

インテリジェントシステム

#3 探索による問題解決：
知識を用いた探索 & 局所探索
informed search & local search

信州大学工学部電子情報システム工学科
丸山稔

知識に基づく探索 (Informed Search, Heuristic Search)

ゴール位置に関する問題固有のヒント・知識を持つ場合の探索方式

ヒント⇒ヒューリスティック (heuristic) 関数の形で与えられる→ $h(n)$

$h(n)$: ノード n の状態からゴール状態に至るまでの最小コスト推定値

uniform-cost searchの場合 : 評価関数←初期状態から n までのコスト

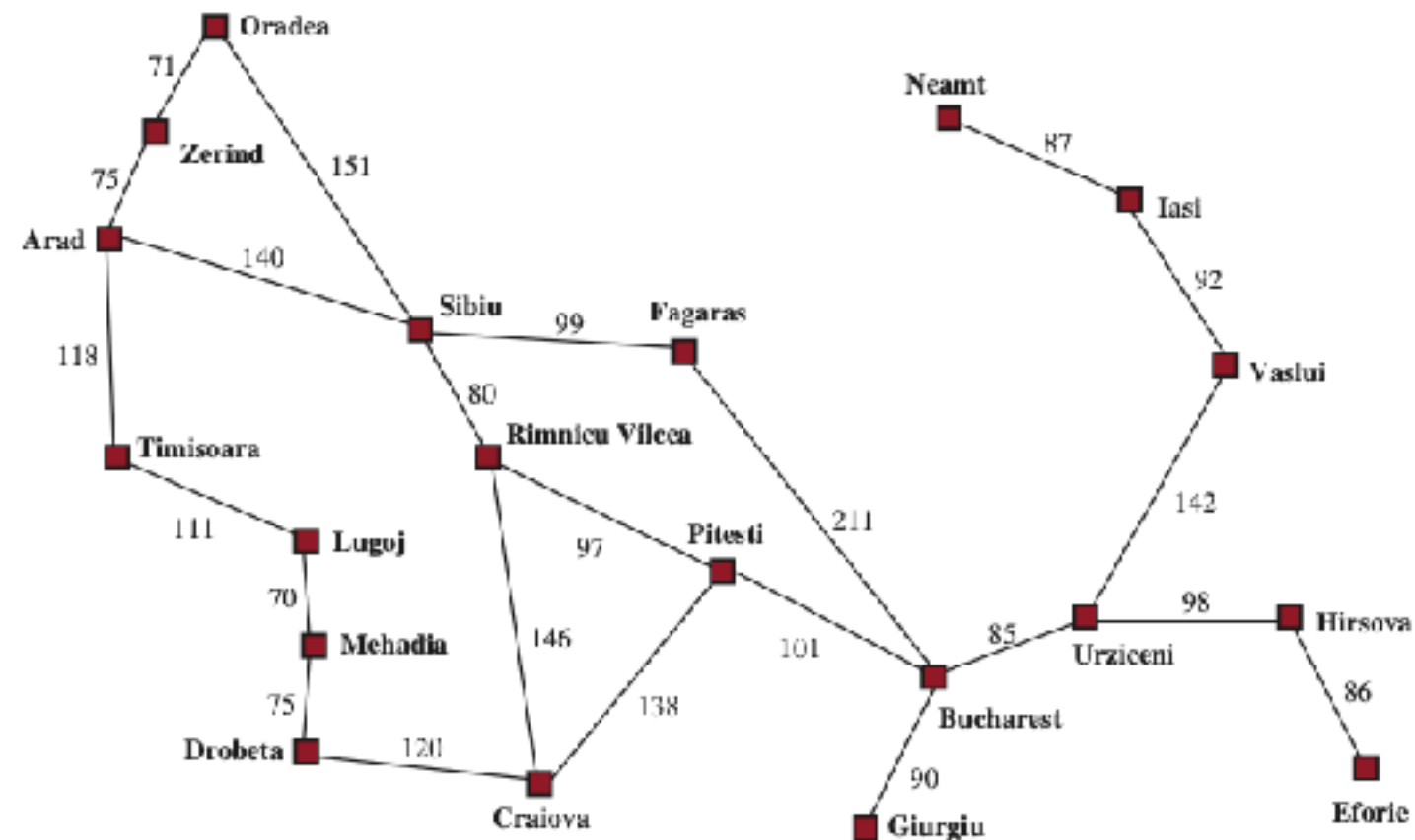
$h(n)$ はこれからかかる最小コストの推定値

貪欲最良優先探索 (greedy best-first search)

$f(n) = h(n)$ とした最良優先探索 (best-first search)

→ frontier中で $h(n)$ が最小のノード (ゴールに最も近いと見えるノード) から展開していく

greedy best-first search適用例：ルーマニア旅行の例に適用

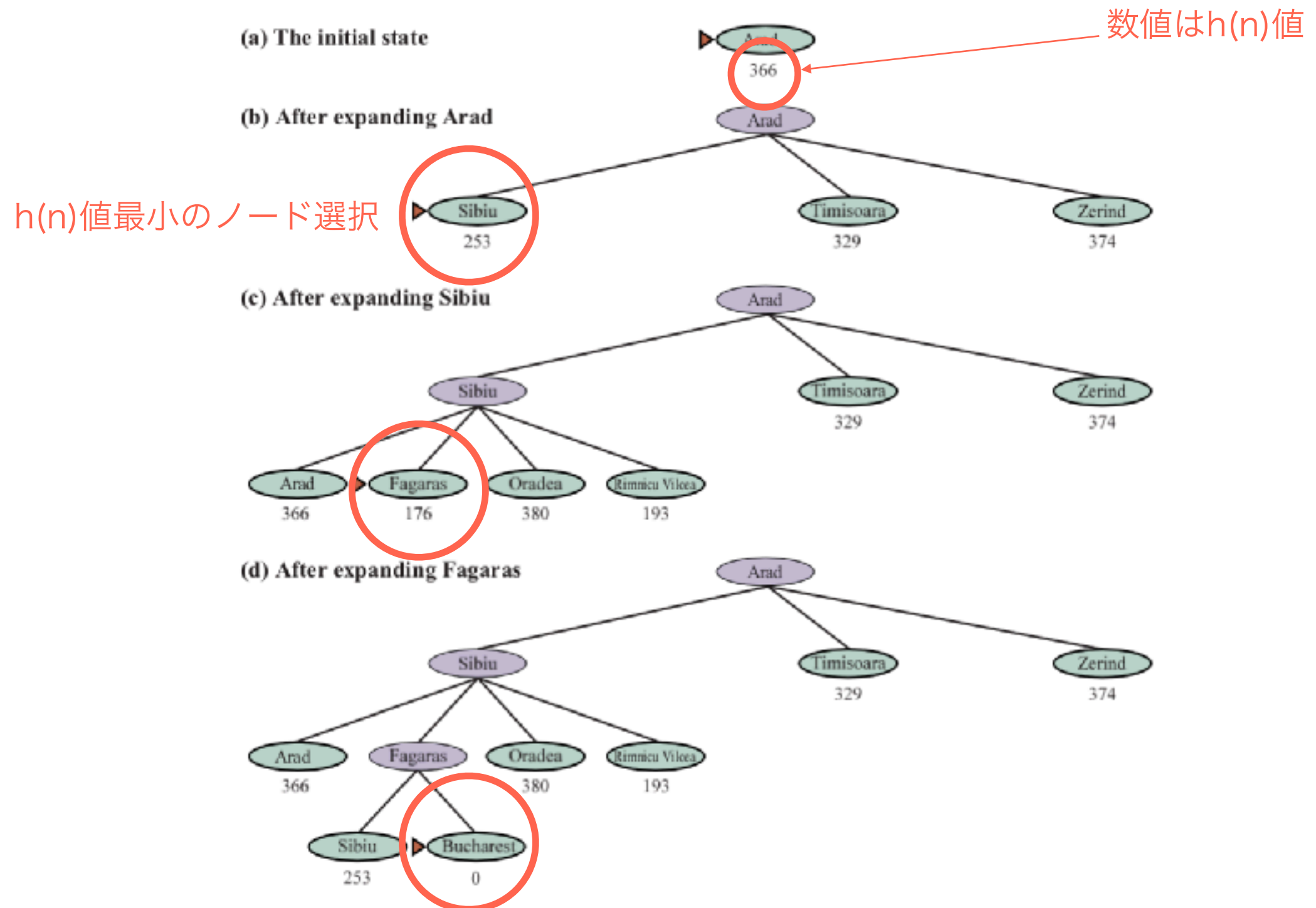


heuristic関数：ゴール（Bucharest）までの直線距離（道路上の走行距離ではなく）

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

greedy best-first search適用例：ルーマニア旅行の例

直線距離ヒューリスティック & tree-like searchを適用



A*-search

最も一般的なinformed searchアルゴリズム

評価関数 $f(n)$ として、 $f(n) = g(n) + h(n)$ を用いる最良優先探索

$g(n)$: 初期状態からノード n までの経路コスト（これまで掛かったコスト）

$h(n)$: n からゴールまでの最短路の推定コスト（これから掛かるコスト最小値）

➡ $f(n) =$ これまでの n までの経路を続けてゴールに至る最良の経路の推定コスト

A*-探索アルゴリズムの性質

- ・（全ての行動コストが $> \epsilon > 0$ 、状態空間は解を持つかまたは有限、の仮定の下で）完全（complete）
- ・コスト最適（cost-optimal）かどうかはヒューリスティックの性質による
⇒ admissibility（許容性）

許容的ヒューリスティック（admissible heuristic） とは？

⇒ ゴールまでのコストを実際より大きく見積もらないヒューリスティック
（即ち…常に楽観的見積もりをするヒューリスティック）

A*探索アルゴリズム：

admissible heuristicの場合コスト最適 (cost-optimal) であることの証明

最適経路のコストを C^* とおく

背理法：A*アルゴリズムはコスト $C > C^*$ の経路を出力すると仮定して矛盾を導く

最適経路上のノード n で未展開のものが存在する

(\because 全て展開済ならば最適経路が見つかるはず)

$g^*(n)$ ：初期状態から n までの最適経路

$h^*(n)$ ： n から最近接 (nearest) ゴール (コスト最小) までの最適経路コスト
とおくと...

$f(n) > C^*$... (\because そうでなければ n は既に展開されているはず)

$f(n) = g(n) + h(n)$... 定義より

$f(n) = g^*(n) + h(n)$... ($\because n$ は最適経路上のノード)

$f(n) \leq g^*(n) + h^*(n)$... admissible heuristicなので $h(n) \leq h^*(n)$

$f(n) \leq C^*$... $g^*(n) + h^*(n) = C^*$ より

 矛盾！

A*-search適用例

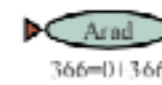
heuristic関数 $h(n)$
Bucharestまでの
直線距離



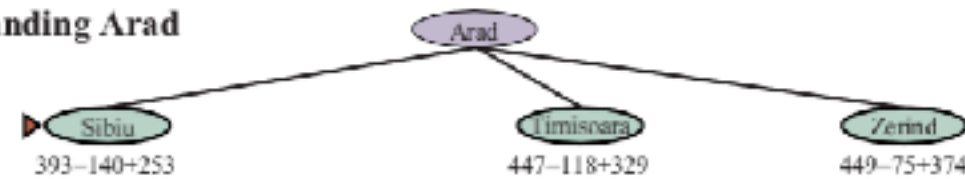
admissible

∴
走行距離以下を常に
与える

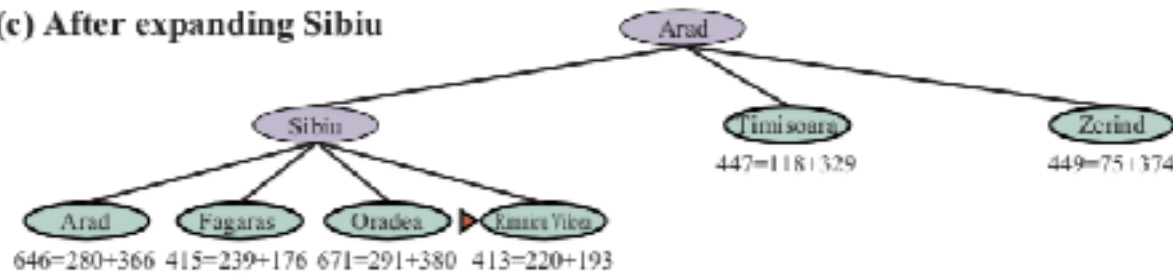
(a) The initial state



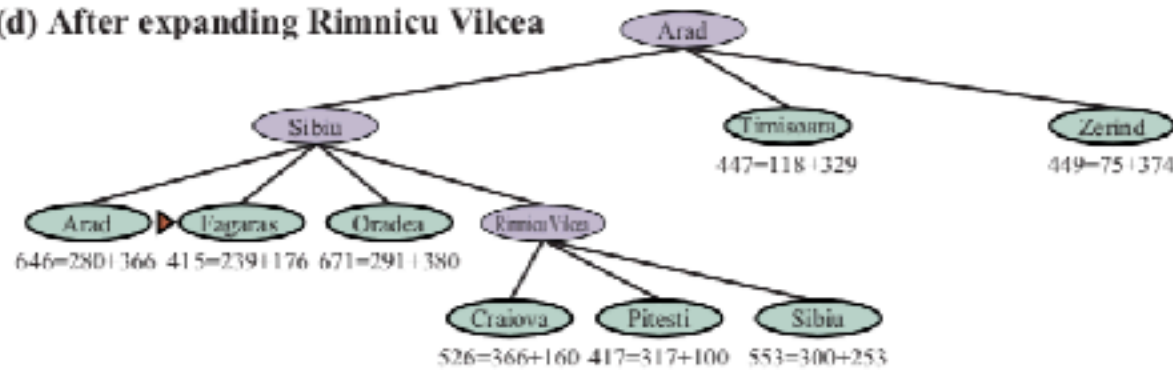
(b) After expanding Arad



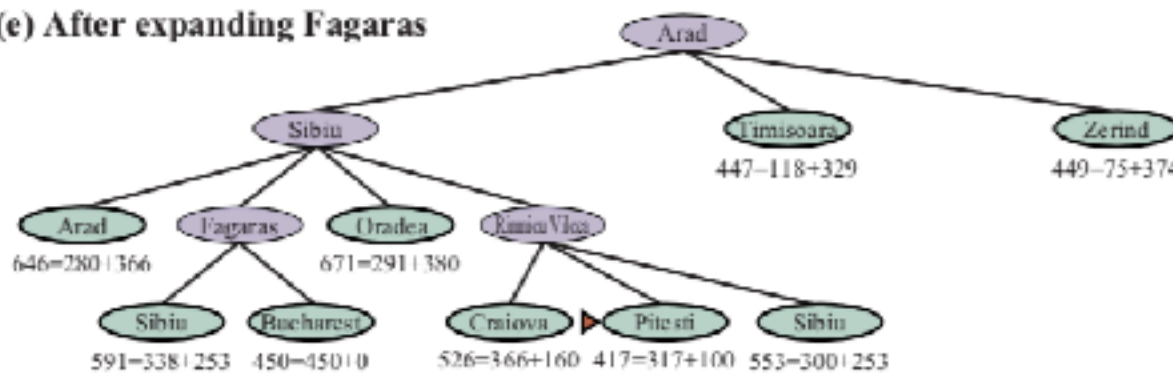
(c) After expanding Sibiu



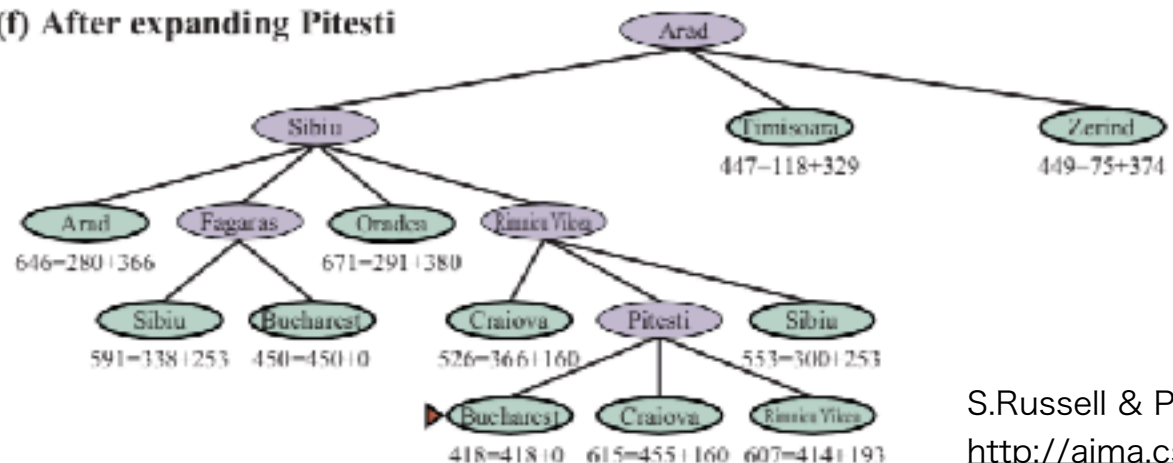
(d) After expanding Rimnicu Vilcea



(e) After expanding Fagaras



(f) After expanding Pitesti



heuristic関数の性質：consistency（無矛盾性）

admissibilityより強い条件

$h(n)$ は以下が成り立つときconsistent

全てのノード n と行動 a により生じる n の全ての後続ノード/子ノード n' について

$$h(n) \leq c(n, a, n') + h(n') \quad \leftarrow \text{三角不等式}$$

$c(n, a, n')$ ：状態 n において行動 a により n' に遷移したときの行動コスト

consistencyの方がadmissibilityより『強い』条件…i.e.

$consistent \Rightarrow admissible$

n からゴール G への最適経路が

$n_0 = n \rightarrow a_1 \rightarrow n_1 \rightarrow a_2 \rightarrow n_2 \rightarrow \cdots \rightarrow n_k = G$ であったとする

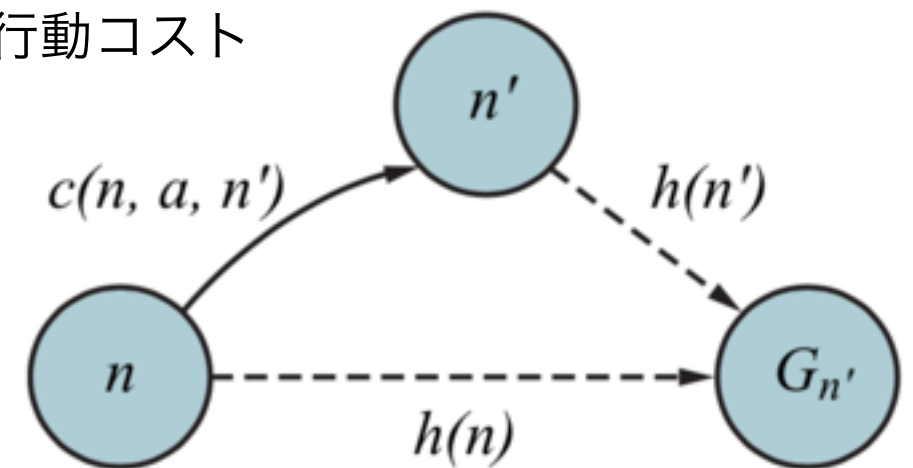
consistencyより： $h(n) \leq c(n, a_1, n_1) + h(n_1)$

$$h(n_1) \leq c(n_1, a_2, n_2) + h(n_2)$$

\vdots

$$h(n_{k-1}) \leq c(n_{k-1}, a_k, n_k) + h(n_k) \quad h(n_k) = h(G) = 0$$

$$h^*(n) = \sum_{i=1}^k c(n_{i-1}, a_i, n_i) \quad \text{だから、各式の和より} \quad \rightarrow h(n) \leq h^*(n)$$



consistent \Rightarrow admissible

admissible \Rightarrow A*は最適経路を与える…したがって、consistent heuristicの場合も同様

consistentなヒューリスティック関数によるA*探索の場合 \Rightarrow f値は経路上で単調増大

経路を拡大していくとき（注：コストは正であることを仮定）

\Rightarrow nを展開してn'へ拡大したとすると…行動コストは正だから $g(n) < g(n')$ …gに関しては経路上単調


$f=g+h$ は経路上単調に増大する？

経路がnからn'へ拡大したとき：

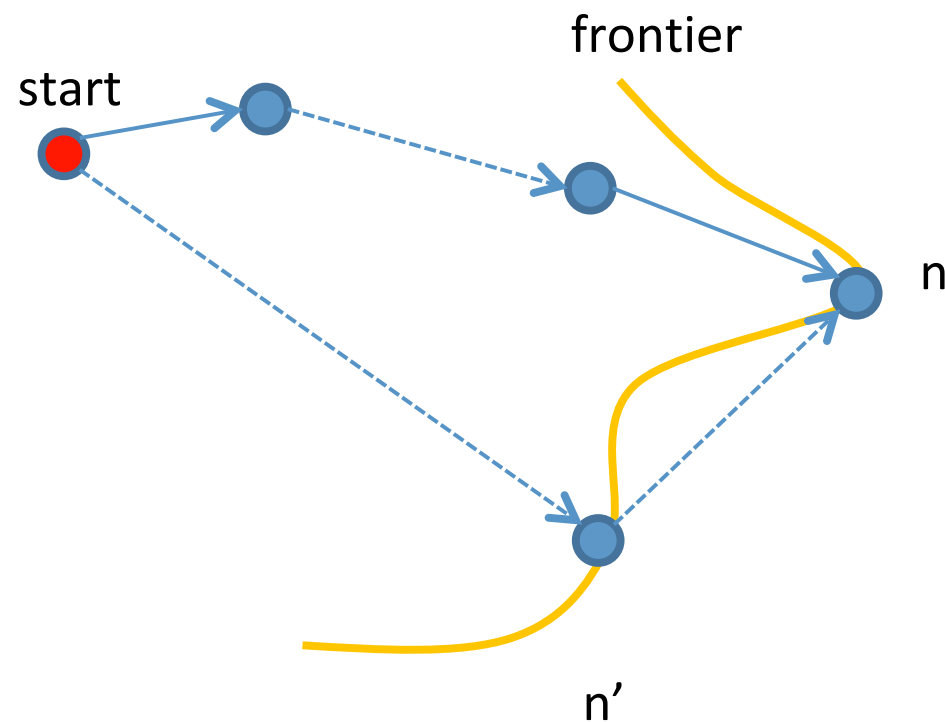
$$g(n) + h(n) \rightarrow g(n') + h(n') = g(n) + c(n, a, n') + h(n')$$

consistencyより：

$$h(n) \leq c(n, a, n') + h(n')$$


$$f(n) = g(n) + h(n) \leq f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n')$$

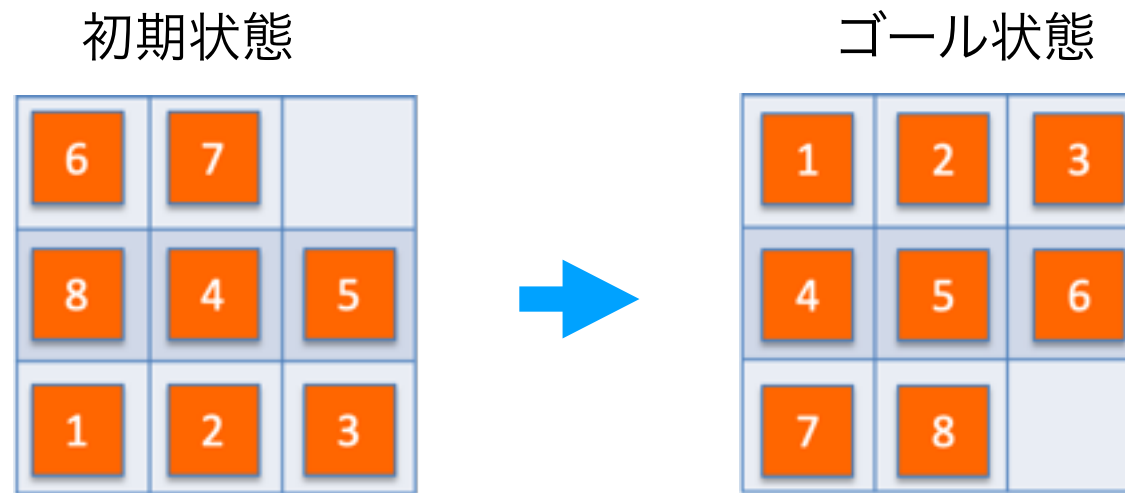
fが単調増大 \Leftrightarrow h: consistent



A*が展開のためにノードnを選択した時点でnに至る
最適経路が見つかった

∴ 最適経路が見つからない（後から見つかる）とする
⇒ 初期状態からnに至る最適経路上の別のノードn'で
frontier中にあるものが存在する
fは経路上で単調非減少だから $f(n) \geq f(n')$
従ってn'はnより先に見つからない（矛盾）

ヒューリスティック関数の例：8-Puzzleの場合



ヒューリスティック関数の例

1. $h1$: ゴール状態とは異なる位置にあるタイルの総数

タイルの移動でゴールと異なる位置にあるタイルの数が減る数は高々 1

⇒admissible heuristic

2. $h2$: 各タイルのゴール状態における位置との間のManhattan距離の和

※(p,q) と (x,y)間のManhattan距離は $=|p-x|+|q-y|$

ゴール状態にあれば $h2=0$

1回のタイル移動で(x,y)のうちどちらか1だけ変化するからManhattan距離の減少は高々 1

初期状態から距離のズレを 0 にするまでに少なくとも $h2$ 回の移動が必要

⇒admissible

許容的 (admissible) なヒューリスティック関数：

最適経路の正確な値 (理想的な推定値) 以下の推定値を与える

理想的な推定値に近づけるには？

→ 許容性を保ったままで、できるだけ大きな値を返すものが望ましい

2つのヒューリスティック関数の比較：

h が h' より優位にある (h dominates h') とは

任意のノード n について

$$h'(n) \leq h(n)$$

であることを言う。

8 Puzzleの場合の h_1 と h_2 の場合 $\Rightarrow h_2$ は h_1 より優位にある

\therefore 現在の状態とゴール状態を各タイル毎に比較すると

ゴール状態とは異なる状態にある各タイル

$$\Rightarrow 1 \leq \text{Manhattan距離}$$

従ってこれらの和を取ると $h_1(n) \leq h_2(n)$

h_2 による A^* の方が h_1 による A^* より効率的

d	Search Cost (nodes generated)			Effective Branching Factor		
	BFS	$A^*(h_1)$	$A^*(h_2)$	BFS	$A^*(h_1)$	$A^*(h_2)$
6	128	24	19	2.01	1.42	1.34
8	368	48	31	1.91	1.40	1.30
10	1033	116	48	1.85	1.43	1.27
12	2672	279	84	1.80	1.45	1.28
14	6783	678	174	1.77	1.47	1.31
16	17270	1683	364	1.74	1.48	1.32
18	41558	4102	751	1.72	1.49	1.34
20	91493	9905	1318	1.69	1.50	1.34
22	175921	22955	2548	1.66	1.50	1.34
24	290082	53039	5733	1.62	1.50	1.36
26	395355	110372	10080	1.58	1.50	1.35
28	463234	202565	22055	1.53	1.49	1.36

Figure 3.26 Comparison of the search costs and effective branching factors for 8-puzzle problems using breadth-first search, A^* with h_1 (misplaced tiles), and A^* with h_2 (Manhattan distance). Data are averaged over 100 puzzles for each solution length d from 6 to 28.

重み付きA*探索 (weighted A*)

A* : 最適経路を見つける、但し、多数のノードを展開してしまう可能性 ← メモリコスト大

⇒ 最適でなくてもよい (準最適) ... 条件緩和したら?

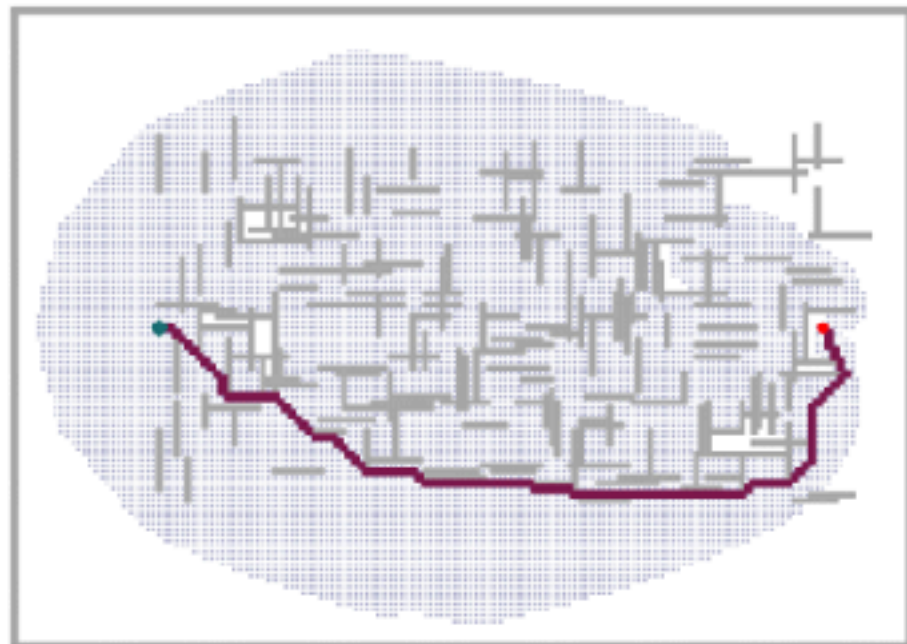
⇒ inadmissible heuristic (過大評価する可能性あり) を用いることを許容

➡ 重み付きA*-探索 (weighted A*) ... heuristic関数に重みを掛ける

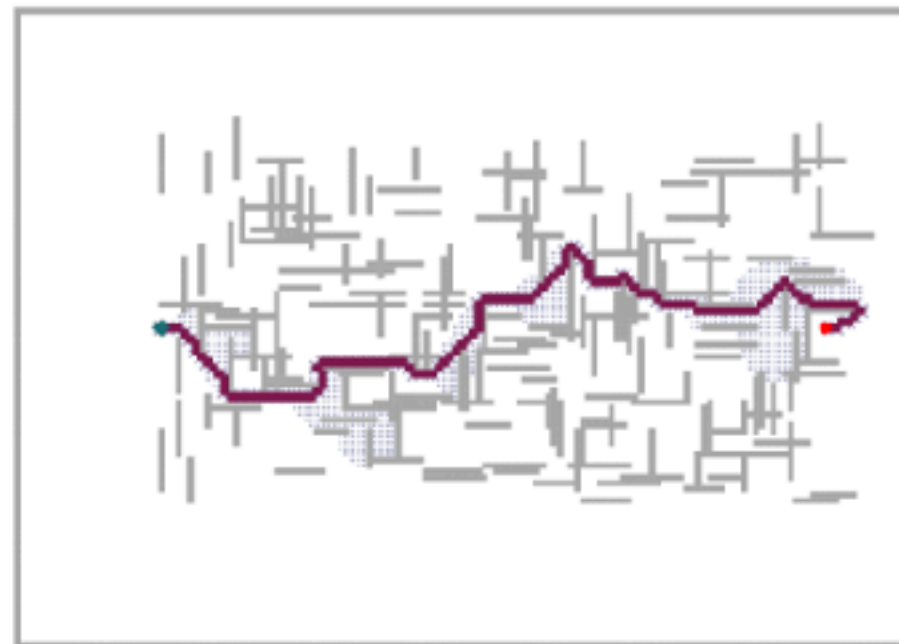
$$f(n) = g(n) + W \times h(n) \quad W > 1$$

⇒ $C^* \leq C \leq WC^*$ の解を見つける ⇒ 実用上は C^* に近い解が得られる

grid上の経路探索例 : 灰色の線分は障害物、dotはreached状態を示す



(a) A*



(b) weighted A* (W=2)

weighted A*の結果
costは最適値より5%大
探索した状態は1/7

各種探索アルゴリズムを重み付きA*とみなすと…

search algorithm	$f(n)$	
A*-search	$g(n) + h(n)$	$W = 1$
uniform-cost search	$g(n)$	$W = 0$
greedy best-first search	$h(n)$	$W = \infty$
weighted A*-search	$g(n) + W \times h(n)$	$1 < W < \infty$

メモリ制限探索 (memory-bounded search)

A*-探索アルゴリズム：最適解を探索…但し必要メモリ量が大きくなる可能性

→ `frontier`, `reached` で使用

メモリ使用を節約できるアルゴリズム

- ・ **Beam search** :

frontierのサイズ制限⇒プライオリティキューのサイズkを固定…最良のf値を持つ状態k個以下を保持
展開されて得られたノードのうちその他のノードは無視

- ・ **IDA* (Iterative-deepening A*)** :

反復深化探索 (IDS, iterative-deepening search) と同様の考え方をA*に適用

IDSがノードの深さに対して適用されたのに対し、f-値に関して制約 (cutoff) を設けて探索
繰り返しによるcutoffの更新→前の繰り返しの際にcutoffを超過したものの中での最小値

局所探索&最適化 (local search & optimization)

これまで扱ってきた問題：解が行動系列（ゴールまでの経路）で与えられる問題

→ 経路に拘らず、「良い」状態を探索したい

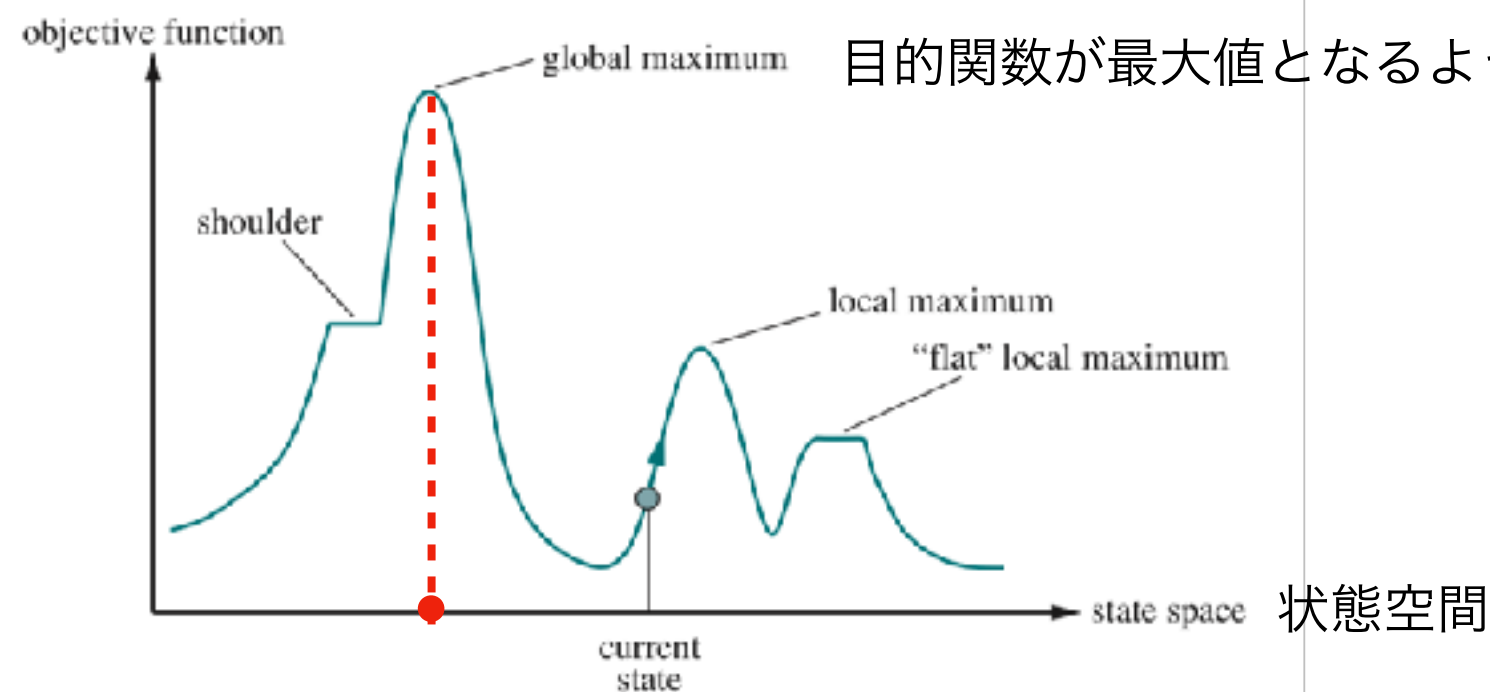
局所探索 (local search)

初期状態から開始し、周辺の状態（近傍..neighboring states）を探索…経路保持は考えない

⇒ 最適化問題 (optimization problem) の解の探索

…目的関数 (objective function) を最適化するような状態の探索

目的関数の値



目的関数が最大値となるような状態（大域的最適解）を見つけない

局所探索手法：山登り法 (Hill-climbing search)

局所探索による目的関数最大化⇒Hill-climbing

(目的関数を最小化したい⇒勾配降下法…gradient descent)

現在の状態を保持→目的関数の評価値が最大になる近傍の状態へ移動…steepest ascent

(目的関数の最小化の場合…steepest descent, 最急降下法)

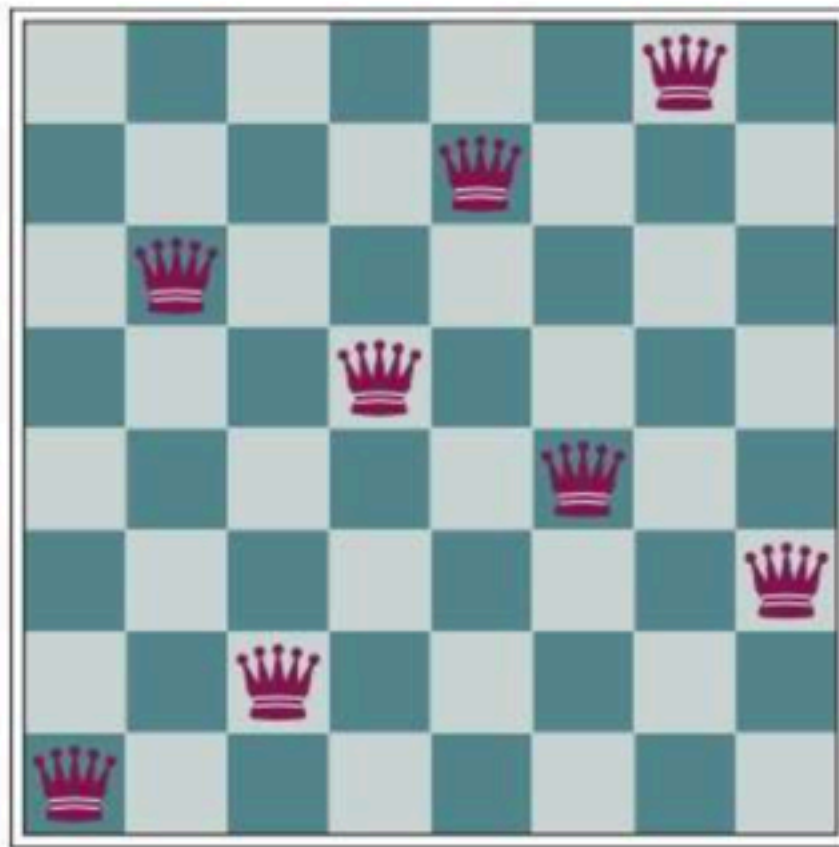
```
function HILL-CLIMBING(problem) returns 局所最適 (最大) 値を与える状態
    current ← problem.INITIAL
    while true do
        neighbor ← currentの後続状態で値が最も大きな状態
        if VALUE(neighbor) <= VALUE(current) then return current
        current ← neighbor
```



局所的なピークに到達したら終了 (局所最適解)

大域的な最適解とは限らない

8-queens問題



(a)



(b)

8x8のボードにクィーンを8個配置：同じ行/列/対角線上にないように配置したい

(a) クィーン配置例…同じ対角線上に配置されたクィーンのペアが存在するので解ではない

(b) heuristicコスト推定値 $h=17$

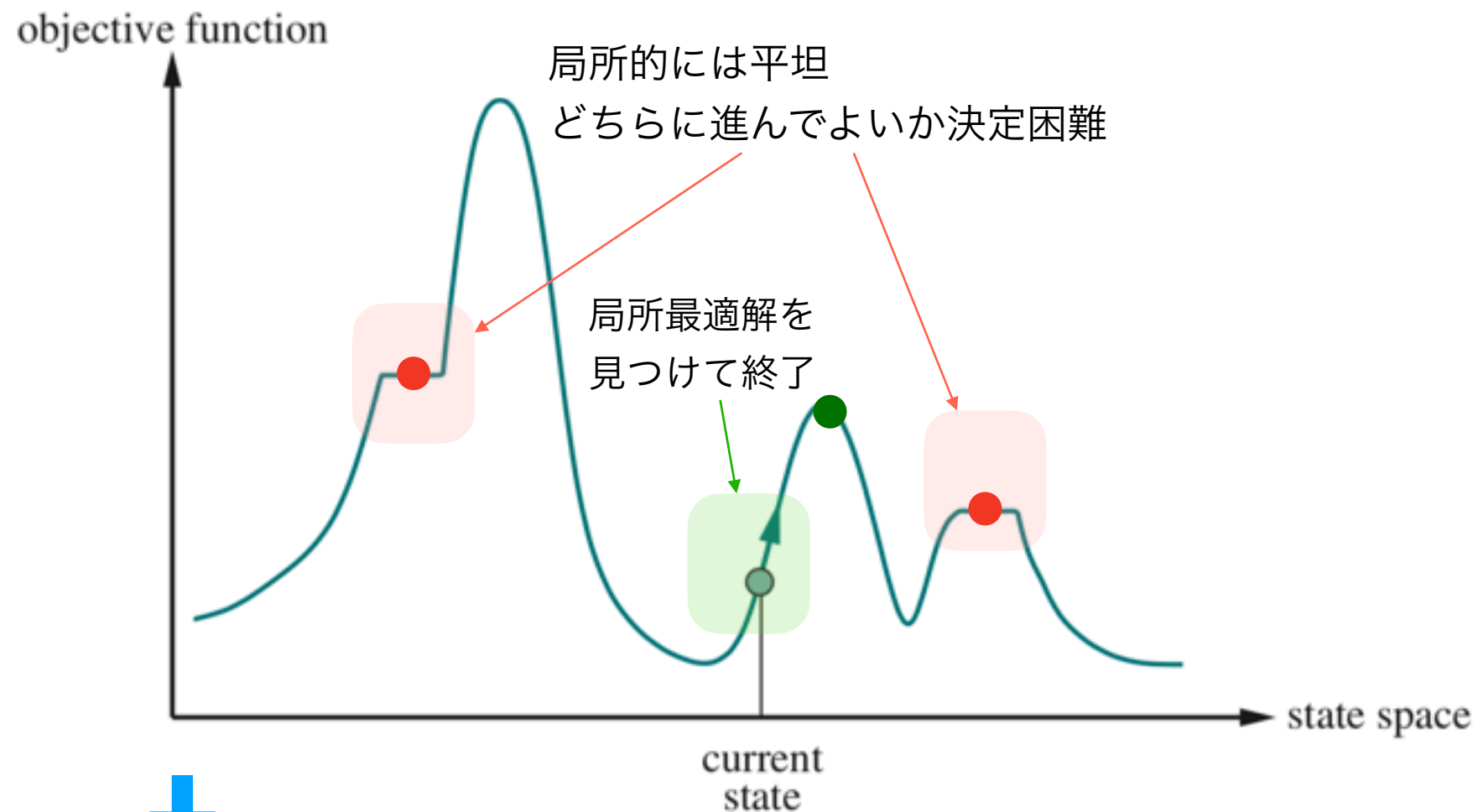
お互いに攻撃し合う関係にあるクィーンのペアの数をコスト推定値とする

8個のクィーンを各列に1つずつ配置→上下に移動→各マスに移動したときのコスト推定値

$h=12$ が最小…8通り存在

山登り法 = 貪欲局所探索 (greedy local search) … 近傍の値しか見ない
目先の最適 (最も良い) 状態に飛びつく

→ 解を見つけられずに終了する可能性



(大域的) 最適解の探索アルゴリズム

- ・ 焼きなまし法 (simulated annealing)
- ・ 局所ビームサーチ (beam search)
- ・ 遺伝的アルゴリズム (genetic algorithms)

などなど

Simulated Annealing

状態 s の評価 $\rightarrow \text{VALUE}(s) \cdots \text{VALUE}$ 値が最小となる状態を探索したい

現在の状態： current \rightarrow 次の状態候補： next

評価値の変動： $\Delta E = \text{VALUE}(\text{current}) - \text{VALUE}(\text{next})$

山登り法 \Rightarrow 次の状態候補の中で ΔE が最大となる next へ移動していく

Simulated Annealing

- current の次の状態 next を近傍からランダムに選択 $\Rightarrow \Delta E = \text{VALUE}(\text{current}) - \text{VALUE}(\text{next})$
- $\Delta E > 0$ のとき状態変化を受け入れる
- $\Delta E < 0$ の（評価値悪化）の場合も一定の確率で状態変化を許容： $P = \exp(\Delta E/T)$
- T ：温度パラメータ \Rightarrow 最初高温 \cdots 状態変化確率高 \rightarrow 徐々に冷却 \cdots 評価値悪化の確率が徐々に低下

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state

current \leftarrow *problem*.INITIAL

for $t = 1$ **to** ∞ **do**

T \leftarrow *schedule*(t)

if $T = 0$ **then return** *current*

next \leftarrow a randomly selected successor of *current*

$\Delta E \leftarrow \text{VALUE}(\text{current}) - \text{VALUE}(\text{next})$

if $\Delta E > 0$ **then** *current* \leftarrow *next*

else *current* \leftarrow *next* only with probability $e^{\Delta E/T}$

局所ビームサーチ (local beam search)

山登り法：近傍中の最適な状態へ移動

→ 最適な 1 つの状態を保持するのではなく、複数個 ($k > 0$) 個の状態を保持

初期狀態：

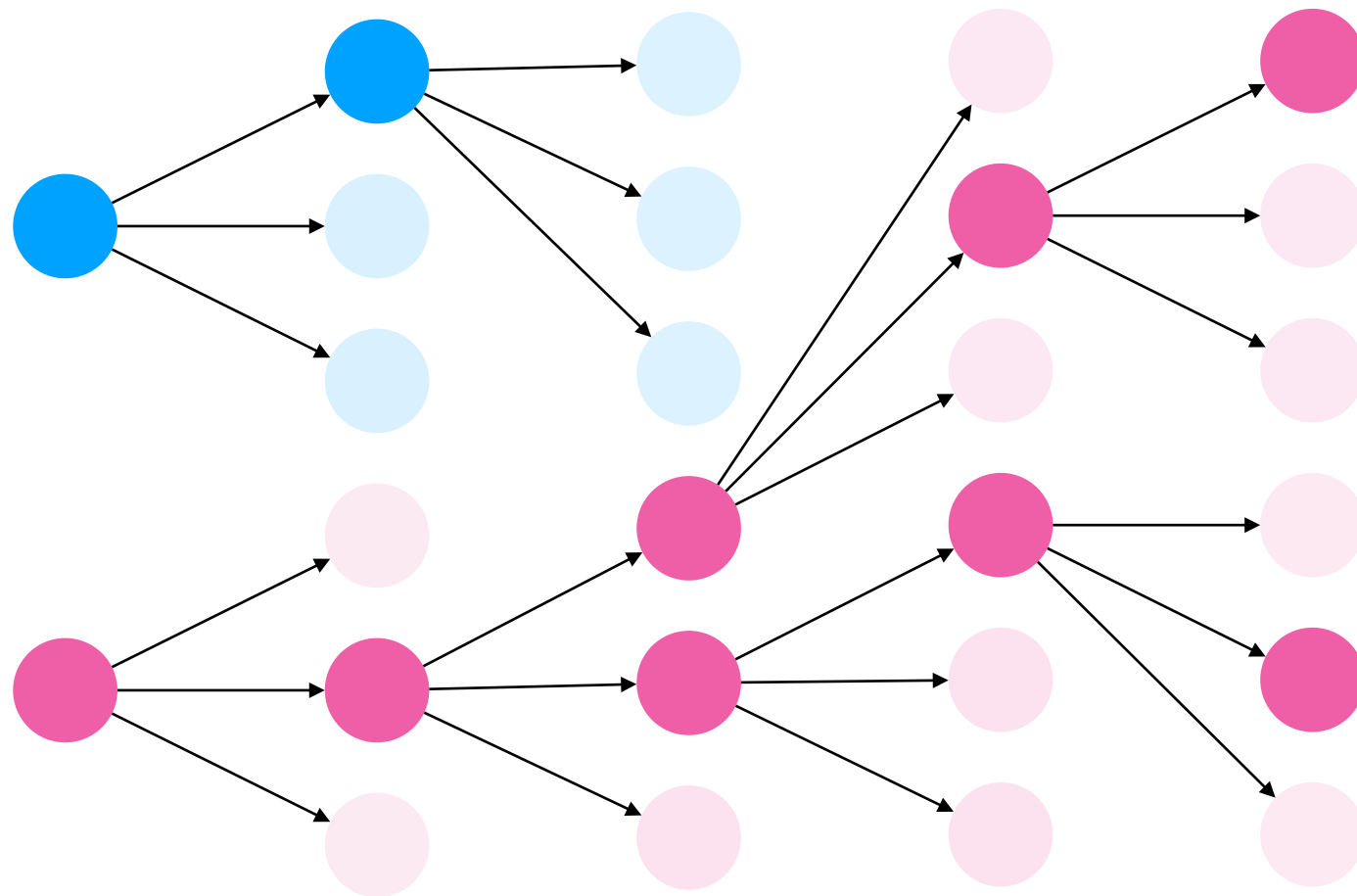
k個のランダムに生成された状態

各ステップで：

保持しているk個の状態について全ての後続状態 (successors) を生成

if その中にゴール状態が存在すれば終了

else successorsの中のベストの状態k個を選択して繰り返し



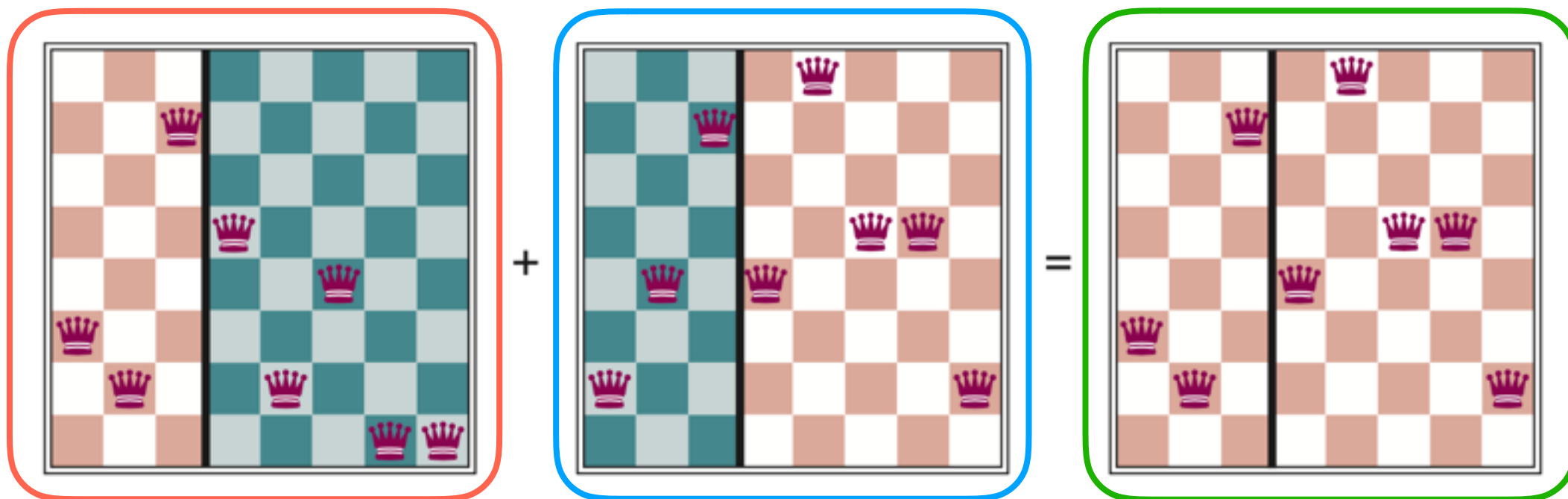
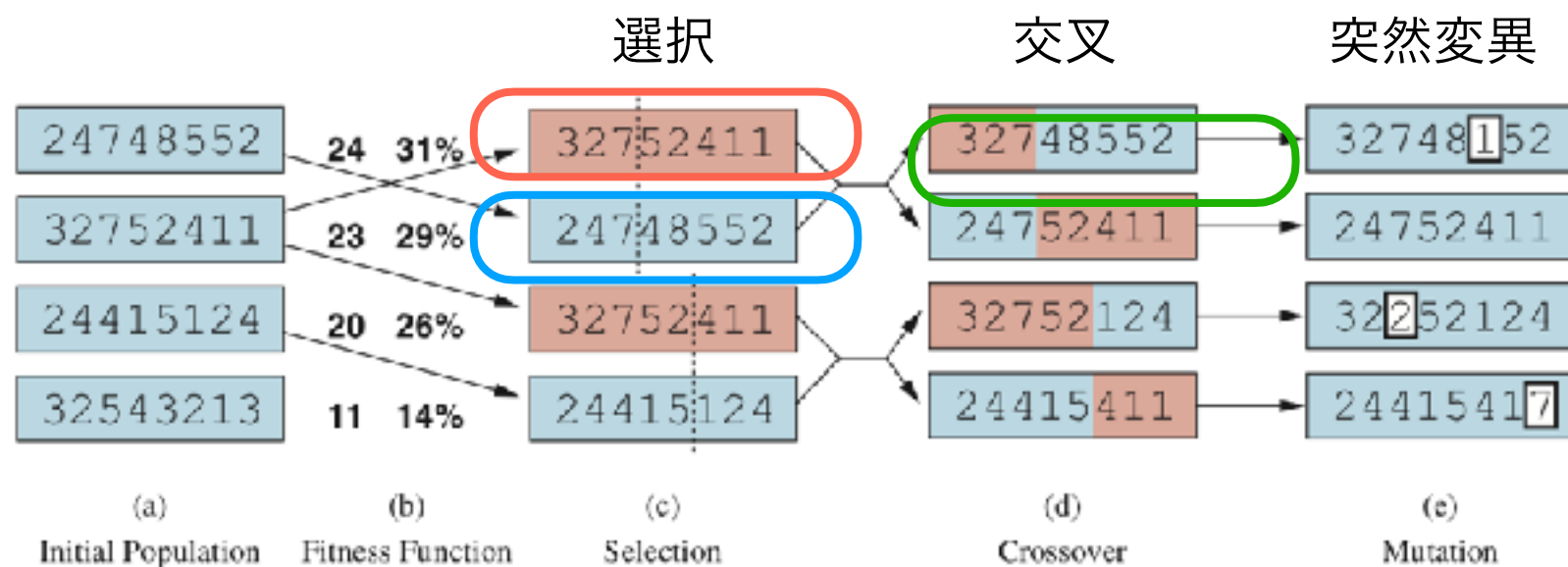
遺伝的アルゴリズム (genetic algorithm)

生物の進化のプロセス（自然淘汰…natural selection）と類似のプロセスによる探索

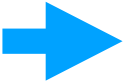
- 個体の集合を用意する
各個体は文字列（string）で表現される（遺伝子の場合は文字A,T,C,G上の文字列）
各個体について評価関数を用いて適合度（fitness score）を計算できる
- 個体 ρ 個から子（offspring）が生成される…通常 $\rho=2$ が多い
→ 親の遺伝子から子の遺伝子が生成される
- 選択（selection）：
個体の集合（親世代の集合）から次世代の親となる個体を選択
e.x. 全個体から適合度に比例した確率で選択
または…ランダムに n 個（ $n > \rho$ ）の個体を選択し、その中で適合度最大の ρ 個を選択
- 組み替え（recombination）
 $\rho=2$ を仮定：親文字列（ $\rho=2$ 個）をランダムに選択した交叉点（crossover point）で分割し
半分ずつ入れ替えて文字列を合成（次スライド参照）→ 2 つの子を生成
- 突然変異
得られた文字列上でランダムなシンボル変化を一定の確率で許容
- 次世代の生成
新たに得られた子から構成または親世代の中から高スコアの個体を含めて次世代集合を生成

遺伝的アルゴリズム (genetic algorithm) の8-queens問題への適用例

string中の数字は各列のクイーン的位置 (y)

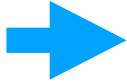


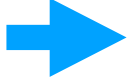
連続状態空間の局所探索：最適化

評価関数： $f(x)$ $x \in \mathbb{R}^d$  $x^* = \arg \max f(x)$ を求めたい…機械学習ではこの種のタスク多数出現

x を微小変動させて $f(x)$ の値を増加させたい…どう動かせばよい？…どのような方向か考えてみる

$x' = x + \varepsilon u$, $\varepsilon > 0$, $\|u\| = 1$, $u \in \mathbb{R}^d$ (ε は微小正数) として、適切な方向 u を考えてみる

 Taylor展開 (1次近似) すると $f(x') = f(x + \varepsilon u) = f(x) + \varepsilon u^T \nabla f(x)$

 $u \propto \nabla f(x)$ … steepest ascent (最小化の場合は steepest descent, 最急降下法)

$$x' = x + \alpha \nabla f(x)$$


Newton-Raphson法

Taylor 展開 (2次近似) してみると…

$$f(x') = f(x + (x' - x)) = f(x) + (\nabla f(x))^T (x' - x) + \frac{1}{2} (x' - x)^T H_f(x) (x' - x)$$

ここに $H_f(x)$ は Hesse 行列… $(H_f(x))_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}(x)$

この2次式を最小化するのは：

 $x' = x - H_f(x)^{-1} \nabla f(x)$