Vol.2018-MPS-118 No.42
Vol.2018-BIO-54 No.42
2018/6/15

共通な部分構造の再利用アルゴリズムを用いたタンパク質リガンドドッキング手法の開発

久保田 陸人[1,2]　柳澤 渓甫[1]　大上 雅史[1,3]　秋山 泰[1,3,a)]

概要：創薬研究では数百万〜数億件というきわめて多数の化合物群を扱う必要があり，ドッキング計算の高速化が求められている．本報告では，化合物の多くが共通な部分構造を持つことに着目し，それらの計算結果を再利用することにより高速なドッキングツールを開発した．ヒューリスティックによる化合物の評価順序の変更や，オフラインキャッシュ問題を高速に解くことによるメモリ戦略の最適化により計算結果の再利用効率を改善し，改善前に比べ精度を落とすことなく約 1.8 倍の高速化を実現した．

キーワード：バーチャルスクリーニング，タンパク質リガンドドッキング，フラグメント分割，オフラインキャッシュ問題

Development of an efficient protein-ligand docking method for virtual screening by reuse of fragments

Rikuto Kubota[1,2]　Keisuke Yanagisawa[1]　Masahito Ohue[1,3]　Yutaka Akiyama[1,3,a)]

Abstract: In drug discovery research, it is necessary to explore a large compound database composed of several millions of compounds, thus acceleration of docking calculation is greatly demanded. Reuse of calculation results is one of feasible ways to accelerate. In this report, we focused on the fact that many of the compounds have common substructures, and developed a high speed docking tool by reusing the calculation results of substructures. By reordering the order of evaluation of compounds by a heuristic method and optimizing memory strategy by solving offline cache problem at high speed, it is possible to improve the reuse efficiency of the calculation results and the calculation speed was increased about 1.8 times faster.

Keywords: virtual screening, protein-ligand docking, compound fragmentation, offline cache problem

1. 序論

創薬研究の初期のスクリーニングにおいてはきわめて多数の化合物群を扱わなくてはならない．購入可能な化合物のデータベースである ZINC [1] での登録数を例に挙げると，その数は 3500 万にも及ぶ．そこで，初期段階でコンピュータを用い薬剤の候補となる化合物をふるいにかける手法（バーチャルスクリーニング）が用いられている [2]．その中で，特に標的タンパク質と薬剤候補化合物それぞれの立体構造情報に基づく手法を Structure-Based Virtual Screening (SBVS) と呼ぶ．

ドッキング計算は，標的タンパク質と薬剤候補化合物の結合構造のスコアを定義して評価をすることにより結合の起こりやすさや結合構造を予測する手法であり，SBVS において広く用いられている [3]．しかし，化合物ライブラリに含まれる膨大な数の化合物を評価するには計算コスト

[1] 東京工業大学 情報理工学院 情報工学系，
Department of Computer Science, School of Computing, Tokyo Institute of Technology
[2] 産業技術総合研究所 産総研・東工大実社会ビッグデータ活用オープンイノベーションラボラトリ，
AIST-Tokyo Tech Real World Big-Data Computation Open Innovation Laboratory (RWBC-OIL), National Institute of Advanced Industrial Science and Technology (AIST)
[3] 東京工業大学 科学技術創成研究院 スマート創薬研究ユニット，
Advanced Computational Drug Discovery Unit (ACDD), Institute of Innovative Research, Tokyo Institute of Technology
[a)] akiyama@c.titech.ac.jp

Vol.2018-MPS-118 No.42
Vol.2018-BIO-54 No.42
2018/6/15

が膨大となってしまうため，ドッキング計算の高速化が求められている．

薬剤候補化合物は共通な部分構造を持つことが多い [4]．そこで，薬剤候補化合物を内部に回転可能な結合のない構造（以下ではこれをフラグメントと呼ぶ）になるまで分割し，フラグメントと標的タンパク質とのエネルギースコアの計算結果を保存し再利用することで，元の化合物のドッキング計算を高速に実現するというアイデアが提案されている [5]．このフラグメントのエネルギースコアを保存するグリッドをフラグメントグリッドと呼ぶ．ただし，球体と考えられる原子とは違い，フラグメントには向きがあるものも多く回転も考慮しなければならない．先行研究 [6] では，3 次元の回転を正十二面体の面と頂点に基づく 60 通りの回転方向を用いて実現していた．よって，フラグメントグリッドは原子グリッドと比べ単純に回転方向の個数倍のメモリを必要とし，例えば一辺 20 Å の立方体内に 0.25 Å 刻み，回転方向 60 方向でエネルギースコアを 4 バイト浮動小数点数で保存した場合，メモリサイズは約 120 MB にもなる．さらにフラグメントは種類の数も原子に比べ多いので，全てのフラグメントグリッドを同時にメモリ上に保持することは非常に困難である．この問題点に関しては，どのフラグメントグリッドを保持するか，というメモリ戦略をオフラインキャッシュ問題に帰着させ，それを独自の高速なアルゴリズムで解く手法が提案されている [5]．

本研究では，化合物をフラグメントに分割し，フラグメントのエネルギースコアの計算結果を保存して再利用する，という手法を用いて 1 つの標的タンパク質に対して複数の薬剤候補化合物を評価することに特化した高速ドッキングツールを開発した．

2. ドッキングシミュレーション

2.1 ドッキングシミュレーションの概要

ドッキングシミュレーションは標的タンパク質と薬剤候補化合物の結合構造をスコアを定義して評価をすることにより，結合が起こるか否か，及び本来の結合構造を予測する手法である．ドッキングシミュレーションはいくつかの分類が可能だが，ここではフレキシブルドッキング [7] とコンフォーマードッキング [8]，化合物に基づいたドッキング [9, 10] とフラグメントに基づいたドッキング [6, 11]，という 2 つの分類方法で既存手法を説明していく．

2.1.1 フレキシブルドッキングとコンフォーマードッキング

ドッキングシミュレーションでは標的タンパク質と薬剤候補化合物の相対位置や回転，さらには化合物内部の単結合の回転なども考えなければならず，探索空間がとても広い．そこで上記を全て考慮して探索を行うフレキシブルドッキングに対し，薬剤候補化合物の内部の回転を考慮しないことにより高速化を図る剛体ドッキング [12] という手法が存在する．しかし当然ながら，薬剤候補化合物の内部の回転を考慮しない分，剛体ドッキングはフレキシブルドッキングに比べて精度が落ちる [13]．そこで，剛体ドッキングを行う前に，OMEGA [14] などの配座生成ツールを用いて薬剤候補化合物の取りうる構造（コンフォーマー）を複数用意しておくことで，薬剤候補化合物の内部の回転を考慮せずに精度を保とうという手法があり，コンフォーマードッキングと呼ばれる．本研究ではこのコンフォーマードッキングを採用する．

2.1.2 化合物に基づいたドッキングとフラグメントに基づいたドッキング

化合物を分割せずにそのままドッキングシミュレーションを行う化合物に基づいたドッキングに対し，化合物を分割し部分構造（フラグメント）に注目する手法が存在しフラグメントに基づいたドッキングと呼ばれる．本研究ではこのフラグメントに基づいたドッキングを採用する．

3. 提案手法

図 1 に，既存手法と提案手法におけるドッキング計算の処理の流れを対比して示す．

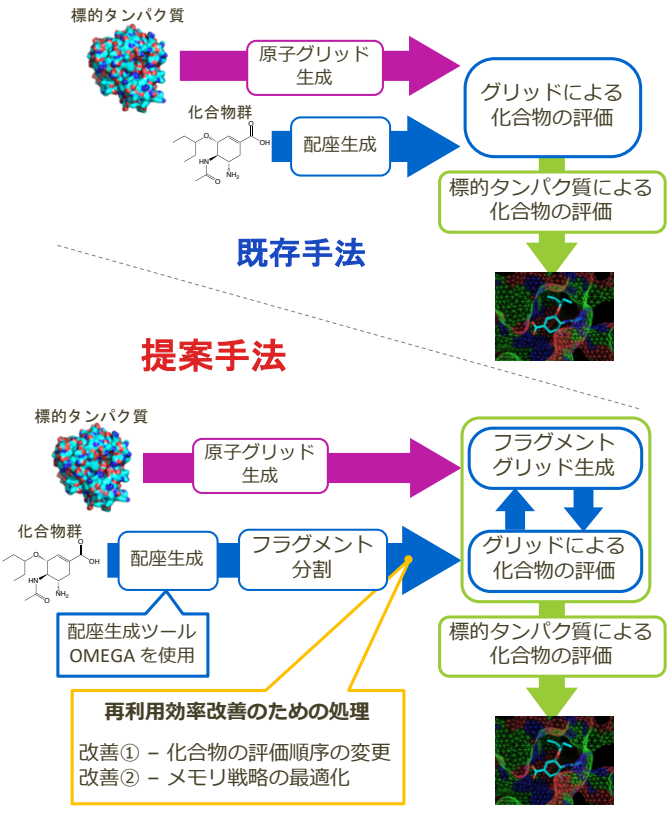

図 1 既存手法と提案手法におけるドッキング計算の処理の流れ

3.1 原子グリッドの生成

原子グリッドと呼ばれる，標的タンパク質と原子 1 つとのエネルギースコアを等間隔に前もって計算・保存したテーブルを用いて，高速にドッキング計算を行う手法が存

在する [9]．本研究でも同様に事前計算で原子グリッドを生成し，原子に対する計算結果の再利用を行う．

各原子種の標的タンパク質とのエネルギースコアを等間隔（初期設定では 0.25 Å ごと）に計算し保存する．この計算はドッキングを行う入力化合物とは関係なく標的タンパク質のみで計算ができる．スコア関数は，化合物側とタンパク質側の全ての原子のペアに対するスコアの総和で定義される，AutoDock Vina [9] における計算法を流用した．

3.2 配座生成

コンフォーマードッキングを行うために，OMEGA [14] を用いて入力化合物の取りうる構造を列挙する．一つの化合物に対して最大 200 個の配座を生成しており，これは OMEGA の初期設定に準拠している．

3.3 フラグメント分割

各化合物において、剛体として扱えるような部分構造（フラグメント）に注目し，フラグメントのエネルギースコアの計算結果を異なる化合物間で再利用することによって，無駄を省き高速化を図る手法が提案されている [5]．

3.3.1 フラグメント分割の基準

フラグメントの分割は，既存手法 [4, 15] において提案された**図 2** に例示されるような分割基準を採用した．なお，水素原子の座標は今回のスコアリング方法では考慮しないのでフラグメント分割のための基準としては無視する．

- Open Babel [16] において実装されたアルゴリズムにより結合が回転可能かどうかを判断する．
- 回転不可能な結合の両端の 2 原子は同一フラグメントに含める．
- 回転可能な結合においても，どちらか片方が 1 原子のみの場合は回転しても構造が変化しないので同一フラグメントとする．
- 環構造の場合は，その環を構成する全ての原子を同一フラグメントに含める．

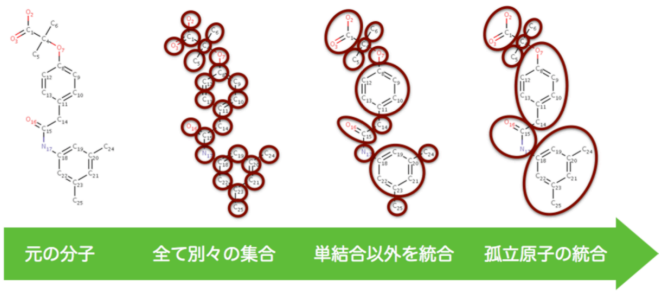

図 2 フラグメント分割の例 [15]

3.3.2 フラグメントの検索性

フラグメントの計算結果の再利用のためには，今着目しているフラグメントがすでに計算したものかどうか，ということが判断できる必要がある．そこで，Open Babel において実装された Canonical SMILES [17] を求めるアルゴリズムによりフラグメントを一意に文字列に変換し，ハッシュテーブルを用いることにより検索を可能とした．

3.3.3 フラグメントの回転角の定義

フラグメントは球体として考えられる原子とは異なり回転を考慮する必要がある．よって，以下のようにフラグメントの回転角を定義する．

(1) フラグメントに対し，結合により連続する 3 原子の組のうち 3 原子が同一直線上にないものを選択し，**図 3** のように $\mathbf{v}_1, \mathbf{v}_2$ を定義する．

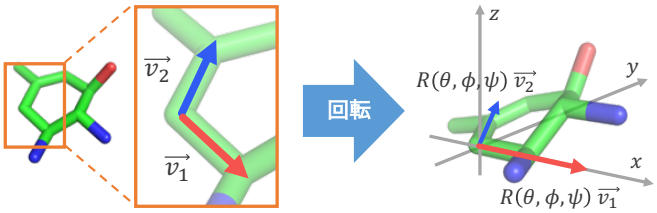

図 3 $\mathbf{v}_1, \mathbf{v}_2$ の決定及び回転角の定義

(2) 同一直線上にない 3 原子が存在しない場合，そのフラグメントは直線状の構造をしているので適当な 2 原子間のベクトルを $\mathbf{v}_1$ とし，$\mathbf{v}_1$ に直交するベクトルのうち 1 つを $\mathbf{v}_2$ とする．

(3) フラグメントの回転角は，上で定義した $\mathbf{v}_1, \mathbf{v}_2$ の基準の向きからの角度のずれで定義した．具体的には，式 (1) で定義される $\mathbf{R}(\theta, \phi, \psi)$ を用いて式 (2) を満たすような θ, ϕ, ψ を求めることで向きを決定した．なお式 (2) を満たすような x_1, x_2, y_2 は一意に定まるが，θ, ϕ, ψ は一意とならない可能性もある．

$$\mathbf{R}(\theta,\phi,\psi) = \mathbf{R}_z(\theta)\mathbf{R}_x(\phi)\mathbf{R}_z(\psi) \tag{1}$$

$$\mathbf{R}_z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}, \mathbf{R}_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix}$$

$$\begin{cases} \mathbf{R}(\theta,\phi,\psi)\mathbf{v}_1 = \begin{pmatrix} x_1, & 0, & 0 \end{pmatrix}^\top & (x_1 > 0) \\ \mathbf{R}(\theta,\phi,\psi)\mathbf{v}_2 = \begin{pmatrix} x_2, & y_2, & 0 \end{pmatrix}^\top & (y_2 > 0) \end{cases} \tag{2}$$

3.3.4 フラグメントの回転の実現

回転角は連続なので，計算結果の再利用をするためには回転角をある刻み幅で近似して計算及び保存をする必要がある．ここでは，先行研究 [6] と同様に正十二面体の面と頂点に基づく 60 通りの回転方向を採用した．このような定義を採用し，かつ回転方向のうち 1 つを恒等写像にすることにより回転演算が閉じる（回転を 2 回適用してもこの 60 通りのいずれかになる）ので，実際に各原子の座標計算を行うことなく回転操作を実現した．

3.4 フラグメントグリッドを用いた化合物の評価

入力の各化合物に対して，フラグメントグリッドを用いてエネルギースコアを計算していく．

まず，フラグメントの標的タンパク質とのエネルギースコアを等間隔（初期設定では 0.25 Å ごと）に計算しフラグメントグリッドを生成する．原子グリッドとは異なり，回転（60 方向）も考慮してそれら全てを計算，保存する．

そしてそのフラグメントグリッドを用いて，元の化合物の標的タンパク質とのエネルギースコアを等間隔（初期設定では 1 Å ごと）に全ての回転方向について計算する．

3.5 標的タンパク質を用いた上位ポーズの再評価

化合物の各座標，回転方向でのグリッドによるスコアを求めたのち，上位 N'_{out} 個（初期設定では $N'_{out} = 10$）のポーズを選ぶ．それらのポーズに対してグリッドを用いずに化合物及び標的タンパク質の各原子の位置情報を用いて厳密なエネルギースコアを計算し，上位 N_{out} 個（初期設定では $N_{out} = 5$）をスコアの良い順に出力を行う．

3.6 再利用効率の向上のための処理

フラグメントグリッドは全てを同時にメモリに保持しておくことは非常に困難であるため，フラグメントグリッドは破棄と再生成を繰り返す必要が出てくる．よって，再利用効率を高めるためにはできるだけ一度生成したフラグメントグリッドを繰り返し再利用することが重要となる．

本研究では，この再利用効率の向上のために以下の 2 つの処理を行った．

- 入力化合物の評価順序の変更
- フラグメントグリッドのメモリ戦略の最適化

3.6.1 入力化合物の評価順序の変更

同じフラグメントが出現するタイミングはなるべく近い方が再利用がしやすくなると考えられるので，図 4 に例示されるような手順で入力化合物の評価順序の変更を行う．

A. 各化合物内のフラグメントの並び替え

フラグメントのエネルギースコアはフラグメントの各原子のスコアの総和で定義されるので，フラグメントグリッドを生成するのにかかる時間はフラグメントの持つ原子の個数に比例する．よってフラグメントの再利用の優先順位を考えると，より原子数の多いフラグメントを再利用した方が効率が良いと考えられる．また，頻繁に出現するフラグメントの計算結果は保持しておくとより多く再利用が可能になるので，より出現回数の多いフラグメントを再利用することも再利用効率の向上に繋がると考えられる．

以上より，フラグメントの並び替えにおける優先度を式 (3) のように設定した．ここでフラグメント f に対し，$size(f)$ は f の原子数，$count(f)$ は入力化合物全体での f の出現回数を表す．$Priority(f)$ は，f のスコア計算を初回以外全て省略できた際に，スコア計算を省略できる原子の総個数と一致する．これを用いてフラグメントを優先度の高い順にランク付けし，各化合物内のフラグメントをランクが昇順になるようにソートすることにより並び替えた．

$$Priority(f) = size(f) \times (count(f) - 1) \qquad (3)$$

B. 入力化合物の並び替え

上記の操作により得られた各化合物のフラグメントのランクの列を用いて，入力化合物をフラグメントのランクの列が辞書順になるようにソートすることにより並び替えた．

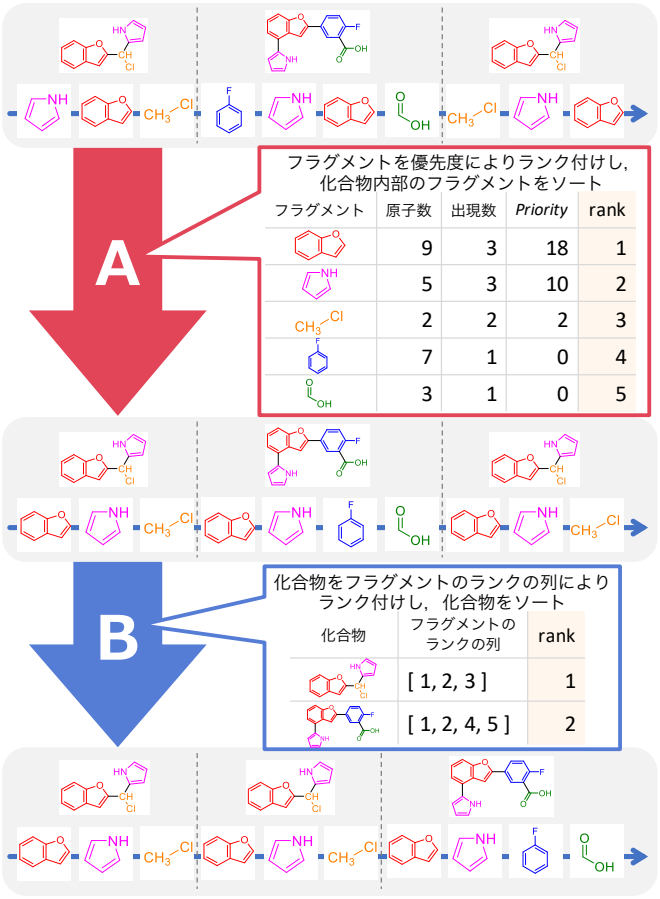

図 4 入力化合物の評価順序の変更の例

3.6.2 フラグメントグリッドのメモリ戦略の最適化

フラグメントグリッドのメモリ戦略はオフラインキャッシュ問題へ帰着させることができ，オフラインキャッシュ問題は最小費用流問題に帰着させて最適解を求められる [18]．

(1) オフラインキャッシュ問題への帰着

3.6.1 節の操作により，入力化合物の集合を評価するために必要となるフラグメントの順序が固定される．したがって，フラグメントグリッドのメモリ戦略はオフラインキャッシュ問題と考えることができる．3.6.1 節で述べたように，フラグメントグリッドを生成するのにかかる時間はフラグメントの持つ原子の個数に比例すると考えられるので，オフラインキャッシュ問題における各フラグメントのコストにはフラグメントの原子数を用いた．

Vol.2018-MPS-118 No.42
Vol.2018-BIO-54 No.42
2018/6/15

(2) 最小費用流問題への帰着

オフラインキャッシュ問題は最小費用流問題に帰着できることが知られている [18]．フラグメントの列を最小費用流問題のグラフへ変換する例を図 5 に示す．具体的には Algorithm 1 に示した擬似コードで変換を行った．キャッシュメモリの容量が M のオフラインキャッシュ問題のメモリ戦略の最適化は，このグラフに流量 $M-1$ のフローを流すことに対応する．

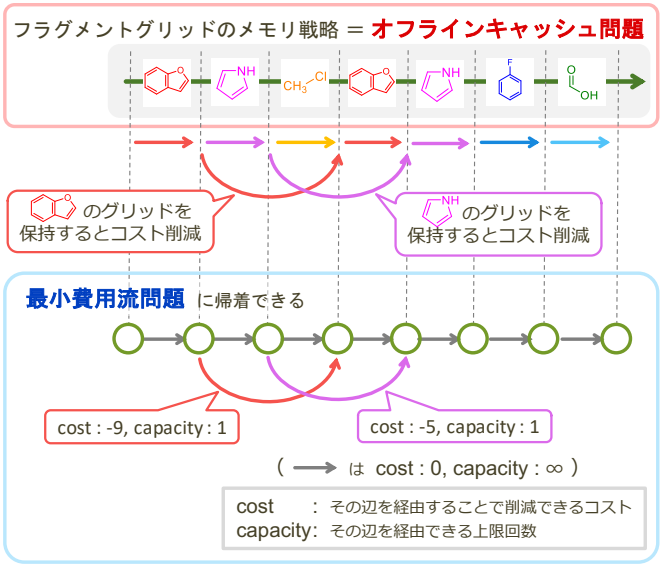

図 5　最小費用流問題のグラフへの変換の例

Algorithm 1 フラグメントの列からグラフへの変換

```
G.init(|fragments|)
                ▷ 頂点数 |fragments| で辺の持たないグラフ G を定義
before.fill(∞)        ▷ before : 各フラグメントの最後に登場した位置
for i ∈ {0, 1, ..., |fragments| − 1} do
  f ← fragments[i]
  if i > 0 then
    G.add_edge(i − 1, i, 0, ∞)
      ▷ G に頂点 i − 1 から頂点 i へコスト 0，容量 ∞ の辺を追加
  end if
  if before[f.id] < i − 1 then
    ▷ 同一フラグメントが既にあり，かつ最後にあったのが直前でない
    G.add_edge(before[f.id] + 1, i, −|f.atoms|, 1)
                          ▷ G に頂点 before[f.id] + 1 から頂点 i へ
                                コスト −size(f)，容量 1 の辺を追加
  end if
  before[f.id] ← i
end for
return G
```

3.6.3　最小費用流問題の解決

最小費用流問題は残余グラフにおける最短路に沿ってフローを流す，ということを貪欲に繰り返すことで最適解が得られる [19]．この最短路の導出にダイクストラ法を用いることにより，流量を F，グラフの辺の本数を $|E|$，頂点の数を $|V|$ としたとき最小費用流が $O(F|E|\log|V|)$ で求められる [20] が，我々は最短路導出に対してオフラインキャッシュ問題から帰着されるグラフに特化した，より高速なアルゴリズムを提案し [5]，それにより高速にオフラインキャッシュ問題を解決した．このアルゴリズムは計算量（推定，未証明）は $O(F|E|\log|V|)$ で変わらないが，従来の手法 [20] に比べ約 7 倍の高速化を実現している．

3.6.4　メモリ戦略への復元

フローを流し切ったグラフから，Algorithm 2 のようにフラグメントグリッドのメモリ戦略を復元する．各フラグメントグリッドの格納先のメモリの情報が *store_index* に保存され，すでに格納先に同一のフラグメントグリッドがあった場合に再計算が不要となり，再利用が達成される．

Algorithm 2 グラフからメモリ戦略への復元

```
size ← （メモリに格納できるフラグメントグリッドの個数）
empty_mem = {0, 1, 2, ..., size − 1}
                ▷ empty_mem : 今現在使われていないメモリの集合
for i ∈ {0, 1, ..., |G.nodes| − 1} do
  if 直前が同一フラグメントである then
    store_index[i] ← store_index[i − 1]
  else if （頂点 i に同一フラグメントから伸びてきた辺があり
          フローが流れている） then
    edge ← （頂点 i に同一フラグメントから伸びてきた辺）
    store_index[i] ← store_index[edge.from − 1]
  else
    （x ∈ empty_mem であるような x を 1 つ選択）
    empty_mem ← empty_mem \ {x}
    store_index[i] ← x
  end if
  if （頂点 i から同一フラグメントへ伸びている辺が無い
      またはフローが流れていない） then
    empty_mem ← empty_mem ∪ {store_index[i]}
  end if
end for
return store_index
```

4.　評価実験

4.1　比較対象

提案手法の速度向上における有用性を示すため，大きく分けて以下の 2 通りの手法と実行速度の比較を行った．

4.1.1　提案手法を部分的に適用した手法

本研究では，フラグメントの再利用効率の向上のため，(1) 化合物の評価順序の変更 (2) フラグメントグリッドのメモリ戦略の最適化，という 2 つの手法を用いている．これらの手法の個々の実行速度への影響を調べるため，(1) 化合物の評価順序の変更を行うか否か，(2) フラグメントグリッドのメモリ戦略の最適化を行うか否か（行わない場合は Least Recently Used によるオンライン最適化を行う），という合計 4 通りの手法で実行速度を比較した．

4.1.2　フラグメントを用いない手法

提案手法では，フラグメントグリッドによりフラグメントの計算結果を再利用することで計算の高速化を図ってい

る．その手法が効率的であるのかを検証するため，フラグメントに分割せず原子グリッドを用いてドッキング計算を行うプログラムを作成し，実行時間の比較を行った．

4.2 データセット

4.2.1 入力化合物

入力化合物には ZINC Drug Database (ZDD) [1] (2,924 entries) を用いた．なお OMEGA による配座生成に失敗したものがあり，実際に用いた化合物は 2,886 種類となった．

これらのうち，ランダムに化合物を選択することにより化合物が 10 種類，100 種類，1,000 種類という部分セットを作成した．これらの部分セットは，化合物の大きさや配座の数，フラグメントの種類数などに偏りが生まれないようそれぞれ 3 セットずつ生成した．それぞれのセットに対して配座生成を行った結果の化合物の配座の総数，またそれらの化合物をフラグメントに分割した際のフラグメントの種類数・総数は表 1 のようになった．

表 1 入力化合物のデータセット

化合物		フラグメント	
種類数	配座総数	種類数	総数
	579	35	3,559
10	861	34	6,929
	924	42	7,817
	7,521	186	74,004
100	7,522	192	57,620
	9,326	196	80,221
	79,846	1,011	653,834
1,000	81,218	968	658,274
	82,583	1,007	674,511
2,886	241,034	2,028	1,973,311

4.2.2 標的タンパク質

今回の速度評価実験での標的タンパク質には， Protein Data Bank (PDB) [21] より，インフルエンザの標的タンパク質として知られている [22] Influenza neuraminidase N1 (PDBID : 2HU4) を用いた．

4.3 実行環境

実験の実行環境を表 2 に示す．なお全ての手法は C++ で実装されており，並列化などは行っておらず 1 CPU コアを用いた速度を計測する．

表 2 実験の実行環境

計算機	TSUBAME 3.0 f_node
CPU	Intel Xeon E5-2680 v4 2.4GHz (14 cores) ×2
メモリ	256 GB RAM
コンパイラ	gcc version 4.8.5 (SUSE Linux)
最適化オプション	-O2

4.4 時間計測方法

4.4.1 計測範囲

本研究では特に，フラグメントグリッドを生成・利用することにより化合物のスコア計算を行う部分の高速化に着目した．そこで，高速化が期待されるその部分に限定して実行時間の計測を行った．

4.4.2 計測方法

C++ の `std::chrono::system_clock` を用いて実行時間の計測を行った．各ケースにおいて 3 回計測を行いその中央値をとった．また入力化合物が 10 種類，100 種類，1,000 種類のケースは，さらに 3 つのセットの計測結果の中央値をとった．

5. 実験結果

以下に示す結果は全て

- 探索空間の一辺の長さ：10 Å
- 各グリッドの一辺の長さ：20 Å
- フラグメントグリッドを保持するメモリ：8,000 MB

での実験結果である．

5.1 提案手法を部分的に適用した手法との実行速度の比較

提案手法の実行速度を，提案手法を部分的に適用した手法と比較した．その結果を表 3 に示す．

表 3 提案手法を部分的に適用した手法との実行時間の比較 [秒]

化合物	メモリ戦略 オフライン		メモリ戦略 オンライン	
種類数	並び替え 有	並び替え 無	並び替え 有	並び替え 無
10	**176**	**176**	**176**	**176**
100	1,169	**1,153**	1,236	1,263
1,000	**8,528**	9,980	10,123	12,771
2,886	**20,796**	28,198	24,274	36,813

この結果より，本研究で高速化のために用いた化合物の評価順序の変更とメモリ戦略の最適化という 2 つの手法は，本研究の実験における最大ケースにおいて 1.77 倍の高速化を実現していることが示された．

5.2 フラグメントを用いない手法との実行速度の比較

提案手法の実行速度を，フラグメントを用いずに原子グリッドのみを用いる手法と比較した．その結果を表 4 に示す．

表 4 フラグメントを用いない手法との実行時間の比較 [秒]

化合物	提案手法	比較対象
種類数	（フラグメント利用）	（フラグメント不使用）
10	176	**104**
100	1,169	**1,063**
1,000	**8,528**	11,801
2,886	**20,796**	48,650

この結果より，フラグメントグリッドを生成，再利用した提案手法は，フラグメントを用いず原子グリッドのみ用いた手法に比べ，本研究の実験における最大ケースにおいて 2.34 倍の高速化を実現していることが示された．

6. 考察

6.1 提案手法の入力化合物数による速度変化

5.1 節に示した実験結果より，入力化合物数の増加に伴い 1 化合物あたりの評価速度が向上していることがわかる．これは化合物数の増加により共通なフラグメントの数が多くなり，フラグメントグリッドの再利用がより積極的に行われた結果であると考えられる．共通なフラグメントの数はさらに多数の化合物群に対しても増えていくことが報告されているので [4]，この評価速度は入力化合物数の増加に伴い引き続き向上することが期待できる．

6.2 提案手法を部分的に適用した手法との実行速度の比較

6.2.1 エネルギースコア計算を行う原子の数と実行時間

3.6.1 節で述べた通りフラグメントのエネルギースコアはフラグメントの持つ原子のスコアの総和であるので，フラグメントグリッド生成にかかる時間はフラグメントの原子数に比例すると考えられる．この検証のため，今回の実験で得られたデータを用いてスコア計算を行った原子数と実行時間の関係をプロットをしたグラフを図 **6** に示す．

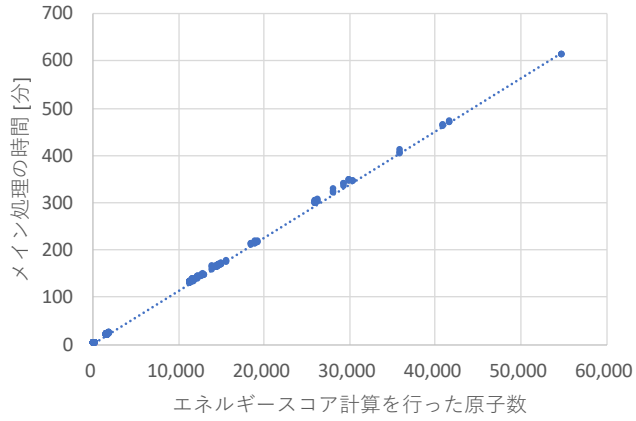

図 6 スコア計算を行った原子数と実行時間のグラフ

今回実行時間を計測した範囲では主にフラグメントグリッドの生成とフラグメントグリッドを用いた化合物の評価という 2 つの処理を行っているが，化合物の評価にかかる時間はフラグメントグリッドの生成に比べて十分に短いので，この結果からフラグメントグリッドを生成するのにかかるコストはフラグメントの原子数に比例すると考えられる．これにより，3.6.2 節においてオフラインキャッシュ問題に帰着させる際のフラグメントグリッドの生成コストに，フラグメントの原子数を用いたことに対する正当性が示されたと言える．

7. 結論

7.1 本論文のまとめと結論

本研究では，化合物が共通の部分構造を持つことが多いことに着目し，計算結果を再利用しながらドッキングシミュレーションを行うツールを開発した．部分構造の計算結果はメモリサイズが膨大になり，全ての部分構造の計算結果をメモリ上に保持することは非常に困難であるため，(1) 化合物の評価順序の変更 (2) メモリに保存する計算結果の最適化 という操作を行い保持するデータを適切に選択することで効率的な再利用を実現した．

計算速度評価の結果，今回行った実験の内入力化合物数が最大のケースにおいて，計算結果の再利用効率の改善前に比べて化合物のエネルギースコア計算の実行速度を 1.77 倍高速にした．また，部分構造には着目せずに原子単位で計算結果を再利用してドッキングシミュレーションを行う手法と比較したところ，化合物のエネルギースコア計算の実行速度を 2.34 倍高速にした．よって，フラグメントに分割し計算結果を効率よく再利用していく，という手法の有用性が示されたと考えられる．さらに，提案手法においてかかる計算時間は部分構造の再利用効率に強く依存するが，化合物が多くなるとその分共通な部分構造が増えるのでこの高速化率はより化合物数の多いケースに対してさらに向上すると考えられる．

7.2 今後の課題

7.2.1 より多数の化合物に対する実験

本研究の実験では化合物の種類数の最大値は 2,886 種類だったが，これは評価実験として十分と言える数ではない．その原因の 1 つとしては，実験に用いた計算機である TSUBAME 3.0 の実験当時のプログラム実行制限時間が 24 時間となっており，それ以上の連続実行が認められていなかったという点が挙げられる．現在は TSUBAME 3.0 の時間制限が緩和されているので，今後より多数の化合物に対する実験を行う予定である．

7.2.2 評価速度の向上

本研究では，フラグメントグリッドの再利用効率を向上することにより高速化を実現したが，現在用いられている既存のドッキングツールに比べると，まだ十分な高速化には至っていないというのが現状である．コンフォーマードッキングを行う既存のドッキングツールである FRED [8] を例に挙げると，1 化合物あたり 5 秒でドッキング計算が可能であると述べており，本研究の提案手法に比べて 2 倍以上の実行速度を示すことが予想される．よって依然として高速化が課題となり，これに対してはフラグメントグリッドの省メモリ化や探索の枝刈りなどの対策が考えられる．

7.2.3 精度の評価及び改善

本研究では，いくつかの手法と実行速度の比較実験を

行ったが，予測結合ポーズがどれだけ正解ポーズに近いか，または結合する化合物と結合しない化合物をエネルギースコアを用いて分類する，といったような精度を評価する実験は行っていない．ただ精度に関しては，大きなフラグメントの回転の際に少し回転角がずれただけで中心から遠い原子の座標は大きくずれてしまうので，フラグメントを使わずに原子グリッドでエネルギースコアを算出するプログラムなどに比べると精度が落ちることが予想される．制度低下の対策としては，回転方向の細分化や重み付き平均などによる補間などを行うことが考えられる．

謝辞 本研究の一部は，JSPS 科研費 (17H01814, 17J06897)，JST CREST「EBD: 次世代の年ヨッタバイト処理に向けたエクストリームビッグデータの基盤技術」(JPMJCR1303), JST リサーチコンプレックス推進プログラム，文部科学省地域イノベーション・エコシステム形成プログラム，AMED BINDS (JP17am0101112) の支援を受けて行われた．

参考文献

[1] J. J. Irwin and B. K. Shoichet. ZINC − a free database of commercially available compounds for virtual screening. *Journal of Chemical Information and Modeling*, 45(1):177–182, 2005.

[2] G. Sliwoski, S. Kothiwale, J. Meiler, and E. W. Lowe. Computational methods in drug discovery. *Pharmacological Reviews*, 66(1):334–395, 2014.

[3] X.-Y. Meng, H.-X. Zhang, M. Mezei, and M. Cui. Molecular docking: A powerful approach for structure-based drug discovery. *Current Computer-Aided Drug Design*, 7(2):146–157, 2011.

[4] K. Yanagisawa, S. Komine, S. D. Suzuki, *et al.* Spresso: an ultrafast compound pre-screening method based on compound decomposition. *Bioinformatics*, 33(23):3836–3843, 2017.

[5] K. Yanagisawa, S. Komine, R. Kubota, M. Ohue, and Y. Akiyama. Optimization of memory use of fragment extension-based protein-ligand docking with an original fast minimum cost flow algorithm. *Computational Biology and Chemistry (In Proc. APBC2018)*, (in press).

[6] Z. Zsoldos, D. Reid, A. Simon, S. B. Sadjad, and A. P. Johnson. eHiTS: A new fast, exhaustive flexible ligand docking system. *Journal of Molecular Graphics and Modelling*, 26(1):198–212, 2007.

[7] D. S. Goodsell and A. J. Olson. Automated docking of substrates to proteins by simulated annealing. *Proteins*, 8(3):195–202, 1990.

[8] M. McGann. FRED pose prediction and virtual screening accuracy. *Journal of Chemical Information and Modeling*, 51(3):578–596, 2011.

[9] O. Trott and A. J. Olson. AutoDock Vina: Improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. *Journal of Computational Chemistry*, 31(2):455–461, 2010.

[10] R. A. Friesner, J. L. Banks, R. B. Murphy, *et al.* Glide: a new approach for rapid, accurate docking and scoring. 1. method and assessment of docking accuracy. *Journal of Medicinal Chemistry*, 47(7):1739–1749, 2004.

[11] M. Rarey, B. Kramer, T. Lengauer, and G. Klebe. A fast flexible docking method using an incremental construction algorithm. *Journal of Molecular Biology*, 261(3):470–489, 1996.

[12] I. D. Kuntz, J. M. Blaney, S. J. Oatley, R. Langridge, and T. E. Ferrin. A geometric approach to macromolecule-ligand interactions. *Journal of Molecular Biology*, 161(2):269–288, 1982.

[13] R. Rosenfeld, S. Vajda, and C. DeLisi. Flexible docking and design. *Annual Review of Biophysics and Biomolecular Structure*, 24(1):677–700, 1995.

[14] P. C. D. Hawkins, A. G. Skillman, G. L. Warren, B. A. Ellingson, and M. T. Stahl. Conformer generation with OMEGA: Algorithm and validation using high quality structures from the protein databank and cambridge structural database. *Journal of Chemical Information and Modeling*, 50(4):572–584, 2010.

[15] 小峰駿汰, 石田貴士, 秋山泰. フラグメント伸長型タンパク質-化合物ドッキングのビームサーチによる高速化. **情報処理学会研究報告 バイオ情報学** *(BIO)*, 2015-BIO-42(62):1–8, 2015.

[16] N. M. O'Boyle, M. Banck, C. A. James, *et al.* Open Babel: An open chemical toolbox. *Journal of Cheminformatics*, 3(1):33, 2011.

[17] D. Weininger. SMILES, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences*, 28(1):31–36, 1988.

[18] A. López-Ortiz and A. Salinger. Minimizing cache usage in paging. In *Approximation and Online Algorithms*, pages 145–158, 2013.

[19] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows*. Elsevier, 2014.

[20] H. Pirsiavash, D. Ramanan, and C. C. Fowlkes. Globally-optimal greedy algorithms for tracking a variable number of objects. In *Proc. of CVPR 2011*, pages 1201–1208, 2011.

[21] P. W. Rose, A. Prli, A. Altunkaya, *et al.* The RCSB protein data bank: integrative view of protein, gene and 3D structural information. *Nucleic Acids Research*, 45(D1):271–281, 2017.

[22] R. J. Russell, L. F. Haire, D. J. Stevens, *et al.* The structure of H5N1 avian influenza neuraminidase suggests new opportunities for drug design. *Nature*, 443(7107):45–49, 2006.