Junfeng Wu

DEVELOPMENT OF AN EFFECTIVE DATA STRUCTURE, SUPPORTING INCREMENTAL CHANGES IN PARALLEL TREE PROCESSING

ABSTRACT

Dynamic tree problem is an important problem in graph theory, it has been studied and applied widely. Dynamic tree data structures are used to support many applications, like edge insertions and deletions for a forest. Dynamic tree algorithms solve the problems with incremental changes in the data while parallel tree contraction provides a technique for the parallel solution of it. Many dynamic algorithms are based on parallel tree contraction, and the data structure used for supporting dynamic trees are ST-Trees, Topology Trees, Top Trees and RC-Trees. RC-Trees works faster than previous data structure when answering dynamic queries, and it takes O (log n) operation times.

Dynamic tree is still not solved perfectly and still under discussing. With technical development, to solve dynamic queries are more important than before. RC-Trees supports the dynamic queries in dynamic tree, but from the nowadays' point of view, it is not very efficient.

In this thesis, we evaluate the previous data structure for solving the dynamic problems, and in conclusion we choose RC-Trees data structure as the basic data structure for developing new data structure. Evaluate the requirement for making the algorithm more efficient in parallel processing. We solve the research problem in this thesis that What data structure can be developed for solving solve dynamic tree contraction problems in parallel based on RC-Trees data structure and What we need to do to implement this algorithm.

Keywords: data structure, dynamic problem, parallel tree contraction

FIGURES

TABLE OF CONTENTS

1 Introduction

Dynamic tree problem is an important problem in graph theory, it has been studied and applied widely. The dynamic tree problem is a proach to maintain a forest with n vertices that changes over time. A forest is a disjoint union of trees. By applying dynamic tree, we can optimize the common algorithm, like solving the maximum-flow problem. Goldberg A. V. et al. (1988). And with dynamic tree, we can get very good time complexity. Dynamic tree data structures are used to support many applications, it was first considered by Sleator D D et al. (1983) and presented supporting edge insertions and deletions (Sleator D D et al. (1983))for a forest. A dynamic tree structure solving the dynamic tree problem by answering queries in a forest of trees when the trees are changing.

Dynamic tree algorithms solve the problems with incremental changes in the data while parallel tree contraction provides a technique for the parallel solution of it. Miller G. L. et al. (1985) and Miller G. L. et al. (1989) first introduced parallel tree contraction and further discussed in Miller G. L. et al. (1991) and Reif J. et al. (1994). Parallel tree contraction is a wide used method for parallel tree processing.

Many dynamic algorithms are based on parallel tree contraction, and the data structure used for supporting dynamic trees are ST-Trees, Topology Trees, Top Trees and RC-Trees, Acar U. et al. (2003). RC-Trees, also known as Rake & Compress Tree, was invited by Acar U. et al. (2003) to solve dynamic tree problems, they dynamized the parallel tree contraction algorithm of Miller G. L. et al. (1985) in their research Acar U. et al. (2004). RC-Trees works faster than previous data structure when answering dynamic queries, and it takes O (log n) operation times. In this thesis, we analyse the adavatages of RC-Trees, and evaluate the demands for developing new data structure based on the RC-Trees data structure for solving the dynamic problems.

Dynamic tree is still not solved perfectly and still under discussing. With technical development, to solve dynamic queries are more important than before. RC-Trees supports the dynamic queries in dynamic tree, but from the nowadays' point of view, it is not very efficient. Multicore computing is widely used in algorithm field, make the algorithm in more efficient parallel process

can reduce the computing time. But there is not a completed implementation of dynamic tree in parallel processing, so in this thesis, we evaluate the possibility to develop a data structure for more efficient dynamic operation for time saving.

The applications of parallel tree contraction are described in the thesis, which can be used to evaluate the efficiency of a data structure. One problem that parallel tree contraction can solve is change propagation. Acar U et al. (2005). Change propagation automatically adjusting the output of an algorithm to changes in the input. Another problem that tree contraction solves is expression evaluation, performing tree contraction in parallel would make the computation more efficiently.

The following research questions are considered in the thesis:

1. What data structure can be developed for solving solve dynamic tree contraction problems in parallel?

2. What we need to do to implement algorithm based this data structure?

3. What applications can be exampled with this algorithm?
4. How is the evaluation of the result from the research?

To research the problems in this thesis, we use literature review. The literature review, follows the introduction of Creswell J. (2007). We use literature review to review the previous related work and find out the answers for the research problem. Literature review makes a survey of related references about the key concept of dynamic tree and parallel tree contraction etc. In the literature review, we list some basic concept related in this field and theories which introduce the algorithm that will be used in the thesis, including dynamic tree problem, parallel tree contraction, RC-Trees data structure. And compare other data structure briefly, like Topology Trees, to evaluate the advantage of the RC-Trees. Then evaluate the RC-Trees which part of it should be improved.

2 Literature review

For developing an effective data structure, which supports incremental changes in parallel tree processing. We need to describe two main definitions which used in the research. Dynamic tree problem is an important problem in graph theory, it has been studied and applied widely. Dynamic tree data structures are used to support many applications, supporting edge insertions and deletions for a forest. Dynamic tree algorithms solve the problems with incremental changes in the data while parallel tree contraction provides a technique for the parallel solution of it. Parallel tree contraction is a wide used method for parallel tree processing. In this literature review, we describe the parallel tree contraction and dynamic tree.

2.1 Parallel Tree Contraction

Trees are fundamental to many computational problems, such as data sorting, nearest neighbour search. A tree is a (possibly non-linear) data structure made up of nodes or vertices and edges without having any cycle. Trees are important in parallel problems, Miller G. L. et al. (1985) and Miller G. L. et al. (1989) first introduced parallel tree contraction and further discussed in Miller G. L. et al. (1991) and Reif J. et al. (1994). Tree contraction is the operation for contracting a tree by removing some of the nodes. Atallah, M. J. (Ed.). (1998). Parallel tree contraction is a framework that supporting efficient parallel algorithms on trees. Morihata A. et al. (2008). An efficient parallel tree contraction algorithm provides the possibility to gather values in the tree efficiently and without conflict, as a result, leading to several efficient parallel algorithms. Morihata A. et al. (2011).

Parallel tree contraction is a bottom-up approach when handling trees. It allows to implement facilitating new algorithms easier with fewer processors and less time and makes complicated problem simpler with using parallel algorithms. Miller G. L. et al. (1985). Using parallel tree contraction, only takes O

(log N) parallel steps for an underlying tree structure regardless of its shape, it speedup the operations. Morihata A. et al. (2014).

In Parallel tree contraction, there are two abstract operations on trees, Miller G. L. et al. (1985) called them Rake and Compress. Rake is the operation that moves all leaves from tree T. Compress is the operation to remove all maximal chains of T in on step, maximal chain here means every node has a parent and one child, the last node of the chain has a child and it is node a leaf. Briefly, compress operation removes every two edge between two nodes in a chain of the tree T. In other word, for instance, in binary tree, the compress operation removes the node which only have one child and its parent has it as the only child. The extra operation, contract, is the simultaneous application of rake and compress to the entire tree. The task for tree contraction is to reduce it to a single node. First, define a rooted tree T with n nodes and root r. To perform the contraction, rake operations and compress operations are done in any order and in parallel. According to Miller G. L. et al. (1985), it only takes O (log n) time to reduce a rooted tree to a single node using O (n / log n) processors.

Morihata A. et al. (2008) introduced a contraction algorithm called Shunt based on Miller G. L. et al. (1985) parallel tree contraction. In this algorithm, it consists basic rake and compress operation, and a new operation – Shunt. A Shunt operation removes the leaf and its parent and connects the sibling of the leaf to its grandparent. Shunt leads to a simple and efficient parallel tree contraction on binary trees.

2.2 Dynamic Tree

The basic dynamic tree is a link and cut tree, it supports the edge insertions and deletions edges. Using the link and cut tree can answer various path queries. Balanced tree is an empty tree or its children from right and left, their deep degree gap is no more than 1, and both of its right sub tree and left sub tree are balanced tree. When trees are balanced, it is easy to compute various properties of trees in parallel. When they are not balanced, however, such computations can be difficult. The parallel tree contraction is well developed and the shunt algorithm is efficient, but for solving the dynamic tree problem, we need to construct a dynamic tree data structure. A data structure is a particular way of organizing data in a computer so that it can be used efficiently. Black, P. E. (2004). Dynamic tree data structures was first considered by Sleator D D et al. (1983). Link and cut are the two operations, link combines two trees into one by adding an edge, cut divides one tree into two by deleting an edge.

There are many data structures for dynamic tree problem. Reif J.et al. (1994) described a data structure for dynamic parallel tree contraction, the tree PT, it adds or deletes leaves of the original tree with incrementally processing parallel requests. If there are U numbers of nodes in the original tree, the dynamic parallel tree contraction takes O (log (|U| log n)) time and using O (|U|log n / log(|U|log n)) processors.

From the basic dynamic tree data structure, there are many new and different data structures are invented. ST-trees, ET-trees, (Tarjan R E, Werneck R F. Dynamic trees in practice. (2007)), and in addition, along with the tree contraction technique, topology trees, top trees and RC-Trees are developed. In the next section we take more detail of them, and compare and evaluate them for the development of a new data structure. Briefly, ST-Trees are the classic dynamic tree data structure created by Sleator D D et al. (1983), supporting path operations. It is also known as the link and cut trees. It was the first to make the operation of dynamic tree into logarithmic time. Topology trees data sturcture is the first contraction-based data structure. But in its data structure, all the vertices in the forest must have constant bounded degree. Tarjan R E. et al. (2007). Acar U. et al. (2003) invented RC-Trees, which requires the bounded degree of underlying tre too, but no require to be a constant. RC-tree is formed from an original tree, it is a support tree. RC-Trees have the operations inherit from the link-cut trees, which are rake, compress and contraction. The Rake node was the sub tree with a certain node as the root in the original tree, Compress node was the sub tree with a certain path as the root in the original tree. Rake and Compress operation can be done in parallel in a certain procedure. RC-Trees use generic interface to separate value updates. The extra implementations are needed to realize the RC-Trees which can make queries faster. In the research Acar U. et al. (2005). RC-Trees data structure is improved with change propagation, which automatically adjusts the output to changes in the input.

Comparing with the dynamic tree data structures and considering the parallel tree contraction, RC-Trees is a suitable data structure to support the incremental changes in parallel tree processing. The RC-trees need implemetation and the original RC-Trees data structure was introduced 10 years ago. To catch up the development of technologies, the data structure needs to be improved.

3 Dynamic tree data structures

Some well-known data structures are introduced to solve the dynamic tree problems. Here we discuss severeal of them: ST-Trees, Topology trees, Top trees and RC-Tree. We descibe the with a high level and show their limitations. The RC-Trees will be introduced more detail, because from the above review, RC-Tree is better for developing a new data structure for nowadays incremental changes in parallel processing.

3.1 ST-Trees

ST-Tree is the best well-know data structure for solving dynamic problems. Introduced by Sleator D D, Tarjan R E. (1983). In the idea of this data structure, link and cut operations are important, so the data structure also know as the link-cut trees.

ST-Trees make operations on a rooted tree, the root of the tree only have children and without parents. Three operations are introduced, link (v, w), which makes the link between the root v of a tree and the vertex w of another tree by adding the edge (v, w), let w be the parent of v; cut(v) deletes the edge between vertex v, v is not the root of a tree, with its parent, and divides the tree into two trees; evert (v) makes the vertex be a root to turn to a new tree, and the root will be the vertex in the new turned tree.

ST-Trees support path operations, the time costs O (log n) time per operation on a weight balanced tree in a worst case, and O (log n) time per operation on a play tree in amortized time case. The ST-Trees also can support other queries, but the information must be aggregated over paths only.

ST-Trees were the first to make the operation time for dynamic tree into logarithmic time, providing the basic concept for the development of data structure for solving dynamic problems.

3.2 Topology Trees

The first contraction-based data structure is the Topology Trees, which introduced by Frederickson, G. N. (1997). Topology Tree is a support tree.

We consider the support tree S of the underlying tree T. Every node in S is the subtree in T, and the cluster of S is the subtree corresponding to a node in the S. The T′ represents the cluster of S. The edge in T is an adjoining edge of T′. The degree of T′ is the number of adjoining edge. The vertex of a cluster T′ from S is a boundary vertex if it has adjoining edge. The number of boundary vertices of a cluster T′ is the boundary size of it. Acar U.et al. (2003).

Frederickson, G. N. (1997) introduced the Topology Trees interpret the cluster as vertices instend of edges as a support tree. The cluster have at most three degrees.

The directed topology tree data structure is developed for maintaining binary trees dynamically. The operation time it takes O (log n) time, where *n* is the number of vertices in the trees. The Topology Trees are the implementation of link-cut trees, so the basic operations of Topology Trees are also the link and cut operation.

Though the Topology Trees make dynamic tree contraction simpler, its requirement for all the vertices in the forest to have degree bounded by a constant limits the development of itself. And if we need to make the algorithm work more efficient, the data structure needs to support the changeable degree of the vertices in support tree.

3.3 Top Trees

Top Trees is a data structure based on a binary tree for unrooted dynamic trees. Alstrup S. et al. (1997). In the unrooted tree, the edges adjacent to each vertex are arbitraily arranged in fixed circular order. The cluster in Top Trees is a subtree or a path of the original underlying tree. And the clusters have boundary size at most two. When take the contraction operation, it takes O (log n) time, and the advantage of it is the cost time is O (log n) with the arbitrary degree of tree. The limitation of Top Trees is it only supports the binary tree.

A tree contraction performs the operations to make all the clusters in a support tree into a single cluster. The Top Trees supprot the contraction for the binary tree.

We can combine two clusters (u, v) and (v, w) when v is the common endpoint of these two clusters and has a degree of two. Here Alstrup S. et al. (1997) use the concep of compress, we call the combined cluster (u, w) as compress cluster. Then, if (w, x) is the successor of (v, x) and v has degree one, these clusters can be combined into a rake cluster. After combinition, the rake cluster has endpoints w and x. Each rake or compress cluster can be viewed as a parent that aggregates the information contained in its children.

The root of the top tree is cluster which is the entire underlying tree. For the dynamic changes from the underlying tree, for instance, link or cut operations are made, the data structure merely updates the contractions affected.

As the application of the data structure, it answers the dynamic problem queries. When performing a path query, for example, we need to know the path between vertex v and w, the expose(v,w) operation is called, which returns a root cluster having v and w as endpoints, and if the v and w are not in the same tree, the operation will return null. However, even if v and w are in the same tree, the contraction in the top tree may need to be change, because the root cluster will be the the path from v to w. Top Trees support aggregation over paths or trees directly. Not like Toplogy Trees, they do not require the degree of a vertex.

3.4 RC-Trees

An RC-Tree is a support tree obtained after the rake and compress operations of parallel tree comtraction by clustering a underlying tree. A support tree is a balanced tree mapped from a n aribitrary tree to be used to answering queries efficiently. Acar U. et al. (2003).

RC-Trees have the rake and compress operations, Rake operations delete all leaves in the tree and Compress operations delete an independent set of vertices on chains. Acar U. et al. (2005). The figure 1 shows the two operations as follows:

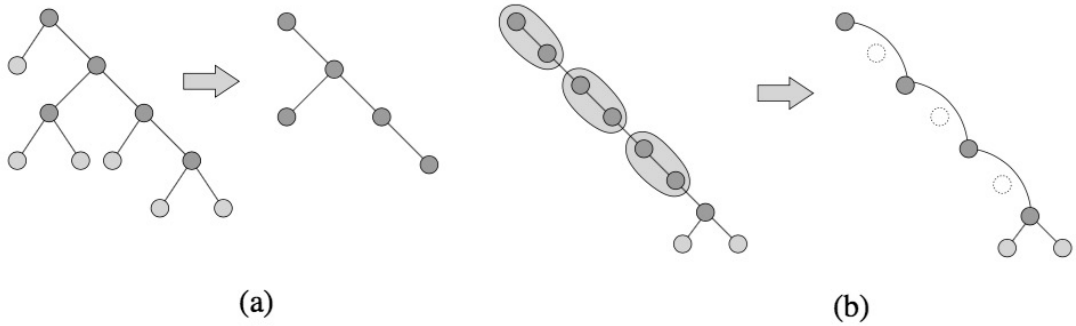

FIGURE 1 Rake(a) and Compress(b) operations.

Acar U. et al. (2003) has also indroduced an operation called finalize. Finalize operation, just as its name implies, it works when the tree is contracted to a single vertex, it stores this vertex. The main idea of RC-Trees: First, we need a unbalanced tree, called primitive tree. Then map it to a balanced tree, RC-Tree. Using the change propagation, introduced in Acar U. et al. (2005), to update RC-Tree dynamically.

The following figures shows the step of a tree contraction. We set a edge-weighted primitive tree, in figure 2, using Rake and Compress operation obtains a clustering, in figure 3. Finally, using finalize operation to store the last vertex, here the completed clustering. Then we get the RC-Tree, shows in figure 4.

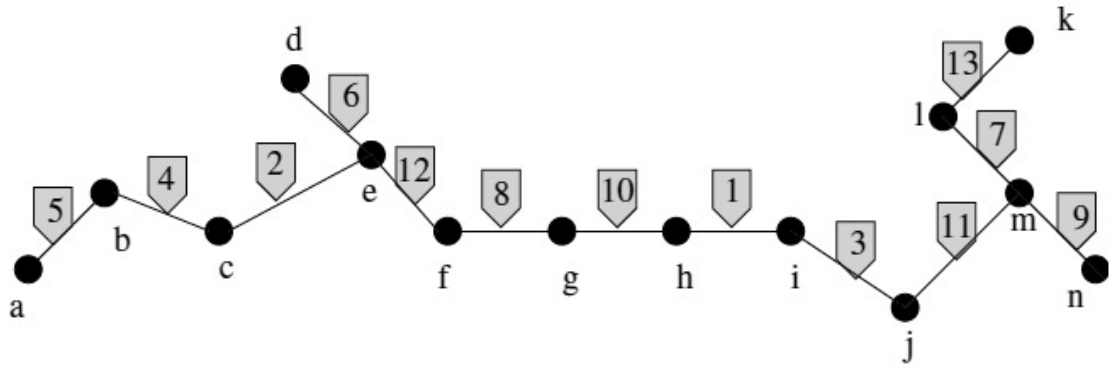

FIGURE 2 A primitive tree, edges are weighted.

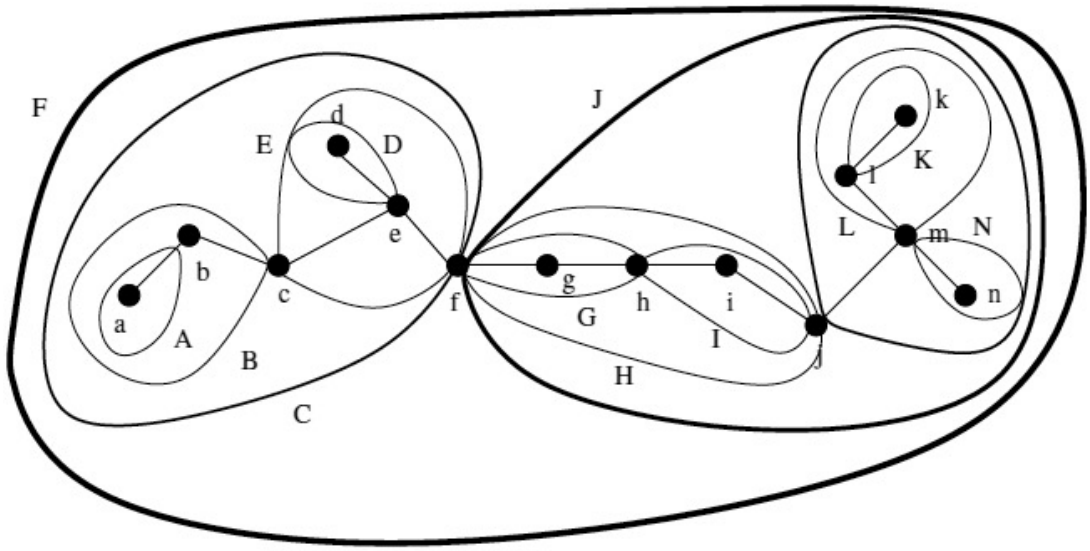

FIGURE 3 A completed clustering.

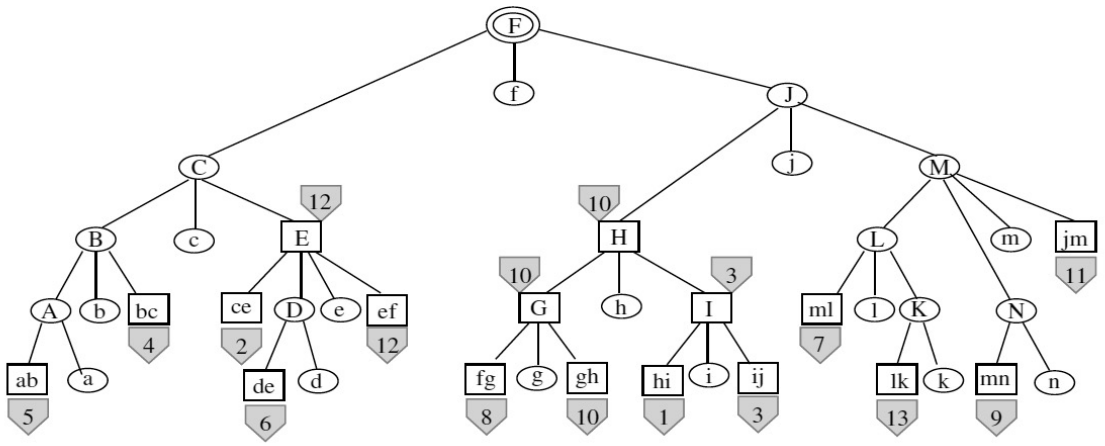

FIGURE 4 A RC-Tree.

One example query that RC-Trees can answer is a path query. A path query is to find the maximum weight path between two selected vertices. The path query can be answered by walking up the path simultaneously with two vertices. Set up two values of each path, left and right, left is the length between the left boundary of the cluster and the vertex, so the same as the right one. When the two vertices meet, the maximum of the value will be the answer.

To find least common ancestors is an application of RC-Trees. We can find the least common ancestor of vertices v and w with respect r, r is the root of the tree, by walking up the tree. The three vertices are operated simultaneously, when they all meet, walk down through the path when the two vertices walking up to find the least common ancestor.

4 Conclusion and Discussion

From the result of literature review, the parallel tree contraction data structure is not enough for solving our research problem. To answering the dynamic queries, we need to build a dynamic tree data structure. Have compared the dynamic tree data structures, RC-Tress is suitable to be a basic data structure used to support the incremental changes in parallel prosecing.

The feature of the RC-Trees itself requires the extra implementation of the RC-Trees, thus the first improvement need to make the implementation of RC-Trees completed. Second, the RC-Trees data structure is old, so we need more modern technology of programming to make the code usable for the modern applications. Third, the RC-Trees in parallel process it not efficient enough, the implementation need to make the parallel process in each rake and compress operations and between rake and compress operations. In other words, in every operation round, each operation of rake should be done in parallel, so as the operations of compress. Then the rake and compress operations themselves should be done in parallel.

REFERENCES

Goldberg A. V., & Tarjan R. E. (1988). A new approach to the maximum-flow problem. *Journal of the ACM (JACM)*, *35*(4), 921-940.

Black, P. E. (2004). Dictionary of algorithms and data structures. National Institute of Standards and Technology.

Sleator D D, Tarjan R E. (1983). A data structure for dynamic trees. Journal of Computer and System Sciences, 26(3), 362-391.

Miller G. L., & Reif J. H. (1985). Parallel tree contraction and its application (pp. 478-489). Defense Technical Information Center.

Miller G. L., & Reif J. H. (1989). Parallel Tree Contraction--Part I: Fundamentals.

Miller G. L., & Reif J. H. (1991). Parallel tree contraction part 2: further applications. SIAM Journal on Computing, 20(6), 1128-1147.

Reif J., Tate S. (1994). Dynamic Parallel Tree Contraction. Proceedings of SPAA. – 1994. – P. 114-121.

Frederickson, G. N. (1997). A data structure for dynamically maintaining rooted trees. Journal of Algorithms, 24(1), 37-65.

Alstrup, S., Holm, J., de Lichtenberg, K., & Thorup, M. (1997). Minimizing diameters of dynamic trees. In Automata, Languages and Programming (pp. 270-280). Springer Berlin Heidelberg.

Acar U., Blelloch G., Vittes J. (2003). Separating Structure from Data in Dynamic Trees. – 2003.

Acar U., Blelloch G, Harper R., Vittes J., Woo S. (2004). Dynamizing Static Algorithms with Applications to Dynamic Trees and History Independence. Proceedings of SODA. – 2004. – P. 531-540.

Acar U., Blelloch G., Vittes J. (2005). An Experimental Analysis of Change Propagation in Dynamic Trees. Workshop on Algorithms Engineering and Experiments. – 2005.

Creswell J. (2007) Review of the Literature, Chapter 2 of Research Design: Qualitative, Quantitative, and Mixed Method Approaches. Thousand Oaks: Sage Publications.

Atallah, M. J. (Ed.). (1998). Algorithms and theory of computation handbook. CRC press.

Morihata A., & Matsuzaki K. (2008). A parallel tree contraction algorithm on non-binary trees (Vol. 336). Technical Report METR 2008-27, Department of Mathematical Informatics, University of Tokyo.

Morihata A., & Matsuzaki K. (2011). A practical tree contraction algorithm for parallel skeletons on trees of unbounded degree. Procedia Computer Science, 4, 7-16.

Morihata A., & Matsuzaki K. (2014). Parallel Tree Contraction with Fewer Types of Primitive Contraction Operations and Its Application to Trees of Unbounded Degree. IPSJ Online Transactions, 7(0), 148-156.

Reid-Miller, M., Miller, G. L., & Modugno, F. (1993). List ranking and parallel tree contraction. Synthesis of Parallel Algorithms, 115-194.

Shun J., Gu Y., Blelloch G. E., Fineman J. T., & Gibbons P. B. (2015, January). Sequential random permutation, list contraction and tree contraction are highly parallel. In Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (pp. 431-448). SIAM.

Tarjan R E, Werneck R F. (2007). Dynamic trees in practice. (2007). Experimental Algorithms. Springer Berlin Heidelberg, 80-93.