人工知能 課題番号 5「制約従属問題を TMS を用いて解く」

工学部電子情報工学科 03-175001 浅井明里

2017 年 10 月 27 日

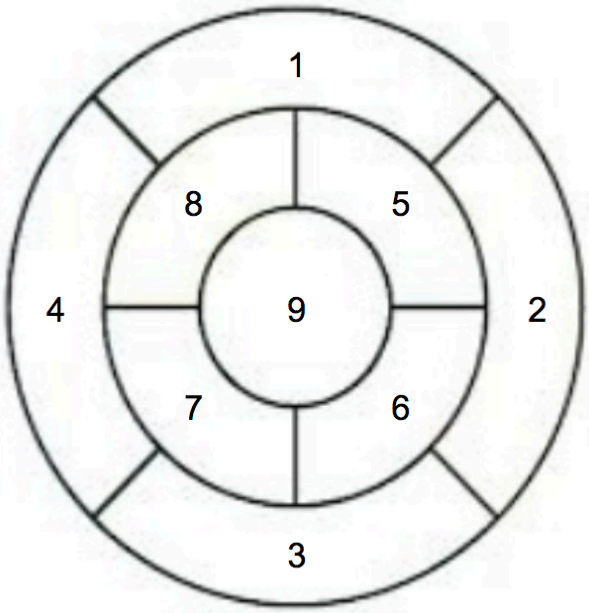

図 1　4 色塗り分け問題の初期状態

1　TMS とは

数学的な論理では仮定は常に正しいとされるが、現実世界においては状況が変化する等により、新しい事実が刻一刻と生まれ、これまでの結論を修正しなくてはいけないことが多々ある。たとえば、「明日はおそらく台風が接近するため天候が大きく荒れるはずだ」「天候が大きく荒れている時に通学をするのは危険なため、おそらく明日の午前の授業は休講となるだろう」「つまり明日の午前は休講となるため、大学に行く必要がない」と推定したとしても、実際に次の日に台風が接近しなかった場合、授業は休講にならず、我々は大学に行かなくてはいけなくなる。こういった非単調論理推論の代表例に TMS(Truth-mentenace system) というシステムがある。

TMS において、データの状態には以下の二種類が存在する。

- In 状態：現在のデータに対応する命題を信じている
- Out 状態：現在のデータに対応する命題を信じている

TMS で重要となるのは「矛盾」の解消であり、矛盾が発生した際には、矛盾を起こす原因のデータを Out とし、代わりに他のデータを In することで、仮説の In/Out 状態を修正して真理を維持する。

2　TMS で領域 4 色塗り分け問題を解く

本レポートでは、いくつかの領域に分割された地図を 4 色で塗り分ける制約従属問題を TMS で解いていく。今回は図 1 で示された地図を 4 色で塗り分ける場合を考える。

2.1 仮説知識、NoGood 知識を制約条件から求める

まず、TMS のデータベースを構成する。ある地図を「赤」「青」「黄」「緑」の四色で塗り分ける時、以下のような制約が存在する。

- それぞれの領域は必ずどれか一色で塗られており、終末状態において白色のまま、もしくは二色で一つの領域が塗られてはならない。
- 隣り合う領域は異なる色で塗られていなくてはいけない。

この制約条件に基づき、TMS の仮説知識のデータを構築する。n 分割された領域について、それぞれ「領域 $k(1 \leq n)$」と名付け、「赤」「青」「黄」「緑」について、「R」「B」「Y」「G」とし、例えば「領域 1 が赤色で塗られている」という知識を「R1」と表すこととする。また、どの色にも塗り分けられていない状態を「白色」すなわち「W」として仮に領域 1 がどの色にも塗り分けられていない時、この状態を「W1」とする。

この表記を用いて、まず「ある領域はどれか一色で塗られている、そうでなければ白色のままである」という仮説知識を、領域 1 について構成する。

- R1：SL：IN:, Out:G1, B1, Y1
- G1：SL：IN:, Out:R1, B1, Y1
- B1：SL：IN:, Out:R1, G1, Y1
- Y1：SL：IN:, Out:R1, G1, B1
- W1：SL：IN:, Out:R1, G1, B1, Y1

他の 8 つの領域についても同様の仮説知識が成立するため、計 $5 \times 9 = 45$ 個の仮説知識が構成できる。

次に、No-Good 知識について考えてみる。この 4 色塗り分け問題における No-Good な状態とは、「彩色されていない領域が地図上にある」もしくは「隣り合う領域が同じ色で彩色されている」という状態である。ある領域が彩色されていないという No-Good 状態は以下の 9 通り存在する。

- NG-W1:NoGood:W1
- NG-W2:NoGood:W2
- NG-W3:NoGood:W3
- NG-W4:NoGood:W4
- NG-W5:NoGood:W5
- NG-W6:NoGood:W6
- NG-W7:NoGood:W7
- NG-W8:NoGood:W8
- NG-W9:N0Good:W9

また、領域 1 から 9 について、隣り合っている領域は表 1 の通りである。No-Good の例は隣接する領域の組み合わせの数が 20 通り存在し、それぞれについて 4 色考える必要があるので $20 \times 4 = 80$ 種類存在し、いずれかの領域が白色となる No-Good と合わせて計 89 種類の No-Good 状態が存在することになる。

表 1　それぞれの領域に隣接する領域と、No-Good の例

領域	隣接する領域	No-Good の例
1	2, 4, 5, 8	NG-R12：NoGood:R1, R2
2	1, 3, 5, 6	NG-R23：NoGood:R2, R3
3	2, 4, 6, 7	NG-R34：NoGood:R3, R4
4	1, 3, 7, 8	NG-R47：NoGood:R4, R7
5	1, 2, 6, 8, 9	NG-R56：NoGood:R5, R6
6	2, 3, 5, 7, 8	NG-R67：NoGood:R6, R7
7	3, 4, 6, 8, 9	NG-R78：NoGood:R7, R8
8	1, 4, 5, 7, 9	NG-R89：NoGood:R8, R9
9	5, 6, 7, 8	NG-R95：NoGood:R9, R5

2.2　4 色塗り分け問題に対する TMS 解法の流れ

TMS により隣接する領域の 4 色塗り分け問題を解くには以下のようなステップが必要である。

1. 全ての仮説知識、NoGood 知識 (制約条件) を設定する。
2. いずれの仮説知識も Out の初期状態から開始する。
3. ここからランダムに仮説知識の要素を Out から選択し In とする。
4. NoGood 知識と照らし合わせて矛盾が生じるか確認する。矛盾がなければ次の領域を塗り分けていく。
5. 矛盾が In 状態となった場合には、この解消のためサポートリストの In 状態に含まれる要素がランダムに選ばれ、そのサポートリストの Out に含まれる要素をランダムに選び、矛盾を生んだ仮説知識については Out とする。
6. これを全ての領域が塗り終わるまで繰り返す。

今回は、このステップを実行し 4 色塗り分け問題を解く、TMSColorSolver.cpp というプログラムを作成しその結果を観測した。

2.3　4 色塗り分けプログラムの実行結果

前述のプログラムを以下のコマンドにより実行した。

```
$ g++ -std=c++11 TMSSolver.cpp
$ ./a.out
```

出力結果については本レポートの最後のページに添付した。

1. まず、初期状態から W1 が In に追加される。
2. W1 は NoGood 状態として定義されているため、Solver はこの矛盾を解消するため、Out から R1 を取り出す。
3. W1 が Out へ、R1 は In に追加される。
4. 同様に W2、W3 も一旦 In に追加された後、NoGood 解消のため別の仮説知識が Out から取り出さ

れる。

5. 領域 2、3 共に黄色で塗られ、これは $Y_{2,3}$ という NoGood 状態であるため、Out から G2 が In に入り、領域 2 は緑色に塗り替えられる。
6. 領域 4 についても同様の手順で色付けがされ、結果として領域 1 から 4 までが一旦図 3 のように塗り分けられる。
7. 領域 5 が一旦赤で塗られるが、$R_{2,5}$ となり NoGood 状態であるため、Out から B5 が取り出され、領域 5 が青く塗り変えられる。
8. Out から B6 が取り出されるが、これは $B_{5,6}$ となるため、Out から G5 が取り出され In となる。
9. 領域 1 と 5 共に緑で塗られることとなり、NoGood 状態となるため、out から R5 が取り出される。
10. 同様の手順で矛盾を解消していき、結果的に領域 6 までが図 3 のように塗り分けられた。
11. 領域 7 も同様の手順で塗り分けていく。まず R7 が Out から取り出されるが、これは $R_{4,7}$ という NoGood 状態となるため、B4 が取り出され、領域 4 が青に塗り替えられるがこれは $B_{4,1}$ という別の NoGood 状態となる。
12. この NoGood の解消のため、G1, R2, G2, B1, G4 が順に Out から In に取り出され、結果として領域 7 までが図 4 のように塗り分けられる。
13. 残り 2 領域についても、ランダムに Out から仮説知識を取り出し、仮に NoGood が In に入った際には矛盾の解消のため異なる仮説知識を Out から取り出す作業を続け、最終的に塗り分け結果は図 5 のようになった。

今回は 9 領域の 4 色塗り分けについて、TMS から正しく正解を導くことができた。TMS はデータの相互依存関係を保持することにより、矛盾が怒った時にその矛盾に関連するデータの In/Out 状態だけを変更して矛盾を解消することができるという特長をうまく体現していると言える。一方で、矛盾が生じた際にどのデータを In/Out にするかはランダムに決定されるために、やや非効率とも思える選択をしている場面も見受けられた。例えば、最終領域を塗り分ける際、人間であれば隣接する 4 領域で使用されていない緑で塗れば完成するとすぐに判断することができるであろうが、

また、今回のプログラムでは一度ランダムに B9 が In となったために、結果として生じた多くの矛盾の解消のため非効率な計算が行われることとなった。今回のような少ない領域の塗り分け問題については TMS でも十分正しい解答を導くことができるが、TMS のようにランダムに矛盾のない解を見つけようとするアルゴリズムは問題がより複雑により多くの状態が想定できる場合にはどれくらいの計算が必要なのか予測できないという問題がある。

参考文献

[1] 伊庭斉志, 『人工知能と人工生命の基礎』, オーム社 , 2013.

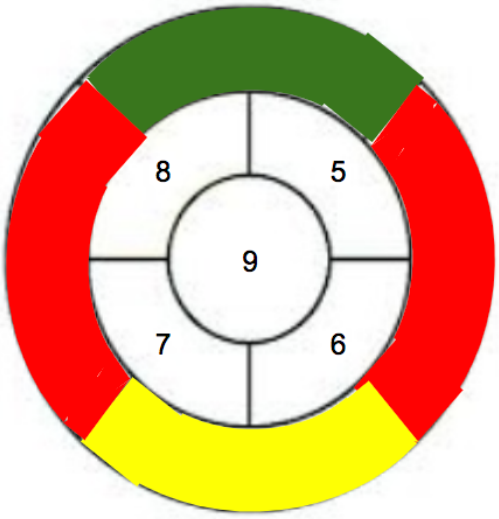

図 2　6 ステップ目終了時の塗り分けの様子

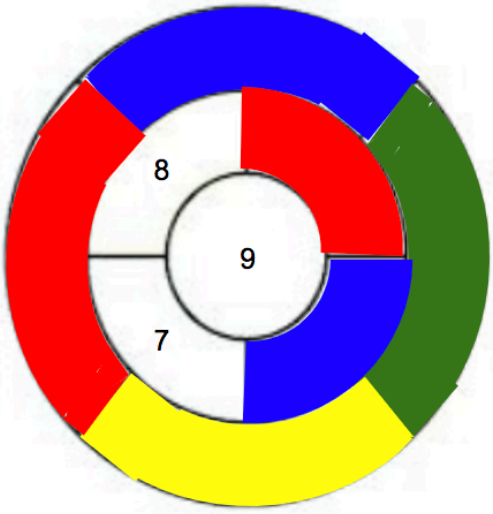

図 3　10 ステップ目終了時の塗り分けの様子

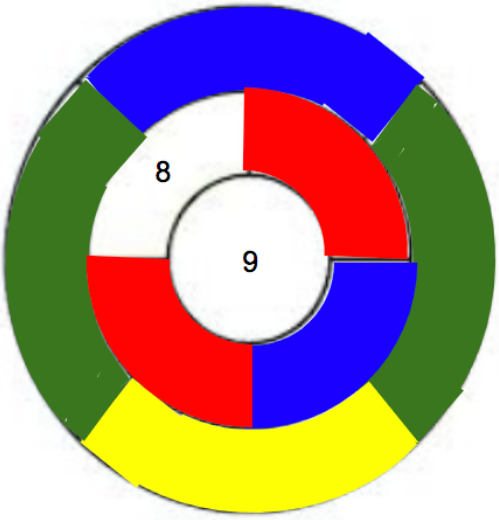

図 4　12 ステップ目終了時の塗り分けの様子

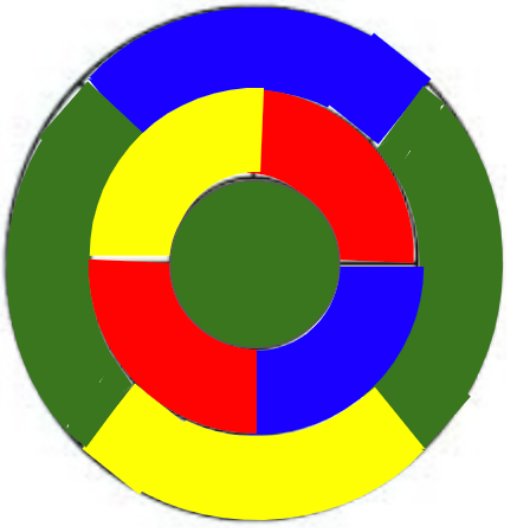

図 5　最終的な塗り分けの結果

以下は TMSColorSolver.cpp の出力結果である。

```
step : W1 turned to In
NG_W1 need to be resolved
step : W1 turned out, R1 turned in
step : W2 turned to In
NG_W2 need to be resolved
step : W2 turned out, Y2 turned in
step : W3 turned to In
NG_W3 need to be resolved
step : W3 turned out, Y3 turned in
NG_Y23 need to be resolved
step : Y2 turned out, G2 turned in
step : W4 turned to In
NG_W4 need to be resolved
step : W4 turned out, R4 turned in
NG_R41 need to be resolved
step : R1 turned out, G1 turned in
NG_G12 need to be resolved
step : G2 turned out, R2 turned in
step : W5 turned to In
NG_W5 need to be resolved
step : W5 turned out, R5 turned in
NG_R25 need to be resolved
step : R5 turned out, B5 turned in
step : W6 turned to In
NG_W6 need to be resolved
step : W6 turned out, B6 turned in
NG_B56 need to be resolved
step : B5 turned out, G5 turned in
NG_G15 need to be resolved
step : G5 turned out, R5 turned in
NG_R25 need to be resolved
step : R2 turned out, Y2 turned in
NG_Y23 need to be resolved
step : Y2 turned out, G2 turned in
NG_G12 need to be resolved
step : G1 turned out, R1 turned in
NG_R41 need to be resolved
step : R1 turned out, B1 turned in
step : W7 turned to In
NG_W7 need to be resolved
step : W7 turned out, R7 turned in
NG_R47 need to be resolved
step : R4 turned out, B4 turned in
NG_B41 need to be resolved
step : B1 turned out, G1 turned in
NG_G12 need to be resolved
step : G2 turned out, R2 turned in
NG_R25 need to be resolved
step : R2 turned out, G2 turned in
NG_G12 need to be resolved
step : G1 turned out, B1 turned in
NG_B41 need to be resolved
step : B4 turned out, G4 turned in
step : W8 turned to In
NG_W8 need to be resolved
step : W8 turned out, Y8 turned in
```

```
step : W9 turned to In
NG_W9 need to be resolved
step : W9 turned out, B9 turned in
NG_B69 need to be resolved
step : B6 turned out, Y6 turned in
NG_Y36 need to be resolved
step : Y6 turned out, G6 turned in
NG_G26 need to be resolved
step : G6 turned out, B6 turned in
NG_B69 need to be resolved
step : B6 turned out, Y6 turned in
NG_Y36 need to be resolved
step : Y6 turned out, B6 turned in
NG_B69 need to be resolved
step : B9 turned out, G9 turned in
step : Successfully finished coloring!
```