III. SEARCH

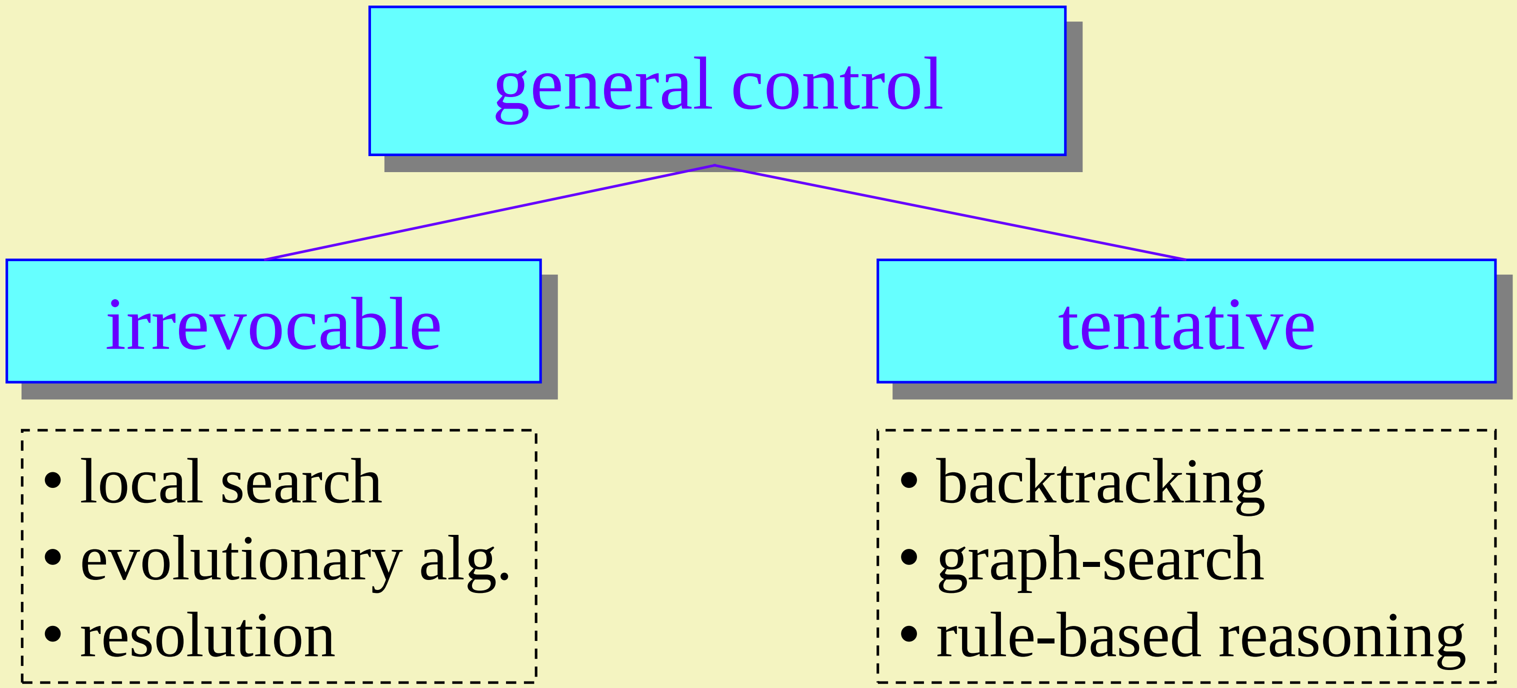
DATA := *initial value*
while ¬*termination condition*(DATA) **loop**
    SELECT R FROM *rules that can be applied*
    DATA := R(DATA)
endloop

Levels of control

control strategy

- general
  - quite independent of the problem and its model
- model dependent
  - independent of the problem but based on the way of modeling of the problem
- heuristic
  - special knowledge about of the problem domain that is built into the search system directly (not included in the model)

General control strategies

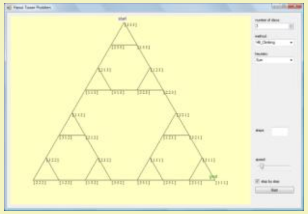


1. Local search

- The global workspace of a local search contains only one (current) node of the representation graph with its small environment. Initially this current node is the start node. The search stops if the current node is a goal node or the search could not step over.
- In each step the current node is exchanged for its better child by a searching rule.
- The control strategy uses an evaluation (objective, fitness, heuristic) function to select a better child node. This function tries to estimate to what extent a node promises the achievement of the goal. This function involves some heuristics.

DATA := *initial value*
while ¬*termination condition*(DATA) **loop**
  SELECT R FROM *rules that can be applied*
  DATA := R(DATA)
endloop

Hill climbing method



- It only stores the current node and its parent that is the former current node.
- In each step the best child (with the smallest value) of the current node is selected except for the parent.

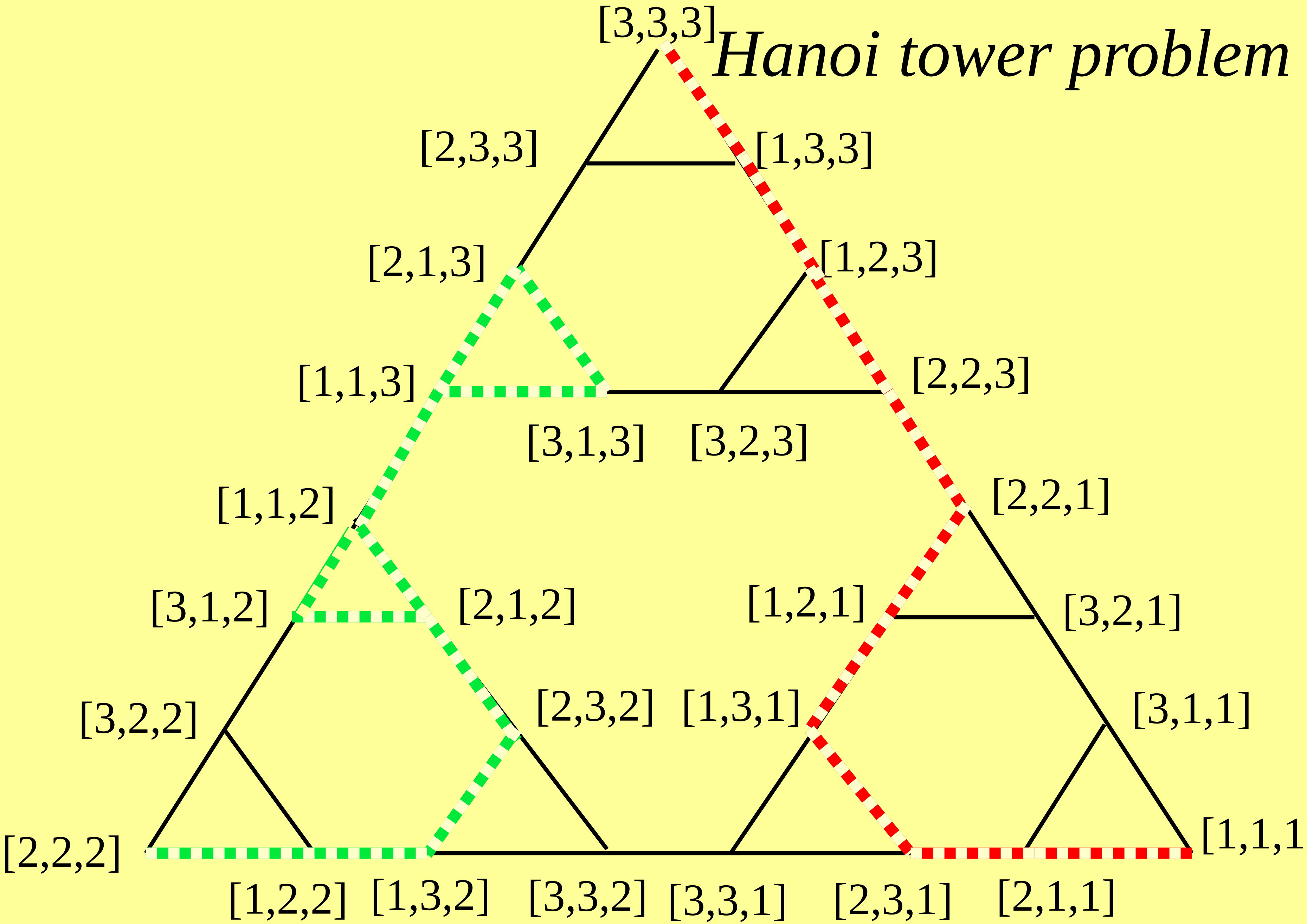
1. *current* := *start*
2. **while** $current \notin T$ **loop**
3.   *current* := $\mathbf{opt}_f(\Gamma(current) - \pi(current))$
4. **endloop**
5. **return** *current*

The original hill climbing method never makes „uphill" moves (it terminates if it cannot step onward).

if $\Gamma(current) = \emptyset$ **then**
  return solution no found
else if $\Gamma(current) - \pi(current) = \emptyset$ **then**
  current := $\pi(current)$
else *current* :=
  $\mathbf{opt}_f(\Gamma(current) - \pi(current))$

It may be useful to log the sequence of the current nodes as a path driving from the start

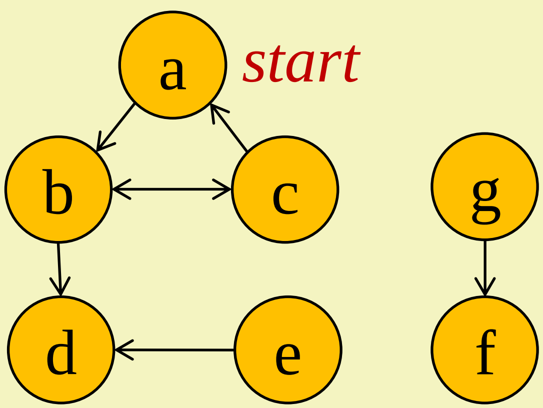
Hanoi tower problem



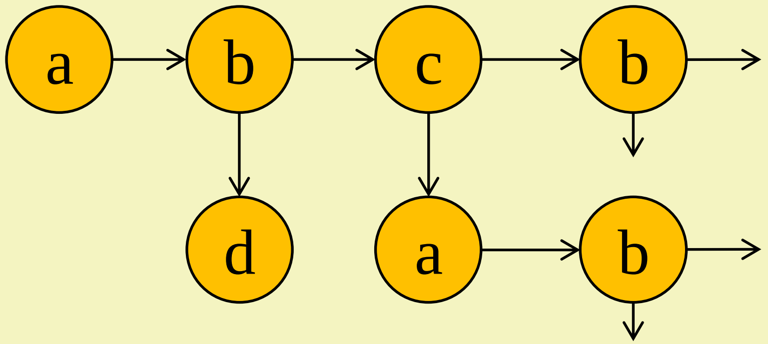
How can a path-finding algorithm perceive the representation graph?

- A search discovers the representation graph step by step. Some parts of the graph are never found and certain parts must be forgotten because of the limited memory. When a node is discovered again but it has been forgotten before, the search registers it as a new node. Thus the search can perceive the graph in a distorted form.

original graph:



distorted graph (tree) when a search can store only one node at a time:



Straightened representation graph

- A search that deletes the nodes discovered earlier or ignores checking them works in a tree (that is the straightened version of the original representation graph).
  Advantage: cycles are vanished meanwhile each path driving from the start node are preserved in this tree.
  Disadvantage: the same node might be occurred several, even infinite times in this tree.
- The bidirectional arcs greatly increase the size of this tree. To avoid this it might be useful to ignore the backward arcs from the current node to the previous current node. It needs only little extra memory: it is enough to store the parent node.

Remarks on Hill climbing method

- Advantage: its implementation is easy
- Disadvantages:
  - It can rarely find the goal without a strong heuristics because after a wrong decision it can lose itself or even stick in a dead end
    - several current nodes ⇒ local beam search
    - several attempts ⇒ random-restart search
    - give up the greedy strategy ⇒ simulated annealing
  - It can lose track around a local optimum or on an equidistant surface of the evaluation function (where neighboring nodes have identical values) if there are cycles in the representation graph (that cannot be recognized).
    - recognize smaller cycles ⇒ tabu search

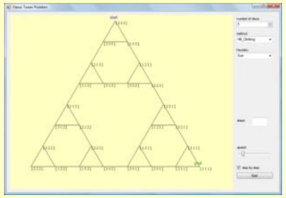
Tabu search

- Besides the current node it stores
  - the best node (*opt*) that has ever been met
  - the tabu set (*Tabu*) that contains the last few current nodes
- In each step
  - the best child of the current node is selected except for the nodes of the *Tabu*
  - if the *current* node is better than *opt* node then *opt* is exchanged for the *current*
  - *Tabu* must be updated with the *current* node
- Termination conditions:
  - if *opt* is a goal
  - if the function value of *opt* has not being changed

DATA := *initial value*
while ¬ *termination condition*(DATA) **loop**
  SELECT R FROM *rules that can be applied*
  DATA := R(DATA)
endloop

Algorithm of tabu search

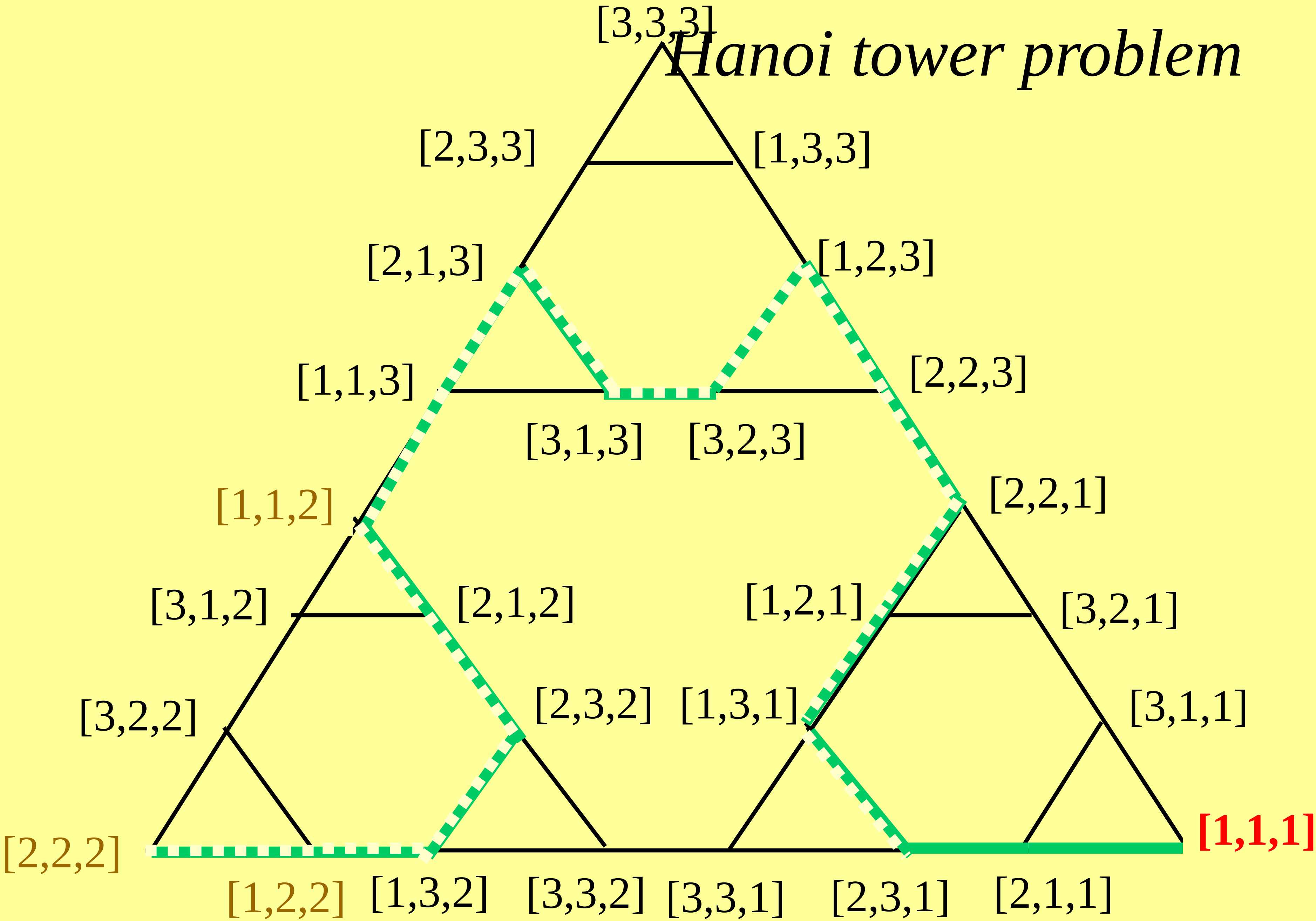
1. $current, opt, Tabu :=\ start, start, \emptyset$
2. **while not** ($opt \in T$ ***or***
   opt has not been changing for a long time) **loop**
3. $current := \mathbf{opt}_f(\ \Gamma(current) - Tabu)\ )$
4. $Tabu :=$ Update($current$, $Tabu$)
5. **if** $f(current) < f(opt)$ **then** $opt :=\ current$
6. **endloop**
7. **return** *current*



if $\Gamma(current) = \emptyset$ **then return** solution no found
else if $\Gamma(current) - Tabu = \emptyset$
  then $current := \mathbf{opt}_f(\Gamma(current))$
else $current := \mathbf{opt}_f(\Gamma(current) - Tabu)$

The original tabu search can stick on a node if all its children are in the tabu set.

Hanoi tower problem



Remarks on Tabu search

- Advantages:
  - Tabu search can recognize the smaller cycles which is shorter than the size of the tabu set so it can dominate the local optimums and the equidistant surfaces.
- Disadvantages:
  - The size of the tabu set can be set only a posteriori.
  - Without a strong heuristics it can rarely find the goal, after wrong decisions it can lose itself or even stick in a dead end.

Simulated annealing

- Instead of selecting the best child of the current node, the *new* node is picked up randomly among the children of the current node.
- If the value of this new node is not worse than the value of the current node ($f(new) \leq f(current)$), then the new node is accepted as the newer current one.
- Otherwise ($f(new) > f(current)$), the probability of the acceptance of the new node is inversely proportional to the difference of the values of the new and the current node ($|f(current) - f(new)|$).

$$e^{\frac{f(current)-f(new)}{T}} > rand[0,1]$$

Annealing schedule

- The algorithm continuously changes the coefficient *T* of the acceptance formula.
- The changing of the coefficient is based on an annealing schedule (T_k , L_k) $k=1,2,\ldots$ that rules that the coefficient be T_1 during L_1 steps, then be T_2 at the next L_2 steps, etc.

f(new)=120, f(current)=107

$$e^{\frac{f(current)-f(new)}{T_k}} > rand[0,1]$$

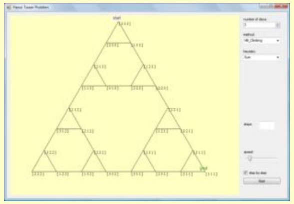
T	***exp(-13/T)***
10^{10}	0.9999…
50	0.77
20	0.52
10	0.2725
5	0.0743
1	0.000002

- If T_1 , T_2 , … is given in a decreasing order, then the probability of the acceptance of the same „bad” node is greater at the start than later.

DATA := *initial value*
while ¬ *termination condition*(DATA) **loop**
    SELECT R FROM *rules that can be applied*
    DATA := R(DATA)
endloop

Algorithm of simulated annealing

1. *current* := *start*; $k := 1$; $i := 1$
2. **while not**(*current*$\in T$ **or** f(*current*) has not been changing) **loop**
3. **if** $i > L_k$ **then** $k := k + 1$; $i := 1$
4. *new* := **select**(Γ(*current*)$-\pi$(*current*))
5. **if** $f(new) \leq f(current)$ **or** $f(new) > f(current)$ **and** $e^{\frac{f(current)-f(new)}{T_k}} > rand[0,1]$
6. **then** *current* := *new*
7. $i := i+1$
8. **endloop**
9. **return** *current*



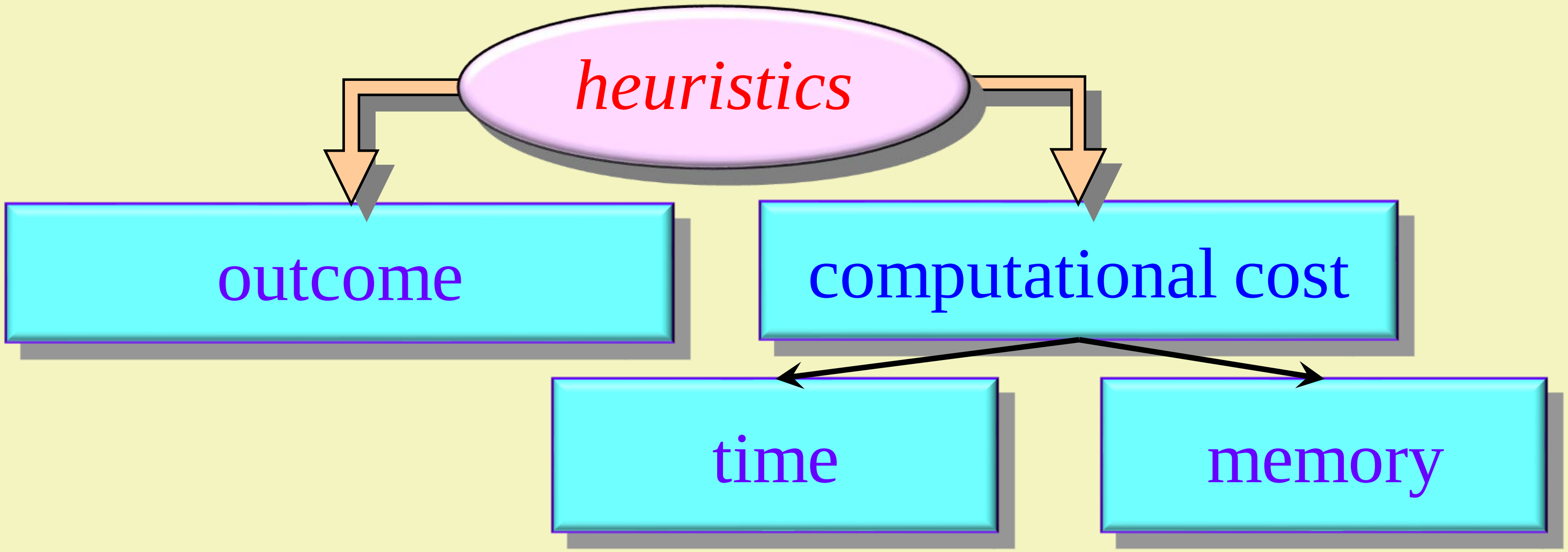
if $\Gamma(current) = \emptyset$ **then return** solution no found
else if $\Gamma(current)-\pi(current) = \emptyset$ **then** $new := \pi(current)$
else $new :=$ **select**$(\Gamma(current)-\pi(current))$

When is local search worth using?

- There is no chance to find solution without strong heuristics.
  - A wrong decision can causes a fatal error.
  - An evaluation function using good heuristics can keep off the dead-ends and traversing cycles.
- There are no dead-ends in a strongly connected representation graph.
  - Opposite to this, in a directed tree almost each branch is a dead-end. In this case only the perfect evaluation function (that never misses the appropriate path) can find solution.

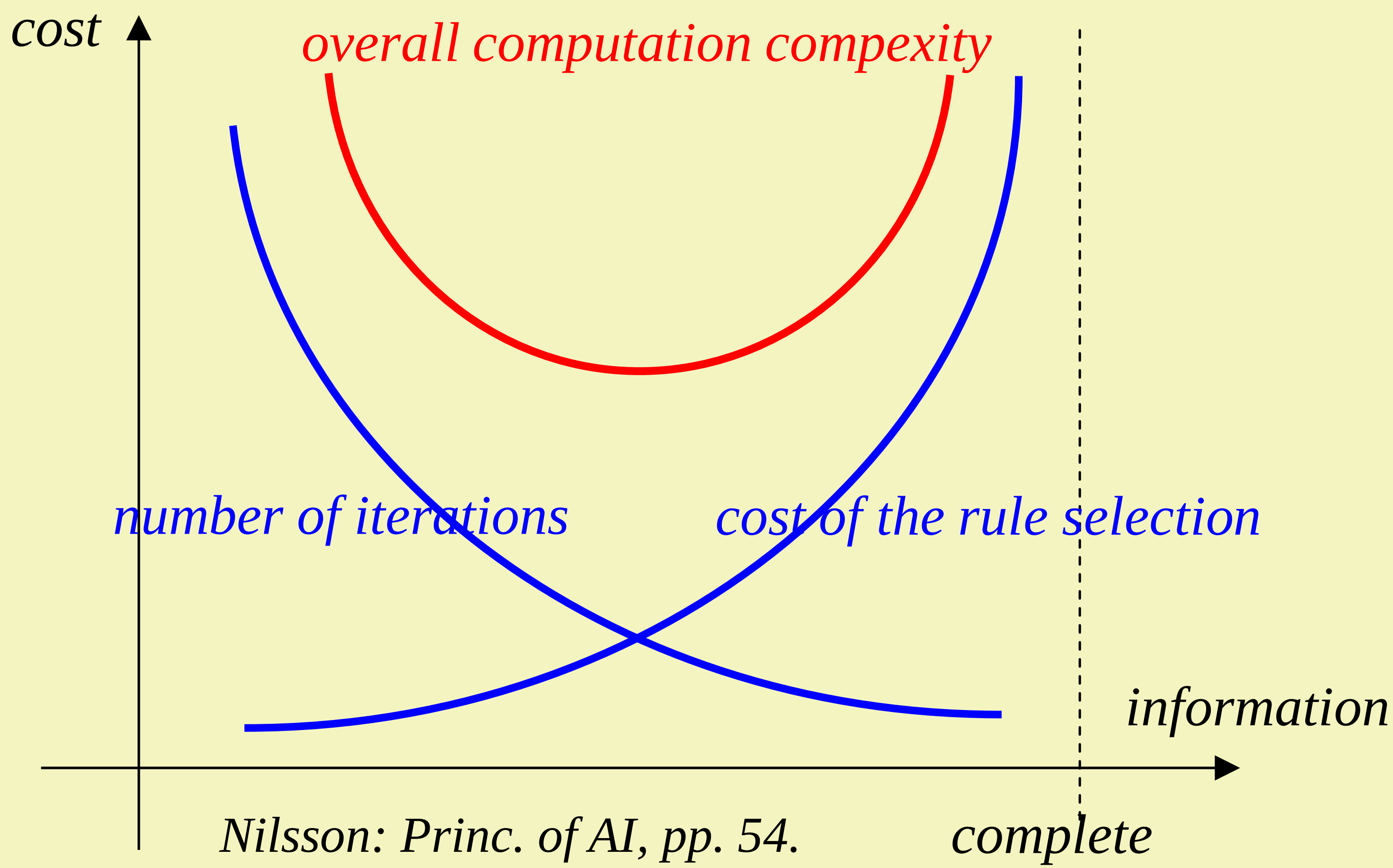
Heuristics in search systems

The heuristics is an idea (special extra information) derived from the problem. It must be built directly into the algorithm (not into the model) in order to get better solution or any solution and to improve the computational cost of the algorithm, but there is no guarantee to achieve these aims.



DATA ← *initial value (start node)*
  while ¬ *termination condition*(DATA) loop
    SELECT R FROM rules that can be applied
    DATA ← R(DATA)
  endloop

Heuristics and the computation complexity



Nilsson: Princ. of AI, pp. 54.

1	2	3
8		4
7	6	5

Heuristics for the 8-puzzle

- Misplaced (W): the number of the misplaced tiles

current:

4

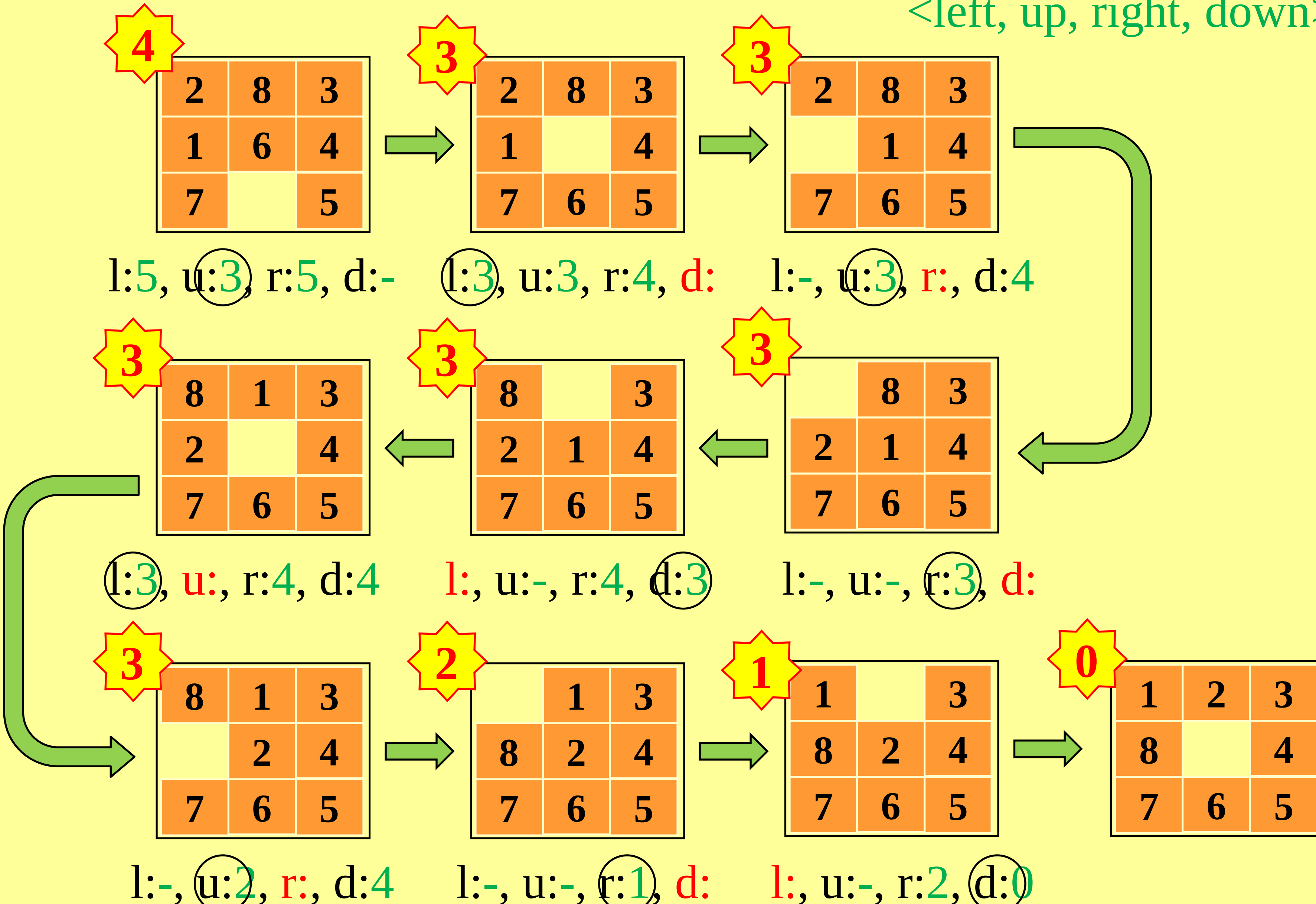
2	8	3
1	6	4
7		5

5 6

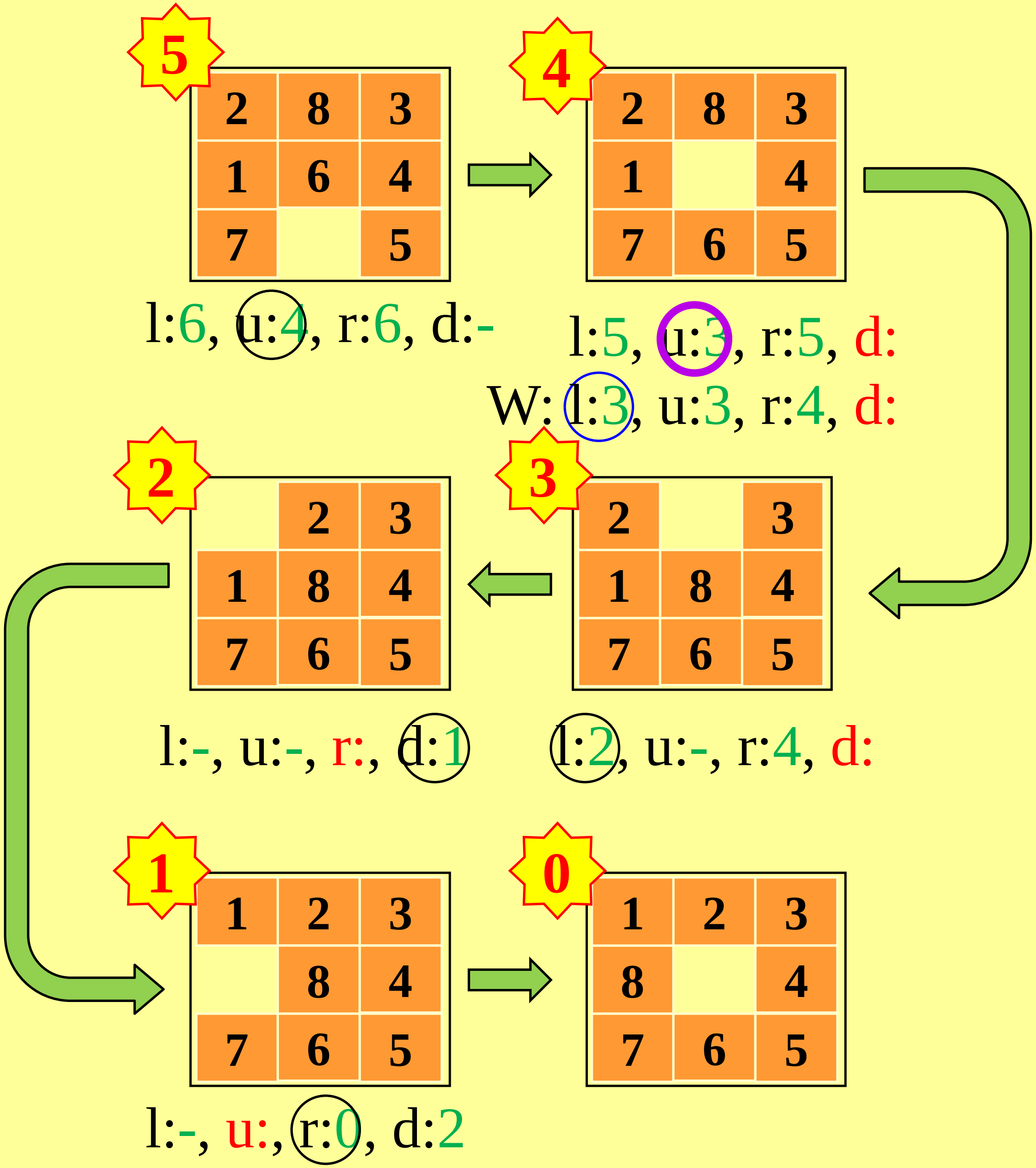
- Manhattan (P): the sum of the distances (number of horizontal and vertical moves) of the tiles from their final position
- Frame (F) or Edge: It uses penalty scores
  - 1 score for each tile that is not followed by its corresponding successor according to the final state on the edges in the clockwise direction
  - 2 scores for each corner that does not contain the corresponding tile according to the final state

W

tie-breaking rule: order of the successors <left, up, right, down>

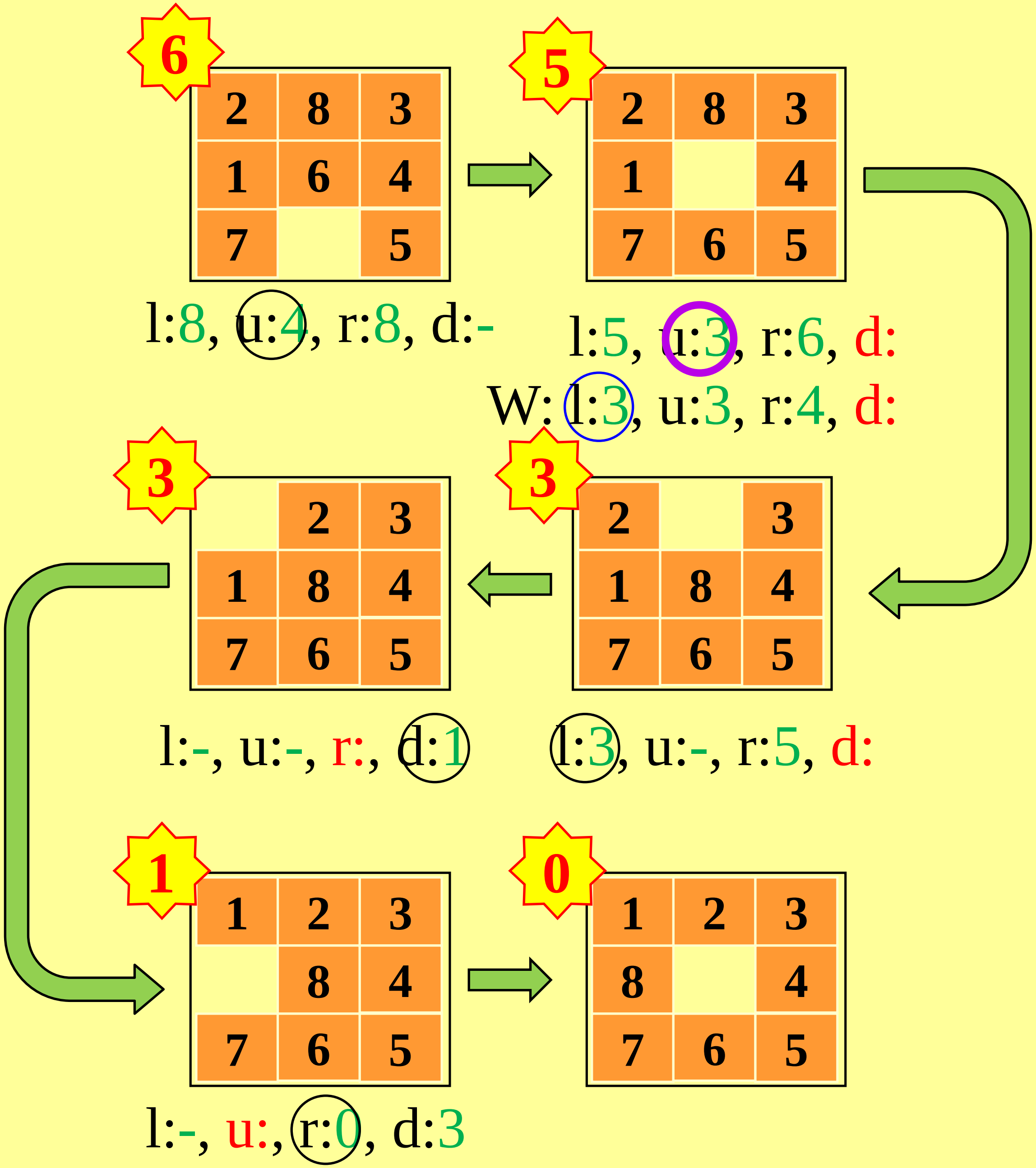


P



5
2 8 3
1 6 4
7 5
l:6, u:4, r:6, d:-
4
2 8 3
1 4
7 6 5
l:5, u:3, r:5, d:
W: l:3, u:3, r:4, d:
3
2 3
1 8 4
7 6 5
l:2, u:-, r:4, d:
2
2 3
1 8 4
7 6 5
l:-, u:-, r:, d:1
1
1 2 3
8 4
7 6 5
l:-, u:, r:0, d:2
0
1 2 3
8 4
7 6 5

F


6
2 8 3
1 6 4
7 5
l:8, u:4, r:8, d:-
5
2 8 3
1 4
7 6 5
l:5, u:3, r:6, d:
W: l:3, u:3, r:4, d:
3
2 3
1 8 4
7 6 5
l:-, u:-, r:, d:1
3
2 3
1 8 4
7 6 5
l:3, u:-, r:5, d:
1
1 2 3
8 4
7 6 5
l:-, u:, r:0, d:3
0
1 2 3
8 4
7 6 5

Straightened state graph