プログラミング 15－2
（1997．9．12）

n-gram解析を用いたプログラム中の非定型パターン・欠損の検出

吉川 裕之* 貴島 寿郎† 梅村 恭司‡

豊橋技術科学大学 情報工学系 梅村研究室
〒441 愛知県豊橋市天伯町雲雀ヶ丘 1-1
Tel. 0532-47-0111(内線 5430)
* hiroyuki@avenue.tutics.tut.ac.jp
† kijima@tutics.tut.ac.jp
‡ umemura@tutics.tut.ac.jp

概要

プログラムでは同じようなパターンの繰り返しが多い．したがって，プログラム中に一定回数以上現れる，一定長さ以上の文字列を取り出すと，プログラムのほとんどの部分を取り出すことが出来る．取り出せた部分は複数回現れていることから，プログラムとして意味がある部分であるといえる．一方，取り出せない部分はプログラム中の特異な部分であると考えられる．我々はプログラムの誤り等が取り出せない部分に含まれるのではないかと考え，この部分にマークをつけるツールを作成した．

本稿ではn-gram解析で取り出せない部分にマークをつけることでプログラム中の特異部分を検出する，言語に依存しないプログラミングングツールを提案し，その評価を行う．

キーワード n-gram，プログラミングツール，統計的言語処理

Detecting irregulars and faults in program using n-gram analysis

Hiroyuki Yoshikawa* Toshiro Kijima† Kyoji Umemura‡

Umemura Laboratory, Department of Information and Computer Sciences
Toyohashi University of Technology
1-1 Tempaku Toyohashi Aichi 441 Japan
Tel. 0532-47-0111(Ext. 5430)

Abstract

There are a lot of patterns in programs. Thus, if we extract strings whose frequency are more than a chance in a program, the strings will cover most part of the program. The parts would be meaningful as program because they appear plural times. In the other hand, the parts that are not extracted may have some characteristics. We think that special parts include irregular or faults, then we made a tool marking these parts.

In this article, we propose the language independent programming tool, marking parts that we can not get with n-gram analysis in the program. Finally we evaluate this tool.

key words n-gram, programming tool, statistical processing

1 はじめに

近年，アプリケーション等のプログラムは巨大化の傾向にある．このような巨大なプログラムでは同様のパターンが繰り返し出てくることが多い．しかしながら，繰り返しパターンに当てはまらない非定型パターン・欠損も存在する．このような非定型パターン・欠損が生じる原因としてプログラムの誤りやプログラミングスタイルの揺らぎなどが考えられる．したがって，大きなプログラム中の非定型パターン・欠損を検出することによってプログラムの誤り箇所等の検出が期待できる．これはプログラミング言語の個別の文法に依存しないことを特徴とする。

本稿では，プログラミングスタイルを含めたチェックツールの第1歩として，大きなプログラム中の非定型パターン・欠損をn-gram解析の手法を用いて非文法的に検出する手法を提案し，その評価を行う．

2 アプローチ

n-gramとはあるn個の文字の組合せである．このn-gramをある文書中にどの程度の頻度で現れるかを調べ，一定回数以上現れたn-gramを取り出すことで文書中の意味のある固まりををとり出す試みがなされている[1][2][3][4]．以下n-gram解析によってとり出された固まりを熟語と呼ぶことにする．

プログラムリストには定型パターンの繰り返しが数多く現れる．このため我々はプログラムリストに対してn-gram解析を行うと，ほとんどの部分が熟語としてとり出すことが出来ると予想した．すなわち，熟語の部分はプログラムとして当然の性質を持つところで，熟語と熟語の間の取り出せなかった部分は特異な性質を持つと考えられる．この熟語間の取り出せなかった部分を我々はグルー (glue) と呼ぶことにした．

図1に熟語とグルーの例を示す．

図1では，2回以上現れた3文字以上の文字列を熟語としている．図1の例では，abc, bcd, abcdeが熟語となり，そのどれにも属さない部分，前から4つ目のcと10, 11番目のf, gがグルーである．なお，bcdeも熟語としての条件を満たしているが，どのbcdeもabcdeの一部分としてしか現れていないので熟語とはみなさない．この例では2回以上現れた3文字以上の文字列を熟語としたが，適切な熟語の条件はテキストの特性によって変化する．

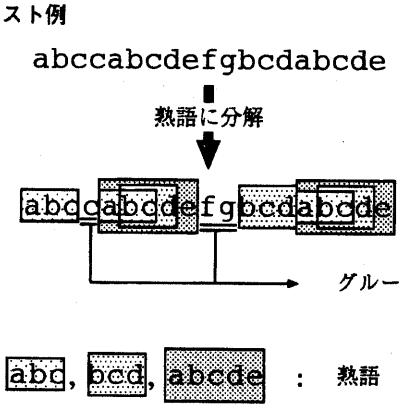

図 1: 熟語とグルーの例

また，熟語の切れ目にも着目した．隣接する2文字について，2文字とも何らかの熟語に含まれているが，この2文字を同時に含む熟語がない場合，この2文字の間を熟語の切れ目とする．我々はこの熟語の切れ目を長さ0のグルーと呼ぶことにした．図1ではbcdとabcdeの間に長さ0のグルーがある．

我々はプログラムリストについてn-gram解析を行い，グルーを検出するプログラムunique_partを作成し，グルーがどのような性質を持っているかを調べた．

3 グルー検出アルゴリズム

グルー検出アルゴリズムを以下に示す．

1. 入力データのsuffix-sortを行う，すなわち，先頭+i文字目で始まり最後(n文字)までの部分文字列を辞書順に並べる．
2. 配列lenを用意する．len[i]は辞書順でi番目とi+1番目の部分文字列が先頭から一致する文字数を表す．
3. カウンタiに0を代入する．
   (a) 辞書順でi番目の部分文字列とi+1番目の文字列が先頭から何文字一致しているかを調べ，その値をlen[i]に代入する．
   (b) iに1を加える．
   (c) i < n-1 ならば(3a)へ．

(d) len[n-1] に 0 を代入する.

4. データ中に複数回現れる文字列で始まる部分文字列の辞書順での場所を表す配列 start と，現在の一致文字数を表す変数 level を用意する.

5. i に 0 を代入する. level に 0 を代入する.

   (a) len[i] > level ならば
      i. start[level] に i を代入する.
      ii. level に 1 を加える.
      iii. (5a) へ.

   (b) len[i] < level ならば
      i. 長さ level の，ある文字列が start[level]−i 個あることがわかる. この文字列の長さと個数が別に定める基準以上ならば，その文字列を熟語とみなす. 熟語の開始位置は辞書順で start[a], start[a]+1, start[a]+2, ···, i-1 である. 開始位置と長さからデータ中の熟語部分を求め，色付けする.
      ii. level から 1 を引く.
      iii. (5b) へ.

   (c) i に 1 を加える.

   (d) i < n ならば (5a) へ.

6. 入力データを出力する. その際，対応部分に色がついていない部分にマークを行い，グルーを示す.

7. 終了

上記のアルゴリズムで一応の結果を得ることができる. しかしながら人間が書いた文書の場合，出現頻度の高い文字列の後ろに偶然ある文字が付くと，出現頻度が高いためこのような偶然が複数回現れることがある. 上で述べたアルゴリズムではこのような文字列を熟語として誤認してしまう. そこで，ある熟語の候補が他の候補と共通の文字列で始まっていないかを調べる. 共通の文字列長が候補の文字列長とほとんど変わらない場合，候補ではなく共通の文字列を熟語として処理する.

このアルゴリズムのデータの長さ n に関する平均的な計算量について考える. suffix-sort は quick sort が使えるので $O(n \log n)$，一致数の計算のところは，一致数の最大値を m とすると $O(mn)$，熟語の候補を調べるのには $O(n)$，候補の不要部分を調べるのには $O(n)$，熟語にマークするのには $O(mn)$ の計算量がかかる. したがって m を定数とすると，このアルゴリズム全体の計算量は $O(n \log n)$ となる[1]となる.

4 グルーの分析

本節では unique_part の性質と実例について述べる. はじめにどのような部分がグルーとして検出されるかを示す. 次にグルーの割合を示し，最後に実行時間とプロセスの大きさを示す.

実験では 2 回以上現れる 5 文字以上の文字列を熟語の候補とみなし，2 つの候補の共通の文字列の長さが (候補の長さ −2) 以上の場合は共通文字列を熟語とした. そうでない場合は候補を熟語とした. 実験は GNU emacs, kcl および我々の研究室で開発したプログラムの autoref について行った. GNU emacs は C 言語で書かれたファイルを結合したものを，kcl は C 言語と Lisp で書かれたファイルを結合したものをデータとして使用した. autoref はすべて C 言語で書かれており，ソースファイルすべてを結合したものをデータとして使用した.

4.1 プログラムの診断に使えるケース

この節ではプログラムのチェックにおいて有益であると思われる性質を示す.

```
void command_assign(int port)
{     char domain[1024];
            ------
```

図 2: 宣言だけで使用しない変数

図 2では domain という変数名がグルーとなっている. これは domain という文字列が他にはない，すなわち宣言はされているが実際には使われていないためである.

図 3はコメントに対する実行例である. コメントは自然言語で書かれるので，実行部分ほど熟語を取り出すことは出来ないと予想した. しかしながら実際はコメントも図 3のように定型パターンを繰り返すことも多い. このような場合，実行部分と同様ほとんどが熟語とみなされるので，コメント内のグルーが何らかの意味を持つことがある.

[1]熟語を構成する文字を袋から任意に取り出したデータに対しては $m = \log n$ となる. この場合の計算量も $O(n \log n)$

```
/* fdが読みとり可能となったときに呼び出す
      ^ ^ ^ ^
   ハンドラを設定する。
   ハンドラを消去する時には((Handler) 0)を指定する。
*/
int selection_read_assign(int fd, Handler fn);


/* fdが書き込み可能となったときに呼び出す
      ^ ^ ^ ^
   ハンドラを設定する。
   ハンドラを消去する時には((Handler) 0)を指定する。
*/
int selection_write_assign(int fd, Handler fn);


/* fdが異常事態になったときに呼び出す
      ^ ^ ^ ^ ^
   ハンドラを設定する。
   ハンドラを消去する時には((Handler) 0)を指定する。
*/
int selection_except_assign(int fd, Handler fn);
```

図 3: コメント部分の例

```
#imclude "config.h"
  ^
#include <stdio.h>
#undef NULL
#include "lisp.h"
#include "commands.h"
#include "buffer.h"
#include "window.h"



#incalude "config.h"
    ^
#include <stdio.h>
#undef NULL
#include "lisp.h"
#include "commands.h"
#include "buffer.h"
#include "window.h"
```

図 4: タイプミスの検出 (2)

図 4.1, 4は熟語とみなされた部分に意図的に1文字変更したり追加したものに対する実験結果である. 変更したところや追加した文字がグルーとなっているのでタイプミスの検出の手がかりにもなることがわかる.

```
/* socket_listenerは、acceptを実行して、
                  ^ ^      ^ ^ ^ ^ ^ ^
  得られたファイルデスクリプタに対して、
---^ ^ ^ ^         /         ^ ^ ^ ^ ^
  入力があったら、commandを実行
---^ ^ ^ ^ ^ ^ ^ ^       ^ ^ ^
  するように設定する関数: command_assign を呼び出す。
---^ ^ ^ ^          ^ ^ ^              ^        ^

*/
```

図 5: 長い熟語中の誤り

図5は長い熟語中の誤りを示す. “ファイルディスクプリタ”のように長い熟語の場合, 中央付近の文字が抜けたとしても前後が別々の熟語となってグルーとならない場合がある. このような場合でも, 熟語の切れ目を示すことで, 誤りを検出できることがある.

4.2 困った性質

```
/* ハンドラの設定などについて、初期化する。*/
--          ^ ^ ^ ^          ^ ^ ^
extern void command_initialize();

/* WWWブラウザからの要求にしたがって、サービスを提供
   ---          ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^
   ハンドラをportのファイルディスクリプタに設定する。
            ^^^^

*/
```

図 6: ファイルの先頭のグルー

図6では2つのコメント開始記号の内, グルーとなっているものとなっていないものがある. グルーとなっているコメント開始記号はファイルの先頭, すなわち他の場所のように前に改行コードがないために熟語とはならなかったのである. これは, あらかじめプログラムの先頭にいくつか改行コードを挿入しておくことで解決できる.

図7はローカル変数がグルーとなっている. これは今回作成したプログラムでは5文字以上の文字列

```
int http_conversion(char * out, int out_size,
                    char * in, int in_size,int seq)
{ int src;
```

図 7: 3 文字では熟語と見なさない

(5-gram 以上) を熟語の候補としたので，src のように短い変数名は熟語と認識されなかったためである．何文字以上を意味のある文字列とするかは非常に難しい問題である．

また，実際のプログラミングにおいて良く似た処理を書く場合，エディタの機能を用いてコピーしてから変更を加えることが多い．コピー元に誤りが含まれていると，コピーによってその誤りが複数回現れることになるので，グルーの検出条件によってはその誤りを検出できない可能性がある．

このように，unique_part でプログラム中のすべての誤りが検出できるわけではない．

4.3 グルーの割合

プログラム中のグルーの割合を表 1に示す．

表 1: グルーの割合

プログラム	グルー (byte)	全体の大きさ (byte)	割合
autoref (1)	2183	30575	7.14%
autoref (2)	617	25608	2.41%
emacs	10024	2066571	0.49%
kcl	8800	2904887	0.30%

ここで，autoref (1)，emacs，kcl はそのまま，autoref (2) はコメントを除去してから，それぞれグルーの検出を行った．

表 1の autoref (1)，(2) より，グルーの内コメントの占める割合が多いことがわかる．これはコメントが自然言語で書かれるためにしかたがない．

また，emacs や kcl のような大きなプログラムだとほとんどの部分が熟語とみなされ，グルーは全体の 1% に満たない．emacs や kcl ほどの大きなプログラムでは図 8のようにコメントもほとんどが熟語とみなされている．

この程度の割合ならば人間がチェックすることができるので，検出の効果が期待できる．

```
/* Primitives for word-abbrev mode.
   Copyright (C) 1985, 1986
                 Free Software Foundation, Inc.

This file is part of GNU Emacs.
```

図 8: コメントも熟語とみなされる例

4.4 実行時間

表 2に各プログラムに対する実行時間を示す．

表 2: 実行時間

プログラム	大きさ [byte]	時間 [sec]		
		real	user	sys
autoref	33137	2.4	2.4	0.0
emacs	2066571	935.3	928.1	1.8
kcl	2904887	1141.2	1083.8	3.2

表 2の値は SPARC station 5 (110MHz, Memory 192Mbyte) で実行した時のものである．

表 2より，大きなプログラムに対しては対話的に使用することはできないが，ツールとして使用不可能なほど実行時間がかかるわけではないことがわかる．

4.5 実行プロセスの大きさ

実行中のプロセスの大きさは，データの 11 〜 14 倍程度である．小さくはないが，データが大きくなってもプロセスが爆発的に大きくなることはない．

5 コンパイラとの比較

プログラミンツールとしての有効性を評価するため次の実験を行い，コンパイラでは検出されない誤りが検出されるか調べた．

1. あるプログラム中の任意の 1 文字を別の文字で置き換える．
2. そのプログラムをコンパイルし，エラーが検出されるか調べる．
3. エラーが検出されなければ，置き換えた文字がグルーとして検出されるか調べる．

実験は4節でも用いたautorefとベンチマークテストプログラムのシリアル版NASのappbtに対して行った．appbtの記述言語はFORTRANで，大きさは150342[byte]である．グルー検出条件は4節の実験と同様である．置き換える文字は“a”を用いた．以下にコンパイラでは検出できずunique_partで検出された誤りの例をいくつか示す．

図9〜12にautorefにおける実験結果を示す．

```
fprintf(stderr, "%s in %d of %a\n"
                                -
                , line, file, message);
```

図9: フォーマット指定の誤り

図9は出力関数のフォーマット指定の誤りである．プログラムの動作に影響を与える誤りではあるが，実際に実行することで場所を特定することができるので，それほど深刻な誤りではない．

```
#define CHECK(x,y) error_aype((x) != (y)
                        ^^^^
                                , __LINE__, __FILE__)
                                 /
```

図10: 使用されないマクロの定義の誤り

```
int seaection_except_assign(int fd, Handler fn);
      -
```

図11: プロトタイプ宣言の誤り

図10，図11，12は，それぞれ使用されないマクロの定義の誤り，プロトタイプ宣言の誤り，コメント内の誤りである．これらの誤りはプログラムの動作に直接関係はないが，指摘することができればプログラムを清書する際に役立つ．

図13，14にappbtにおける実験結果を示す．

図13は配列の添字の誤りである．本当はk-1となっているべきところが，-がaに置き代わっている．コンパイラはka1という変数とみなしてしまうので，誤りを検出できなかった．このような誤りは実際にあり得そうであり，発見が困難である．

図14は変数名の誤りである．この誤りも図13と同様に，タイプミス等により起こり得る，発見の困難な誤りである．

```
/* fdが読みとり可能となったときa呼び出す
      - - - -                  -
   ハンドラを設定する。
   ハンドラを消去する時には((Handler) 0)を指定する。
*/
```

図12: コメント内の誤り

```
$                       / b(3,3,i,j,ka1)
                                  -
```

図13: 配列の添字の誤り

5.1 評価まとめ

4節の実験により検出されたグルーには様々な誤りが含まれることがわかった．また，実行時間やプロセスの大きさはツールとして使用できる範囲内にある．

5節の実験では，コンパイラでは検出できない誤りを検出できることがわかった．特に図13，14の誤りは，いずれも変数を宣言無しで使えるFORTRANの特徴によって生じた，発見し難い誤りである．このことから，特にFORTRANやLispのような非宣言型の言語においてプログラムのチェックに役立つと考えられる．

しかしながら，コンパイラで簡単に検出される誤りでも複数回現れると，unique_partでは検出できない可能性がある．したがって，unique_partはコンパイラを取り替えるものではない．

以上のことから，コンパイル後に改めてチェックするツールとして十分役に立つといえる．

6 関連研究

C言語にはlint等のさまざまなプログラムのチェックツールがある[5][6][7]．これらはCのプログラムを検査して，誤りや移植性に欠ける箇所，無駄な記述等を検出する強力なチェックツールである．これらのツールはC言語の文法に基づいてチェック

```
flux(4,j) = ay3 * ( u41j - u41jm1 )
              -
```

図14: 変数名の誤り

しているので，Cのプログラムしかチェックできない．これに対して，本稿で述べたアルゴリズムはプログラミング言語の文法に依存しないので，さまざまな言語で書かれたプログラムをチェックすることが出来る．

大量のデータから意味のある固まりを取り出す研究もある[8]．本研究はその中の一つと位置付けられる．

7 まとめ

プログラムリストに対してn-gram解析を行うツールunique_partを作成した．このツールで，プログラムリスト中の特異な部分を検出できることを示した．また，実行時間やプロセスの大きさがプログラミングツールとして使用できる程度であることも示した．

コンパイラでは検出できない誤りを検出する評価実験では，いくつかの誤りが検出できることを示した．特にFORTRANプログラムに対しては，発見困難な誤りを検出することができた．

しかしながら，コンパイラで簡単に検出できる誤りでも，複数回現れるとunique_partでは検出できない可能性がある．このため，unique_partはコンパイラの誤り検出の機能に置き代わるものではない．

これらのことから，unique_partをコンパイラのエラー検出機能を補うツールとして提案する．コンパイルはできるが動作がおかしいプログラムをチェックするツールとしては，十分役に立つと考えられる．

今後の課題としては，実際に使用してもらうことで，さらに評価を深めることがあげられる．また，熟語検出条件の文字列の長さと出現頻度を変更すると異った結果が得られる．これらの条件の最適な値を求めることも今後の課題である．

参考文献

[1] 長尾 眞，森 信介，「大規模日本語テキストのnグラム統計の作り方と語句の自動抽出」，情報処理学会研究会報告自然言語処理 96–1, pp.1-8, 1993.

[2] 長尾 眞，森 信介，「nグラム統計によるコーパスからの未知語抽出」，情報処理学会研究会報告自然言語処理 208–2, pp.7-12, 1995.

[3] 下畑 さより，杉尾 俊之，永田 淳次，「隣接文字の分散値を用いた定型表現の自動抽出」，情報処理学会研究会報告自然言語処理 110–11, pp. 71-78, 1995.

[4] 中渡瀬 秀一，木本 晴夫，「統計的手法によるテキストからの重要語抽出メカニズム」，情報処理学会研究会報告情報学基礎 39–6, pp.41-48, 1995.

[5] S.C. Johnson, "Lint, a C Program Checker", Unix Programmer's Supplementary Documents, Vol1.1, 9, Univ. of California, 1986.

[6] 青木 圭子，瀧塚 孝志，橋本 和男，小花 貞夫，「C言語プログラム検査ツールの実装と適用結果」，情報処理学会研究会報告ソフトウェア工学 100-9, pp.63–70, 1994.

[7] 掛下 哲郎，小田 まり子，「正規表現によるCプログラムの落し穴検出ツール」，情報処理学会研究会研究報告ソフトウェア工学 86-7, pp.43–49, 1992.

[8] Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smith and Ramasamy Uthurusamy, "*Advances in Knowledge Discovery and Data Mining*", American Association for Artifical Intelligence, 1996.