Vol.2016-SE-193 No.5
2016/7/13

メタヒューリスティクスを用いた集約可能コードクローン量の推定

石津 卓也[1,a)]　吉田 則裕[2]　崔 恩瀞[3]　井上 克郎[1]

概要：コードクローンの集約はソフトウェア保守の対象となるソースコードを減少させる．しかし，集約に要する期間，費用の推定が困難であるために集約の計画が立ちにくいという問題がある．研究では，集約可能コードクローン量の推定が開発者の集約を支援することを目的としている．オーバーラップしているコードクローンから集約する組み合わせを推定するためにメタヒューリスティクスを用いた．

キーワード：コードクローン　メタヒューリスティクス

Estimating the amount of refactorable clones based on metaheuristic algorithms

Takuya Ishizu[1,a)]　Norihiro Yoshida[2]　Eunjong Choi[3]　Kathuro Inoue[1]

1. まえがき

ソフトウェアの保守はソースコードの行数に依存して困難なものになる．行数を増大させる要因の１つとしてコードクローンが挙げられる [3][6]．コードクローンに対する保守作業として集約が挙げられる．集約とは，互いにコードクローンとなっているコード片の集合（以下，クローンセットと呼ぶ）に対して，クローンセットに含まれるコードクローンを１つのクラスやメソッドにまとめることである [1][3]．また，集約可能なコードクローンを集約して減る行数を集約可能コードクローン量と呼ぶことにする．集約を行うことで，保守の対象であるコードクローンを除去できる．そのため，コードクローンを集約することで，保守するソースコードの行数が減少して，ソースコードの保守が容易になる．コードクローンが多く，開発者が判断しなければならない機会が多いソースコードは，集約により減らせるプログラム全体の集約可能コードクローン量がわかりにくかったり，その判断にかかる費用や期間が見えにくかったりするので，集約の計画が立たなくて開発を敬遠しがちである．そこで，コードクローンに対する集約による効果を事前に推定して集約を支援することを考える．開発者は事前に集約可能コードクローン量を知ることで率先して集約を実行する．コードクローンがオーバーラップしている場合，一方のコードクローンを集約するとオーバーラップをしていたコードクローンはコード片が部分的に欠損するため，集約が不可能になる．よって，集約するコードクローンの組み合わせに対して，全通りを計算して最善だった順番を採用することが考えられる．しかし，実際のソースコードにおいて，コードクローンを選択する組み合わせは時間的に計算するのが困難なほど現実的ではない．そこで，効率的にどのコードクローンを集約するのかを求める探索手法が必要である．本研究では，適用実験としてオーバーラップを考慮したコードクローンを含むソフトウェアを対象にアルゴリズムを用いて集約可能なコードクローン量を計算して，一般的にメタヒューリスティクスで用いられる４つのアルゴリズムを比較している．メタヒューリスティクスであるアルゴリズムには貪欲法や山

[1] 大阪大学
Osaka University
[2] 名古屋大学
Nagoya University
[3] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology
[a)] t-ishizu@ist.osaka-u.ac.jp

登り法，焼きなまし法，遺伝的アルゴリズムが代表として挙げられる [7]．また，Edwards らは，CCFinderX のようなコード検出ツールの結果を用いて，集約可能なコードクローンの大きさを算出するツール CCM のアルゴリズムを考案した [2]．本研究では，メタヒューリスティクスを用いたコードクローン集約量の推定値を CCM の算出した集約量に対する近似率を求めることで，推定値がどれくらい正確なのかを調査する．

2. 背景

2.1 コードクローン

コードクローンとは，プログラムテキスト中の同一，あるいは，類似したコードの断片を意味する [6]．また，一般的に，同一コードクローンから構成される集合をクローンセット (Cloneset) と呼ぶ．コードクローンの存在はソフトウェアの保守作業を困難なものにすると言われている．例えば，あるコード片を修正する場合，コードクローンに関しても同様の修正が行われる可能性がある．コードクローンに気づかずにソースコードの修正を疎かにすると，ソースコードにバグが混入する可能性がある．そのため，コードクローンの存在を知ることはソフトウェアの保守の観点からすると重要なことである．しかし，ソフトウェアの規模が大きい場合，開発者がソースコードを読んですべてのコードクローンの存在を知ることは非効率的であるため，一般的にはコードクローンを自動で検出するツールが用いられる．コードクローンに対する保守作業として集約が挙げられる．集約とは，クローンセットに対して，クローンセットに含まれるコードクローンを 1 つのクラスやメソッドなどのサブルーチンにまとめて，コードクローンを呼び出し文に置き換えることである．図 1 は，コードクローンが 1 つのサブルーチンに集約されて，コードクローンを呼び出し文に置き換えている様子を表している．集約を行うことで保守の対象であるコードクローンを除去できる．そして，コードクローンの除去により，ソースコードの同時修正を行う手間が省ける．そのため，コードクローンの集約はソースコードの保守作業の労力を軽減する役割を持つ．

2.1.1 コードクローンのオーバーラップ

コードクローンのオーバーラップとは，複数のコードクローンのコード片が部分的に重複していることを指す．コードクローンがオーバーラップしている場合，一方のコードクローンを集約するとオーバーラップしていたコードクローンはコード片が部分的に欠損するため，集約が不可能になる．図 2 は任意のソースコードに含まれる 2 つのコードクローン a1，b1 がオーバーっらぷしている例を示している．青で塗られたコード片 a1 と赤で塗られたコード片 b1 はそれぞれ異なるクローンセットに属しており，2 つのコードクローンが部分的に重複している様子がわかる．

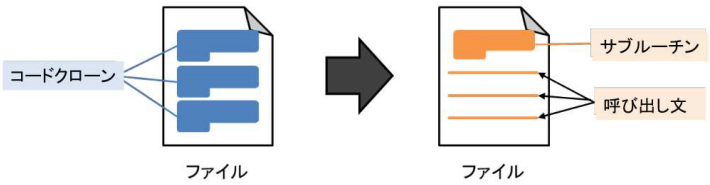

図 1 例：コードクローンの集約
Fig. 1 An Example of Refactoring Code Clone

2.1.2 コードクローンの集約

コードクローンの集約を行うには，コードクローンをどこに集約するか，あるいは，変数名や変数の型，予約語といったコードクローンの差異をどう吸収するか，開発者が逐一ソースコードを読んで，コードクローンが集約の対象になりうるのか判断をしなければならない [3]．コードクローンが多く，開発者が判断しなければならない機会が多いソースコードは，集約により減らせるプログラム全体の集約可能コードクローン量がわかりにくかったり，その判断にかかる費用や期間が予想しずらかったりするので，集約の計画が立たなくて開発を敬遠しがちである．そこで，コードクローンに対する集約による効果を事前に推定して集約を支援することを考える．開発者は事前に集約可能コードクローン量を知ることで率先して集約を実行する．オーバーラップを考慮したコードクローンの集約可能コードクローン量を推定するには，集約するコードクローンの順番によって集約可能コードクローン量が変化することを考慮す る必要がある．理想は集約するコードクローンの組み合わせに対して，全通りを推定して最 善だった順番を採用すればいい．しかし，実際のソースコードにおいて，コードクローンを 選択する組み合わせは時間的に計算するのが困難なほど現実的ではない．そこで，集約するコードクローンを選ぶ順番を求める問題に対して，既存の探索的手法の適用が考えられる．

2.2 メタヒューリスティクス

メタヒューリスティクスとは，特定の問題に依存しない，組み合わせ最適化問題における， 近似解を求める解法をもつアルゴリズムの基本的な枠組みのことである．アルゴリズムとし て，局所的な探索に基づくアルゴリズムや，大域的な，個体群に基づくアルゴリズムが挙げ られる．局所的な探索に基づくアルゴリズムとは，現在の解から近傍を求めて，現在の解を 近傍の解に更新する作業を繰り返すことで近似解を求めるアルゴリズムである．大域的な個体群に基づくアルゴリズムとは，解の候補を生物の個体群に見立て最適解を求めるアルゴリズムである．このアルゴリズムでは，個体が次の世代に生き残るのかを決定する関数を工夫して生物の生殖や自然淘汰，突然変異などを再現する．

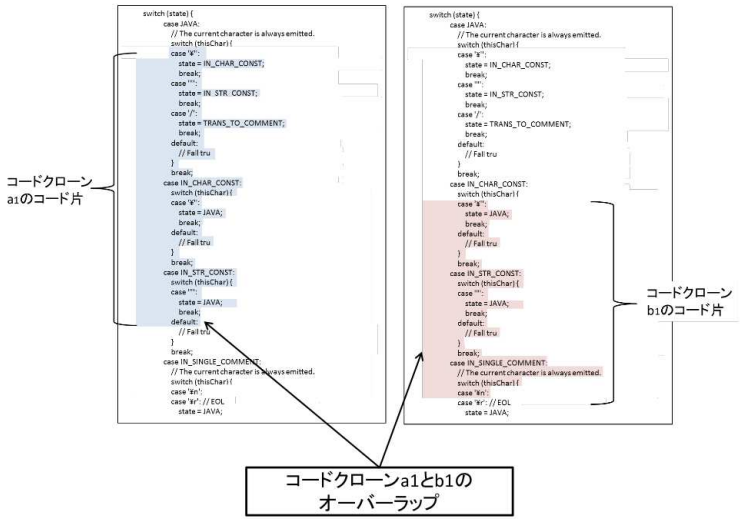

図 2 コードクローンがオーバーラップする例
Fig. 2 An Example of Overlapping Code Clones

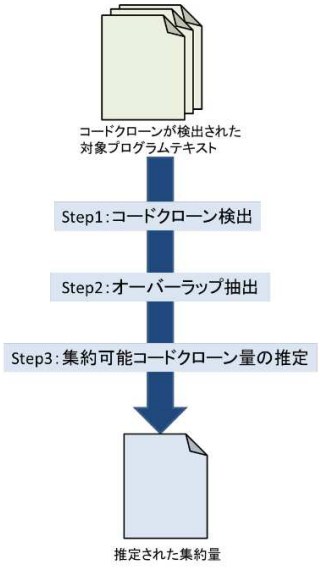

図 3 提案手法の流れ
Fig. 3 An Overview of the Proposed Method

これを繰り返して残った個体が最適解に近似する．

本研究では，コードクローンの集約可能コードクローン量をメタヒューリスティクスで一 般的に用いられる 4 つのアルゴリズムに適用して，それらの評価を比較する．メタヒューリ スティクスで用いる 4 つのアルゴリズムとして貪欲法 (Greedy Algorithm)[5]，山登り法 (Hill Climbing)[4]，焼きなまし法 (Simulated Annealing)[8]，遺伝的アルゴリズム (Genetic Algorithm)[9] が代表的なアルゴリズムとして挙げられる．HC や SA は局所的な探索アルゴリズムであり， GA は大域的な，個体群に基づく探索アルゴリズムである．

本研究では，遺伝的アルゴリズムとして，多目的最適化問題によく用いられる NSGAII （Non-dominated Sorting Genetic Algorithms-II）アルゴリズムを使用した．多目的最適化問題とは，目的関数を 2 つ以上持っている最適化問題のことである．本研究で は，目的関数が集約可能コードクローン量の 1 つしかない単一目的最適化問題を扱うが，NSGAII は目的関数が 1 つでも適用可能である．また，NSGAII アルゴリズムの 使用にあたって，Java のフレームワークとしてMOEA を利用した．MOEA では，多目的最適化問題に対応する，遺伝的アルゴズムや遺伝的プログラミングがオープン ソフトウェアとして組み込まれている．

3. 提案手法

本節では，集約可能コードクローン量を推定する手法について説明する．図 3 は，提案手法の流れを表している．本研究では，メ タヒューリスティクスの 4 つのアルゴリズムの中から最適なアルゴリズムを決定することを 目的にしている．そのため，各アルゴリズムを適用して得られたコードクローンの集約可能 コードクローン量を比較する．本手法では，比較を行うために必要なコードクローンの集約 可能コードクローン量を求める． 本研究における，オーバーラップを含む，クローンセット合成を行ったコードクローンの集約可能コードクローン量の推定する手法の流れを説明する．

Step1 CCFinder のコードクローン検出結果を入力する．
Step2 入力したコードクローン間に発生したオーバーラップを抽出する．
Step3 メタヒューリスティクスなアルゴリズムを用いて，コードクローン全体の集約可能コードクローン量を推定する．

本研究で提案する集約可能コードクローン量を推定する手法を図示した．Step1 では，コードクローンの集約可能コードクローン量を推定するソースコードに対して，字句解析に基づく CCFinder によるコードクローンを検出を行い，その出力ファイルを入力に用 いる．Step2 では，入力した位置情報を基に，どのコードクローン同士がオーバーラップし ているかを判別する．Step3 では，検出されたすべてのコードクローンに対する，集約可能 コードクローン量を推定する．また，無向グラフ G をもつクローンセットの集合に対して， 各クローンセットを分割する．

3.1 オーバーラップの検出

Step1 では，集約可能コードクローン量を推定するソースコードに対して，コードクローンを検出を行った．本手法では，字句単位の解析コードクローン検出ツール CCFinder を用いた．コードクローン検出に字句単位の解析ツールを用いる理由は，字句単位のような検出粒度が細かい検出法の方が集約可能なコードクローンを検出しやすいためであ

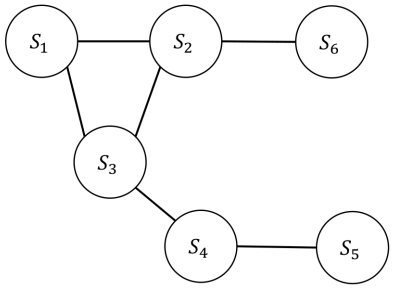

図 4 コードクローン間のオーバーラップを表現する無向グラフの例
Fig. 4 An Example of an Undirected Graph that Represents Overlaps Among Clone Sets

る．メソッド単位やクラス単位の検出では集約可能なコードクローンは発生しにくい．また，CCFinder は様々な大規模ソフトウェアに適用され，その有用性が確認されているため，本手法で使用した．

3.2 オーバーラップの抽出

Step2 では，Step1 で検出したコードクローンをもとに，どのコードクローン同士がオーバーラップしているかを判別して，オーバーラップしているクローンセットを抽出する．同時に図 4 のような無向グラフ G を構成する，クローンセット S_1 から S_6 までを頂点とし，各頂点間に接続する無向辺に対して，隣接する 2 頂点はオーバーラップの関係にある．例えば，図 4 の場合，クローンセット S_1 は 2 つのクローンセット S_2，S_3 とオーバーラップしていることがわかる．

2 つのコード片 a_1，a_2 が重複する条件は次の 2 つである．ただしコード片 a_1 の開始行番号はコード片 a_2 の開始行番号より先にあるものとする．

(1) 2 つのコード片が同一ファイル内に存在する．
(2) コード片 a_1 の行番号 ≤ コード片 a_2 の行番号 ≤ コード片 a_1 の終了行番号を満たす．

以上，2 つの条件を満たす 2 つのコード片 a_1，a_2 はオーバーラップしている．また，2 つのコード片 a_1，a_2 に関して，$a_1 \in S_1$，$a_2 \in S_2$ を満たすクローンセット S_1，S_2 が存在するとする．このとき，$S_1 \neq S_2$ ならば 2 つのクローンセット S_1，S_2 はオーバーラップしている．また $S_1 = S_2$ ならば，そのクローンセットは自分自身とオーバーラップしている．無向グラフ G に属する任意のクローンセットは，無向グラフ G に属する他のクローンセットの少なくとも 1 つ以上とオーバーラップしている．

3.3 集約可能コードクローン量の推定

オーバーラップしているコードクローンに対して，集約可能コードクローン量を推定する場合，最大の集約可能コードクローン量を推定するためにはコードクローンを集約する順番を求めなくてはならない．理想的な解法としては，集約するコードクローンから構成される集合の全組み合わせに対して集約可能コードクローン量を推定して，最大の集約可能コードクローン量となる集合を解とすればよい．しかし，全組み合わせに対して，集約可能コードクローン量を推定するのは現実的ではない．本研究では，そのような問題に対して，メタヒューリスティクスを用いてコードクローンの集約可能コードクローン量の推定を行う．

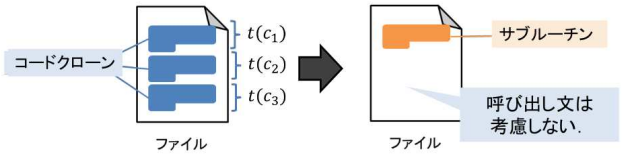

図 5 集約可能コードクローン量の推定の概要
Fig. 5 An Overview of Estimating the Amount of Code Clone

本研究では，コードクローンの集約手法について特定の手法を選択しない．いずれの集約手法を選択しても，コードクローンであるコード片をプログラムに 1 つ残して，その他のコード片に関しては集約により消去されるという考え方を適用する．そのため，特定の集約手法に依存しない，コードクローンの集約可能コードクローン量を推定する．

集約可能コードクローン量を評価する指標として，ソースコードの行数やトークン数が候補として挙げられる．本研究では，開発者が集約の実行を率先して実行するかどうかを判断する基準としてコードクローンの集約可能コードクローン量を推定する．そのため，開発者が直感的に数値で評価しやすいと考えられるソースコードの行数を評価の指標とする．

任意のクローンセット S_k に関して，S_k 内に含まれる集約可能なコードクローンの個数を n，i 番目のコードクローンを $c_{k,i}$ とする．またコードクローン $c_{k,i}$ のコード片の行数を $t(c_{k,i})$ とする．

クローンセット S_k の集約を考える．コード片 1 つを残してその他のコード片を消去するのだから，クローンセット S_k の集約可能コードクローン量 $L(S_k)$ は次のように推定される．

$$L(S_k) = \frac{n-1}{n} \sum_{i=0}^{n} t(c_{k,i})$$

図 5 はサブルーチンに集約されるコードクローンの行数が $t(c_i)$ で与えられている様子を表している．ただし，本研究では，呼び出し文の行数を考慮していない．これは，集約可能コードクローン量が集約される行数だけではなく，集約されるトークン数を表すこともあるためである．

3.4 オーバーラップを含む集約可能コードクローン量の推定

複数のクローンセットのオーバーラップしている状態は

図 4 のような無向グラフによって表現できる．クローンセット S_1 から S_6 までを頂点とし，各頂点間に接続する無向辺に対して，隣接する 2 頂点はオーバーラップの関係にある．例えば，図 4 の場合，クローンセット S_1 は 2 つのクローンセット S_2，S_3 とオーバーラップしていることがわかる．

図 4 のような無向グラフで隣接する複数のクローンセットからなる集合を G とする．また，集合 G に含まれる任意のクローンセットに関して，互いにオーバーラップしないようにクローンセットを選択してなす集合を H とする．すなわち，集合 H に含まれるクローンセットに対する無向グラフでは，無向辺を持たない．例えば，図 4 の場合，集合 G と，集合 H の一例として挙げる H_1，H_2 は次のようになる．

$$G = \{S_1, S_2, S_3, S_4, S_5, S_6\}$$

$$H_1 = \{S_1, S_4, S_6\}$$

$$H_2 = \{S_3, S_5, S_6\}$$

2 つの集合 H_1，H_2 に含まれるクローンセットは互いにオーバーラップしていないので，クローンセット S_k の集約可能コードクローン量を $L(S_k)$ とすると，2 つの集合の集約可能コードクローン量 $L(H_1)$，$L(H_2)$ は次のようになる．

$$L(H_1) = L(S_1) + L(S_4) + L(S_6)$$

$$L(H_2) = L(S_3) + L(S_5) + L(S_6)$$

一般化すると，集合 H の集約可能コードクローン量 $L(H)$ は次のようになる．

$$L(H) = \sum_{S_i \in H} L(S_i)$$

集合 G の集約可能コードクローン量 $L(G)$ について，$L(G)$ の数値が最大となるようなクローンセットの選び方として集合 H_{max} が存在する．集合 H の集合を M とした場合，集合 G の集約可能コードクローン量 $L(G)$ は次のように表す．

$$L(G) = L(H_{max}) = \max_{H \in M}\{L(H)\}$$

3.5 クローンセットの合成

オーバーラップする 2 つのクローンセットのコードクローン集約量は，一方のクローンセットの集約を行い，もう一方のクローンセットの集約は行わないことで求めることができる．しかし，集約を行わないクローンセットはオーバーラップをしているコード片を除けば，残ったコード片を組み合わせることで新たなコードクローンを作り，集約することが可能になる．本研究では，2 つのクローンセットを合成することでコードクローン集約量を求めることにした．図 6 は 2 つのオーバーラップしたクローンセット A，B を合成して新たなクローンセット C を作る様子を示している．クローンセット A はクローンセット B とファイル B 上でオーバーラップしているために集約ができなかった．そこで，クローンセット A の各コードクローンに関して，クローンセット B とオーバーラップしている部分としていない部分に分割する．こうすることで，クローンセット A を分割でき集約が可能になる．このようなクローンセット A の集約はクローンセット B とのみオーバーラップしている想定で行われている．しかし，仮にクローンセット A がクローンセット B 以外の他のクローンセットとオーバーラップしている場合も考えられる．そのような場合でも，クローンセット A の図 6 のような分割はクローンセット B の間しか行らないため，分割後のクローンセット A を独立したクローンセットにするのではなく，オーバーラップしていたクローンセット B と合成した新たなクローンセットを作成する．また，本研究では．クローンセット A とオーバーラップしているクローンセットが 1 つではない場合，クローンセット A が合成を行う対象はただ 1 つのクローンセットとしている．そのクローンセットはクローンセット A と合成して最大の集約量になるようなクローンセットを選択する．クローンセット S_A とオーバーラップしているクローンセットの集合 G_A について，$S_X (\in G_A)$ を考える．このとき，クローンセット A とのご異性による集約量は次の数式で求められる．

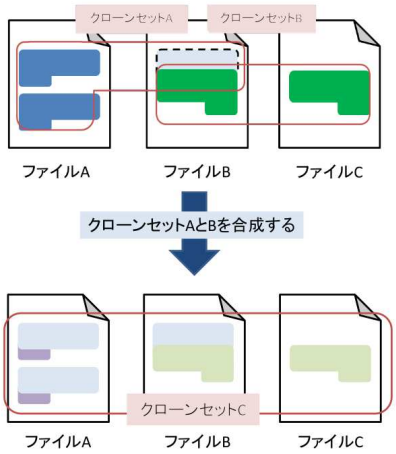

図 6 クローンセットの合成例
Fig. 6 An Example of Marging Clone Sets

$$L(S_A) + L(S_X) - \sum_{\substack{0 \le i \le |S_A| \\ 0 \le j \le |S_X|}} (c_{Ai} \cap c_{Xj})$$

この式で求められた集約量が最大となったクローンセット S_X をクローンセット A と合成する．

3.6 探索的手法への適用

集約可能コードクローン量を決める集約の順番を求めるために，SBSE を用いる．そのために，Representation,

Operators, Fitness Function を定義する必要がある．Step3 で，検出されたすべてのコードクローンに対する，集約可能コードクローン量を推定するアルゴリズムは Greedy，HC，SA，GA の 4 つである．

3.6.1 Representation

集約可能コードクローン量を推定する N 個のクローンセット全体に対して，i 番目のクローンセット S_k の状態 $S(i)$ を次のように表す．ただし，i は $N \geq k$ を満たす整数で，クローンセット同士を互いに識別するための ID である．

$$S(i) = Overlap(i) = p, q, r$$

$Overlap(i)$ は i 番目のクローンセット S_i とオーバーラップしているクローンセットの ID 集合を返す．例の場合，p，q，r 番目のクローンセットはそれぞれ i 番目のクローンセットとオーバーラップしている．このとき，クローンセット S_i は無向グラフ G に属しているものとする．ただし，無向グラフ G に属する任意のクローンセットは，無向グラフ G に属する他のクローンセットの少なくとも 1 つ以上とオーバーラップしている．無向グラフ G から任意のクローンセットを適当に選択し，それらのクローンセットを要素に持つクローンセットの新たな集合 H を考える．ただし，集合 H に含まれる任意のクローンセットは，集合 H に含まれるクローンセットとオーバーラップしていないものとする．

R は図 7 のようにバイナリ表現で考える．無向グラフ G に含まれる n 個のクローンセットをそれぞれ R で表されるバイナリの各位置に対応付ける．集合 H に含まれるクローンセットに対応する位置を 1，含まれないクローンセットに対応する位置には 0 を表示する．

Initial Representation

Greedy 法の初期値の決定は次の手順で行う．すべてのクローンセットを集約可能コードクローン量の大きい順番にソートして，集約可能コードクローン量が大きいクローンセットから優先的に集約をしていく．ただしすでに集約したクローンセットとオーバーラップしていたために集約不可なクローンセットは集約しない．本研究におけるメタヒューリスティクスの初期値は Greedy 法で決定した集約の順番と同じとする．これは Greedy と比べて他のアルゴリズムの集約可能コードクローン量がどれだけ増加するのかを比べるためである．

3.6.2 Operators

集合 $H(k)$ に対する操作 Opr は次のように表せる．

集合 $H(k)$ に対する集合 H（k+1）の選び方はアルゴリズムに依存する．例えば，山登り法の場合，集合 $H(k+1)$ は集合 $H(k)$ の近傍である．または，遺伝的アルゴリズムの場合は，集合 $H(k+1)$ は集合 $H(k)$ を基に，生殖や自然淘汰，突然変異を起こすことで残る次の世代である．

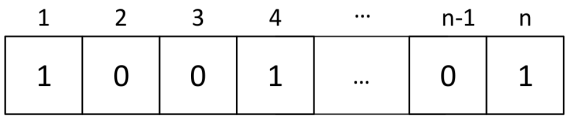

図 7 クローンセット集合の場合の Representation ト
Fig. 7 Representation for a Set of Clone Sets

$$R(k+1) = Opr(R(k)) = Opr(H(k))$$

HC

$R(k)$ の近傍を選ぶアルゴリズムは以下のとおりである．

Step1$R(k)$ の要素の中から任意の，0 である i 番目の要素を選ぶ．

Step2i 番目の要素を 1 にする．

Step3i 番目の要素とオーバーラップしているクローンセットと対応している要素のうち，1 である要素すべてを 0 にする．

Step4$R(k)$ の要素の中から，集約しても他のクローンセットとオーバーラップしないクローンセットに対応する要素をすべて 1 にする．

Step5得られた近傍の中から Fitness Function の評価に従い，$R(k+1)$ を返す．

Step1 で選択される要素は，$R(k)$ 中の要素 0 の数だけ存在する．また，Step3 の実行により，オーバーラップしているクローンセットは解消される．Step4 は Step3 により 0 になった要素とオーバーラップしていたクローンセットが集約可能になっている可能性があるためである．ただし，1 つのクローンセットの集約を解除することで 2 つ以上のクローンセットが集約可能になっていた場合，2 つ以上のクローンセット同士がオーバーラップする可能性がある．このとき，クローンセット間でオーバーラップが発生しないようなクローンセットの全組み合わせを近傍として与える必要がある．

SA

$R(k)$ の近傍の選択は HC と同じでアルゴリズムを用いる．ただし Step1 に対して，SA は乱数を用いて，ただ 1 つの要素を選択する．また，Step5 に対して，SA では遷移確率というパラメータを用いる．遷移確率とは $R(k)$ が次の状態 $R(k+1)$ に遷移する際に，最適解に比べて評価値が離れた場合でも，より悪い状態を採択する確率である．本研究では，遷移確率は 0.1 から 0.9 までを 0.1 刻みで実行した．4 章で示す SA の集約可能コードクローン量は，本実験では遷移確率による大きな差異はないためそれらの平均値としている．

GA

MOEA は Java のフレームワークで広く用いられるため，GA には多目的最適化問題に対して有効な，MOEA に搭載されている NSGAII アルゴリズムを使用する [*1]．変更可能なパラメータは評価回数，個体群数，交叉率，突然変異率の 4 つである．評価回数は個体群が世代交代を再現する回数のことである．本研究では，2000 回で実行した．個体群数とは，1 世代に生存している個体群の数である．本研究では MOEA フレームワークのデフォルト値である 100 とした．次に，交叉率とは世代交代の際に個体群を交配するかしないかを決める確率のことである．本研究では，デフォルト値である 1.0 とした．最後に，突然変異とは個体群に含まれる個体を突然変異させる確立である．本研究では，デフォルト値である突然変異率は無向グラフ G に含まれるクローンセット数の逆数とした．個体群数，交叉率，突然変異率に対して，デフォルト値を使用した理由は，これらの値を動かしてもあまり結果に影響せず，代表としてデフォルト値を使用しても問題がなかったためである．

3.6.3 Fitness Function

集合 $H(k)$ に属するクローンセット全体の集約可能コードクローン量が大きいほど，より多く集約されたと見做せる．そのため，評価値は集合 $H(k)$ に属するクローンセット全体の集約可能コードクローン量とする．評価関数 $F(R(k))$ は，集合 H に属するクローンセット S_i の集約可能コードクローン量の総和である．

$$F(R(k)) = F(H(k)) = L(H(k)) = \sum_{S_i \in H} L(S_i)$$

4. 適用実験

本章では，本研究の手法に適した，最大の集約可能コードクローン量を示すアルゴリズムが何であるのかを確認するために，適用実験を行った．

4.1 対象データ

本研究では, 3 つのプロジェクト Apache Ant, ArgoUML, The Apache Xerces(以下 Xerces) のソースコードに含まれるコードクローンに対して集約可能量を推定した．3 つのプロジェクトは広く知られるオープンソースウェアで，コードクローンに関する研究の対象プロジェクトとしてよく用いられる．各ソースコードの行数と，含まれるコードクローンが占める行数とオーバーラップしているコードクローンが占める行数，また，それらが占める割合を表 1 にまとめた．最もコードクローンが占める行数が多いプロジェクトは ArgoUML で，最も少ないプロジェクトは Apache Ant である．また，各ソースコードで検出されたコードクローン，クローンセットの個数をまとめている．Apache Ant はコードクローン数，クローンセット数ともに 3 つの中で最小で，クローンセットを分割するとおよそ 2 倍ほどに増える．コードクローン数，クローンセット数ともに最大は ArgoUML で，クローンセットを分割するとおよそ 3 倍ほどに増える，Xerces も同様に，クローンセットを分割するとおよそ 3 倍ほどに増える．

[*1] http://moeaframework.org/

4.2 各アルゴリズムによる集約可能コードクローン量推定結果の比較

各アルゴリズムごとの集約量コードクローン量について比較する．SA，GA は評価回数に応じて集約可能コードクローン量が変化するので，SA，GA の評価回数 2,000 回を実行する．また，表 2 は，CCM の各プロジェクトに対する集約量の結果である．表 3 は，本研究の集約量の各アルゴリズムごとの結果と表 2 の結果に対する近似率を表している．表 3 をみると，いずれの近似率も 90%を超えていることがわかる．また，最も高い近似率を算出したのは遺伝的アルゴリズムである．プロジェクト Apache Ant, ArgoUML，に対して最も高い近似率を表した．また，最も低い近似率を表したアルゴリズムは Greedy であった．Xerces に対して 91.3%であった．これは，ソースコードに含まれるコードクローン間に発生しているオーバーラップの数が多いことを示している．また，クローンセットの合成であっても完全にはコードクローンを集約しきれていないことを示している．

表 1 各プロジェクトの概要
Table 1 Statistics of each project

	Apatch Ant	ArgoUML	Xerces
ソースコードの行数	265828	389915	238183
コードクローンの行数	23278(8.76%)	59787(15.33%)	51731(21.72%)
オーバーラップしているコードクローンの行数	13442(5.06%)	41434(10.63%)	36697(15.41%)
コードクローン数	5785	21427	17249
クローンセット数 (マージ無)	1740	4795	3792
クローンセット数 (マージ有)	3552	15142	10341

表 2 CCM が計測したコードクローン集約量
Table 2 Amount of Refactorable Code Clones by the CCM

	Apache Ant	ArgoUML	Xerces
CCM	9667	26901	24336

表 3 各アルゴリズムによるコードクローン集約量の推定
Table 3 Estimating the Amount of Refactorable Code Clones by each Algorithm

	Apache Ant	ArgoUML	Xerces
Greedy	9389(97.1%)	25614(95.2%)	22220(91.3%)
HC	9521(98.4%)	26372(98.0%)	23309(95.7%)
SA	9576(99.0%)	26591(98.8%)	23417(96.2%)
GA	9590(99.2%)	26787(99.5%)	23384(96.0%)

4.3 考察

各アルゴリズムの推定に対する振る舞いを考察する．

アルゴリズムの比較：Greedy Algorithm

Greedy アルゴリズムは他 3 つのアルゴリズムに比べて，集約可能コードクローン量が最小であった．また，実装がもっとも単純なアルゴリズムである．コードクローン間でオーバーラップを発生していない場合の集約可能コードクローン量は Greedy で得られる集約可能コードクローン量に一致する．そのため，対象のソースコードから検出したコードクローンの数やオーバーラップの発生が少ない場合には，最適なアルゴリズムだと考えられる．反対に，Xerces や大規模な商用ソフトウェアのようにオーバーラップしているコードクローンが多いと，Greedy で推定される集約可能コードクローン量の誤差は大きくなると考えられる．

アルゴリズムの比較：HC

HC アルゴリズムは Greedy アルゴリズムより推定される集約可能コードクローン量が多く，しかし SA や GA に比べると少ないアルゴリズムである．HC は評価関数が凸関数にできれば大きな効果を持つ，局所的な探索が得意なアルゴリズムである．しかし，極大値にはまると，それを大域的な最大値として推定してしまう弱点がある．本研究のようなクローンセットを集約する順番を 1 つ変えるだけで評価値が大きく変動してしまう問題を抱えるため，SA に比べてあまり良い結果を出すことができなかったと考えられる．

アルゴリズムの比較：SA

SA アルゴリズムは評価回数を基準にした場合，Xerces を対象とした場合，最大の集約可能コードクローン量を示したアルゴリズムである．HC と同様に，SA は極大値にはまると，それを大域的な最大値として推定してしまう弱点がある．しかし，本研究では GA と比較しても，わずかに下回るかほぼ同数の集約量を示す結果となった．Apatche Ant と Xerces ではほぼ同数であったが，ArgoUML ではやや大きく集約量に差があったと考え，GA を週力量の推定に最も適したアルゴリズムだとした．

アルゴリズムの比較：GA

GA アルゴリズムは本研究で Apatche Ant と ArgoUML で最大の集約可能コードクローン量を示したアルゴリズムである．本研究の集約可能コードクローン量の推定にメタヒューリスティクスを適用した場合，局所的な探索手法よりも大局的な探索手法が優れていることを示した．

5. まとめと今後の課題

本研究では，オーバーラップを考慮したコードクローンの集約可能コードクローン量を推定するために，探索的手法を利用した．また，HC や SA，GA などメタヒューリスティクスを，実行時間やコードクローン集約可能コードクローン量を対象に性能を比較した．また，CCM と近似率を比較して推定の正確さを示した．最大の集約可能コードクローン量を示したのは GA で，次いで SA だった．しかし、その差は小さく、適応するソフトウェアのオーバーラップの状態にもよるところが大きいと考えられる．今後の課題として本研究で対象にしたソースコードから検出されたコードクローンが一般的なソフトウェアのコードクローン量に比べて小さいため，より大規模なソフトウェアを対象にした実験が必要になる．

謝辞 本研究は JSPS 科研費 25220003，26730036，15H06344 の助成を受けたものです．また，大阪大学 Ali Ouni 特任助教には本研究で多くの御助言をいただきましたこと，心より感謝します．

参考文献

[1] Bouktif, S., Antoniol, G., Merlo, E., Neteler, M.：A novel approach to optimize clone refactoring activity, *Proceedings of the 8th annual conference on Genetic and evolutionary computation GECCO '06*, pp. 1885–1892 (2006).

[2] Edwards III, B., Wu, Y., Matsushita, M. and Inoue, K.: Estimating Code Size After a Complete Code-Clone Merge, 情報処理学会研究報告，Vol. 2016-EMB-41, No. 3, pp. 1 – 8 (2016).

[3] 肥後芳樹，楠本真二，井上克郎：コードクローン検出とその関連技術，電子情報通信学会論文誌，Vol. J91-D, No. 6, pp. 1465–1481 (2008).

[4] Hiroi, M.: Algorithms with Python ヒューリスティック探索． http://www.geocities.jp/m_hiroi/light/pyalgo28.html(2016 年 2 月 16 日).

[5] Hiroi, M.: 欲張り法 (greedy strategy)． http://www.geocities.jp/m_hiroi/light/pyalgo22.html(2016 年 2 月 16 日).

[6] 井上克郎，神谷年洋，楠本真二：コードクローン検出法，コンピュータソフトウェア，Vol. 18, No. 5, pp. 47–54 (2001).

[7] O'Keeffe, M., Cinneide, M. O.：Search-based refactoring: an empirical study, *Journal of Software Maintenance and Evolution: Research and Practice - Search Based Software Engineering [SBSE]*, Vol. 20, No. 5, pp. 345–364 (2008).

[8] 輪湖純也：Simulated Annealing/SA-概論-． http://mikilab.doshisha.ac.jp/dia/research/person/wako/(2015 年 1 月 31 日).

[9] 同志社大学 知的システムデザイン研究室 遺伝的アルゴリズム研究グループ：遺伝的アルゴリズム概要． http://mikilab.doshisha.ac.jp/dia/research/pdga/research.html(2016 年 1 月 31 日).