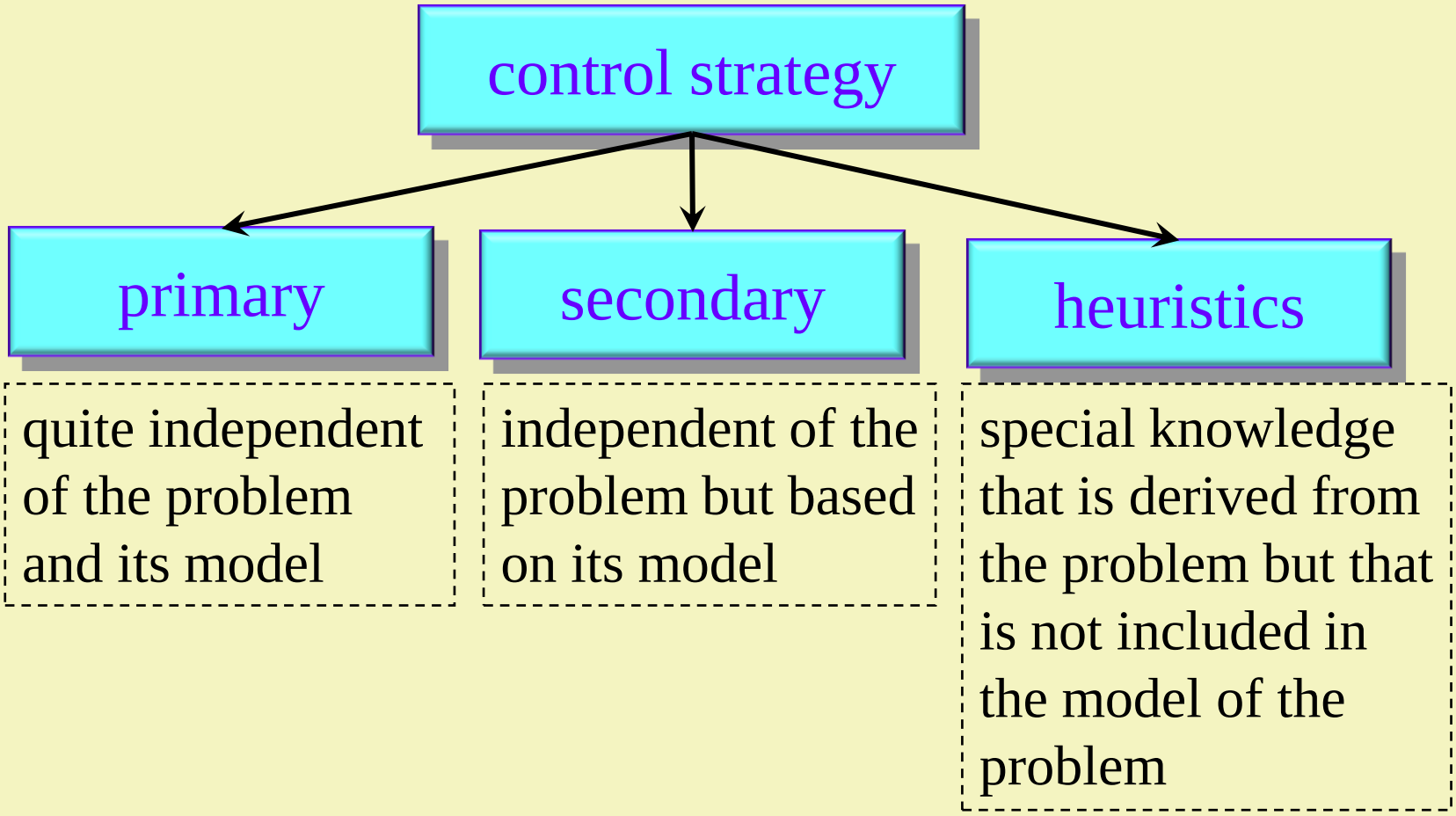
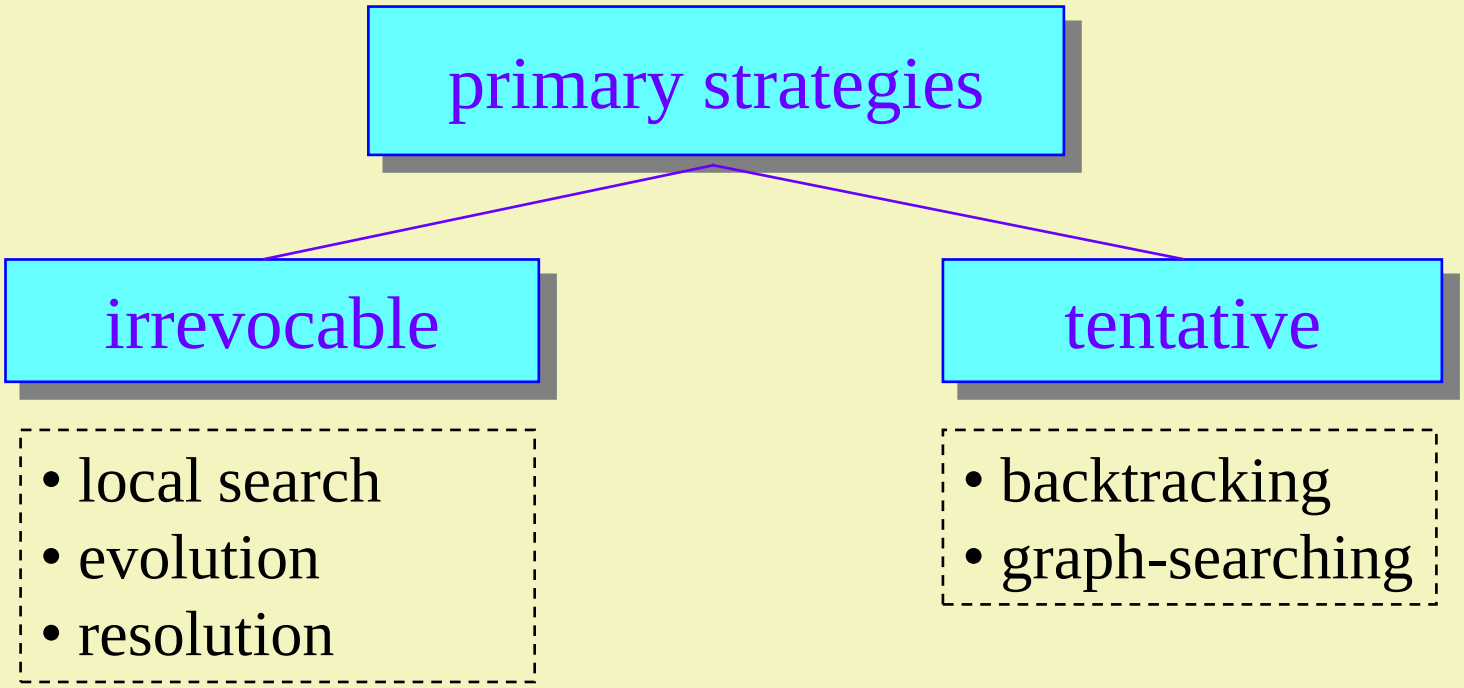
III. SEARCH

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

Levels of control



Primary control strategies



1. Local search

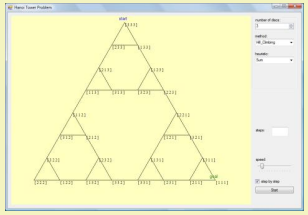
- The global workspace of this kind of search system contains only one current node of the representation graph (initially this is the start node) with a small environment of this node.
- In each step the current node is exchanged for a better node of its environment (searching rule).
- To select (control strategy) this better node an evaluation function (objective function, fitness function) will be used that tries to estimate to what extent a node promises the achievement of the goal. This function involves some heuristics.
- It stops if a goal node is found.

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

Hill climbing algorithm

- It only stores the current node and its parent ($\pi(current)$)
- In each step the best child of the current node is selected except for the parent of the current node.

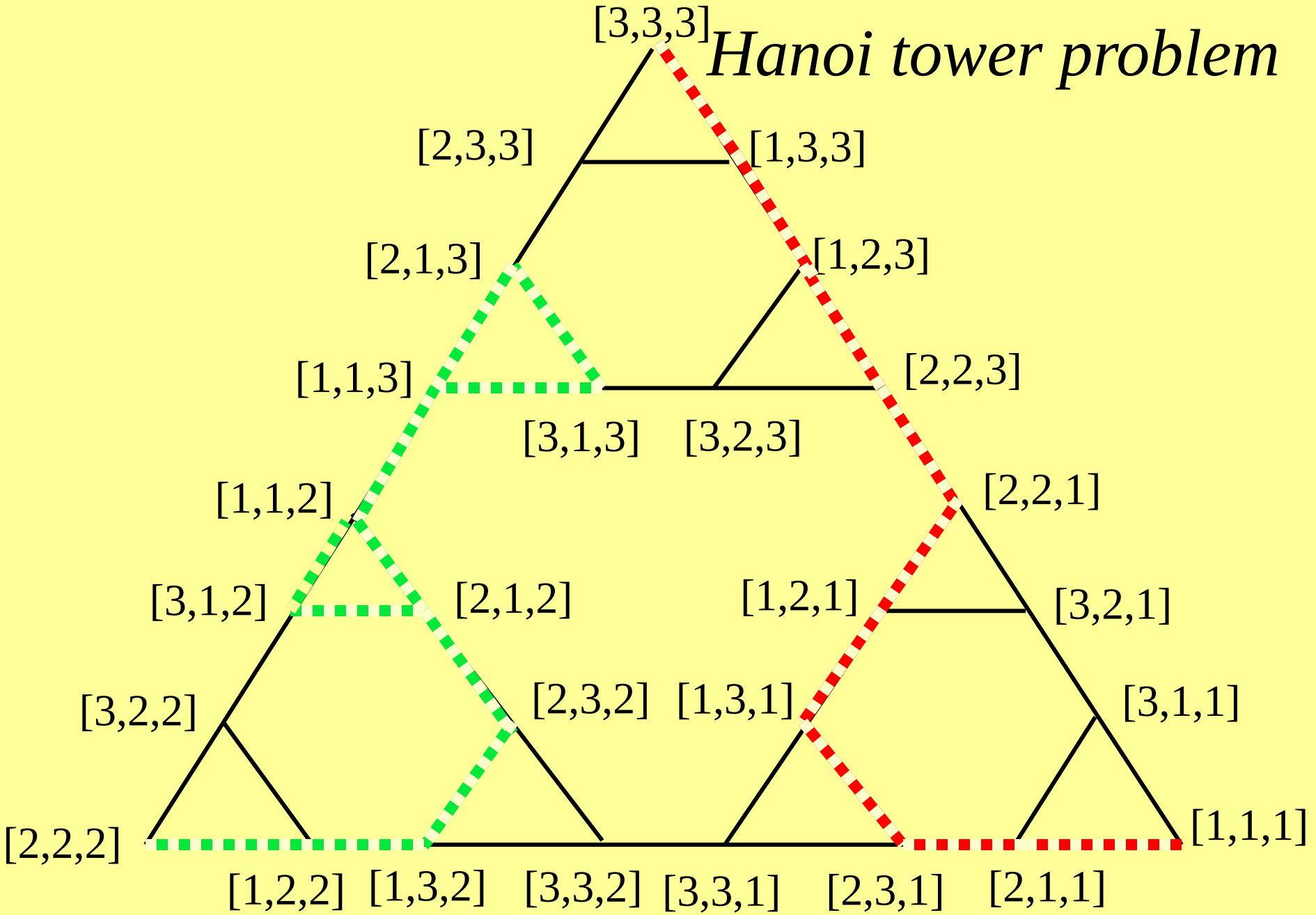
A modification of the original hill climbing method:



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

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

Hanoi tower problem



Remarks

- Disadvantages:

  1. Without a strong heuristics it can rarely find the goal, and after a wrong decision it can stick in a dead end.

  - several current nodes ⇒ local beam search
  - several attempts ⇒ random-restart search
  - give up the greedy strategy ⇒ simulated annealing

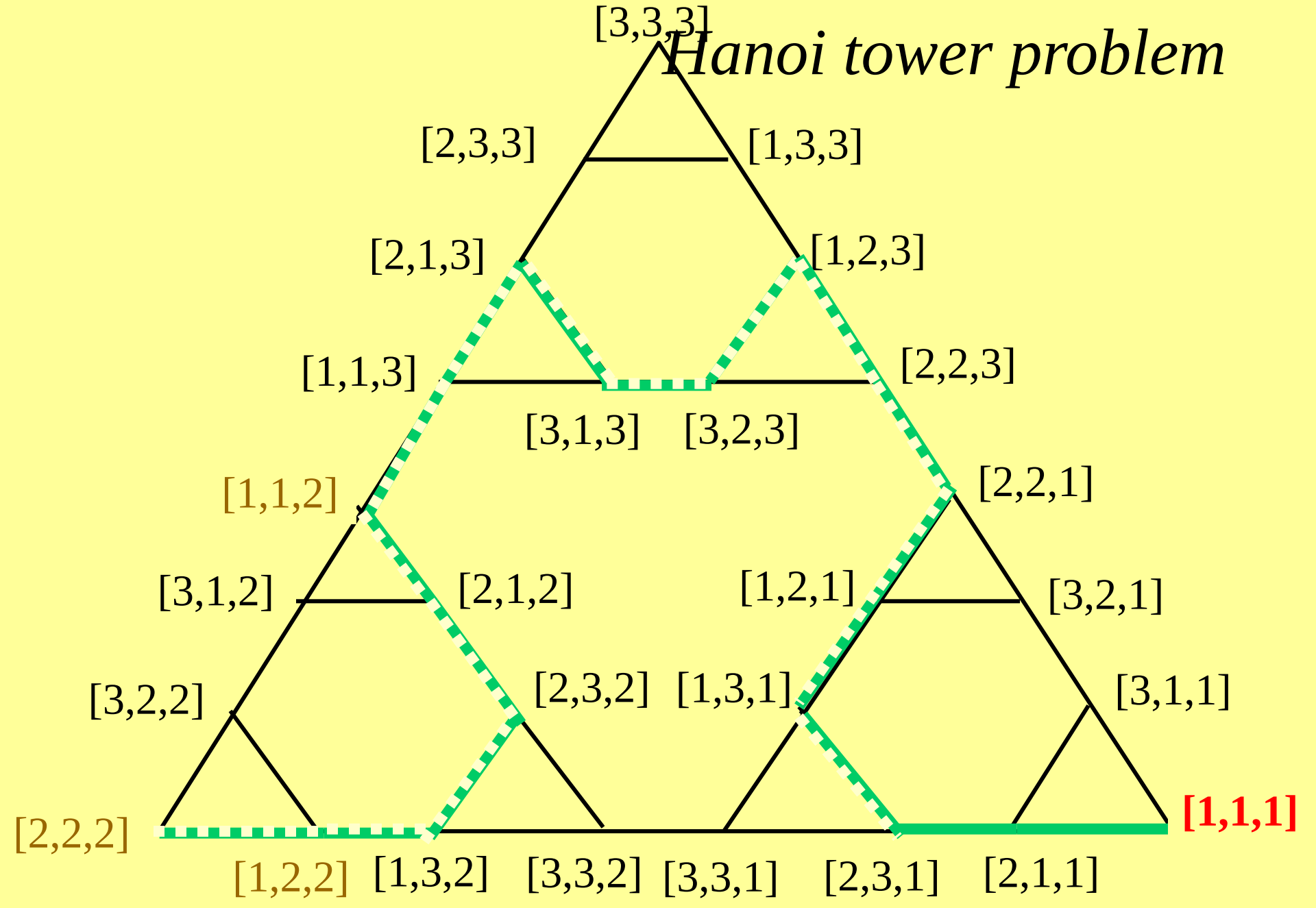
  2. If there are circles (that cannot be recognized) this search can lose track around a local optimum or on an equidistant surface (where neighbor nodes have identical values) of the evaluation function .

  - recognize small circles ⇒ 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 values of *current* or *opt* is not being changed

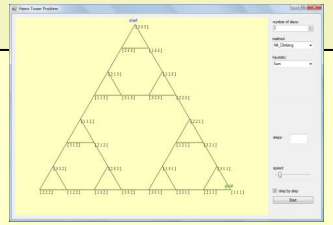
Hanoi tower problem



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

Algorithm of tabu search

A modification of the original tabu search:



1. *current*, *opt*, *Tabu* := *start*, *start*, ∅
2. **while not** (*opt*∈*T* ***or***
   opt has not been changing for a long time) **loop**
3. *current* := **opt**$_f$(Γ(*current*)−*Tabu*))
5. *Tabu* := Update(*current*,*Tabu*)
6. **if** *f*(*current*) < *f*(*opt*) **then** *opt* := *current*
7. **endloop**

if Γ(*current*) = ∅ **then return** solution no found
if Γ(*current*)−π(*current*) = ∅ **then** *current* := **opt**$_f$(*Tabu*)
else *current* := **opt**$_f$(Γ(*current*)−π(*current*))

Remarks

- Advantages:
  - It can recognize the smaller circles if their lengths are less 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.
  - The original tabu search can stick on the node if all its children are in the tabu set.

Simulated annealing

- It can select a worse neighbor node instead of the best one.
- The new node (*new*) is selected randomly among the children of the current node.
- If the value of the 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, 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 ($|f(current) - f(new)|$).

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

Annealing schedule

- The schedule (T_k , L_k) $k=1,2,\ldots$ rules that at first the value of the coefficient T be T_1 during L_1 steps, then be T_2 at the next L_2 steps, etc.:

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

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

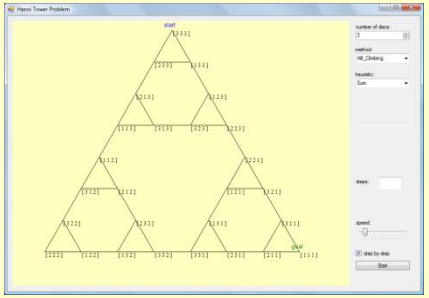
f(new)=120, f(current)=107

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

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*
2. **while not**(*current* $\in T$ **or** *f*(*current*) has not been changing) **loop**
3. **for** $i = 1 .. L_k$ **loop**
4. **if** $\Gamma(current) = \emptyset$ **then return** solution no found
5. **if** $\Gamma(current) - \pi(current) = \emptyset$ **then** *new* := π(*current*)
   else *new* := **select**($\Gamma(current) - \pi(current)$)
6. **if** *f*(*new*) ≤ *f*(*current*) **or**
   f(*new*) > *f*(*current*) **and** $e^{\frac{f(current) - f(new)}{T_k}} > rand[0,1]$
   then *current* := *new*
7. **endloop**
8. *k* := *k*+*1*
9. **endloop**

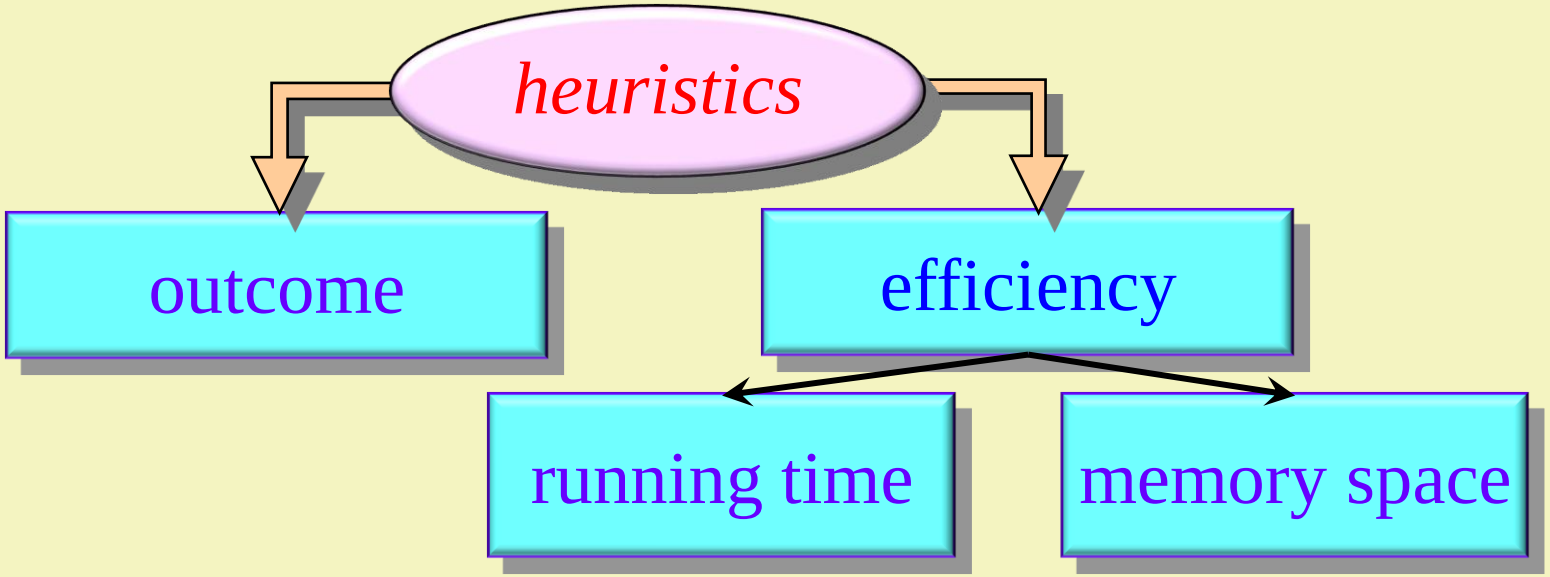


When is local search worth using?

- There is no chance for the local searches to find solution if they have not good, strong heuristics.
- The only expectance is when the representation graph is strongly connected.
  - But it is not a necessary condition of the solution.
  - If the graph is a directed tree then only the perfect (never miss) evaluation function can find solution.

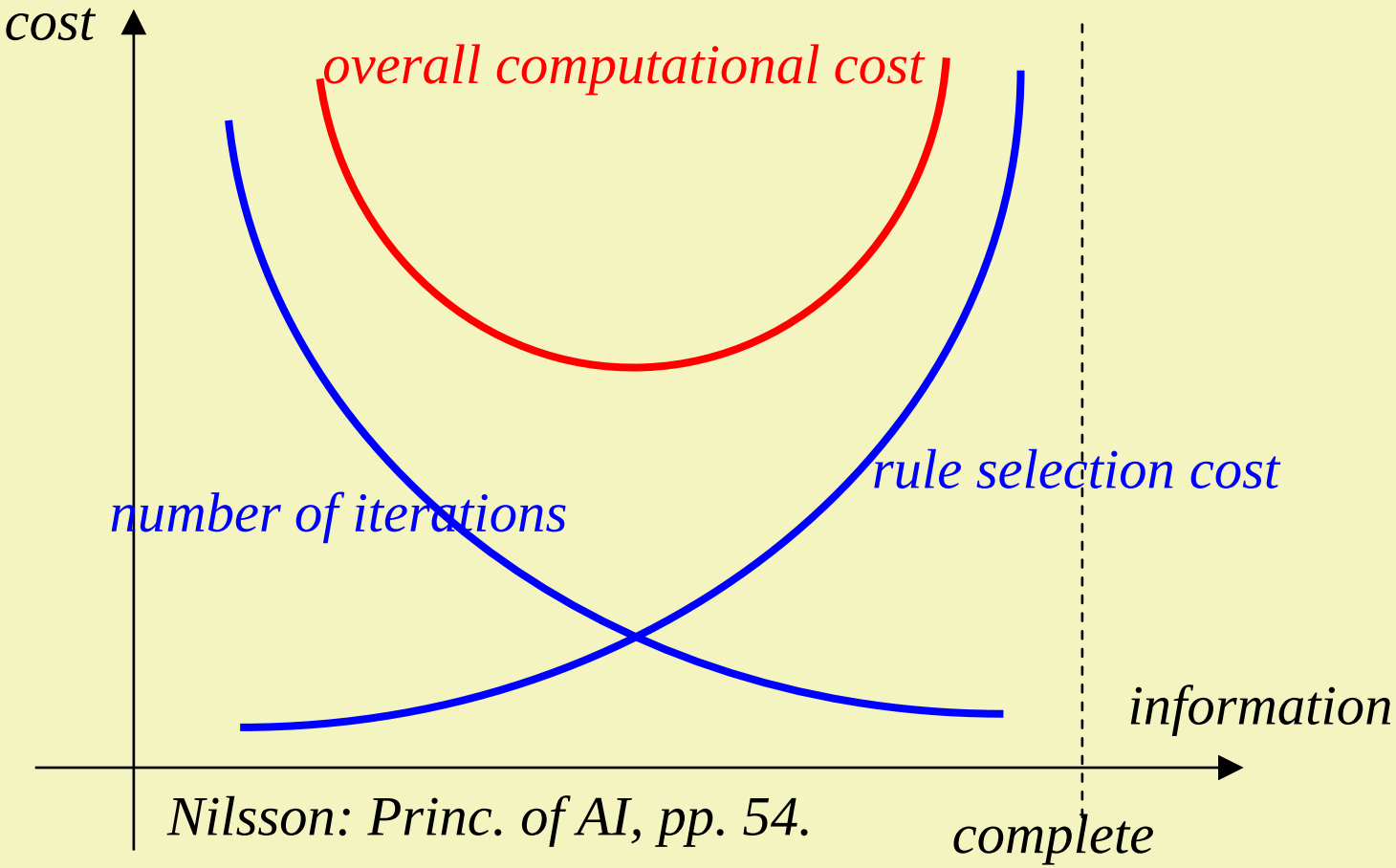
Heuristics in search systems

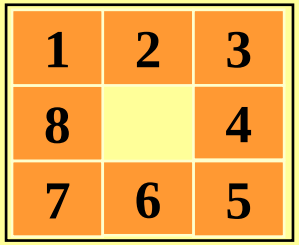
The heuristics is an idea (special extra information) derived from the problem. It must be built directly into the algorithm (not into the model) in order to get better solution or any solution and to improve the efficiency of the algorithm, but there is no guarantee to achieve these aims.



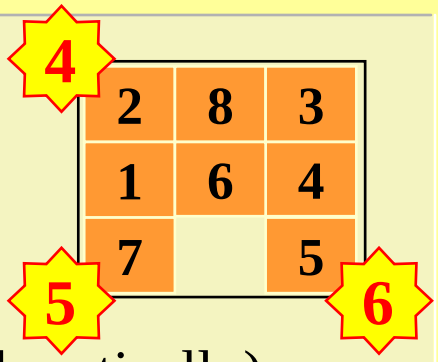
DATA ← *initial value (start node)*
while ¬ *termination condition*(DATA) loop
SELECT R FROM rules that can be applied
DATA ← R(DATA)
endloop

heuristics and the computational cost





Heuristics for the 8-puzzle



- Misplaced: the number of the misplaced tiles

$$W(this) = \sum_{ij} 1, \quad this_{ij} \neq 0 \wedge this_{ij} \neq goal_{ij}$$

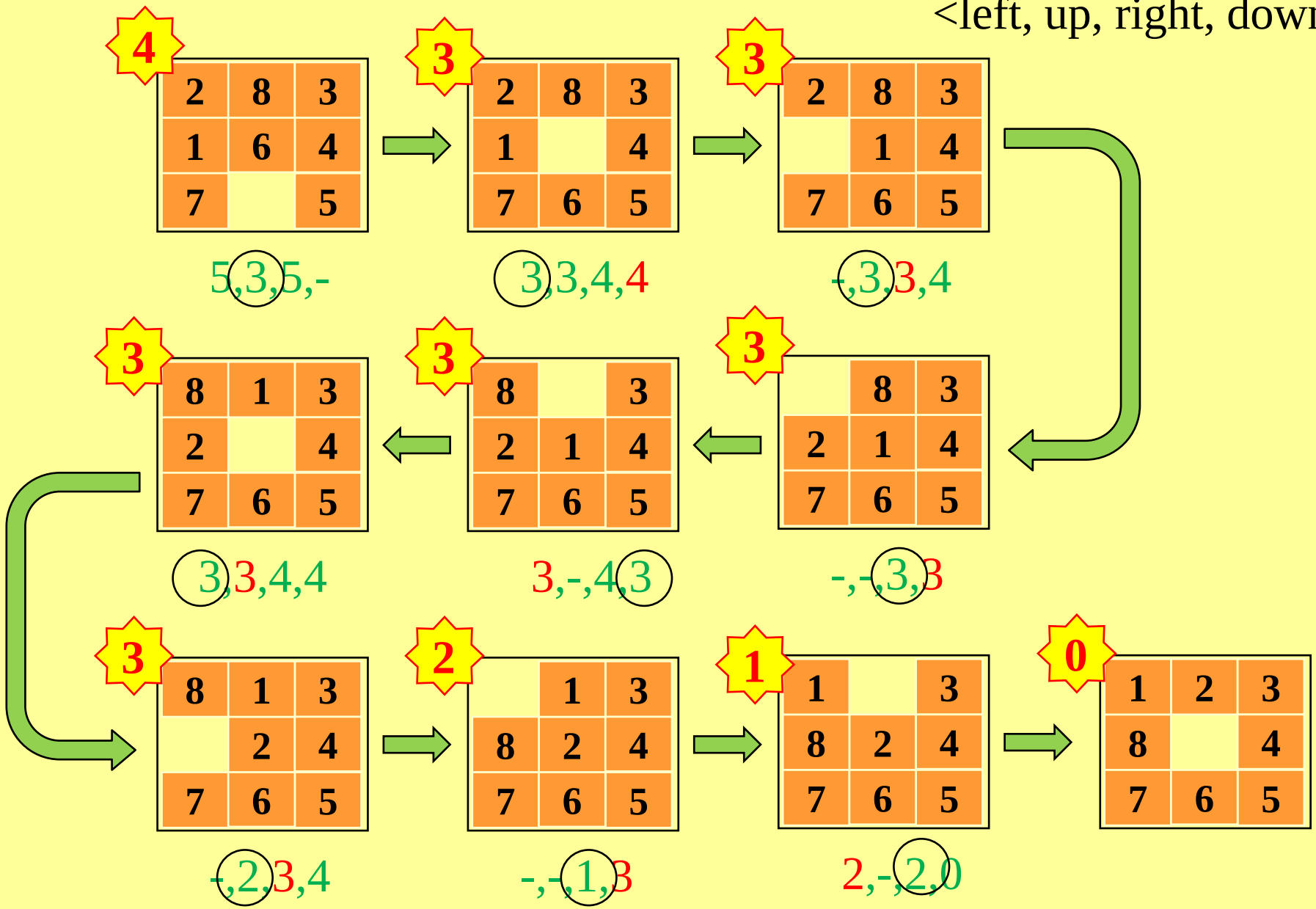
- Manhattan: the sum of the Hamilton distances for all tiles (a tile can be moved only horizontally and vertically) between their current position and goal position

$$P(this) = \sum_{ij} |(i,j) - goal\_position(this_{ij})|$$

- Edge (penalty score): It allots
  - 1 score for each tile on the edge that is not followed by its corresponding successor
  - 1 score for each tile on the edge that should be in the center
  - 2 scores for each corner that does not contain the corresponding tile

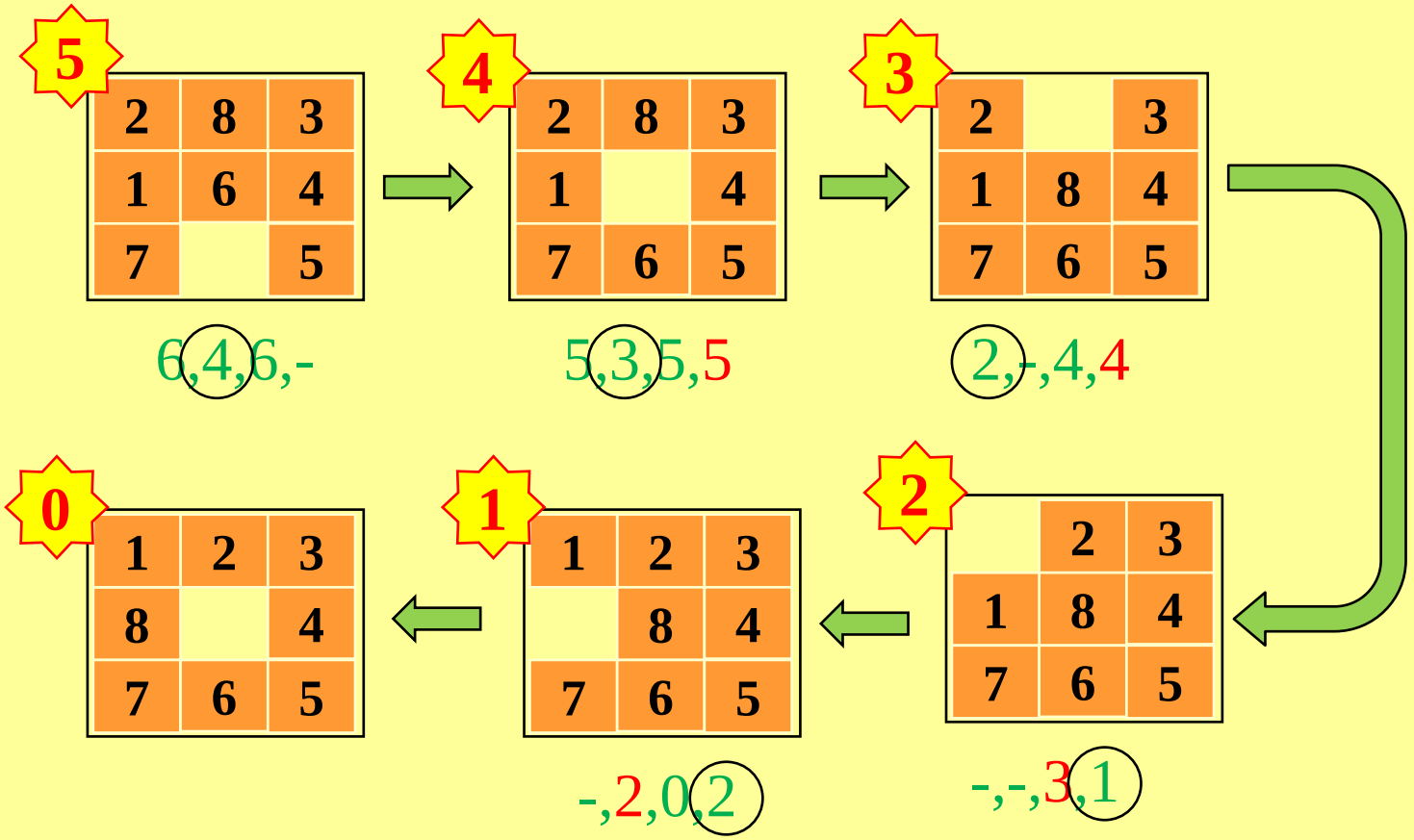
W

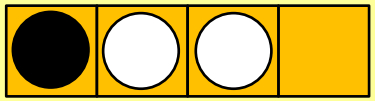
order of the successors:
<left, up, right, down>



P

order of the successors:
<left, up, right, down>



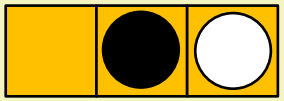
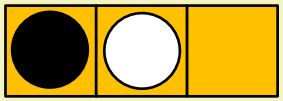


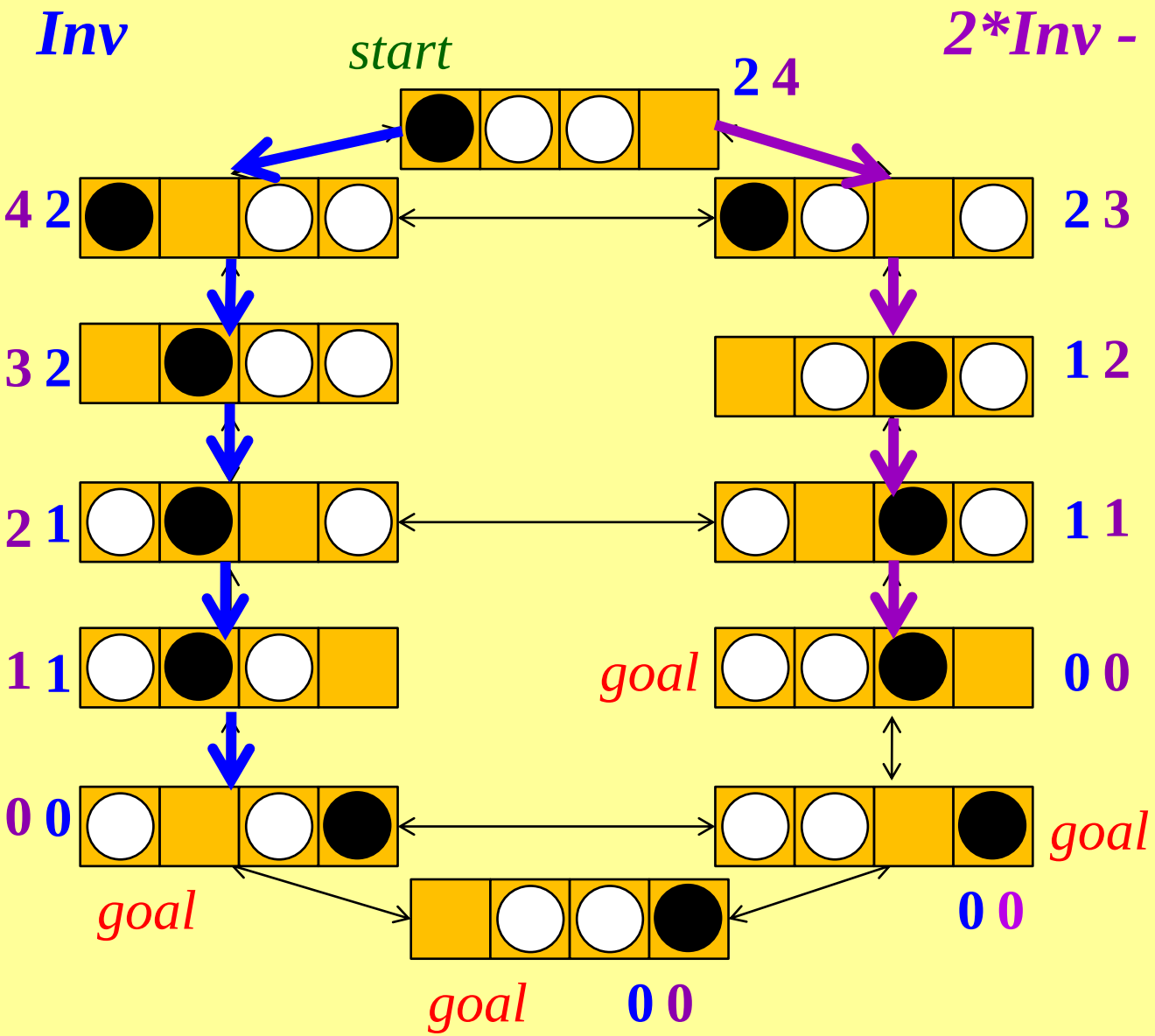
Heuristics for Black&White puzzle

- Inversion number of the permutation: *I(this)*

  how many swaps must be taken to achieve the goal permutation (where all white tiles precede all black ones)

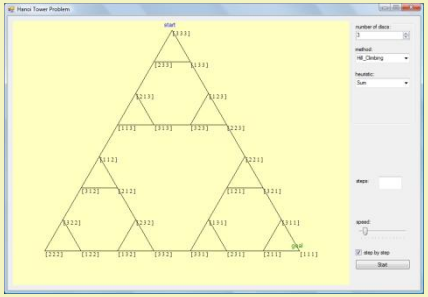
- Modified inversion number:

  *M(this)= 2*I(this) – (1* ***if*** *this* has  or  part)


Inv
start
2*Inv - ?1
2 4
4 2
2 3
3 2
1 2
2 1
1 1
1 1
goal
0 0
0 0
goal
goal
0 0
goal
0 0

Heuristics for Hanoi tower problem

- Count: $C(this) = \sum_{\substack{i=1..n \\ this[i]\neq 1}} 1$
- Weighted count: $WC(this) = \sum_{\substack{i=1..n \\ this[i]\neq 1}} i$
- Sum: $S(this) = \sum_{i=1..n} this[i]$
- Weighted sum : $WS(this) = \sum_{i=1..n} i*this[i]$
- Modified sum: $MWS(a) = WS(this) - \sum_{\substack{i=2..n \\ this[i-1]>this[i]}} 1 + \sum_{\substack{i=2..n-1 \\ this[i-1]=this[i+1] \wedge this[i]\neq this[i-1]}} 2$



Heuristics for Hanoi tower problem

- Perfect: $P(this) = f(1)_1$ $\quad f:[1..n+1]\rightarrow \mathbb{N}\times\mathbb{N}$

  $f(i)_1$: number of the steps to achieve the goal if the discs only between i^{th} and n^{th} count, as if the smaller discs did not exist

  $f(i)_2$: the peg that is unnecessary when the solution of the problem of the discs between i^{th} and n^{th} is started.

  When the $i–1^{th}$ disc is also counted then it always gets in the way (and it always must be set aside) before moving bigger discs, except for the first step if it is on the peg $f(i)_2$. Consequently the number of the steps calculated earlier should be doubled and at last it is placed onto the good peg.

  $f(n+1) = (0, 1)$

  $$f(i-1)= \begin{cases} (\ 2*f(i)_1 \quad , f(i)_2\ ) & \text{ha } this[i-1]=f(i)_2 \\ (2*f(i)_1+1,\ 6-f(i)_2-this[i]\ ) & \text{ha } this[i-1]\neq f(i)_2 \end{cases}$$

Homework

- Find a heuristics evaluation function for the Knight problem that can guide the knight to find the goal and present its solution path (the trace of the knight) generated by the hill climbing method!