

知識に基づく探索手法

Chapter 4

Material

- Chapter 4 Section 1 - 3
- Exclude memory-bounded heuristic search

概要

- 最良優先探索
- 欲張り探索
- A* 探索
- 発見的方法
- 局所探索アルゴリズム
- 山登り法
- 焼きなまし法
- 局所ビーム探索
- 遺伝的アルゴリズム

復習：木探索

function TREE-SEARCH(*problem*, *fringe*) **return** a solution or failure

fringe ← INSERT(MAKE-NODE(INITIAL-STATE[*problem*]), *fringe*)

loop do

if EMPTY?(*fringe*) **then return** failure

node ← REMOVE-FIRST(*fringe*)

if GOAL-TEST[*problem*] applied to STATE[*node*] succeeds

then return SOLUTION(*node*)

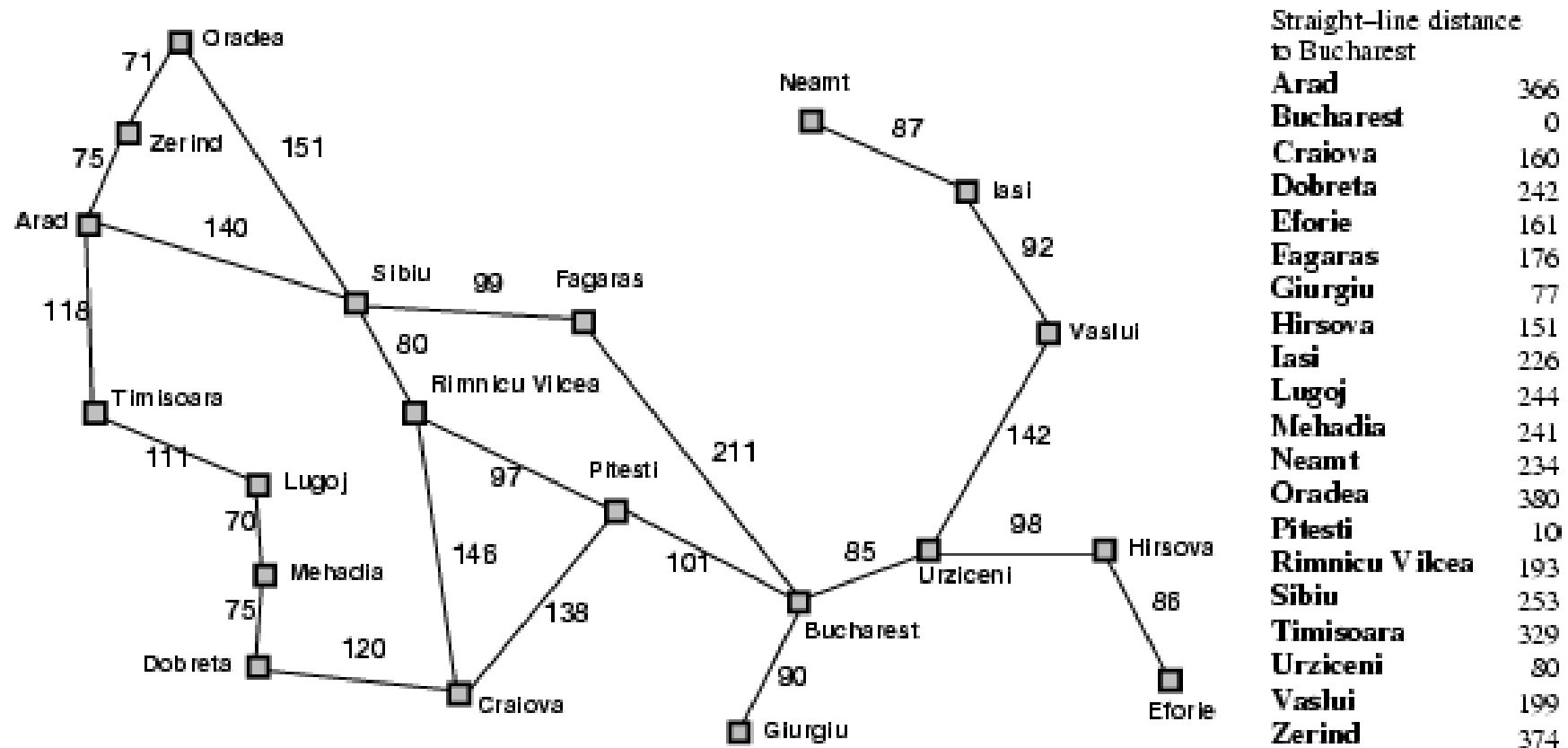
fringe ← INSERT-ALL(EXPAND(*node*, *problem*), *fringe*)

- 探索戦略はノードの拡張順位を選択することで定義

最良優先探索

- 考え方: それぞれのノードで**評価関数** $f(n)$ を使う
 - “望ましさ”の評価
 - もっとも望ましいと思われるまだ拡張されていないノードを拡張する
- 実現:
望ましさで下降するように木のふちでノードに順位付けをする
- 特別な場合:
 - 欲張り探索
 - A* 探索

Romania with step costs in km



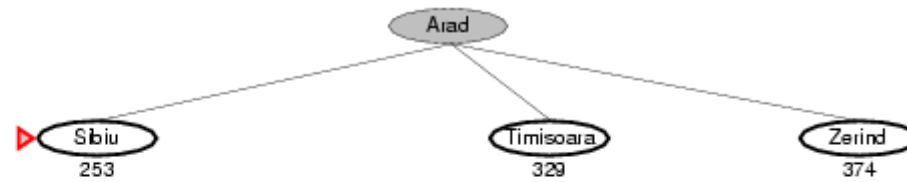
欲張り探索

- 評価関数 $f(n) = h(n)$ (heuristic)
- = n からゴールへのコストを見積もる
- 例: $h_{SLD}(n) = n$ to からBucharestへの直線距離
- 欲張り探索はゴールに最も近いと思われるノードを拡張する

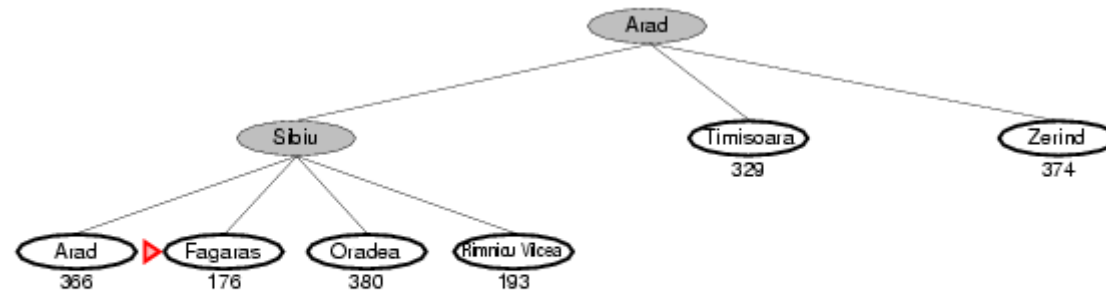
欲張り探索: 例



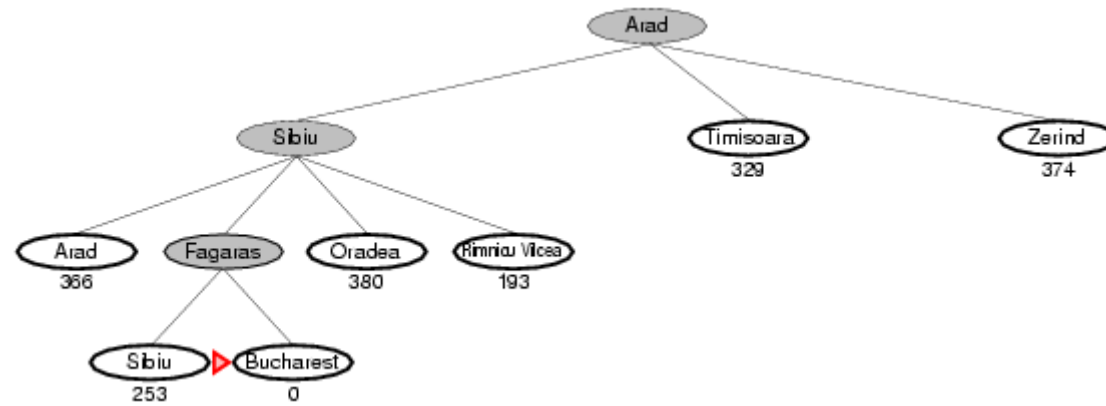
欲張り探索: 例



欲張り探索: 例



欲張り探索: 例



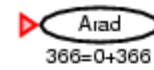
欲張り探索の性質

- Complete? No – ループで行き詰る
例: lasi → Neamt → lasi → Neamt →
- Time? $O(b^m)$, よい発見的方法は劇的な改良をもたらす
- Space? $O(b^m)$ – メモリに全てのノードを記憶
- Optimal? No

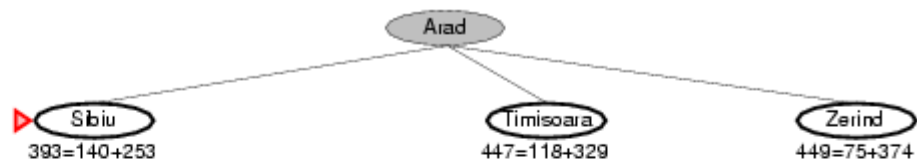
A* 探索

- 考え方: 既にコストがかかりすぎると分かっている経路を避ける
- 評価関数 $f(n) = g(n) + h(n)$
- $g(n) = n$ にたどり着いたコスト
- $h(n) = n$ からゴールへの見積もりコスト
- $f(n) = n$ を經由してゴールへ至る経路の見積もられた総コスト

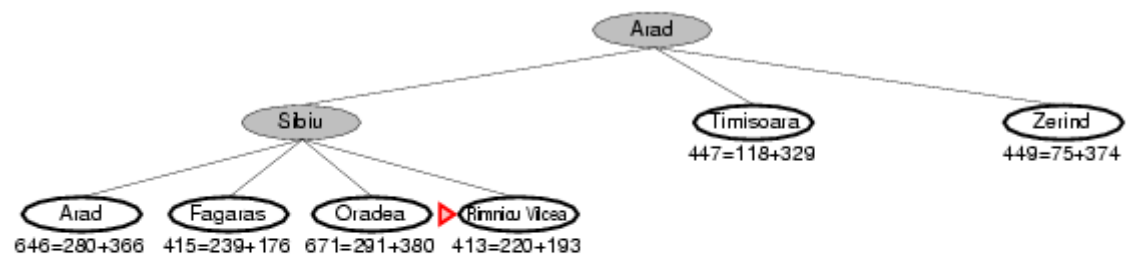
A* 探索: 例



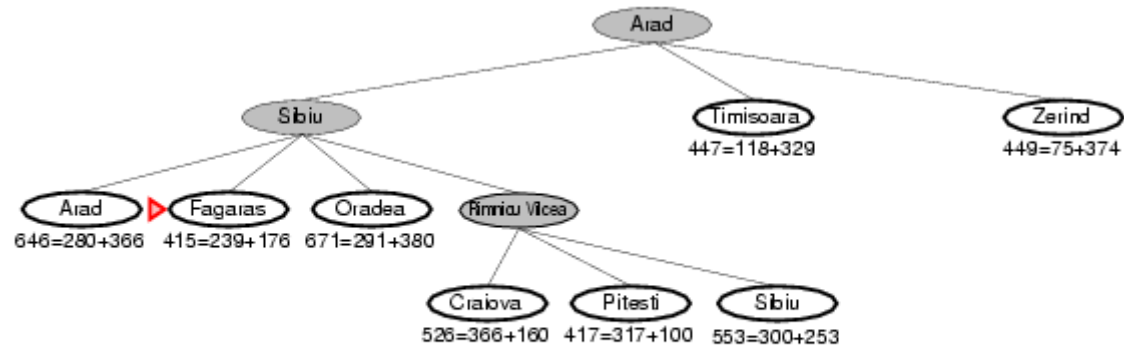
A* 探索: 例



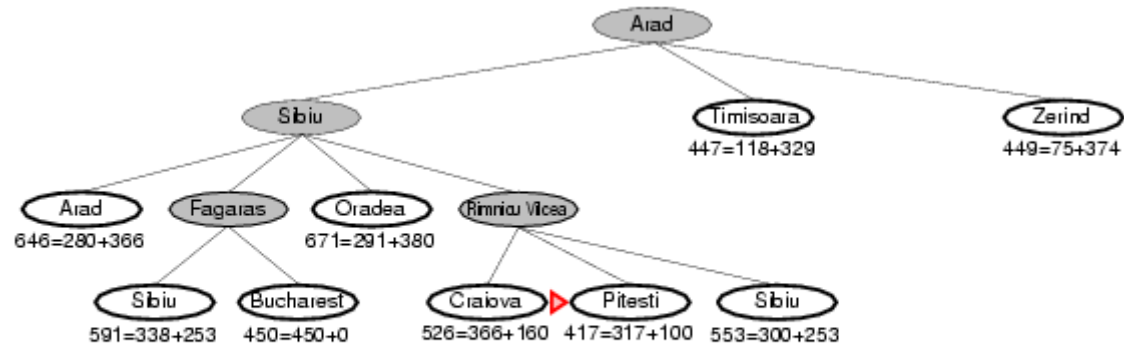
A* 探索: 例



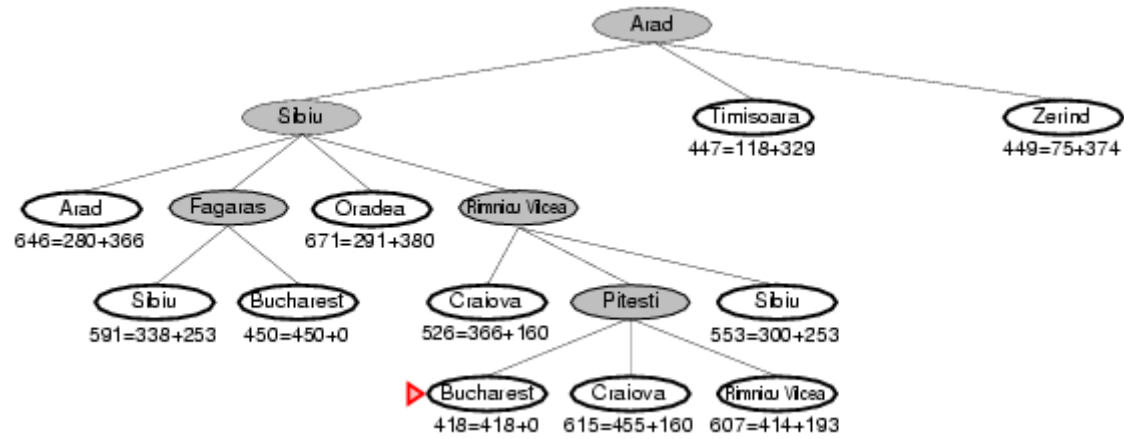
A* 探索: 例



A* 探索: 例



A* 探索: 例

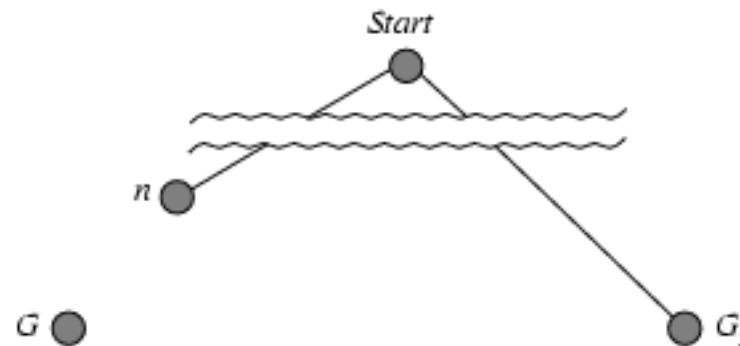


許容的な発見的方法

- 全てのノード n に対して、 $h^*(n)$ が n からゴールに至る真のコストであるとき、 $h(n) \leq h^*(n)$ が成り立てば、発見的方法 $h(n)$ は許容的である
- 許容的な発見的方法は決してゴールに至るコストを過大評価することはない
- 例: $h_{SLD}(n)$ (実際の道の長さを過大評価しない)
- 定理: $h(n)$ が許容的であるとき、TREE-SEARCH を用いる A^* 探索は最良である

A* 探索の最適性 (証明)

- 次善の最適なゴール G_2 が生成されたとし、フリンジに蓄えられているとする。最適なゴール G への最短経路上にある拡張されていないノードを n とする



- $f(G_2) = g(G_2)$
- $g(G_2) > g(G)$
- $f(G) = g(G)$
- $f(G_2) > f(G)$

since $h(G_2) = 0$

since G_2 is suboptimal

since $h(G) = 0$

from above

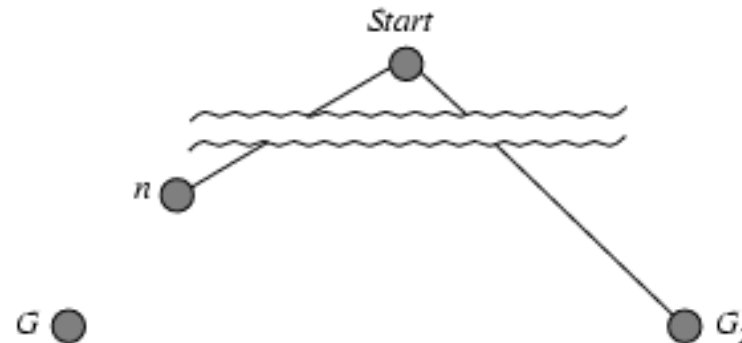
$g(n)$ = n にたどり着いたコスト

$h(n)$ = n からゴールへの見積もり
コスト

$f(n)$ = n を経由してゴールへ至る
経路の見積もられた総コスト

A* 探索の最適性(証明)

- 次善の最適なゴール G_2 が生成されたとし、フリンジに蓄えられているとする。最適なゴール G への最短経路上にある拡張されていないノードを n とする



- $f(G_2) > f(G)$ from above
- $h(n) \leq h^*(n)$ since h is admissible
- $g(n) + h(n) \leq g(n) + h^*(n)$
- $f(n) \leq f(G)$

$h^*(n) = n$ から
ゴールへの真
のコスト

従って $f(G_2) > f(n)$ 、A* は決して G_2 を拡張しようとはしない

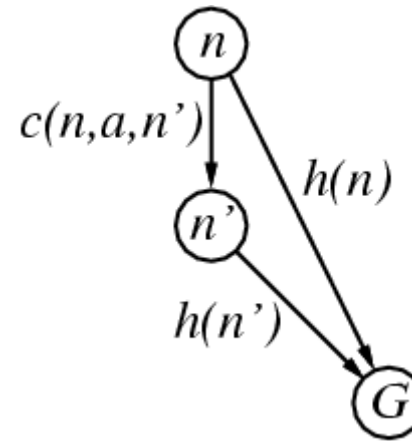
一貫性のある発見的方法

- 全てのノード n と、 n からいかなる動作 a によって生成される n に続く全てのノード n' に対して、次のことが成り立てば発見的方法は**一貫性**がある

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

- h に一貫性があるなら、次のことが成り立つ

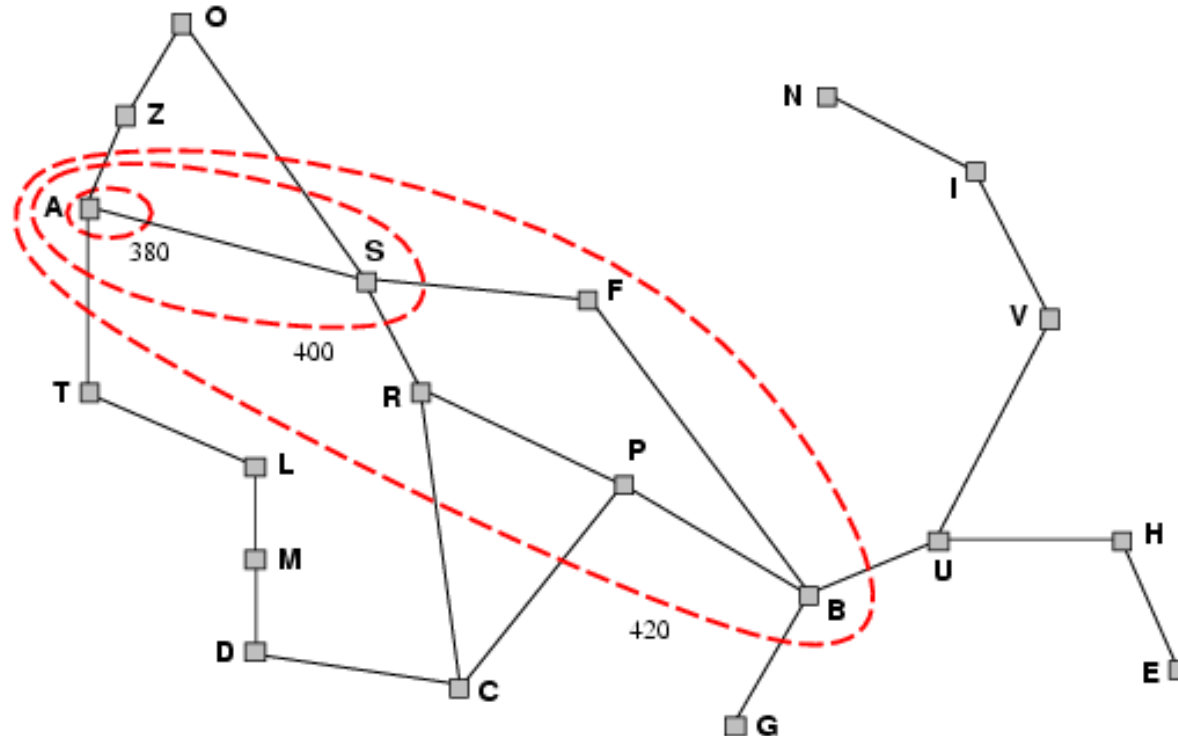
$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n,a,n') + h(n') \\ &\geq g(n) + h(n) \\ &= f(n) \end{aligned}$$



- 即ち $f(n)$ はいかなる経路においても減少することはない
- **定理:** $h(n)$ が一貫性があるなら GRAPH-SEARCH を用いる A^* 探索は最良である

A* 探索の最適性

- A* 探索は f の値を増大させる順にノードを拡張する
- だんだんにノードの “ f -等高線” を付け加える
- 等高線 i は $f=f_i$ を全て持っている。ここで $f_i < f_{i+1}$



A* 探索の性質

- Complete? Yes ($f \leq f(G)$ であるノードが無数にない限り)
- Time? 級数的
- Space? 全てのノードをメモリに記憶
- Optimal? Yes

許容的な発見的方法

例: 8パズル

- $h_1(n)$ = 間違って置かれているタイルの数
- $h_2(n)$ = マンハッタン距離の総和
(正しい場所へのタイルの数)

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

- $h_1(S) = ?$
- $h_2(S) = ?$

許容的な発見的方法

例: 8パズル

- $h_1(n)$ = 間違って置かれているタイルの数
- $h_2(n)$ = マンハッタン距離の総和
(正しい場所へのタイルの数)

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

- $h_1(S) = ?$ 8
- $h_2(S) = ?$ $3+1+2+2+2+3+3+2 = 18$

優勢

- すべての n に対して $h_2(n) \geq h_1(n)$ (両方とも許容的)
- h_2 は h_1 に **優勢** である
- h_2 は探索に向いている
- 典型的な探索コスト (拡張されるノードの数):
 - $d=12$ 反復深化探索 = 3,644,035 nodes
 - $A^*(h_1) = 227$ nodes
 - $A^*(h_2) = 73$ nodes
 - $d=24$ 反復深化探索 = too many nodes
 - $A^*(h_1) = 39,135$ nodes
 - $A^*(h_2) = 1,641$ nodes

弛緩問題

- 動作に対して制限を緩くした問題を**弛緩問題**と呼ぶ
- 弛緩問題にしたときの最良解のコストは最初の問題を許容的発見的手法にしたものである
- タイルを**どこへでも**移動できるように8パズルのルールを緩めたとき、 $h_1(n)$ は最短の解を与える
- タイルが**どの隣の四角**でも移動できるとルールを緩めたとき、 $h_2(n)$ は最短の解を与える

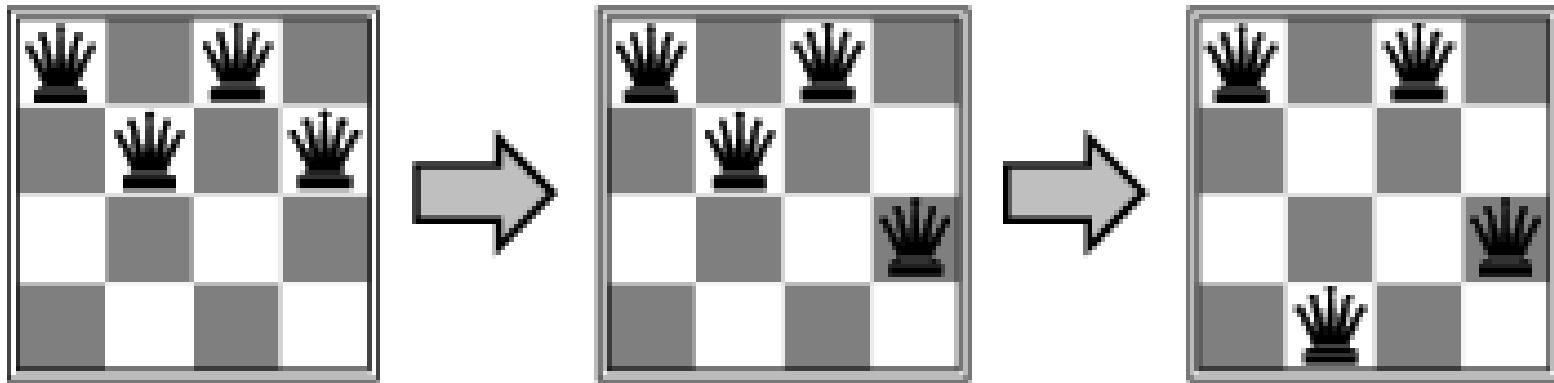
局所探索アルゴリズム

- 多くの最適化問題では、ゴールへの**経路**には無関係で、ゴールそれ自身が解である
- 状態空間 = “完全な” 構成の集合
- 制約を満たす構成を見つける。例: nクイーン
- このような場合 **局所探索アルゴリズム** を用いる
- 一つの “現在の” 状態を維持して、それを改良する

ゴールへの見積りができない

例:nクイーン

- どの二つのクイーンともたて、横、斜めに並ばないように $n \times n$ のボードに n のクイーンを置く



山登り法

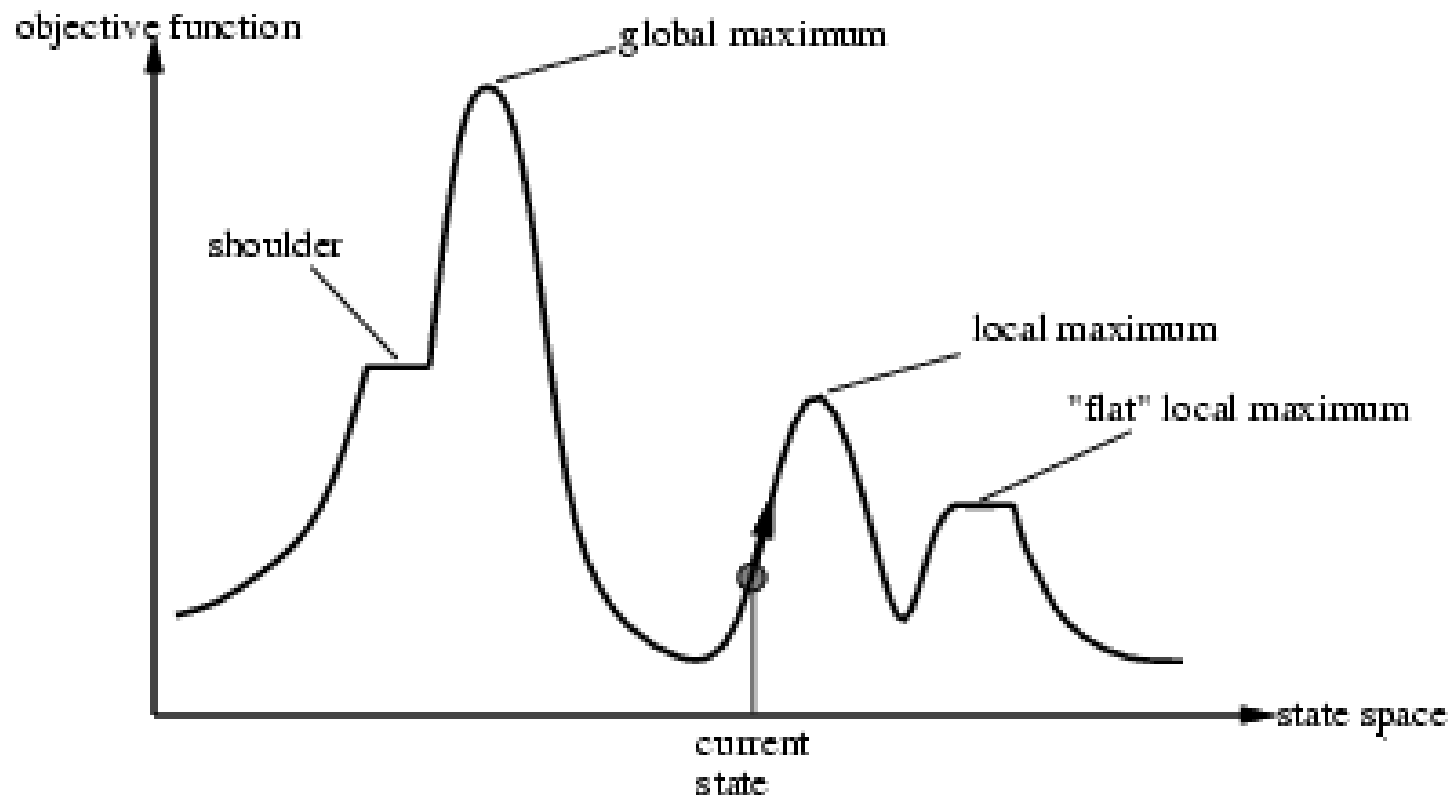
- “健忘症を伴いながら深い霧の中エベレストに登るよう”

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  inputs: problem, a problem
  local variables: current, a node
                  neighbor, a node

  current ← MAKE-NODE(INITIAL-STATE[problem])
  loop do
    neighbor ← a highest-valued successor of current
    if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
    current ← neighbor
```

山登り法

- 問題: 初期状態によって局所最大に陥る

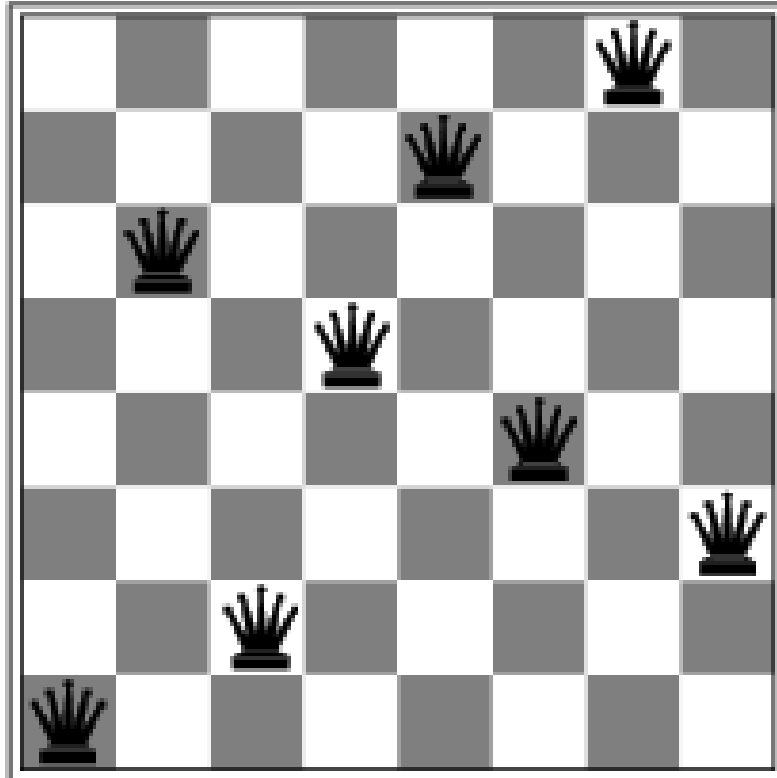


山登り法: 8クイーン問題

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	♙	13	16	13	16
♙	14	17	15	♙	14	16	16
17	♙	16	18	15	♙	15	♙
18	14	♙	15	15	14	♙	16
14	14	13	17	12	14	12	18

- h = 直接、あるいは、間接に互いにアタックしているクイーンの数
- $h = 17$ (上の図)

山登り法: 8クイーン問題



- A local minimum with $h = 1$

焼きなまし法

- 考え方: ある“良くない”動きを許すことで局所最大を抜け出すが、その回数をだんだんに減らす

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
         schedule, a mapping from time to “temperature”
  local variables: current, a node
                  next, a node
                  T, a “temperature” controlling prob. of downward steps

  current ← MAKE-NODE(INITIAL-STATE[problem])
  for t ← 1 to ∞ do
    T ← schedule[t]
    if T = 0 then return current
    next ← a randomly selected successor of current
     $\Delta E$  ← VALUE[next] - VALUE[current]
    if  $\Delta E > 0$  then current ← next
    else current ← next only with probability  $e^{\Delta E/T}$ 
```

焼きなまし法の性質

- 証明せよ: T が十分にゆっくり下がるなら、焼きなまし法は確率が1に近づきながら大局的な最適を見出すことができる
- VLSIのレイアウト、飛行機のスケジューリングなどで広く使われている

局所ビーム探索

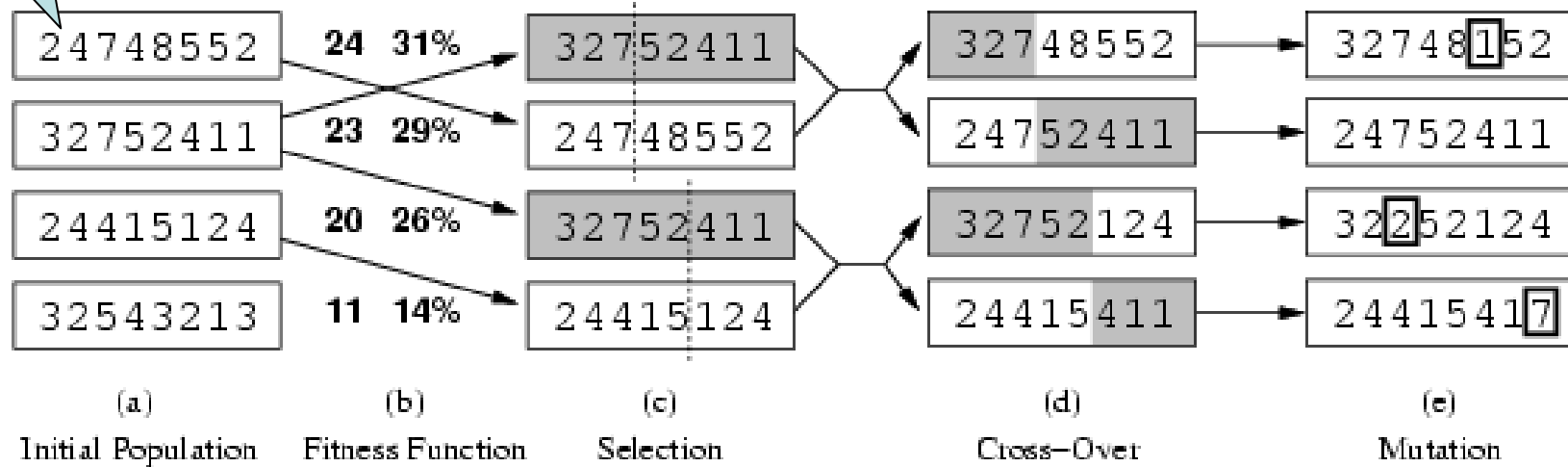
- 一つではなく k 状態のトラックを維持する
- K 個のランダムに生成した状態でスタートする
- 各繰り返しで、 k 状態に次に続く全ての後継の状態を生成する
- どれか一つがゴール状態であれば停止する。そうでなければ、完全なリストから k 個の最良の後継の状態を選び、繰り返す

遺伝的アルゴリズム

- 後継の状態は二つの親を結びつけることで生成される
- K 個のランダムに生成した状態でスタートする(**population**)
- 有限のアルファベットの文字列として状態は表される (often a string of 0s and 1s)
- 評価関数 (**fitness function**). 良い状態に対しては高い得点
- 選択、交叉、突然変異により次の世代を選ぶ

遺伝的アルゴリズム

各列でのクイーンのある行番号



- Fitness function: number of non-attacking pairs of queens (min = 0, max = $8 \times 7/2 = 28$)
- $24/(24+23+20+11) = 31\%$
- $23/(24+23+20+11) = 29\%$ etc

遺伝的アルゴリズム

