

アルゴリズムとデータ構造⑫

～ 難しい問題への対処 ～

鹿島久嗣

難しい問題への対処：

- 分枝限定法
- 貪欲法
- 局所探索

難しい問題：

でも、実用上、解かないといけないときはある...

- 非常に難しい問題：NP完全・困難

- おそらく多項式時間アルゴリズムが存在しない

- 例：巡回セールスマン問題

- 最短のハミルトン閉路を見つける

- しかし、多くの実用上重要な問題がこのクラスに属する

- どうしても解かなければならないときがある

難しい問題への対処法：

計算量や性能の保証はないが実用上有効な方法

- NP困難の時点で「理論的に」効率の良い解法は望めない
- いくつかの「実用上は」有用な方法がある：
 - 分枝限定法
 - 局所探索
 - ...

↔ 近似アルゴリズム：最適解の保証はないが、最適解からどの程度悪いかという保証があるようなアルゴリズム

分枝限定法：

場合分けと探索の打ち切りによって効率よく最適解を探索

- 分枝：問題を場合分けによって複数の部分問題にする
 - 部分問題の解のうち最良のものが元の問題の最適解
 - 場合分けは木構造によって表現できる
- 限定：元々の問題よりも解きやすい緩和問題を解き、これ以上場合分けしても見込みのない探索を打ち切る
 - 緩和問題：簡単に解けて、元の問題の解を大まかに見積り可能
 - 打ち切り：緩和問題の解と暫定的な最適解を比較して、以降の場合分けを打ち切る

巡回セールスマン問題： NP困難な最小化問題

- 辺 $(v_i, v_j) \in E$ に非負のコスト $c(v_i, v_j) \geq 0$ がついたグラフ
- 最小のコストのハミルトン閉路を求めるNP困難問題
- 最小化問題としての定式化：

$$\text{minimize}_{\{x_{i,j}\}_{(i,j) \in E}} \sum_{(i,j) \in E} c(v_i, v_j) x_{i,j}$$

$$\text{s. t. } x_{i,j} \in \{0,1\}$$

$\{x_{i,j} \mid x_{i,j} = 1\}$ がハミルトン閉路をなす

— $x_{i,j} \in \{0,1\}$ は辺 (i,j) を閉路に含むかどうかを指定

巡回セールスマン問題の緩和問題： 閉路の条件をなくせば貪欲法で解ける

- ハミルトン閉路の条件を外す：

$$\text{minimize}_{\{x_{i,j}\}_{(i,j) \in E}} \sum_{(i,j) \in E} c(v_i, v_j) x_{i,j}$$

$$\text{s. t. } x_{i,j} \in \{0,1\}, \sum_{(i,j) \in E} x_{i,j} = N (= \text{頂点数})$$

- 貪欲法：最も効果の高いものから順に解に加える
 - コストの小さい辺ら順に N 本を採用する
 - 得られる解はハミルトン閉路とは限らない（通常違う）
- 緩和した問題の解空間はもとの問題の解空間を含む

分枝操作：

緩和解のサイクルを切ることで場合分けを行う

- 得られた解が閉路でない場合は、どこかにサイクルがある
- サイクルが生成されないように条件を加える
 - サイクル上の辺のうちひとつを使えないようにする
 - サイクル長が L であれば、 L 通りの可能性
- $L = 3$ で e_1, e_2, e_3 の3辺からなるサイクルがあるとする：
 1. $e_1 = 0$ とした問題
 2. $e_1 = 1, e_2 = 0$ とした問題
 3. $e_1 = 1, e_2 = 1, e_3 = 0$ とした問題

分枝操作と暫定解：

場合分けを進めて探索を行い、暫定解を見つける

- 分枝の候補のうちひとつを選び、その緩和問題を解き：
 1. 部分問題の最適解（ハミルトン閉路）でなかった場合
 - その解にサイクルがある場合は、さらに分枝操作
 2. 得られた場合 or 解がない場合：その先の探索は打ち切り
- 暫定解：現在までの最適解
 - 分枝による探索は深さ優先で、一旦解を得ることを優先
 - ひとまず暫定解を得たら、以降はこれを基準に考える（暫定解のコストを T とする）

限定操作（枝刈り）：

暫定解より悪い緩和解は、それ以降の探索を打ち切る

- 深さ優先探索：部分問題の最適解が見つかったら、ひとつ前の分枝の次の場合分けに向かう
- 限定操作：緩和問題の解のコストが暫定解のコスト T 以上であった場合、そこから先の探索を打ち切る
 - 緩和問題のコストは常に真のコスト以下なので、今後その解から探索を進めても改善は望めない

分枝限定法の別の例： ナップサック問題

- ナップサックに詰められる品物の価値の合計は最大いくつ？
 - N 個の品物： i 番目の品物の重さ w_i 、価値は q_i
 - 合計 M までの重さの品物が詰められるナップサックがある
- 定式化：
$$\text{maximize}_{\{x_i\}_i} \sum_i q_i x_i \quad \text{s. t.} \quad \sum_i w_i x_i \leq M, x_i \in \{0,1\}$$
- 分枝操作： $x_i \in \{0,1\}$ のいくつかを固定
- 緩和：連続化 $x_i \in [0,1]$ により解の上界を与える
 - 連続緩和した問題は簡単に解ける（重いほうから詰める）

枝刈りの有効利用の例： データマイニングにおける頻出パターン発見

- 解の性質を用いた探索の打ち切り（枝刈り）はしばしば用いられるテクニック
- データマイニング：膨大なデータから有用な知見を発見
 - ーマーケットバスケット分析：データマイニングの応用のひとつ。
購買データを分析してマーケティングの知見を発見
- 頻出パターンマイニング：同時に購入される傾向のある商品の集合を発見する
 - ー例：「ビールとオムツ」、店内の商品配置、オンラインショッピングサイトの「おすすめ」

マーケットバスケット分析の例： 購買データからの頻出パターン発見

■ 購買データ（レシート）

客	購入した商品
1	ごはん、味噌汁、とんかつ
2	ごはん、味噌汁、とんかつ
3	ごはん、スープ、ハンバーグ
4	パン、スープ、ハンバーグ
5	パン、牛乳、とんかつ

■ 2 回以上現れる商品の組み合わせを見つける

- 1 つ： ごはん、パン、スープ、味噌汁、とんかつ、ハンバーグ
- 2 つ： {ごはん、味噌汁} {味噌汁、とんかつ} {ごはん、とんかつ} {スープ、ハンバーグ}
- 3 つ： {ごはん、味噌汁、とんかつ}

頻出パターン発見における課題：

アイテムの組み合わせが多く、すべてのチェックは困難

- 問題： K 回以上現れるアイテムの組み合わせを全て見つけよ
- アイテムが N 種あるとすると、 2^N 個の組み合わせをチェックする必要がある
 - 素朴にすべてをチェックするのは現実的ではない
- 全てをチェックすることなく、条件を満たす組み合わせをみれなく発見したい

頻出パターン発見の基本方針と観察： 小さい集合からチェック、見込みのない組み合わせを見切る

- 基本方針：組み合わせの数を徐々に増やしていく
 - アイテム 1 つ、アイテム 2 つの組み合わせ、アイテム 3 つの組み合わせ、...
- 重要な観察：
 - 出現回数が K 回未満のアイテムの組を含むアイテムの組の出現数は K 回未満
 - これを使って探索を打ち切ることができる

局所探索：

現在の解を少し修正してよりよい解に移動する

- 現在の解 \mathbf{x} の近傍を定義し、その中で現在よりもよい解 \mathbf{x}' に移動する
 - 連続最適化における勾配法：関数 $f(\mathbf{x})$ を最大化するために、現在の解 \mathbf{x} を $f(\mathbf{x} + \Delta\mathbf{x})$ が増加する方向に更新
- 離散的な最適化問題では、現在の解 \mathbf{x} の近傍が自明ではないため適切な近傍集合 $N(\mathbf{x})$ を定義する必要がある
 - 例：解が k ビット列であれば、いずれかを反転したもの（ k 通り）の集合を近傍とする

局所探索の方法： 山登り法、アニーリング、...

- 現在の解 \mathbf{x} から近傍のうちのひとつ $\mathbf{x}' \in N(\mathbf{x})$ に移動する
- 山登り法：近傍 $N(\mathbf{x})$ のうち、もっともよい解を \mathbf{x}' として採用
 - 局所解に陥る可能性が高い
- アニーリング（焼きなまし）：局所解を避けるための方法
 - 近傍 $N(\mathbf{x})$ のうち、解をひとつ \mathbf{x}' 取り出す
 - 解が改善するなら \mathbf{x}' を採用する
 - 解が悪くなる変更も、ある確率 $\left(e^{\frac{f(\mathbf{x}') - f(\mathbf{x})}{T}} \right)$ で採用
 - T は「温度」パラメータ；下げると解が悪化する変更を採用しない

近傍の定義： 巡回セールスマン問題の場合

- 現在の解（ハミルトン閉路）の辺 2 つを交差させて別の解をつくる
 - 現在のハミルトン閉路 x に属するふたつの辺 $(v_i, v_j), (v_k, v_l)$ を考える
 - $(v_i, v_j), (v_k, v_l)$ のかわりに $(v_i, v_k), (v_j, v_l)$ を辺にする（近傍のひとつ）
 - すべての近傍のうち、もっともコストが小さいものに移動
- たとえば分枝限定法でひとつ解が得られたときに使う