特別研究報告

題目

Dockerfile に対する構文種別に着目した コードクローン検出手法の提案

指導教員

楠本 真二 教授

報告者

鶴 智秋

2021 年 2 月 9 日

大阪大学 基礎工学部 情報科学科

2020 年度 特別研究報告

Dockerfile に対する構文種別に着目した コードクローン検出手法の提案

鶴 智秋

内容梗概

本論文では，Docker に対する Type-2 コードクローンの検出手法を提案する．Docker とは，コンテナ型仮想環境を実現するプラットフォームであり，サーバインフラを支える技術として注目されている．Docker では仮想環境実現の手順を，Dockerfile と呼ばれる一種のソースコードの形式で記述する．そのため，似た構造の繰り返しや重複といったコードクローンが必ず含まれる．本研究は，Dockerfile における Type-2 クローンの検出を目的として， Dockerfile 固有のネスト構造という性質に着目した検出手法を提案する．提案手法では，構文要素に対して適切な正規化を行い， Dockerfile 構文と Shell Script 構文に分離し，それぞれの構文ごとに接尾辞配列アルゴリズムを用いてコードクローン検出を行う．GitHub 上に公開されている約 2,000 個の Dockerfile を対象に適用実験を行い，高い適合率で Type-2 クローンを検出した．また，Dockerfile における定型処理を発見した．

主な用語

Docker，Dockerfile，コードクローン，コードクローン検出

目次

1 はじめに 1

2 準備 3

2.1 コードクローン 3

2.2 Docker 3

2.3 Dockerfile における重複コード検出手法 3

2.4 接尾辞配列アルゴリズム 4

3 提案手法 6

3.1 概要 6

3.2 正規化対象トークンの検討 6

3.3 構文種別の分離 9

3.4 接尾辞配列の構成及び比較 9

4 実験 11

4.1 実験目的 11

4.2 実験対象 11

4.3 実験結果 11

4.3.1 検出できるクローンの個数及び精度 11

4.3.2 検出した Type-2 クローンの一例 14

5 おわりに 15

謝辞 16

参考文献 17

図目次

1 Shell Script 命令単位で分割された RUN 命令の一例 5
2 接尾辞配列の構成手順及びクローン検出方法 5
3 提案手法の概要 7
4 実際の Dockerfile に対するトークン正規化の例 8
5 Shell Script 命令列に Dockerfile 命令列が挿入された Type-1 クローンの一例 10
6 抽象化されたトークン名が異なる Type-1 クローンの一例 12
7 提案手法で発見した定型処理（Type-2） 13

表目次

1 主な Dockerfile 構文の一覧 4
2 正規化対象とするトークン 7
3 正規化対象としないトークン 7
4 検出できたクローン数及び適合率 13

1　はじめに

コンテナ仮想化と呼ばれるリソース効率に優れた仮想環境を実現するプラットフォームとして，Docker が着目されている [1]．Docker はインフラの構成自動化，IaC（Infrastructure as Code）を実現する技術の一つであり，コンテナ構築手順を Dockerfile と呼ばれるソースコードに記述する．インフラ構築手順のコード化により，誰がいつ実行しても全く同じインフラ環境を迅速に再現することが可能となる [2] [3]．Docker は近年注目されている技術であり，多くの企業や OSS で採用されて始めている [4] [5] [6]．一方で，Docker をはじめとする IaC の周辺技術は黎明期にあり，研究が未熟な領域が存在することも指摘されている [7].

本研究では，Dockerfile で発生するコードクローン（以後，クローン）について考える．クローンとはソースコード内における一致，または類似コード片のことであり，様々な研究者によって盛んに研究されている [8] [9] [10] [11]．クローンの検出により，利用 API の推薦 [12] や，冗長な記述に対するリファクタリングの推薦 [13]，同一変更を要する箇所に対する変更推薦 [14]，ライセンス違反の検出 [15]，といった様々な応用が可能となる．

これまでに Java や C などの手続き言語に限らず，様々な言語やファイルを対象としたクローン検出技術が研究されてきたが，Dockerfile に対するクローン検出の提案は少ない．一部，Oumaziz らによって Dockerfile を対象とした重複コード片の検出手法が提案されている [16]．この手法では，Dockerfile のトークンを分離し，転置索引と呼ばれるアルゴリズムを用いて重複箇所を検出する．しかし，この手法は完全一致する重複コード片（Type-1 クローン）のみを対象としており，変数名の違いを許容したクローン（Type-2）の検出は不可能である．

本研究では，Dockerfile に対する Type-2 コードクローンの検出手法を提案する．本研究の貢献は以下の通りである．

- Dockerfile における Type-2 クローンの検討と定義：Type-2 クローンでは，プログラムの振る舞いに影響を与えないトークンの違いを許容する．よってその検出には，変数名や関数名，定数の適切な正規化が必須である．Dockerfile では変数以外にも，ファイルパスやポート番号などの様々な定数要素が含まれる．さらには，Shell Script 構文内でのパラメタ順序やオプション名のエイリアスなどのように，振る舞いに影響を与えない記法が多数存在する．本研究では，正規化対象とするトークン要素を整理することで，Dockerfile における Type-2 クローンを検討する．
- 構文種別を考慮した Dockerfile におけるクローン検出手法の提案：Dockerfile は，単一ファイル内に複数言語の構文を記述可能なネスト言語である [1] [17]．コンテナ内部での処理を Shell Script 構文で記述し，それ以外のコンテナの外界との処理（ホスト OS からのファイルコピーやコンテナの公開ポート番号の指定など）を Dockerfile 構文で記述する．提案するクローン検出手

法では，まず与えられた Dockerfile から抽象構文木を作成し，トークンの正規化を行う．さらに 2 種類の構文を分離し，接尾辞配列アルゴリズムを適用することで構文種別を考慮したクローン検出を実現する．

- 公開されている Dockerfile に対する Type-2 クローン検出，及び Dockerfile における定型処理の発見：GitHub 上の 223 リポジトリ内に存在する 1,897 個の Dockerfile を対象として，提案手法を用いた Type-2 クローン検出を行う．提案手法を用いた結果，適合率 95 % で Type-2 クローンを検出できた．また，空のコンテナからディストリビューションを構成するといった処理のように，Dockerfile における定型処理を発見した．

2 準備

2.1 コードクローン

コードクローン（クローン）とは，ソースコード内における一致コード片または類似コード片を指す．一致コード片及び類似コード片の集合はクローンセットと呼ばれる．クローンは，一般的にその類似度の違いに基づき，以下の 3 種類に分類される [8].

- Type-1: 空白や改行，及びコメントの差異を除いて完全一致するクローン
- Type-2: 変数名や関数名などの識別子名の差異を除いて一致するクローン
- Type-3: 命令文単位での挿入や削除，変更が行われたクローン

これまでに，一般的なプログラミング言語において，多数のクローン検出手法が提案されている [8] [9] [10] [11]．また，ビルドツールや要求定義書などといった，プログラミング言語以外の言語においてもクローン検出研究が行われている [18] [19] [20] [21] [22] [23] [24].

2.2 Docker

Docker は，コンテナと呼ばれる OS レベルの仮想化を実現するプラットフォームである [25] [26]. Docker は近年注目されている技術であり，IT 企業の 87 % で使用されているほか，多様な OSS で採用されている [4] [5] [6]．Docker は IaC (Infrastructure as Code) と呼ばれるインフラの構成自動化を実現する重要な技術の一つであり，コンテナ構築手順をソースコードとして記述できる [2] [3]．Docker コンテナ構築手順を記したソースコードを Dockerfile と呼び，Dockerfile から構成されるコンテナのテンプレートをイメージと呼ぶ．Dockerfile は表 1 に示す主な Dockerfile 構文で記述される [27]．また，Dockerfile は RUN 命令に Shell Script 構文を内包できるという性質を持つ，ネスト言語でもある [1] [17].

2.3 Dockerfile における重複コード検出手法

Oumaziz らは，Dockerfile に対する重複コード検出手法を提案している [16]．彼らは，Dockerfile を構文解析したのち，Dockerfile 命令 6 種を 1 単位とした転置索引アルゴリズム [28] を適用して，重複コードを検出する．なお，Dockerfile において，単一の RUN 命令内部に複数の Shell Script 命令を記述するプラクティスが存在する [29]．彼らの手法では，Shell Script 命令に内在する重複コードも検出するために，構文解析時に RUN 命令を内部の Shell Script 命令単位で分離させる（図 1）.

Oumaziz らの手法は，トークンの正規化を適用していないため，文の類似ではなく完全一致，すなわち Type-1 クローンのみを検出対象としている．また，Dockerfile 構文と Shell Script 構文とを区別し

ていないため，例えば，Shell Script 構文（RUN 命令の中身）の内容が本質的にクローンの関係にあるものの，片方の Dockerfile では RUN 命令の途中に ENV 命令等の Dockerfile 構文が挿入されている場合，クローンとして検出することはできない．さらに，転置索引アルゴリズムでは最短クローン長を固定しているため，長さがそれ未満のクローンを検出できない [28].

2.4 接尾辞配列アルゴリズム

接尾辞配列は，文字列検索アルゴリズムで使用されるデータ構造の一種であり，検索文字列における接尾辞を辞書順に並べ替えて得られる配列である [30] [31]．接尾辞配列を用いることにより，長さが n の文字列から，長さが m である検索対象文字列の出現位置を $O(m + \log n)$ の時間計算量で求められる [30]．また，SA-IS 法を用いることで，接尾辞配列を $O(n)$ の時間計算量で構築できる [32] [33]．接尾辞配列は，クローン検出の分野においても採用されている [34] [35]．図 2 に，トークン列 “A B A C D A B” を基にした接尾辞配列の構成手順及びクローン検出方法を示す．まず，接尾辞列挙により入力トークン列の各接尾辞を配列に格納する（手順 1）．次に接尾辞配列を辞書順にソートする（手順 2）．最後に，接尾辞配列内の接尾辞同士に対し，前方一致する部分トークン列同士をトークンセットとして出力する（手順 3）．トークン列 “A B A C D A B” の例では，“A”，“A B”，“B” の 3 種類がクローンである．

表 1: 主な Dockerfile 構文の一覧

命令名	概要
FROM	コンテナの基となるイメージの指定
ARG	Dockerfile 内部における一時変数定義
ENV	コンテナ内部における環境変数定義
RUN	Shell Script で記述されるコンテナ構築手順
COPY	ホスト OS 上のファイルをコンテナへコピー
ADD	ホスト OS 上のファイルをコンテナへコピー tar アーカイブであるならばコンテナ上で展開
WORKDIR	ワーキングディレクトリ変更
USER	ユーザ名・グループ名の指定
EXPOSE	コンテナ外部に開放するポート番号の指定
ENTRYPOINT	コンテナ起動時に実行するコマンドの指定
CMD	コンテナ起動時に実行するコマンド引数の指定

```
RUN apt-get update && apt-get install -y postgresql-client && ¥
    rm -rf /var/lib/apt/lists/*
```

RUN命令に内包された
複数のShell Script命令を
“&&” または “;” 単位で
独立したRUN命令に分離

```
RUN apt-get update
RUN apt-get install -y postgresql-client
RUN rm -rf /var/lib/apt/lists/*
```

図 1: Shell Script 命令単位で分割された RUN 命令の一例

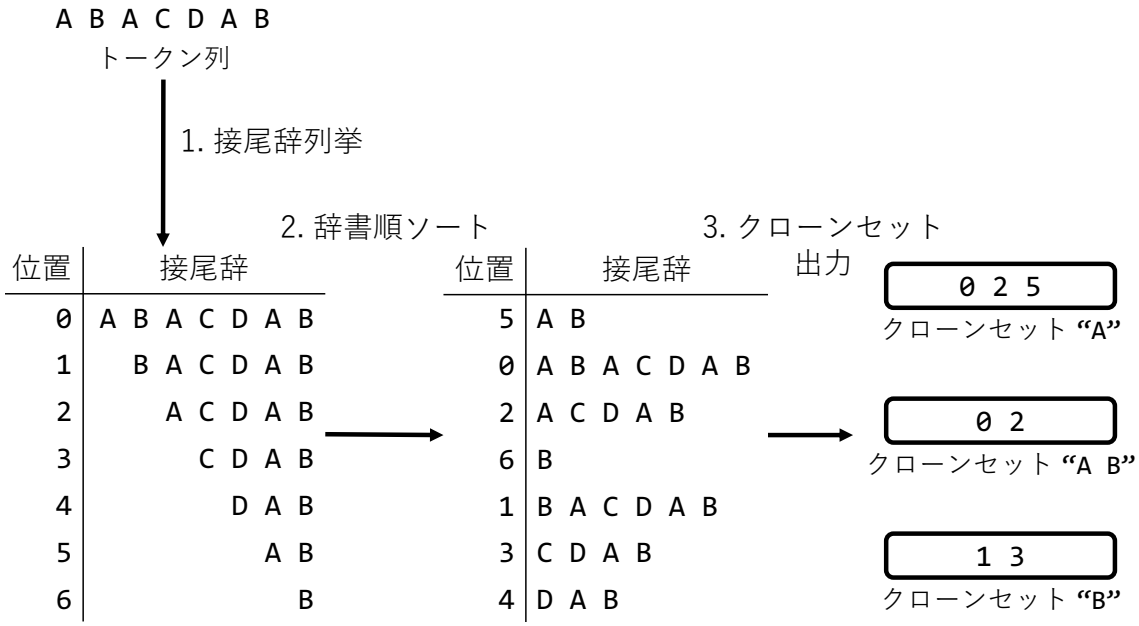

図 2: 接尾辞配列の構成手順及びクローン検出方法

3 提案手法

3.1 概要

提案手法の概要を図 3 に示す．提案手法の入力は複数の Dockerfile であり，出力は検出したクローンセットである．手法の流れについて図 3 に従って説明する．まず，入力として与えた複数の Dockerfile から，それぞれに対応する抽象構文木を生成する（手順 1）．次に，各トークンの正規化を行う（手順 2）．この正規化処理により Type-1 クローンの検出から Type-2 クローン検出への拡張が可能となる．さらに命令単位のハッシュ化（手順 3），及び構文毎の統合（手順 4）を適用する．Dockerfile 構文と Shell Script 構文をあらかじめ分離し，続く接尾辞配列アルゴリズム（手順 5）とクローンセットの検出（手順 6）を行うことで，構文毎のクローンの検出を実現する．

以降の節では，手順 2 で正規化するトークン（3.2 節），手順 4 で Dockerfile 構文と Shell Script 構文にあらかじめ分離する理由（3.3 節），そして手順 5 で接尾辞配列アルゴリズムを採用する理由（3.4 節）について述べる．

3.2 正規化対象トークンの検討

Type-2 クローンの検出のためには，ソースコード内のトークンを適切に正規化する必要がある．Java や C 等を対象とした一般的なコードクローン研究においては，変数名や関数名などの識別子名を正規化対象とすることが多い [36]．変数名や関数名は一種のラベル情報であり，そのラベルが異なっても同一処理を実現可能なためである．Dockerfile でも一時変数（ARG 命令で宣言される変数）などが存在しており，正規化の対象となる．さらに，ソースコードの処理内容の同一性という観点では，Shell Script でのオプション順序（`ls -l .` と `ls . -l`）や，オプション名のエイリアス（`ls -list` と `ls -l`）など，同一処理を実現可能な別記法が多数存在する．これらを適切に正規化することで Dockerfile の Type-2 クローンが検出可能となる．

本研究では，正規化対象とするトークンを表 2，正規化対象としないトークンを表 3 の通り定義した．図 4 は，提案手法によるトークン正規化の例である．正規化対象トークンの一つに，Shell コマンドにおけるパラメタ・オプション順序がある．Shell コマンドは，コマンドの種別ごとにパラメタやオプションの順序が定められている．コマンドによっては，パラメタやオプションの記述順序の違いを許容している．そのため，パラメタやオプションが異なるにもかかわらず同じ動作を行うコマンドが，複数の Dockerfile 内で発生する．例えば，`RUN apt-get install --yes --no-install-recommends` と `RUN apt-get install --no-install-recommends --yes` はパラメタ・オプション順序が異なる

[*1] https://github.com/docker-library/rabbitmq/blob/master/Dockerfile-ubuntu.template

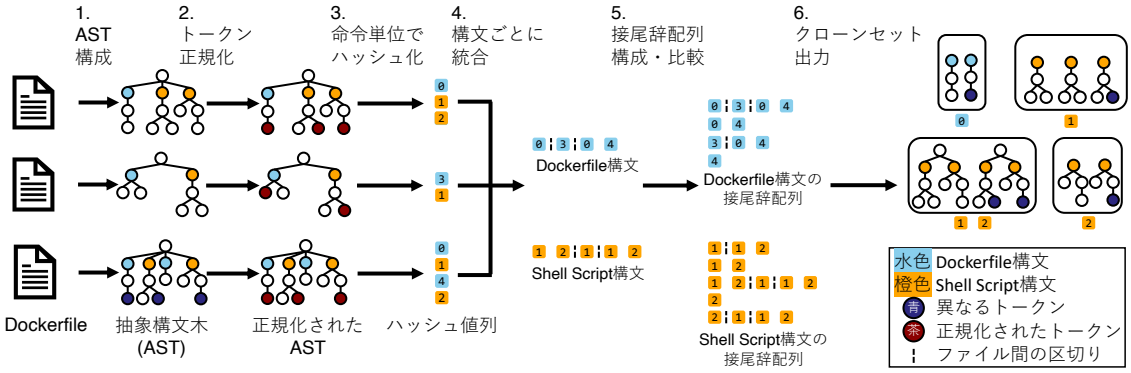

図 3: 提案手法の概要

表 2: 正規化対象とするトークン

トークン名	理由と正規化の例
一時変数名	コンテナ内の外部プロセスに影響を与えないため
	`ARG tag=bionic` → `ARG $$VAR0=bionic`
パラメタ・オプション順序	意味的に同値なコマンドを実行できるため
	`set -xe` → `set -e -x`
オプションのエイリアス	意味的に同値なコマンドを実行できるため
	`rm -fr` → `rm --force --recursive`
ファイルパス	パスの差異が振る舞いに影響を与えないため
	`cd webapps/convertigo` → `cd $$PATH0`
ユーザ・グループ	ユーザやグループの差異が振る舞いに影響を与えないため
	`USER spark:spark` → `USER $$USER0:$$GROUP0`
コンテナのポート番号	ポート番号の差異が振る舞いに影響を与えないため
	`EXPOSE 80` → `EXPOSE $$PORT0`
FROM 命令のタグ	タグによるコンテナの機能差がほとんど無いため
	`FROM ubuntu:16.04` → `FROM ubuntu:$$TAG0`

表 3: 正規化対象としないトークン

トークン名	理由
環境変数名	コンテナ内の外部プロセスに影響を与えるため
FROM 命令のイメージ名	イメージごとに内包されるパッケージマネージャが異なるため

```
3  FROM ubuntu:18.04
   ...
16 ARG PGP_KEYSERVER=ha.pool.sks-keyservers.net
   ...
59 RUN set -eux; ¥
60     ¥
61     savedAptMark="$(apt-mark showmanual)"; ¥
62     apt-get update; ¥
63     apt-get install --yes --no-install-recommends ¥
       ...
72     ; ¥
73     rm -rf /var/lib/apt/lists/*; ¥
       ...
83     for key in $OPENSSL_PGP_KEY_IDS; do ¥
84         gpg --batch --keyserver "$PGP_KEYSERVER" --recv-keys "$key"; ¥
85     done; ¥
   ...
255     chown -R rabbitmq:rabbitmq "$RABBITMQ_HOME"; ¥
   ...
292 EXPOSE 4369 5671 5672 15691 15692 25672
```

(a) 正規化前

```
3  FROM ubuntu:$$TAG0
   ...
16 ARG $$VAR0=ha.pool.sks-keyservers.net
   ...
59 RUN set -e -u -x; ¥
60     ¥
61     $$VAR1="$(apt-mark showmanual)": ¥
62     apt-get update; ¥
63     apt-get install --no-install-recommends --yes ¥
       ...
72     ; ¥
73     rm --force --recursive $$PATH0/*; ¥
       ...
83     for $$VAR2 in $OPENSSL_PGP_KEY_IDS; do ¥
84         gpg --batch --keyserver "$$VAR0" --recv-keys "$$VAR2"; ¥
85     done; ¥
   ...
255     chown -R $$USER0:$$GROUP0 "$RABBITMQ_HOME"; ¥
   ...
292 EXPOSE $$PORT0 $$PORT1 $$PORT2 $$PORT3 $$PORT4 $$PORT5 $$PORT6
```

茶色 抽象概念への変換
水色 オプションのエイリアス
下線 オプションの分離・順序変更

$OPENSSL_PGP_KEY_IDSは
環境変数であるため,
正規化を施さない.

(b) 正規化後

図 4: 実際の Dockerfile [*1] に対するトークン正規化の例

クローン片の一例である．

3.3 構文種別の分離

Dockerfile は Shell Script 構文を内包したネスト言語である [1] [17]．そのため，Dockerfile でトークンを正規化するならば，Dockerfile 構文における抽象構文木と Shell Script 構文の抽象構文木をそれぞれ構成しなければならない．

また，Dockerfile は，一方の構文で記述された命令列に対して，もう一方の構文で記述された命令及びその構文の命令列を挿入できる．例えば，Dockerfile 命令列に Shell Script 構文を内包した RUN 命令を挿入できる．一方，RUN 命令が内包する複数の Shell Script 命令列は，Shell Script 命令単位で分離可能である．したがって，RUN 命令を Shell Script 命令単位で分離して，Shell Script 命令列に Dockerfile 命令及び Dockerfile 命令列を挿入できる．一般的なプログラミング言語と同様に，Dockerfile における命令文単位の修正で生じる上記のような類似コードは，Type-3 クローンとして定義可能である．しかし，各構文に着目すると，構文ごとで命令文単位での修正は行われていない．したがって，本研究では，Dockerfile がネスト言語である特徴に着目し，構文種別が異なる命令列単位の修正によって発生する類似コード片は Type-2 クローンに分類する．

図 5 は，Shell Script 命令列に Dockerfile 命令列が挿入されたクローンの一例である．mpibench イメージ Dockerfile [*2] (図 5(b)) の 23 行目 ∼ 25 行目には, OpenFORM イメージ Dockerfile [*3] (図 5(a)) で出現しなかった Dockerfile 命令列が挿入されている．提案手法では，図 5 のようなクローンが検出可能である．

3.4 接尾辞配列の構成及び比較

Oumaziz らは，Dockerfile 開発者が長さが 6 未満の命令列も必要としていると述べている [16]．本研究では，接尾辞配列は可変長クローンを検出可能であるという特徴に着目した [34]．提案手法では，Dockerfile 命令及び Shell Script 命令 1 個ごとにハッシュ化を施しているため，任意の長さにおける Dockerfile 命令列や Shell Script 命令列をクローン片として検出できる．

なお，提案手法では，全ての Dockerfile のハッシュ値列を構文種別ごとに統合した巨大なハッシュ値列に対して，接尾辞配列アルゴリズムを適用する．単純な接尾辞配列アルゴリズムでは，異なるファイル間で統合されたコード片をクローンとして検出するため，ハッシュ値ごとに，そのハッシュ値の基となった構文が含まれている Dockerfile の情報を含める必要がある．

[*2] https://github.com/Azure/batch-shipyard/blob/master/recipes/mpiBench-Infiniband-MPICH/docker/Dockerfile

[*3] https://github.com/Azure/batch-shipyard/blob/master/recipes/OpenFOAM-Infiniband-OpenMPI/docker/Dockerfile

```
21 # set up ssh keys
22 RUN mkdir -p /var/run/sshd ¥
23     && ssh-keygen -A ¥
       ...
30     && chmod 700 /root/.ssh ¥
31     && cp /root/.ssh/id_rsa.pub /root/.ssh/authorized_keys
32
33 # download and install mlnx
34 RUN wget -q -O - http://www.mellanox.com/downloads/ofed/
   MLNX_OFED-4.6-1.0.1.1/MLNX_OFED_LINUX-4.6-1.0.1.1-rhel7.6-x86_64.tgz
    | tar -xzf - ¥
35     && ./MLNX_OFED_LINUX-4.6-1.0.1.1-rhel7.6-x86_64/mlnxofedinstall
    --user-space-only --without-fw-update --all --force ¥
36     && rm -rf MLNX_OFED_LINUX-4.6-1.0.1.1-rhel7.6-x86_64
```

(a) OpenFOAM イメージ Dockerfile [*3] から抽出したクローン片

```
11 # set up ssh keys
12 RUN mkdir -p /var/run/sshd ¥
13     && ssh-keygen -A ¥
       ...
20     && chmod 700 /root/.ssh ¥
21     && cp /root/.ssh/id_rsa.pub /root/.ssh/authorized_keys
22
23 # set up workdir
24 ENV INSTALL_PREFIX=/opt
25 WORKDIR /tmp/mpi
26
27 # download and install mlnx
28 RUN wget -q -O - http://www.mellanox.com/downloads/ofed/
   MLNX_OFED-4.6-1.0.1.1/MLNX_OFED_LINUX-4.6-1.0.1.1-rhel7.6-x86_64.tgz
    | tar -xzf - ¥
29     && ./MLNX_OFED_LINUX-4.6-1.0.1.1-rhel7.6-x86_64/mlnxofedinstall
    --user-space-only --without-fw-update --all --force ¥
30     && rm -rf MLNX_OFED_LINUX-4.6-1.0.1.1-rhel7.6-x86_64
```

(b) mpibench イメージ Dockerfile [*2] から抽出したクローン片

図 5: Shell Script 命令列に Dockerfile 命令列が挿入された Type-1 クローンの一例

4 実験

4.1 実験目的

実験の目的は，提案手法による Dockerfile の Type-2 クローン検出能力の確認である．比較対象としては，Oumaziz らによる Type-1 クローン検出手法 [16] を用いる．以降では，この手法を既存手法と呼ぶ．計測指標は，検出クローンの個数と適合率であり，Type-2 クローンは Type-1 クローンを包含するため提案手法の検出数が増えると期待される．適合率とは，実際にクローンである候補の割合である [37]．適合率が高いほど，よいクローン検出手法と言える．なお対象とした Dockerfile における真のクローンの集合は未知であるため，検出したクローンセットから各手法ごとでランダムに 101 個抽出し，目視確認により適合率を算出する．

4.2 実験対象

GitHub 上で人気がある 223 リポジトリのうち，それらのリポジトリ内に存在する 1,897 ファイルを対象とする．なお，Docker ではテンプレートファイルにコンテナ構築の全体の流れを記述し，ディストリビューションやバージョンの細かな差異を可変変数で定義する，というプラクティスが存在する．このテンプレートを実行，あるいはコンパイルすることで目的の Dockerfile を自動生成する．この自動生成された Dockerfile は Type-2 クローンの関係にあるが，自動生成を理由としたクローンであり検出対象とすべきではない．よって，テンプレートファイルが存在するリポジトリに対しては，そのリポジトリに存在する Dockerfile ではなく，テンプレートそのものを対象とする．

多くの Dockerfile では Shell Script 構文として Bash shell を採用している [1] [17]．したがって，本実験では Shell Script 構文を Bash に限定する．また，抽象構文木構成のために構文解析を行う Bash コマンドを，Henkel らが調査した Dockerfile で頻繁に用いられる上位 50 種類に限定する [1] [17]．なお，クローン検出において，転置行列の大きさは通常 7 または 10 以下でなければならない [28]．そのため，既存手法における転置索引の大きさは，彼らの論文で採用された値である 6 とする．

4.3 実験結果

4.3.1 検出できるクローンの個数及び精度

表 4 は，実験で検出できたクローン片やクローンセット，適合率を載せた実験結果である．

まず，既存手法と正規化ありの提案手法について比較すると，提案手法が検出したクローン片の数及びクローンセット数は，既存手法で検出できたクローン片及びクローンセット数よりも多い．提案手法では，トークンを適切に正規化して，接尾辞配列を用いて任意の長さを持つクローン片を検出できる．

```
23  RUN set -ex ¥
        ...
29      && wget -O python.tar.xz
     "https://www.python.org/ftp/python/${PYTHON_VERSION%%[a-z]*}/
    Python-$PYTHON_VERSION.tar.xz" ¥
```

(a) 正規化前の alpine 版 python イメージ Dockerfile [*4]

```
23  RUN set -ex ¥
        ...
29      && wget -O $$PATH0
     "https://www.python.org/ftp/python/${PYTHON_VERSION%%[a-z]*}/
    Python-$PYTHON_VERSION.tar.xz" ¥
```

(b) 正規化後の alpine 版 python イメージ Dockerfile

```
11  RUN apt-get update && apt-get install -y --no-install-recommends ¥
            ...
15      && rm -rf /var/lib/apt/lists/*
    ...
20  RUN set -ex ¥
21      ¥
22      && wget -O python.tar.xz
     "https://www.python.org/ftp/python/${PYTHON_VERSION%%[a-z]*}/
    Python-$PYTHON_VERSION.tar.xz" ¥
```

(c) 正規化前の debian 版 python イメージ Dockerfile [*5]

```
11  RUN apt-get update && apt-get install -y --no-install-recommends ¥
            ...
15      && rm -rf $$PATH0/*
    ...
20  RUN set -ex ¥
21      ¥
22      && wget -O $$PATH1
     "https://www.python.org/ftp/python/${PYTHON_VERSION%%[a-z]*}/
    Python-$PYTHON_VERSION.tar.xz" ¥
```

(d) 正規化後の debian 版 python イメージ Dockerfile

図 6: 抽象化されたトークン名が異なる Type-1 クローンの一例

```
1 FROM scratch
  ...
3 ADD rootfs.tar.xz /
4 CMD ["/bin/bash"]
```

(a) CRUX イメージ [*6] の Dockerfile から抽出したクローン片

```
1 FROM scratch
  ...
3 ADD smgl-stable-0.62-docker-x86_64.tar.xz /
4 CMD ["/bin/bash"]
```

(b) Source Mage イメージ [*7] の Dockerfile から抽出したクローン片

図 7: 提案手法で発見した定型処理（Type-2）

そのため，提案手法が既存手法より多くのクローンを検出できるという期待にそった結果となった．

次に，正規化を施した提案手法と正規化を施さない提案手法を比較すると，正規化を施した提案手法の方が，正規化を施さない提案手法よりも検出できるクローン片の数及びクローンセット数が少ない．正規化ありの提案手法（Type-2）は正規化なしの提案手法（Type-1）を包含するためクローンの検出数が多い，という期待を反する結果となった．その原因として，正規化前のトークン名が同じであるにも関わらず，正規化処理によって抽象化されたトークン名に差異が生じたためと考えられる．図 6 は，本実験における正規化処理の影響で検出できなかったクローンの一例である．alpine 版 python イメージ Dockerfile [*4]（図 6(a)）における 29 行目と debian 版 python イメージ Dockerfile [*5]（図 6(c)）における 22 行目は Type-1 クローンである．しかし，本実験では，図 6(b) 及び図 6(d) のように，トークンの出現順にナンバリングするという正規化処理を施すため，抽象化されたトークン名に差異が生じる．抽象化されたトークン名の差異により，正規化を施す提案手法が，正規化を施さない提案手法で検出できた Type-1 クローンを検出できなくなるため，正規化なしの提案手法（Type-2）が正規化なしの

表 4: 検出できたクローン数及び適合率

		クローン片数	クローンセット数	適合率
Oumaziz らの手法 [16]		24,687	4,630	100.00 %
提案手法（正規化なし）	Dockerfile 構文	93,435	18,884	100.00 %
	Shell Script 構文	211,104	56,118	100.00 %
提案手法（正規化あり）	Dockerfile 構文	94,153	18,757	95.05 %
	Shell Script 構文	207,748	54,369	98.02 %

提案手法（Type-1）を包含できない．図 6 のように，抽象化されたトークン名の差異によって引き起こされるクローンの検出漏れを解決するため，クローン片の数及びクローンセット数が減少しないように改善した正規化処理の提案は今後必須である．

最後に，既存手法と正規化ありの提案手法の適合率について比較すると，正規化を施した提案手法の適合率は既存手法の適合率より低い．一般的に正規化を施すほど適合率は下がりやすい．Type-1 クローンよりも Type-2 クローンが，Type-2 クローンよりも Type-3 クローンの方が検出は困難となる．しかしながら，提案手法は適合率 95 % と高く，Type-2 クローン検出手法として有用であると言える．

4.3.2 検出した Type-2 クローンの一例

本実験では，提案手法で検出した Type-2 クローンの内，Dockerfile における定型処理を発見した．図 7 は，提案手法で発見した，定型処理を含む Type-2 クローンの一例である．どちらの Dockerfile も，scratch と呼ばれる空のコンテナへ，tar アーカイブをホスト OS からコピーし，その tar アーカイブをコンテナ上で展開して，新たなディストリビューションを構成する．

*4 https://github.com/docker-library/python/blob/master/Dockerfile-alpine.template
*5 https://github.com/docker-library/python/blob/master/Dockerfile-debian.template
*6 https://github.com/cruxlinux/docker-crux/blob/master/Dockerfile
*7 https://github.com/vaygr/docker-sourcemage/blob/master/stable/Dockerfile

5　おわりに

本研究では，Dockerfile における Type-2 クローンの定義，及び Dockerfile 固有の文法構造に着目した Type-2 クローンの検出手法を提案した．提案手法を用いて，223 個の GitHub リポジトリ上に存在する 1,897 ファイルに対して実験を行ったところ，Type-2 クローンを検出できた．また，提案手法の適合率が Dockerfile 構文及び Shell Script 構文のそれぞれで 95 % 以上であったため，提案手法が有効であるといえた．

本研究の今後の課題としては以下が考えられる．

- トークン正規化による検出クローンの減少：4.3.1 節で述べた通り，本研究の実験では，トークン正規化によって検出できるクローンが減少した．そのため，この問題を改善する手法の提案は今後必須である．1 つの解決策として，一次変数同士を同一視してクローン検出するというように，正規化した同種トークンを区別しない方法が考えられる．しかし，この解決策ではトークン種別以外を同一視するため，適合率が低くなると考えられる．
- 構文を分離しない Type-2 クローン検出：提案手法では，両構文を分離して各構文ごとにクローンを検出しているため，Dockerfile 命令列と Shell Script 命令列の双方を含むクローンを検出できない．Dockerfile 構文と Shell Script 構文が一式となるクローンも活用できると考えられるため，両構文を含有するクローンも検出する必要がある．
- Type-3 クローンへの拡張：Dockerfile には，パッケージマネージャの違いにより発生するクローンが存在する．パッケージマネージャによって必要な Shell Script コマンドが異なるため，文単位で修正されたクローンを検出する手法が必要である．

謝辞

楠本真二教授は，私が弊学に入学した時からお世話になり，研究室配属後も本研究を進めるにあたって直接，間接の御支援をくださりました．

肥後芳樹准教授は，本研究におけるコードクローン手法への助言など，本研究のキーアイデア面において有益かつ的を射た御進言をくださりました．

柗本真佑助教は，Docker をはじめとした技術の提供や研究への向き合い方，そして論文執筆など，本研究の大部分において多数の貴重な御指導をくださりました．

大阪大学大学院情報科学研究科コンピュータサイエンス専攻博士前期課程 2 年の中川将氏は，本研究に関する相談やコードクローン検出の実装方法に関する助言など，本研究を進めるにあたって多大なるご助力をくださいました．

大阪大学大学院情報科学研究科コンピュータサイエンス専攻博士前期課程 2 年の華山魁生氏は，本研究の実験で用いたデータセットの用意など，本研究の Docker に関する事柄でお力添えをくださいました．

また，本研究を進めるにあたってお世話になった楠本研究室の皆様に心から感謝申し上げます．

最後に，本研究に至るまでに講義・演習・実験等でお世話になりました，大阪大学大学院情報科学研究科の諸先生方に，この場をお借りして感謝の言葉を述べさせていただきます．

参考文献

[1] Jordan Henkel, Christian Bird, Shuvendu K Lahiri, and Thomas Reps. A dataset of dockerfiles. In *Proceedings of International Conference on Mining Software Repositories*, pp. 528–532, 2020.

[2] Yujuan Jiang and Bram Adams. Co-evolution of infrastructure and source code-an empirical study. In *Proceedings of Working Conference on Mining Software Repositories*, pp. 45–55, 2015.

[3] Matej Artac, Tadej Borovssak, Elisabetta Di Nitto, Michele Guerriero, and Damian Andrew Tamburri. Devops: introducing infrastructure-as-code. In *Proceedings of International Conference on Software Engineering Companion*, pp. 497–498, 2017.

[4] Portworx. Annual container adoption report. `https://portworx.com/wp-content/uploads/2019/05/2019-container-adoption-survey.pdf`, 2019. [Online; accessed 18. Jan. 2021].

[5] Jürgen Cito, Gerald Schermann, John Erik Wittern, Philipp Leitner, Sali Zumberi, and Harald C Gall. An empirical analysis of the docker container ecosystem on github. In *Proceedings of International Conference on Mining Software Repositories*, pp. 323–333, 2017.

[6] Yang Zhang, Bogdan Vasilescu, Huaimin Wang, and Vladimir Filkov. One size does not fit all: An empirical study of containerized continuous deployment workflows. In *Proceedings of European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 295–306, 2018.

[7] Ramtin Jabbari, Nauman bin Ali, Kai Petersen, and Binish Tanveer. What is devops? a systematic mapping study on definitions and practices. In *Proceedings of the Scientific Workshop Proceedings of XP2016*, pp. 1–11, 2016.

[8] Stefan Bellon, Rainer Koschke, Giulio Antoniol, Jens Krinke, and Ettore Merlo. Comparison and evaluation of clone detection tools. *IEEE Transactions on Software Engineering*, Vol. 33, No. 9, pp. 577–591, 2007.

[9] Abdullah Sheneamer and Jugal Kalita. A survey of software clone detection techniques. *International Journal of Computer Applications*, Vol. 137, No. 10, pp. 1–21, 2016.

[10] Hou Min and Zhang Li Ping. Survey on software clone detection research. In *Proceedings of International Conference on Management Engineering, Software Engineering and Service Sciences*, pp. 9–16, 2019.

[11] Andrew Walker, Tomas Cerny, and Eungee Song. Open-source tools and benchmarks for code-clone detection: past, present, and future trends. *ACM SIGAPP Applied Computing Review*, Vol. 19, No. 4, pp. 28–39, 2020.

[12] Iman Keivanloo, Juergen Rilling, and Ying Zou. Spotting working code examples. In *Proceedings of International Conference on Software Engineering*, pp. 664–675, 2014.

[13] Norihiro Yoshida, Seiya Numata, Eunjong Choiz, and Katsuro Inoue. Proactive clone recommendation system for extract method refactoring. In *Proceedings of International Workshop on Refactoring*, pp. 67–70, 2019.

[14] Manishankar Mondal, Chanchal K Roy, and Kevin A Schneider. An exploratory study on change suggestions for methods using clone detection. In *Proceedings of International Conference on Computer Science and Software Engineering*, pp. 85–95, 2016.

[15] Akito Monden, Satoshi Okahara, Yuki Manabe, and Kenichi Matsumoto. Guilty or not guilty: Using clone metrics to determine open source licensing violations. *IEEE software*, Vol. 28, No. 2, pp. 42–47, 2010.

[16] Mohamed A Oumaziz, Jean-Rémy Falleri, Xavier Blanc, Tegawendé F Bissyandé, and Jacques Klein. Handling duplicates in dockerfiles families: Learning from experts. In *Proceedings of International Conference on Software Maintenance and Evolution*, pp. 524–535, 2019.

[17] Jordan Henkel, Christian Bird, Shuvendu K Lahiri, and Thomas Reps. Learning from, understanding, and supporting devops artifacts for docker. In *Proceedings of International Conference on Software Engineering*, pp. 38–49, 2020.

[18] Shane McIntosh, Martin Poehlmann, Elmar Juergens, Audris Mockus, Bram Adams, Ahmed E Hassan, Brigitte Haupt, and Christian Wagner. Collecting and leveraging a benchmark of build system clones to aid in quality assessments. In *Proceedings of International Conference on Software Engineering*, pp. 145–154, 2014.

[19] Christoph Domann, Elmar Juergens, and Jonathan Streit. The curse of copy&paste—cloning in requirements specifications. In *Proceedings of International Symposium on Empirical Software Engineering and Measurement*, pp. 443–446, 2009.

[20] Elmar Juergens, Florian Deissenboeck, Martin Feilkas, Benjamin Hummel, Bernhard Schaetz, Stefan Wagner, Christoph Domann, and Jonathan Streit. Can clone detection support quality assessments of requirements specifications? In *Proceedings of International Conference on Software Engineering*, pp. 79–88, 2010.

[21] Hui Liu, Zhiyi Ma, Lu Zhang, and Weizhong Shao. Detecting duplications in sequence dia-

grams based on suffix trees. In *Proceedings of Asia Pacific Software Engineering Conference*, pp. 269–276, 2006.

[22] Harald Störrle. Towards clone detection in uml domain models. *Software and Systems Modeling*, Vol. 12, No. 2, pp. 307–329, 2013.

[23] Mohamed Oumaziz, Alan Charpentier, Jean-Rémy Falleri, and Xavier Blanc. Documentation reuse: Hot or not? an empirical study. In *Proceedings of International Conference on Software Reuse*, pp. 12–27, 2017.

[24] Florian Deissenboeck, Benjamin Hummel, Elmar Jürgens, Bernhard Schätz, Stefan Wagner, Jean-François Girard, and Stefan Teuchert. Clone detection in automotive model-based development. In *Proceedings of International Conference on Software Engineering*, pp. 603–612, 2008.

[25] Docker. Docker overview. `https://docs.docker.com/get-started/overview/`. [Online; accessed 18. Jan. 2021].

[26] Docker. What is a container? `https://www.docker.com/resources/what-container/`. [Online; accessed 18. Jan. 2021].

[27] Docker. Dockerfile reference. `https://docs.docker.com/engine/reference/builder/`. [Online; accessed 20. Jan. 2021].

[28] Benjamin Hummel, Elmar Juergens, Lars Heinemann, and Michael Conradt. Index-based code clone detection: incremental, distributed, scalable. In *Proceedings of International Conference on Software Maintenance*, pp. 1–9, 2010.

[29] Docker. Best practices for writing dockerfiles. `https://docs.docker.com/develop/develop-images/dockerfile_best-practices/`. [Online; accessed 18. Jan. 2021].

[30] Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. *Siam Journal on Computing*, Vol. 22, No. 5, pp. 935–948, 1993.

[31] Simon J Puglisi, William F Smyth, and Andrew H Turpin. A taxonomy of suffix array construction algorithms. *Acm Computing Surveys*, Vol. 39, No. 2, pp. 4–es, 2007.

[32] Ge Nong, Sen Zhang, and Wai Hong Chan. Linear suffix array construction by almost pure induced-sorting. In *Proceedings of Data Compression Conference*, pp. 193–202, 2009.

[33] Ge Nong, Sen Zhang, and Wai Hong Chan. Two efficient algorithms for linear time suffix array construction. *IEEE Transactions on Computers*, Vol. 60, No. 10, pp. 1471–1484, 2010.

[34] Hamid Abdul Basit and Stan Jarzabek. Efficient token based clone detection with flexible tokenization. In *Proceedings of European Software Engineering Conference and the ACM*

SIGSOFT Symposium on the Foundations of Software Engineering, pp. 513–516, 2007.

[35] Guanhua Li, Yijian Wu, Chanchal K Roy, Jun Sun, Xin Peng, Nanjie Zhan, Bin Hu, and Jingyi Ma. Saga: Efficient and large-scale detection of near-miss clones with gpu acceleration. In *Proceedings of International Conference on Software Analysis, Evolution and Reengineering*, pp. 272–283, 2020.

[36] Chanchal K Roy, James R Cordy, and Rainer Koschke. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Computer Programming*, Vol. 74, No. 7, pp. 470–495, 2009.

[37] Stéphane Ducasse, Oscar Nierstrasz, and Matthias Rieger. On the effectiveness of clone detection by string matching. *Journal of Software Maintenance and Evolution: Research and Practice*, Vol. 18, No. 1, pp. 37–58, 2006.