主専攻実験 第2回発表会

0-1整数計画ソルバーの作成

6/29 (Wed)

岡部 純弥

概要

- 0-1整数計画問題（連続ナップザック問題）を解くソルバーの実装
  - 実装にあたっては 分枝限定法 を用いた
- さまざまなソルバーでの実行時間比較
  - 全探索，自作ソルバー，gurobi

問題の定式化

定義

- 品物の数：n
- 品物 $i \quad (0 \leq i \leq n-1)$ の値段 ：$p(i)$
- 制約の数：m
- $m \times n$ 行列：r
- $1 \times m$ 行列：b

問題の定式化
最適化問題の定式化

目的関数

$$\text{Maximize} \sum_{j=0}^{n-1} p(j)x(j)$$

制約条件

$$\forall i \in \{k \mid k \in \mathbb{N}, 0 \leq k \leq m-1\}; \sum_{j=0}^{n-1} r(i,j)x(j) \leq b(i)$$

分枝限定法とは

組合せ最適化問題における厳密解を求めるためのアルゴリズムの一つ

分枝 操作と **限定** 操作を繰り返すことで高速に解を導出する

分枝：

元の問題を部分問題へと分解する

限定：

分枝操作で得た問題が **“解いても無駄”** と判断したらその問題は解かない

分枝限定法とは
もう少し厳密に．．

“解いても無駄か”の判断 <- 上界と下界の評価

上界：線形緩和

下界：Greedy法

実装の方針

1. 暫定値の初期化，問題の集合 queue を queue = {元の問題}　で初期化する
    1. 暫定値の初期化には 貪欲法 を用いる
2. queue に問題が存在すれば，問題を1つ取り出し “解いても無駄か” 判定する
    1. 問題が存在しなければ探索は終了
3. 取り出した部分問題の最適値が暫定解よりも良い場合は更新する
4. 取り出した部分問題をさらに部分問題に分解し，queue に追加する
    1.部分問題が存在しない場合は何も行わない

実装

- TDD（Test-Driven Development）で実装
  - eg. 最初は線形緩和問題の求解はソルバーで行った（-> リファクタリングで自前実装の単体法へ）
  - Pytest を用いてユニットテスト
- ソースコードやドキュメントは https://github.com/Okabe-Junya/MajorExperimentsA/tree/main/task2/src を参照

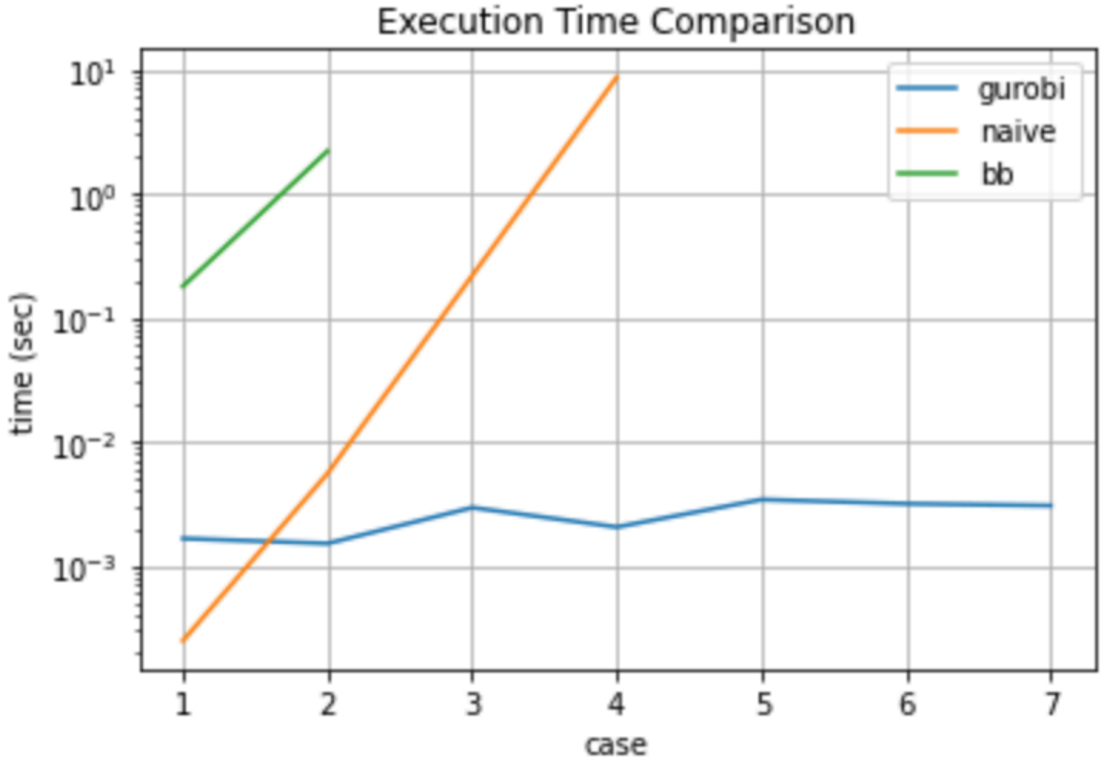
実装の工夫

- モジュール化 により設計，テストを行いやすくした
- 緩和問題を解く際に「自前実装」「ソルバー」を選択できるブール変数を用意
- 高速化（後述）

計算機実験
実行時間の比較

- 全探索，bb，Gurobiで比較実験
  - 上限10秒で打ち切り

全探索（指数オーダー）より性能が悪い



テストケースと実行時間の比較（縦軸は対数スケール）

計算機実験
反省点

- 限定操作がうまく機能していない
    - 末尾の品物から順に判定している
- 幅優先探索をしている
- そもそもPythonが（他の言語比較して）遅い

計算機実験
反省点1

価値	100	1	0	0	0	0	0	0
重さ	1	100	100	100	100	100	100	100

（例）重さ101以下で価値を最大にする品物の選び方は？（暫定値：100）

計算機実験
反省点1

価値	100	1	0	0	0	0	0	0
重さ	1	100	100	100	100	100	100	100

（例）重さ101以下で価値を最大にする品物の選び方は？（暫定値：100）

計算機実験
反省点1

価値	100	1	0	0	0	0	0	0
重さ	1	100	100	100	100	100	100	100

末尾の品物を選択して
分枝->限定

計算機実験
反省点1

価値	100	1	0	0	0	0	0	0
重さ	1	100	100	100	100	100	100	100

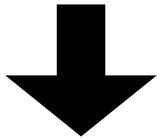
末尾から2番目の品物を
選択して分枝->限定

以降も同様に続くため限定操作がほとんど意味をなさない！！

計算機実験
改善1

価値	100	1	0	0	0	0	0	0
重さ	1	100	100	100	100	100	100	100

最適解に含まれそうな品物から分枝/限定操作を行いたい

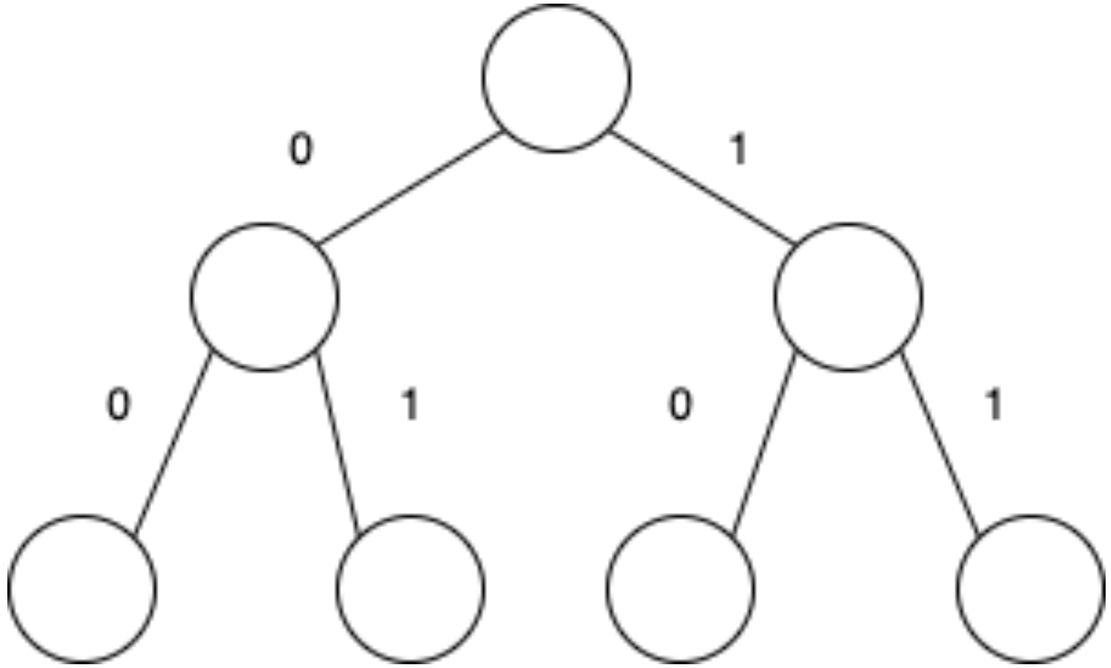


“価値” が高い順に分枝/限定操作を行う

計算機実験
反省点2

幅優先での探索

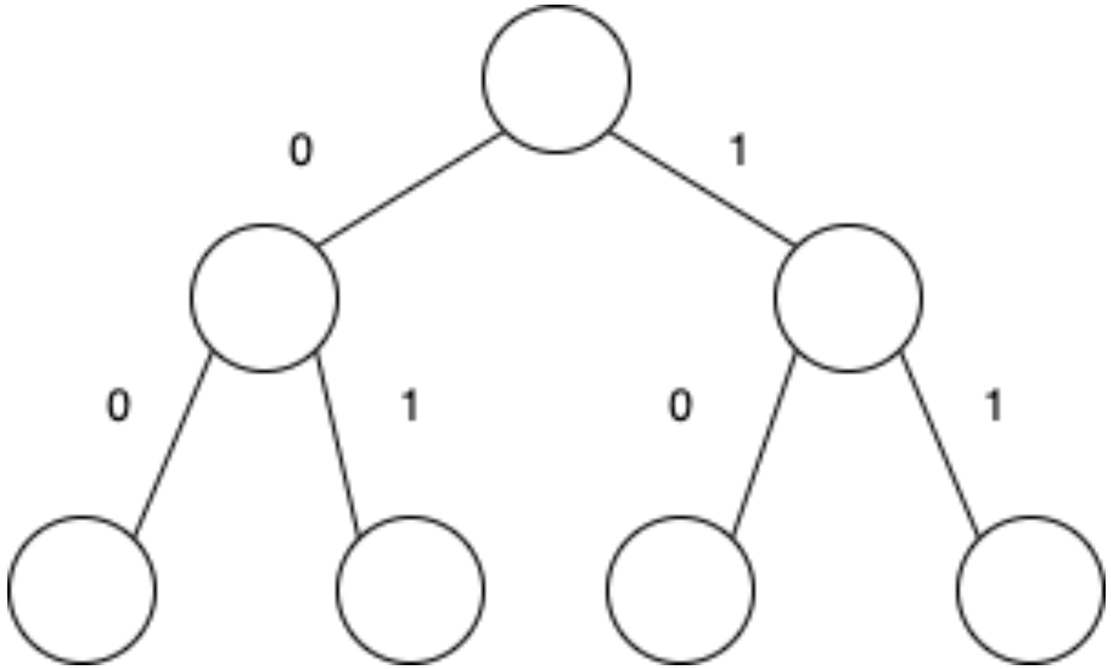
葉に辿り着くまでに時間がかかる



葉で下界の更新が行われるのでできるだけ早く葉を探索したい

計算機実験
改善2

Queue -> Stack で深さ優先に変更



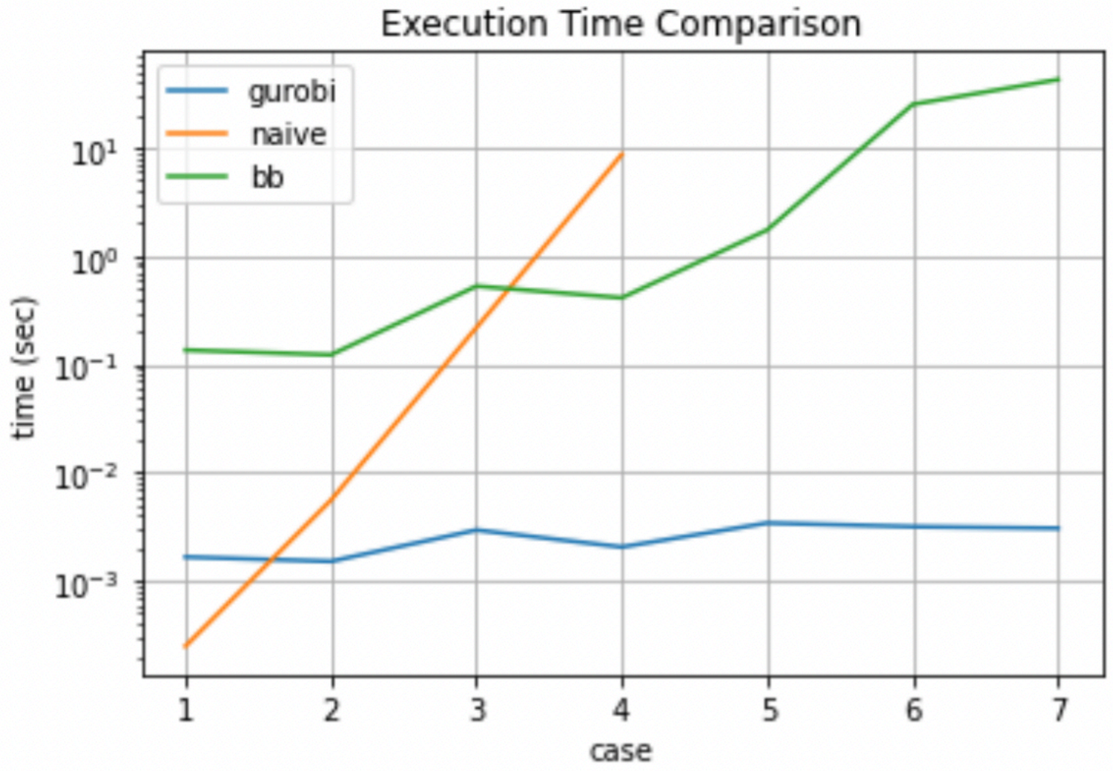
葉で下界の更新が行われるのでできるだけ早く葉を探索したい

計算機実験
改善

- 限定操作がうまく機能していない -> 価値の大きさ順にソート
  - 末尾の品物から順に判定している
- 幅優先探索をしている　-> 深さ優先探索
- そもそもPythonが（他の言語比較して）遅い -> Cythonの導入

計算機実験
改善

- ケース7 $(n = 50)$ でも40秒程度で実行可能



十分な高速化は実現できたが・・・

まとめ
できたこと

- 最適化ソルバーを実装した
  - 分枝限定法を用いた実装
- その高速化を行った
  - 今回のテストケースのサイズ $(n \leq 50)$ であれば1分以内には実行できた
  - それでも gurobi のような有償ソルバーには足元にも及ばない

まとめ
できなかったこと

- 並列処理
- 最良優先探索の実装
  - 評価値を与えてより最適解を求めやすい探索を行う
- 他言語での実装
  - C/C++, Go, Rust, Julia あたりで書き直せれば早くなりそう
    - 最適化を意識しながらコーディングをするのが苦手なので要検討