Chess-Ant

リリース *0.0.9*

Akihiro Kuroiwa

2023 年 12 月 15 日

目次

第 1 章

Chess-Ant 紹介

Chess problems を MCTS Solver と Genetic Programming で解くシミュレーター

`chess_ant.py` は deap/examples/gp/ant.py のコードを改変しています。

1.1 準備

Ubuntu で：

```
sudo -H -s
apt install python3-pip
pip3 install -r requirements.txt
exit
```

或いは：

```
pip3 install chess
pip3 install deap
pip3 install mcts
pip3 install mcts-solver
```

或いは：

```
pip3 install chess-ant
```

- python-chess: a chess library for Python
- DEAP
- MCTS

- chess-ant
- chess-classification
- mcts-solver

1.2 使用法

サンプルの chess problems は pgn/ にあります。 Jerry は Forsyth-Edwards Notation (FEN) を貼り付けるのに便利です。

```
cd chess-ant/chess-ant/
git checkout -b test-run
python3 chess_ant.py --help
python3 chess_ant.py --auto --fen "7k/1Q6/8/8/5N2/1B6/8/3K4 w - - 0 1"
python3 chess_ant.py -a -c -p "my-pgn" -l1 -n100 -g10 -f "7k/1Q6/8/8/5N2/1B6/8/3K4 w - -␣
↪0 1"
```

chess-ant を PyPI からインストールした場合：

```
chess-ant --help
chess-ant -a -n100 -g5 -f "7r/8/8/8/7k/2q5/6P1/6NK b - - 0 1"
```

このコマンドは誤答を出力するでしょう。時間はかかりますが、以下のコマンドは正しく出力するでしょう。

```
chess-ant -a -n1000 -g5 -f "7r/8/8/8/7k/2q5/6P1/6NK b - - 0 1"
```

1.3 Chess-Classification

Version 0.0.1 の genPgn.py はウォルラス演算子を含むので、Python>=3.8 でしか動作しません。Simple Transformers をインストールする前に Pytorch をインストールして下さい。

```
sudo -H -s
pip3 install pandas
pip3 install simpletransformers
apt install stockfish
pip3 install chess-classification
exit
genPgn --help
genPgn -l 10 -t 1 -p "train-pgn" -f "3qkbnr/8/8/8/8/8/PPPPPPPP/RNBQKBNR w - - 0 1"
```

(次のページに続く)

(前のページからの続き)

```
cat train-pgn/train-*.pgn >> train-pgn/1.pgn
rm train-pgn/train-*.pgn
genPgn -l 10 -t 1 -p "train-pgn" -f "rnbqkbnr/pppppppp/8/8/8/8/8/3QKBNR w - - 0 1"
cat train-pgn/train-*.pgn >> train-pgn/2.pgn
rm train-pgn/train-*.pgn
genPgn -l 10 -t 1 -p "train-pgn" -f "4k3/pppppppp/8/8/8/8/PPPPPPPP/4K3 w - - 0 1"
cat train-pgn/train-*.pgn >> train-pgn/3.pgn
rm train-pgn/train-*.pgn
importPgn -p "train-pgn"
genPgn -l 10 -t 1 -p "eval-pgn" -f "3qkbnr/8/8/8/8/8/PPPPPPPP/RNBQKBNR w - - 0 1"
cat eval-pgn/train-*.pgn >> eval-pgn/1.pgn
rm eval-pgn/train-*.pgn
genPgn -l 10 -t 1 -p "eval-pgn" -f "rnbqkbnr/pppppppp/8/8/8/8/8/3QKBNR w - - 0 1"
cat eval-pgn/train-*.pgn >> eval-pgn/2.pgn
rm eval-pgn/train-*.pgn
genPgn -l 10 -t 1 -p "eval-pgn" -f "4k3/pppppppp/8/8/8/8/PPPPPPPP/4K3 w - - 0 1"
cat eval-pgn/train-*.pgn >> eval-pgn/3.pgn
rm eval-pgn/train-*.pgn
importPgn -p "eval-pgn"
```

```
from chess_classification.chess_classification import ChessClassification
classification = ChessClassification()
```

保存したモデルの読み込み：

```
classification = ChessClassification("local-path/your-outputs")
```

学習か再学習：

```
classification.train_and_eval("train-pgn/fen.json", "eval-pgn/fen.json")
```

動作確認：

```
my_fen = "7r/8/8/8/7k/2q5/6P1/6NK b - - 0 1"
classification.predict_fen(my_fen)
```

予測	ラベル
1-0	2
0-1	1
1/2-1/2	0

chess_ant.py との併用：

```
chess-ant -d -n100 -g5 -f "6rk/4pppp/8/8/3Q4/8/RB2PPPP/R6K w - - 0 1"
```

保存したモデルの読み込み：

```
chess-ant -d -n100 -g5 -f "6rk/4pppp/8/8/3Q4/8/RB2PPPP/R6K w - - 0 1" --model-outputs
↪"local-path/your-outputs"
```

- Simple Transformers
- Start Locally | PyTorch
- pandas
- Chess-Classification

課題:

- 遅い。
- 低い正答率。
- 並列化。
- 将棋のような他のボードゲームへの対応。
- Universal Chess Interface (UCI) への対応。
- Docstring.
- スパゲティ・コードを茹でる。

第 2 章

Chess-Ant 論文

author

Akihiro Kuroiwa

date

2021/12/25

abstract

DeepMind が開発した AlphaFold のように、 MCTS は bioinformatics や cheminformatics へ応用できる。Chess-Ant では AlphaZero に影響を受けつつも、それとは異なる方法を試す。MCTS Solver と Genetic Programming の組み合わせが実用に耐えうるか、その可能性を示すのが目標だ。

2.1 AlphaZero からの影響

AlphaGo Zero や AlphaZero で導入された Polynomial Upper Confidence Tree (PUCT) の変種は事前確率 $P(s,a)$ で定数 $C(s)$ を補完している。Chess-Ant では従来の Conventional Upper Confidence Tree (UCT) を用い、定数 $C(s)$ の調整を Genetic Programming (GP) に置き換える[*1]。$1/\sqrt{1}$ から $1/\sqrt{9}$ まで GP は状況に応じて定数 C の値を選ぶ。定数 C の値により、 `chess_ant.py` は積極的に調べたり、消極的な調べ方をする。

AlphaZero's PUCT[*2][*3][*4][*5]:

$$a_t = arg\,max_a(Q(s_t,a) + U(s_t,a))$$
$$U(s,a) = C(s)P(s,a)\frac{\sqrt{N(s)}}{1+N(s,a)}$$
$$C(s) = log\frac{1+N(s)+c_{base}}{c_{base}} + c_{init}$$

[*1] Cazenave, Tristan. “ Evolving Monte-Carlo Tree Search Algorithms. ” (2007).
[*2] AlphaZero: Shedding new light on chess, shogi, and Go
[*3] Silver, David et al. “ A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. ” Science 362 (2018): 1140 - 1144.
[*4] Foster, David. (2017). AlphaGo Zero Explained In One Diagram
[*5] Tadao Yamaoka ’ s diary

Deep neural network $(p, v) = f_\theta(s)$ はパラメータ θ により末端のノード s_L を評価する：

$$(p, v) = f_\theta(s_L)$$

初期化：

$$\begin{aligned} N(s_L, a) &= 0 \\ W(s_L, a) &= 0 \\ Q(s_L, a) &= 0 \\ P(s_L, a) &= p_a \end{aligned}$$

更新：

$$\begin{aligned} t &\leq L \\ N(s_t, a_t) &= N(s_t, a_t) + 1 \\ W(s_t, a_t) &= W(s_t, a_t) + v \\ Q(s_t, a_t) &= \frac{W(s_t, a_t)}{N(s_t, a_t)} \end{aligned}$$

	詳細
(s,a)	それぞれの state-action のペア
N(s)	親（ノード）の訪問数
N(s,a)	訪問数
W(s,a)	action-value の合計
Q(s,a)	action-value の平均
P(s,a)	s での a を選択する事前確率
C(s)	Exploration（と Exploitation）の割合
p	（Deep neural network により予測された）手を選ぶ確率のベクトル
v	（Deep neural network により予測された）スカラー値

Chess-Ant's UCT[*6][*7][*8][*9][*10]:

$$a_t = arg\,max_a(Q(s_t, a) + C_{gp}\sqrt{\frac{2lnN(s_t)}{N(s_t, a)}})$$

$$C_{gp} = \begin{cases} 1/\sqrt{1}, \\ 1/\sqrt{2}, \\ 1/\sqrt{3}, \\ 1/\sqrt{4}, \\ 1/\sqrt{5}, \\ 1/\sqrt{6}, \\ 1/\sqrt{7}, \\ 1/\sqrt{8}, \\ 1/\sqrt{9}, \end{cases} \quad \text{If the previously selected node state is under certain conditions}$$

判断する条件は以下の通り：

条件	詳細
if_improvement	前回よりも UCT が増加したら
if_shortcut	move_stack が前回よりも短くなったら
if_result	前回の rollout が勝ちか負けか引き分けか
if_pruning	UCT が無限大か無限小か、何れでもないか
if_is_check	前回選ばれた node で check だったか

また、GP は各 generation において、やり直すことができるので、訪問回数による検索初期の UCT の高評価を防げる。この論文のような効果を得られるかも知れない[*11]。

2.2 MCTS Solver

MCTS Solver[*12][*13] の導入により、高速化と誤答を減らす為の枝切りを行う。

論文の擬似コードを大幅に変更している理由は、継承クラスが参照する パッケージ のコードの書き方が異なるからだ。

[*6] Auer, Peter et al. “ Finite-time Analysis of the Multiarmed Bandit Problem. ” Machine Learning 47 (2004): 235-256.
[*7] Kocsis, Levente and Csaba Szepesvari. “ Bandit Based Monte-Carlo Planning. ” ECML (2006).
[*8] Swiechowski, Maciej et al. “ Monte Carlo Tree Search: A Review of Recent Modifications and Applications. ” ArXiv abs/2103.04931 (2021): n. pag.
[*9] Wikipedia contributors. "Monte Carlo tree search." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 18 Oct. 2021. Web. 25 Dec. 2021.
[*10] Wikipedia contributors. "モンテカルロ木探索." Wikipedia. Wikipedia, 8 Oct. 2021. Web. 25 Dec. 2021.
[*11] Imagawa, Takahisa and Tomoyuki Kaneko. “ Improvement of State ’ s Value Estimation for Monte Carlo Tree Search. ” (2017).
[*12] Winands, Mark & Björnsson, Yngvi & Saito, Jahn-Takeshi. (2008). Monte-Carlo Tree Search Solver. 25-36. 10.1007/978-3-540-87608-3_3.
[*13] Baier, Hendrik & Winands, Mark. (2015). MCTS-Minimax Hybrids. IEEE Transactions on Computational Intelligence and AI in Games. 7. 167-179. 10.1109/TCIAIG.2014.2366555.

さらに、Python では goto が無いので、余計なコードがある。擬似コードでは MCTS Solver 内部で全て完結するが、 chess_ant.py では _executeRound() に分けて書いてある。 mctsSolver() 内で rollout を実行し、枝切りなどの処理をし、rollout と同様に reward を出力し、 backpropogate() に入力する。

Negamax と同様に MCTS solver も再帰関数であり、停止条件が必要だ。

2.3 並列処理

mcts-solver 0.0.5 から、OpenAI の ChatGPT と Google Bard の助けを得てコードを変更し、並列処理[*14][*15] を行えるようにした。Tree 並列化は root 並列化に比べて、locks を用いなければならないので作業が難しい。これらの変更が正しく動作しているか確証が持てないので、今後大幅に変更するかも知れない。Context managers は便利だけれども、使用法を誤ると大変なことになる。

2.4 変更履歴

chess-classification と chem-classification は同じアルゴリズムを用いているので、同時期に同じ変更を施している。

chess-classification 0.0.4 で fen を token に区切ってデータセットを作成するようにした。その fen は 1 文字ずつではなく、一列ごとに区切ってある。

chess-ant 0.0.5 まで UCB1 の数式に誤りがあったので math.sqrt(2) を追加し、訂正した。mcts の作者が、mcts.py 内で explorationConstant = 1 / math.sqrt(2) を初期値としたのは、math.sqrt(2) を打ち消すためと思われる。新たに selectNodeEphemeralConstant という terminal を chess-ant 0.0.6 で追加し、ephemeral constant の代替とした。

Issues #658 によると、Deap は Python 3.10 で動作せず、version 3.11 で動作する。

~/.bashrc 或いは ~/.bash_aliases で：

```
# Python version
alias python3='/usr/bin/python3.11'
```

続いて **source** を実行する：

```
source ~/.bash_aliases
```

別の方法として、 **update-alternatives** がある。

[*14] Chaslot, Guillaume & Winands, Mark & Herik, H.. (2008). Parallel Monte-Carlo Tree Search. 60-71. 10.1007/978-3-540-87608-3_6.

[*15] Soejima, Yusuke & Kishimoto, Akihiro & Watanabe, Osamu. (2009). Root Parallelization of Monte Carlo Tree Search and Its Effectiveness in Computer Go.

```
update-alternatives --install /usr/bin/python3 python3 /usr/bin/python3.10 2
update-alternatives --install /usr/bin/python3 python3 /usr/bin/python3.11 1
update-alternatives --config python3
python3 --version
```

最優先に設定した元のヴァージョンに戻すには：

```
update-alternatives --auto python3
```

2022 年 12 月現在、gnome-terminal は Ubuntu 22.04.1 LTS の Python 3.11 では起動しないことを覚えておいて欲しい。

より一般的な方法は `venv` を用いる：

```
sudo apt install python3.11-venv
python3.11 -m venv ~/.venv3.11
source ~/.venv3.11/bin/activate
which pip3
```

終了するには：

```
deactivate
```

`chess-classification` 0.0.5 から、保存したモデルの読み込みが出来る。

学習時間を短縮するために、[*16] checkpoint が google/electra-small-discriminator[*17] の electra model に変更した。

私が用意した pgn ファイルは chess problems と言うよりも寧ろ tactics[*18][*19] に近いので、データセットは tactics から作成する方が効率が良い。正解率を高めるためには、モデルの学習とモデルの評価にそれぞれ 300 行以上の pgn データが必要だ。

2.5 開発予定

課題: 別のプロジェクトとして FEN の勝敗評価に Natural Language Processing (NLP) 用の deep learning[*20][*21] を導入する。

[*16] Rajapakse, Thilina. (2020). Battle of the Transformers: ELECTRA, BERT, RoBERTa, or XLNet
[*17] ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators
[*18] Download tactics database
[*19] Gorgonian's Chess Site
[*20] Savransky, Dmitriy. (2020). How to Use GPT-2 for Custom Data Generation. INTERSOG Inc.
[*21] Noever, David et al. “ The Chess Transformer: Mastering Play using Generative Language Models. ” arXiv: Artificial Intelligence (2020): n. pag.

genPgn.py は stockfish で自動対局をし、PGN files を出力する。ちなみに、Version 0.0.1 の genPgn.py はウォルラス演算子を含むので、Python>=3.8 でしか動作しない。 importPgn.py は PGN files からデータセットを作成する。 chess_classification.py は Simple Transformers による学習済みモデルを生成する。この学習済みモデルを用いて、AlphaZero のように chess_ant.py の rollout の代用とする予定だ。

2.6 参照

2.7 参考文献

- Home Page of John R. Koza
- Astro Teller | Technical Papers
- Lones, Michael. (2014). Genetic Programming: Memory, Loops and Modules. David Corne: Open Courseware.
- Alpha " Othello " Zero
- Czech, Johannes et al. " Monte-Carlo Graph Search for AlphaZero. " ArXiv abs/2012.11045 (2020): n. pag.
- Prasad, Aditya. (2018). Lessons From Implementing AlphaZero
- chess-alpha-zero
- Yao, Yao. (2018). API Python Chess: Distribution of Chess Wins based on random moves
- Stöckl, Andreas. (2018). Writing a chess program in one day
- Stöckl, Andreas. (2019). An incremental evaluation function and a test-suite for computer chess
- Stöckl, Andreas. (2019). Reconstructing chess positions
- Python Chess
- Fiekas, Niklas. (2015). An implementation of the Bratko-Kopec Test using python-chess
- Bratko-Kopec Test
- Kurt & Rolf Chess Homepage of Kurt Utzinger
- PGN Mentor
- Hart, Alex. (2011). Alpha Beta pruning on a Minimax tree in Python
- PythonChessAi
- easyAI
- Shrott, Ryan. (2017). Genetic Programming applied to AI Heuristic Optimization

- Alpha-Beta Pruning
- Hartikka, Lauri. (2017). A step-by-step guide to building a simple chess AI
- Simplified Evaluation Function
- Brynjulfsson, Erlingur. A Genetic Minimax Game-Playing Strategy
- Öberg, Viktor. “ EVOLUTIONARY AI IN BOARD GAMES : An evaluation of the performance of an evolutionary algorithm in two perfect information board games with low branching factor. ”(2015).
- Agapitos, Alexandros & Lucas, Simon. (2006). Learning Recursive Functions with Object Oriented Genetic Programming. 3905. 166-177. 10.1007/11729976_15.
- Yu, Tina and Christopher D. Clack. “ Recursion , Lambda Abstractions and Genetic Programming. ” .
- YouTube channel of David Beazley
- Welcome to AntWiki

第 3 章

py-chessboardjs

pywebview と chessboard.js を利用した Chess GUI です。主要ファイルは `start.py`, `js/my-script.js` と `index.html` です。

Chess-Ant は遅すぎてチェス・エンジンとして機能しません。更に、チェス・エンジンを呼び出す際の不具合もあります。実験目的として pgn ファイルを読み込み、chess problem を解かすことは出来ます。

3.1 インストール方法

まず pywebview と PyGObject のマニュアルを読み、作業前に依存パッケージをインストールして下さい。

If you are Ubuntu user:

```
sudo apt install python3-venv
python3.11 -m venv ~/.venv3.11
source ~/.venv3.11/bin/activate
which pip
pip install py-chessboardjs[gtk]
```

If you want to install it on local repository:

```
cd py-chessboardjs
pip install .[gtk]
```

QT user:

```
pip install py-chessboardjs[qt]
```

CEF user:

```
pip install py-chessboardjs[cef]
```

Install your favorite UCI engine:

```
sudo apt install stockfish
```

3.2 使用法

```
py-chessboardjs-gtk
```

```
py-chessboardjs-qt
```

```
py-chessboardjs-cef
```

3.3 関連リンク

- pywebview
- chessboard.js
- chess.js
- Bootstrap

第 4 章

Chem-Ant 紹介

MCTS Solver と Genetic Programming により、目標分子に似た分子を出力する為の材料候補を選ぶ。

`similarity_ant.py` は deap/examples/gp/ant.py のコードを改変しています。

4.1 準備

Ubuntu で：

```
sudo -H -s
apt install python3-pip
pip3 install -r requirements.txt
exit
```

或いは：

```
pip3 install deap
pip3 install mcts
pip3 install rdkit
pip3 install global_chem_extensions
pip3 install mcts-solver
```

或いは：

```
pip3 install chem-ant
```

`chem-ant` を `chem-classification` と共に使いたければ：

```
pip3 install simpletransformers
```

或いは：

```
pip3 install chem-classification
```

- DEAP
- MCTS
- RDKit
- rdkit-pypi
- Global-Chem
- chem-ant
- chem-classification
- mcts-solver

4.2 使用法

初期値では、 `smiles.csv` から分子のリストを得る。ターゲットは Nirmatrelvir だ。そのリストからフラグメントの材料として最適なものが選ばれる。出力される csv ファイルには、mcts 実行途中で出来た分子も含まれる。csv ファイルを smiles list として再利用したければ、 **`--select`** オプションを追加する。インストールせずに直接実行したければ、 **`python3 similarity_mcts.py --help`** とする。

```
similarity-mcts --help
similarity-mcts -i -l1 -e3 -r10 -b500 -p train_smiles
similarity-mcts -i -l1 -e3 -r10 -b500 -p eval_smiles
```

ターゲットを指定して実行したければ：

```
similarity-mcts -i -l1 -e3 -r10 -b500 -p train_smiles -t
↪"CC(C)(C)C(NC(=O)C(F)(F)F)C(=O)N1CC2C(C1C1CCNC1=O)C2(C)C"
similarity-mcts -i -l1 -e3 -r10 -b500 -p eval_smiles -t
↪"CC(C)(C)C(NC(=O)C(F)(F)F)C(=O)N1CC2C(C1C1CCNC1=O)C2(C)C"
```

`similarity-mcts` は、smiles のリストからフラグメントの材料と成り得る候補を選び出力する。mcts を実行せず、単にその smiles list からターゲットに似た分子を出力したければ：

```
similarity-genMols --help
similarity-genMols -t "CC1(C2C1C(N(C2)C(=O)C(C(C)(C)C)NC(=O)C(F)(F)F)C(=O)NC(CC3CCNC3=O)C
↪#N)C" -m "CC1=CC=CC=C1C(C)C" "Cc1ccccc1CC(C#N)NC1CCNC1=O" -f "gen2.csv"
```

4.3 Chem-Classification

chem-classification 用に json format のデータセットを出力する：

```
importSmiles -t "CC1(C2C1C(N(C2)C(=O)C(C(C)(C)C)NC(=O)C(F)(F)F)C(=O)NC(CC3CCNC3=O)C#N)C"␣
↪-p "train_smiles"
importSmiles -t "CC1(C2C1C(N(C2)C(=O)C(C(C)(C)C)NC(=O)C(F)(F)F)C(=O)NC(CC3CCNC3=O)C#N)C"␣
↪-p "eval_smiles"
```

regression model 用のデータセットを出力したければ：

```
importSmiles -t "CC1(C2C1C(N(C2)C(=O)C(C(C)(C)C)NC(=O)C(F)(F)F)C(=O)NC(CC3CCNC3=O)C#N)C"␣
↪-p "train_smiles" -r
importSmiles -t "CC1(C2C1C(N(C2)C(=O)C(C(C)(C)C)NC(=O)C(F)(F)F)C(=O)NC(CC3CCNC3=O)C#N)C"␣
↪-p "eval_smiles" -r
```

classification model を訓練し、Nirmatrelvir と YH-53 の類似性を予測する：

```
from chem_classification.similarity_classification import SimilarityClassification
s = SimilarityClassification()
s.train_and_eval("train_smiles/smiles.json", "eval_smiles/smiles.json")
s.predict_smiles_pair([
↪"CC1(C2C1C(N(C2)C(=O)C(C(C)(C)C)NC(=O)C(F)(F)F)C(=O)NC(CC3CCNC3=O)C#N)C",
↪"CC(C)CC(C(=O)NC(CC1CCNC1=O)C(=O)C2=NC3=CC=CC=C3S2)NC(=O)C4=CC5=C(N4)C=CC=C5OC"])
```

保存したモデルの読み込み：

```
s = SimilarityClassification("local-path/your-outputs")
```

regression model を訓練し、 Nirmatrelvir と YH-53 の類似性を予測する：

```
from chem_classification.similarity_classification import SimilarityRegression
s = SimilarityRegression()
s.train_and_eval("train_smiles/smiles.json", "eval_smiles/smiles.json")
s.predict_smiles_pair([
↪"CC1(C2C1C(N(C2)C(=O)C(C(C)(C)C)NC(=O)C(F)(F)F)C(=O)NC(CC3CCNC3=O)C#N)C",
↪"CC(C)CC(C(=O)NC(CC1CCNC1=O)C(=O)C2=NC3=CC=CC=C3S2)NC(=O)C4=CC5=C(N4)C=CC=C5OC"])
```

similarity-mcts により出力された json files により訓練された、もう一つの regression model は、材料候補からターゲット分子との類似性を予測する。そして、 **similarity-ant** と連携できる：

```
similarity-mcts -i -l2 -e3 -r10 -b100 -p "train_smiles" -f "smiles.json" -j
similarity-mcts -i -l2 -e3 -r10 -b100 -p "eval_smiles" -f "smiles.json" -j
```

注釈: chem-ant 0.0.7 から、分子のフラグメントをトークンとしてデータセットを作成するように変更したので、2 つの regression models の差は無くなった。

---

chem-classification と **similarity-ant** の連携（現状では機能せず）：

```
similarity-ant -n20 -g5 -b 1 -p gen_smiles -d -o "local-path/your-outputs"
```

Regression model の chem-classification と **similarity-ant** の連携：

```
similarity-ant -n20 -g5 -b 1 -p gen_smiles -r -o "local-path/your-outputs"
```

第 5 章

Chem-Ant 論文

author

Akihiro Kuroiwa

date

2022/07/08

abstract

2019 年に `chess-ant` を書き始めたのだが、最初は minimax にこだわり、遅々として作業は進まなかった。2020 年にクルーズ船ダイヤモンド・プリンセス号の COVID-19 集団感染があり、いよいよパンデミックが注目され始めた頃、 `chess-ant` のアルゴリズムを治療薬の開発に役立てようと思い立った。その後、 MCTS solver の論文を読み、性能が向上した。同時に cheminformatics のソフトウェアの使い方も身についていった。SARS-CoV-2 のパンデミックが収束した後も、次なるパンデミックが待ち構えている。我々の技能で社会に貢献しよう。

5.1 UCSF Chimera

私の古いラップトップ Fujitsu LIFEBOOK AH42/C に UCSF ChimeraX をインストールしようとしたら、次のようなエラーが表示された：

```
ERROR: ChimeraX requires OpenGL graphics version 3.3.
Your computer graphics driver provided version 2.1
Try updating your graphics driver.
```

従って、この実験では、 UCSF Chimera を用いる。その利点は、 binding site と呼ばれる結合部位をマウスで指定できることにある。

```
chmod u+x chimera-alpha-linux_x86_64.bin
./chimera-alpha-linux_x86_64.bin
```

Ubuntu の ~/.profile で：

```
if [ -d "$HOME/.local/bin" ] ; then
    PATH="$HOME/.local/bin:$PATH"
fi
```

私の場合、インストールする場所は 2 箇所に分かれる：

```
~/.local/bin/
~/.local/UCSF-Chimera64-2022-05-18/
```

ターミナルを開いてコマンドラインで **chimera** を実行するか、デスクトップに出来たアイコンを右クリックし、実行権限を与え、ダブルクリックする。

実験に先立ち、最新リリースの AutoDock Vina installer from GitHub を入手し、マニュアルに沿いインストールする。バイナリーファイル（実行ファイル）の在り処は、次のコマンドで確認できる：

```
cd ~/.local/bin/
ln -s vina_1.2.3_linux_x86_64 vina
which vina
```

動作検証をするために、次のように実験を進める：

1. ターゲット分子 Nirmatrelvir のみをフラグメントにし、分子を幾つか出力する。
2. ターゲット分子を含む材料候補を **similarity-mcts** で選び、分子を幾つか出力する。
3. 両者の 差集合を 得る。

```
similarity-genMols -t "CC1(C2C1C(N(C2)C(=O)C(C(C)(C)C)NC(=O)C(F)(F)F)C(=O)NC(CC3CCNC3=O)C
↪#N)C" -m "CC1(C2C1C(N(C2)C(=O)C(C(C)(C)C)NC(=O)C(F)(F)F)C(=O)NC(CC3CCNC3=O)C#N)C" -b70␣
↪-p "gen_smiles" -f "gen1-1.csv"
similarity-mcts -l2 -e3 -r10 -b100 -p "gen_smiles" -f "gen1-2.csv"
```

実行後、次のように表示されただろう：

```
Material candidates: {'CC1(C2C1C(N(C2)C(=O)C(C(C)(C)C)NC(=O)C(F)(F)F)C(=O)NC(CC3CCNC3=O)C
↪#N)C', 'CCCC1=NC(=C(N1CC2=CC=C(C=C2)C3=CC=CC=C3C4=NNN=N4)C(=O)O)C(C)(C)O'}
```

similarity-genMols を実行する際に気をつけることがある。Python ではシングルクォートとダブルクォートを区別しないが、 bash や dash では区別する。更に、コマンドラインでカンマは不要だ：

```
similarity-genMols -t "CC1(C2C1C(N(C2)C(=O)C(C(C)(C)C)NC(=O)C(F)(F)F)C(=O)NC(CC3CCNC3=O)C
↪#N)C" -m "CC1(C2C1C(N(C2)C(=O)C(C(C)(C)C)NC(=O)C(F)(F)F)C(=O)NC(CC3CCNC3=O)C#N)C"
```

(次のページに続く)

(前のページからの続き)

```
↪"CCCC1=NC(=C(N1CC2=CC=C(C=C2)C3=CC=CC=C3C4=NNN=N4)C(=O)O)C(C)(C)O" -b100 -f gen1-2.csv
similarity-genMols -t "CC1(C2C1C(N(C2)C(=O)C(C(C)(C)C)NC(=O)C(F)(F)F)C(=O)NC(CC3CCNC3=O)C
↪#N)C" -m "CCCC1=NC(=C(N1CC2=CC=C(C=C2)C3=CC=CC=C3C4=NNN=N4)C(=O)O)C(C)(C)O" -b100 -f␣
↪gen1-3.csv
```

```
import pandas as pd
df1_1 = pd.read_csv("gen_smiles/gen1-1.csv", header=0, index_col=0)
df1_2 = pd.read_csv("gen_smiles/gen1-2.csv", header=0, index_col=0)
df1_3 = pd.read_csv("gen_smiles/gen1-3.csv", header=0, index_col=0)
df1_4 = pd.concat([df1_1, df1_1, df1_2, df1_3, df1_3], axis=0)
df1_4.drop_duplicates(subset="smiles", keep=False, inplace=True)
df1_4.sort_values(["lipinski", "dice_similarity"], inplace=True, ascending=False)
df1_4.reset_index(drop=True).to_csv("gen_smiles/gen1-4.csv")
```

Ligand file を Open Babel で作成する。`gen3.csv` を開き、 **`similarity-mcts`** で得た高得点の分子の smiles を指定しよう。水素原子を付加し、部分電荷を割り当てるのを忘れないように。Ubuntu で：

```
sudo apt install openbabel
obabel -L
obabel -L charges
obabel -h -c -ican -:"CCCC1C2C(CN1C(=O)C1C3C(CN1C(=O)C(F)(F)F)C3(C)C)C2(C)C" -opdbqt -O␣
↪ligand.pdbqt --gen3D --partialcharge gasteiger
```

UCSF Chimera に戻ろう。上記のファイルを開き、以下のようにメニューを辿る：

1. *File → Fetch Structure by ID → PDB(mmCIF) → 7tll*
2. *File → Open → ligand.pdbqt → file type PDB*
3. *Tools → Surface/Binding Analysis → AutoDock Vina*

我々の *Output file* は `all` だ。*Receptor* と *Ligand* を指定しよう。*Resize search volume using* をマウス操作の為にチェックする。上記のコマンドで得た vina のパスを *Executable location* に書く。

私の場合、 binding site をマウスで指定すると、 *Presets* メニューで切り替えるまで枠が表示されない。実験結果を後程、再確認したければ `all.receptor.pdb` を開き、次のようにする：

1. *Tools → Surface/Binding Analysis → ViewDock → all.pdbqt*
2. *Move → Play*

5.2 AutoDock Vina

UCSF Chimera により出力された receptor file を再利用し、コマンドラインであなたが独自に作成した ligand file を実験したいだろう。設定ファイル `conf.txt` を用意しよう：

```
receptor = all.receptor.pdbqt
ligand = ligand.pdbqt

out = all.pdbqt

center_x = -2.68714
center_y = -1.23572
center_z = 13.8821

size_x = 25.747
size_y = 22.6627
size_z = 22.1881
```

`similarity-mcts` は今回は Catechin と謎めいた分子 Gnididin[*1] を選んだ：

```
similarity-mcts -l2 -e3 -r10 -b100 -p "gen_smiles" -f "gen2-2.csv"
```

```
Material candidates: {'C1C(C(OC2=CC(=CC(=C21)O)O)C3=CC(=C(C=C3)O)O)O',
↪'CCCCCC=CC=CC(=O)OC1C(C23C4C=C(C(=O)C4(C(C5(C(C2C6C1(OC(O6)(O3)C7=CC=CC=C7)C(=C)C)O5)CO)O)O)C)C
↪'}
```

`gen2-2.csv`:

```
,smiles,dice_similarity,lipinski
0,C=C(C)C12OC3(CO)OC1C1C4OC4(CO)C(O)C4(O)C(=O)C(C)=CC4C1(O3)C(C)C2CO,0.19672131147540983,
↪1.0
```

残念ながら、この分子は Gnididin のみで生成されたフラグメントにより出来る。

```
obabel -h -c -ican -:"C=C(C)C12OC3(CO)OC1C1C4OC4(CO)C(O)C4(O)C(=O)C(C)=CC4C1(O3)C(C)C2CO
↪" -opdbqt -O ligand.pdbqt --gen3D --partialcharge gasteiger
```

AutoDock Vina を実行する：

[*1] Sisakht, M., Mahmoodzadeh, A., & Darabian, M. (2021). Plant-derived chemicals as potential inhibitors of SARS-CoV-2 main protease (6LU7), a virtual screening study. Phytotherapy research : PTR, 35(6), 3262–3274. https://doi.org/10.1002/ptr.7041

```
vina --config conf.txt
```

```
mode |   affinity | dist from best mode
     | (kcal/mol) | rmsd l.b.| rmsd u.b.
-----+------------+----------+----------
   1       -8.765          0          0
   2        -8.31       1.82      6.828
   3       -8.252      2.441      4.152
   4       -8.086      1.664      7.041
   5        -7.85      2.301      7.148
   6       -7.825      1.726      6.693
   7       -7.797      3.008      6.184
   8       -7.412      2.183      7.011
   9       -7.339      2.426      4.168
```

5.3 鋳型と鋳物

参照した論文に書かれているアミノ酸相互作用は[*2] PDB ID: 6LU7 に基づく。一方、我々の実験は the SARS-CoV-2 Mpro Omicron P132H に基づいており、 PDB ID: 7TLL を使用する。Active site amino acid の冒頭 3 文字はアミノ酸の略号で、残りは配列上の位置を示す。UCSF Chimera で確認しよう：

1. *Presets → Interactive 1 (ribbons)* と **chimera** のメニューで辿る。
2. カーソルを binding site 上の receptor の active site amino acid に合わせると、その位置が表示される。
3. **chimera** で *Tools → Sequence → Sequence* を辿り、ヌクレオチドやアミノ酸の sequence alignment を表示し、 fast format で保存する。
4. 活性部位のアミノ酸を確認したいならば、配列上のその場所で右クリックする。

鋳型から鋳物を鋳造すれば、元の鋳型に合うはずだ。私が `smiles.csv`[*3] にアミノ酸とヌクレオチドを追加した理由でもある。Docking simulation における binding site と ligand の関係も同様だと言えるかどうか、以下の方法で実験する：

1. 該当箇所を `rdkit` で smiles に変換する。範囲は Phe140 から Glu166 迄だ。
2. その smiles 文字列は長過ぎるので、 fragments に分解し、幾つかの分子に出力する。

[*2] SAMANT, L., & Javle, V. (2020). Comparative Docking Analysis of Rational Drugs Against COVID-19 Main Protease. ChemRxiv. doi:10.26434/chemrxiv.12136002.v1 This content is a preprint and has not been peer-reviewed.

[*3] PubChem

```
from rdkit import Chem
from rdkit.Chem import BRICS
Chem.MolToSmiles(Chem.MolFromFASTA("FLNGSCGSVGFNIDYDCVSFCYMHHME"))
smiles =
↪'CC[C@H](C)[C@H](NC(=O)[C@H](CC(N)=O)NC(=O)[C@H](Cc1ccccc1)NC(=O)CNC(=O)[C@@H](NC(=O)[C@H](CO)NC(=O)CN
↪'
allfrags = set()
allfrags.update(BRICS.BRICSDecompose(Chem.MolFromSmiles(smiles), returnMols=True))
builder = BRICS.BRICSBuild(allfrags)
generated_smiles = []
for i in range(30):
    mol = next(builder)
    mol.UpdatePropertyCache(strict=True)
    generated_smiles.append(Chem.MolToSmiles(mol))
generated_smiles
['CSCC[C@H](SC)C(=O)N[C@@H](CCC(=O)O)C(=O)O', 'CSCC[C@H](SC)C(=O)Nc1c[nH]cn1',
↪'CSCC[C@H](SC)C(=O)Nc1ccc(O)cc1', 'CSCC[C@H](SC)C(=O)Nc1ccccc1',
↪'CS[C@@H](CC(C)C)C(=O)Nc1ccc(O)cc1',
↪'CC(C)C[C@H](N[C@@H](CCC(=O)O)C(=O)O)C(=O)Nc1ccc(O)cc1',
 'CC(C)C[C@H](Nc1ccc(O)cc1)C(=O)Nc1ccc(O)cc1', 'CC(C)C[C@H](Nc1ccccc1)C(=O)Nc1ccc(O)cc1',
 'CC(C)C[C@H](Nc1c[nH]cn1)C(=O)Nc1ccc(O)cc1', 'CS[C@@H](CC(C)C)C(=O)Nc1c[nH]cn1',
↪'CS[C@@H](CC(C)C)C(=O)Nc1ccccc1', 'CS[C@@H](CC(C)C)C(=O)N[C@@H](CCC(=O)O)C(=O)O',
↪'CS[C@@H](CC(=O)O)C(=O)NC(=O)[C@H](CC(C)C)SC',
↪'CS[C@@H](CC(C)C)C(=O)NC(=O)[C@@H](SC)C(C)C',
 'CS[C@@H](CC(N)=O)C(=O)NC(=O)[C@H](CC(C)C)SC', 'CS[C@@H](CC(C)C)C(=O)NC(=O)[C@H](CS)SC',
 'CS[C@@H](CC(C)C)C(=O)NC(=O)[C@@H](N)Cc1c[nH]cn1',
↪'CS[C@@H](CC(C)C)C(=O)NC(=O)[C@@H](N)Cc1ccc(O)cc1',
 'CS[C@@H](CC(C)C)C(=O)NC(=O)[C@@H](N)Cc1ccccc1', 'CS[C@@H](CO)C(=O)NC(=O)[C@H](CC(C)C)SC
↪', 'CS[C@@H](CC(C)C)C(=O)NC(=O)[C@H](CC(C)C)SC',
↪'CC[C@H](C)[C@H](SC)C(=O)NC(=O)[C@H](CC(C)C)SC', 'CSCC(=O)NC(=O)[C@H](CC(C)C)SC',
↪'CC(C)C[C@H](N[C@@H](CCC(=O)O)C(=O)O)C(=O)Nc1ccccc1',
↪'CC(C)C[C@H](Nc1c[nH]cn1)C(=O)Nc1ccccc1', 'CC(C)C[C@H](Nc1ccc(O)cc1)C(=O)Nc1ccccc1',
↪'CC(C)C[C@H](Nc1ccccc1)C(=O)Nc1ccccc1',
↪'CC(C)C[C@H](Nc1ccc(O)cc1)C(=O)N[C@@H](CCC(=O)O)C(=O)O',
↪'CC(C)C[C@H](Nc1ccccc1)C(=O)N[C@@H](CCC(=O)O)C(=O)O',
↪'CC(C)C[C@H](N[C@@H](CCC(=O)O)C(=O)O)C(=O)N[C@@H](CCC(=O)O)C(=O)O']
```

1. 出力された分子から、ドッキング・シミュレーションで好成績となった分子を選ぶ。勿論、選択肢の中での好成績だが。

2. その分子をターゲットに **similarity-mcts** を動かす。

```
obabel -h -c -ican -:"CC(C)C[C@H](Nc1ccc(O)cc1)C(=O)N[C@@H](CCC(=O)O)C(=O)O" -opdbqt -O␣
↪ligand.pdbqt --gen3D --partialcharge gasteiger
vina --config conf.txt
```

```
mode |   affinity | dist from best mode
     | (kcal/mol) | rmsd l.b.| rmsd u.b.
-----+------------+----------+----------
   1       -7.192          0          0
   2       -7.014      2.837      4.843
   3       -7.002      1.339      2.292
   4       -7.001      2.143      4.417
   5       -6.894      1.303       2.67
   6       -6.759      2.539       6.62
   7       -6.578      2.329      7.121
   8       -6.547      3.004      7.767
   9        -6.51        1.3      2.908
```

```
similarity-mcts -i -l2 -e3 -r10 -b100 -p "gen_smiles" -f "gen3-2.csv" -t
↪"CC(C)C[C@H](Nc1ccc(O)cc1)C(=O)N[C@@H](CCC(=O)O)C(=O)O"
```

```
Material candidates: {'CC1CCC2C(C(OC3C24C1CCC(O3)(OO4)C)OC)C', 'C(CCN)CC(C(=O)O)N',
↪'CC(C)C[C@H](Nc1ccc(O)cc1)C(=O)N[C@@H](CCC(=O)O)C(=O)O'}
```

```
,smiles,dice_similarity,lipinski
0,CCNC1CCNC1=O,0.5454545454545454,1.0
1,CC(Nc1ccccc1)C(=O)Oc1ccccc1,0.5269607843137255,1.0
2,COC(=O)[C@H](CC(C)C)Nc1ccc(O)cc1,0.5084427767354597,1.0
3,CC(C)C(=O)OC(=O)C(C)C,0.5078125,1.0
4,O=C1NCCC1Nc1ccc(F)cc1,0.5066162570888468,1.0
5,COC(=O)[C@H](CC(C)C)OC,0.5040983606557377,1.0
6,C(c1nn[nH]n1)c1nn[nH]n1,0.4921875,1.0
7,CCOc1nn[nH]n1,0.4765625,1.0
8,COc1ccc(O)cc1,0.46875,1.0
9,CO[C@@H](CCC(=O)O)C(=O)O,0.4609053497942387,1.0
10,CC(C(=O)N1Cc2ccccc2CC1C(=O)O)N1Cc2ccccc2CC1c1ccccc1,0.4494649227110582,1.0
11,CC(C(=O)N1Cc2ccccc2CC1C(=O)O)N1Cc2ccccc2CC1C(=O)O,0.4436183395291202,1.0
12,c1ccc(C2CC3CCCC3N2c2ccccc2)cc1,0.4426666666666666,1.0
```

(次のページに続く)

(前のページからの続き)

```
13,COC(=O)[C@H](CC(C)C)NC1OC2OC3(C)CCC4C(C)CCC(C1C)C24003,0.43861607142857145,1.0
14,O=C(O)C1Cc2ccccc2CN1c1ccccc1,0.4348387096774193,1.0
15,CC1CCC2C(C)C(Nc3ccc(O)cc3)OC3OC4(C)CCC1C32004,0.4311717861205916,1.0
16,CO[C@@H](CC(C)C)C(=O)NC1OC2OC3(C)CCC4C(C)CCC(C1C)C24003,0.4242761692650334,1.0
17,CC(C(=O)Oc1ccccc1)N1C(c2ccccc2)CC2CCCC21,0.4230287859824781,1.0
18,CCOc1ccccc1C(=O)O,0.4228971962616822,1.0
19,Cc1cc(NC2CCNC2=O)no1,0.4113924050632911,1.0
```

短い smiles は無視する：

```
obabel -h -c -ican -:"CC(C(=O)N1Cc2ccccc2CC1C(=O)O)N1Cc2ccccc2CC1c1ccccc1" -opdbqt -O␣
↪ligand.pdbqt --gen3D --partialcharge gasteiger
vina --config conf.txt
```

```
mode |   affinity | dist from best mode
     | (kcal/mol) | rmsd l.b.| rmsd u.b.
-----+------------+----------+----------
   1       -8.417          0          0
   2       -8.302      2.467      6.754
   3       -8.003       4.96      7.503
   4       -7.624      3.907      6.517
   5       -7.506      5.171      7.983
   6       -7.485      2.979      5.385
   7       -7.433      5.416      9.353
   8       -7.375      2.857      5.204
   9       -7.296      2.998      5.526
```

類似性の目標となるターゲット分子そのものが必要な場合は、上記の方法が役立つかも知れない。

課題:

- MCTS solver を別パッケージにする
- ターゲット分子が無くともドッキング・シミュレーションで高得点を取れるシミュレーションは出来るか？
- Type bool が 1.0 や 0.0 のように csv file に出力されてしまう。
- **similarity-ant** は遅すぎて実用性に程遠い。
- そもそも **similarity-mcts** は正しく動作しているのか？
- Version 0.0.3 の **similarity-mcts** は MCTS solver をインポートするので、この文書と出力が若干異なる。

- 絡み合ったスパゲッティコードを解く。

5.4 参照

5.5 参考文献

- 化学の新しいカタチ
- Python for chemoinformatics
- English version of Python for Chemoinformatics (pdf)
- Sharif, Suliman. Understanding drug-likeness filters with RDKit and exploring the WITHDRAWN database. (2020).
- Panikar, S., Shoba, G., Arun, M., Sahayarayan, J. J., Usha Raja Nanthini, A., Chinnathambi, A., Alharbi, S. A., Nasif, O., & Kim, H. J. (2021). Essential oils as an effective alternative for the treatment of COVID-19: Molecular interaction analysis of protease (Mpro) with pharmacokinetics and toxicological properties. Journal of infection and public health, 14(5), 601–610. https://doi.org/10.1016/j.jiph.2020.12.037
- @cat_lover. 構造生成メモ. (2021).
- GB-GM
- Jensen, J. (2019). Graph-based Genetic Algorithm and Generative Model/Monte Carlo Tree Search for the Exploration of Chemical Space. ChemRxiv. doi:10.26434/chemrxiv.7240751.v2 This content is a preprint and has not been peer-reviewed.

第 6 章

Chem-Ant プロジェクト - 創業メンバー募集

6.1 概要:

Chem-Ant プロジェクトでは、治療薬候補の探索に革新的なアプローチを追求しています。現在、プロジェクトは創業メンバーを募集しています。これは、rdkit と deap の genetic programming、そして MCTS solver を組み合わせたシミュレーションのプロジェクトで、まだ会社は設立されていません。現在はプロジェクトリーダーである私一人です。

6.2 募集ポジション:

- ソフトウェア開発者
- データサイエンティスト
- 化学情報学者
- マーケティング担当
- その他、プロジェクトに興味をお持ちの方

6.3 求めるスキル:

- Cheminformatics、Molecular Biology、または関連する分野の知識
- プログラミングスキル（Python 等）
- チームワークとコミュニケーション能力

6.4 応募方法:

ご興味をお持ちいただけましたら、git log にあるメールアドレスにご連絡ください。また、自己紹介やスキルセットを添えていただけると幸いです。

6.5 注意:

Chem-Ant はまだ設立されておらず、現在はプロジェクトとして進行中です。これは将来の企業設立に向けた先駆的なプロジェクトであり、共に成長し、新たな治療薬開発のフロンティアに挑戦したい方を歓迎します。

第 7 章

索引

- genindex
- modindex
- search