Fast and Precise Code Clone Reduction (2006/9/5)

米澤研 M2 佐藤秀明

コードクローンとは

- 文面が似ているソースコード断片の集合
  - ソースのコピー＆ペーストで発生
- 大規模なシステム開発では有害
  - メンテナンス性の低下

ソースコードを解析して
コードクローンを発見できないか？

コードクローンの定義付け

- 「互いに等しいトークン列の組」と定義
  - ただし変数のリネームについては等しいとみなす

```
int f(int x){
    int a = 3;
    return a * x;
}
```

クローン

```
int g(int y){
    int b = 3;
    return b * y;
}
```

コードクローン発見の既存手法

- 接尾辞配列を利用 [Basit et al. 2005]
  - テキスト検索の手法を応用してクローンを発見
- 接尾辞配列：文字列の全接尾辞を辞書順にソート
  - 省スペース
  - 単純な構造

actaat$

index	pos	実体
0	3	aat$
1	0	actaat$
2	4	at$
3	1	ctaat$
4	2	taat$
5	5	t$
6	6	$

既存手法の問題

- 解析に時間がかかる
  - 接尾辞配列の構築
  - クローンの発見

より高速な解析の必要性

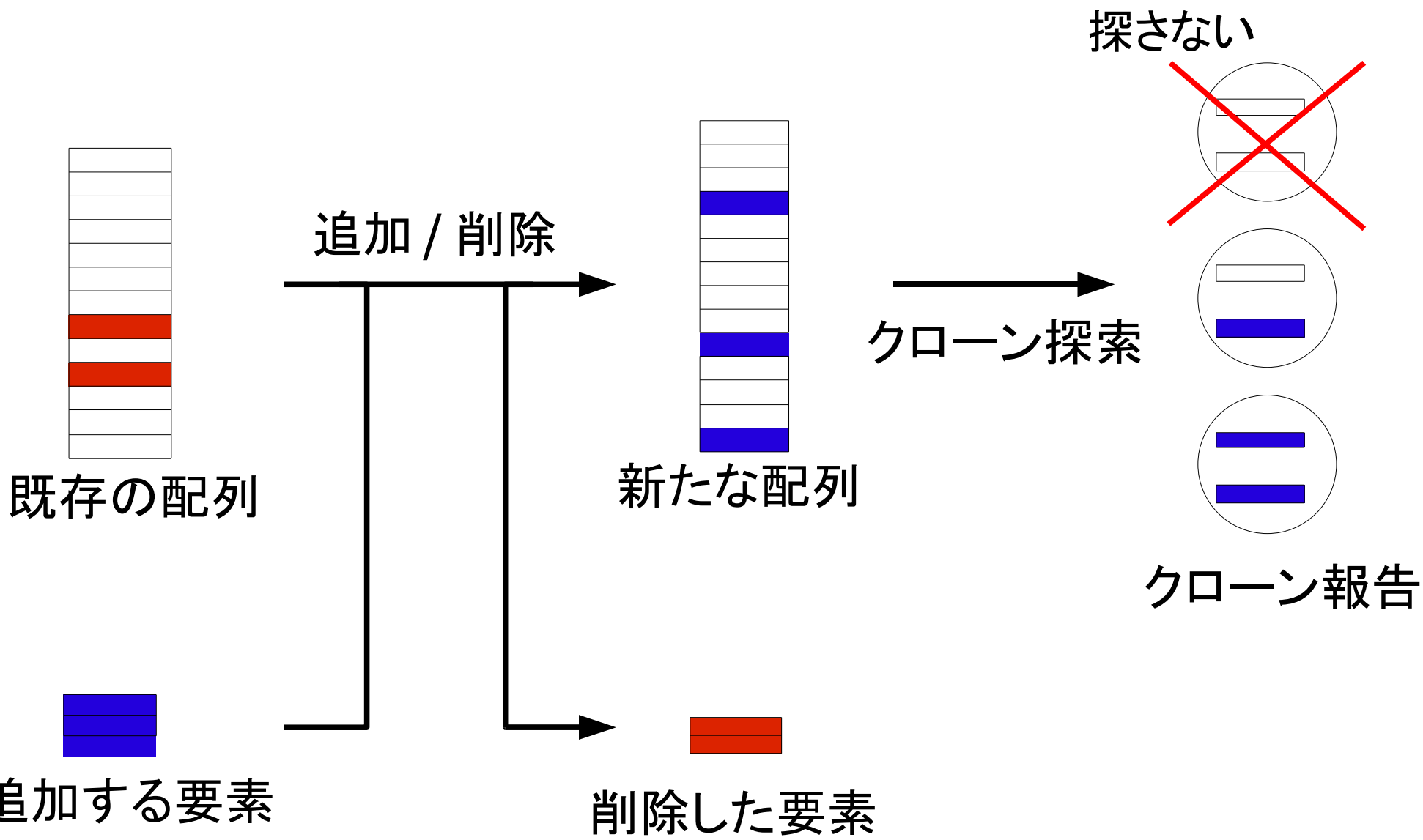
我々のアプローチ

- ソースコードの変化に合わせて逐次的に解析
  - 差分のみ解析することで毎回の計算量を削減

解析の方針

- 前回作った接尾辞配列に今回生まれた差分を反映
  - 毎回全てを作り直すより速い
- 今回新たに生まれたクローンのみ発見
  - 調べるべきクローンの組み合わせを減らす

解析手順



実装

- ソースコード読み込み
  - 変数名の違いを吸収 [Baker 1992]
- 接尾辞配列構成
  - [Larsson et al. 1999] を利用
- 接尾辞配列の更新
  - 変更箇所の情報はソースファイル単位で丸ごと差し替え
- クローン発見
  - [Abouelhoda et al. 2002] を逐次的な手法に改変

言語の選択

- 実装言語：C
  - 実行速度を重視
- 対象言語：Java
  - 広く使用されている

評価目的

- 逐次的手法による解析時間の削減を確認する
  - 巨大なソースコードに対する有効性

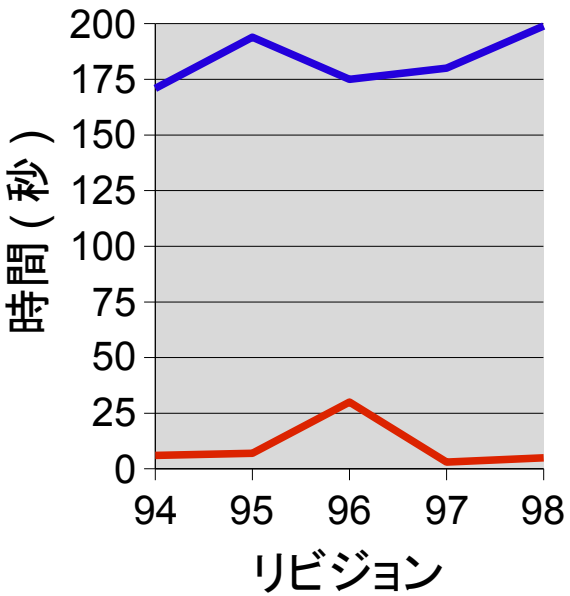
評価条件

- 対象ソースコード：Java SE 6 の weekly snapshots
  - ファイル数： 約 15,000
  - 総トークン数： 約 13,000,000
- 解析対象とするトークン列の長さ：35 以上
  - 小さすぎて無意味なクローンを排除 [Basit et al. 2005]

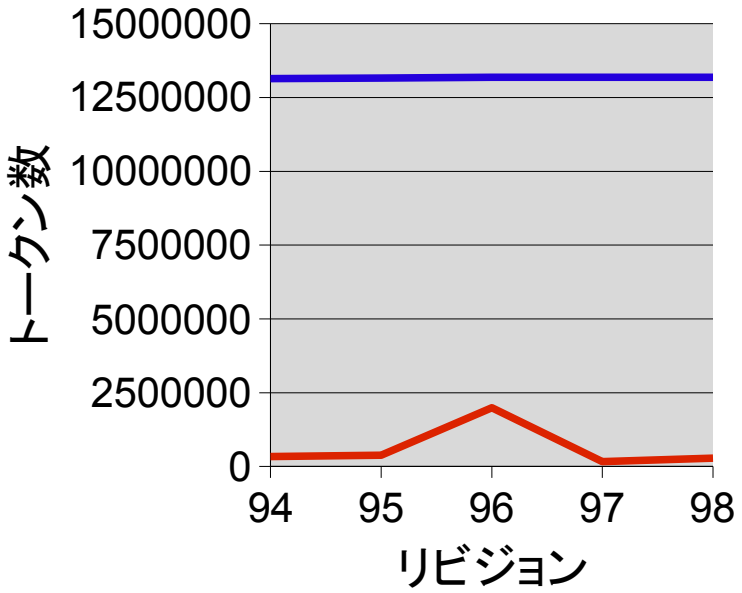
評価結果

- 全解析と同じ結果をより短い時間で得られた
  - おおむね 95% 程度の時間短縮
  - 更新分量が多くても解析時間は許容範囲
    - リビジョン 96

解析時間



新たに読み込んだトークン数



CPU: Pentium 4 2.6GHz
メモリ：2GB
OS: Linux 2.4.31
コンパイラ：gcc 3.3.2

これからの予定

- クローンの自動除去機能でユーザを支援
  - 見つけたクローンに除去手法の雛形を適用
    - メソッド抽出
    - Generics
    - 各種リファクタリング手法
  - ユーザの承認を受けて自動的にソースコードを変換

最終目標：高速かつ正確なクローンの除去

まとめ

- 逐次的手法によるコードクローンの高速な発見
  - 接尾辞配列の逐次的な構築
  - 報告対象のクローン候補数を絞る
- 解析時間の短縮を確認
- クローンの除去作業の簡便化につなげたい