

コードクローン検出とその関連技術

肥後 芳樹

楠本 真二

井上 克郎

コードクローンについて

- ソースコード中に存在する互いに一致または類似したコード片
- ソフトウェアの保守を困難にする
(例) コード片Aにバグがあった場合、Aのコードクローンすべてに、同様のバグの疑いがある

コードクローンの検出

- 対象ソフトウェアが大きい
 - チェックする箇所が膨大になる
 - 人がすべての重複部分を認識することは非現実的
- ツールを使った自動的なコードクローン検出が行われる
- ソフトウェアの分析手法としてコードクローン検出技術が注目を集めている

背景と目的

- 背景
 - コードクローンに関する様々な研究が行われ、その情報を効率的に収集することは難しい
 - 目的
 - 現在までのコードクローンに関する研究成果を体系的にまとめる
- ※ この論文が発表されたのは2008年6月

内容

- コードクローンの発生理由
- コードクローンの定義
- コードクローンの検出技術
- 検出技術の比較
- コードクローンの可視化
- コードクローンの集約
- コードクローンの修正支援

内容

- コードクローンの発生理由
- コードクローンの定義
- コードクローンの検出技術
- 検出技術の比較
- コードクローンの可視化
- コードクローンの集約
- コードクローンの修正支援

コードクローンの定義

- 様々な検出手法が提案されているが、どれも異なったコードクローンの定義をもつ
 - 厳密で普遍的な定義は存在しない

- Bellonは、コードクローン間の違いに基づき、3つに分類

[8] S. Bellon, “Detection of software clones,” Technical Report, Institute for Software Technology, University of Stuttgart, 2003, available at <http://www.bauhaus-stuttgart.de/clones/>

[9] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo, “Comparison and evaluation of clone detection tools,” IEEE Trans. Software Engineering, vol.31, no.10, pp.804–818, Oct. 2007.

Bellonによるコードクロンの分類

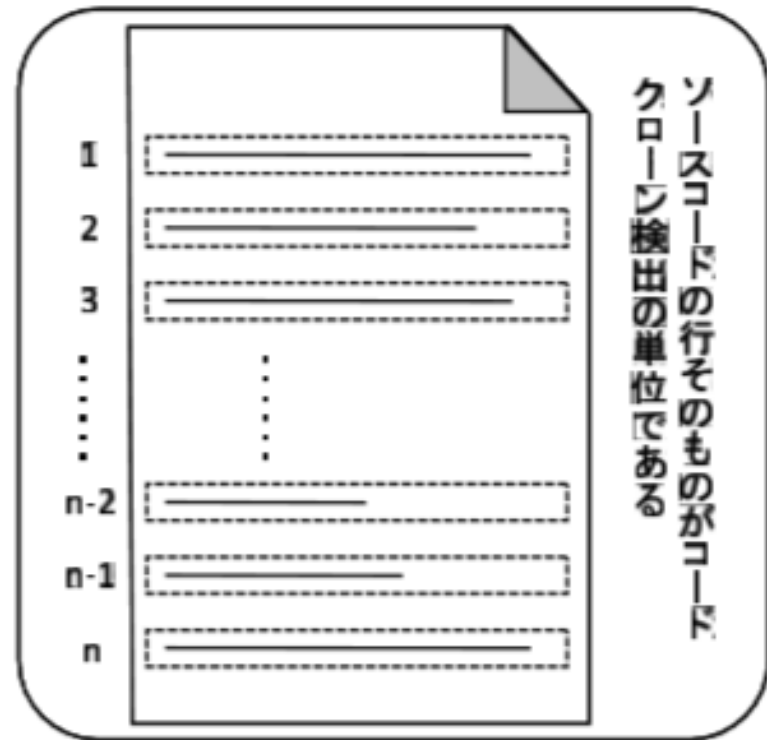
- タイプ1
 - コーディングスタイルを除いて、完全に一致するコードクロン
- タイプ2
 - ユーザ定義名、また一部の予約語のみが異なるコードクロン
- タイプ3
 - タイプ2における変更に加えて、文の挿入や削除、変更が行われたコードクロン

検出技術の分類

- 既存の検出技術は、コードクローンをどの単位で検出するかによって、大まかに5つに分類できる
 1. 行単位の検出
 2. 字句単位の検出
 3. 抽象構文木を用いた検出
 4. プログラム依存グラフを用いた検出
 5. メトリックスやフィンガープリントなど、その他の技術を用いた検出

行単位の検出

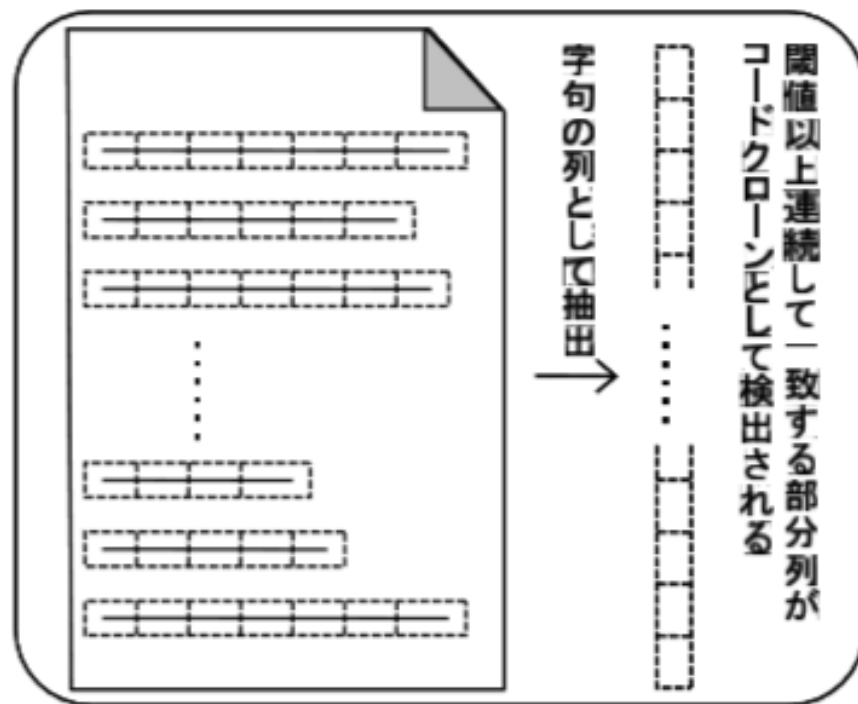
- 他の検出技術に比べ
検出速度が高速
- コーディングスタイル
が違う場合、検出でき
ない



(a) 行単位の検出

字句単位の検出

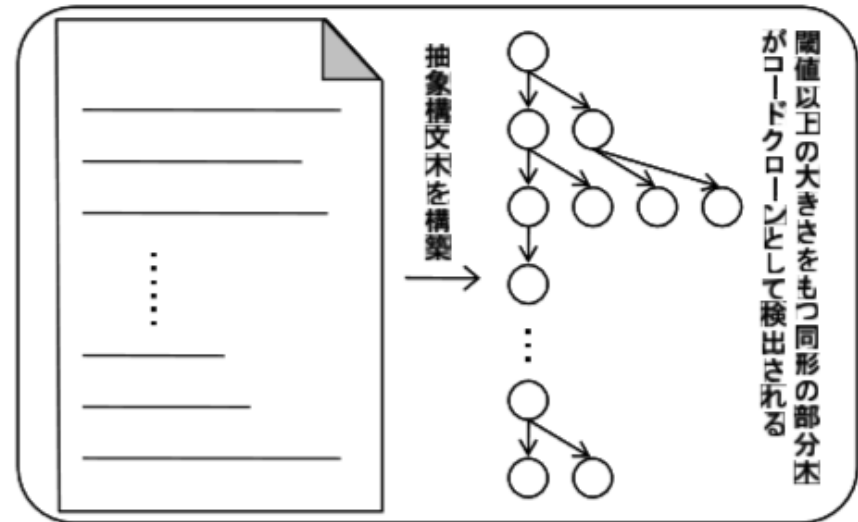
- 検出結果がコーディングスタイルに依存しない
- ソースコードを検出用の中間表現に変換する必要もないため、高速に検出できる



(b) 字句単位の検出

抽象構文木を用いた検出

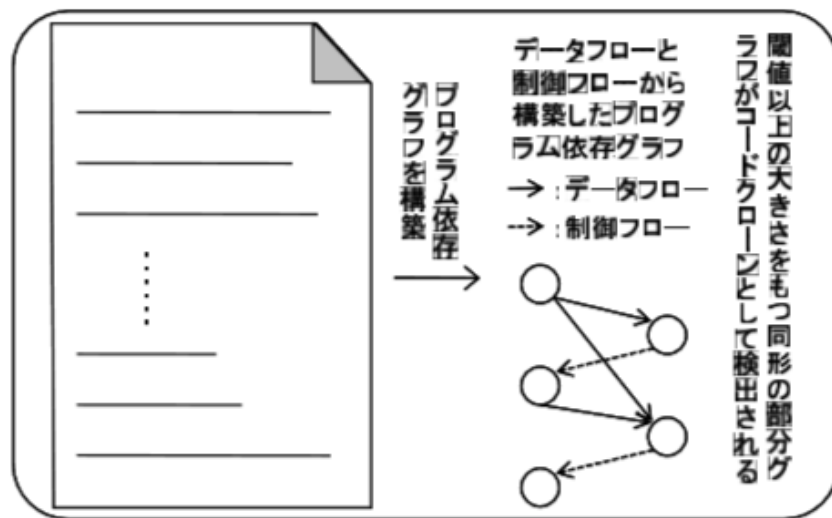
- 検出結果がコーディングスタイルに依存しない
- プログラムの構造を無視した類似部分は検出されない



(c) 抽象構文木を用いた検出

プログラム依存グラフを用いた検出

- 他の技術では検出することができないコードクローンを検出できる



(d) プログラム依存グラフを用いた検出

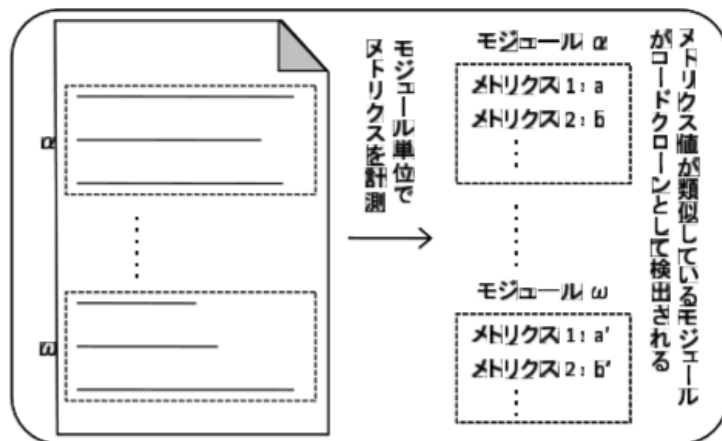
```
...
bb  tmpa = UCHAR(*a),
##  tmpb = UCHAR(*b);
bb  while (blanks[tmpa])
bb      tmpa = UCHAR(++a);
##  while (blanks[tmpb])
##      tmpb = UCHAR(++b);
bb  if (tmpa == '-') {
bb      tmpa = UCHAR(++a);
bb      ...
bb  }
##  else if (tmpb == '-') {
##      if (...UCHAR(++b)...) ...
```

図 4 巻き付きコードクローン

Fig.4 Intertwined code clone.

メトリックスやフィンガープリントなどその他の技術を用いた検出

- メトリックスを用いた検出
 - タイプ3のコードクローンも検出できる
 - 問題点
 1. サイズの小さいモジュール → メトリックス値に差ができず誤検出の可能性が高くなる
 2. サイズの大きいモジュール → その内部の一部が類似していてもコードクローンとして判定されない



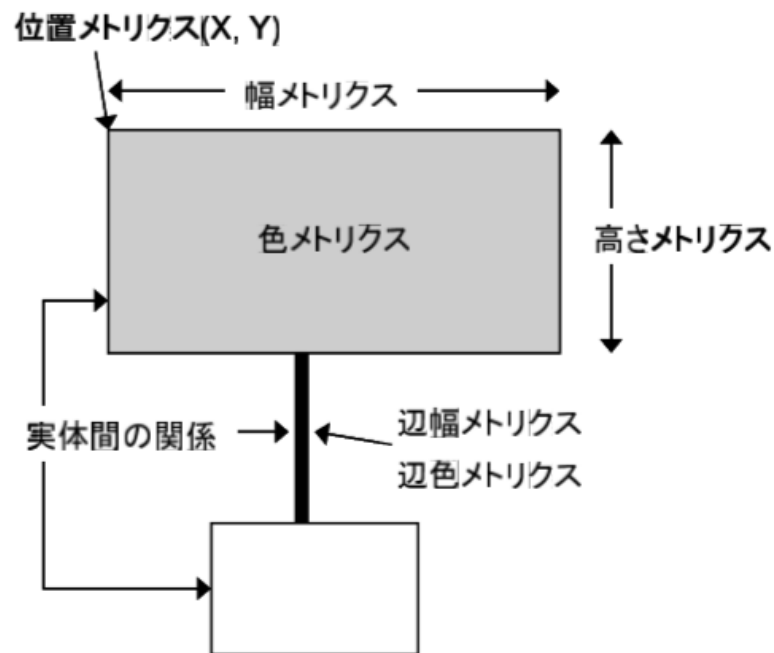
(d) メトリックスを用いた検出

メトリックスやフィンガープリントなどその他の技術を用いた検出

- フィンガープリントなどその他の技術を用いた検出
 - ソフトウェアが改変されても、フィンガープリントなどの値自体は変わらない
 - プログラムの剽窃を検出
 - プログラムの作者を特定

コードクローンの可視化

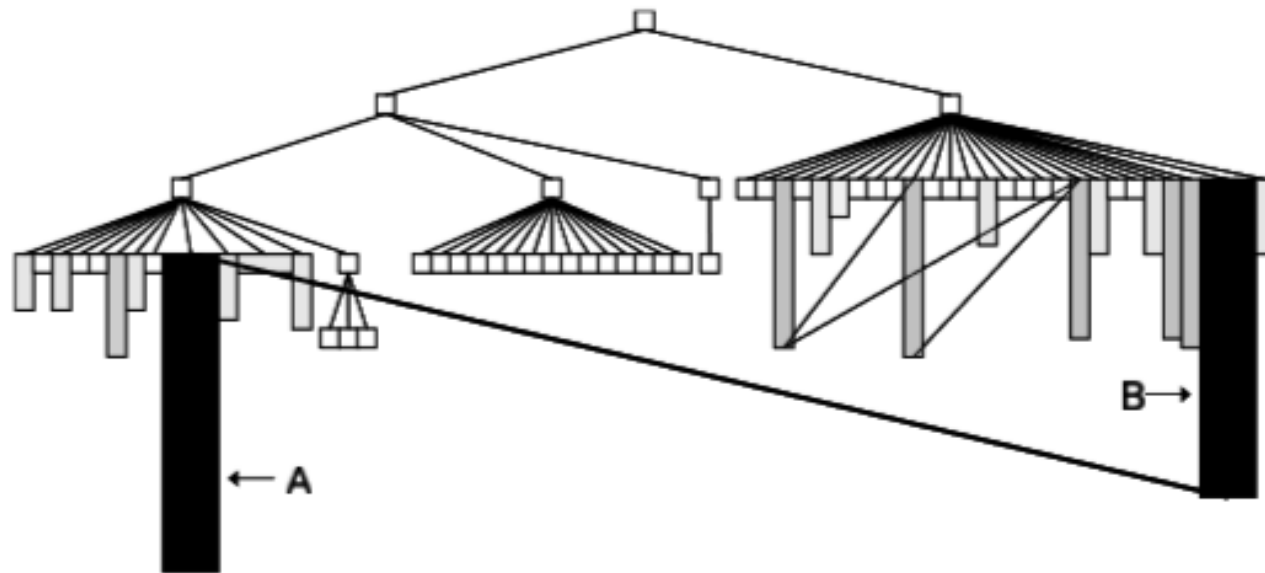
- Riegerらが提案
 - Polymetric Viewを用いたコードクローン情報の可視化手法
 - 様々な粒度で抽象化されたコードクローン情報を提供



(a) Polymetric View のモデル

[56] M. Rieger, S. Ducasse, and M. Lanza, “Insights into system-wide code duplication,” Proc. 11th Working Conference on Reverse Engineering, pp.100–109, Nov. 2004.

Polymetric Viewを用いた可視化の例



(b) Polymetric View を用いた可視化の例

- 要素 : ソースファイル
- 幅メトリックス : 要素内で閉じたコードクローン
- 高さメトリックス : 要素が他の要素と共有しているコードクローンの量
- 辺 : 両端のソースファイル間で共有されているコードクローンの量

所感

- 可視化について、Polymetric Viewが面白いと感じた
 - 色や幅、高さ、辺を使って分かりやすい