ヒューマンインタフェース 62−11
（1995. 9. 14）

システムの状態と依存関係に基づくコマンド予測

久保 康司† 山本 浩一† 守田 了‡ 田中 稔‡
kubo@cs.csse.yamaguchi-u.ac.jp

†山口大学大学院工学研究科 ‡山口大学工学部
〒755 宇部市常盤台 2557

コマンド実行時のシステムの状態とコマンドの依存関係に基づいてコマンドを予測する手法を提案する．コマンド予測では，一連の作業に使用するコマンド列を抽出することが重要である．本手法では，コマンド履歴をコマンドをノード，システムの状態を元にしたコマンドの依存関係をアークとするグラフに変換し，出力弧が複数存在するノードでグラフを分解する．分解されたグラフを一連の作業に使用するコマンド列として抽出する．そのため，コマンド履歴中の繰り返しを検索するアプローチと異なり，入力コマンドが少ない場合でも実行回数の少ないコマンドを抽出できる．また，コマンドインタフェースは複数の作業を並行して行えるため，実行順序に依らないコマンドの依存関係の抽出が重要である．本手法では，定義されたコマンドの機能を記述したコマンド辞書を用いずに，コマンド実行時のシステムの状態からコマンドの依存関係を抽出するため，ユーザ独自のコマンドやシェルスクリプトを含むコマンドを予測できる．本手法を実際のコマンド履歴ならびに人工的に生成したコマンド履歴を用い，FRQ, LRU 法と比較し，本方法の有効性を確認した．

Command Prediction Based on System's State and Commands Dependence

Kouji KUBO Kouichi YAMAMOTO Satoru MORITA Minoru TANAKA

Yamaguchi University
2557 Tokiwadai, Ube-shi, Yamaghchi, 755, Japan

We propose command prediction method based on system's state and commands dependence. It is important to extract command sequences used in a sequential work. In this method, command history is converted into directed graph, that the node is command and the arc is command dependence. The graph is decomposed of the node with two or more output arcs. Command sequences are extracted from the decomposed graph. Therefore, the command with a little number of executions can be extracted. Moreover, because two or more work can be done in command interface concurrently, it is important to extract the dependence of the commands which does not depend on the execution order. Because the command dependence is extracted from the state of the system observed at command execution without using the dictionary which describes the function of the command, the command which contains the user-tailored shell script and command can be predict. This command prediction method is compared with FRQ and LRU method using actual command history and the command history generated artificially. It is found the effectiveness of this method.

1 はじめに

パーソナルコンピュータの普及によるユーザ層の拡大とアプリケーションプログラムの巨大化に伴い，ユーザインタフェースに対する支援機能の重要性が高まっている．Croft[1] は，ユーザインタフェースにおける文脈の利用とユーザへの適応の必要性を説いている．

ユーザが次に使用するコマンドを提案するインタフェースは主に 2 つの方法で実現されている．1 つは支援のベースとなる情報として入力コマンド系列を使用する方法である [2] [5]．[2] は，コマンド履歴をリストに保存し，繰り返しを抽出しコマンドを予測するが，コマンドを同じ順序で実行しなければ繰り返しと判定されず支援が行えない欠点がある．瀬下 [5] は Edwin[2] でリストに保存されていた操作履歴をコマンド間の依存関係で結んだグラフで表現することで，実行順序の変化への対応している．これらの手法は新たな入力の度にリストやグラフの照合を行ない，繰り返しを抽出する．このため，入力コマンド数が少ない場合や実行回数の少ないコマンド列の場合予測が行なえなかった．

もう 1 つはあらかじめ作業に関係のあるコマンドや操作手順などの支援情報を持ちこれに応じてユーザにガイダンスを与えるものである [1]．支援情報にコマンド実行に必要な前提条件や次に起動されるべきコマンドなど詳細な情報を記述する事できめ細かい支援を可能としている．支援情報をあらかじめ持つため，支援時の計算コストは小さいが，支援情報の記述に専門的な知識が必要であり，個人ユーザに適応するための支援情報のカスタマイズや新しいコマンドへの対応には限界がある．

本研究では，前者の立場を取り，コマンドインタフェースを対象とし，ユーザの入力を基にコマンドの予測を行なうが，コマンドの依存関係を表現したグラフを出力弧が複数存在するノードで分解し，各作業に使用されたコマンド列を得る．このため繰り返しを検索する他のアプローチと異なり，入力コマンドが少ない場合でも実行回数の少ないコマンド列を抽出可能である．

また，コマンドインタフェースは複数のコマンドが並行して実行でき，実行順序が複数存在する作業があるため，コマンドの実行順序に依らないコマンド間の依存関係の判定が重要である．従来法では，予測のベースに履歴を使用しても，依存関係はコマンドの機能を記述したコマンド辞書を用いて判定していた．このため，ユーザ独自のコマンドやシェルスクリプトに対しては，コマンド辞書の再定義が必要であった．本手法ではコマンド実行時の計算機システムの状態を検査する事で，コマンドの依存関係を抽出するため，ユーザ定義のコマンドやシェルスクリプトにも対応可能である．

以下では UNIX コマンドを対象として，2 システムの状態とコマンドの依存関係について，3 コマンド連鎖グラフを用いた作業系列の抽出と分類について説明する．4 で本手法を人工的に生成したコマンド履歴と実際のコマンド履歴を用いて FRQ, LRU 法 [4] と比較した結果，本方法の有効性が確認された．5 ではまとめを行う．

2 システムの状態とコマンド間の依存関係

2.1 システムの状態とコマンドの実行

図 1の (1) から (6) は UNIX システム上でプログラムの作成，(7) から (11) は文書の作成で使用されたコマンド列の例である．粒度の小さいコマンドを使用する場合，ユーザは目的を達成するために，複数のコマンドから成るコマンド列を実行する．(1) から (6) では，作業を行なうディレクトリに移動し，ディレクトリの内容を確認 (2) した後，二つのソースファイルを編集 (3,4) し，コンパイル (5)，実行 (6) している。これらの各コマンドは目的を部分的に達成するコマンドであり，先に実行されたコマンドによって後で実行するコマンドの前提条件を整えている．計算機を使用する場合，この様なほぼ定型といえる一連の操作の実行は頻繁に存在する．計算機上で行なわれる作業は，“a. 作業を行なうための環境に移動する.”，“b. 必要な作業を実現する.” の 2 つの動作を繰り返し実行している．

入力されるコマンドのうち，図 1の (1) から (6) や (7) から (11) の様な関連のあるコマンドを自動的に抽出するためには，コマンド間の依存関係を判定する必要がある．2 つのコマンド A と B 間の依存関係は，コマンド A の動作と，コマンド

```
(1) cd example
(2) ls
(3) vi main.c
(4) vi sub.c
(5) gcc -o test main.c sub.c
(6) test
(7) cd txt
(8) vi repo.tex
(9) jlatex repo.tex
(10) jdvi2kps repo.dvi > repo.ps
(11) lpr repo.ps
(12) cd ..
(13) vi ex01.c
(14) gcc -o ex01 ex01.c
(15) ex01
```

図 1: コマンド列の例

B を実行するための前提条件を調べ，A の動作が，B の前提条件を満たしているかを調べることで判定できる．この依存関係の判定には各コマンドの前提条件と動作の記述が必要であるが，中山 [6] ではコマンド毎に用意したファイル依存関係テンプレートを用意し，実行に関係するファイルの推定を行なっている．このようにコマンドの入出力ファイルに関する情報をあらかじめ用意する方法では，ユーザが独自に作成したコマンドやシェルスクリプトへの対応が困難である．このため，コマンドの入出力動作の推定は，実行時のシステムの状態の変化を観察することで行なった．

2.2 コマンド間の依存関係

コマンド実行において，コマンドは計算機システム上のある状態 (S_0) の元で実行され，ファイルや，プロセスの状態，画面の表示などの状態を変化させる．こうして，コマンドの実行が行われる度に計算機の状態は $S_1, S_2, ...$ と遷移する．このため，以前に実行されたコマンドがシステムにもたらす状態の変化と，これから実行するコマンドの前提条件を調べることで，一連の入力コマンドから依存関係のあるコマンドを抽出する．しかしながら，コマンドがシステムにもたらす変化は多種多様であり，システムの全てを検査することは事実上不可能である．UNIX コマンドの予測を行う場合，実行されるコマンドとそのパラメータを予測すればいいため，システムの検査を行うキーを入力されたコマンドに現れるトークンに限定する事で検査の範囲を狭めることが出来る．実験では，コマンドに現れるトークンの文字列をキーにしてファイルシステムのカレントディレクトリの

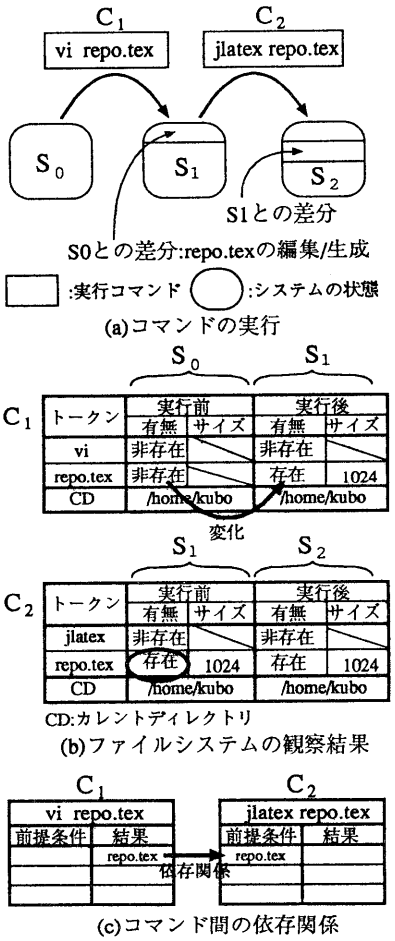


図 2: コマンド間の依存関係の取得

検査を行う事で，システムの状態の変化を取得した．このため，カレントディレクトリの位置も検査の項目に加えた．図 2に依存関係の検査の様子を示した．(a) でまず，状態 S_0でコマンド C_1 vi repo.tex が実行されるとシステムは状態 S_1に変化する．この時ファイルシステムの検査によって，(b) の C_1の表を得ることが出来る．C_1の表の実行前の列は状態 S_0 での観察結果であり，実行後の列は S_1での観察結果である．実行前と実行後 2 つの列を比較することでコマンドの実行結果を推定する．この結果ファイル repo.tex の編集または生成が検出される．次に，コマンド C_2 jlatex repo.tex が実行された場合も同様の検査を行なう ((b)C_2)．この時，コマンド実行前にファイルの存在が確認

されたトークンは前提条件 (repo.tex) とする．C_2 の結果は，トークンに現れないため実行前後の観察結果によって検出されない．(a) のコマンド実行に対して (c) に示した表を各コマンドについて得ることが出来る．この表の，2つのコマンド C_i, $C_j(i < j)$ についてjの依存するコマンドを調べる場合，C_iの結果と C_jの前提条件に同じ項目がある場合 C_j は C_iに依存しているとする．但し，C_jの同じトークンに対する前提条件を満たすコマンドが複数存在する最新のコマンドに対して依存関係があるとする．

各コマンドの前提条件と結果の検出は次の基準で決定する．

前提条件 コマンドに現れるトークンの文字列をキーとして，ファイルシステムのカレントディレクトリの検査をし，有効な検査結果が得られた項目とカレントディレクトリを前提条件とする．

結果 コマンド実行後にもう一度ファイルシステムを検査し，実行前の検査結果との差 (ファイルの有無やサイズの変化，カレントディレクトリの変化) が検出された場合これをコマンドの結果とする．

3 コマンド連鎖グラフを用いた作業系列の抽出

操作履歴をコマンド予測などのユーザ支援のベースとして使用する場合，その記録方法によって記憶効率や支援能力が左右される．UNIX システムの csh, tcsh に代表されるコマンドインタフェース上の操作は，以下に示す特徴を持つ．

1. ユーザはコマンドを組み合わせて作業を達成する．
2. 現在の入力コマンドによって以後のコマンド入力が明示的に制約されないため，複数の作業を並行して実行する．
3. 各作業を構成するコマンド間に明確な実行順序が存在しない場合，作業の実行毎にコマンドの実行順序の変化が生じる場合がある．

本手法では，過去に入力されたコマンド列から，各々の作業に使用されたコマンド列を抽出するため，計算機上で行われる作業について次の仮定を設ける．

仮定 1 作業は1つ以上の依存関係を持つコマンドによって構成される．

仮定 2 作業には開始点があり作業内の全てのコマンドは開始点に相当するコマンドに依存する．

仮定1は入力されたコマンド列のなかに実行順序に制約を加えられたコマンド群が存在する事を仮定している．仮定2では特定の作業に使用されるコマンドの全てに最低1つは共通した前提条件を持つことを仮定している．通常コマンドの実行は，プログラム作成やレポート作成などの作業ごとに異なったディレクトリで実行されるため妥当であると考える．

[1][5] ではコマンド履歴をリストやグラフで表現し，新たな入力に対してそのつど，リストやグラフのマッチングにより繰り返し動作やプログラムコンポーネントの抽出し，コマンドの予測を行なっていた．

本手法では，コマンド履歴をコマンドをノード，依存関係をアークとするグラフで表現し，出力弧が複数存在するノードでグラフを分解する事で，作業に使用されたコマンド群をタスクとして抽出する．タスクはそれが使用されたシステムの状態によって分類される．コマンドの予測は，コマンドを実行時のシステムの状態と，予め分類されたタスクによって行う．

3.1 コマンド連鎖グラフ

まず，入力されたコマンドを依存関係に基づく有効グラフ $G = (V, A)$ で記述する．ここで，$v_i \in V$は入力されたコマンドに相当し，入力された順に順序付けられているものとする．$(v_i, v_j) \in A, i < j$が v_jが v_iに依存しているという，コマンド間の依存関係を表現している．このグラフをコマンド連鎖グラフと呼ぶこととする．

コマンド連鎖グラフは，入力されたコマンドの依存関係を表現し，ノード v_i からアークを逆方向に辿ることで得られるノードの集合 $S \subseteq V$が v_iに相当するコマンドの実行のために，前もって実行されたコマンドの集合を表現している．ノード vの入力弧の数，出力弧の数を定義する関数，

$$\delta^+(v) = v\text{の入力弧の数}$$

$$\delta^-(v) = v\text{の出力弧の数}$$

を定め，2ノード間のアークの距離を1とすると，$v \in S$は，$\delta^-(v) = 0$であるノードからの最大距離が小さい順に順序付けられる．図3は図1から生成したコマンド連鎖グラフである．

図3から$\delta^+(v_i) = 0$であるノードを選んでコマンド列を復元すると図4が得られる．図内で各コマンドの前に"[]"で付けられた番号は，コマンドの実行順序であり，"()"で付けられた番号は図1でコマンドに付けられた番号である．

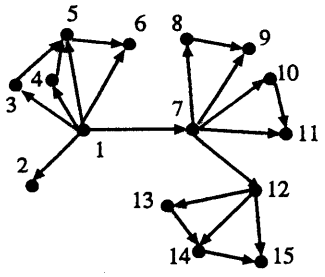


図 3: 図1から生成したコマンド連鎖グラフ

コマンド列1	コマンド列2
[1](1) cd example [2](2) ls	[1](1) cd example [2](3) vi main.c [2](4) vi sub.c [3](5) gcc -o test main.c sub.c [4](6) test
コマンド列3	コマンド列4
[1](1) cd example [2](7) cd txt [3](8) vi repo.tex [4](9) jlatex repo.tex	[1](1) cd example [2](7) cd txt [3](10) jdvi2kps repo.dvi > repo.ps [4](11) lpr repo.ps
コマンド列5	
[1](1) cd example [2](7) cd text [3](12) cd ..	[4](13) vi ex01.c [5](14) gcc -o ex01 ex01.c [6](15) ex01

図 4: 図3から得られるコマンド列

3.2 作業系列の抽出と分類

3.1で復元されたコマンド列 (図4) はコマンド列 3,4,5 に別の作業で使用されたコマンド cd example, cd txt が含まれている．これはコマンド連鎖グラフに特定の作業で使用されたコマンド間の依存関係とともに，作業間の遷移も保存されているためである．そこで，作業間の遷移と作業内のコマンド間の依存関係を切り分けるために以下の方法をとる．

コマンド連鎖グラフにおいて$\delta^+(v) \geq 2$であるノードvは，仮定2に基づけば作業の開始点と成るコマンドであり，このコマンドに依存する作業を代表するコマンドであると考える (図1では (1), (7) と (12) がこれにあたる)．図5(a)で白丸で示したものがこれらのノードである．黒丸は$\delta^+(v) \leq 1$なるノードである．$\delta^+(v) \geq 2$であるvだけから成る部分グラフが図5(b)である．UNIX コマンドでは，同じ効果を持つコマンドが複数存在するため，コマンドの前提条件と結果を出力する関数$M(v)$を定義し，図5(b)内の前提条件と結果が等しいノードを1つのノードにまとめる (図5(c))．この操作によってユーザが行う作業の遷移が得られる．このグラフを作業遷移グラフGSと呼ぶ．次に，コマンド連鎖グラフから$v \in GS$を取り除いて得られる各連結部分グラフ，$GT = (VT, AT)$は個々の作業の詳細を表現しているため，タスクと呼ぶことにする．$v, v \in GS$とGTとのコマンド連鎖グラフ内での依存関係を関係を$R(v, GTk)$によって保存する．以上の操作でコマンド連鎖グラフは

$$G = (V, A)$$

であり，作業遷移グラフが

$$GS = (VS, AS)$$

但し

$$v_j \in VS, v_i \in V, \delta^+(v_i) \geq 2, v_j = M(v_i)$$

である時，タスクGTは連結であり

$$GT \subset (V - V', A - A')$$

但し

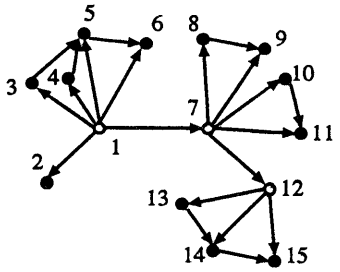
$$V' = \{v \in V, \delta^+(v_i) \geq 2\}$$

$$A' = \{(v_i, v_j) | (v_i, v_j) \in A, \delta^+(v_i) \geq 2 \vee \delta^+(v_j) \geq 2$$

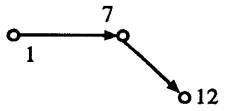
である．これらの関係は

$$R(v, GT) \text{ 但し } v \in GS$$

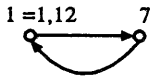
である．以後，GS, GT, Rを作業情報と呼ぶ．これによって，作業間の遷移に使用されたコマンドと作業に使用されたコマンドを分離して管理する．



(a) コマンド連鎖グラフ内の分岐点



(b) 分岐点から成る部分グラフ



(c) (b)内の等値なノードの統合

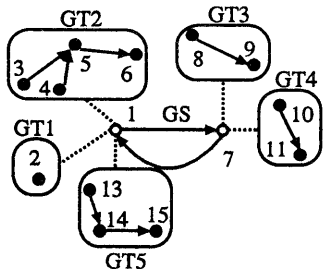
図 5: 作業の遷移の抽出

図 6は図 1から得られた作業情報であり，図 7はこれから得られたコマンド列である．図 4と比較すると作業内のコマンドだけが分離されていることが分かる．

3.3 コマンドの予測

3.2 で作成した作業情報を用いたコマンド予測の方法を説明する．現在までのコマンド入力によって作業情報が生成されているとし，次に入力されるコマンドの候補を次の手順で決定する．

1. 現在のコマンドの結果を用いて同じ結果を持つノード $v \in VS$を検索する．この様な vが見つからない場合，予測の提示は行なわず，新たなノードをコマンド連鎖グラフに追加し，状態グラフ，タスク，関係を更新する．
2. GS内で vの出力弧で直接結ばれるノードの集合 (PGS) を得る．
3. ノード vを用いて関係 Rを検索し，関係のあるタスクの集合 (PGT) を得る．
4. 2 で選ばれたノードと，3 で選ばれたタスクに含まれるノードを予測の候補とする．



GS=({1,7},{(1,7),(7,1)})
GT1=({2},{})
GT2=({3,4,5,6},{(4,5),(3,5),(5,6)})
GT3=({8,9},{(8,9)})
GT4=({10,11},{(10,11)})
GT5=({13,14,15},{(13,14),(14,15)})
R={(1,GT1),(1,GT2),(1,GT5),(7,GT3),
(7,GT4)}

図 6: 図 1 から作られた作業情報

コマンド列1	コマンド列2
[1](2) ls	[1](3) vi main.c [2](4) vi sub.c [2](5) gcc -o test main.c sub.c [3](6) test
コマンド列3	コマンド列4
[1](8) vi repo.tex [2](9) jlatex repo.tex	[1](10) jdvi2kps repo.dvi > repo.ps [2](11) lpr repo.ps
コマンド列5	
[1](13) vi ex01.c [2](14) gcc -o exo1 ex01.c	[3](15) ex01

図 7: 図 6 から得られるコマンド列

以降，予測候補の更新は，前回の予測が失敗した場合と PGS内の候補で予測が正答した場合に行なう．

得られた候補の順序付けには前回の予測がどの候補によって成功したかによって異なった戦略をとる．

前回の予測が成功した場合，前回の入力コマンド v_{-1}は PGSか PGTのいずれかに含まれている．予測に使用される候補を，PGS，PGTに含まれるタスクで v_{-1} を含むタスク (CT)，PGTに含まれるタスクで v_{-1}を含まないタスク (OT)，の 3 種類に分類し，これらの候補の期待度を定める．

$v \in PGS$の場合：ユーザは前回のコマンドで作業間遷移を終了し，以降のコマンドで特定の作業を行なうと考える．このため，ユーザ

の行なうタスクの特定が必要となる．この時，$v_{-1} \in GT_k \in PGT$なるGT_kは存在しないので，候補の期待度は，$PGT > PGS$となる．

$v \in PGT$の場合：ユーザは現在特定の作業を行なっている最中であると考える．このため，CTの作業を続行する．候補の期待度は，CTが一番大きく，ユーザが特定のディレクトリでの作業を集中して行なうと仮定すれば，$CT > OT > PGS$となる．

ユーザが次のコマンドで別の作業を行なうことが予測される場合，可能性のある全てのタスクを提示する必要がある．しかし，通常作業を行なう場合，作業の途中から行なう場合は少ないため，タスクの最初のコマンドで正答となる場合が多い．この様な場合はタスクの最初のコマンドだけを予測の上位に置き，他を予測の下位に置く事にする．また，PGS内のノードとPGT内のタスクの順序はノードやタスクが生成された順序を使用し期待度の順にコマンドを提示する．

図6の作業情報が作成されている状況で，ノード1に相当するコマンドが入力された場合，$PGS = \{7\}$, $PGT = \{GT1, GT2, GT3, GT5\}$となり，候補の順序は，[2, 3, 13, 7, 4, 5, 6, 14, 15]となる．この予測において，3で予測が正答した場合，$CT = \{GT2\}$, $OT = \{GT1, GT5\}$となり，候補の順序は,[3, 4, 5, 6, 2, 13, 7, 14, 15]となる．

4 実験

本節では，コマンド予測における本手法，FRQ, LRU法[4]の正当率を比較する．実験には人工的に生成したコマンド列と研究室で収集したコマンド履歴を使用した．コマンド列の生成は以下の手順で行った．まず，同一のディレクトリで実行され，1から5コマンドで構成されるコマンド列をl個用意し，これらのコマンド列の実行ディレクトリを変化(m種類)させる事で基本となるl × m個のコマンド列を用意した．これらのコマンド列から一様乱数に従って独立にコマンド列を順に選び，この操作をコマンドの総数がn個になるまで繰り返した．

図8はl=5, m=10, n=300で生成したコマンド列を使用した予測実験を本手法，0〜3次のFRQ, LRU法を用いて10回繰り返した場合の平均の正当率の変化を示している．各手法において候補の最大数は20とし，候補内に入力コマンドが含まれる場合に正答とした．LRU, FRQ法の中で正答率の良かった0次，1次の場合でも300コマンド入力時に10%程度の正答率であり，50コマンド以下では殆ど正答していないに対して，本手法では少ないコマンド数で急激に正答率が上昇し，300コマンドでは3割以上の正答率を記録している．FRQ,LRU法の正答率が低いのは，平均の候補の数が本手法では9.8コマンドであるのに対して，FRQ,LRU法では1番平均候補数の多い0次の場合でも1.3コマンドしかないためである．

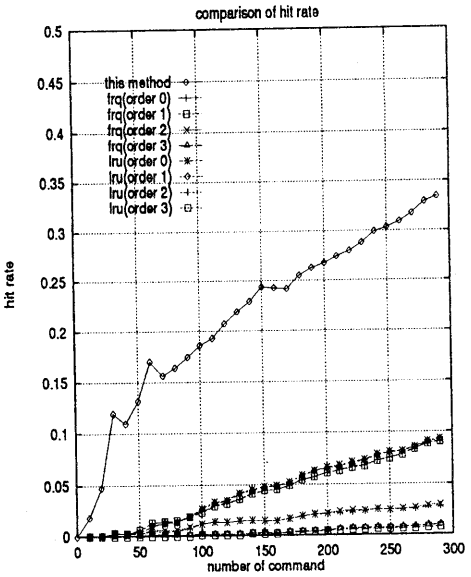


図8: FRQ, LRU法との予測正答率の比較 (生成コマンド列, l=5, m=10, n=300)

図9は研究室で収集したコマンド履歴を使用し，本手法，FRQ, LRU法を用いた場合の正当率を示している．履歴の収集はUNIXシステムの代表的なシェルであるtcshを改造し，入力されたコマンドのトークンをキーとしてファイルシステムの検査をコマンドの実行前後に行う事で行った．このため各ユーザの通常の計算機の使用状況が履歴に保存されている．実際のユーザのコマンド履歴を用いた場合でも人工的に生成した入力を用いた場合と同様に，FRQ, LRUの各方法と比較して高い正答率を記録している．また，このユーザは

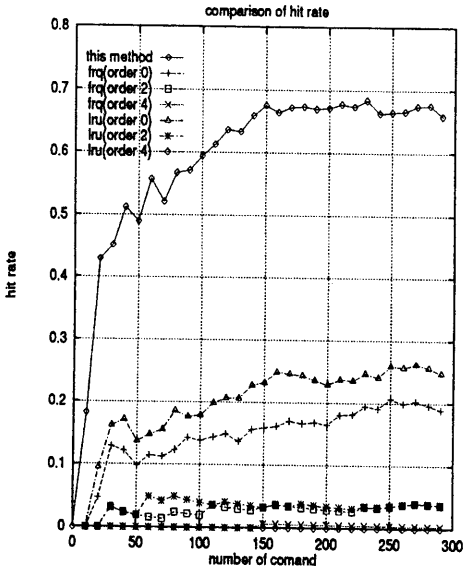


図 9: FRQ, LRU 法との予測正答率の比較 (コマンド履歴)

比較的定型の作業を繰り返し行なう傾向が強かったため数十コマンドの入力で 5 割程度の正答率に急激に到達している.

5 おわりに

本稿では, コマンド間の依存関係と, コマンドが実行されたシステムの状態に注目して, コマンド予測を行なう方法を提案した. 本手法は作業情報の生成と予測の 2 つの部分に分けられ, 予測はグラフ内のノードと関係の検索で終了するため, 予測の計算コストは小さい. また, ユーザの入力コマンド系列を使用するため, ユーザに適応した支援が行なえ, 予測正答率も FRQ, LRU 法に比べて高い正答率を記録している. この高い正答率が, 3 での 2 つの仮定を裏付けているものと考える. 今後の課題として, 予測候補をより少なくするための順位付け戦略の改良が挙げられる.

参考文献

[1] W. B. Croft, The role of context and adaptation in usr interface, Int. j. Man-Machine Studies, Vol.2, pp283-292, Academic Press (1984)

[2] Edwin Bos, Some Virtues and Limitation of Action Inferring Interfaces, In Proceedings of UIST-92, pp.79-88 (1992)

[3] Arad A. Myers, Demonstrational Interfaces: A Step Beyond Direct Manipulation, IEEE Trans. COMPUTER (1992)

[4] S. Greenberg, I. H. Witten, Supporting command reuse: mechanisms for reuse, Int. J. Man-Machine Studies, 39, pp.391-425 (1993)

[5] 瀬下, 加藤, 平川, 市川, ユーザ操作からのプログラム・コンポーネントの抽出, WISS93, pp.249-256 (1993)

[6] 中山, 宮本, 川合 , コマンド履歴からの動的スクリプト生成, WISS'94, pp.155-164, (1994)

[7] 久保, 守田, 田中, コマンド連鎖グラフを用いたコマンド列の予測, 中国支部連大, (1993)

[8] 久保, 守田, 田中, 依存関係に基づく作業に応じたコマンド列の抽出, 中国支部連大, (1994)