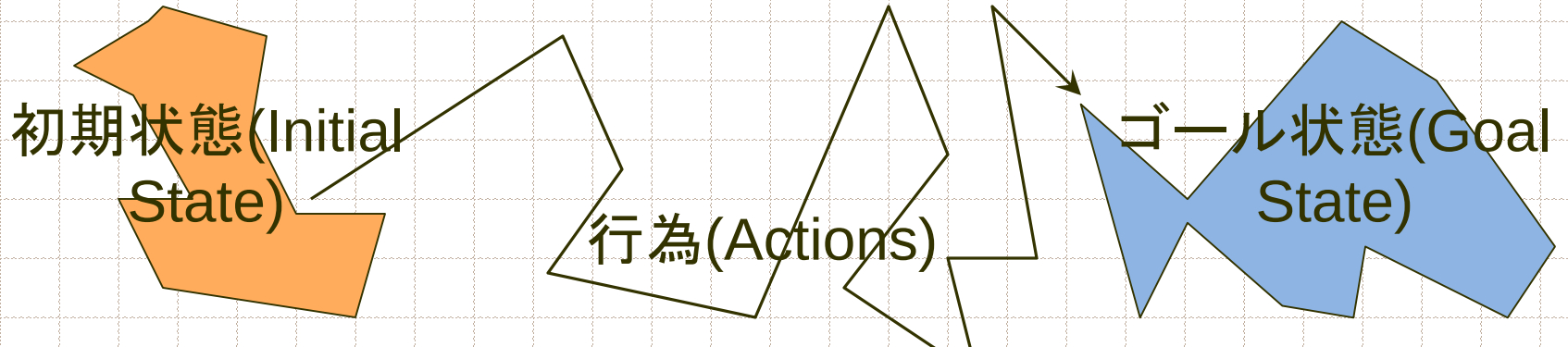
知能システム学
第9回-10回

探索による問題解決(3)、
情報を持つ探索

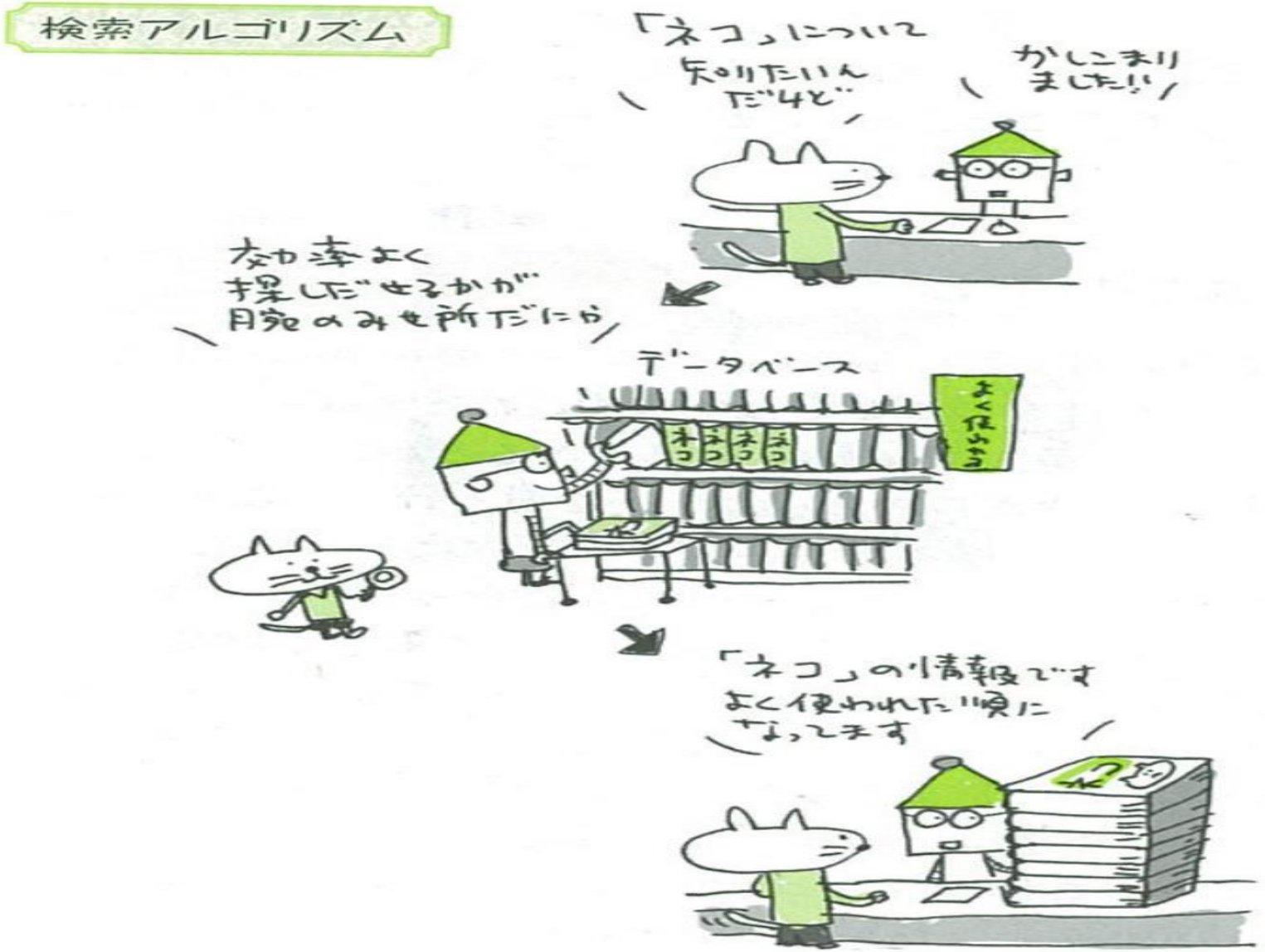
ソフトウェア情報学部
David Ramamonjisoa

ゴールを用いるエージェントの構築

- エージェントの環境を状態で表現する(**state**)?
- エージェントのゴールは何か(**goal** to be achieved?)
- 可能な行為は何かある(What are the **actions**?)
- 問題を解決するためにどのような情報が状態と状態遷移に記述するべきか



検索アルゴリズム



絵でわかる人工知能（三宅）

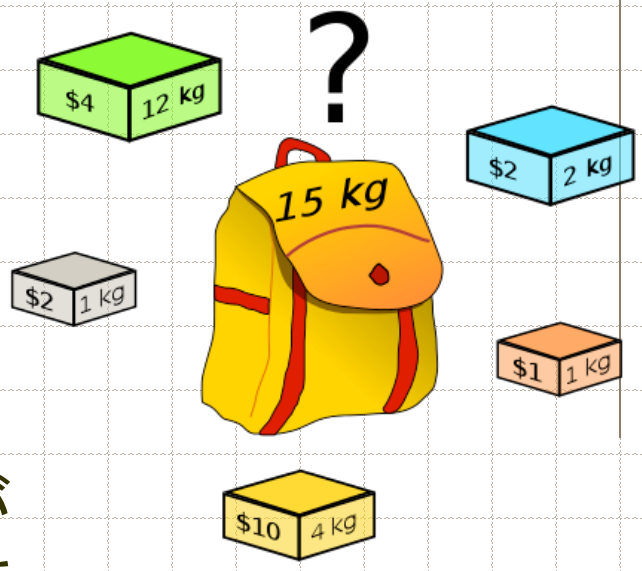
探索戦略

- 探索戦略選択のための四つの基準
  - 完全性： 解が存在するとき，それを見つけることが保証されているか？
  - 最適性： いくつか異なる解があるとき，戦略は最も良い解を見つけるか？
  - 時間計算量： 解を見つけるまでにどれくらい時間が掛かるか？
  - 空間計算量： 探索を行うためにどのくらいメモリを必要とするか？
- 情報を持つ探索(informed search)，ある探索
  - 現在の状態からゴールに至る順路の中で、最小コストの順路（最適順路）などを考えて効率的に行う.

情報をもつ探索戦略(Informed search strategies)

- 情報をもつ探索戦略は問題定式化の使用可能情報のみ利用する。
- 欲張り探索(greedy search)
- A*探索(A* search)

ナップサック問題

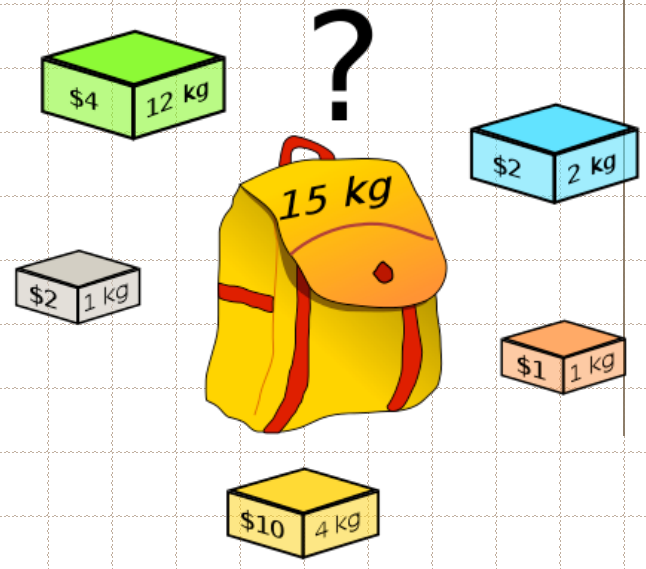


容量 C のナップサックが一つと、
n 種類の品物（各々、価値 pi, 容積 ci）が
与えられたとき、ナップサックの容量 C を
超えない範囲でいくつかの品物をナップサック
に詰め、ナップサックに入れた品物の価値の和を最大化する
にはどの品物を選べばよいか

貪欲アルゴリズムを用いて、問題を解く

https://ja.wikipedia.org/wiki/ナップサック問題

ナップサック問題



品物リスト＝[（12kg,$4),(1kg,$2),(2kg,$2),(1kg,$1),(4kg,$10)]

貪欲アルゴリズムを用いて、問題を解く

https://ja.wikipedia.org/wiki/ナップサック問題

貪欲アルゴリズム

- 最も品物の重さが15kgという基準ならば、
- 品物のリストを重さの順にソートする
- ソートされた品物を一個づつに**ナップサックにいれる。品物の重さの合計は15kgまで続く。**

品物リスト＝[（12kg,$4) ,(4kg,$10), (2kg,$2), (1kg,$2),(1kg,$1)]

ナップサック＝[（12kg,$4) , (2kg,$2), (1kg,$2)]

重さの合計(15kg)＝12kg+2kg+1kg

値段の合計＝4+2+2=$8

貪欲アルゴリズム

- 最も品物の重さが15kgかつ多く品物をナップサックに入れるという基準ならば、
- 品物のリストを重さの逆順にソートする
- ソートされた品物を一個づつに**ナップサックにいれる。品物の重さの合計は15kgまで続く。**

品物リスト＝[(1kg,$2),(1kg,$1),(2kg,$2),(4kg,$10),(12kg,$4)]

ナップサック＝[(1kg,$2),(1kg,$1),(2kg,$2),(4kg,$10)]

重さの合計(max 15kg)＝1kg+1kg+2kg+4kg=8kg

値段の合計＝2+1+2+10=$15

貪欲アルゴリズム

- 最も品物の値段が高いという基準ならば、
- 品物のリストを値段の順にソートする
- ソートされた品物を一個づつに**ナップサックにいれる。品物リストの中から品物の重さの合計は15kgまで続く。**
  - for item in itemList:
  - 　　if (totalWeight+item)<=15: …

品物リスト＝[(4kg,$10), (12kg,$4) , (2kg,$2), (1kg,$2),(1kg,$1)]

ナップサック＝[(4kg,$10) , (2kg,$2), (1kg,$2),(1kg,$1)]

貪欲アルゴリズム

- 最も値段が高いものか、最も重いものか
- 最も重さが15kgという意味だとすれば、
  - 12kg（＄4）＋2kg（＄2）＋1kg（＄2or＄1）＝15kg（＄8）

  が解となる。
- 最も値段が高いという意味だとすれば、
  - 4kg（＄10）＋2kg（＄2）＋1kg（＄2）＋1kg（＄1）＝8kg（＄15）

  が解となる。

情報を持つ探索（ヒューリスティック探索）

- 情報のある探索は、ヒューリスティック探索(heuristic search)ともいう。
- 深さ優先、あるいは幅優先探索に、ヒューリスティックスを導入して、最も見込みのありそうな枝から探索を進める。
- ヒューリスティックスとしては、たとえば、ゴールに最も近づく道を選ぶ。

情報を持つ探索

- 料金と交通手段に問わずに時間的に短い
- 高速道路が一番
- 情報は岩手県立大学から盛岡駅までのお予想かかる時間のリスト

→これがヒューリスティック

最良優先探索 - 欲張り探索 (Greedy Best first search)

- ゴールへ到達するのに予想されるコストを最小化する
  - ゴール状態に最も近いと判断した状態へ到達するまでのコストを見積もりする

function Best-First-Search(*問題*, *Greedy-Search*) returns 一つの解 or 失敗
  *問題*の初期状態で，探索木を初期設定する.
  loop do
    if 展開すべき候補ノードがない **then return** 失敗
    *Greedy-Search(問題)*によって展開すべき葉ノードを選ぶ
    if そのノードがゴール状態 **then return** 対応する解
    else そのノードを展開して結果のノード群を探索木に加える
end

探索戦略

- 探索戦略選択のための四つの基準
  - **完全性**： 解が存在するとき，それを見つけることが保証されているか？
  - **最適性**： いくつか異なる解があるとき，戦略は最も良い解を見つけるか？
  - **時間計算量**： 解を見つけるまでにどれくらい時間が掛かるか？
  - **空間計算量**： 探索を行うためにどのくらいメモリを必要とするか？
- 情報ある探索
  - 現在の状態からゴールに至るステップ数や経路コストに関する情報を持っている．ここでは，情報ある探索を取り上げる．

評価関数

- 決定を行うための知識は、評価関数 f(n) (evaluation function)、すなわちその節点を展開することの好ましさ（あるいは好ましくない）に比例した数値を返す関数

- ヒューリスティック関数
  - コスト見積もりするための関数、通常 h という文字で表す。

$$h(n) = \text{節点} n \text{の状態からゴール状態までの最短の道の見積もりコスト}$$

最良優先探索 or 欲張り探索(Best first search or Greedy Search)

- アルゴリズム

step 1 出発点をリストLに入れる.

step 2 if L=空 then 探索は失敗，終了.

step 3 Lの先頭の節点nを取り除く.

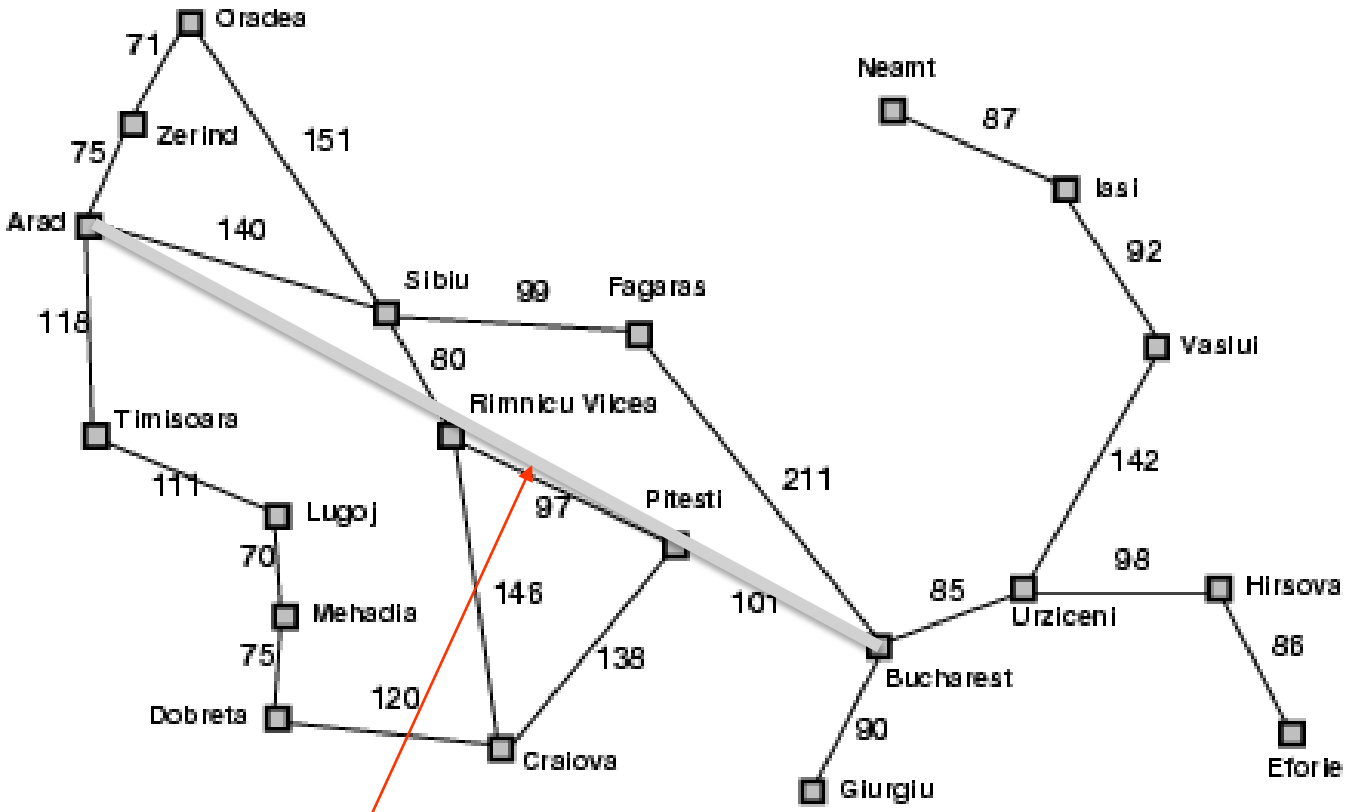
step 4 if nが目標節点である then 探索は成功，終了.

step 5 if nが展開できる（子節点を持つ）

then 展開し、すべての子節点からnへのポインタを付ける．すべての子節点をリストLに入れ、リストの要素節点m_iを$h(m_i)$の昇順でソートする． step 2へ

else step 2へ

例1：ロマニアの都市間の直線距離（コスト）(Romania with step costs in km)



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

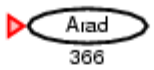
Arad-Bucharest =366km

ヒューリスティック関数の値

欲張り探索(Greedy best-first search)

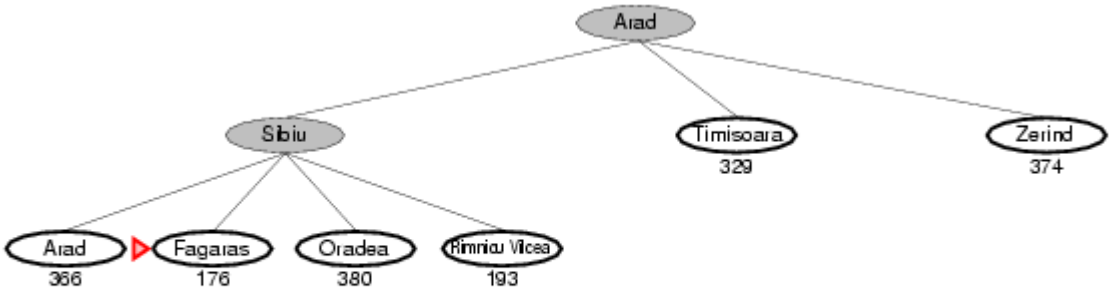
- *評価関数　f(n) = h(n)* (ヒューリスティック関数)
- *f(n) = nからゴールまでのコスト見積もり* (estimate of cost from *n* to *goal)*
- $h_{SLD}(n)$ = *nから*Bucharestまでの直線距離 (straight-line distance(SLD) from *n* to Bucharest)
- *展開すべき次の節点にf(n) を用いる欲張り探索を行う。*

欲張り探索の例(Greedy best-first search example)

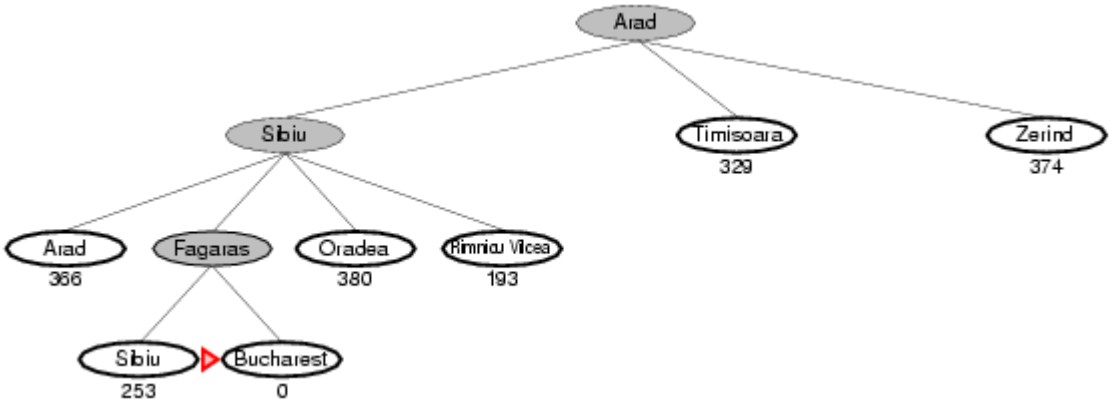


欲張り探索の例(Greedy best-first search example)

欲張り探索の例(Greedy best-first search example)



欲張り探索の例(Greedy best-first search example)



欲張り探索の性質(Greedy best-first search Properties)

- 完全(Complete)? No –解は永久に求まらない。Iasi → Neamt → Iasi → Neamt → 行ったり、来たりになってしまう。
- 時間計算量(Time)? $O(b^m)$, 良いヒューリスティック関数があれば、実質的に減らすことができる。
- メモリ計算量(Space)? $O(b^m)$ – 全てのノードをメモリに保持される
- 最適(Optimal)? No

欲張り探索アルゴリズム

最良優先探索のアルゴリズム

① **OPEN**＝(出発節点 S)
// 初期状態を表す出発節点を OPEN リストに入れる
CLOSED＝{ };
// **CLOSED** リストは調べ終わった節点を入れる集合

② もし，**OPEN** が空ならば，終了［失敗］
// 目標節点がない

③ p:＝先頭要素（**OPEN**）
// OPEN の先頭要素を p とする［選択］

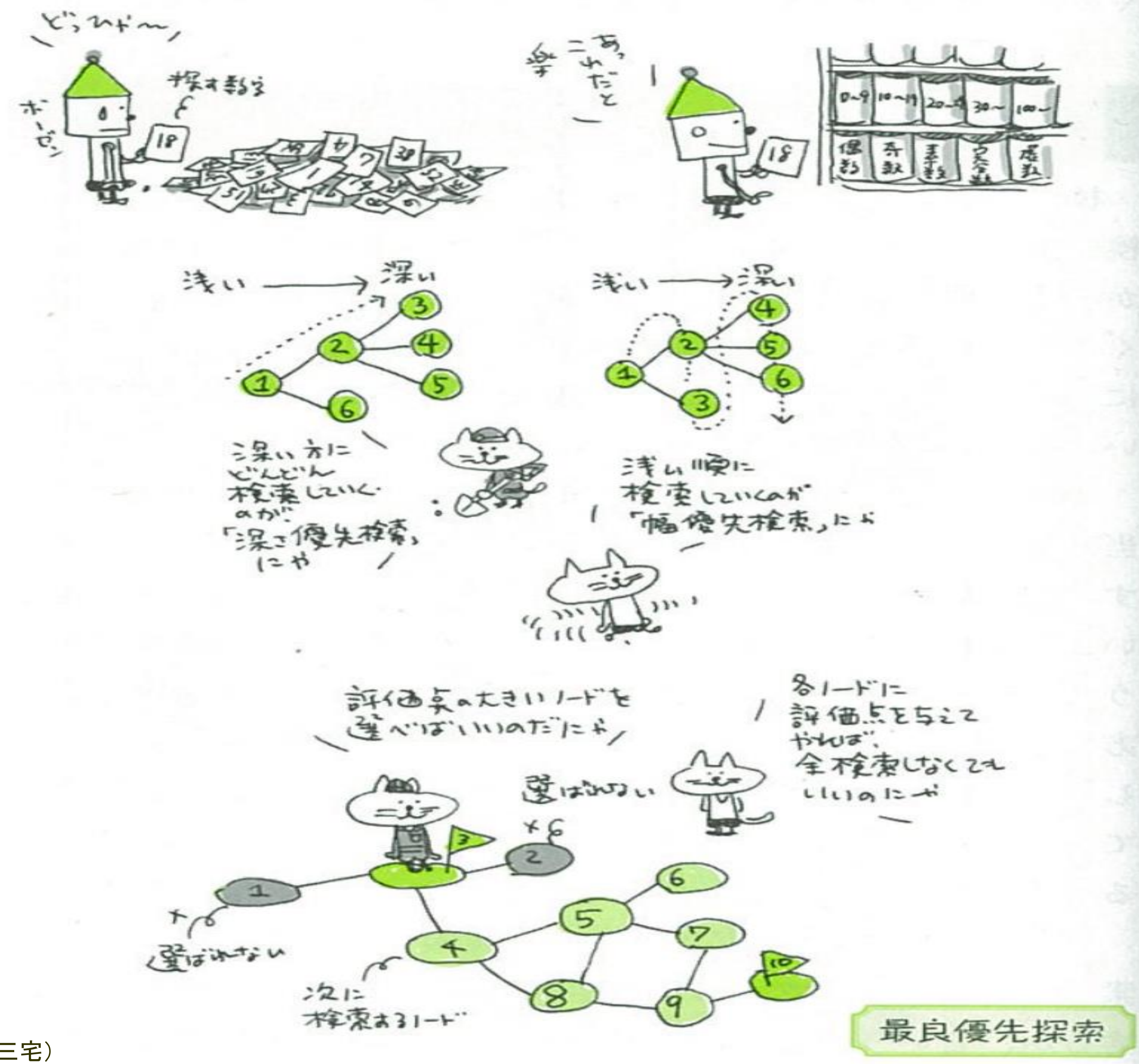
④ もし，p が目標節点 G ならば，終了［成功］
// p が目標節点である

⑤ p を展開；p を OPEN から削除し，**CLOSED** に格納
// p は調査済み

⑥ 子節点 q が **OPEN** にも **CLOSED** にも含まれない場合
q を OPEN に入れる　q から p へのポインタを付加

⑦ **OPEN** の節点の評価値 $h^*(p)$ によるソート（昇順）

⑧ ②へ戻る
// ②～⑦はループをなす



絵でわかる人工知能（三宅）

目次

- 情報を持つ探索(ヒューリスティック探索)
  - 最良優先探索
    - 欲張り探索
    - A*探索

A* 探索(A* search)

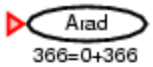
- 発想：経路全体のコストを最小化する
  評価関数 $f(n) = g(n) + h(n)$

- $g(n)$ = nまでの経路のコスト (出発節点から節点nまでの経路のコスト)
- $h(n)$ = nからゴールまでの最短経路の見積もりコスト(estimated cost from n to goal)
- $f(n)$ = n経由の最短解の見積もりコスト(estimated total cost of path through n to goal)

A＊探索

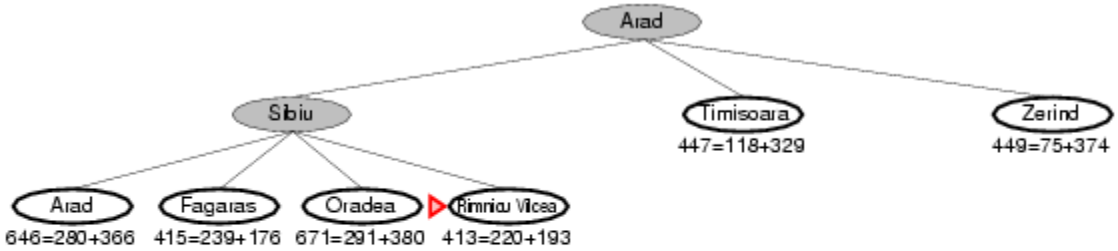
- A*アルゴリズム
  - これまでの経路コストと現在地からゴールまでのコストの予測値の和を評価値とする。
  - A*アルゴリズムは、探索コストが最小の解を必ず求めることができる。
- 欲張り探索との相違
  - これまでの経路コストを考慮している点。

A* 探索例1(A* search example)

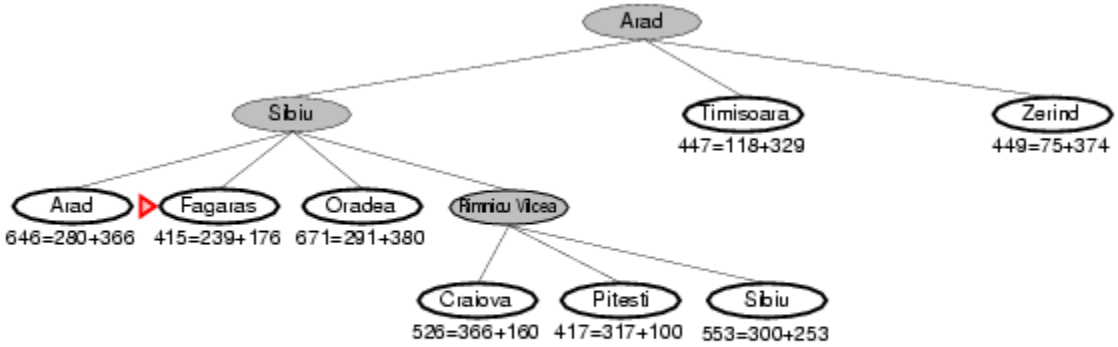


A* 探索例1(A* search example)

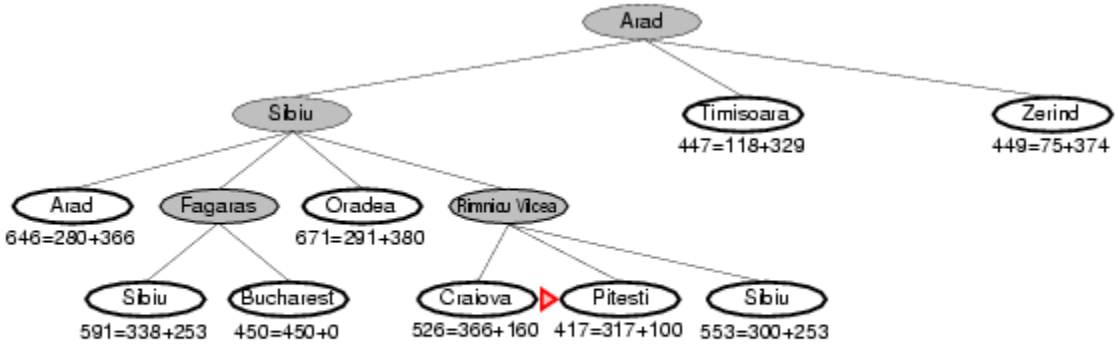
A* 探索例1(A* search example)



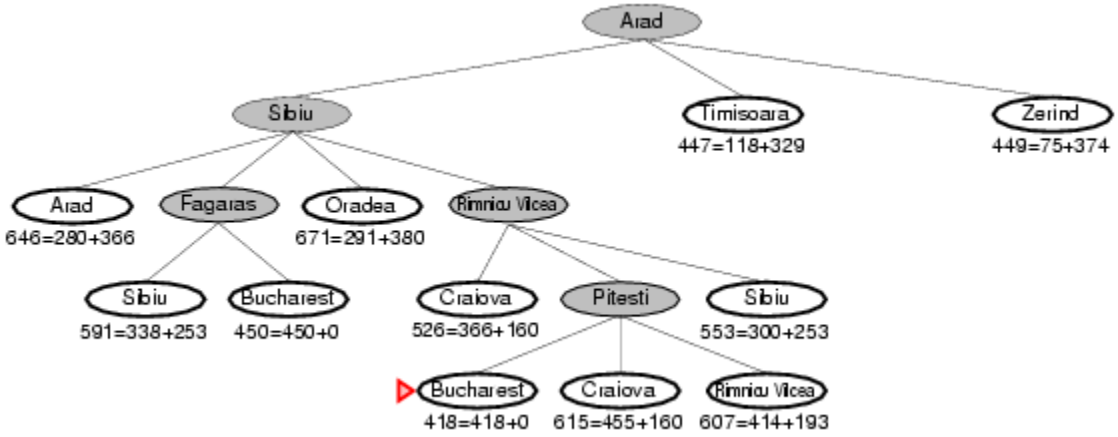
A* 探索例1(A* search example)



A* 探索例1(A* search example)



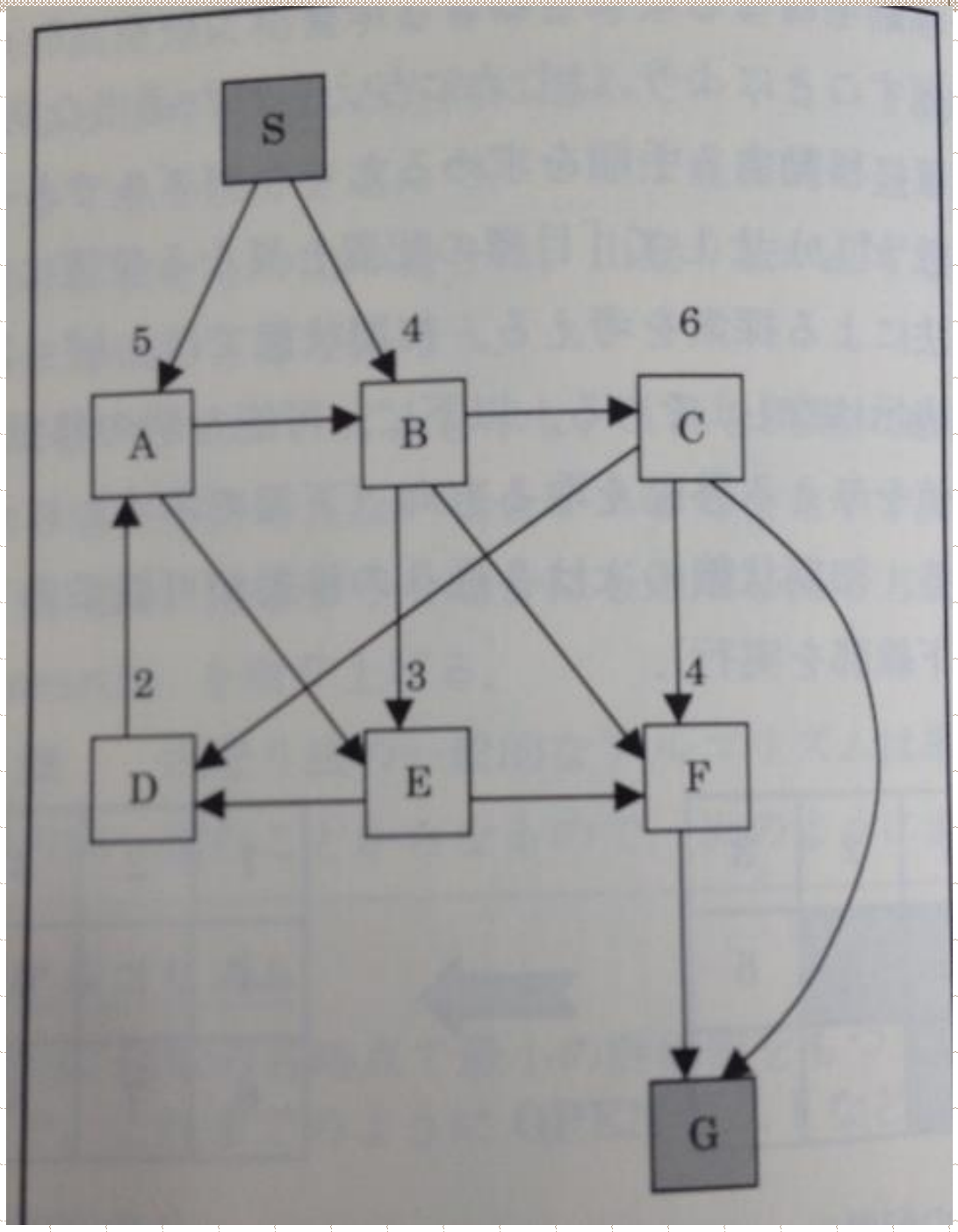
A* 探索例1(A* search example)



許容的ヒューリスティック (Admissible heuristics)

- $h(n)$ が許容的であれば、$f(n)$ は n を経由する最良解の実際の解のコストを決して越えない見積もりになっている。
- 許容的ヒューリスティックはゴールまでのコストを絶対過大に見積もらない。
- $h_{SLD}(n)$ は許容的ヒューリスティックである。
- 定理: $h(n)$ は許容的であれば, A*探索 TREE-SEARCH は最適である。

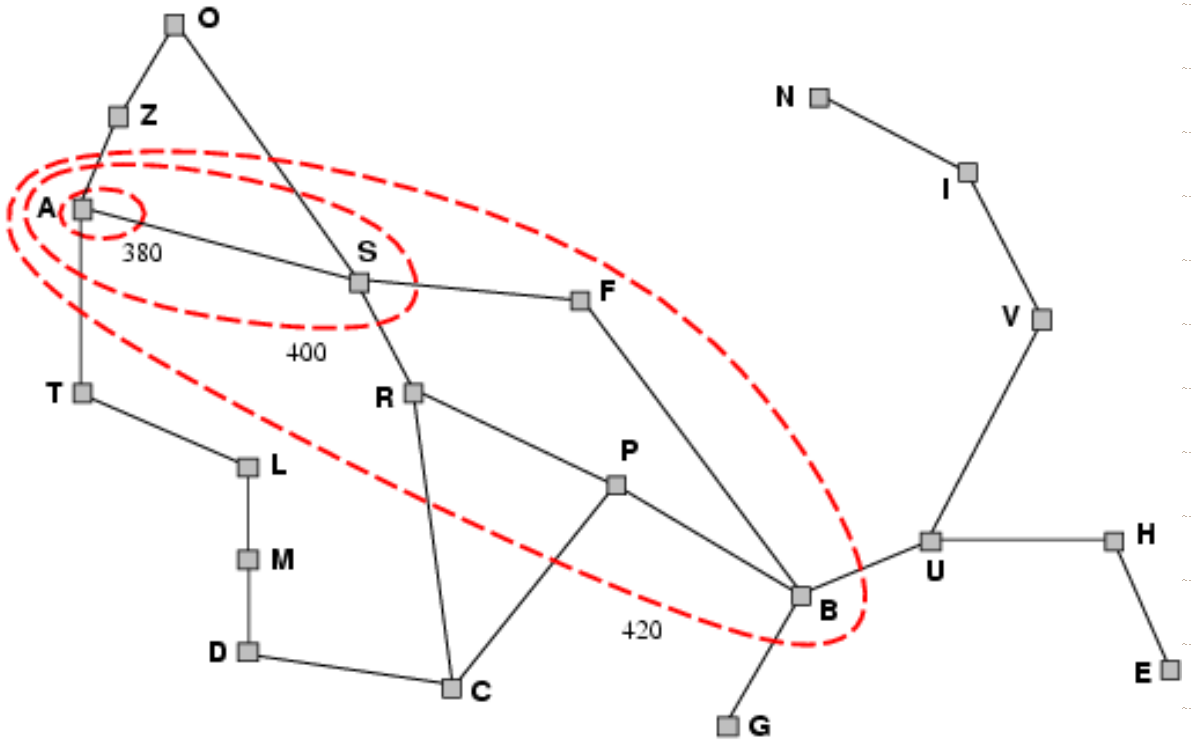
例題



A*探索による等高線

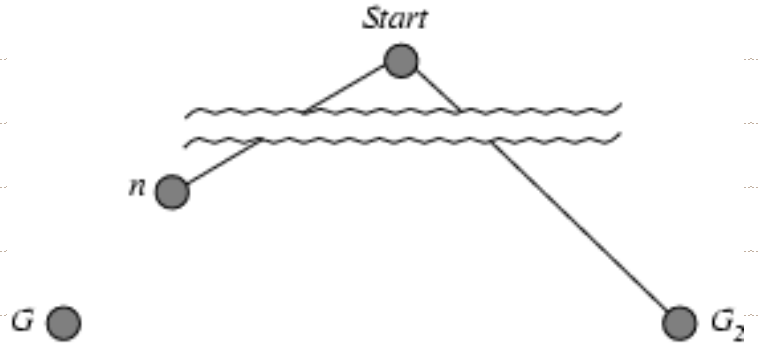
- 状態空間における等高線を概念的に描くことができる。
- Aradを出発節点としたときのルーマニア地図における*f=380,f=400,f=420*の等高線、等高線の中の　*f* コストは等高線の値よりも小さい。

A*探索による等高線



A*の最適性の証明

- Gを最適ゴール状態として、その経路コストをf*とする。G_2 を準最適ゴール状態、すなわち$g(G_2) > f^*(G)$とする。A*探索が持ち行列から G_2 を選んだ状況を仮定しよう。G_2 は目標状態なので、準最適解で探索が終了することになる。これが不可能であることを示す。



- Gへの最適経路上の葉節点 n を考える。
- $f^* > f(n)$ が成り立つ、
- $f(n) > f(G_2)$ が成り立つ、
- $f^* > f(G_2)$ が成り立つ、
- $f^* > g(G_2)$ が証明できたことになる。A*は最適アルゴリズムである。

A*探索の性質(Properties of A*)

- 完全(Complete)? Yes
-
- 時間計算量(Time)? 急激的(Exponential)
-
- メモリ計算量(Space)? すべてのノードはメモリ上で保持する(Keeps all nodes in memory)
-
- 最適(Optimal)? Yes

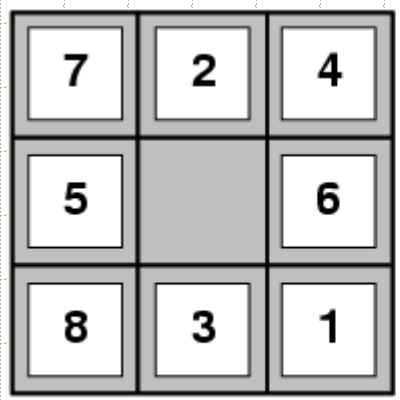
ヒューリスティック探索の問題点

- 評価関数の与える評価値が実際の値に十分近ければ、うまくいく。
- そうでなければ、局所解に陥ったり、遠回りをしなければならなかったりする。
- 最良優先探索でも、評価関数によっては探索コストが最小の解を必ず見つけ出せるという保証はない。
- 旅行計画では、直線距離を評価関数に使うのが、妥当。
- それでうまくいかない例ある

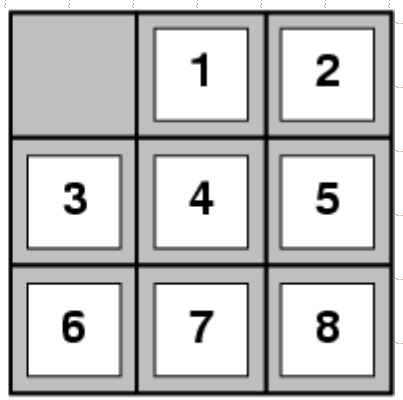
許容的ヒューリスティック (Admissible heuristics)

例3, 8-パズル:

- $h_1(n)$ = ゴールの位置にないタイルの数(number of misplaced tiles)
- $h_2(n)$ = ゴール状態からのタイルの距離の和(マンハッタン距離-total Manhattan distance)



Start State



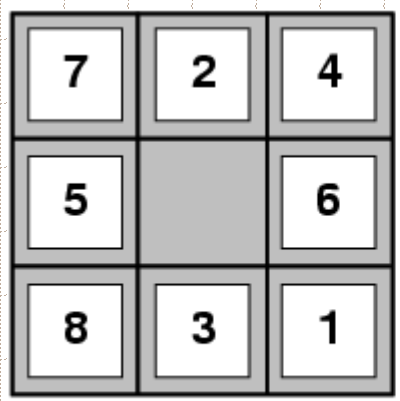
Goal State

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

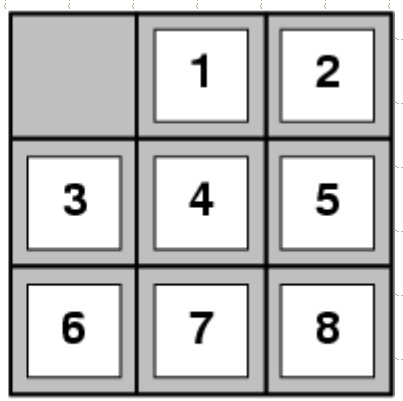
許容的ヒューリスティック (Admissible heuristics)

例3, 8-パズル:

- $h_1(n)$ = ゴールの位置にないタイルの数(number of misplaced tiles)
- $h_2(n)$ = ゴール状態からのタイルの距離の和(マンハッタン距離-total Manhattan distance)



Start State



Goal State

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

優位(Dominance)

- If $h_2(n) \geq h_1(n)$ すべての n に対して
- then h_2 は h_1 より優位にある。
- h_2 のA*探索はh_1のA*探索より少ない節点しか展開しない。
- 探索のコストの比較(Typical search costs (average number of nodes expanded)):
- *d=12* IDS = 3,644,035 nodes
  $A^*(h_1)$ = 227 nodes
  $A^*(h_2)$ = 73 nodes
- *d=24* IDS = 数えられない(too many nodes)
  $A^*(h_1)$ = 39,135 nodes
  $A^*(h_2)$ = 1,641 nodes

反復深化探索とA*探索のコスト、ヒューリスティック等の比較

	Search Cost			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	–	539	113	–	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

ヒューリスティック関数の作り方

- *h2* のような関数を作るのだろうか。コンピュータにこのようなヒューリスティックを自動的に作らせることは可能のだろうか。
- *h1, h2* は8パズルの残りの経路の長さを見積もったものであるが、見方を変えれば、このパズルをもっと単純化したパズルの経路の長さを完全に正解に与えていると考える。
- ルールを変えたとすれば、*h2* は最短解のステップ数を世界に与えている。これを弱条件問題と呼ぶ。
- 良いヒューリスティックを見つけるための統計情報を用いることである。
-

ヒューリスティック探索のまとめ

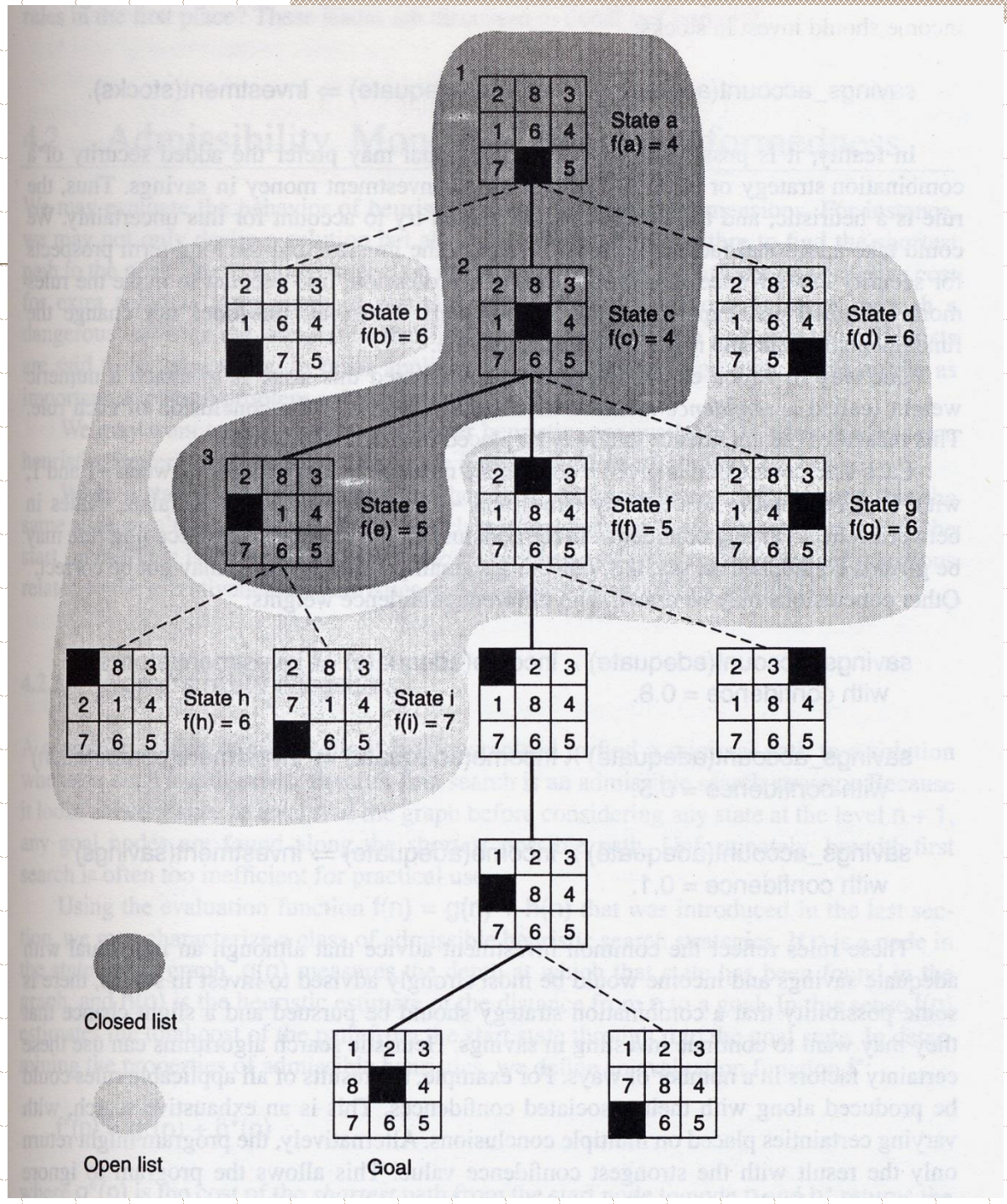
- ヒューリスティック探索として、最良優先探索、欲張り探索、A*アルゴリズムについて学んだ。
- 最良優先探索は、すでに通ってきた分岐点を含めて、最良の道を選ぶ。
  - コスト最小化の節点から展開する一般的探索である。
  - ゴールへ到達するための見積もりコストを最小化するのであれば、欲張り探索を用いることができる。
  - 最良優先探索は、解に到達するが、最適解を得られない可能性がある。

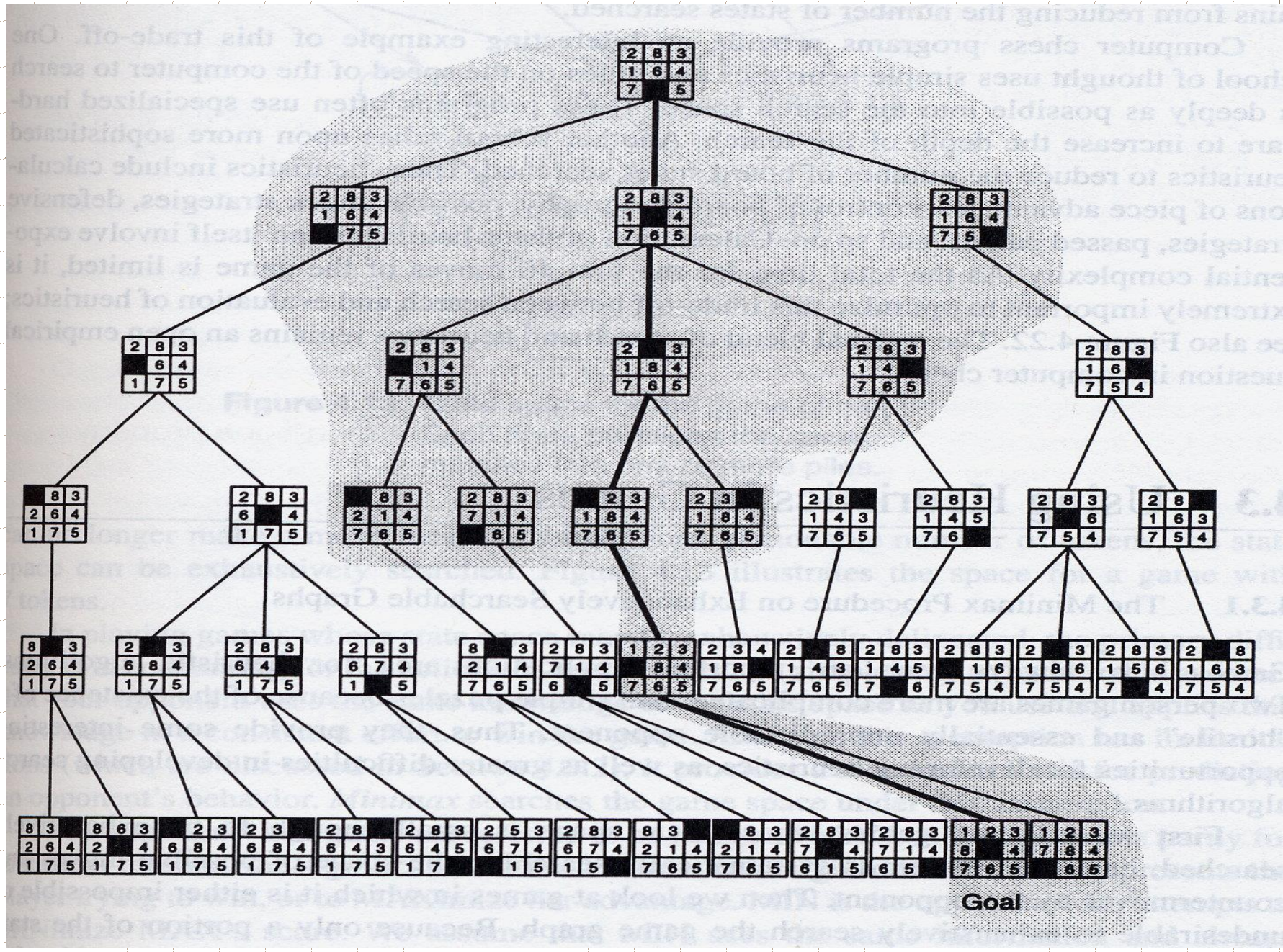
ヒューリスティック探索のまとめ

- A*アルゴリズムは、これまでの経路コストと現在地からゴールまでのコストの予測値の和を評価値とする。
  - A*アルゴリズムは、探索コストが最小の解を必ず求めることができる。
  - A*は完全で、最適で、すべての最適探索アルゴリズムの中で効率が最適である。たが空間(メモリ)計算量は爆発してしまう。
- 許容的ヒューリスティック:

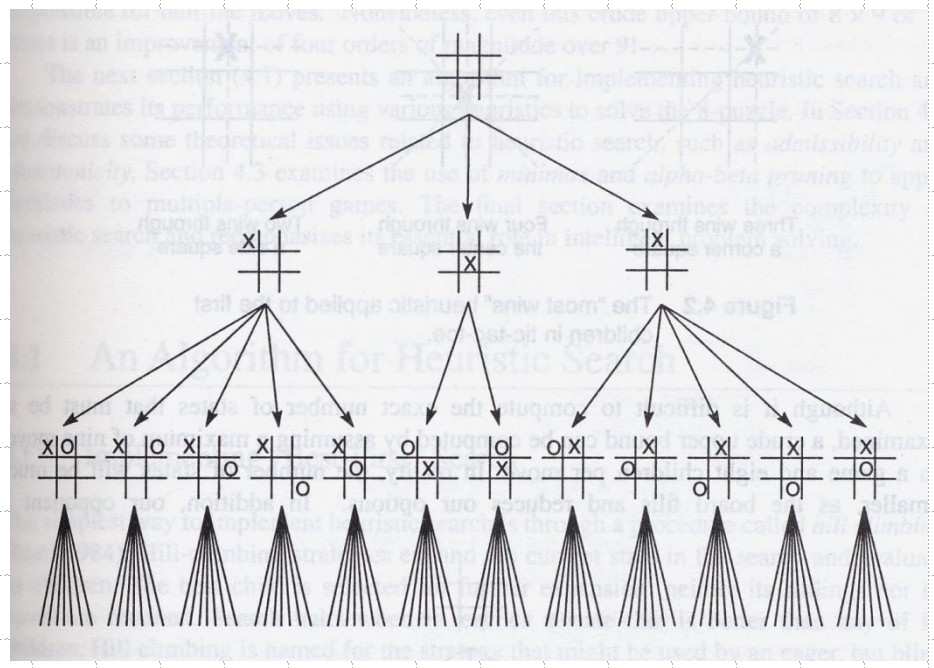
  $h(n)$ が許容的であれば、$f(n)$ は n を経由する最良解の実際の解のコストを決して越えない見積もりになっている。
- ヒューリスティックアルゴリズムの時間計算量は評価関数の質に依存する。良いヒューリスティックは、問題の定義を分析したり、たくさんの例題で実験をしてそれを一般化したり、という方法によって作ることができる。

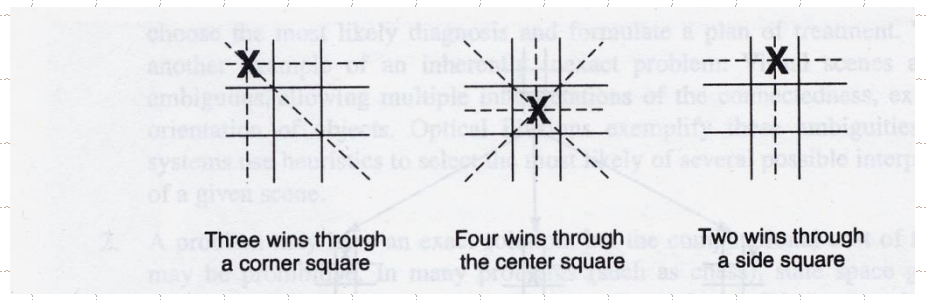
- h1：正しい位置に置かれていない駒の個数
- h2:各駒の現在位置と正しい位置との間の距離の総数


1
2 8 3
1 6 4
7 5
State a
f(a) = 4
2
2 8 3
1 4
7 6 5
State c
f(c) = 4
2 8 3
1 6
7 5 4
State b
f(b) = 6
2 8 3
1 6 4
7 5
State d
f(d) = 6
3
2 8 3
1 4
7 6 5
State e
f(e) = 5
2 3
1 8 4
7 6 5
State f
f(f) = 5
2 8 3
1 4
7 6 5
State g
f(g) = 6
8 3
2 1 4
7 6 5
State h
f(h) = 6
2 8 3
7 1 4
6 5
State i
f(i) = 7
2 3
1 8 4
7 6 5
2 3
1 8 4
7 6 5
1 2 3
8 4
7 6 5
Closed list
Open list
1 2 3
8 4
7 6 5
Goal
1 2 3
7 8 4
6 5


Goal

三段目ならべのヒューリスティック





三段目ならべのヒューリスティック