

Dynamic Algorithms, Spring 2008: Project

Version 1 – March 19, 2008

This project has both theoretical and practical aspects. The subproblems outlines a possible approach. If you follow the suggested approach, you must answer all subproblems to get full credit. However, you may choose a different approach, but then you should make a similarly thorough analysis and documentation to get full credit.

You are allowed to work in groups (of at most 3 students) and make a single hand-in per group. The hand-in should be done using LaTeX or similar. Handwriting is not acceptable. For figures, xfig combines nicely with LaTeX. After having made a drawing in xfig, you export it in LaTeX format, and insert the resulting code in your LaTeX file.

The evaluation of your hand-in will put weight on simple short precise answers/documentation. For the programming problems, the evaluation will put weight on

- a precise description of the purpose of the experiment
- a well motivated design of the experiment
- a clear description of the outcome of the experiment, and
- a careful analysis and discussion of the outcome compared to theoretical predictions.

Deadline for hand-in: Friday May 23rd at 12 noon, by email.

Problem 1

In this problem, we consider the dynamic minimum spanning tree problem **dynamic MST**:

- `init(n , weights)`. G becomes the complete graph with n vertices $\{0, \dots, n-1\}$ and the $\binom{n}{2}$ edges have weights as specified in “weights”.

- `change(i, j, w)`. Change the weight of edge (i, j) to w .
- `mst?` Return the total weight of a minimum weight spanning tree for G .

Note that for simplicity, we avoid an i/o expensive listing of a complete minimum weight spanning tree. Instead our query only returns the total weight of such a tree.

In the course notes you have encountered a quite sophisticated data structure that allows a solution taking amortized time $O(\log^2 n)$ per operation. In this project, we will look at a much simpler solution that may achieve time $O(n \log n)$ per operation. This is still much better than the $O(n^2)$ bound obtainable by an offline based approach, and therefore potentially practical. The simpler solution is based on the sparsification technique that applies to a wide range of dynamic graph problems. Sparsification is due to Eppstein et al., *Sparsification - a technique for speeding up dynamic graph algorithms*, [<http://doi.acm.org/10.1145/265910.265914>]. Given an off-line algorithm of time complexity $f(m)$ where m is the number of edges in the graph, sparsification will sometimes allow the construction of a dynamic algorithm taking time close to $f(n)$, where n is the number of nodes in the graph.

Sparsification consists in maintaining a balanced binary tree T , where each leaf corresponds to an edge in the current graph G . This leads naturally to each internal node α of the binary tree being in correspondence with a subgraph G_α of G (in that G_α contains the edges associated with the leaves in the subtree rooted at α). Each internal node α maintains a sparse data structure holding some properties of the subgraph G_α . The term sparse means in the present context that the data structure at node α should have size $O(n)$ even if there are close to n^2 edges in G_α . The challenge is to maintain the data structures under change operations. Also the data structure at the tree root must support the relevant queries.

In order to apply sparsification to **dynamic MST** we need a definition for the off-line problem of computing a minimum spanning forest (**off-line MSF**)

- **input**. n and E , where the nodes of the graph are $\{0, \dots, n-1\}$ and each of the m edges in E is described as a triple (i, j, w) , where i and j are the end nodes and w is the integer weight of the edge.

- **output.** A list of the at most $n - 1$ edges in a minimum spanning forest $F \subseteq E$ for the input graph.

From earlier courses (dADS2 or similar) you know various solutions for **off-line MSF** of time complexity at most $O(m \log n)$, when you assume that the integer weights on edges can be stored in a single computer word, and that weights can be compared in time $O(1)$. Let \mathcal{A} denote any such solution for **off-line MSF**. This leads to the following sparsification based solution for **dynamic MST**:

- We maintain a balanced binary tree T , where each leaf has associated an edge e of G in a one to one relation. Each internal node α of T will maintain a minimum spanning forest F_α of the associated graph G_α . α will also maintain the weight of F_α .
- In an **init** operation, T may be constructed bottom up, such that for an internal node α with sons β and γ , one may compute F_α from $F_\beta \cup F_\gamma$ using the off-line algorithm \mathcal{A} .
- In a **change** operation, T need be updated on a path from a single leaf to the root only. This can be done just as in the **init** case. Note that the tree structure is not changed and no rebalancing is needed.

Subproblem 1.1. In the above dynamic algorithm, we are implicitly assuming that $F_\alpha \subseteq F_\beta \cup F_\gamma$. Explain why this assumption is legal.

Subproblem 1.2. Argue that **init**, **change** and **query** may be implemented in time $O(n^2 \log^2 n)$, $O(n \log^2 n)$ and $O(1)$ respectively, when assuming only that \mathcal{A} solves **off-line MSF** in time $O(m \log n)$.

The minimal requirement for your project hand-in

- Make two implementations of **dynamic MST**. One implementation should be based directly on a solution \mathcal{A} for **off-line MSF** in that almost nothing is done on an update, and an MST is computed from scratch on a query. The other solution should be at least as efficient as the above sketched sparsification based solution.

Apart from coding these algorithms you must describe them in pseudo code and give time complexities of the operations. You must explain

enough details of the implementations to convince the reader that you have eliminated obvious sources of inefficiency. As an example of the latter, don't use red-black trees for the sparsification tree, since you don't need rebalancing. An array based representation of the sparsification tree with sons of $a[i]$ in $a[2i]$ and $a[2i+1]$ is likely to be much more efficient.

- You must argue for correctness of your implementations. Be careful! It may be insufficient to compare results of the two algorithms. If you have used the same off-line MST algorithm as a subroutine in both dynamic algorithms, then a single implementation error in that algorithm, may make both dynamic algorithms give identical wrong answers. You may overcome this problem by implementing an extra independent MST algorithm. The textbook Cormen, Leiserson, Rivest, Stein, *Introduction to algorithms*, MIT Press 2001, presents 2 distinct off-line MST algorithms. You may also compare to a library implementation.

Alternatively, one may use an algorithm that verifies correctness of a proposed MST. The paper Blum, Kannan, *Designing programs that check their work*, [<http://doi.acm.org/10.1145/200836.200880>] discusses the general problem of verifying that output is correct. It is actually known how to verify an MST in time $O(m)$. For completeness a reference is included: King, *A Simpler Minimum Spanning Tree Verification Algorithm*, [<http://dx.doi.org/10.1007/BF02526037>], though the algorithm is too complicated for implementation within this course project.

- You should make an experiment to compare the two implementations. You may find a test data set at

<http://www.daimi.au.dk/~gudmund/dynamicF08/mst/>

The file `init` contains $\binom{n}{2}$ lines of the form (i, j, w) for setting up a graph with $n = 1000$ nodes, where edge (i, j) has weight w . The file `change` contains 10^6 lines of the form (i, j, w) that similarly specifies 10^6 change operations. Ideally, you should run both algorithms on all data, i.e. set up the graph and make 10^6 change operations with queries after each change. However, to ensure moderate time usage, perform experiments for the first 10^i changes only, where you let i grow by one $i = 0, 1, 2, \dots$, and stop one or both algorithms, if time consumption becomes excessive.

Present the results of your experiment. That includes the result returned by the query (the total weight of an MST) after the first 10^i changes for each i . You should make tables and graphs to compare time performance of the 2 algorithms on the data set. Make comments on the performance (are there surprises?)

To get full credit for your project you must do more than the minimal requirement, namely

- Your fast dynamic MST must have time complexity at most $O(n \log n)$ for **change**.
- Your fast dynamic MST must have time complexity at most $O(n^2 \alpha(n))$ for **init**.

For these tasks you may follow the suggested approach below or you may use different techniques and algorithms, but the bounds on time complexities should be analysed properly in any case.

Suggested approach

How to speed up the **change** operation. Note that in the sparsification based algorithm, we apply the off-line algorithm A in a very special case, namely to the union of two spanning forests. We will argue that it is possible to recompute F_α from F_β and F_γ in time $O(n)$ when implementing a **change** operation.

Subproblem 1.3. Look at the case, when the weight of the edge e in a single leaf of the sparsification tree is changed. Let α be a node on the path from this leaf to the root. Let F_α and F'_α be the MSF at the node before and after the change of the weight of e . Argue that $|F_\alpha| = |F'_\alpha|$ and if $F_\alpha \neq F'_\alpha$ then either $\{e\} = F_\alpha \setminus F'_\alpha$ or $\{e\} = F'_\alpha \setminus F_\alpha$.

Subproblem 1.4. Argue how to determine if you are in the case $\{e\} = F'_\alpha \setminus F_\alpha$ and find $\{f\} = F_\alpha \setminus F'_\alpha$. Hint: use depth first search on F_α .

Subproblem 1.5. Argue how to determine if you are in the case $\{e\} = F_\alpha \setminus F'_\alpha$ and find $\{f\} = F'_\alpha \setminus F_\alpha$: Hint: tentatively remove e from F_α , mark the resulting two components differently and look for a replacement edge.

How to speed up the `init` operation. Consider using Kruskals algorithm for computing F_α from $F_\beta \cup F_\gamma$. The bottleneck is sorting of the edges, which we don't need to do from scratch.

Subproblem 1.6. Devise a strategy so you only spend time $O(n\alpha(n))$, when computing F_α from $F_\beta \cup F_\gamma$. (Hint: transfer more information from β and γ than just the MSF's)

We need not restrict our selves to have single edges at leaves of the sparsification tree. A leaf may have associated a subgraph with $O(n)$ edges.

Subproblem 1.7. Devise a simple way of distributing the $\binom{n}{2}$ edges of G evenly among $O(n)$ leaves, while taking the result of the previous subproblem into account.

Subproblem 1.8. Combine all your subresults and analyse the resulting implementation of the `init` operation.