平成 27 年度 修士論文

ユーザのブラウジング習慣に基づく携帯端末での Web ページプリフェッチ技法

電気通信大学 大学院情報システム学研究科
情報システム基盤学専攻
1453019　李 明元

主任指導教員　多田 好克　教授
指導教員　本多 弘樹　教授
指導教員　小宮 常康　准教授

提出日　平成 28 年 1 月 28 日

目次

図目次

ソースコード目次

表目次

第 1 章

はじめに

近年携帯端末の普及に伴って、PC の代わりに携帯端末用 Web ブラウザを使用するユーザが増えてきた。そこで、携帯端末用 Web ブラウザのパフォーマンスに対する要求が高くなってきた。

Web プリフェッチは既存でよく Web ブラウザのパフォーマンス向上のために使用されている方法の 1 つである。Web プリフェッチはユーザの Web 履歴で次にユーザがアクセスする可能性のある Web リソースを事前に端末にロードする。次にユーザが使用する時、サーバとのやり取りなしで、速くユーザに画面に表示し、遅延を削減する。

既存の PC での Web ブラウザの Web プリフェッチ方法は沢山あるが、携帯端末バッテリ、使用可能な通信データ量、メモリ容量などの制限で流用できないものが多い。その理由としては、速度を出すために、携帯端末ではバッテリの使用が減り、通信データ量が大幅に増えるためである。既存の PC 上のプリフェッチ方法は適用できない。そこで、既存の手法を携帯端末で使用可能に改良する必要がある。

本研究では携帯端末でのブラウジング習慣に基づき、ユーザがよくアクセスする Web ページでマルコフ連鎖を作成し、さらに、条件によりキーワードをベースにしたプリフェッチ方法を合わせ、携帯端末のバッテリ、通信データの節約でき、かつ、ブラウジングパフォーマンスを向上する Web ページプリフェッチ技法を提案し、実験で評価する。

第 2 章

序論

2.1 背景

近年携帯端末の普及に伴って PC の代わりに携帯端末のみで Web ブラウジングを行うユーザが増加してきた。そこで、携帯端末の Web ブラウザのパフォーマンスに対する要求も高くなった。Web プリフェッチは Web ブラウザのパフォーマンス向上によく使用される方法の 1 つである。Web プリフェッチはユーザの Web 履歴を利用し、ユーザが今後使用する可能性がある Web リソースを事前に端末にロードする。ユーザがそのデータを使用する時、サーバとのやり取りをせずに、ユーザに提供でき、遅延時間の削減で、Web ブラウザのパフォーマンスを向上する。ユーザが今後使用する可能性がある Web リソースを予想するには、沢山のデータを使用し、そのデータを利用した数学モデルの作り、または予想根拠になるデータをマイニングし、パターンによるリソースプリフェッチを行う。

しかし、既存の PC 上のブラウザでの Web プリフェッチ方法は、携帯端末に適用できない場合が多い。携帯端末はバッテリの使用量とデータの通信量が限られている。既存の PC 上のブラウザでの Web プリフェッチ方法は計算量、バッテリの使用時間が多く、ユーザの使用できるデータ通信量を浪費する。そこで、携帯端末の特徴とユーザの使用習慣に合わせたプリフェッチ方法が必要となる。

2.2 目的

本研究ではユーザのブラウジング習慣に基づいて、携帯端末に合わせた Web ページのプリフェッチ技法を提案する。ユーザの Web 履歴を利用し、よくアクセスされる Web ページをマルコフ連鎖で連結する。その遷移確率に PageRank という概念を導入し、プリフェッチすべきページを算出する。また、ユーザの履歴を利用し、ユーザの興味のあるキーワードを取得する。条件を満たした場合、キーワードベースにしたプリフェッチを行う。本技法では、以上の 2 つ方法を合わせ、携帯端末のバッテリ使用量、通信データ量の節約を目標している。また、提案手法について実験で評価を行う。

2.3 携帯端末用 Web ブラウザの利用特徴

多くのユーザは本人の好みの Web サイトと好みのカテゴリがあり、習慣により Web サイトのカテゴリにアクセスする。習慣というのは、ユーザの Web ページのアクセス順番、よくアクセスする Web サイト、Web ページのカテゴリおよび関連キーワードを示している。例として、図 2.1はサッカーと政治に興味があるユーザブラウジング例になる。ユーザがニュースサイトのホームページ A にアクセスした時、緑の矢印を辿り順番でスポーツページ B、サッカーページ D をアクセスし、サッカーページで自分の好きな選手、チームに関するニュース E と F などを複数回サッカーページに戻り関連ニュースを読む。また、読み終わったら、戻りボタンを押し赤い矢印を辿りサッカーページ D、スポーツページ B、ホームページ A に戻る。そこで、政治のページ C に入り、自分が興味持っているニュースを読む。以上のパターンが現在多くのユーザに発生している。この情報を利用するためには、Web ページの特徴も理解する必要がある。

図 2.1 ブラウジング習慣例 出典: http://www.asahi.com

2.4 ユーザのブラウジング習慣と Web ページの特徴

図 2.2に同じドメイン名の Web ページの構成 [7] を示す。丸は Web ページを示し、矢印はリンクと所属関係を示す。矢印元の Web ページは矢印先の Web ページのリンクがあり、矢印先の Web ページは矢印もとの Web ページに属しているのを意味している。例として、ページ A にページ B のリンクがあり、ページ B はページ A に属していることを示している。図 2.1のページ A から F は図 2.2の A から F に対応している。

ページ A からページ D まで同じ順番でアクセスし、異なる日にまたは時間帯に複数回アクセスする可能性が非常に高い。しかし、ページ E とページ F は一回しかアクセスしないし、別日には他のページに替えられている可能性が高い。

既存研究 [2] ではページ A からページ D まで各ページをノードとし、マルコフ連鎖を作成し、ユーザが再度ページ A にアクセスした時、次にアクセスする可能性があるページ B とページ C をプリフェッチする。そこで、ユーザがページ B またはページ C にアクセスする時の待ち時間を減らす。他には、ページ E とページ F がこの Web ページにアクセスした主な理由などで、ページ E とページ F も予想する研究 [1] ではユーザがページ A からページ F のユーザが押したリンクと読んだドキュメントを解析し、図 2.1のページ E とページ F に青の四角に囲まれている “AC ミラン”、“本田”、“マインツなどのキーワードを取り出す。そして、そのキーワードを使用しユーザがページ D にアクセスした時、ページ E とページ F をプリフェッチし、待ち時間を減らす。

しかし、以上の 2 種類の研究が携帯端末に使用できない理由としては計算に使用するデータ量が多いため、計算するにはバッテリの消費量が多い。そこで、既存手法を携帯端末で負担が重すぎずに実行する必要がある。また、本論文で以上の操作は特定のユーザに限らず、多くのユーザの共通のものだと考えている。

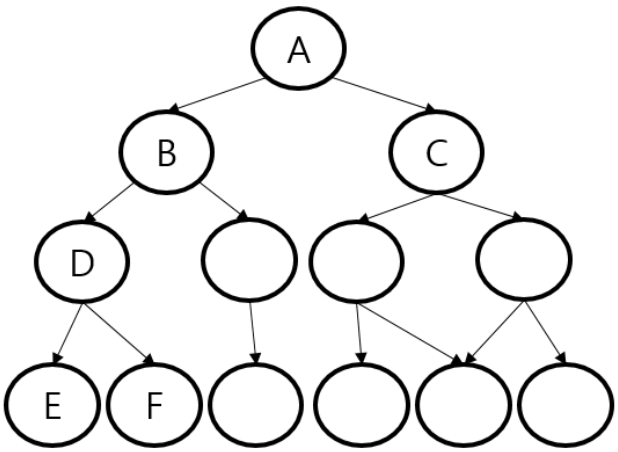

図 2.2　ドメイン名が同じ Web ページの構成

第 3 章

関連研究

3.1 マルコフ連鎖を利用した Web プリフェッチに関する研究

“未来にどこに行くかは現在どこにいるかによる”というのはマルコフ連鎖を利用したプリフェッチ方法の主張点になる。既存研究 [2] では Web ログを利用し各ノードが Web ページを示しているマルコフ連鎖を作成する。各ノードがユーザのアクセス履歴で繋がっている。すべてのユーザがアクセスした適切なページがマルコフ連鎖のノードになる。マルコフ連鎖のノード A とノード B 間の遷移確率はユーザがノード A からノード B までのアクセス回数による。すべてのノードの間が接続可能性があるので、この研究では “ノード数×ノード数”の行列を作成しそれぞれのアクセス回数により遷移確率が記入している。

この研究はすべてのユーザのアクセスしたことがある Web ページの間で、リンクで繋いでいなくてもプリフェッチが可能になっている。しかし、ユーザがアクセスしたことがないページの予想はできないという欠点がある。また、計算量が多いので、携帯端末に適用できない。

3.2 キーワードをベースにした Web プリフェッチに関する研究

キーワードをベースにしたプリフェッチ方法はユーザの好みをキーワードで把握する。ユーザが Web ページをアクセスした時、その Web ページリンクのアンカーテキストにキーワードが存在しているかを確認し、プリフェッチを行う。関連研究では [1] ユーザの Web ページアクセス行動からキーワードを取り出す。またそのキーワードをニューラルネットワークを利用し、それぞれのキーワードのウェイトを計算する。最後にその計算結果により Web ページのプリフェッチを行う。

キーワードをベースにしたプリフェッチ方法はユーザがアクセスしたことのない Web ページもプリフェッチできるという特徴がある。しかし、ユーザの Web ページアクセス行動からキーワードを取り出し、またそのキーワードのウェイトを計算するために、ニューラルネットワークを利用し計算を行うので、計算量は携帯端末では、負担できない

程度になる。

本研究では複数回アクセスする可能性が高い Web ページに対し、マルコフ連鎖を利用したプリフェッチ方法を使用する。また、初めてアクセスする Web ページに対し、キーワードをベースにしたプリフェッチ方法を使用する。また実現するため既存研究方法をある程度変更し、携帯端末に合わせたプリフェッチ技法を提案する。

第 4 章

提案手法

4.1 紹介

提案手法のブラウジング習慣と技術の繋がりを図 4.1で説明する。図 4.1は “海外サッカー” と “プロ野球” が好きなユーザのブラウジング習慣例を示している。

1: ユーザが URL の入力で P1 ページにアクセスする。2: “プロ野球” リンクを押してプロ野球のニュースが乗せてあるページ P3 にアクセスする。3: ユーザが興味持っているニュースがないから緑の四角に囲まれている戻りボタン押し、P1 に戻る。4: “海外サッカー” リンクを押し海外サッカーページ P2 にアクセスする。5: ユーザが興味がある記事を見つけリンク押し、PN1 にアクセスする。6: 読み終わり、戻りボタンを押し P2 に戻る。7: また自分の興味ある記事を見つかり、リンクを押し、PN2 にアクセスする。8: 読み終わったユーザが戻りボタンを押し P2 に戻り、次の記事を探す。

今回の提案手法は以上の操作を 2 つの部分に分け、異なる技術で対応する。黄色の点線で囲まれている部分は操作 1 から 4 までになる。この部分は本研究でマルコフ連鎖を利用して対応する。青の点線で囲まれている部分は操作５から操作８になるが、この部分はキーワードベースのプリフェッチ方法を利用し対応する。茶色の点線で囲まれている部分と青の点線で囲まれている部分でユーザは同じ行動を行ったが、異なる技術で対応している。その理由は、黄色の点線に囲まれている部分のサイトはユーザが再度アクセスする可能性が非常に高い。しかし、青の点線で囲まれている部分の PN1 と PN2 はユーザが再度アクセスする可能性が非常に低いサイトになり、一回しかアクセスしない可能性が高い。そこで、キーワードベースのプリフェッチ方法を利用し、対応する。

4.2 構造

本研究の提案手法の構造を図 4.2に示す。大きく点線の四角で囲まれているブロック、連鎖、キーワードベースプリフェッチの 3 つに分かれている。ブロックと連鎖はマルコフ連鎖を使用し、キーワードペースプリフェッチは、キーワードをベースにしたプリフェッチ方法を使用する。

図 4.1　ブラウジング習慣例 出典:http://m.yahoo.co.jp

4.3　マルコフ連鎖

マルコフ連鎖は確率過程の一種であり、各状態から次の状態に遷移する確率は過去の状態によらず、現在の状態のみによる系列である。本研究ではユーザが次にアクセスする Web ページは過去の履歴より現在アクセスしている Web ページとの関連が強いことを仮定する。そこで、マルコフ連鎖を利用し、ユーザが次にアクセスする Web ページの予想に使用する。既存手法ではマルコフ連鎖の各状態が各 Web ページになり、遷移確率はアクセス回数により計算する。本研究では遷移確率をアクセス回数とマルコフ連鎖の各ページのページランクで計算する。ページランクについて 4.3.4 節で詳しく説明する。

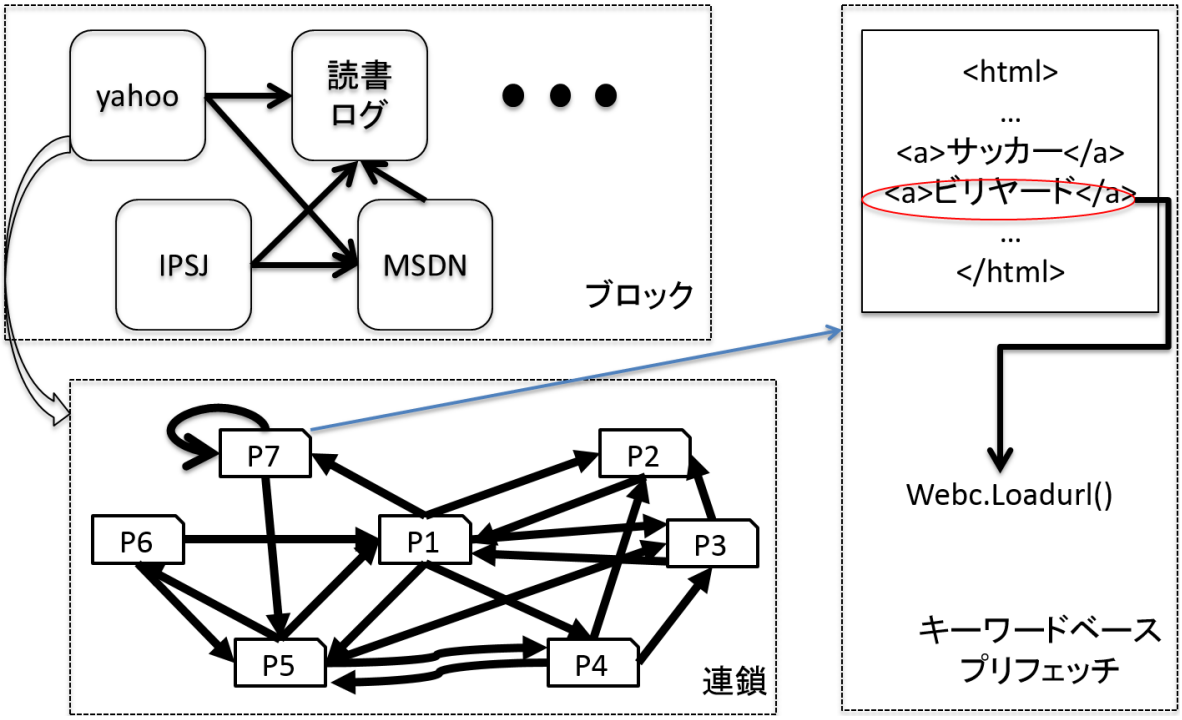

図 4.2　提案手法構造

4.3.1　本研究のマルコフ連鎖の応用

マルコフ連鎖はユーザがマルコフ連鎖に属しているページを再度アクセスするのを前提とする。既存手法のマルコフ連鎖を図 4.3に示す。図 4.3の四角のノード P1、P2 などは、Web ページを示している。矢印は遷移を示す。P1 から P2 を指す矢印は、P1 から次に P2 をアクセスするのを示している。ページにより複数の別のページに遷移して行ったり、また複数のページから遷移してきたりもする。これを決めるのが、ユーザの過去の Web 履歴である。また、各矢印にはそれぞれの遷移確率が付与される。式 4.1に P_i から P_j への遷移確率を示す。

$$P_i\text{から } P_j\text{への遷移確率} = \frac{P_i\text{から } P_j\text{に遷移した回数}}{P_i\text{から } P_i\text{以外のページに遷移した回数}} \tag{4.1}$$

以上は既存のマルコフ連鎖を利用したプリフェッチの方法になる。本研究は携帯端末を使用するため、ユーザのすべての Web 履歴を利用し、マルコフ連鎖を作成すると携帯端末のバッテリに対する負担が大きいと判断した。本研究では、計算量を減らすため、マルコフ連鎖を 2 つのブロック、連鎖に分けた。ブロックは直接プリフェッチの情報はない。ブロックを利用した理由は、マルコフ連鎖をより速く探索でき、計算量を減らすためである。ブロックのノードには主に同じドメイン名の Web ページで作成されたマルコフ連

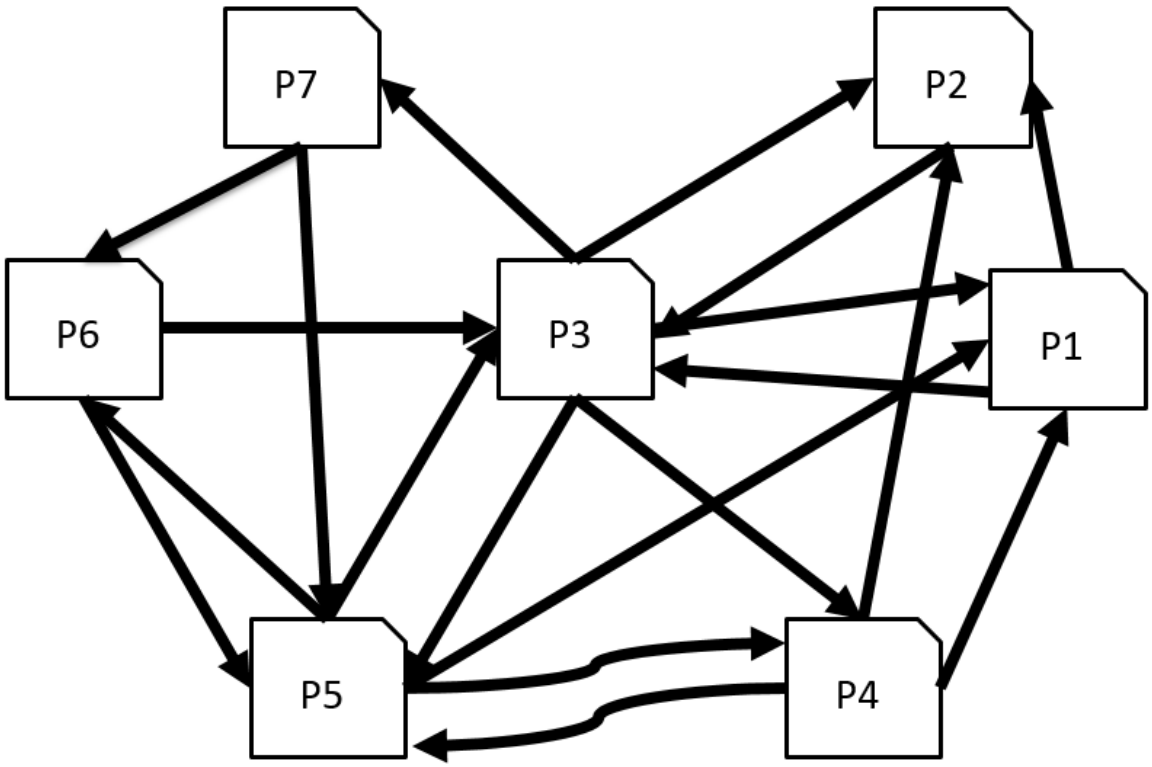

図 4.3　既存手法のマルコフ連鎖例

鎖のアドレスなどの情報が入っている。詳しくは 4.2.2 節で説明する。連鎖のノードには Web ページの情報が入っている。既存手法にのマルコフ連鎖と似ているが、異なる点としては、ドメイン名が異なる Web ページは同じ連鎖には存在しない。連鎖上のノードである条件を満たす場合にキーワードベースのプリフェッチを行う。詳しくは 4.3 節で説明する。

4.3.2　ブロック

既存手法でのマルコフ連鎖の探索時間、計算量を減らすためにドメインベースで作成されたマルコフ連鎖、ブロックを定義した。ブロックの例を図 4.4に示す。各四角はノードになり、矢印は遷移を示している。遷移確率は式 4.2に示す。

$$\text{ブロック } A \text{ からブロック } B \text{ の遷移確率} = \frac{\text{ドメイン名 } A \text{ の } Web \text{ ページからドメイン名 } B \text{ の } Web \text{ ページに遷移した回数}}{\text{ドメイン名 } A \text{ の } Web \text{ ページからドメイン名 } A \text{ 以外の } Web \text{ ページに遷移した回数}} \tag{4.2}$$

しかし、図 4.4の “dokusho-log” ブロックのように、他のブロックに遷移しない場合などが発生する可能性も考えられる。この状況が発生する理由は本ブロックから他のブロック

に遷移した回数が少なく計算していない時と、遷移は行ったがブロックに連鎖がない場合が考えられる。そこで、本研究では全体のブロックにランクをつけ、次アクセスするページがない時ランクが高いブロックの連鎖の Web ページをプリフェッチする構造になっている。ランクの付け方は 4.5 節で説明する。

ユーザが URL を入力した時、その URL のドメイン名を取り、そのドメイン名のブロックを探し、それがある場合にはそのブロックの連鎖を探索する。

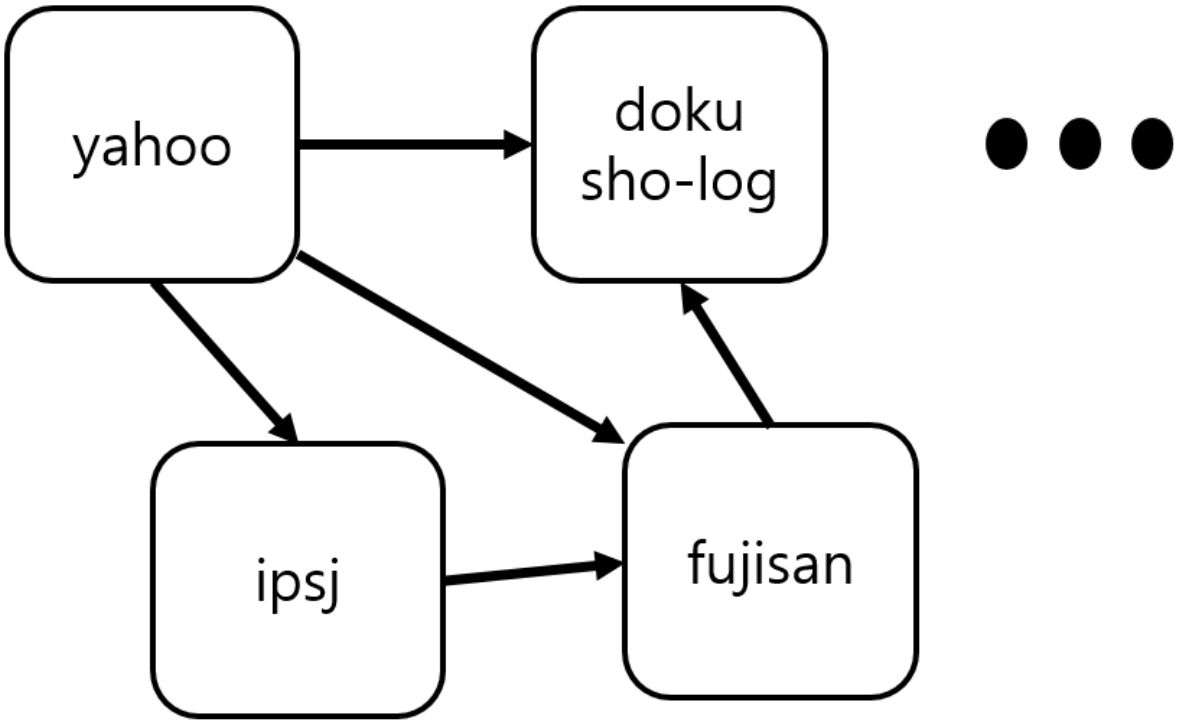

図 4.4　ブロックの例

4.3.3　連鎖

連鎖はドメイン名が同じ Web ページで作成されたマルコフ連鎖である。各ノードは Web ページを示していて、URL、ドメイン名、次にアクセスする Web ページなどの情報が入っている。同じドメイン名の Web ページ間で、2.4 節で挙げた例のパターンがよく発生するので、同じドメイン名の Web ページのみで構成した。

しかし、本論文では既存手法を異なる部分もある。図 4.5に例を示す。図 4.3と他の部分は同じになっているが、異なる部分は P7 には自分を指す矢印が追加されている。

図 4.6の例で説明する。赤い四角に囲まれている部分はユーザが押した部分になる。緑の矢印はユーザがリンクを辿り、アクセスしたことを意味する。赤い矢印はユーザが戻りボトンを押し、アクセスが戻ったことを意味する。青の四角に囲まれている部分は Web

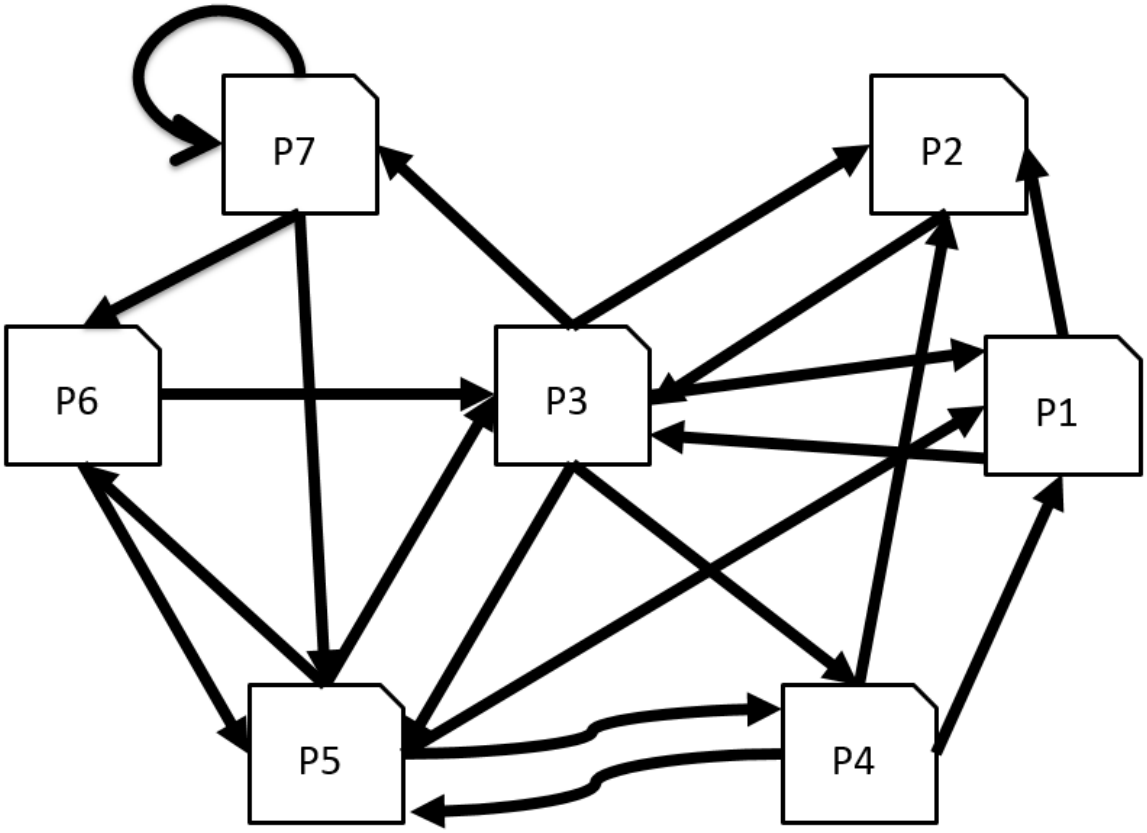

図 4.5　連鎖の例

ページのドメイン名になり、すべての Web ページが同じドメイン名であることを示す。以下は “海外サッカー” が好きなユーザが行った操作になる。PN は指定したページではなく、ユーザの興味があるページを示す。1:ユーザが Yahoo のホームページの P1 の “スポーツナビ” リンクを辿り、P3 にアクセスする。2:“スポーツナビ” ページの P3 から、“海外サッカー” リンクで “海外サッカー” ページ P7 にアクセスする。3:P7 から自分の興味があるページ PN にアクセスする。4:PN ページを読み終わったユーザは P7 に戻り、また自分の興味があるリンクを探す。5:“海外サッカー” が好きなユーザは操作 3 と操作 4 を複数回繰り返す。

図 4.6の P1、P3、P7 は図 4.5の P1、P3、P7 の例になる。P1、P3、P7 はユーザが複数回アクセスするが、PN には再度アクセスする可能性は非常に低い、そこで、既存のマルコフ連鎖を利用して Web プリフェッチを行った研究では PN はマルコフ連鎖に入れない場合が多かった。本研究も、既存研究と同じく PN はマルコフ連鎖の状態としては議論しない。しかし、ユーザが P1、P3、P7 を辿ったのは、手段になり、PN にアクセスしたのが目的になる。そこで、PN を速くユーザに示すのも Web ブラウザのパフォーマンス

図 4.6　ユーザ操作例 1 出典:http://m.yahoo.co.jp

向上とユーザのアクセス環境の改善に重要な部分になる。本研究では既存研究では議論されていない PN をプリフェッチするため、キーワードベースのプリフェッチ方法を使用する。キーワードベースのプリフェッチ方法はページ P7 の解析が必要になるので、マルコフ連鎖を利用したプリフェッチ方法よりバッテリの消費量が多くなると予想し、本研究ではマルコフ連鎖に自分アクセスという概念を導入した。

図 4.3の既存研究 [2] のマルコフ連鎖例と図 4.5の連鎖例の異なる部分の P7 に自分から出てまた自分をアクセスする矢印がある。この矢印は、図 4.5の例の P7 から自分の興味があるページ PN にアクセスし、読み終わったらまた P7 に戻り次の PN にアクセスする操作を複数回繰り返す時、本研究では自分をアクセスしたという意味で導入した。図 4.7の P7 が図 4.5の P7 になり、PN1 と PN2 はユーザの興味がある Web ページを示している。赤い四角に囲まれている部分はユーザが押した部分になり、青の矢印に囲まれている部分は Web ページのドメイン名になり、すべての Web ページが同じドメイン名の Web ページということを示している。緑の矢印はリンクを押し、次の Web ページにアクセスするという意味で、赤い矢印は戻りボタンを押し、前の Web ページに戻るのを示す。図 4.7の例が頻繁に発生する Web ページを本研究では自分にアクセスする Web ページで

図 4.5の自分を指す矢印が付けられる。自分にアクセスする Web ページでは、マルコフ連鎖を利用した Web プリフェッチとキーワードをベースにした Web プリフェッチ両方が発生する。

図 4.7　ユーザ操作例 2 出典:http://m.yahoo.co.jp

4.3.4　ランク付け

ページランク [3] は Web ページの重要度を決定するためのアルゴリズムであり、検索サイトでよく使用されている。ページランクは多くのサイトにリンクされているサイトは必ず重要なサイトだというのを主張している。ページランクは式 4.3で計算される。

$$\text{ページ } n \text{ のページランク} = \sum \frac{\text{ページ } n \text{ にアクセスするページのページランク}}{\text{ページ } n \text{ にアクセスするページからアクセスしたページ数}} \tag{4.3}$$

ページランクの計算方法について例で示す。表 4.1は図 4.5の各ノード間の関連を示し

ている。ユーザがリンクを辿り、アクセスした場合に 1 を入れた。表 4.1で 7 × 7 の正方行列 A を作る。要素は 0 か 1 である。n 行目を横に見ればページ n からアクセスしているページを表していることになる。ページランク式の遷移確率行列 P は、A を転置すると n 行目を横に見ればページ n にアクセスしたページを表しているとこになる。また、それぞれの列を非零要素数で割った、7 × 7 の正方行列になる。分数になっているが分母はページ n からアクセスしたページの数になる。

表 4.1　アクセス表

ページ	P1	P2	P3	P4	P5	P6	P7
P1	0	1	1	0	0	0	0
P2	0	0	1	0	0	0	0
P3	1	1	0	1	1	0	1
P4	1	1	0	1	0	0	0
P5	1	0	1	1	0	1	0
P6	0	0	1	0	1	0	0
P7	0	0	0	0	1	1	1

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

$$B = \begin{pmatrix} 0 & 0 & 1/5 & 1/3 & 1/4 & 0 & 0 \\ 1/2 & 1 & 1/5 & 1/3 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 1/4 & 1/2 & 0 \\ 0 & 0 & 1/5 & 1/3 & 1/4 & 0 & 0 \\ 0 & 0 & 1/5 & 0 & 0 & 1/2 & 1/3 \\ 0 & 0 & 0 & 0 & 1/4 & 0 & 1/3 \\ 0 & 0 & 1/5 & 0 & 0 & 0 & 1/3 \end{pmatrix}$$

ページランクの計算方法は以下になる。

1. すべてのページに同じページランクを与える PR1=PR2=PR3=PR4=PR5=PR6=PR7 = 1

2. ページランクの公式を利用し計算を行う。例:PR1 は行列 B の第 1 行の要素にそれぞれのページランクをかけてすべて足した結果となる。$PR1 = \frac{PR3}{5} + \frac{PR4}{3} + \frac{PR5}{4}$

3. 変わったページランクを利用し 2 を再度行う。

4. ページランクが収束するまで操作 3 を複数回繰り返す。

以上の操作でページランクは収束し、ある値に近づく。先の例ではその結果は以下のようになる。

$$PR = \begin{pmatrix} 0.13 \\ 0.15 \\ 0.28 \\ 0.10 \\ 0.16 \\ 0.08 \\ 0.10 \end{pmatrix}$$

次式は本研究のページ遷移確率の計算式である。

$$P_i\text{から } P_j\text{への遷移確率} \quad = \frac{P_i\text{から } P_j\text{に遷移した回数}}{P_i\text{から } P_i\text{以外のページに遷移した回数}} \quad *P_j\text{のページランク} \tag{4.4}$$

式 4.4 の i と j は同じくなる可能性もある。それは図 4.5の P7 が自分から出た矢印が自分を指している場合になる。ページランクの導入でユーザの習慣のみで遷移確率を決めなく、より客観的にすべての要素間の接続関係で遷移確率をきめた。理由として、今回提案手法で取り扱うデータ量が少ないので、ユーザの行動が 2、3 回変わっても影響を受けるためである。

本研究では、ブロック間の遷移をより精確にするため、ブロックランクを導入した。ブロックランクの計算方法は式 4.5 になる。ブロックは存在するが他のブロックと関連がない場合もあるから、ページランクの計算方法と異なる。

$$\text{ブロック } N \text{ のブロックランク} \quad = \frac{\text{ブロック } N \text{ の } Web \text{ ページにアクセスした回数}}{\text{すべてのブロックの } Web \text{ ページにアクセスした回数}} \tag{4.5}$$

4.4 キーワードベースのプリフェッチ

本研究のマルコフ連鎖を利用したプリフェッチ方法では、ユーザがアクセスしたことがない Web ページの予想には使用できない。そこで、本研究ではユーザがアクセスしたことがない Web ページの予想にはキーワードベースにしたプリフェッチ方法を使用した。既存の研究 [1] ではすべての Web ページでキーワードをベースにしたプリフェッチ方法を利用して、すべての Web ページの HTML ファイルの解析を行った。本研究で計算量を減らすために、キーワードベースのプリフェッチは連鎖の自分をアクセスする Web

ページで行う。そこで、HTML ファイルの解析を頻繁に行わずに、プリフェッチ操作ができる。

4.4.1 キーワードの取得

本研究はすべてのプリフェッチの根拠になるデータの取得を Web 履歴のみで行う。既存研究 [1] では、HTML ファイルのすべてのテキストを利用しキーワードを取りだした。しかし、この操作はユーザがブラウジングしている時行う必要がある。本研究では根拠になるデータをユーザが利用時に取得すると、パフォーマンスに影響を与える一方、携帯端末のメモリ空間も大幅に使用することになる。そこで、本研究では、Web ページのタイトルからキーワードを取得している。

しかし、すべての Web ページのタイトルからキーワード取ると必要のない情報が多い。また、キーワードベースのプリフェッチ方法は HTML ファイルのアンカーテキストにキーワード含まれているかをそれぞれ確認するので、キーワードの数も重要である。そこで、本研究ではキーワードをドメイン名ベースに分けた。理由は 2 個があって、まず、多くのキーワードは同じドメイン名に属している Web ページのみから取得できる可能性が高いと判断したためである。また、ドメイン名ベースに分けると、HTML のテキストを解析する時の手間を減らせると判断した。図 4.7の PN1 と PN2 ユーザは興味がある Web ページで一回しかアクセスしない可能性が非常に高い。そこで、本研究の連鎖には入る可能性がほぼない。しかし、PN1 と PN2 のタイトルにはユーザが興味を持っているキーワードが含まれている可能性が非常に高い。本研究では PN1、PN2 と類似なページのタイトルからキーワードを取り出す。取り出し方は 5 章で詳しく説明する。

本研究は上記の例と異なり、ドメイン名が異なる Web ページでキーワードの共有する可能性も考え、HTML ファイルでドメイン名が同じキーワードの調査をした後、見つからなかった場合に、他のキーワードの出現頻度が最も高いキーワードから再度調べる。

4.4.2 キーワードリスト

取得したキーワードをキーワードリストで管理する。連鎖のノードにページランクが付いているのと同じく、キーワードの優先順位を決めるために各キーワードのウェイトが必要になる。本研究のウェイトの計算方法は式 4.6 になる。

$$\text{ウェイト} = \frac{\text{ドメイン } A \text{ でキーワード } K \text{ の出現頻度}}{\text{ドメイン } A \text{ のすべてのキーワードの出現頻度}} \tag{4.6}$$

第 5 章

提案手法の実装

5.1 実装環境

本研究の実装環境を表 5.1 に示す。

表 5.1 実装環境

端末	Nexus 6
OS	Android 5.1.1
ブラウザ	Android Webview

Webview[6] は Android の開発における GUI の一種である。Webkit ベースのブラウザ機能を Webview というビュークラスからアクセスできる。本研究では Webview をブラウザとして使用し、提案手法の実装を行った。また、ブロック、連鎖、キーワードリストの作成には Python3 を利用した。

5.2 動作の流れ

図 5.1は JIS X0121[8] を参考してフローチャートであり、提案手法の流れを示す。1. Webview を立ち上げ、最初ブロックファイルを読み込む。2. ユーザからの URL の入力を受け取る。3. URL が入力されたら、URL のロードと URL に関する情報を検索を同時に行う。4. 情報がある場合にプリフェッチ先の確認を行う。5. マルコフ連鎖でのプリフェッチまたはキーワードベースのプリフェッチを行う。6. 続いてプリフェッチ先の情報がある場合に操作 5 に戻り、ない場合に操作 2 に戻る。

URL に関する情報は連鎖ファイルになり、5.3節で詳しく説明する。ブロックファイルの作成については 5.4節で詳しく説明する。URL の検索及びプリフェッチに関して 5.6節で説明する。

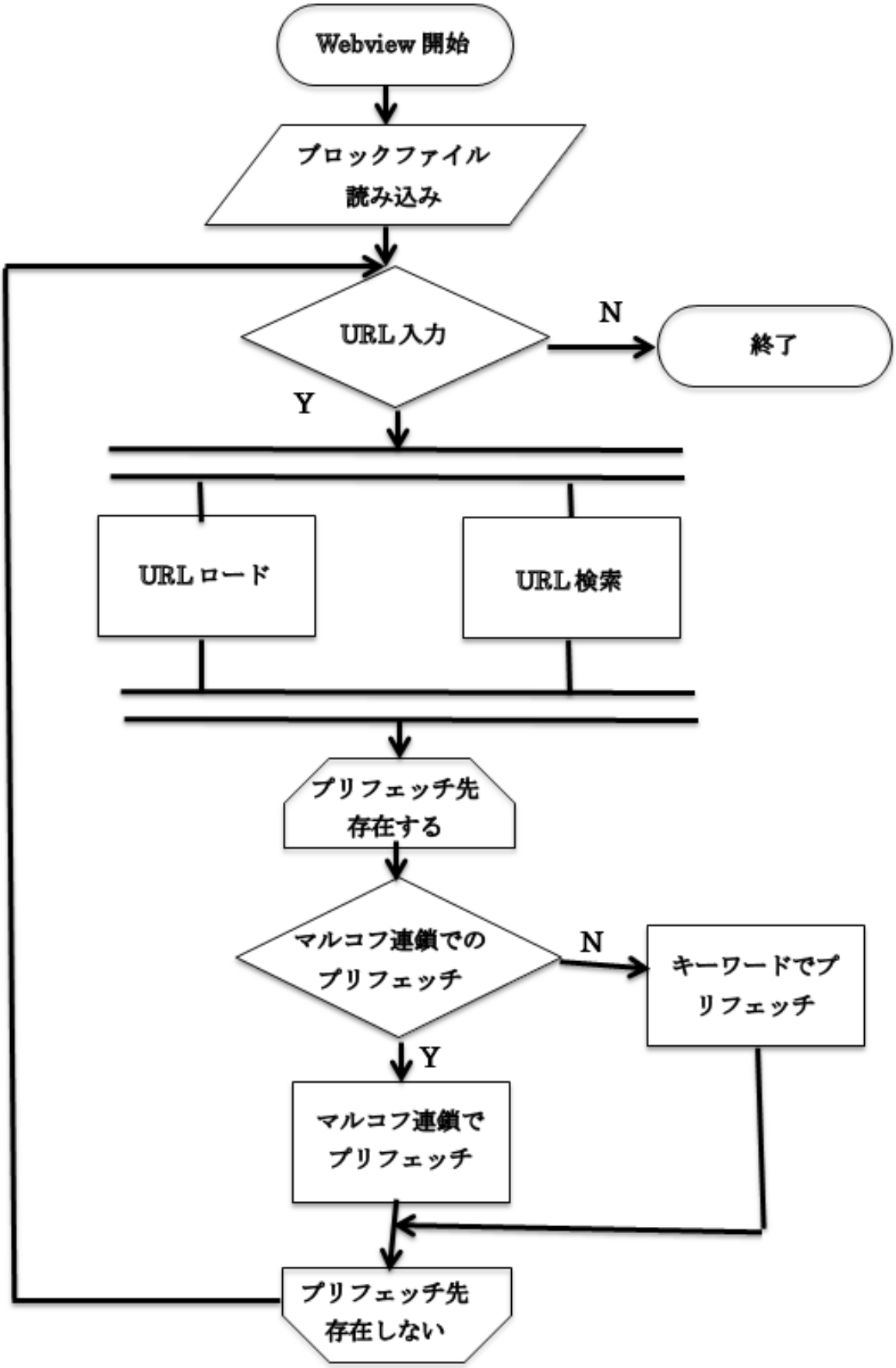
Webview 開始
ブロックファイル
読み込み
URL 入力
N
終了
Y
URL ロード
URL 検索
プリフェッチ先
存在する
マルコフ連鎖での
プリフェッチ
N
キーワードでプ
リフェッチ
Y
マルコフ連鎖で
プリフェッチ
プリフェッチ先
存在しない

図 5.1　提案手法の流れ

5.3 連鎖の作成

計算量を減らすために、本研究では最近 30 日の履歴を利用し、連鎖を作成した。取り扱ったデータはユーザが 30 日内の異なる日に 4 回以上アクセスした Web ページになる。4 回に決めた理由は、ユーザが週 1 回しかアクセスしないが、毎週必ずアクセスする Web ページまでは連鎖のノードとするためである。連鎖の情報を保存するには CSV ファイルを利用している。連鎖のノードのデータ構造を表 5.2に示す。ドメイン名はノードのドメイン名で、URL はノードの URL である。Web ページ 1、2、3 は本ノードにアクセスした時、プリフェッチする Web ページの URL になる。順番で Web ページ 1 の優先順位が最も高い。連鎖ノードのデータ構造にページランクがない理由はユーザがブラウジング行う時、数値計算で、各ノードの比較行うとパフォーマンスに影響するためである。事前に連鎖を作成する時、比較行い、優先順位が付いている次にアクセスする Web ページ 1、2、3 に分けた。また、同じ連鎖に属している Web ページは上からページランクの高い順に並べた。

表 5.2　連鎖ノードのデータ構造

ドメイン名	URL	Web ページ 1	Web ページ 2	Web ページ 3

連鎖の作成に Python3 とい言語を使用した。操作の手順は以下のようになる

1: 最近 30 日以内にアクセスした Web ページの中から以下の 2 パターン探す。1 つはページ A とページ B が両方 30 日間の異なる日に 4 回以上アクセスした Web ページの時ページ A からページ B にアクセスする回数 (表 5.3)。もう 1 つはページ C が 30 日間の異なる日に 4 回以上アクセスしたページで、ページ D は 1 回もしくは 2 回アクセスした Web ページになっている時、ページ C からページ D にアクセスし、またページ D に戻ったパターンを探す (表 5.4)。結果をファイルに保存する。

2: 1 の結果を利用し、同じドメイン名の Web ページを同じ組にし、マルコフ連鎖を作成する。マルコフ連鎖の各ノードのページランクを計算する。計算結果を別ファイルに保存する。本研究では Python3 の数値計算ライブラリの Numpy[5] を利用した。そのソースコードを 付録 Aのソース A.1に示す。

3: 1 と 2 の結果を利用し、本研究で使用する連鎖情報があるファイルを作成する。各行は 5.2に示している情報が入っている。作成のソースコードは 付録 Aのソース A.2に示す。

作成できた連鎖のサンプルを図 5.2に示す。各行は表 5.2に示している構造と同じくなっている。ドメイン名と URL しかない行はそのページから他のドメインにアクセスす

表 5.3　パターン 1

アクセス時間	タイトル	アクセス回数	URL
1 月 2 日 10 時 20 分 30 秒	ページ A	10 回	http://www.example_page.com
1 月 2 日 10 時 21 分 10 秒	ページ B	15 回	http://sports.example_page.com

表 5.4　パターン 2

アクセス時間	タイトル	アクセス回数	URL
1 月 3 日 22 時 33 分 44 秒	ページ C	10 回	http://news.example_page.com
1 月 3 日 22 時 34 分 13 秒	ページ D	1 回	http://news.example_page.com/badnews.html
1 月 3 日 22 時 36 分 59 秒	ページ C	10 回	http://news.example_page.com

る確率が高い場合と、リンク押した時に新しいタブで開く Web ページになっている場合になる。そこで、その場合に他のドメインの Web ページのプリフェッチとキーワードをベースにしたプリフェッチが行う。

```
endforthischain
youku,http://i.youku.com/u/UMzE2OTY2NjUy?from=y1.3-game-dota-136-10557.93254-93252.2-2, , , ,
youku,http://game.youku.com/index/dota,http://i.youku.com/u/UNDI1NTMxMjMy?from=y1.3-game-dota-136-1\
0557.93254-93252.1-2,keyword,http://i.youku.com/u/UMzE2OTY2NjUy?from=y1.3-game-dota-136-10557.93254\
-93252.2-2,
youku,http://i.youku.com/u/UNDI1NTMxMjMy?from=y1.3-game-dota-136-10557.93254-93252.1-2,http://i.you\
ku.com/u/UMzE2OTY2NjUy?from=y1.3-game-dota-136-10557.93254-93252.2-2, , ,
youku,http://www.youku.com/,http://game.youku.com/,keyword, ,
youku,http://game.youku.com/,http://game.youku.com/index/dota, , ,
endforthischain
baykoreans,http://baykoreans.com/,http://baykoreans.com/entertain,http://baykoreans.com/drama,http:\
//baykoreans.com/movie,
baykoreans,http://baykoreans.com/movie,keyword,http://koreayh.com/,http://baykoreans.com/index.php?\
mid=movie&page=2,
baykoreans,http://baykoreans.com/drama,http://baykoreans.com/movie,keyword, ,
baykoreans,http://baykoreans.com/index.php?mid=movie&page=3,keyword, , ,
baykoreans,http://baykoreans.com/index.php?mid=movie&page=2,http://baykoreans.com/index.php?mid=mov\
ie&page=3, , ,
baykoreans,http://baykoreans.com/entertain,http://baykoreans.com/movie, , ,
endforthischain
```

図 5.2　連鎖サンプル

5.4 ブロックの作成

ブロックのデータ構造を表 5.5に示す。ドメイン名はブロックに属している Web ページのドメイン名になる。開始行はドメイン名が同じ Web ページが連鎖ファイルでの開始位置を示している。終了行はドメイン名が同じ Web ページのが連鎖ファイルでの終了位置を示している。

表 5.5 ブロックノードのデータ構造

ドメイン名	開始行	終了行

5.5 キーワードリストの作成

キーワードリストの各行の構成を表 5.6に示す。計算量を減らせるためにキーワードリストの構成要素にドメイン名を付けた。ウェイトがより高いキーワードがあっても、そのドメインで取得したキーワード優先で調べる機能を実現するためである。理由としては、毎日サッカーのニュースを読んでいる人が、週一回見るアニメサイトにもサッカーのニュースページで取得したキーワードが出現する可能性が低いためである。

キーワードは表 5.4のページ D のタイトルから取り出す。例として、ユーザが Web ページでアクセスしたことがない Web ページにアクセスする理由はそのリンクのアンカーテキストにニューザの興味を起こすキーワードが含まれている可能性が高いからである。

キーワードの取得方法について、実装では実験対象が日本語の Web ページなっているため、MeCab を利用し名詞を取った。ドメインベースで出現頻度が上位 10 個づつリストに保存した。

表 5.6 キーワードリストのデータ構造

ドメイン名	キーワード	ウェイト

5.6 プリフェッチ

提案手法で上で作成したファイルを利用して図 5.3に示したように Web ページのプリフェッチを行う。青の線は提案手法により行う処理になり、オレンジ色の丸はユーザが

押した部分になる。提案手法が実装される前の Webview は URL が入力されたら緑の矢印を辿り、キャッシュにその URL のリソースが存在するかを確認する。存在する場合にキャッシュからそのまま提供する。これは最も早くユーザに Web ページの情報を提供できる “道” である。存在していない場合に、赤い矢印を辿り、サーバにデータを請求する。サーバはデータを端末に送り、ブラウザがレンダリングを行いユーザに提供する。これは一般的の Web ページのリソースのやり取りの仕方であるが、最も遅い “道” である。

本研究での流れを図 5.3の例で説明する。

1: ユーザが最初 “www.example_pageA.com” というページを入力したら、緑の矢印または赤の矢印を辿り URL のリソースをロードする。

2: 同時に青の矢印を辿り “block.csv” に “example_pageA” というドメイン名が存在するかを確認する。

3: 存在する場合にブロックファイルに記入している行番号 “10” を取得し、“chain.csv” ファイルの “10” 行目から読み込む。

4: “chain.csv” ファイルに “www.example_pageA.com” が存在する場合にその行の次の赤い点線に囲まれている、Web ページ URL を “loadurl()” という関数でメインロードが終了した時点でサーバに請求し、サーバから送られたデータをキャッシュに保存する。この URL は次にアクセスする可能性が最も高い Web ページなっている。

5: 操作 4 の “www.example_pageA.com” の次が黒い点線で囲まれている “keyword” という文字になっている時は、“keyword list” を参照し、ドメイン名が “example_pageA” になっているキーワード “MS” を読み込む。そこで、“www.example_pageA.com” にキーワード “MS” を含んでいるアンカーテキストを探索し、そのリンクの URL を “loadurl()” 関数でサーバに請求する。またそのデータをキャッシュに保存する。キーワードベースのプリフェッチはメインロードが終了した時点に開始する。

6: ユーザがリンクを辿り、または URL 入力で新しいページにアクセスしたら、操作 2 から 5 まで再度行う。

今回の提案手法では、ページのプリフェッチはユーザが入力した URL のロードと同時に開始する。しかし、URL のロードはメインロードが終了した時点で行う。毎回最大 3 つの Web ページをプリフェチする。キーワードベースのプリフェッチを行う時には、URL の HTML の取得が必要になり、ユーザが入力した URL のロード終了時に行う。携帯端末の他のアプリケーションに影響与えないため、提案手法では最大 15 個の Web ページをキャッシュに保存する。Web ページの平均サイズ [4] が 2MB になっている調査結果から、提案手法のキャッシュサイズは 30MB に決めた。プリフェッチされたデータは FIFO を利用してデータの置き換え wo 行う。キーワードリストのキーワードを各ドメインことでマクス 10 個まで保存できる。また、すでにプリフェッチしており、キャッシュにまだ存在しているページを再度プリフェッチしないため “preftched_url” というリスト

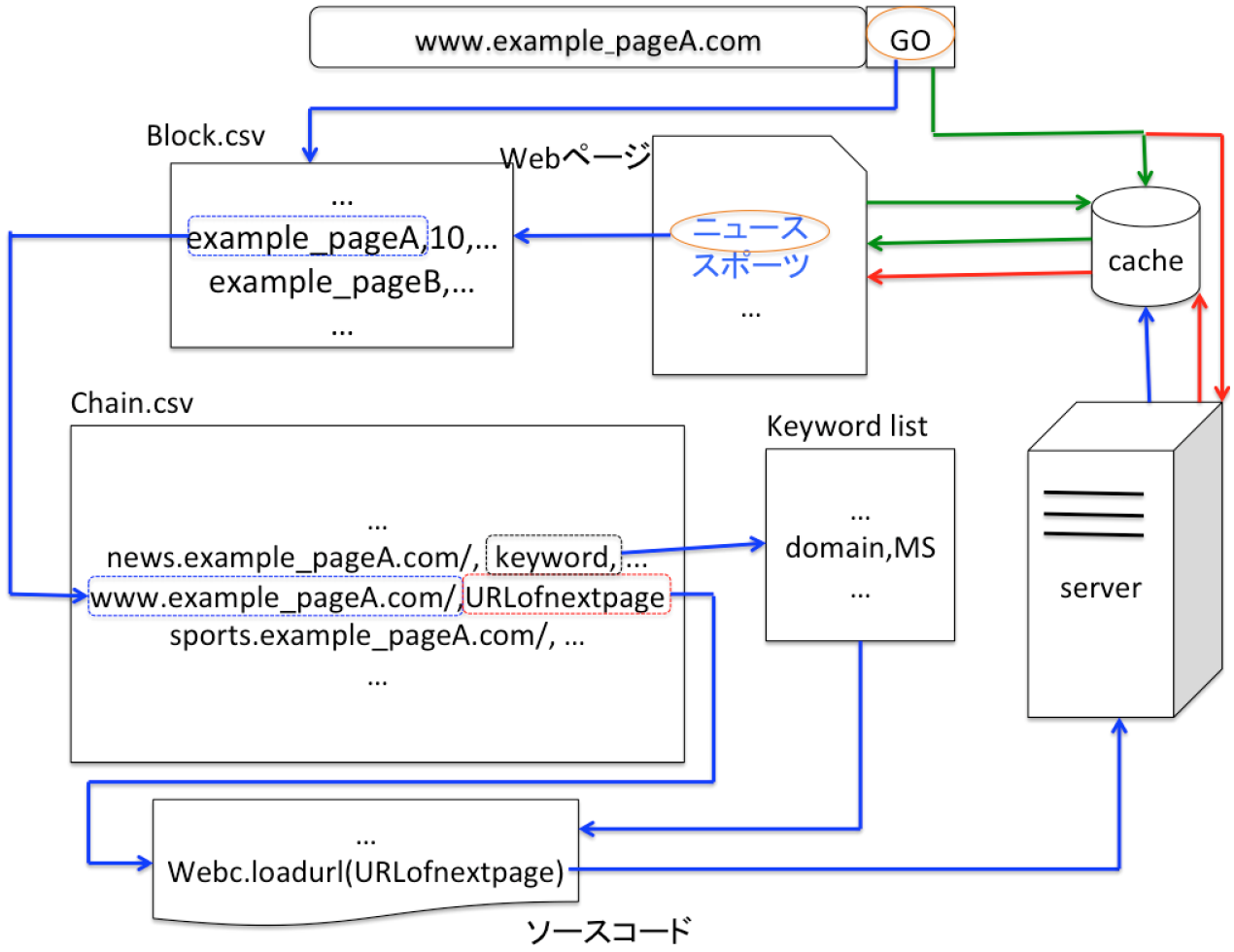

図 5.3　提案手法実行例

を使用し管理を行う。

5.7　修正

ユーザの習慣が変化し、新しい興味が増えたりするので、連鎖、ブロック、キーワードリストを週 1 回修正する。手法は各ファイルの作成手法と同じで、データは最近 1 ヶ月の Web 履歴を利用する。修正にはプリフェッチよりバッテリ消費が多いため、充電中でユーザが利用していない時に行う。本研究はヒットによるキーワードリストの変化、連鎖及びブロックの変化は行わない。

第 6 章

評価実験

6.1 実験環境

実験のデータは情報系大学院生 1 名の Web 履歴を利用し、2 回の実験を行った。

実験 1 は 2015 年 10 月 1 日から 2015 年 10 月 30 日の Web 履歴を利用し、作成されたブロック、連鎖とキーワードリストを利用して実験を行った。実験 1 では作成されたブロック、連鎖、キーワードリストを利用し、10 月 31 日から 11 月 6 日までのアクセス履歴で、1 日のアクセス時刻が早い上位 50 個づつ選択し、アクセスを行った。

実験 2 は 2015 年 10 月 8 日から 2015 年 11 月 6 日までの Web 履歴を利用して実験を行った。実験 1 と同じく、作成されたブロック、連鎖、キーワードリストを利用し、11 月 7 日から 11 月 13 日までのアクセス履歴で、1 日のアクセス時刻が早い上位 50 個づつ選択し、アクセスを行った。

6.2 ヒット率と有用率

ヒット率とは、ユーザがアクセスした Web ページの中、プリフェッチでロードされた Web ページの割合いを言う。有用率はプリフェッチした Web ページの中、実際使用された Web ページの割合を言う。

実験 1 と実験 2 のヒット率の結果を図 6.1に示す。縦軸は割合を示している。横軸は日を示しており、実験を始めた最初日を 1 とする。結果から見ると、ヒット率日により高くなったり、低くなったりする。日により、パフォーマンスの向上が異なる。実験 1 と実験 2 のヒット率は最多値で、それぞれ 40% と 44% になる。また、最小値は、8% と 12% になる。実験 1 と実験 2 の結果を合わせて平均的にヒット率は 25% になる。

実験 1 と実験 2 の有用率の計算結果を図 6.2に示す。縦軸は割合を示しており、横軸は日を示している。結果を見ると、有用率は 60% から 70% の間で比較的に安定している。高くてそれぞれ 71% と 72% になり、低くて 50% と 61% になっている。実験 1 と実験 2 の結果を合わせ、平均の有用率は 65% になる。

ヒット率と有用率の結果から、日によりヒット率が変動しているがその結果は本研究で

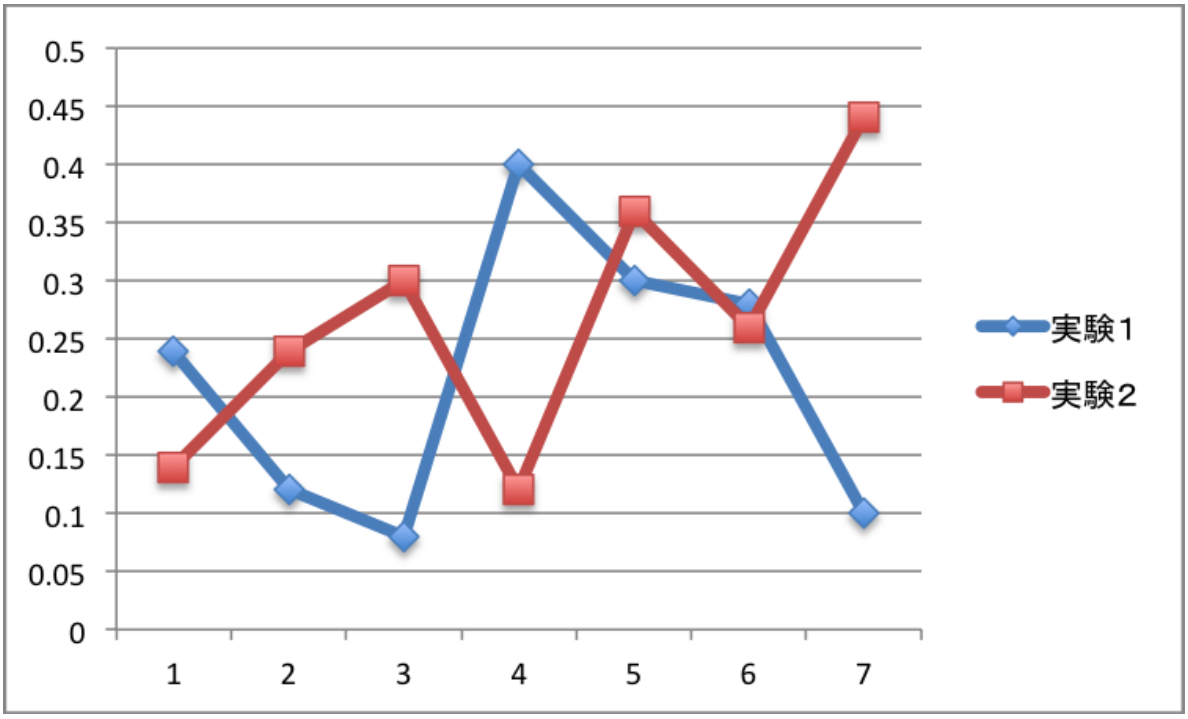

図 6.1　実験 1 と実験 2 のヒット率

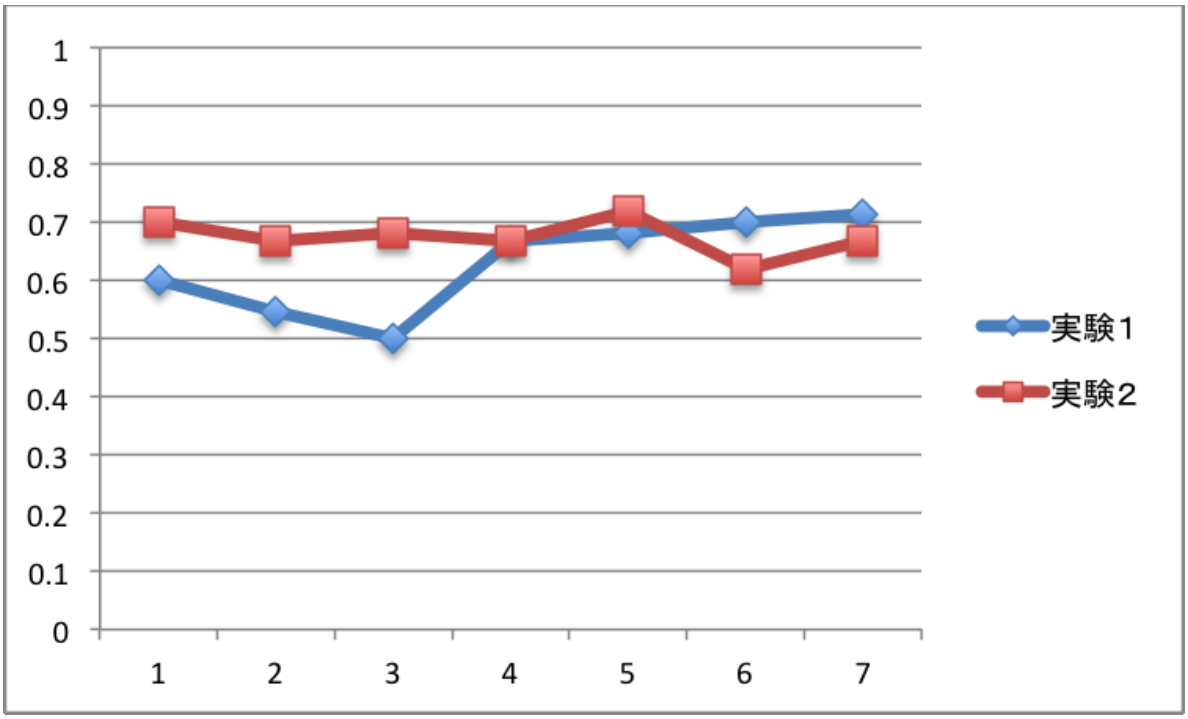

図 6.2　実験 1 と実験 2 の有用率

は検索サイトでのプリフェッチができていないからである。しかし、比較的に安定している有用率から、ヒット率の低下の影響で無駄なプリフェッチが行っていないことが分かった。

6.3 データ通信量

プリフェッチが行うと必ずヒットミスがある。プリフェッチしたが使用されていない Web ページをロードミスと言う。ロードミスは PC 上では影響はないが、データ通信量が限られている携帯端末には影響がある。そこで、本研究では、プリフェッチされたが、使用されていない Web ページの割合の計算も行った。その結果は図 6.3に示す。

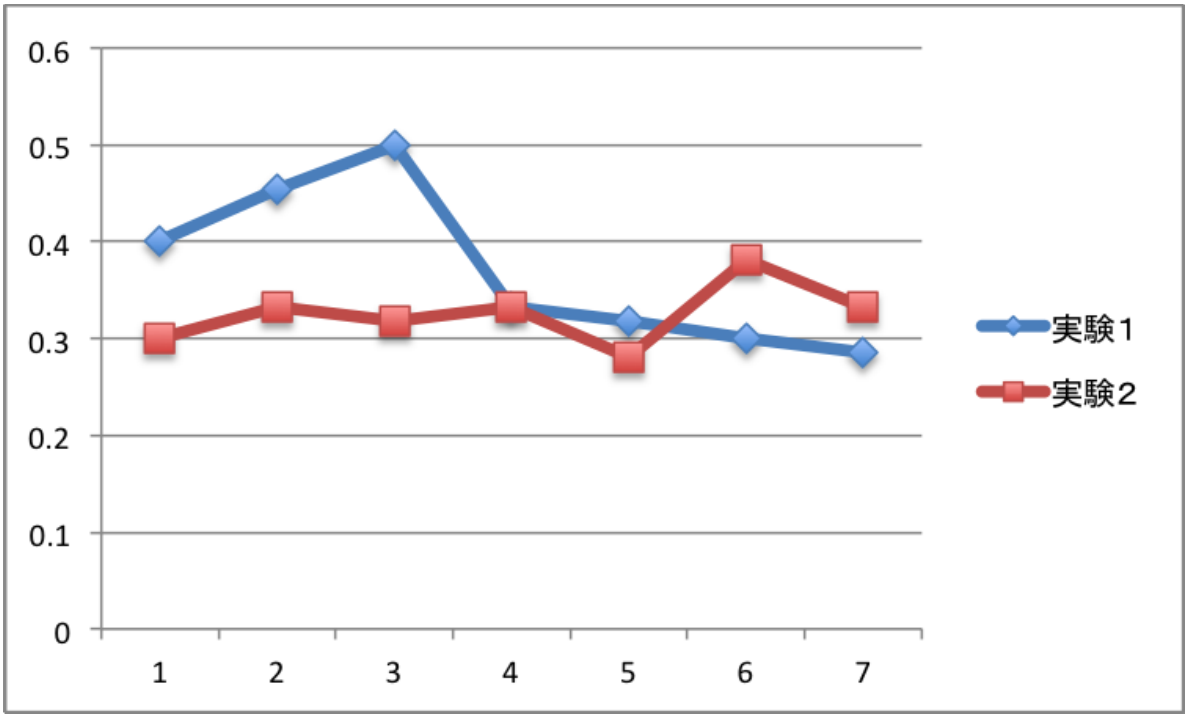

図 6.3 実験 1 と実験 2 のロードミスがプリフェッチした Web ページに占めしたの割合

結果からプリフェッチされた Web ページの中使用されていない Web ページの割合が安定しており、30% 程度になっている。より良い評価結果を得るために、ロードしたすべての Web ページの中で占めている割合を計算し図 6.4に示す。結果から日により少し差があるが、平均的に 10% 未満程度になっていた。

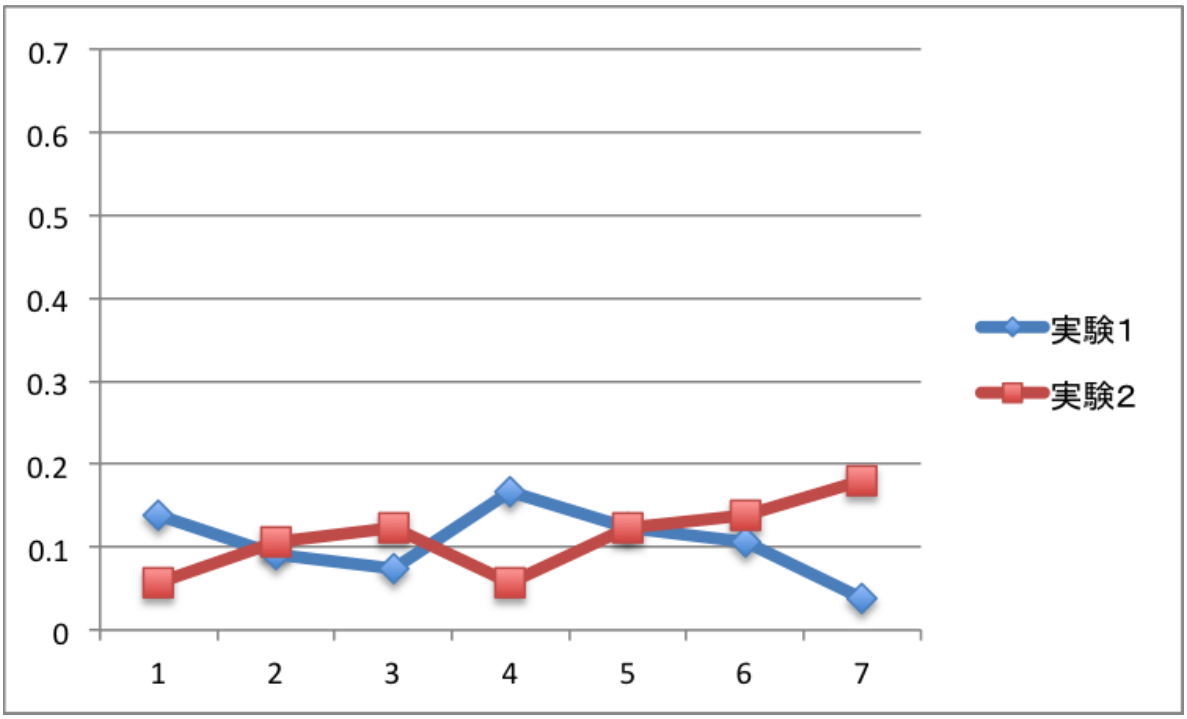

図 6.4　実験 1 と実験 2 のロードミスが全体ロードした Web ページに占めしている割合

6.4　バッテリの消費

本研究で主にバッテリ消費が多い部分はユーザ履歴を利用し、ブロック、連鎖、キーワードリストの作成する部分である。しかし、本研究の関連ファイルの作成は週 1 回ユーザが使用していない充電時に行うため、リアルタイムで、関連ファイルを修正している既存手法に比べ、バッテリの消費量がほぼ無視できる程度になる。

6.5　評価まとめ

提案手法の評価から、提案手法を使用するとアクセスの 25% の Web ページがプリフェッチされることが分かった。既存研究 [2][1] より比較的に低いヒット率になっているが、有用率が高くて、ロードミスが 10% 未満程度になっていることは、低いデータ通信量のオーバーヘッドで 25% の Web ページのロードが速くなり、ユーザには快適な Web 環境を提供していることが分かった。

第 7 章

おわりに

本研究は携帯端末の Web ブラウザのパフォーマンスの向上のために、ブラウジング習慣に基づいた携帯端末での Web ページのプリフェッチ技法を提案した。既存研究でマルコフ連鎖のノードが多いので、計算量が多い、携帯端末に適用できないという欠点をブロックと連鎖に分け、携帯端末でも簡単に使用できるよう改良した。また、マルコフ連鎖を利用したプリフェッチ方法ではユーザがアクセスしたことがない Web ページのプリフェッチにはキーワードをベースにしたプリフェッチ技法を利用した。また、実装を行い評価を行った。評価の結果ヒット率が平均 25% 程度になった。また、有用率が 65% で、ロードミスがすべての Web ページロードの 10% になた。パフォーマンスの向上が既存研究より低いが、計算量、通信データ量などを携帯端末に合わせた上で設計した Web ページプリフェッチ技法であるため、ユーザにより快適なネット環境を提供したと評価できる。

今後の展望として、携帯端末を利用し、検索サイトの利用頻度が最も高いので、検索サイトでのプリフェッチも実現できるとヒット率とパフォーマンスより向上すると考えられる。また連鎖の作成のため、データの絞り込みをより緩めに変更すると、データ通信量がより増加する恐れがあるが、パフォーマンスの向上がより改善できると考えられる。本研究では、ヒットによるキーワードリストと連鎖の再計算を行っていない。ヒットにより事前に作成されたファイルがある程度の変化も与えられたらより良いプリフェッチ技法になると考えている。

謝辞

本研究を遂行するにあたっては、いろいろな方々にお世話になりました。

まず、指導教員の多田好克先生には日頃から熱心なご指導、そしてご鞭撻を賜わりました。また、ご多忙中にもかかわらず論文の草稿を丁寧に読んで下さり、大変貴重なご助言をいただきました。ここに厚く御礼申し上げます。

講座内進捗や講座内輪講などにおいて研究を進める上での貴重な意見や助言を頂いた、小宮先生には深く感謝します。

研究室先輩から研究についての助言いただき、同期の皆様から、日本語の文章を修正していただきました。ここに感謝の意を表します。

参考文献

[1] C-Z. Xu, S. Member, et al.: A Keyword-Based Semantic Prefetching Approach in Internet News Services, IEEE Transactions on Knowledge and Data Engineering, VOL.16, pp.601-611(2004).

[2] J. Zhu, J. Hong and J.G. Hughes.: Using Markov Chains for Link Prediction in Adaptive WebSites, Proc. First International Conference on Computing in an Imperfect World, pp.60-73(2002).

[3] S. Brin, L.Page, et al.: The PageRank citation ranking: bringing order to the web, Tech. Rep. 1999-0120, ComputerScience Department, Stanford University, Stanford, Calif, USA(1999).

[4] "Average Web page size", https://gigaom.com/2014/12/29/the-overweight-web-average-web-page-size-is-up-15-in-2014/, accessed 2015-01-28

[5] "Numpy Library"https://networkx.github.io/documentation/latest/reference/generated/networkx.conver_matrix.to_numpy_matrix.html, accessed 2015-01-28

[6] "Webview Class", http://developer.android.com/intl/ja/reference/android/webkit/WebView.html, accessed 2015-01-28

[7] "Website Structure Understanding and its Applications", http://research.microsoft.com/en-us/projects/website_structure_mining/, accessed 2015-01-28

[8] "情報処理用流れ図・プログラム網図・システム資源図記号"、http://kikakurui.com/x0/X0121-1986-01.html、参照 2015-01-28

付録 A

ソースコード

```
1 import networkx as nx
2 line2= f1.readline() # The next line
3 g = nx.DiGraph() # prepare for make markov chain
4 elenum=0    #chain element number
5 tmplist=[]  #row number
6 url =[]#chain element
7 dicttmp={}  #temporary dict to connet the element number with url
8 def addno(string):  # make node for the markov chain
9     global url
10    global dicttmp
11    global elenum
12    global g
13    if(string.find("\n")!= -1):
14        string = string.strip("\n")
15    url.append(string)
16    elenum=elenum+1
17    dicttmp[string]=str(elenum)
18    g.add_node(elenum)
19 def added(string1,string2):  # add the edge of two nodes in markov chain
20    global g
21    global dicttmp
22    if(string1.find("\n")!= -1):
23        string1 = string1.strip("\n")
24    if(string2.find("\n")!= -1):
25        string2 = string2.strip("\n")
26    g.add_edge(int(dicttmp.get(string1)),int(dicttmp.get(string2)))
27 for line in f :  #read file
28    line2=f1.readline()
29    if(line.find(">>>") != -1): # if the line reading now is the last line of
30                                #the same domain name web pages
31        pr=nx.pagerank_numpy(g,alpha=0.85)
32        for word in url:
33            f2.writelines(word+","+str(pr.get(int(dicttmp.get(word))))+"\n")
34        dicttmp.clear()
35        url.clear()
36        g.clear()
37        elenum=0
38        tmplist.clear()
39        f2.writelines("end\n")
```

```
40          continue
41      if(line[0]=="<"): # "<" means the web page access itself
42          line = line.strip("<")
43          if(line.strip("\n") not in url and line.find(">>>") == -1) :
44              addno(line)
45          else:
46              added(line,line)
47      elif(line.find(">")!=-1 and line != ">>>\n"): # ">" means it access a web page
48                                                    # with different domain name
49          tmplist=line.split(">")
50          if(tmplist[0] not in url and tmplist[0].find(">>>") == -1):
51              addno(tmplist[0])
52      else:             # else type is the url in this line will access the url of next li
53          if(line.strip("\n") not in url and line.find(">>>")== -1):
54                 addno(line)
55          if(line2.strip("\n") not in url and line2.find(">>>")== -1):
56              if(line2.find(">")):
57                  tmplist1=line2.split(">")
58                  addno(tmplist1[0])
59                  added(line,tmplist1[0])
60              else:
61                  addno(line2)
62                  added(line,line2)
```

ソースコード A.1　ページランク計算コード

```
1  url = []  # URLs with the same domain name
2  pr=[]  #pagerank of the URLs with the same domain name
3  domainfornode=[] #domain name for the markov chain node
4  urlfornode=[] # url for the markov chain node
5  next1fornode=[] # The next access Web page with the highest priority
6  next2fornode=[] # the page will be prefetch second time
7  next3fornode=[] # the page will be prefetch at last chance
8  dict={} # for sort the next access web pages by pagerank*access times
9  for line in readpr:
10     f=open("tempfile.csv","r",encoding="utf-8")
11     if(line.startswith("end")==True): # all of the Web pages with the same domain name
12         for lines in f: # read the access patten
13             forcount=lines.split(",")
14             urlsinline = forcount[0].split(">")
15             tmplista=urlsinline[0].strip("\n")
16             tmplistb=urlsinline[1].strip("\n")
17             if(tmplistb in url):
18                 dict[tmplista+">>"+tmplistb]=str(float(forcount[1].strip("\n"))
19                                                  *float(pr[url.index(tmplistb)]))
20         d = [(v,k) for k,v in dict.items()]
21         d.sort()
22         d.reverse()
23         for count, word in d[:]: # input the next1,2,3 and
24                                  #if it access it self input "keyword"
```

```
25              lastlist=word.split(">>")
26              if lastlist[0] in urlfornode :
27                  if(lastlist[0] != lastlist[1] and
28                              next1fornode[urlfornode.index(lastlist[0])] == ""):
29                      next1fornode[urlfornode.index(lastlist[0])] = lastlist[1]
30                  elif(lastlist[0] != lastlist[1] and
31                              next2fornode[urlfornode.index(lastlist[0])] == ""):
32                      next2fornode[urlfornode.index(lastlist[0])] = lastlist[1]
33                  elif(lastlist[0] != lastlist[1] and
34                              next3fornode[urlfornode.index(lastlist[0])] == ""):
35                      next3fornode[urlfornode.index(lastlist[0])] = lastlist[1]
36                  elif(lastlist[0] == lastlist[1] and
37                              next1fornode[urlfornode.index(lastlist[0])] == ""):
38                      next1fornode[urlfornode.index(lastlist[0])] = "keyword"
39                  elif(lastlist[0] == lastlist[1] and
40                              next2fornode[urlfornode.index(lastlist[0])] == ""):
41                      next2fornode[urlfornode.index(lastlist[0])] = "keyword"
42                  elif(lastlist[0] == lastlist[1] and
43                              next3fornode[urlfornode.index(lastlist[0])] == ""):
44                      next3fornode[urlfornode.index(lastlist[0])] = "keyword"
45          for i in range(len(urlfornode)): # make markov chain
46              markovchain.writelines(domainfornode[i]+","+urlfornode[i]+","+
47                    next1fornode[i]+","+next2fornode[i]+","+next3fornode[i]+",\n")
48          f.close()
49          markovchain.writelines("endforthischain\n")
50          url.clear()
51          pr.clear()
52          domainfornode.clear()
53          urlfornode.clear()
54          next1fornode.clear()
55          next2fornode.clear()
56          next3fornode.clear()
57          dict.clear()
58          lastlist.clear()
59          tmplist.clear()
60      else: # if the line not the "end"
61
62          tmplist = line.split(",")
63          url.append(tmplist[0])
64          pr.append(tmplist[1].strip("\n"))
65          ttmp=tmplist[0].strip("http://")
66          if ttmp.startswith("s://"): ttmp=ttmp.strip("s://")
67          textdo=ttmp.split(".")
68          #init all element
69          if(textdo[1]=="com" or textdo[1] == "tv" or
70             textdo[1]=="net" or textdo[1]=="co" or
71              textdo[1] == "gl" or textdo[1]=="org"):
72              domainname=textdo[0]
73              domainfornode.append(domainname)
74              urlfornode.append(tmplist[0])
```

```
75              next1fornode.append("")
76              next2fornode.append("")
77              next3fornode.append("")
78          else:
79              domainname=textdo[1]
80              domainfornode.append(domainname)
81              urlfornode.append(tmplist[0])
82              next1fornode.append("")
83              next2fornode.append("")
84              next3fornode.append("")
```

ソースコード A.2　連鎖の作成