ギャップ集合を用いた 15パズルの最適解探索の高速化

Fast Search for Optimal Solutions of the Fifteen Puzzle Using Gap Sets

山本 修身
Osami Yamamoto

名城大学 理工学部 情報工学科
Department of Information Engineering, Faculty of Science and Technology, Meijo University
osami@meijo-u.ac.jp

佐藤根 寛
Hiroshi Satone

名城大学 理工学研究科 情報工学専攻
Department of Information Engineering, Graduate School of Science and Technology, Meijo University
103430851@ccalumni.meijo-u.ac.jp

keywords: the fifteen puzzle, heuristic search, IDA* algorithm, pattern database, the gap- $2n$ set

Summary

The fifteen puzzle is a sliding puzzle which has fifteen pieces on which numbers from 1 to 15 are printed. Using the IDA* algorithm with an admissible evaluation function, we can obtain an optimal solution of the puzzle. The performance of the algorithm depends on the evaluation function. The most simple evaluation function is the Manhattan evaluation function, whose value is the sum of the Manhattan distances from the positions of the corresponding pieces in the goal configuration. In this paper, we propose an evaluation function whose values are greater than or equal to that of the Manhattan evaluation function. Our evaluation function refers an approximated database of the gap- $2n$ set. The database is computed beforehand like pattern databases, but it is completely different from pattern databases. The belongingness of a configuration of pieces to the set has to be checked by the database. Using an evaluation function based on the gap-8 set, we were able to reduce the number of search nodes to about 2.5×10^{-4} times in average with the IDA* algorithm compared with the Manhattan evaluation function. We also show that combining an evaluation function by gap-8 set and an evaluation function by additive pattern databases of disjoint seven and eight pieces, we were able to reduce the number of search nodes by about 53 percent compared with the evaluation function only by the additive pattern databases.

1. はじめに

15パズルはスライディングパズルと呼ばれるパズルの一種である。4×4に仕切られた枠に1から15までの番号が書かれた15枚のコマが置かれており、一カ所だけコマが置かれていない場所（空所）がある。空所に隣り合ったコマは、空所にずらすことにより、位置を変えることができる。この操作を繰り返して、あらかじめ決められたゴール状態の配置までコマの配置を変化させることが、このパズルの目的である。本論文では図1に示す状態をゴール状態の配置とする。

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

図1 本論文で用いる15パズルのゴール状態のコマの配置。

このようなパズルを最少の手数で解くための手順を見つける問題は、発見的探索手法を説明する場合の代表的な例題となっている。また、このような問題を解くための手法としては、A*アルゴリズム [Hart 68] や IDA*アルゴリズム [Korf 85] が良く知られている。これらのアルゴリズムの性能は、そこで用いられる評価関数の性能に大きく依存する。これらのアルゴリズムで用いられる評価関数は、パズルのコマの配置を入力とし、その配置からゴール状態に至る最短手順数の下界値を出力とするものである。このような評価関数を用いると、上記アルゴリズムは常に最短手順を出力することが知られているが、効率については、評価関数に大きく依存する。評価関数値が最短手順数の下界値になっているという条件の下で、より大きな値をとれば、より短い時間で最短手順が得られる。したがって、最短手順数を超えないなるべく大きな値とする評価関数を用いれば、より効率的に探索することができる。

本論文では、あらかじめ計算されたデータベースに基づいて、良く知られた15パズルのマンハッタン評価関数に

ある値を足し加えた評価関数を用いることにより, IDA* アルゴリズムによって 15 パズルの最適解の効率的な探索が可能になることを示す. ここでは, マンハッタン評価関数の値がゴール状態までの最短手順の数に等しいコマの配置を列挙し, これをテーブルに記録する. このような配置の集合を「ギャップゼロ集合」と呼ぶ. このテーブルを参照することにより, 与えられた配置についてマンハッタン距離関数値に 1 以上の値を足し加えることができるかどうかを調べ, 可能であれば値を足し加える. 本論文では, さらにマンハッタン距離関数値と最短手順数のギャップがある値以下の配置の集合に対応するテーブルを計算し, これを用いて, 許容的な評価関数を実現する. 我々の方法は, 結果的に Hansson ら [Hansson 92] による Linear Conflict という手法の拡張になっている.

あらかじめ計算されたコマの配置のパターンに関する巨大なデータベース (パターンデータベース) を用いることにより, 非常に高性能な許容的な評価関数を実現できることが良く知られている [Korf 02, Felner 04a]. ここで述べる評価関数はこの方法とは異なるものである. パターンデータベースがパズルの部分パターンからゴールまでの最短手順数を得る関数をテーブル化しているのに対して, 我々の方法は, 特定のパターンの集合をテーブルとして保存し用いるものである.

さらに, パターンデータベースと本論文で示す方法を組み合わせることにより, パターンデータベース単独による評価関数よりもさらに高速に最適解を探索することのできる評価関数を実現できることを示す.

2. 15 パズルの基本的性質

本節では, 15 パズルに関する基本的な性質について述べる. 本節で述べる内容は基本的に 15 パズル以外の $n^2 - 1$ 型のスライディングパズルについても同様に成立するものであるが, ここでは 15 パズルに関する性質として述べる.

まず, パズルのコマの配置に関するパリティについて述べる. 本論文においては, ゴール状態と同じパリティをもつ配置のみを考えるので, この性質が重要な意味を持つ事はないが, すべての配置からゴール状態へ遷移させることが不可能であるという事実は 15 パズルにおいて重要である. パリティについてはコマの順列を考えることで容易に導くことができる [Johnson 79]. 以下に結果を示す.

命題 1 15 パズルのコマの配置のうち, スライドによる遷移によって最終状態に到達させることができるものは, すべての配置のうち半数である. すなわち, $16! = 20,922,789,888,000$ 通りの配置のうち, ちょうど半数の配置についてはどのようにスライドさせても最終状態に到達させることができず, 残りの半数については最終状態に到達させることができる.

つぎに, 以下に示すようにマンハッタン評価関数 $M(c)$ を定義する.

【定義 1】 15 パズルにおいて, それぞれのコマの配置を記述したものを c と表すとき, コマ i ($i = 1, \dots, 15$) の位置 (左上から数えた 2 次元整数座標) を $P(c, i)$ と書くことにする. このとき, マンハッタン評価関数 $M(c)$ は与えられた配置 c について以下のように定義される:

$$M(c) = \sum_{i=1}^{15} \|P(c, i) - P(g, i)\|_1. \quad (1)$$

ただし, g はゴール状態のコマの配置を表し, $\|\cdot\|_1$ は整数 2 次元ベクトルのそれぞれの成分の絶対値の和を表す. すなわち, $\|(n, m)\|_1 = |n| + |m|$ と定義される.

マンハッタン評価関数について以下の命題が成立する.

命題 2 15 パズルの任意のコマの配置 c について, ゴール状態 g まで遷移させるための手数は $M(c)$ 回以上である.

《証明》 15 パズルのそれぞれのコマに注目する. 15 パズルのルールとして, 一度に 1 つのコマをスライドさせることしかできない. 従って, あるコマ i の位置 (その配置を c とおく) からゴール状態 g まで遷移させるためには, 最短手順数で移動させたとしても,

$$k_i = \|P(c, i) - P(g, i)\|_1 \quad (2)$$

回の移動が必要となる. これは, それぞれのコマが一度に横または縦に 1 つだけ移動することによる. さらに, 一度にいずれかのコマを移動させることしかできないことから, 最短手順数であっても $\sum_{i=1}^{15} k_i$ 回のコマの遷移が必ず必要である. 以上より, この命題は成立する (証明終り)

命題 2 により, 最低限必要な手順数についてはわかるが, さらに 15 パズルには以下に示す様な顕著な性質が存在する.

命題 3 任意のコマの配置 c からゴール状態 g までの最短手順数を $h(c)$ と書けば, $h(c) - M(c)$ は常に 0 以上の偶数である.

《証明》 この命題は, コマをスライドさせてゴールに至るまでの過程において, $M(c)$ がどのように変化して行くかを考えれば, ほぼ明らかである. 初期状態の配置を c_0 として, ゴールに至る最短手順数の遷移が以下になっていると仮定する: $c_0, c_1, c_2, \dots, c_N = g$. ただし, N を最短手順数とする. このとき, マンハッタン距離関数値の列 $M(c_0), M(c_1), M(c_2), \dots, M(c_N)$ を考える. ここで, $c_N = g$ なので, 明らかに $M(c_N) = 0$ である. また, それぞれの遷移における $M(c)$ の変化 $M(c_i) - M(c_{i+1})$ は, M の定義より, ± 1 である. ここで, この列の $+1$ の個数を k_{+1} , -1 の個数を k_{-1} とおけば, $M(c_0) + k_{+1} - k_{-1} = 0$ である. 一方, 定義より, 明らかに $N = k_{+1} + k_{-1}$ であり, さらに $N = h(c_0)$ と定義

しているので, $M(c_0) = k_{-1} - k_{+1} = 2k_{-1} - h(c_0)$ である. これより, $M(c_0) + h(c_0) = 2k_{-1}$ であり,

$$h(c_0) - M(c_0) = 2(h(c_0) - k_{-1}) \quad (3)$$

となり, 任意のコマの配置について, 最短手順数とマンハッタン距離関数値の差は必ず偶数となる. 命題 2 より, この値が 0 以上であることは示されているので, 本命題を得る (証明終り)

この命題より, マンハッタン距離関数値を計算することによって, 最短手順数の偶奇についても知る事ができる. また, 最短手順数の下界を得るためには, 0 を含む偶数をマンハッタン距離関数に足し加えればよい事もわかる. この事実は本論文で重要な役割を演じる.

3. 探索アルゴリズムについて

実際に与えられたコマの配置からゴール状態の配置まで遷移させるためのコマの動きを探索するには, 発見的探索手法を用いる. 本論文では, いくつかの配置について, IDA* アルゴリズム [Korf 85] を用いて最短手順数の配置の遷移を探索し, その探索ノード数や実行時間を計測する. 前節の命題 3 より, $M(c) + \text{奇数}$ 番目の配置はゴール配置とは成り得ないので, 初期配置からゴール配置までの探索の深さの限界は, $M(c)$ からスタートして 2 ずつ増やしていけば良いことがわかる.

4. ギャップ- $2n$ 配置集合とそこから導出される評価関数

本節では, 本論文で中心的役割を演じるギャップ集合と呼ばれる配置の集合を定義し, それを用いて許容的な評価関数を構成する方法について述べる. まず, ギャップ集合を定義するためにマンハッタン遷移を定義する.

【定義 2】あるコマの配置 c_1 は c_2 へ 1 回のコマのスライドによって遷移させることができるとする. $M(c_2) = M(c_1) - 1$ となっているとき, この遷移のことをマンハッタン遷移と呼ぶ. ある配置 c がマンハッタン遷移のみを用いてゴール状態 g まで遷移させることができるとき, c のことをギャップゼロ配置と呼ぶ. また, すべてのギャップゼロ配置を集めた集合のことをギャップゼロ (配置) 集合と呼び, G_0 と書く. また, ある配置 c から 1 回のマンハッタン遷移によって遷移させることのできる配置の集合を $MT(c)$ と書き, さらに c から 1 回の遷移で得られる配置のうち非マンハッタン遷移によるものの集合を $MT^+(c)$ と書く.

例えば, 図 2 において, 左上の配置から右上の配置へ 1 のコマを移動させる遷移 t_1 は, 1 のコマのこの配置での位置とゴール状態での位置のマンハッタン距離が 2 から 1 に減るので, マンハッタン遷移となる. 同様にして, 遷移 t_3 もマンハッタン遷移となる. これに対して, 遷移

図 2 マンハッタン遷移 (t_1 と t_3) と非マンハッタン遷移 (t_2).

図 3 ギャップゼロ配置 c からゴール状態へ至るマンハッタン遷移の列. この場合 2 通りの遷移が存在する.

t_2 はコマ 6 のマンハッタン距離を増やす方向に移動させてしまうのでマンハッタン遷移ではない.

定義から明らかに, 任意のギャップゼロ配置 c からゴールまでの最短手順は, $M(c)$ に等しい. また, 逆にゴールまでの最短手順が $M(c)$ に等しいような配置はギャップゼロ配置である. 以下にギャップゼロ配置に関するほぼ自明な性質を示す.

命題 4 任意のマンハッタン遷移 c について配置の列 $c = c_0, c_1, \dots, c_N = g$ で遷移 $c_i \rightarrow c_{i+1}$ がすべての $i = 0, \dots, N-1$ についてマンハッタン遷移になっているものが存在し, その列の中に同じ配置が 2 回出現することはない.

《証明》 定義より, ほぼ自明である. マンハッタン遷移になっていることから, $M(c_0) = N, M(c_1) = N-1, \dots, M(c_{N-1}) = 1, M(c_N) = 0$ となることから, ゴール配置に至る列の中のそれぞれの配置に同じものは存在しない (証明終り)

この性質から G_0 は深さ優先探索を用いて列挙することができる. ただし, それぞれのギャップゼロ配置について, このような列は唯一とは限らないことから列挙さ

れた配置の中に重複が生じる可能性がある．実際，図 3 に示すように左上の 3 つのコマのみがゴール状態と異なるような配置は 2 通りの移動によってゴール状態までマンハッタン遷移により変化させることができる．

ゴール状態から逆順に，非マンハッタン遷移の集合 $MT^+(c)$ を計算しながら，深さ優先探索によってギャップゼロ集合の要素を列挙することができる．命題 4 に示したように任意のノードから根までの経路上に同じ配置が出現することはないので，この木の深さは有限であり，深さ優先探索によって求めることができる．

前述のように，このアルゴリズムの出力は，ある 1 つのパス上に同じ配置が並ぶことはないが，別の経路で同一のパターンに辿り着く可能性があり，そこから下のレベルで同じ配置が複数回出力されることになる．このアルゴリズムから出力される配置は適当な方法によって，データベース等に蓄える必要がある．その際に重複を除く処理を行う必要がある．

このようにして得たギャップゼロ集合 G_0 を用いることによって，マンハッタン距離関数を用いて新たな評価関数を以下のようにして作ることができる：

$$h_0(c) = \begin{cases} M(c), & (c \in G_0) \\ M(c) + 2, & (c \notin G_0) \end{cases} \quad (4)$$

ギャップゼロ集合 G_0 に与えられた配置 c が含まれない場合に +2 することができるのは，命題 3 による．詳しい計算については第 5 節および第 7 節で述べるが，15 パズルに関しては， G_0 は 88,728,780 個の要素を含む．この集合を用いた上記評価関数を用いることにより，300 個のランダムに生成した配置について IDA* アルゴリズムを用いてゴール状態までの最短手順を計算するとき，マンハッタン距離関数を用いたときに比べて探索数を 17% 程度にすることができる．

ここで， G_0 を拡張して， n を任意の正の整数としたとき，ゴール配置 g までの最短手順数とマンハッタン距離関数 $M(c)$ の差が高々 $2n$ の配置をすべて集めた集合をギャップ $2n$ (配置) 集合 G_{2n} (あるいは単にギャップ集合) と定義する． G_{2n} について以下の命題が成り立つ．

命題 5 ある配置 c について $c \notin G_{2(n-1)}$ かつ $c \in G_{2n}$ であるとき， c からゴール配置 g へ至る任意の最短手順には，ちょうど n 個の非マンハッタン遷移が含まれている．逆に，ゴールまでの最短手順がちょうど n 個の非マンハッタン遷移を含む配置 c について， $c \notin G_{2(n-1)}$ かつ $c \in G_{2n}$ が成り立つ．

《証明》配置 c について $c \notin G_{2(n-1)}$ かつ $c \in G_{2n}$ であると仮定する．このとき，定義より $h(c) = M(c) + 2n = N$ である． $c = c_0$ からコマを動かすことにより，配置の列 $c_0, c_1, c_2, \dots, c_N = g$ が得られる．それぞれの配置におけるマンハッタン距離の列 $M(c_0) = M(c), M(c_1), \dots, M(c_N) = 0$ は，隣り合う値との差が +1 または -1 でなければならない．+1 の個数を a ，-1 の個数を b と

おけば， $a + b = M(c) + 2n$ ， $a - b = -M(c)$ より， $a = n$ ， $b = M(c) + n$ を得る．これより， a は非マンハッタン遷移の個数であるであり，これは n となる．

一方，最短手順にちょうど n 個の非マンハッタン遷移を含むと仮定すれば，最短手順の数は $M(c) + 2n$ であるので， $c \notin G_{2(n-1)}$ かつ $c \in G_{2n}$ である（証明終り）

以上より，一般の G_{2n} の要素の列挙についてもギャップゼロ集合と同様に深さ優先探索を用いて計算することができる．この場合，ゴール状態からスタートして，高々 n 回非マンハッタン遷移が出現するパスをすべて探索する．

ただし，2 以上のギャップがある場合には，実際の計算においては， $c_1 \rightarrow c_2 \rightarrow c_1$ というような遷移を含む場合があり，これは，明らかに最短経路ではないので，この部分を除去した配置が出現しているはずである．したがって，このようなものを数えないような最適化により計算量を押さえることができる．

ギャップ $2n$ 配置集合の列 $G_0, G_2, \dots, G_{2n}$ が得られているとき，これを用いてつぎのような評価関数を得ることができる：

$$h_n(c) = \begin{cases} M(c), & (c \in G_0) \\ M(c) + 2, & (c \notin G_0 \text{ かつ } c \in G_2) \\ M(c) + 4, & (c \notin G_2 \text{ かつ } c \in G_4) \\ \vdots & \vdots \\ M(c) + 2(n+1), & (c \notin G_{2n}) \end{cases} \quad (5)$$

この関数が許容的な評価関数であることは，定義からほぼ明らかである．ここで問題となるのは，このような集合を列挙することはできても，それぞれのパズルの配置がそれぞれのギャップ集合に含まれるか否かを判定することが困難であるということである．これについては次節で考察する．

5. ギャップ $2n$ 配置集合の近似について

G_0 の要素数は前述のように 9 千万弱である． G_0 よりもギャップの大きな一般の G_{2n} は，非常に大きな要素数の集合となることが考えられる．ギャップ集合に基づく評価関数を計算するためには，それぞれのパターンがギャップ集合に含まれるか否かの判定を効率的に行わなければならない．前節で述べた方法によって列挙された配置をどのように蓄えるかということが問題となってくる．

本論文では，いくつかのコマが配置のどの位置に存在するかということから，それぞれの配置のパターンをより情報量の少ないパターンに射影することによって， G_{2n} を部分集合として含む大きな集合の情報を蓄え，それにより前節で述べた評価関数よりも値の小さな評価関数を構成する．同様の手法はパターンデータベースにおける圧縮パターンデータベースに見られる [Felner 04b]．

評価関数の構成を考える前に，それぞれの G_{2n} の大きさを見積もる．ギャップ集合の大きさは全配置の数に比

表 1 ギャップ $2n$ 集合 G_{2n} の大きさの見積り

$2n$	大きさの見積り	実際の大きさ	存在の割合
0	8.83×10^7	88,728,780	4.21×10^{-6}
2	1.36×10^9	1,331,820,797	6.50×10^{-5}
4	1.15×10^{10}	—	5.51×10^{-4}
6	6.53×10^{10}	—	3.12×10^{-3}
8	1.15×10^{11}	—	5.50×10^{-3}

べれば小さいものの、その個数は莫大である。また、前節で述べたアルゴリズムでは、重複を含んで列挙されるため、与えられた配置がこの集合に含まれるか否かを判定するには、重複を取り除く形でテーブルにしなければならない。本論文で用いる方法は、この集合に含まれる要素については確実に「ギャップ集合に含まれる」と判定されるが、含まれない場合にも「ギャップ集合に含まれる」と判定されることを許容するものである。前節で説明したアルゴリズムによって出力された要素について、適当な関数 f を用いて、より小さな整数の空間へ写像する。ここでは、簡易的に

$$f(c) = \left(\sum_{i=0}^3 \sum_{j=0}^3 16^{4i+j} Q(c, (i, j)) \right) \bmod p \quad (6)$$

と定義した。ただし、 $Q(c, (i, j))$ は、配置 c における位置 (i, j) でのコマの番号（空所の場合には 0）とし、 p は大きな素数とする。このように定義すれば、 $f(c)$ は 0 から $p-1$ までの整数に写像される。また、全く同じ配置については同じ値に写像されることになる。適当に大きな整数 p を用いて写像すれば、この値はランダムにこの区間の整数に写像されると考えられる。この p は非常に大きな数であるので、この大きさの配列を用意することは難しいが、ある整数 $q \ll p$ を用意し、0 から $q-1$ までの整数 i について、 $f(c)$ がその値をとる頻度を計算する。それぞれの頻度を $\text{freq}_f(i)$ と書けば、

$$\frac{\sum_{i=0}^{q-1} \text{freq}_f(i)}{\#\{0 \leq i < q \mid \text{freq}_f(i) \neq 0\}} \times N \quad (7)$$

によって、 G_{2n} 集合の大きさを見積もることができる。ただし、 N は重複を含めて前節で述べたアルゴリズムによって出力された配置の個数である。(7)に基づいて、ギャップ集合の要素数を見積もったものを表 1 に示す。 $p = 793,753,737,353$, $q = 10^7$ とした。真の要素数のわかっている G_0 および G_2 の要素数も同時に示している。これを見ると低いところではある程度一致していることが見てとれ、 $G_4 \sim G_8$ についてもある程度一致していると考えられる。また、 G_8 は現在列挙できていない最大の集合であり、 G_8 でも配置全体の 0.5%程度を占める程度である。 G_{10} 以上については、計算量の問題から現在のところ計算できていない。

与えられたコマの配置が、このようなギャップ集合に入っている場合には、確実に入っていると判断するが、

表 2 $|\varphi_{0,7}(G_{2n})|$ の大きさと $\varphi_{0,7}$ の占有率

$2n$	$ \varphi_{0,7}(G_{2n}) $	$ G_{2n} $ の見積り	$ D $ での占有率
0	5,377,918	8.83×10^7	0.010
2	40,513,679	1.36×10^9	0.078
4	147,416,370	1.15×10^{10}	0.284
6	312,563,994	6.53×10^{10}	0.602
8	446,090,977	1.15×10^{11}	0.860

入っていない場合には、かならずしも入っていないと判定できないアルゴリズムを用いて、評価関数を構成する。集合の大きさから、正確なギャップ集合を列挙することは不可能なので、ある程度の情報を蓄えることによって、このような関数を実現する。ある程度の割合の要素について、ギャップ集合に入っていないくても入っていると判断することを許容する。

ここで用いる方法は、コマ配置を空所と 1 から 7 までのコマの位置のみの情報へ射影する関数を用いる。この関数を $\varphi_{0,7}(c)$ とおいて、コマの配置全体の集合と、空所と 1 から 7 までのコマの位置情報の全体の集合をそれぞれ C, D とおけば、 $\varphi_{0,7}: C \rightarrow D$ と書ける。ここで、命題 1 より $|C| \doteq 1.049 \times 10^{13}$ であり、 $|D| = 16!/8! = 518,918,400 \doteq 5.19 \times 10^8$ である。この写像は、8 から 15 までのコマの位置を見ないことにより、小さな空間へ射影しているので、 $\varphi_{0,7}$ は全射であり、 C の要素は、偏りなく D へ射影されると考えられる。 G_{2n} は C の部分集合であり、 $\varphi_{0,7}$ による G_{2n} の像の大きさ $|\varphi_{0,7}(G_{2n})|$ は実際に計算することができる。これを表 2 に示す。この結果を見ると、 G_{2n} の要素数は $n \geq 2$ については $|D|$ を大きく上回っている。もし、 $\varphi_{0,7}$ がランダムに D へ写像されるのであれば、 $2n \geq 2$ については、 $\varphi_{0,7}$ の値域はほとんど D 全体に広がると考えられるが、実際には広がらないで、 D の一部分に集中している。このような結果になる理由は不明であるが、我々は、この性質を利用してギャップ集合を近似する。

長さ $|D|$ の配列を用意し、それぞれの要素の初期値を 0 としておく。ギャップ集合の要素を列挙しながら、 $\varphi_{0,7}$ によって写像される値に対応する配列要素を 1 に変更することにより、ギャップ集合を記録する。この配列を $\text{gap}_{2n}[i]$ とおき、ある配置 c が G_{2n} に入っているかどうかは、 $\text{gap}_{2n}[\varphi_{0,7}(c)]$ が 1 であるかどうかによって判定する。この場合、ある配置 c について $c \in G_{2n}$ ならば $\text{gap}_{2n}[\varphi_{0,7}(c)] = 1$ であるはずであり、正しく判定できる。一方、 $c \notin G_{2n}$ の場合、 c が C 上の任意の値を等確率にとると仮定すれば、誤り確率

$$\Pr(\text{gap}_{2n}[\varphi_{0,7}(c)] = 1 \mid c \notin G_{2n}) \quad (8)$$

は表 2 の占有率に等しくなる。すなわち、この場合、 $2n = 6$ の場合には、ギャップ集合に入らない要素の 60%程度について、誤ってギャップ集合であると判定してしまうことになる。

10	6	5	4
----	---	---	---

図 4 15 パズルの第 2 行における Linear Conflict の例．コマ 4, 5, 6 について 4 と 5, 5 と 6 についてどのように動かしてもゴール状態にするには、いずれかのコマを迂回させるため結局この部分だけで $M(c) + 4$ 回の移動が必要となる．

$\varphi_{0,7}$ を用いる場合、それぞれの G_{2n} につき大きさ $|D|$ の配列が必要になり、そのために、それぞれにつき $|D|/8 \approx 62$ MiB の記憶容量を必要とする．しかし、第 7 節で述べるように、実はいくつかの G_{2n} についてまとめて記録の方が効率的である．たとえば、 $G_0 \sim G_8$ の情報をまとめて記録するためには、理論的には約 186 MiB の記憶容量で十分である．我々の実装では、個別のギャップ集合を近似したテーブルを用いるのではなく、それぞれの G_{2n} をまとめたテーブルを利用している．

また、上記方法では、 G_6 の判定において、ギャップ集合に入っていないのに入っていると判定されるものは 60% 程度、 G_8 の判定においては、86% となり、判定の効率が良くない．そこで、 $\varphi_{0,7}$ とは別に 8 から 15 のコマの位置によって作られる空間 E への写像 $\varphi_{8,15}: C \rightarrow E$ を考え、同様にして配列の情報を取得し、それぞれの判定方法で共にギャップ集合に帰属すると判定された場合のみギャップ集合に入っていると判定することにより、誤り確率を減らすことができる．実際、この方法によって、 G_6 による誤り確率を 40% 程度にすることができ、 G_8 については、75% 程度にすることができる．

以上に述べた G_{2n} の近似方法で得られた情報を式 (5) で示される $h_n(c)$ に適用することにより、評価関数を定義する．ある配置 c が G_{2n} の要素でないのに、要素であると判定した場合、 $h_n(c)$ は小さな値となるので、この関数は許容的な評価関数となる．

空所と 1 から 7 までのコマの位置を用いた写像方法は、他のいくつかのコマの選択方法と比較した限りでは、前述の占有率が比較的低い組み合わせであった．しかし、どのコマを用いることが最適であるかについては不明であり、今後検討する必要がある．

6. Linear Conflict とパターンデータベースについて

次節では、ここで提案する手法、Linear Conflict (LC) による方法およびパターンデータベース (PDB) による方法を比較する．そのため、本節では、簡単に LC および PDB による方法を説明する．

LC [Hansson 92] は、スライディングパズルのそれぞれの行と列に注目する．ある配置において、ある特定の行に置かれているいくつかのコマについて、そのゴール状態での位置もその行にあり、いずれかのコマを迂回させなければ、ゴール状態に移動させることが出来ない状

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

図 5 ここで使用する加算的 PDB のコマの分類方法．それぞれ 5-5-5 (左上)、7-7-1 (右上)、7-8 (左下) と表記する．

況では、 $M(c)$ にパターンに応じて 2, 4 または 6 を加えたものも許容的であることがわかる．図 4 の例では 15 パズルの第 2 行だけについて $M(c)$ に 4 を加えても許容的である．全ての行と列についてこのような値を足し加えたものが LC によって得られる評価関数である．定義より、 G_0 の要素である配置は、このようなパターンを含まない．また、 G_{2n} は LC によって $M(c)$ に $2n + 2$ 足し加えられるようなパターンを含まない．したがって、十分に大きい n についてそれ以下のギャップ集合が得られているのであれば、それらを用いた評価関数は、LC による評価関数よりも常に大きくなるが、実際には G_{2n} の n は小さな値についてしか計算できないので、両者は異なるものである．

本論文で用いる PDB は Korf ら [Korf 02] および Felner ら [Felner 04a] による加算的 PDB であり、15 パズルについては、パズルのコマをいくつかのグループに分けて、それぞれについて構成する (図 5 参照)．

ここでの PDB は、それぞれのコマのグループのそれぞれのコマが枠のどの位置にあるかというパターンについて、グループ以外のコマの違いを無視し、動きを数えないことにより、ゴール状態から最少何回でその状態することができるかを記録したデータベースである．動きを逆回しにすることにより、データベースの内容は同時に、そのパターンから最少何回でゴール状態に至ることができるかという回数となる．

ここで用いる PDB は、図 5 に示す 5-5-5, 7-7-1, 7-8 の 3 種類である．それぞれの PDB の容量を表 3 に示す．PDB のサイズはコマのグループサイズが大きくなると飛躍的に大きくなる．

7. 計算機実験による結果

第 5 節で述べたギャップ集合を近似したテーブルを実際に G_8 まで計算し、それにより構成された評価関数を用いてランダムに生成した 300 種類の 15 パズルの配置についてゴール状態までの最少手順を IDA* アルゴリズムにより求め、探索した平均ノード数および CPU 時間

を Linear Conflict や加算的 PDB による評価関数を用いた場合と比較した。

ギャップ集合の計算については、 G_0 から G_4 までは、後述する 1 台の計算機を用いて計算し、その計算時間はそれぞれ 47 秒、1,124 秒、 1.39×10^4 秒であった。 G_6 と G_8 については、性能の異なる計算機を複数台用いて深さ優先探索を分割して計算した。探索ノード数はそれぞれ 9.59×10^{11} と 9.32×10^{12} 程度であり、のべ計算時間は後述の計算を含めてそれぞれ 44 時間と 440 時間程度であった。それぞれの 15 パズルの配置から最少手順を求める計算には、つぎの計算機およびコンパイラを用いた：Apple Mac Pro (CPU: Quad-Core Intel Xeon 2.26 GHz $\times$ 2, Memory: 16GB), GCC 4.2.1 (コンパイルオプション: -O3)。また、それぞれの G_{2n} を近似するファイルは、それぞれのギャップについて用意するのではなく、限定されたコマの配置について、それに対応するパターンがどの集合に含まれるかによって 1~3 ビットで表現している。たとえば、 G_0 から G_4 までを用いる場合には、「 G_{2n} 以上に含まれる」($n = 0, 1, 2$)、または「いずれにも含まれない」の 4 通りの可能性があるので、それぞれのパターンに対応して 2 ビットで表現される。同様に、 G_6 や G_8 では 3 ビットで表現されるが、実際には実装の都合で 4 ビットを用いている。これらのファイルサイズはすべて同じである（計算可能かどうかは別にして G_{28}

表 3 それぞれの評価関数による平均ノード探索数 A 、平均 CPU 時間 T (秒)、および DB サイズ E (MiB)。MD はマンハッタン距離関数をそのまま評価関数として用いたものであり、LC は Linear Conflict による評価関数である。Gap $2n$ はギャップ $2n$ 集合 G_{2n} の近似による評価関数である。特に Gap6-1 は空所と 1 から 7 までのコマの位置でテーブルを作ったものであり、Gap6-2 はそれに加えて 8 から 15 までのコマの位置によるテーブルを併用したものである。Gap8-1 と Gap8-2 についても同様である。それらの下の評価関数はそれぞれ PDB による評価関数である。また、下 3 行には LC とギャップ集合を併用した評価関数の結果を示した。

評価関数	A	T	E
MD	5.26×10^8	143.5	—
LC	5.85×10^7	28.3	—
Gap0	9.25×10^7	60.7	62
Gap2	1.63×10^7	10.7	124
Gap4	3.03×10^6	2.03	124
Gap6-1	8.12×10^5	0.54	247
Gap6-2	5.15×10^5	0.56	494
Gap8-1	4.85×10^5	0.33	247
Gap8-2	1.28×10^5	0.13	494
5-5-5	4.30×10^6	2.38	157
7-7-1	1.66×10^5	0.097	115
7-8	4.32×10^4	0.027	577
LC + Gap0	3.08×10^7	31.32	62
LC + Gap6-2	4.95×10^5	0.69	494
LC + Gap8-2	1.28×10^5	0.17	494

表 4 PDB による評価関数とギャップ集合による評価関数を用いた平均探索ノード数と平均 CPU 時間 (秒) (括弧内)。第 1 行は PDB のみを用いたときの結果である。

	5-5-5	7-7-1	7-8
	4.30×10^6 (2.38)	1.66×10^5 (0.097)	4.32×10^4 (0.027)
Gap2	1.80×10^6 (1.94)	1.45×10^5 (0.169)	4.04×10^4 (0.048)
Gap4	7.94×10^5 (0.852)	1.15×10^5 (0.129)	3.53×10^4 (0.043)
Gap6-2	2.52×10^5 (0.378)	5.84×10^4 (0.092)	2.45×10^4 (0.042)
Gap8-2	9.84×10^4 (0.143)	4.00×10^4 (0.063)	2.05×10^4 (0.037)

までは同じ 4 ビットで足りる)。パターンの写像方法を 2 種類用いる場合にはテーブルが 2 つになるので、ファイルサイズは 2 倍となる。

計算の結果を表 3 に示す。 G_8 をコマ 1~7 によるテーブルとコマ 8~15 によるテーブルで近似した評価関数 Gap8-2 を用いて、マンハッタン距離評価関数を用いた場合に比べて平均で約 4,000 分の 1 程度に探索ノード数を減らすことができる。これに対して、PDB による評価関数では、7-8 のデータベースで 10,000 分の 1 以下にすることができる。

また、PDB による評価関数とギャップ集合による評価関数の値のうち大きい方を採用する評価関数を用いることにより、探索ノード数を減らすことができる。表 4 に結果を示す。Gap8-2 を用いることにより、平均探索ノード数を PDB 5-5-5 は 40 分の 1 以下に、7-8 は 47% 程度にすることができた。一方、平均 CPU 時間については、PDB のテーブルとギャップ集合のテーブルを共に参照しなければならず、5-5-5 と 7-7-1 では Gap8-2 との組み合わせで改善しているが、7-8 ではギャップ集合のすべて組み合わせについて、むしろ余計に時間がかかってしまっている。

LC とギャップ集合による評価関数については、Gap0 と LC では LC の方が探索回数、計算時間共に小さくなる。しかし、Gap2 以上のギャップ集合については、LC よりも良い結果が得られた。LC のみを用いることによって、平均探索ノード数は MD の約 1/9 であり、Gap0 のみを用いると、MD の約 1/6 である。LC と Gap0 のうち大きい方を用いることによって、平均探索ノード数は 30,804,190 となる。これは MD の探索ノード数の約 1/17 である。前述のように全体で +2 できるような Linear Conflict のあるパターンは G_0 に含まれないので、Gap0 でも +2 されているが、それ以上のものは Gap0 では 4 以上加算することができない。一方、Gap0 には LC とは全く関係のないようなパターンについても +2 することが可能であるため、両者を組み合わせた関数はお互いに欠けた部分

を補い合っていると考えられる．これに対して，Gap8-2 と組み合わせると平均探索回数はほとんど変わらなくなる．これは，ほとんどの Linear Conflict を持つ配置がすでに Gap8-2 によって処理されてしまっているからであると考えられる．

8. ま と め

本論文は，15 パズルにおけるマンハッタン距離評価関数にある値を加えた許容的評価関数を提案した．まず，マンハッタン距離評価関数と現実ゴール状態まで変化させるのに必要な手数とのずれが少ないコマの配置を列挙し，それをテーブルとして蓄える．我々の方法は，そのようなずれの少ない配置の集合に含まれない配置については，マンハッタン距離評価関数にある値を加えても本来の最短手順数を超えないという性質に基づいている．特に 15 パズルについては，マンハッタン距離評価関数の値と現実のゴール状態までの最少手数の差は必ず偶数であるという性質から，例えば，ギャップゼロ集合（関数値の差が 0 であるコマ配置の集合）に入らなければ，マンハッタン距離評価関数値を 2 増やすことが可能である．さらに差が 2 以下であるコマの配置の集合（ギャップ 2 集合）に含まれなければ，4 増やすことが可能となる．本論文では，ギャップ 8 集合までを近似的に求め，それを用いて評価関数を構成した．これと IDA* アルゴリズムを用いた 15 パズルの最短手順による解法により平均探索回数をマンハッタン距離評価関数の 4,000 分の 1 程度にすることができた．

加算的 PDB を用いた場合，7-8 のパターンでは，マンハッタン距離評価関数の約 10,000 分の 1 程度まで平均探索ノード数を削減することができ，それと比較すると Gap8-2 でも 3 倍程度の開きがある．用いるテーブルのサイズは PDB の方が約 20% ほど大きい．ギャップ集合による方法は，PDB による方法とは，全く別の発想に基づくものであり，これら 2 つの方法を組み合わせた場合でもある程度性能を向上させることが可能である．実際，7-8 の PDB による評価関数値と Gap8-2 による値のうち大きい方の値を採用することにより，7-8 の PDB による評価関数の 47% 程度まで平均探索ノード数を減らすことができた．

ここで述べたギャップ集合は，パターンデータベースのように，ゴールに至るまでの手順数を記憶するのではなく，与えられた配置がギャップ集合の要素であるか否かを判定できれば良い．本論文では，コマの部分的なパターンを用いてこれを記憶したが，たとえば，ギャップ 6 集合ではこの集合に属さないのに属していると判定される割合が 60% 程度存在する．これは，別のコマの部分的なパターンを併用することにより，40% 程度まで落とすことができたが，それでも不十分である．ギャップ 8 集合については 2 つの集合を用いても，誤って判定され

る確率はそれほど低くならない．ギャップ集合をより効率的に蓄える方法があれば，平均探索ノード数をもっと減らすことができると考えられる．これは今後の課題である．

また，ギャップ集合による方法は，原理的には 24 パズル [Korf 96] など，よりコマの多い同様のスライディングパズルにも用いること可能であるが，実際には 24 パズルについては，ギャップゼロ集合ですら，膨大な計算時間を要し現在のところうまく応用できていない．ギャップ集合をよりコンパクトに表現する方法などの開発が必要であると考えられる．また，ギャップ集合をなるべく重複を少なくした形で出力する方法などについても検討が必要である．これらも今後の課題である．

◇ 参 考 文 献 ◇

- [Felner 04a] Felner, A., Korf, R. E., and Hannan, S.: Additive pattern database heuristics, *Journal of Artificial Intelligence Research*, Vol. 22, No. 1, pp. 279–318 (2004)
- [Felner 04b] Felner, A., Meshulam, R., Holte, R., and Korf, R.: Compressing pattern databases, in *AAAI-04*, pp. 638–643 (2004)
- [Hansson 92] Hansson, O., Mayer, A., and Yung, M.: Criticizing solutions to relaxed models yields powerful admissible heuristics, *Information Sciences*, Vol. 63, No. 3, pp. 207–227 (1992)
- [Hart 68] Hart, P. E., Nilsson, N. J., and Raphael, B.: A formal basis for the heuristic determination of minimum cost paths, *IEEE Transactions on Systems Science and Cybernetics*, Vol. 4, No. 2, pp. 100–107 (1968)
- [Johnson 79] Johnson, W. W. and Story, W. E.: Notes on the “15” puzzle, *American Journal of Mathematics*, Vol. 2, No. 4, pp. 397–404 (1879)
- [Korf 85] Korf, R. E.: Depth-first iterative-deepening: an optimal admissible tree search, *Artificial Intelligence*, Vol. 27, No. 1, pp. 97–109 (1985)
- [Korf 96] Korf, R. E. and Taylor, L. A.: Finding optimal solutions to the twenty-four puzzle, in *AAAI-96*, pp. 1202–1207 (1996)
- [Korf 02] Korf, R. E. and Felner, A.: disjoint pattern database heuristics, *Artificial Intelligence*, Vol. 134, No. 1-2, pp. 9–22 (2002)

〔担当委員：岸本 章宏〕

2010 年 10 月 27 日 受理

—— 著 者 紹 介 ——

山本 修身 (正会員)

1987 年 東京大学大学院 工学系研究科 計数工学専攻 修士課程修了．東京工業大学理学部情報科学科助手，株式会社管理工学研究所 研究員，青森大学工学部情報システム工学科助教授を経て，現在，名城大学理工学部情報工学科教授．博士（工学）．日本応用数理学会，情報処理学会，電気学会，各会員．

佐藤根 寛

2009 年 名城大学 大学院 理工学研究科 情報科学専攻 修了．修士（工学）．現在，名城大学 大学院 理工学研究科 研究生．