

5

制約充足問題

本章では、各状態を小さなブラックボックスとして扱うのではなく、各状態がどのように構成されているかを考えることにより、種々の新しい強力な探索手法が得られ、問題の構造と複雑さに関する深い理解が得られるこことを示す。

3章と4章では、状態(state)で構成される空間を探索することにより問題を解くというアイデアについて検討した。これらの状態は、領域依存のヒューリスティックスによって評価され、ゴール状態かどうかのチェックが行われた。このような探索アルゴリズムからみれば、各状態は識別可能な内部状態をもたないブラックボックス(black box)でしかない。各状態は問題依存の手続き、すなわち、後者閾数、ヒューリスティック閾数、ゴールテストによってのみアクセスされる任意のデータ構造で表現される。

本章は制約充足問題(constraint satisfaction problem: CSP)を扱う。制約充足問題での状態(およびゴールテスト)は、個々の問題に依存せず、標準的で構造化された、非常に単純な表現(representation)(5.1節)をもつ。探索アルゴリズムは、状態の構造を利用して定義され、問題依存ではなく、汎用的なヒューリスティックスを用いることにより、大規模な問題を解くことが可能となる(5.2~5.3節)。また、ゴールテストが標準化された単純な表現をもつことから、問題の構造を解析することが可能となつていていることでも重要である(5.4節)。このことにより、問題を分割する方法、および問題の構造と、問題の解くことの難しさの間の密接な関係が明らかとなる。

5.1 制約充足問題

目的関数

すべての変数に値が割り当てられた割当であり、制約充足問題の解(solution)とは、すべての制約を満足する完全な割当である。ある種の制約充足問題では、この条件に加えて、解が目的関数(objective function)を最大化することも要求する。

以下に具体例を示そう。ルーマニアには飽きて、オーストラリアの地図を見ているとしよう。図5.1(a)に示すように、この地図は州および準州を示している。ここで、各地域を赤、緑、青のいずれかで、隣合う地域が異なる色になるように塗り分けるという課題を考えよう。この課題を制約充足問題として定式化するために、各地域に対する変数 WA, NT, Q, NSW, V, SA, T を定義する。各変数の領域は集合 $\{red, green, blue\}$ である。隣合う領域が異なる色であるという制約は、たとえば、 WA と NT に関する許される組合は、以下のペアであると表現される。

$$\{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)\}$$

(制約は、制約充足アルゴリズムがこのような表現を処理可能であれば、より簡潔に、等しくないという関係を使って $WA \neq NT$ のようにも表現できる。)この問題には多くの解がある。一例を示す。

$$\{ WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = red \}$$

制約充足問題は、図5.1(b)に示すような制約グラフ(constraint graph)として表現できる。制約グラフでは頂点が変数を、辺が制約を意味する。

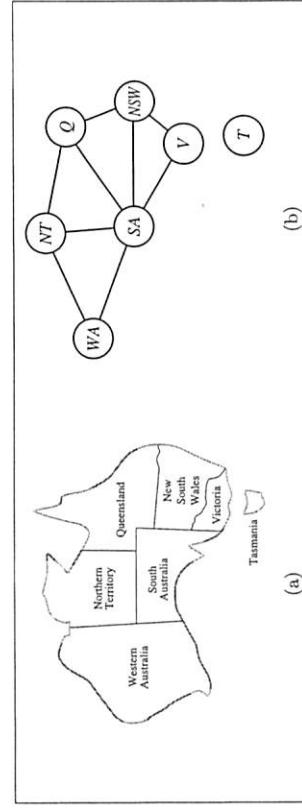


図5.1 (a) オーストラリアの州および準州、この地図を塗り分ける問題は制約充足問題として表現できる。目的は隣接する地域を異なる色で塗り分けることである。(b) 地図の色塗り問題の制約グラフによる表現。

問題を制約充足問題として表現することにはいくつかの利点がある。制約充足問題における状態は、変数の集合に対して値が割り当てられているという標準的な構造にしたがう。これにより、後者閾数およびゴールテストは、任意の制約充足問題に適用可能な汎用的な方法で記述できる。また、問題依存の知識を必要としない、効率的なヒューリスティックスを用いることができる。さらに、制約グラフの構造を用いて、解を求めるプロセスを簡化することができます。ある場合には複雑さの指數的な削減が可能となる。制約充足問題という問題の表現は、本書で扱う一連の問題表現の体系で最初に示される、最も簡単なものである。明らかに、制約充足問題に関する限り、通常の探索問題と同様な漸増的定式化(incremental formulation)が可能である：

形式的には、制約充足問題(constraint satisfaction problem: CSP)は、変数(variable)の集合 X_1, X_2, \dots, X_n および制約(constraint)の集合 C_1, C_2, \dots, C_m によって定義される。各変数 X_i は、可能な値(value)に対応する、空集合ではない領域(domain) (D_i) をもつ。各制約(C_i)は、ある変数の組合せを指定する。問題における状態は、変数の部分集合、もしくは全て、許される値の組合せを指定する。問題に対する状態は、変数の部分集合、もしくは全体に対する値の割当で(assignment) $\{X_i = v_i, X_j = v_j, \dots\}$ に対応する。どの制約にも違反しない割当を、無矛盾(consistent)、あるいは正当であるという。完全な割当てとは、

制約充足問題
変数
制約
値
領域
割当
無矛盾

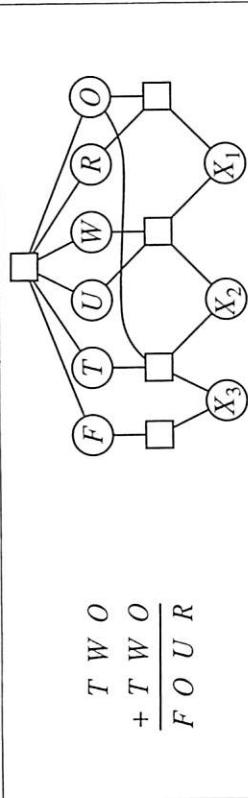


図 5.2 (a) 覆面算の問題。各文字は異なる数字に対応する。目的は、和の計算が正しく、かつ、桁の最初の文字がゼロでないという制約を満足するような、各文字から数字への対応を求める。 (b) 覆面算に対する制約充足グラフ。各列の計算にに関する制約と、各制約が示されている。各制約は、関連する変数と結ばれた四角の箱で表現される。

節でポイントを示す。

5.2 制約充足問題のための後戻り探索

前節では、探索問題としての制約充足問題の定式化を示した。この定式化を用いれば、3 および 4 章で示した任意の探索アルゴリズムは、制約充足問題を解くために用いることが可能である。たとえば、幅優先探索を、前節で示した汎用の制約充足問題の定式化に対して用いることを考えよう。この結果が悲惨であることは明白である。トップレベルでの分歧数は、任意の n 変数に対して、任意の d の値が割当て可能であるため、 nd となる。次のレベルでは、分歧数は $(n-1)d$ であり、 n レベルまで同様に考えることができる。この結果、完全な割当てには d^n 個の葉節点をもつ木を生成することになる！

一見、妥当そうであるが単純にすぎることの定式化は、可換性(commutativity)とよばれる、すべての制約充足問題において共通の重要な性質を無視している。問題が可換であるとは、ある行動の集合に関して、それらを適用する順序が結果に影響を与えないことを意味する。制約充足問題に関しては、この性質は成立している。すなわち、複数の変数に対して値を割り当てる際には、同じ変数に対して同じ値を割り当てる順序に依存せず、同じ部分的な割当てる。よって、すべての制約充足問題を解くアルゴリズムは、探索木中の各節点で、單一の変数に関する可能な値の割当てるのみを考慮して後続節点の生成を行う。たとえば、オーストラリアの地図の色塗り問題では、根節点において、 $SA = red$, $SA = green$ 、および $SA = blue$ から、どれか一つを選ぶという選択を行なうかもしないが、 $SA = red$ と $WA = blue$ からどちらかを選ぶという選択は決して行わない。このような制限を設けることにより、葉節点の個数が d^n となるという望ましい結果が得られる。

後戻り探索(backtracking search)という用語は、一度に一つの変数に対して値を選択し、各変数に関して、正当な値が存在しない場合に後戻りをする深さ優先探索を示すのに

```

function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return RECURSIVE-BACKTRACKING({ }, csp)
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp), assignment, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment according to CONSTRAINTS[csp] then
      add {var = value} to assignment
      result ← RECURSIVE-BACKTRACKING(assignment, csp)
      if result ≠ failure then return result
      remove {var = value} from assignment
    return failure
  
```

図 5.3 制約充足問題のための単純な後戻りアルゴリズム。本アルゴリズムは 3 で示した再帰的深さ優先探索をベースにしている。関数 SELECT-UNASSIGNED-VARIABLE および ORDER-DOMAIN-VALUES は、本文で説明した汎用のヒューリスティックスを実装するのに用いることができる。

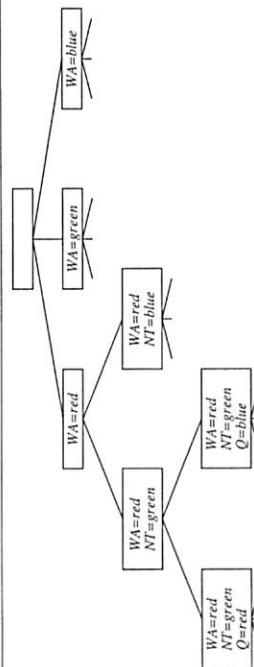


図 5.4 図 5.1 に示した地図の色塗り問題に関する、単純な後戻りアルゴリズムでの探索木の一部。

用いられる。このアルゴリズムの詳細を図 5.3 に示す。このアルゴリズムでは、76 ページに示した、後続節点の生成において、一つの変数のみを考慮の対象とするという方法が用いられている。また、このアルゴリズムでは、現在の割当てをコピーするのではなく、拡張することによって後続節点を得ている。制約充足問題の問題の表現は標準化されているので、BACKTRACKING-SEARCH に対して、領域依存の初期節点、後者閲数、ゴルナルストを与える必要はない。図 5.4 にオーストラリアの地図の色塗り問題での探索木を示す。この例では、 WA, NT, Q, \dots の順に変数の値の割当てが行われている。单純な後戻り探索は、3 章の用語を用いれば知識なしのアルゴリズムである。よって、大規模な問題に関して有効であることは期待できない。図 5.5 の最初の例は、単純な後戻り探索のいくつかの例題に関する評価結果を示しており、この予測と一致している。

4 章では、我々は知識なしの探索アルゴリズムの性能を、問題に関する知識から得られる領域依存のヒューリスティック閲数を用いて改善した。一方、制約充足問題に関しては、このような領域依存の知識なしに、性能改善が可能なことが示される。領域依存の知識を使ふ代わりに、我々は以下の課題に對処することにする：



Problem	Backtracking	BT+MRV	Forward Checking	FC+MRV	Min-Conflicts
USA	(> 1,000K)	(> 1,000K)	2K	60	64
n-Queens	(> 40,000K)	13,500K	(> 40,000K)	81K	4K
Zebra	3,859K	1K	35K	0.5K	2K
Random 1	415K	3K	26K	2K	15K
Random 2	942K	27K	77K		

図 5.5 様々な制約充足アルゴリズムの、いくつかの例題に関する比較。アルゴリズムは左から右に、単純な後戻り探索、MRV ヒューリスティックを用いた前向きチェック、制約違反削除による、各セル中の値は、5 回の実行による解を得るのに要する制約チェックの回数の中間値である。右上の二つのセル以外では、示された数値は 1000(K) を単位としている。括弧中に示された数値は、与えられた上限値以内で解が得られなかったことを示している。最初の問題は、アメリカ合衆国の 50 の州に関するものである。二番目の問題では、 n が 2 から 50 までのすべての n ケイーン問題を解くために必要な数値を示している。三番目の問題は練習問題 5.13 に示すゼブラパズルである。最後の二つは人工的にランダムに生成された問題である。この結果は、MRV ヒューリスティックを用いた前向きチェックが、他の後戻り探索アルゴリズムよりもすべての問題において優れているが、制約違反削除局所探索よりも常に優れているわけではないことを示している。

1. 次にどの変数に関して値を割り当てるべきか？ また、どのような順序で値を選ぶべきか？

2. 現在の値の割当てが、他のまだ値を割り当てていない変数に対して与える影響は何か？

3. 経路が失敗したとき、すなわち、ある変数に関して正当な値が存在しなくなった場合、探索アルゴリズムは、それ以降の経路で同じ失敗の繰返しを避けることができるか？

以下の節では、上記の課題に関して順に答えを示していく。

変数および値の順序づけ

後戻り探索アルゴリズムは、次に示す行を含んでいる。

```
var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
```

デフォルトでは、SELECT-UNASSIGNED-VARIABLE は、リスト VARIABLES[csp] で与えられた順序で、最初のまだ値が割り当てられない変数を選択する。このような静的な変数の順序づけは多くの場合非効率的である。たとえば、 $WA = red$, $NT = green$ なる割当てをした場合、 SA には正当な値は一つしかない。よって、この場合は、次に Q ではなく、 $SA = blue$ なる割当てをすべきである。また、 SA に値を割り当てたあとは、 Q , NSW , および V の割当てはすべて決まってしまう。この直観的に自然なアイデア、すなわち、“正当な”値の数が最少の変数を選ぶことは、最少残存値 (minimum remaining value: MRV) ヒューリスティック とよばれる。このヒューリスティックは、“最大被制約変数”もしくは“早期失敗”ヒューリスティックともよばれてきた。後者の用語は、最も早く失敗を生じることにより、枝刈りができる変数を選ぶとするという意味からつけられた。変数 X に正当な値が存在しない場合、MRV ヒューリスティックは X を選択し、失敗がただちに判明する。よって、他の変数に値を割り当て、いすれば X に値を割り当てる際に失敗する

無意味な探索を避けることができる。図 5.5 の BT+MRV と示された二番目の列は、このヒューリスティックを用いた場合の性能を示している。単純な後戻り探索と比較して、性能は問題によって 3 倍から 3,000 倍向上している。注意すべきこととして、これまでの議論では、どのようにしてこのヒューリスティックを実装するか、そのコストはどの程度かについては触れていないことがある。次節で、妥当なコストでのヒューリスティックを実装する方法を示す。

オーストラリアの色塗り問題で、最初の変数を選ぶ際には、すべての変数の残存値の数は 3 なので、MRV ヒューリスティックは助けにならない。この場合は、次数ヒューリスク ティック (degree heuristic) が役に立つ。このヒューリスティックは、他のまだ値が割り当てられていない変数と、最も多くの制約をもつ変数を選択することにより、将来の選択における分岐数を減らそうとする。図 5.1 では、 SA は最大の次数は 0 で、それ以外の変数の次数は 2 ないし 3 である。実際、 SA を選んでから、次数ヒューリスティックを用いれば、問題は失敗なしに解くことができる。すなわち、各選択ボットにおいて、正当な色を選び続けて、後戻りなしに解に到達することができる。通常は最少残存値ヒューリスティックのほうが、次数ヒューリスティックよりも強力なガイドとなるが、タイブレークの場合に、次数ヒューリスティックを用いることが可能である。

ある変数を選択したとして、次にアルゴリズムはその変数に値を割り当てる順序を決める必要がある。この目的のために、最少割約値 (least constraining-value) ヒューリスティックが有効な場合が存在する。このヒューリスティックは、制約グラフ中で、近傍の変数に關して、ある値を選ぶことにより、選べなくなる値の個数が最も少ない値を優先して選択する。たとえば、図 5.1において、部分的な制約で $WA = red$ および $NT = green$ を行っている状況を考えよう。また、次に値を割り当てる変数を Q とする。明らかに $blue$ は良い選択である。これを選ぶと、 Q の近傍である SA の最後の正当な値がなくなってしまう。よって、最少割約値ヒューリスティックは $blue$ よりも red を選好する。このヒューリスティックは将来の変数の値の割当てに關して、最大限の柔軟性を残そうとするものと考えることができる。もちろん、制約充足問題において、すべての解を求める場合には、いずれはすべての値を選ぶ必要があるので、最初に選ぶ値のみならず、値の順序そのものが問題とならない。制約が強すぎて解が存在しない問題に關しても同様である。

制約を用いた情報の伝播

今までのところ、我々の探索アルゴリズムは、SELECT-UNASSIGNED-VARIABLE で変数が選ばれた時点においてのみ、変数に関する制約を用いた処理を行っていた。しかしながら、探索中のより早い時点で、あるいは探索を始める前に制約を用いた処理を行うことにより、探索空間を劇的に削減することができる。前向きチェック

探索中で、制約をより有效地に活用する方法の一つとして、前向きチェック (forward checking) とよばれる方法がある。この方法では、変数 X の値が割り当てられた時点で、まだ値が割り当てられておらず、 X と制約で結ばれている各変数 Y について、その領域から X の割当てと制約を満足しない値を取り除く。図 5.6 に、地図の色塗り問題における前向きチェックの動作を示す。この例では、注意すべき重要な点が二つある。まず最初に、 $WA = red$ と

前向きチェック

	WA	NT	Q	NSW	V	SA	T
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After WA=red	Ⓐ G B	G B	R G B	R G B	R G B	G B	R G B
After Q=green	Ⓑ B	B	Ⓐ G	R	B	R G B	B
After V=blue	Ⓑ B	B	Ⓐ G	R	Ⓐ G	R G B	R G B

図 5.6 地図の色塗り問題における前向きチェックの処理の経過。WA = red が最初に割り当てられ、前向きチェックは red を近傍の変数 NT, SA の領域から取り除く。Q = green を割り当てたあと、green が NT, SA および NSW の領域から取り除かれ、V = blue を割り当てたあと、blue が NSW および SA の領域から取り除かれ、SA は正当な値をもたなくなる。

$Q = \text{green}$ の割当てを行った時点で、NT と SA の領域は単一の値のみを含む。すなわち、我々は WA と Q から情報を探査することにより、これらの変数に関する分岐をなくすことを可能にしている。明らかに、前向きチェックは red で、Q が green の場合、NT と SA の両方を blue でなければならぬ。WA が red で、Q が green の場合、NT と SA の両方は blue でなければならない。しかしながらこれらの変数は近傍であるため、同じ値をとることはできない。前向きチェックで行われる先読みみは、このような矛盾を検出するのに不十分である。制約伝播 (constraint propagation) とは、ある変数に関する制約の影響を、他の変数に対して伝播することに関する一般的な用語である。この場合は、WA と Q から、NT と SA に対して影響を伝播し (これは前向きチェックによって実行される)、さらに制約が NT と SA に対する伝播されれば矛盾が検出される。制約の伝播は効率的に行いたい、单纯な探索を行う場合と比較して、より多くの労力を費して探索空間を削減しても無意味である。

前向きチェックは多くの矛盾を検出できるが、すべての矛盾が検出できるわけではない。たとえば、図 5.6 の 3 行目をみてよう。WA が red で、Q が green の場合、NT と SA の両方は blue でなければならない。しかしながらこれらの変数は近傍であるため、同じ値をとることはできない。前向きチェックで行われる先読みみは、このような矛盾を検出するのに不十分である。制約伝播 (constraint propagation) とは、ある変数に関する制約の影響を、他の変数に対して伝播することに関する一般的な用語である。この場合は、WA と Q から、NT と SA に対して影響を伝播し (これは前向きチェックによって実行される)、さらに制約が NT と SA に対する伝播されれば矛盾が検出される。制約の伝播は効率的に行いたい、单纯な探索を行う場合と比較して、より多くの労力を費して探索空間を削減しても無意味である。

アーケ無矛盾性 (arc consistency) というアイデアは、前向きチェックにおける有向アーケ、たとえば SA から NSWへのアーケを意味する。変数 SA と NSW の現在の領域に關して、このアーケが無矛盾であるとは、SA のすべての値 x に關して、NSW の現在の領域に x との制約を満たすある値 y が存在することを意味する。図 5.6 の 3 番目の行において、SA の領域は $\{\text{blue}\}$ で、NSW の領域は $\{\text{red}, \text{blue}\}$ である。SA = blue に対しては、NSW に無矛盾な割当て、すなわち、NSW = red が存在する。よって、SA から NSW へのアーケは無矛盾である。一方、反対向きの NSW から SA へのアーケは無矛盾ではない。なぜなら、NSW = blue に対して、SA の無矛盾な割当ては存在しない。このアーケは、NSW の領域から blue を取り除くことによりアーケ無矛盾とすることができます。

探索プロセスの同じ局面で、SA から NT へのアーケの無矛盾性を達成することができる。図 5.6 の 3 番目の行では、両方の変数の領域は $\{\text{blue}\}$ であり、アーケ無矛盾性を達成することにより、SA の領域は空集合となる。すなわち、アーケ無矛盾性を達成することが可能となり、前向きチェックでは検出できなかった矛盾を、より早期に検出することが可能となっている。

アーケ無矛盾性のチェックは、探索プロセスを始める前に、前処理として実行することもできるし、(前向きチェックのように) 変数の値を決定したあとの制約伝播のステップとして実行することもできる (後のアルゴリズムはアーケ無矛盾性維持、Maintaining Arc Consistency を意味する MAC とも呼ばれる)。どちらの場合でも、アーケ無矛盾性の処理は、矛盾が存在しなくなるまで繰り返し実行される必要がある。この理由は、アーケ無矛盾とするためにある変数の領域から値を取り除かれたことが複数回となって、新たな矛盾が生じる可能性があるためである。AC-3 とよばれる、アーケ無矛盾性を達成するアルゴリズムは、チェックすべきアーケをキューで管理している (図 5.7 を参照)。各アーケ (X_i, X_j) は順にキューから取り出されてチェックされる。ある変数 X_i の領域から値を取り除かれた場合、 X_i に向かうすべてのアーケ (X_k, X_i) がキューに再挿入される。アーケ無矛盾性アルゴリズムの複雑さは以下のようによく評価できる。二項制約充足問題は、高々 $O(n^2)$ のアーケをもつ。各アーケ (X_k, X_i) は、キューに対して高々 d 回挿入される。これは、 X_i から取り除かれた値の個数が高々 d であるためである。一つのアーケの無矛盾性チェックポイントとして、 $V = \text{blue}$ としたあと、SA の領域が空集合となつて、このため、前向きチェックは、部分的な割当て $\{WA = \text{red}, Q = \text{green}, V = \text{blue}\}$ が問題で与えられた制約に関して矛盾することを発見し、アルゴリズムはただちに後戻りをする。

```

function AC-3(csp) returns the CSP, possibly with reduced domains
inputs: csp, a binary CSP with variables { $X_1, X_2, \dots, X_n$ }
local variables: queue, a queue of arcs, initially all the arcs in csp

while queue is not empty do
     $(X_i, X_j) \leftarrow \text{REMOVE-FIRST}(queue)$ 
    if REMOVE-INCONSISTENT-VALUES( $X_i, X_j$ ) then
        for each  $X_k$  in NEIGHBORS[ $X_i$ ] do
            add  $(X_k, X_i)$  to queue
    end

    function REMOVE-INCONSISTENT-VALUES( $X_i, X_j$ ) returns true iff we remove a value
    removed  $\leftarrow \text{false}$ 
    for each  $x$  in DOMAIN[ $X_i$ ] do
        if no value  $y$  in DOMAIN[ $X_j$ ] allows  $(x, y)$  to satisfy the constraint between  $X_i$  and  $X_j$ 
            then delete  $x$  from DOMAIN[ $X_i$ ]; removed  $\leftarrow \text{true}$ 
    return removed

```

図 5.7 アーケ無矛盾性アルゴリズム AC-3。AC-3 を適用したあと、すべてのアーケはアーケ無矛盾となるか、あるいはある変数の領域が空となり、制約充足問題が解消される。“AC-3”という名前は、このアルゴリズムが論文で示された三番目のページショーンであるため、アルゴリズムの考案者によって名づけられた (Mackworth, 1977)。

¹ Mohr and Henderson (1986) に示されている AC-4 アルゴリズムは $O(n^2 d^2)$ で実行可能である。練習問題 5.10 を参照のこと。

てより良い方法があることを示す。

図5.1の例で、固定された変数の順序 Q, NSW, V, T, SA, WA, NT を用いた場合にどうなるかを検討してみる。部分的な割当て $\{Q = red, NSW = green, V = blue, T = red\}$ を行つたとして、次の変数 SA に関する、すべての値が制約に違反することとなる。変数 T に戻り、Tasmaniaの色を変えようとする！明らかにこれは無駄である。 Tasmaniaは島であり、この色を変えてでも South Australiaに関する問題は解決しない。後戻りに関するより知的な方法は、直前の変数ではなく、失敗を引き起こした原因となつている変数の集合のいすれかにかかるまで後戻りすることである。このような変数の集合は競合集合 (conflict set) とよばれる。この例では、 SA に対する競合集合は $\{Q, NSW, V\}$ である。一般に、 X に対する競合集合は、 X と制約で結ばれた、すでに値が割り当てられた変数に対して後戻りが実行される。前述の例で、飛越し後戻り法は、BACKTRACKING-SEARCHを通して V に関する新しい値を割り当てる。飛越し後戻り法は、 $conf(X_i)$ 中で、最近に値が割り当てられた変数を以下のように変更することにより簡単に実現できる。すなわち、正的な値をチェックするときに競合集合を蓄積する。正的な値が存在しない場合、最後に値が割り当てられた競合集合中の変数に対して後戻りを実行する。

目ざとい読者は、前向きチェックでは追加のコストなしに競合集合が与えられるることに気がついたろう。 X に値を割り当てるごとに、 Y の領域から値が取り除かれたなら、 X は Y の競合集合に加えられる。さらに、 Y の領域の最後の値が取り除かれたなら、 Y の競合集合に含まれる変数は、 X の競合集合に加えられる。これにより、 Y に到達した場合に、必要があれば、どこに後戻りすべきかの情報を得られる。

さらに目ざとい読者は、奇妙なことに気づいたろう。飛越し後戻りは、変数のすべての値が、現在の割当てと矛盾する場合に生じる。一方、前向きチェックはこのことを検出し、このような節点を訪れることがない。実際、以下の性質が成立する。飛越し後戻りで枝刈りされる分岐は、前向きチェックによつても枝刈りされる。よつて、単純な飛越し後戻りは、前向きチェック探索においては不要であり、より強力な無矛盾性チェック、たとえば MAC に開いても同様である。

上記の事実はあるが、飛越し後戻りの基礎となるアイデア、すなわち、失敗の原因に戻るということは有用である。飛越し後戻りは、変数の領域が空となつたときに失敗に気がつくが、多くの場合にはこれが生じるはるか前に、その分岐は失敗する運命にある。たとえば、以前に考察した部分的な割当て $\{WA = red, NSW = red\}$ を考えよう（以前の考察では、この割当てが矛盾であることが示された）。次に $T = red$ と割り当て、さらに、残りの四つの変数 NT, Q, V, SA に対して割当てを行つたとしよう。これら変数に関して、どの変数に対する割当てを行つても制約は満足されない。ここで、どの変数に対して後戻りするかが問題となる。 NT に関する问题是、現在の割当てと無矛盾な値が存在するので、 NT だけを考えなくてよい。すなわち、完全な競合集合は得られない。一方、わかっていることは、四つの変数 NT, Q, V, SA が、全体として現在の割当てに屬して失敗しているということである。よつて、 NT のような変数の競合集合に関する问题是、以下のようより詳細な定義が必要である。すなわち、競合集合は、 NT および後続の変数に関するといううことである。すなわち、競合集合は、 NT および後続の変数に関するということである。すなわち、競合集合は、 NT および後続の変数に関するということである。アルゴリズムは Tasmaniaを飛び越して NSW に後戻りする。このように定義された競合集合を用いる飛越し後戻り法は、幾合に基づく飛越し後戻り (conflict-directed backjumping)

とよばれる。

この新しい定義の競合集合がどのように計算されるかを説明しよう。方法は簡単である、分歧の“末端”での失敗では、競合集合は以前の定義と同じである。色塗り問題の例では、 SA が失敗したとき、競合集合はたとえば $\{WA, NT, Q\}$ となる。飛越し後戻りは Q に対してなされる。ここで、 Q は、 SA の競合集合を、(Q 自身を除いて) 自分自身の競合集合 $\{NT, NSW\}$ に併合する。この結果、新しい競合集合は $\{WA, NT, NSW\}$ となる。ここで、 Q に他に無矛盾な値がないれば、この競合集合中の変数の現在の割当てに対して、 Q 以降の変数に関する無矛盾な割当てが存在しない。後戻りは、競合集合中で最近に値が割り当てられた NT に対してなされ、 NT は自分自身の競合集合 $\{WA\}$ に、 $\{WA, NT, NSW\} - \{NT\}$ を併合し、新しい競合集合は (前の段落で示したように) $\{WA, NSW\}$ となる。よつて、アルゴリズムは、望みどおり NSW に対して後戻りする。この手続きは以下のようにまとめる。すなわち、 X_j が現在、値を決めようとしている変数で、 $conf(X_j)$ が競合集合とする。もし、 X_j のすべての値が失敗し、競合集合 $conf(X_j)$ 中で、最近に値が割り当てられた変数が X_i なら X_i に後戻りし、 X_i の競合集合を以下とする。

$$conf(X_i) \leftarrow conf(X_i) \cup conf(X_j) - \{X_j\}$$

競合に基づく飛越し後戻りは、探索木中の正しい場所に後戻りすることを可能にするが、同じ過ちを他の分岐で繰り返すこととはできない。制約学習 (constraint learning) とよばれる手法は、探索の過程で得られた新しい制約を問題に加えることにより、同じ過ちを繰り返すことを防ぐことを可能にしている。

5.3 制約充足問題のための局所探索

局所探索アルゴリズム (4.3節参照) (多くの制約充足問題に対して非常に有効である。局所探索アルゴリズムでは、完全な状態による定式化が用いられる。初期状態は、すべての変数に対する、任意の値の割当てである。後者関数は通常、いずれか一つの変数の値を変えた状態である。たとえば 8 クイーン問題では、初期状態は、八つの列それぞれに、ランダムにクイーンを配置した状態であり、後継関数は、任意の列を選んで、その列のクイーンの位置を変更する。別の方法として、1, 2, …, 8 の各行に一つのクイーンを配置し、列をランダムに並べ替えることによって初期状態を生成し、後継関数としては任意の二つの列の入替を用いることも可能である。² 本書ではすでに、山登り型の探索アルゴリズムを n クイーン問題に適用する方法を示している (113 ページ)。また、制約充足問題の特殊な場合である SAT を解くアルゴリズムである WALKSAT (225 ページ) についても紹介した。変数に対する新しい値を選ぶ方法として、最も自明なヒューリスティックとして、他の変数との制約違反の個数を最少化するものを選ぶ制約違反最少化 (min-conflict) ヒューリスクである。制約違反最少化ヒューリスティックを用いたアルゴリズムを図 5.8 に、8 クイーン問題に適用した過程を図 5.9 に、評価結果を図 5.5 の最後の列に示す。

制約違反最少化ヒューリスティックは、多くの制約充足問題に適切な初期制約充足化ヒューリスティックは、多くの制約充足問題に適切な初期制約充足化ヒューリスティックを用いる飛越し後戻り法は、幾合に基づく飛越し後戻り (conflict-directed backjumping)



² 局所探索は、目的関数をもつ制約充足問題に適用可能なように用いることができる。この場合、任意の山登り型および焼きなまし型のテクニックを、目的関数を最適化するように用いることができる。

```

function MIN-CONFLICTS(csp, max-steps) returns a solution or failure
inputs: csp, a constraint satisfaction problem
        max-steps, the number of steps allowed before giving up
current ← an initial complete assignment for csp
for i = 1 to max-steps do
    if current is a solution for csp then return current
    var ← a randomly chosen, conflicted variable from VARIABLES[csp]
    value ← the value v for var that minimizes CONFLICTS(var, v, current, csp)
    set var = value in current
return failure

```

図 5.8 制約充足問題を局所探索で解くための Min-Conflicts アルゴリズム。初期状態はランダム、もしくは各変数に関して、順に他の変数との制約違反を最小化する値を割り当てる欲張り法で得られる。CONFLICTS 関数は、現在の他の変数への値の割当てのことで、ある値に関する制約違反の個数をカウントする。

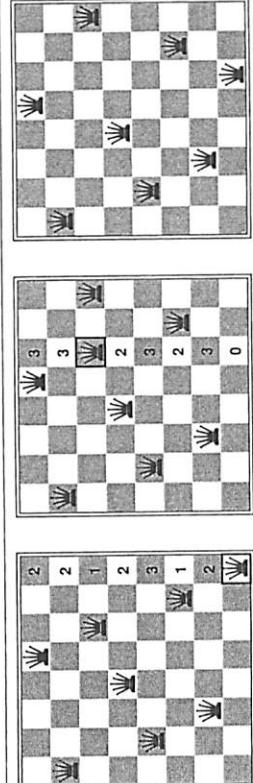


図 5.9 制約違反最少化ヒューリスティックを用いた、2ステップで得られた解。各段階では、ある列に関してクイーンの位置を変更する。制約違反の個数（この場合は、他のクイーンの数）を各位置に示す。アルゴリズムは制約違反を最少化する位置に移動する。タイアラームはランダムに行われる。

状態が与えられた場合に、驚くほどに有効である。図 5.5 の最後の列に、制約違反最少化ヒューリスティックを用いた局所探索アルゴリズムの性能を示す。さらに驚くべきことは、初期状態を生成する時間を考えなれば、制約違反最少化ヒューリスティックを用いた局所探索アルゴリズムが n クイーン問題を解く時間は、ほとんど問題のサイズに依存しない。このアルゴリズムは、100 万クイーン問題でも、平均して（初期配置から）50 ステップで解くことができる。この注目すべき結果を契機として、局所探索および、7 章で取り上げる簡単な問題と難しい問題の識別に関する、数多くの研究が 1990 年代に行われた。大雑把に言って、 n クイーン問題は、解が高い確率で状態空間中に分散して存在するため、局所探索にとって簡単な問題である。たとえば、このヒューリスティックはハッブル宇宙望遠鏡での観測のタスクのスケジューリング問題に用いられ、1 週間分の観測のタスクのスケジューリングに 3 週間（！）を要していたのが、10 分に削減することができた。

局所探索のもう一つの利点として、問題が変化するというオンラインの状況で利用可能となることがある。この性質はスケジューリング問題で特に重要である。1 週間分の航空会社のスケジュールは、数千の飛行機、数万人の人員に関する割当でとなる場合がある。ある空

港での悪天候により、スケジュールが実行不可能となつた場合には、なるべく少ない変更でスケジュールを修正したい。これは、現在のスケジュールを初期状態として、制約違反最少化ヒューリスティックで簡単にに対応できる。新しい制約を加えた後戻り型探索は、通常より多くの時間を要するし、現在のスケジュールを大きく変更した解を見つける可能性がある。

5.4 問題の構造

本節では、制約グラフで表現される問題の構造を用いて、高速に解を得る方法を示す。ここで示す方法の多くは非常に一般的なものであり、制約充足問題以外の問題領域、たとえば確率推論にも用いることができる。結局のところ、問題が複数の副問題に分割できない限り、実世界の問題に対応するのは不可能であろう。図 5.1 (b) を、問題を分割するという観点から見直してみよう。非常に明確なこととして、Tasmania は島であり、本土と陸続きではないことがある。³ Tasmania の色を決める問題と、本土の色を決める問題は独立な副問題 (independent subproblem) であることは直観的で明らかであり、任意の本土に関する副問題の解のと、任意の Tasmania に関する副問題の解の組合せは、全体の問題の解となる。副問題の独立性は、制約グラフで連結された要素 (connected component) をチェックすることによって確かめられる。各要素は制約充足問題の副問題 (connected component) をチェック S_i が CSP_i の解であれば、 $\bigcup_i S_i$ は $\bigcup_i CSP_i$ の解である。なぜこの性質が重要なのだろうか？以下の例を考えよう。 CSP_i が c 個の変数をもち、問題全体としては n 個の変数があるが、 c は一定であるとしよう。この場合 n/c 個の副問題が存在する。それそれは高々 d^c の努力で解けるので、全体の問題は高々 $O(d^c n/c)$ で解ける。これは n に対して線形である。問題の分割なしには、全体の問題を解く努力は $O(d^n)$ であり、これは n に対して指数的である。具体例で考えると、 $n=80$ のブール制約充足問題を $c=20$ の4つの副問題に分割できれば、最悪ケースの計算量は、宇宙の寿命から 1 秒未満に短縮できる。

完全に独立な副問題は、存在すればばらしいが、実際ににはめったにない。多くの場合、副問題は連結されている。連結された副問題の最も単純な場合として、制約グラフが木 (tree) となる場合がある。制約グラフが木であることは、任意の二つの節点を結ぶ経路があるが、一つであることを意味する。図 5.10 (a) に概念的な例を示す。⁴ 以下の性質が示される。任意の木構造の制約充足問題は変数の個数に対する線形時間で解ける。このアルゴリズムは以下のステップからなる：

1. 任意の変数を根節点として選び、根から葉まで、各変数に対して、その親ノードが上位となるように変数を順位づける（図 5.10 (b) 参照）。この順序を X_1, \dots, X_n とする。根節点以外の変数は、唯一の親節点をもつ。
2. n から順に、 2 までの j に関して、 X_i を X_j の親として、 (X_i, X_j) に関するアーチ無矛盾性を達成する。必要ならば $DOMAIN[X_i]$ から値を取り除く。

³ 注意深い地図作成者、もしくは郷土愛の強い Tasmania 人は、Tasmania が本土のある州の一部であるという誤解を避けるために、本土の近くの島は異なる色に塗るべきであると主張するかもしない。

⁴ 狙念ながら木構造の地図をもつ地域は世界中にごくわずかしかない。例外として、Sulawesi (インドネシア中部) の島がある。

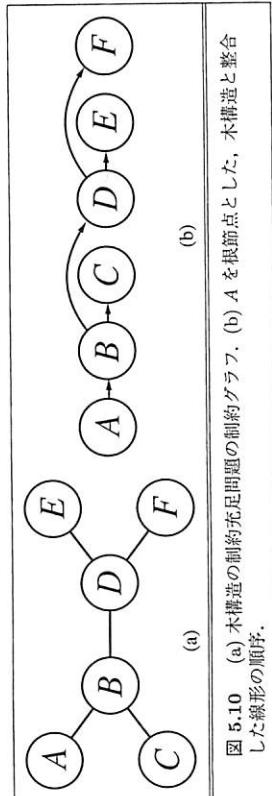


図 5.10 (a) 木構造の制約充足問題の制約グラフ. (b) A を根節点とした、木構造と整合した綱形の順序.

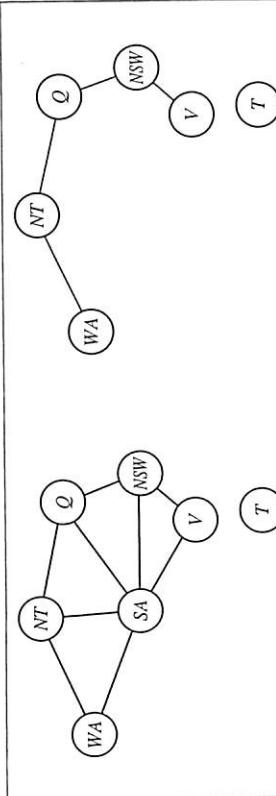


図 5.11 (a) 図 5.1 に示したものとの制約グラフ. (b) SA を除いたあとの制約グラフ.

3. 1 から順に n まで j に関する、変数 X_j に、親節点 X_i の値と無矛盾な値を割り当てる.

このアルゴリズムには注意すべき二つのポイントがある。まず、ステップ 2 のあと、制約充足問題は与えられた順序に関してアーケ無矛盾である。そのため、ステップ 3 での値の割当てでは後戻りは生じない（148 ページの k 無矛盾性に関する議論を参照）。次に、ステップ 2 で、アーケ無矛盾性を逆順で適用しているために、ある変数の値を取り除くことが、すでに達成されたアーケ無矛盾性を壊してしまうことはない。上記のアルゴリズムは全体で $O(nd^2)$ で実行される。

さて、木に関して効率的なアルゴリズムが得られたところで、より一般的な制約グラフを、なんとか木に帰着させることを考えよう。基本的には、節点を取り除く方法と、節点をまとめる方法の 2 種類が考えられる。

最初のアプローチでは、いくつかの変数に値を割り当てて、残りの変数が木構造となるようになる。図 5.11(a) に再掲するオーストラリアの制約グラフを考えよう。South Australia を取り除くと、(b) に示すようにグラフは木となる。幸運にも、この操作は（実際の陸地ではなく、グラフ上で）、以下のように実現できる。すなわち、 SA の値を決定し、その値と矛盾する値を、残りの変数の領域から取り除けばよい。

SA および SA に関する制約が取り除かれたあとの制約充足問題は、 SA に選ばれた値との制約を満足する（この方法は二項制約についてうまくいくが、高次の制約に関してはより複雑な処理が必要である）。よって、残りの木に関しては上記のアルゴリズムで解いて、全体の問題の解を得ることができる。もちろん、（地図の色塗り問題ではなく）一般

の問題では、 SA の値の割当てが不適切で、解が得られない場合があり、その場合は他の値を試してみる必要がある。全体のアルゴリズムは以下のとおりである：

1. $\text{VARIABLES}[csp]$ から部分集合 S を、 S を取り除いた残りの制約グラフが木となるよう選ぶ。 S は閉路切断集合 (cycle cutset) とよばれる。
2. S に含まれる変数の、 S 中の制約をすべて満足する割当てに対して以下を実行する。
 - (a) S の割当てと矛盾する値を領域から取り除く。
 - (b) 残りの制約充足問題が解をもてば、この解と S への割当てを返す。

閉路切断集合の大きさが c であれば、アルゴリズム全体としての実行時間は $O(d^c \cdot (n-c)d^2)$ となる。もしグラフが“ほとんど木”であれば c は小さく、純粹な後戻り法を用いる場合と比較して、計算量の削減の割合が膨大となる。しかしながら、最悪の場合 c の大きさは $(n-2)$ となる。また、最小の大きさの閉路切断集合を求めるることは NP 完全であるが、いくつかの効率的な近似アルゴリズムが知られている。このような手法は切断集合の条件付け (cutset conditioning) とよばれ、確率推論への適用に関して 14 章でも一度取り上げられる。

二番目のアプローチは、木分解 (tree decomposition) とよばれる、制約グラフを連結された副問題に分割する手法に基づくものである。各副問題は独立に解かれ、これらの副問題の解が合成される。多くの分割統治型のアルゴリズムと同様、この方法はどの副問題も大きくなりすぎない場合に有効である。図 5.12 は地図の色塗り問題を、五つの副問題に分割する木分解の例を示す。木分解は以下の条件を満足する必要がある：

- もとの問題の変数は、少なくとも一つの副問題に現れる。
 - もとの問題で二つの変数間に制約があれば、この二つの変数は、少なくとも一つの副問題に (変数間の制約も含めて) ともに現れる。
 - もし、ある変数が二つの副問題に現れるなら、この変数は、二つの副問題を結ぶ絆路中の、すべての副問題に現れる。
- 最初の二つの条件は、分解された問題中に、もとの問題の変数と制約が漏れなく表現されていることを保証する。三番目の条件はや人為的にみえるが、同じ変数には同じ値が割り当てられることを保証するためのものである。すなわち、リンクで結ばれた副問題間に同じ変数に同じ値が割り当てられることが保証される。たとえば、 SA は図 5.12 の連結された四つの副問題のすべてに現れている。図 5.12 で、上記の条件が満足されており、正しい木分解となっていることが確かめられるであろう。
- 各副問題を独立に解き、すべての解を得る。もし、いずれかの副問題に解がなければ、もとの問題にも解はない。すべての副問題で解が得られたら、以下のようにもとの問題の解を得る。各副問題を、領域が副問題の解であるような“巨大変数”と考える。たとえば、図 5.12 の色塗り問題の、最も左の副問題は三つの変数からなり、六つの解、たとえば $\{WA = red, SA = blue, NT = green\}$ をもつ。次に、前述の木に対する効率的なアルゴリズムを用いて、巨大変数からなる制約充足問題を解く。副問題間の制約は、単に同じ変数には同じ値が割り当てられていることを要求する。たとえば、最初の副問題の解 $\{WA = red, SA = blue, NT = green\}$ との制約を満たす、二番目の副問題の解 $\{SA = blue, NT = green, Q = red\}$ のみである。

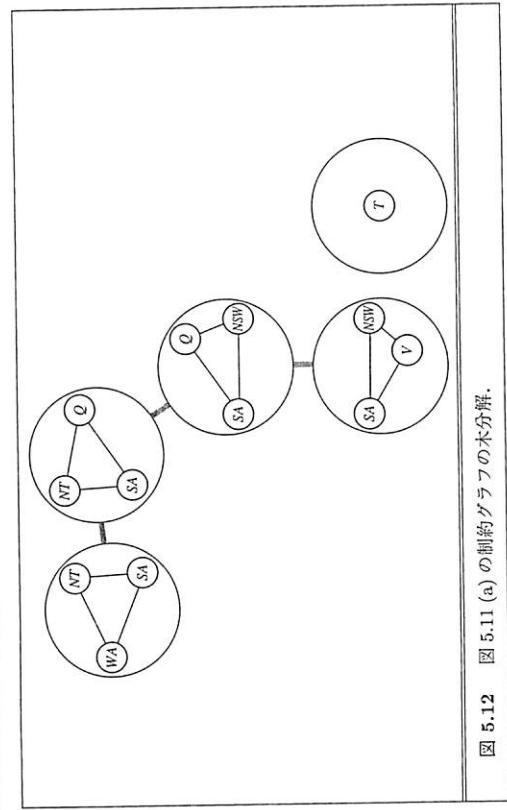


図 5.11 (a) の制約グラフの木分解.

与えられた制約グラフに対応する木分解は多数存在する。良い木分解を選ぶ基準は、木の幅をなるべく小さくすることである。ある木分解に対して、得られた木の幅 (tree width) を、最大の副問題のサイズから 1 減らした値と定義する。また、グラフに関する木の幅を、すべての可能な木分解で得られる木の幅の最小値と定義する。もしグラフが、幅 w の木分解をもち、その木分解を与えられれば、問題は $O(nd^{w+1})$ で解ける。よって、グラフの木の幅が限定された制約充足問題は多項式時間で解ける。残念ながら、最小の木の幅をもつ木分解を見つけることは NP 完全であるが、実用的にうまく働くヒューリスティックな方法が提案されている。

5.5 まとめ

- 制約充足問題 (CSP) は、変数と制約からなる。多くの実世界の重要な問題が制約充足問題として表現することができる。制約充足問題の構造は制約グラフで表現できる。
- 深さ優先探索の一種である後戻り探索は制約充足問題を解くために一般的に用いられる。
- 最少残存値および次数ヒューリスティックは、後戻り探索において、次に値を割り当てる変数を選択するために用いられる、領域に依存しないヒューリスティックである。
- 最少制約値ヒューリスティックは、値の順序づけのために用いられる。
- 後戻り探索アルゴリズムにおいて、構成した部分的な割当ての影響を伝播させるごとに、問題の分歧数を大幅に削減することができます。前向きチェックはこれを実現する簡単な方法である。アーケ無矛盾性を達成することは、より強力ではあるが、実行により大きなコストを要する。
- 後戻りは、ある変数に関して正当な値が存在しない場合に生じる。競合に基づく飛

- 越し後戻りは、問題の原因に対して直接戻りを実行する。
- 制約違反最少化ヒューリスティックを用いた局所探索は制約充足問題に適用され、大きな成功を収めている。
- 制約充足問題を解くことで解くことができる。切断集合の条件づけを用いて、一般の制約充足問題を木構造に変換することができる。この方法は、小さい切断集合が見つけられれば効率的である。木分解のテクニックは、制約充足問題を副問題で構成される木に変換し、制約グラフの木の幅が小さい場合には有効である。

文献と歴史ノート

制約充足に関する最初の仕事は、主に数値制約に関するものであった。整数値に関する方程式制約は 17 世紀のインドの数学者 Brahmagupta によって研究された。これらは、ギリシャの数学者 Diophantus (c. 200-284) にちなんで、ディオファントスの方程式 (Diophantine equation) とよばれる。実際、Diophantus は正の有理数の領域を考えていた。線形方程式を変数を消去することにより系統的に解く方法は Gauss (1829) で研究されている。線形不等式制約の解に関する研究は Fourier (1827) までさかのぼる。

有限領域の制約充足問題も長い歴史をもっている。たとえば、(地図の色塗り問題を特殊な場合として含む) グラフの色塗り (graph coloring) 問題は、古くから数学で扱われている問題である。Biggs, Lloyd, and Wilson (1986)によれば、4 色問題の予想 (任意の平面グラフは 4 以下の色で塗り分けられる) は、de Morgan の学生であった Francis Guthrie によって 1852 年に最初に示された。この予想に関する証明したとの主張が発表されてきたが、実際にには正しく証明されておらず、ごく最近に、Appel and Haken (1977) によって、計算機の助けを借りて証明された。

特定のクラスの制約充足問題は、計算機科学の歴史の随所に現れている。最も影響力のある、一般的なクラスの問題として制約充足問題を見いただしたのは Ugo Montanari (1974) である。高次の制約を、補助変数を用いて二項制約に変換する方法 (練習問題 5.11 参照) の起源は 19 世紀の論理学者 Charles Sanders Peirce によるものである。このアイデアは、あたた初期の研究の一例として、SKETCHPAD システム (Sutherland, 1963) がある。これは图形に跨る幾何制約を解くものであり、近年の描画プログラムと CAD ツールの先駆けである。一般的なクラスの問題として制約充足問題を見いただしたのは Dechter (1990) である。このアイデアは、Dechter (1990) により、制約充足問題の分野に紹介され、Bacchus and van Beek (1998) によって精緻化された。選好を含む制約充足問題は最適化の分野において広く研究が行われている。選好の導入による制約充足問題の拡張に関する Bistarelli, Montanari, and Rossi (1997) を参照のこと。バケット消去アルゴリズム (Dechter, 1999) は最適化問題に対して最も適用可能である。

制約充足のための後戻り探索は Bitner and Reingold (1975) によるが、基本的なアルゴリズムは彼らによれば 19 世紀までさかのぼる。Bitner and Reingold は、彼らが最大制約変数ヒューリスティックとよんでいる、MRV ヒューリスティックも提唱している。Brelaz (1979) は次数ヒューリスティックを、MRV ヒューリスティックのタイブレックのために用いた。得られたアルゴリズムは非常に単純であるにもかかわらず、現在にいたるまで、任意のグラフの k 色塗り問題を解く、最も効率的なアルゴリズムである。Haralick and Elliot (1980)

は最少制約値ヒューリスティックを提案した。

制約伝播方式は, Waltz (1975)による計算機視覚における多面体の線分のラベル付け問題での成功により注目された。Waltzは,多くの問題において,制約伝播により後戻りが全く不要となることを示した。Montanari (1974)は制約ネットワークと経路無矛盾性の概念を導入した。Alan Mackworth (1977)はアーケ無矛盾性を達成するAC-3アルゴリズム,および後戻り探索に対して,あるレベルの無矛盾性を達成するアルゴリズムAC-4はMohr and Henderson (1986)によって開発された。Mackworthの論文が發表されてからすぐあとに,様々な研究者により,無矛盾性を達成するためのコストと,探索の削減量のトレードオフに関する実験が始められた。Haralick and Elliot (1980)では,McGregor (1979)による,最低限の前向きチェックが良いと結論したが,一方, Gaschnig (1979)は,のちにSabin and Freuder (1994)によってMACと名づけられた。値が決定されるたびに,完全なアーケ無矛盾性を達成するアルゴリズムが良いと結論した。後の論文は,より難しい制約充足問題に関しては,完全なアーケ無矛盾性を達成することも割に合うということのもつともらしい証拠を示している。Freuder (1978, 1982)は k 無矛盾性の概念を示し,この概念と制約充足問題を解く標準化されたアーケ無矛盾性アルゴリズムを議論した。Apt (1999)は,制約伝播アルゴリズムを解析する,汎用のアルゴリズムの枠組みを示した。

高次制約を扱う特殊な方法は,主に制約論理プログラミング(constraint logic programming)の分野において開発が行われてきた。Marriott and Stuckey (1998)はこの分野の研究をよく網羅している。Allaliif制約はRegin (1994)で研究されている。境界制約はVan Hentenryck, Saraswat, and Deville (1998)によつて制約論理プログラミングに導入された。

基本的な飛越し後戻り法は,John Gaschnig (1977, 1979)によるものである。Kondrak and van Beek (1997)は,このアルゴリズムは本質的に前向きチェックに含まれることを示した。競合に基づく飛越し後戻り法はProsser (1993)により考案された。最も一般的で強力な知的後戻り法は,実は非常に早くから,Stallman and Sussman (1977)において用いられていた。彼らの依存関係に基づく後戻り(dependency-directed backtracking)というアイデアは,10.8節で示される真偽値管理システム(truth maintenance system)(Doye, 1979)へと発展した。これらの二つの分野の関係はde Kleer (1989)で分析された。

StallmanとSussmanの研究では,探索の間に得られた途中結果を記録し,あとで利用するという制約記録(constraint recording)というアイデアも導入されていた。このアイデアは,Dechter (1990a)により形式化され,後戻り探索に導入された,後向きマーキング(packmarking)(Gaschnig, 1979)は,矛盾する/しない値の割当てのペアを記録し,制約の再チェックを避ける非常に単純な方法である。後向きマーキングは競合ベースの飛越し後戻りと組み合わせることが可能で,Kondrak and van Beek (1997)は,これらのアイデアを単独で用いた場合のアルゴリズムをおそらく包含するであろう。ハイブリッド型のアルゴリズムを提案している。動的後戻り(dynamic backtracking)(Ginsberg, 1993)はある変数よりも他の変数の割当てに開いて,今回の変更によって矛盾が生じない限り,なるべく多くの割当てを保存しようとする方法である。

制約充足問題における局所探索は,スケジューリング問題で広く用いられている焼きなまし法(simulated annealing) (4章参照)に関するKirkpatrick, Gelatt, and Vecchi (1983)

の研究から注目されるようになった。制約違反最少化ヒューリスティックは,Gu (1989)により最初に用いられ,Minton, Johnston, Phillips, and Laird (1992)で独立に提案された。Sosic and Gu (1994)は,このヒューリスティックにより300万クイーン問題が1分以内で解けることを示した。 n クイーン問題における,制約違反最少化ヒューリスティックを用いた局所探索の驚くべき成功は,“簡単な”問題と“難しい”問題の本質,およびそのような問題がどのように分布しているかの再評価を引き起した。Peter Cheeseman et al. (1991)はランダムに生成された制約充足問題の難しさを調査し,このような問題のほとんどは,非常に簡単であるか,もしくは解がないかのどちらかであることを発見した。問題の生成器に設定されたパラメータが,ちょうど半分の問題が解けて半分の問題が解けないような,非常に狭い範囲に設定された場合のみ,“難しい”問題のインスタンスが得られる。この現象に関しては7章で再度取り上げる。

制約充足問題において,グラフの構造と複雑さを結びつける研究はFreuder (1985)が起源であり,アーケ無矛盾な木における探索は後戻りなしに実行できることを示した。閉路をもたないハイバーグラフに拡張された同様な結果は,データベースの研究コミュニティで得られた(Beeri, Fagin, Maier, and Yannakakis, 1983)。これらの論文が發表されてから,制約充足問題を解く複雑さと,制約グラフの構造について,より一般的な結果について,Robertson and Seymour (1986)によって導入された。Dechter and Pearl (1987, 1989)は,Freuderの研究に基づき,(彼女らは説導された幅(induced width)とよんでいる)同じ概念を制約充足問題に導入し,5.4節で紹介した木分解方法を開発した。この研究,およびデータベース理論での結果を利用して,Gottlob, Leone, and Scarcello (1999a, 1999b)は超木幅(hypertree width)とよばれる,超グラフとしての特徴に基づく概念を導入した。超木幅が w である任意の制約充足問題は $O(n^{w+1} \log n)$ なる時間で解けることを示したのに加えて,超木幅は,従来定義してきた“幅”という尺度を含むこと,すなわち,他の尺度の幅では上界値が定まらないのに,超木幅の上界値は定まる場合があることを示した。制約充足問題に関する技術的良好なサーベイとして,Kumar (1992),Dechter and Frost (1999),およびBartak (2001)があり,人工知能の百科事典の項目としてDechter (1992)とMackworth (1992)がある。Pearson and Jeavons (1997)は,効率的なアルゴリズムが存在する制約充足問題のクラスに関するサーベイであり,構造的な分解方法と,領域および制約の満たす性質から導かれるものの両方をカバーしている。Kondrak and van Beek (1997)は後戻り探索アルゴリズムの解析的なサーベイで,Bacchus and van Beek (1998)により実験的なサーベイである。Tsang (1993)およびMarriott and Stuckey (1998)による教科書は,本章で取り上げた範囲よりも,はるかに深い内容を扱っている。Freder and Mackworth (1994)によって編集された論文集には,いくつかの興味深いアプリケーションが示されている。制約充足に関する論文は定期的に人工知能のジャーナル Artificial Intelligence,および専門の論文誌 Constraints に発表されている。この分野の主要な国際会議として,制約プログラミングの原理と実用に関する国際会議(International Conference on Principles and Practice of Constraint Programming: CP)がある。

練習問題

5.9 木構造の制約充足問題で、AC-3アルゴリズムを実行した場合の最悪ケースの複雑度はどうなるか？

- 5.10 AC-3は、 X_i の領域から値が取り除かれると、必ず (X_k, X_i) をキーに戻すが、 X_k のそれぞれの値が、 X_i に残された複数の値と無矛盾である場合には、必ずしも (X_k, X_i) をキーに戻す必要はない、すべてのアーケ (X_k, X_i) に対して、 X_k のそれぞれの値に関する記録することにする。このような数値を効率的に修正する方法を示し、その結果、無矛盾性を達成する合計時間が $O(n^2 d^2)$ となることを示せ。

- 5.11 补助変数を用いて、單一の三項制約、たとえば“ $A + B = C^n$ ”を、三つの二項制約に変換せよ。領域は有限であることを仮定して良い（ヒント：もともと存在する変数の組合せを値とする補助変数を考えよ。また、“ X は組合せ Y の最初の要素である”といった制約を考えよ）。次に、同様な方法により、四つ以上の変数の関連する制約を二項制約に変換できることを示せ。最後に、変数の領域を変更することにより、単項制約を除去で能く変換できることを示せ。これにより、任意の制約充足問題は二項制約のみをもつ制約充足問題に变换できることが示される。

- 5.12 変数の個数が n の制約充足問題で、グラフの閉路切断集合が k 個以下の節点のみを含むことがわかっているとする。実行時間が $O(n^k)$ 以上とならない、簡単な閉路切断集合を求めるアルゴリズムを示せ。切断集合のサイズに対して多項式時間の、最小の閉路切断集合を求める近似アルゴリズムに関する研究を調査せよ。そのようなアルゴリズムの存在により、閉路切断集合を用いるアルゴリズムは実用的になるか？

- 5.13 以下の論理パズルを考える：五つの家があり、それぞれは異なる色で、異なる国籍の人が住んでいて、それぞれ異なる銘柄のタバコ、飲物、ベットを好む。以下の事実から、次の質問に答へよ。“シマウマはどの家に住んでいるか、また、水を好む人はどの家に住んでいるか？”

英国人は赤い家に住んでいる。
スペイン人は犬を飼う。

- ノルウェー人は一番左の家に住む。
黄色い家に住む人はケールズを吸う。
チエスター・フィールズを吸う人は、狐を飼う人の隣に住む。
ノルウェー人は青い家の隣に住む。
ワインストンを吸う人はカツツミを飼う。
ラッキーストライクを吸う人はオレンジジュースを飲む。
ウクライナ人はお茶を飲む。
日本人はハーフメントを吸う。
緑の家の人はコーヒーを飲む。
緑の家は、アイボリーの家のすぐ右にある。
中央の家の人はミルクを飲む。

- この問題を制約充足問題として定式化する場合の、異なる表現方法について議論せよ。ある表現よりも別の表現のほうが望ましいとする理由があるか？
- 5.14 図5.1の問題で、AC-3アルゴリズムを用いて、アーケ無矛盾性により、部分的な値の割当で $\{WA = red, V = blue\}$ が矛盾であることが示せることが確かめよ。

⁵ Ginsberg, Frank, Halpin, and Torrance (1990)は、クロスワードパズルを解く方法について議論している。Littman et al. (1999)は、より困難なクロスワードパズルを解く方法を取り組んでいる。

- 5.1 自分の言葉で、以下の用語を定義せよ：制約充足問題、制約、後戻り探索、アーケ無矛盾性、飛越し後戻り、制約違反最小化。
- 5.2 図5.1の地図の色塗り問題の解の個数はいくつか？
- 5.3なぜ、制約に関しては最も強く制約された変数を選ぶのが良く、値に関しては最も弱く制約された値を選ぶのが良いのかを説明せよ。

- 5.4 クロスワードパズルの問題を（解くのではなく）作ることを考える。⁵ すなわち、問題は、格子に対する書いた言葉を割り当てることがある。空白と塗り潰されたマス目からなる格子、（辞書などの）言葉のリストが与えられており、目的はリストの部分集合で空白のマス目を埋めることである。この問題を以下の二つの方法で正確に定式化せよ。
- a. 一般的な探索問題として定式化する。適切な探索アルゴリズムを選び、必要なならばヒューリスティック閾数を定義する。空白に対して、一度に一つの文字を割り当てるのと、一つの言葉を割り当てるのどちらが良いか？
- b. 制約充足問題として定式化せよ。変数は、言葉に対応させるべきか、それとも文字に対応させるべきか？

- 5.5 以下の問題に関して、制約充足問題として正確な定式化を示せ。

- a. 四角形に関する間取りの設計（floor planning）：大きな四角形の中に、複数の小さな四角形を重ならないように配置する。
- b. 講義のスケジュール（class scheduling）：一定の数の教授、講義室、講義があり、講義に対して可能な時間枠が与えられる。各教授は教えることのできる講義の集合をもつ。

- 5.6 計算機を用いて、図5.2の覆面算を解いてみよ。後戻り法、前向きチェック、MRV、最少制約値ヒューリスティックを用いよ。

- 5.7 図5.5では、 n クイーン問題に関する様々なアルゴリズムの評価結果を示している。同じアルゴリズムを、以下の方法でランダムに生成された地図の色塗り問題で評価せよ。長さ1の正方形中に、 n 個の点を配置する。点 X をランダムに選び、以下の条件を満たす Y を選んで直線で結ぶ。 Y は X に最も近く、まだ X と結ばれておらず、 X と Y を結ぶ線は既存の線と交差しない。上記の処理を、線が加えなくなるまで繰り返す。可能な限り大きな n の範囲で、 $d = 3$ および $d = 4$ の両方に関して比較の表を作れ。結果についてコメントせよ。

- 5.8 図5.1の問題で、AC-3アルゴリズムを用いて、アーケ無矛盾性により、部分的な値の割当で $\{WA = red, V = blue\}$ が矛盾であることが示せることが確かめよ。