# I/O-Efficient Algorithms for Graphs of Bounded Treewidth

Anil Maheshwari* Norbert Zeh*

October 19, 2000

**Abstract**

We present I/O-efficient algorithms for the single source shortest path problem and NP-hard problems on graphs of bounded treewidth. The main step in these algorithms is a method to compute a tree-decomposition for the given graph I/O-efficiently.

## 1 Problem Statement and Background

Given an undirected graph $G = (V, E)$, a *tree-decomposition* $\mathcal{D} = (\mathcal{X}, T)$ consists of a tree $T = (I, F)$ and a collection $\mathcal{X}$ of sets $\{X_i : i \in I\}$ such that (a) $\bigcup_{i \in I} X_i = V$, (b) for every edge $\{v, w\} \in E$, there is a tree node $i$ such that $v \in X_i$ and $w \in X_i$, and (c) for all $i, j, k \in I$, if $j$ lies on the path between $i$ and $k$ in $T$, then $X_i \cap X_k \subseteq X_j$. A tree-decomposition is of *width* $k$ if $|X_i| \leq k + 1$, for all $i \in I$. The *treewidth* of a graph $G$ is the minimum width over all possible tree-decompositions of $G$. A rooted tree-decomposition is *nice* if all nodes of $T$ have at most two children and the following hold: (1) If node $i$ has degree 2, and $j$ and $k$ are the children of $i$, then $X_i = X_j = X_k$; (2) if node $i$ has degree 1, and $j$ is the child of $i$, then either $X_j \subseteq X_i$ and $|X_i| = |X_j| + 1$ or $X_i \subseteq X_j$ and $|X_j| = |X_i| + 1$.

The notion of treewidth was introduced by Robertson and Seymour, where it played a central role in the theory of graph minors. Several intractable problems can be solved in polynomial or even linear time for graphs of bounded treewidth (BTW); these include the vertex cover, chromatic number, clique, independent set, and Hamiltonian cycle problems [2, 3]. Moreover, many well-known classes of graphs have BTW, including trees, partial $k$-trees, series-parallel, $k$-outerplanar, Halin, control flow graphs of goto-free programs, and chordal/interval/circular arc graphs with maximum clique size $k$, where $k$ is an integer constant.

Data sets in large applications are too massive to fit completely inside the internal memory of computers and the resulting input/output (I/O) communication between the fast internal memory and slow external memory (e.g. disks) can be a significant performance bottleneck. The challenge in the design of I/O-aware algorithms is to organize the external memory (EM) so that the "locality" present in the problem can be exploited, thereby reducing the number of block transfers from or to EM. Here we design algorithms with respect to the Parallel Disk Model, consisting of an EM containing $D$ disks attached to a machine with internal memory size $M$. Each of the $D$ disks is divided into blocks of $B$ consecutive data items. Up to $D$ blocks, at most one per disk, can be transferred between internal and external memory in a single I/O-operation. The complexity of an algorithm is the number of I/O operations it performs. A data set of size $N$ can be sorted in $sort(N) = \Theta((N/DB)\log_{M/B}(N/B))$ I/Os and scanned in $scan(N) = \Theta(N/DB)$ I/Os. See Vitter's survey [10] on EM models and algorithms.

## 2 New Results

The main result is:

THEOREM 2.1. *Given a graph $G$ of size $N$ and a constant $k$, it takes $O(sort(N))$ I/Os to test whether $G$ has treewidth at most $k$ and, if so, compute a nice tree-decomposition of width at most $k$ and size $O(N)$ for $G$.*

Our tree-decomposition algorithm uses the following results as subroutines:

- an $O(sort(|V| + |E|))$ I/O algorithm for maximal matching in an arbitrary graph $G = (V, E)$. (In [1], a deterministic $O(\frac{|E|}{|V|} sort(|V|) \log_2 \frac{|V|}{|M|})$ I/O and a randomized $O(sort(|E|)$ I/O algorithm are presented.)
- an $O(sort(N))$ I/O algorithm for computing a perfect elimination ordering for a triangulation with tree-decomposition of width at most $k$, and
- a technique for concatenating directed acyclic s-t graphs where all edges of one of the DAGs may switch direction during concatenation.

Using a new $O(scan(N + I))$ I/O algorithm for evaluating rooted trees whose vertices are sorted in preorder, where $I$ is the total amount of information sent along

*Carleton Uni., Ottawa, {maheshwa,nzeh}@scs.carleton.ca.
Supported by NSERC and NCE GEOIDE research grants.

all edges of $T$, we obtain the following result.

THEOREM 2.2. *Given a tree-decomposition $\mathcal{D} = (\mathcal{X}, T)$ of width at most $k$ and size $O(N)$ for a graph $G = (V, E)$, it takes $O(scan(N))$ I/Os to solve the following problems:*

*(i) Single source shortest path (SSSP) in directed graphs,*

*(ii) Vertex cover, chromatic number, clique, independent set, and Hamiltonian cycle.*

Ours is the first EM algorithm for computing tree-decompositions of BTW graphs.[1] For outerplanar graphs, algorithms from [8] can be used to obtain a tree-decomposition of width 2; together with our SSSP algorithm, this results in an SSSP algorithm of excellent practicality. For recent results on SSSP and related problems refer to [10, 7, 9]; determining the optimal I/O bounds for SSSP is a challenging open problem. One of the main contributions of this paper is to show that BTW graphs allow an I/O-efficient solution for the SSSP problem. Many EM algorithms for basic graph, computational geometry and matrix problems have been proposed; but we are not aware of any previous results on NP-hard problems.

## 3 Overview of our Algorithms

To compute a tree-decomposition, we follow the framework of the linear time algorithm of [4]. Using our algorithm for computing a maximal matching, we choose a set $X$ of vertices to be removed from the graph $G = (V, E)$ and recursively compute a tree-decomposition $\mathcal{D}$ for the graph $G - X$. We augment $\mathcal{D}$ to a tree-decomposition $\mathcal{D}'$ for $G$, possibly at the cost of increasing the width of the decomposition from $k$ to $2k$. Then $\mathcal{D}'$ in turn is used to derive a decomposition $\mathcal{E}$ of width at most $k$ for $G$. As we implement every recursive step in $O(sort(N))$ I/Os, and the set $X$ has size $\Omega(N)$, the I/O-complexity of our algorithm is $O(sort(N))$.

The construction of $\mathcal{E}$ as well as our algorithms for the applications in Theorem 2.2 follow a dynamic programming approach. First $\mathcal{D}'$ is transformed into a nice tree-decomposition $\mathcal{D}''$ using the perfect elimination ordering algorithm. Then $\mathcal{D}''$ is processed bottom-up, followed by a top-down phase. As all sets $X_i$ have constant size, we use the time-forward processing technique [6] to realize these phases. This approach is feasible because the vertices in $X_i$, where $i$ is a join node with children $j$ and $k$, form a constant size separator between $G_j$ and $G_k$. Hence, there is only a very limited amount of interaction between these subgraphs.

[1] This problem has a rich history and we refer the reader to Bodlaender's home page `http://www.cs.ruu.nl/~hansb` for excellent surveys.

The first phase of our construction algorithm for $\mathcal{E}$ tests whether $G$ has tree-width $k$, simulating the approach in [5]. The second phase constructs $\mathcal{E}$ from the information collected in the first phase and is non-trivial due to the non-availability of random access capabilities in EM. The concept of flippable DAGs, sketched next, is an important tool for performing this phase in $O(sort(N))$ I/Os.

Given two planar DAGs $D_1$ and $D_2$ with sources $s_1$ and $s_2$ and sinks $t_1$ and $t_2$, we may decide to construct a new DAG $D$ by putting an edge from $t_1$ to $t_2$ and flipping all edges in $D_2$. The resulting DAG $D$ may itself be involved in further concatenations of this type. If we always explicitly flip all edges in the second graph, it is easy to construct an example where $\Omega(N^2)$ edge flips are required. We present a solution involving only $O(1)$ updates per concatenation such that once all graphs have been concatenated to a graph $D'$, every edge of $D'$ is examined exactly once to decide whether or not it has to be flipped in order to obtain the final DAG.

## References


[1] J. Abello, A.L. Buchsbaum, J.R. Westbrook. A functional approach to external graph algorithms. *ESA'98*, pp. 332–343, 1998.

[2] S. Arnborg, A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial $k$-trees. *DAM*, 23:11–24, 1989.

[3] H. Bodlaender. Dynamic programming on graphs with bounded treewidth. *ICALP'88*, pp. 105–118, 1988.

[4] H. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comp., 25:1305–1317, 1996.

[5] H. Bodlaender, T. Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. J. Algorithms, 21:358–402, 1996.

[6] Y.-J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamassia, D. E. Vengroff, J. S. Vitter. External-memory graph algorithms. *SODA'95*, 1995.

[7] V. Kumar, E. Schwabe. Improved algorithms and data structures for solving graph problems in external memory. *SPDP'96*, pp. 169–176, 1996.

[8] A. Maheshwari, N. Zeh. I/O-efficient algorithms for outerplanar graphs. *ISAAC'99*, pp. 307–316, 1999.

[9] U. Meyer. External Memory BFS on Undirected Graphs with Bounded Degree. *SODA'01*, to appear, 2001.

[10] J. S. Vitter. External memory algorithms and data structures. In J. Abello, J. S. Vitter (eds.), *External Memory Algorithms and Visualization*, AMS 1999.