ダブル配列による動的辞書の構成と評価

矢田 晋† 田村 雅浩 森田 和宏 泓田 正雄 青江 順一
徳島大学知能情報工学科

1 はじめに

ダブル配列は極めて高速な前方一致検索を提供するデータ構造であり，言語処理における静的な語彙辞書などに利用されている [3, 5]．ダブル配列の動的な更新を効率化する手法 [4, 6] が提案されているものの，記憶領域と検索時間において静的な辞書に劣るだけでなく，更新時間が安定しないという欠点もあり，現状での利用例は少ない．そこで，本研究では，ダブル配列の動的な更新をさらに効率化する手法を提案するとともに，静的なダブル配列を含めた評価をおこなう．

2 関連研究

2.1 ダブル配列

ダブル配列は準静的検索のために提案されたデータ構造であり，Minimal Prefix（MP）トライを構成する木構造と文字列リストをそれぞれ一次元配列により表現する [2]．ダブル配列は時間効率・空間効率の両面に優れており，前方一致検索に対応する辞書を構成できるという特徴から，言語処理における利用例が多い．

2.2 ダブル配列による動的辞書

初期のダブル配列が持つ最大の欠点は，新たなキーを追加して辞書の語彙を拡張するとき，未使用要素を見つけるために，木構造に割り当てた配列全体を線形探索しなければならないことである．これは大規模な辞書を構築する際に致命的な問題となるため，森田らによって，探索範囲を限定する手法や未使用要素を連結する手法が提案された [4]．また，未使用要素の連結を双方向にする手法や木構造の探索を高速化する手法 [6] などにより，ダブル配列の更新に潜むボトルネックの多くが解消されている．しかし，語彙や登録順序によって更新時間が大幅に変動するため，辞書の更新に必要なリソースの算出が難しいという欠点を持つ．

2.3 ダブル配列による静的辞書

語彙の変化する頻度が低い状況では，辞書の更新にかかるリソースより，辞書のサイズや検索時間の重要性が高くなる．そのため，語彙を逐次拡張しないことを前提に，コンパクトかつ高速なダブル配列を静的に構築する手法 [7] が提案されている．1,000 万件のキーにより構成される大規模な語彙からでも数十秒程度で静的に辞書を構築できるため，リアルタイムに更新する必要性がなければ，動的辞書を用いるよりも効率的な検索システムを実現できる．

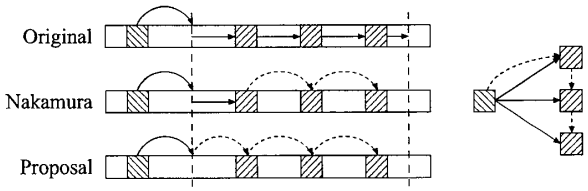

図 1: ダブル配列における子の列挙とノードリンク

3 提案手法

3.1 提案手法の概要

本稿では，中村らの手法 [6] を改良したノードリンクにより更新時間を短縮する手法と，ブロック単位で未使用要素を管理して更新時間の変動を制御する手法を提案する．提案手法を用いると，従来手法では更新時間の悪化が見られる語彙に対して，空間効率の悪化を抑えつつ高速に辞書を更新することができる．

3.2 ノードリンクによる探索の効率化

ダブル配列にキーを追加する場合，子や兄弟を列挙する機能が木構造に必要であり，初期の実装では配列上のそれらしい範囲を全探索することで実現していた．この操作に対し，中村らの手法 [6] は，兄弟間にリンクを張ることで一つ目以降の列挙を高速化する．

提案手法は，兄弟間のリンクに加えて親から子へのリンクを張ることにより，一つ目を見つけるための探索を省略できるようにする．また，中村らの手法が絶対インデックス（通常 4 bytes）を用いるのに対し，相対インデックス（通常 1 byte）を用いることでダブル配列のサイズ拡大を防ぐ．手法による列挙方法の違いを図 1 に示す．

3.3 ブロック管理による探索範囲の限定

未使用要素を連結したダブル配列 [4] では，最悪の場合，キーの追加にともなって未使用要素を全探索することになる．そのため，未使用要素の蓄積はダブル配列の更新時間を大きく悪化させる．探索範囲の限定 [4] は更新時間の制御に有効であるものの，空間効率が大きく悪化するという欠点を持つ．

そこで，ブロック単位で未使用要素を管理する手法を提案する．提案手法は，ブロック内の未使用要素を連結し，ブロック同士を連結する．また，未使用要素の数と過去の探索で利用に失敗した回数を利用して，表 1 に示すようにブロックを分類する．さらに，分類に応じた用途を与えることにより，語彙や登録順序に適し

Sequential Insertions and Performance Evaluations for Double-arrays
Susumu YATA, Masahiro TAMURA, Kazuhiro MORITA, Masao FUKETA, and Jun-ichi AOE
Information Science and Intelligent Systems, University of Tokushima
†Supported by Research Fellowships of the JSPS for Young Scientists.

表 1: ブロックの分類

分類	分類条件	用途
Full	$E = 0$	使用しない
Closed	$E \geq 1$ & $F \geq T$	シングル要素の移動先
Open	$E > 1$ & $F < T$	あらゆる要素の移動先

※ E:未使用要素数, F:連続探索失敗回数, T:閾値

表 2: 評価対象とした手法の一覧

更新	手法	概要
動的	Nakamura	中村らの手法
	Proposal	ノードリンクとブロック管理
静的	Darts	探索範囲の限定
	Darts-C	探索範囲の限定，圧縮

た探索範囲の制御を実現できる．提案手法では，一度利用が難しいと分類されたブロックにも再利用の機会があり，空間効率と更新時間のいずれかが極端に悪化しないように制御される．

4 評価

中村らの手法と提案手法を C++ で実装し，静的なダブル配列である Darts[3] および Darts-clone[5] とともに，Core2 Quad 2.83 GHz を用いて性能を評価した．評価対象とした手法の一覧を表 2 に示す．実験に用いた語彙は，英語版 Google n-gram コーパス（Web 1T 5-gram Version 1）[1] の unigram であり，長さの平均が 8.27 bytes のキー 13,588,391 件により構成されている．

構築時間の比較を図 2 に示す．キーの追加は辞書順を基本とし，ランダム順に追加した結果はサフィックス "-R" により区別している．提案手法については，閾値 $T = 32$ を用いた結果を示している．図 2 より，中村らの手法には更新時間の急激な悪化が見られるのに対し，提案手法は静的辞書と同等の時間で辞書を構築できていることが分かる．なお，提案手法が特に有効となるのは，短いキーを中心とした大規模な語彙を扱う場合など，木構造中の分岐数が大きくなる状況である．言語処理で用いる基本語彙など，従来手法でも高速に動作する語彙を対象とする場合，ブロック管理による効率化が働かないため，提案手法による更新時間の短縮は 10–50%程度にとどまる．

構築された辞書の性能を表 3 に示す．検索時間は，すべてのキーをランダム順に検索するのに要した時間を示している．表 3 より，提案手法を適用した結果において，未使用要素の蓄積による検索用領域の拡大が確認できる．なお，更新時間と空間効率のトレードオフに対しては，閾値により比重を変更することができる．また，リンクの表現方法を改良したため，更新用領域が縮小されている．検索時間については，要素の移動によってキャッシュ効率が低下するため，ランダム順にキーを追加した動的辞書の性能が最も低くなっている．

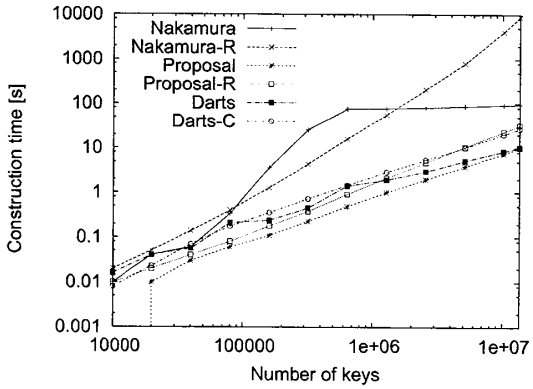

図 2: 各手法による辞書の構築時間

表 3: 各手法により構築された辞書の性能

手法	サイズ [KB] 検索用	更新用	検索時間 [s] ランダム順
Nakamura	195,373	80,325	9.04
Nakamura-R	195,580	80,428	10.11
Proposal	195,378	41,105	9.30
Proposal-R	208,358	44,426	10.23
Darts	406,358		8.51
Darts-C	146,037		8.92

5 まとめ

本稿では，リンクの拡張とブロック単位の管理によりダブル配列の更新を効率化する手法を提案し，その有効性を示した．

謝辞 本研究にご協力いただきました，広屋修一様，織田美樹男様に感謝致します．

参考文献

[1] T. Brants and A. Franz. Web 1T 5-gram Version 1. http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2006T13, 2006.

[2] 青江. ダブル配列による高速ディジタル検索アルゴリズム. 電子情報通信学会論文誌, J71–D(9):1592–1600, September 1988.

[3] T. Kudo. Darts: Double-ARray Trie System. http://chasen.org/~taku/software/darts/, 2008.

[4] K. Morita, M. Fuketa, Y. Yamakawa, and J. Aoe. Fast insertion methods of a double-array structure. *Software—Practice and Experience*, 31:43–65, 2001.

[5] S. Yata. darts-clone — Google Code. http://code.google.com/p/darts-clone/, 2009.

[6] 中村, 野村, 望月. 遷移先節点集合を導入したトライ構造における更新手法の実現. 情報処理学会研究報告, 2006(33):1–6, 2006.

[7] 矢田, 森田, 泓田, 平石, 青江. ダブル配列におけるキャッシュの効率化. 第 5 回情報科学技術フォーラム（FIT2006）, pp. 71–72, September 2006.