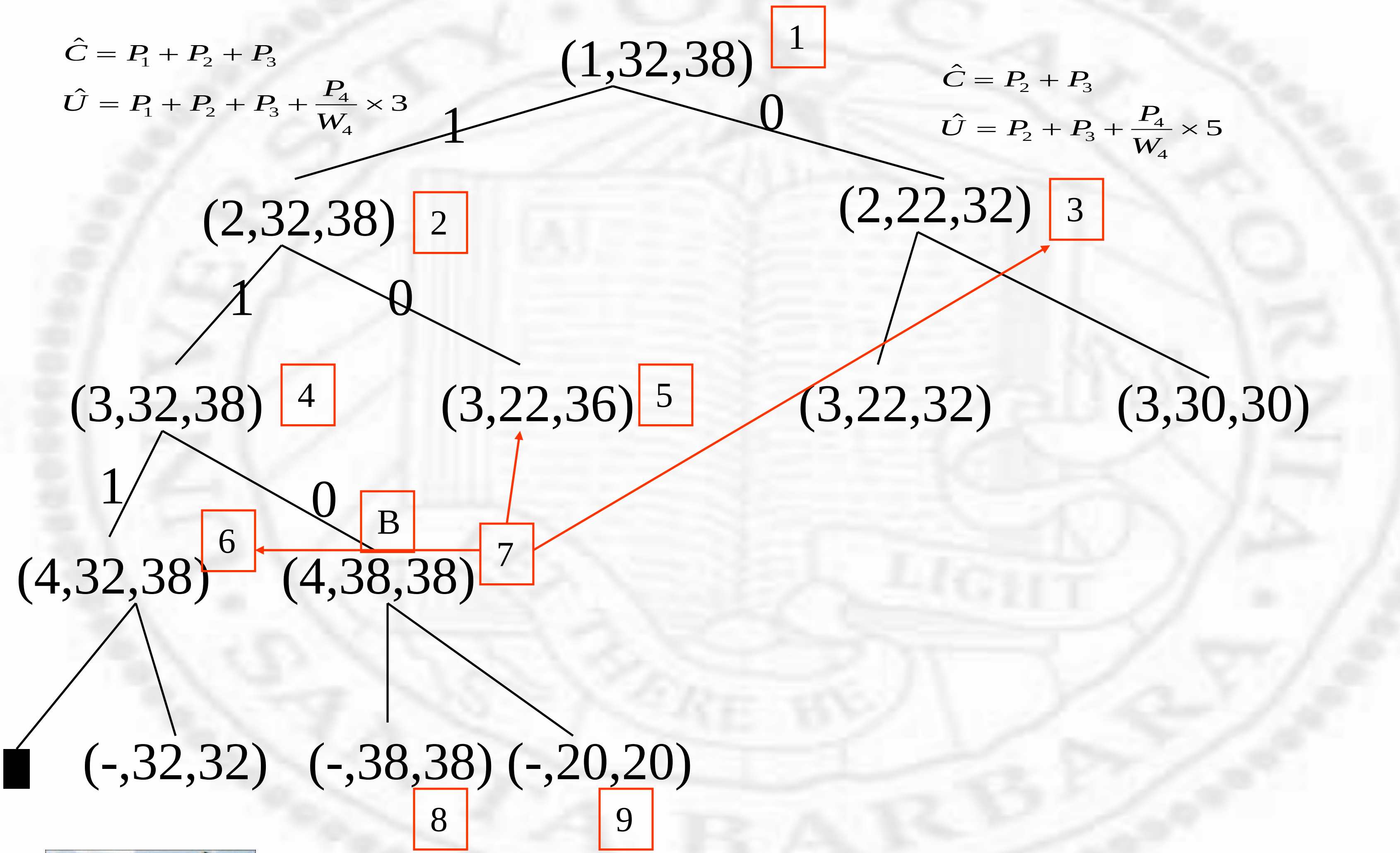
M=15 P=(10,10,12,18)

$$\hat{C} = P_1 + P_2 + P_3$$

$$\hat{U} = P_1 + P_2 + P_3 + \frac{P_4}{W_4} \times 3$$

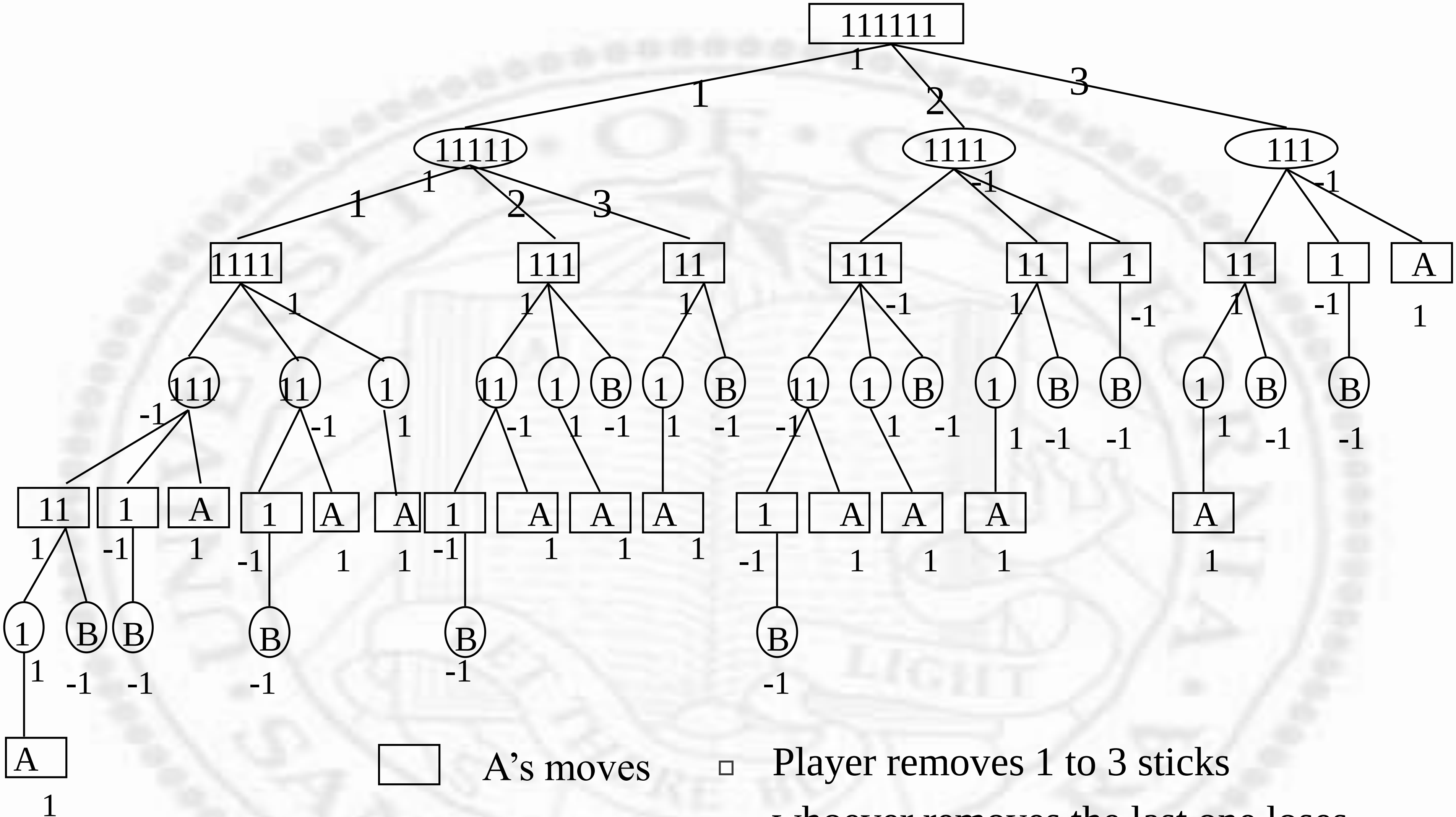
$$\hat{C} = P_2 + P_3$$

$$\hat{U} = P_2 + P_3 + \frac{P_4}{W_4} \times 5$$



Branch-and-Bound in Game Trees

- Explore many possibilities
- Evaluate their relative merits
- Look ahead finite steps
- *Opposite goals at alternate steps*



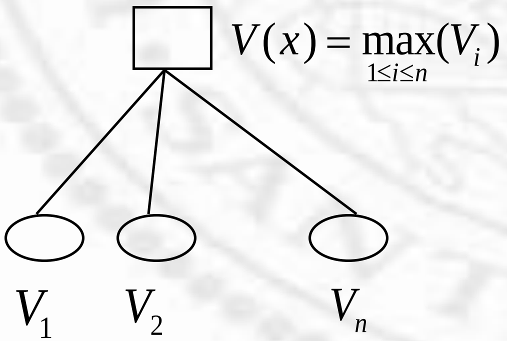
A's moves

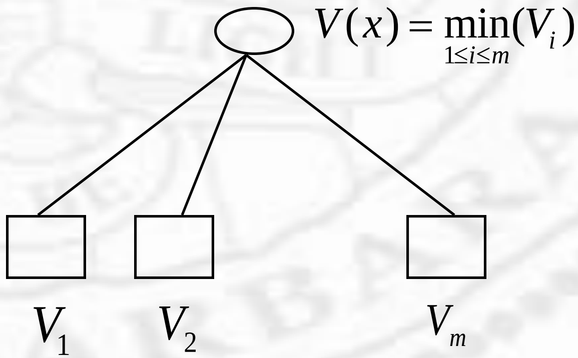
B's moves

- Player removes 1 to 3 sticks
- whoever removes the last one loses
- 1 if A wins, -1 if A loses

- Game tree: all possible instances of a finite games
- Game instance: a particular branch from root to a terminal board configuration
- Terminal configuration:

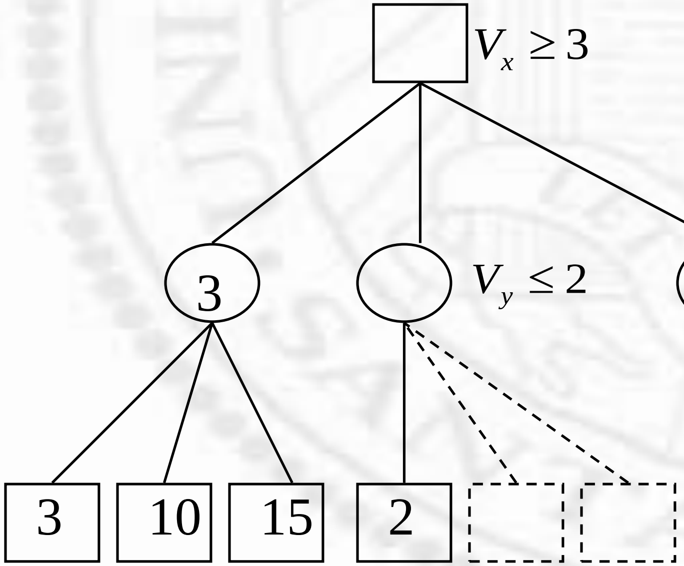
$$V(x)=\begin{cases}1 & \text{if x is a winning configuration for A}\\ 0 & \text{if x is a draw configuration for A}\\ -1 & \text{if x is a losing configuration for A}\end{cases}$$



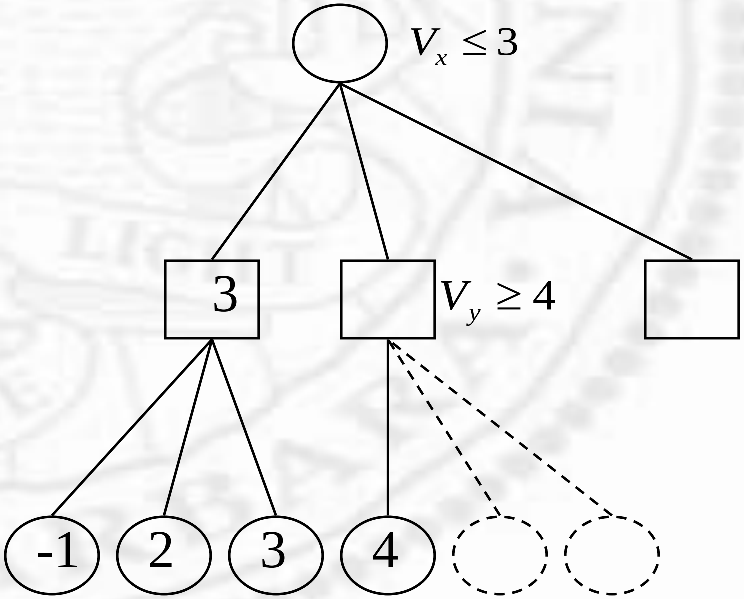


- For chess
  - not possible to generate the whole game tree
  - only a few levels can be generated and evaluated
  - an intelligent evaluation strategy is needed
  - the move strategy is still the same
    - A to maximize the value
    - B to minimize the value

- Alpha cutoff
  - min V at max position
  - if the value of a min position is less than the parent's alpha, the min position need not be explored further



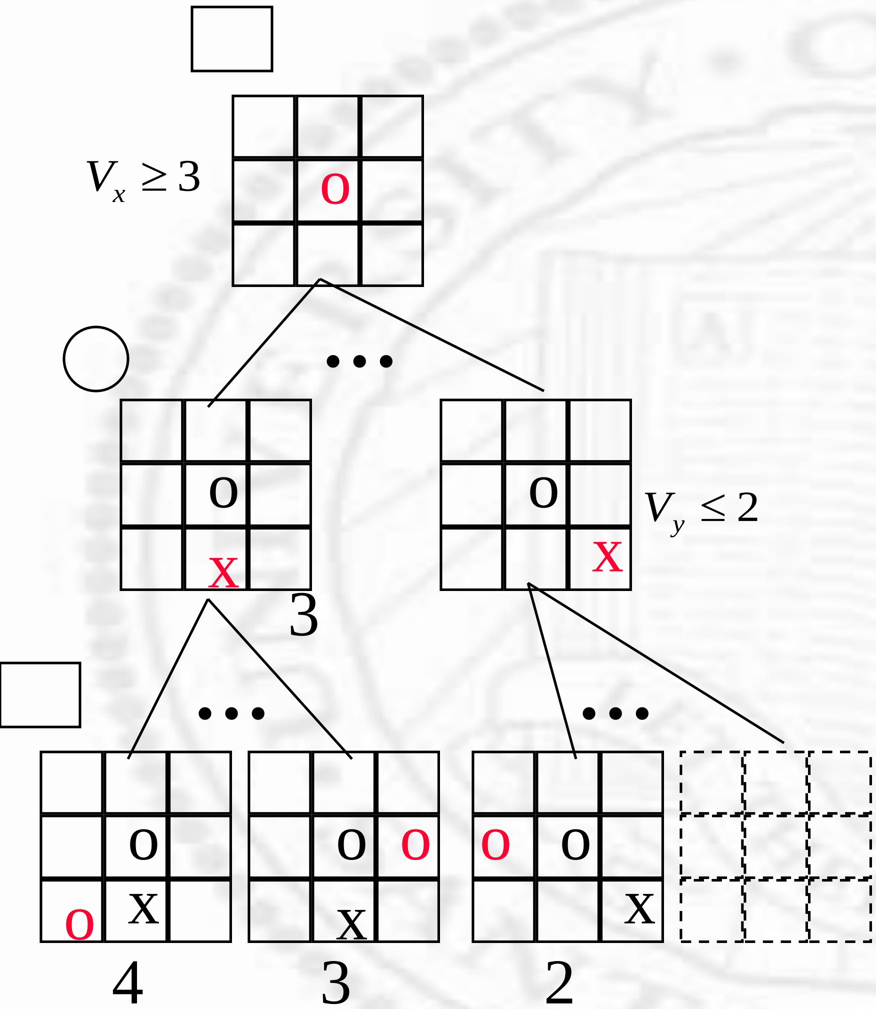
- Beta cutoff
  - max V at min position
  - if the value of a max position is greater than the parent's beta, the max position need not be explored further



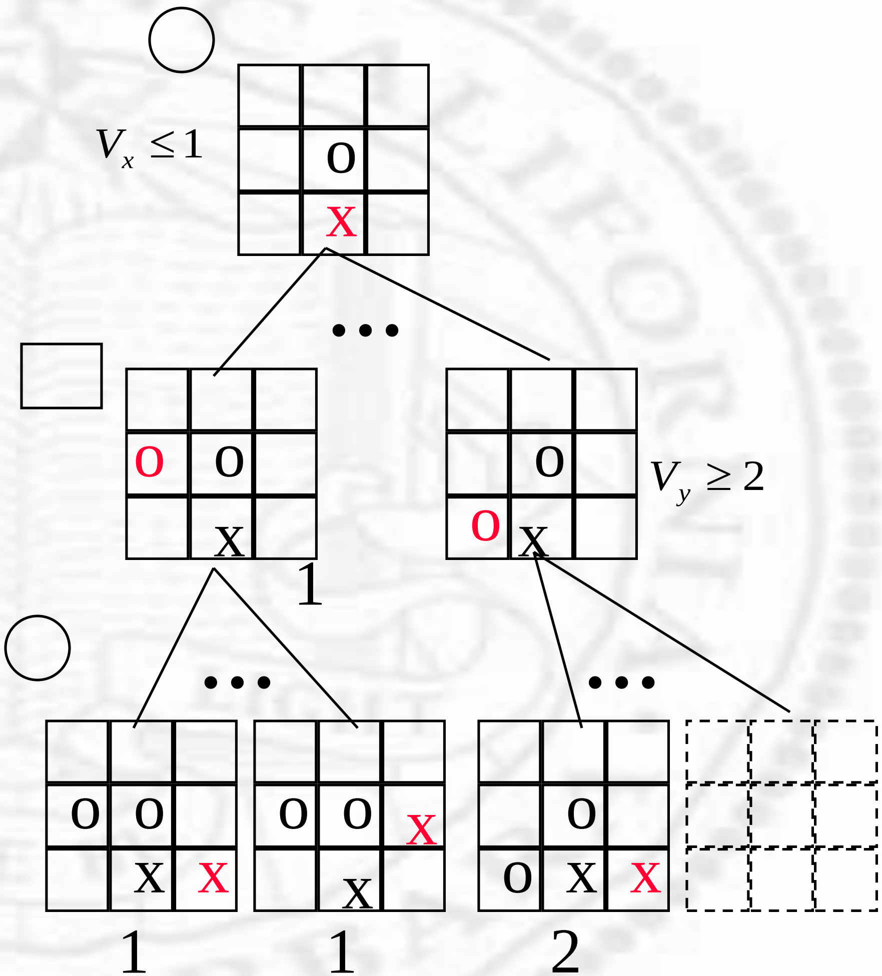
- Example tic-tac-toe
  – evaluation function

- 6 ways to win for (0) - 4 ways to win for (x)=2 in (0)'s favor

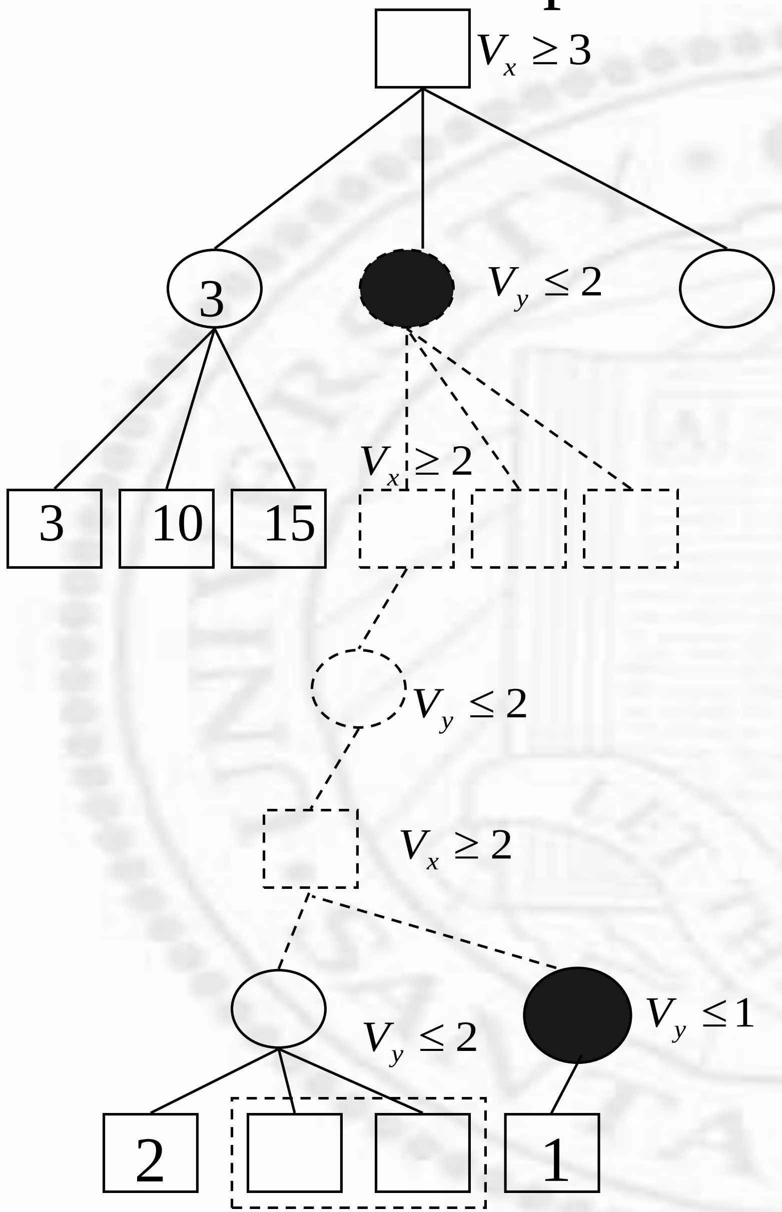
- Alpha cutoff



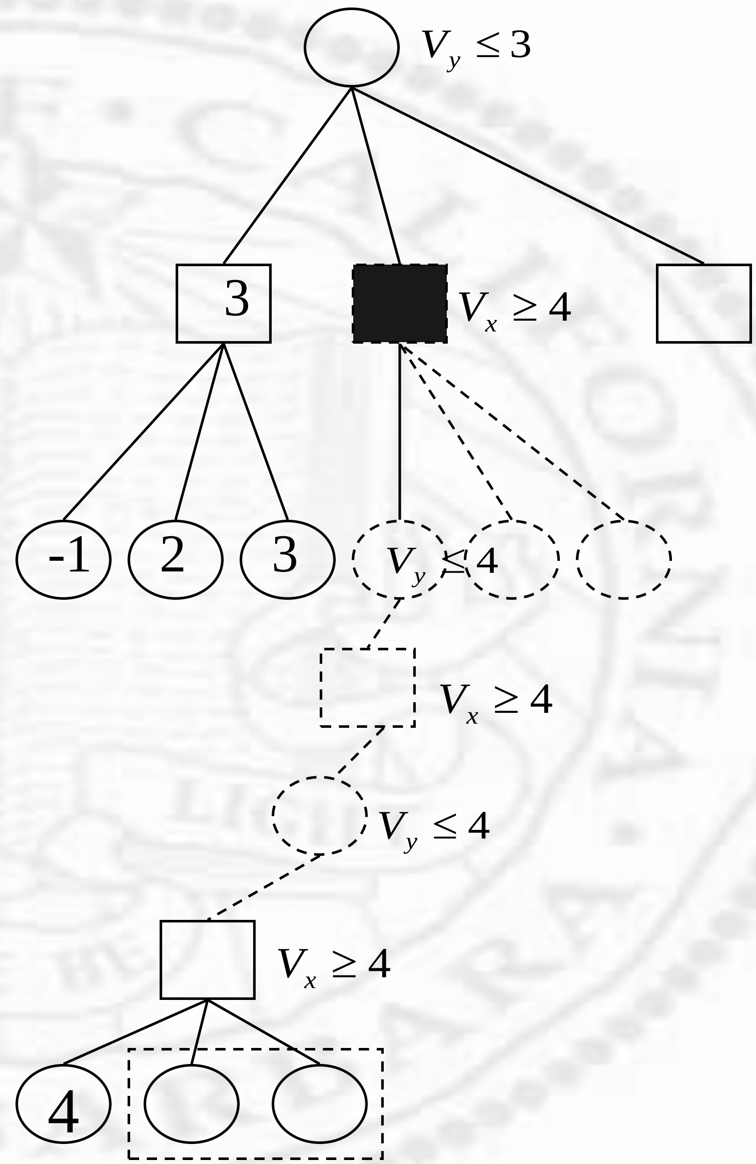
- Beta cutoff



- Multi-level Alpha cutoff



- Multi-level Beta cutoff



Black: bounded