情報システム工学実験 3 数理計画法の実装

2022531033 関川謙人

2024 年 7 月 31 日

1　実験

1.1　使用ツール

今回は以下のツールを用いてカックロを解くプログラムを作成した。

- Python
- Python-mip

1.2　プログラム

プログラムの流れについて説明する。このプログラムは各マスについて 1〜9 の数値を制約に基づいて選択する形で解を求める。

1.2.1　変数

初めに三次元のブール配列を定義する。この三次元配列の奥行きは 9 とする。この三次元配列は各マスに対してどの数値を選択するかを表す。

1.2.2　制約

制約について説明する。設けた制約は以下の通りである。

制約 1 各マスにつき一つの数字しか選択できない
制約 2 斜め線の右上の数字とその右の連続した空白に入る数字の合計は一致する。
制約 3 斜め線の右下の数字とその下の連続した空白に入る数字の合計は一致する。
制約 4 連続した空白に同じ数字は入らない。

制約を設定するときは縦横すべてのマスを走査してデータによって処理を分けた。
斜め線であった場合の処理は以下の通り

1. 右上と右下の数値を記録する。
2. 縦の連続する空白の数を数える。
3. 制約 2 と制約 4 を設定する。
4. 横の連続する空白の数を数える。
5. 制約 3 と制約 4 を設定する。

設定部分のソースコードは以下の通り。

```
# 縦についての制約
# 制約4
for k in range(9):
  md += xsum((sol[y + i][x][k]) for i in range(1, rows_flag)) <= 1, 'cols1_' + str(y) + '_' +
```

```
    str(x) + '_' + str(k)
# 制約2
md += xsum((sol[y + i][x][k]) for i in range(1, rows_flag)) <= 1, 'cols1_' + str(y) + '_' + str(
  x) + '_' + str(k)

# 横についての制約
# 制約4
for k in range(9):
  md += xsum((sol[y][x + j][k]) for j in range(1, cols_flag)) <= 1, 'rows1_' + str(y) + '_' +
    str(x) + '_' + str(k)
# 制約3
md += xsum((sol[y][x + j][k] * (k+1)) for j in range(1, cols_flag) for k in range(9)) == int(
  latter), 'rows2_' + str(y) + '_' + str(x)
```

空白であった場合は制約 1 を設定する。

```
  # 制約1
  md += xsum(sol[y][x][k] for k in range(9)) == 1, 'sol1_' + str(y) + '_' + str(x)
```

2　実験結果

実験の結果、最終的なタイムは 175 秒であった。しかし、カックロを設計した時点では 200 秒であった。タイムを縮めるために以下のことを試みた。

- 条件文を変える
- 制約 1 をなくす

条件文を変えるというアプローチについては一定の効果があった。初めのプログラムでは以下のように空白の判定を「
もしくは x ではない」という条件で行ったが、「データが存在しない」という条件に変更した。

```
  # 初めのプログラムにおける空白の判定
  '\\' in data[y][x] or 'x' in data[y][x]
  # 変更後のプログラムにおける空白の判定
  not data[y][x]
```

この変更によってタイムは約 25 秒短縮された。

制約 1 をなくすというアプローチについては逆効果であった。制約 1 をなくすことでタイムが遅くなった。コンテストを解かせた際には 300 秒を超過してしまい、解を求めることができなかった。

また最後のタイム計測において Windows の WSL 環境の python3 では実行できたが、演習室の MAC の python3 では実行できなかった。この原因は不明であるが、python3 で実行せずに python の方を利用すると解決した。

3　考察

条件文の変更でタイムが短縮されたのは、「\ か x」の判定をおこなわなければならなかったところを「データが存在するのか」という基準に変更したことにある。すなわち判定が二回から一回になったことで計算量が減り、タイムが縮まったと考えられる。

制約 1 をなくしたことでタイムが遅くなってしまった理由は探索の数にある。制約 1 がなければ数値が決まっても探索をしようとする。この結果、制約をなくすことがかえって計算量を増やすことになったと考えられる。

python3 で実行できなかった具体的な理由については不明であるが、python3 に原因があったと仮定するならば、バージョンの問題であると考えられる。