ディスク上の大規模グラフデータを対象とした集合単位ナビゲーションの効率化*

只石正輝 (学籍番号 200821664)
研究指導教員: 森嶋厚行
副研究指導教員: 鈴木伸崇

1. はじめに

今日，計算機で扱われているデータを表現する形式として，ノード (頂点) とノード間をつなぐ矢印であるエッジ (辺) で構成されたグラフ構造が広く利用されている．たとえば，計算機のディレクトリ構造はグラフ構造の一種である木構造で表現されている．また, SNS の一つである twitter [1] では各ユーザをノードとし，ユーザ u_1 がユーザ u_2 を follow している (u_1 が u_2 の発言を取得している) というユーザ間のつながり ($u_1 \overset{follow}{\rightarrow} u_2$) をエッジとしたグラフ構造で表現されている．また，これらのデータのサイズは非常に巨大である．

したがって，大規模なグラフ構造に対する問合せの効率化は重要な問題である．たとえば，次のような問合せを効率的に処理することは重要である．**Q1:**コンピュータのディレクトリ構造において，あるディレクトリ d 以下に存在する全てのファイルの集合を計算する問合せ．**Q2:**人間関係等を表した SNS データにおいて，あるユーザ u が follow している人物の集合を取得する問合せ．

しかし，これらの問合せの処理の効率化は難しい．なぜなら，大規模グラフデータはディスクに格納して演算を処理する必要があるが，ディスク上のグラフデータに対する問合せ処理の効率的なアルゴリズムは知られていないためである．

そこで本研究では，計算機のディスク上に格納された大規模グラフデータを効率よく処理するための手法を開発した．具体的には，**Q1** や **Q2** などの問合せを記述可能な演算である次の 8 つの集合単位ナビゲーションの効率化を目的として研究を行った．

$$a \rightarrow X,\ a \overset{*}{\rightarrow} X,\ X \rightarrow a,\ X \overset{*}{\rightarrow} a,$$
$$a \overset{l}{\rightarrow} X,\ a \overset{l*}{\rightarrow} X,\ X \overset{l}{\rightarrow} a,\ X \overset{l*}{\rightarrow} a.$$

ここで，$a \rightarrow X$ は，あるノード a からエッジを矢印の向きに従って，1 回たどることで得られるノード集合を計算する演算である．また，$a \overset{*}{\rightarrow} X$ は，あるノード a からエッジを矢印の向きに従って，0 回以上たどることで得られるノード集合を計算する演算である．$X \rightarrow a$，$X \overset{*}{\rightarrow} a$ はそれぞれ，エッジを矢印の向きとは逆の向きに，1 回，または 0 回以上たどることで得られるノード集合を計算する演算である．残りの 4 つの演算は，$a \rightarrow X$，$a \overset{*}{\rightarrow} X$，$X \rightarrow a$，$X \overset{*}{\rightarrow} a$ の演算をラベル l を持つエッジのみに限定して行う演算である．

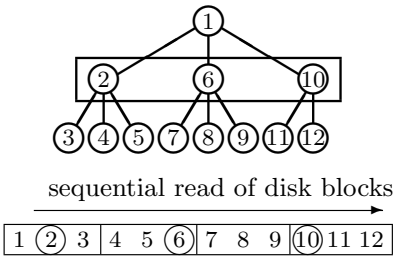

図 1　深さ優先順での $a \rightarrow X$

Q1 や **Q2** は，これらの演算を利用することで記述可能である．たとえば **Q1** は $d \overset{*}{\rightarrow} X$，**Q2** は $u \overset{follow}{\rightarrow} X$ と記述可能である．

2. ディスク上の大規模グラフデータを対象とした集合単位ナビゲーションの効率化

提案手法では，集合単位ナビゲーションの処理をするために必要な I/O コストが小さくなるような順序で各ノードをディスクに配置することで，問合せ処理の効率化を行う．その基本となるアイデアは，問合せの解となるノードができるだけ連続して配置されるような順序でノードをディスクに配置することである．この配置順序によって，アクセスする必要があるディスクページの数が少なくなり，効率的に問合せを処理することが可能となる．

ノードの配置順序としては，一般には深さ優先順や幅優先順が考えられる．しかし単純な深さ優先順では $a \rightarrow X$ の解が，また単純な幅優先順では $a \overset{*}{\rightarrow} X$ の解が連続して配置されない．その例を図 1 を用いて説明する．図 1(下) は，図 1(上) の木の各ノードを深さ優先順に配置したものである．このとき，$a \rightarrow X$ の解は連続して配置されるとは限らない．たとえば，図 1(下) において $1 \rightarrow X$ の解となるノード集合 ({2,6,10}) は連続して配置されていない．また，木を幅優先順で配置した場合，$a \overset{*}{\rightarrow} X$ は連続して配置されない．

本研究で提案するノードの配置順序を図 2 に示す．図 2 において，各ノードに振られた番号が各ノードの配置順序を表す．提案するノード配置順序

* “Efficient Processing of Set-Based Navigations for Large Graph Data on the Disk” by Masateru TADAISHI

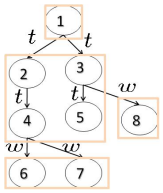

図 2　提案するノード配置順序

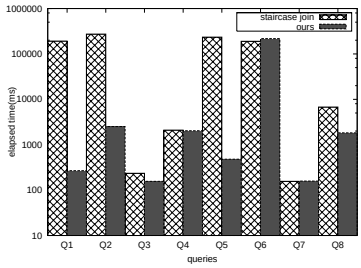

図 3　実験結果

では，$a \xrightarrow{l} X$，$a \xrightarrow{l*} X$，$a \rightarrow X$，$a \xrightarrow{*} X$ の問合せはいずれも効率的に処理可能となる．まず，$a \xrightarrow{l} X$，$a \xrightarrow{l*} X$ の両方の解が連続して配置される．たとえば，$1 \xrightarrow{t} X$ や，$1 \xrightarrow{t*} X$ はいずれも連続して配置されている．次に，$a \rightarrow X$ でアクセスする必要がある連続したディスク領域の数は，a を始点とするエッジのラベルの数となる．たとえば，$3 \rightarrow X$ の解は，2 つの連続したディスク領域 ($\{5\}$，$\{8\}$) にアクセスすることで解を取得可能である．また，$a \xrightarrow{*} X$ の処理でアクセスする必要がある連続したディスク領域の数は 2 つとなる．たとえば，$2 \xrightarrow{*} X$ の解は，2 つの領域 ($\{4\}$，$\{6,7\}$ が存在する領域) にアクセスすることで解を取得可能である．

提案するノードの配置順序は次の 2 つの Step で決定する．**Step 1** ノードをクラスタリングし，各クラスタを深さ優先順に配置．**Step 2** クラスタ内のノード配置を決定．以降ではそれぞれについて順に説明する．

Step 1 ノードのクラスタリング． 図 2 を用いてノードのクラスタリングについて説明する．図 2 において矩形が一つのクラスタに対応する．クラスタリングは次の 3 つを順に適用することで行う．**1.** ルートを一つのクラスタとする．**2.** ルートから同一のエッジラベルで到達可能なノード集合を一つのクラスタとする．**3.** 別のエッジラベルが出現したとき **2.** で得られたクラスタの各ノードをルートとし，**2.** を適用する．たとえば図 2 において，ルートノードを一つのクラスタとし，ルートノードから同一のエッジラベル t で到達可能なノード集合 $\{2,3,4,5\}$ を一つのクラスタとする．最後に，異なるエッジラベル w が出現したとき，クラスタの各ノード (3，4) をそれぞれルートとし，再帰的にクラスタリングを行う．その後，各クラスタを深さ優先順に配置する．

Step 2 ノードの配置順序の決定． 各ノードの順序は次のように決定する．**1.** 兄弟ノードごとにグルーピング．**2.** 各兄弟グループを深さ優先順に配置．たとえば図 2 での $\{2,3,4,5\}$ の順序を決定する場合，兄弟ノードごとに $\{2,3\}$，$\{4\}$，$\{5\}$ の 3 つのグループにグルーピング後，各兄弟グループを深さ優先順に配置するため，$\{2,3\}$，$\{4\}$，$\{5\}$ の順に各ノードを配置する．

提案したノード配置順序では $X \rightarrow a$，$X \xrightarrow{l} a$，$X \xrightarrow{*} a$，$X \xrightarrow{l*} a$ の解は連続して配置されない．そのため，問合せを効率的に処理できない可能性がある．この問題に対して，木のルートのように多くの問合せの解に含まれるノードをキャッシュとしてメモリ上に保持することで I/O コストを小さくする．

3. 評価実験

提案手法を評価するために実験を行った．実験環境は次のとおりである．OS Windows XP SP2, CPU Pentium M 1.73 GHz，メモリ 512MB．またプログラムの実装言語は Java である．

実験では，提案手法と深さ優先順で各ノードを配置し問合せを処理する手法である Staircase join [2] との比較を行った．通常の Staircase join ではキャッシュを考慮していないため，提案手法でも同様にキャッシュを利用せずに問合せを処理した．

実験内容． 実験で用いた木は，著者が所属する研究室のファイルサーバである．ファイルサーバに存在するファイル／ディレクトリの総数は 8,024,321 である．この木のエッジラベルは，ディレクトリからディレクトリへのエッジの場合 **subdir**，それ以外の場合 **contains** となる．

実験では，次の 8 つの集合単位ナビゲーションを処理し，実行時間を測定した．(Q1: $d \overset{subdir}{\rightarrow} X$, Q2: $d \overset{subdir*}{\rightarrow} X$, Q3: $X \overset{subdir}{\rightarrow} d$, Q4: $X \overset{subdir*}{\rightarrow} d$, Q5: $a \rightarrow X$, Q6: $a \xrightarrow{*} X$, Q7: $X \rightarrow a$, Q8: $X \xrightarrow{*} a$) ここで，d，a は，それぞれディレクトリ，ノード (ディレクトリかファイルのいずれか) を表している．

実験結果． 図 3 に実験結果を示す．図 3 において，実行時間はそれぞれのナビゲーションを 100 回行った結果の合計である．実験から，問合せ処理の実行時間は，Staircase join とほぼ同じ (Q3，Q4，Q6，Q7) か，Staircase join よりも効率的に問合せを処理可能である (Q1，Q2，Q5，Q8) ことが分かった．

4. まとめ

本研究では，大規模グラフに対する集合単位ナビゲーションを効率化するためのディスク格納手法を提案した．問合せを効率的に処理するために，問合せの解となるノードが連続して配置されるようにディスク上に各ノードを配置した．実験では，提案手法が既存の手法よりも優れていることを示した．

文献

[1] http://twitter.com
[2] Torsten Grust, Maurice van Keulen, and Jens Teubner. Staircase join: Teach a relational DBMS to watch its (axis) steps. In VLDB, pp. 524–525, 2003.