Updating Directed Minimum Cost Spanning Trees

Orestis A. Telelis

joint work with
Gerasimos G. Pollatos Vassilis Zissimopoulos

Department of Informatics and Telecommunications
University of Athens, Hellas (Greece)

Workshop on Experimental Algorithms, 2006

Outline

Introduction

A DMST is a min-cost subset of (directed) edges $T \subseteq E$ such that:

1. T is acyclic
2. $(u, v), (x, y) \in T \Rightarrow v \neq y$
3. T is maximal with respect to 1 and 2.

Branchings vs. Trees

- G strongly connected $\Rightarrow$ a DMST T is a tree of $n - 1$ edges.
- Otherwise T may be forest of directed trees called branching.
- We can always augment G with infinite cost edges to make it strongly connected.

Algorithms

For a digraph $G(V, E)$, $|V| = n$, $|E| = m$:

1. Same algorithm (*Edmonds' algorithm*) independently by:
   Edmonds 1967, Chu& Liu 1965, Bock 1971: $O(mn)$
2. Best known implementation for general digraphs:
   Gabow, Galil, Spencer, Tarjan 1986: $O(m + n \log n)$
3. Best known implementation for sparse digraphs ($O(n)$ edges):
   Mendelson, Tarjan, Thorup, Zwick, 2004: $O(m \log \log n)$

Dynamic Updates

Efficiently updating the DMST $T \subseteq E$ of a digraph $G(V, E)$ when:

1. An edge is deleted from the graph.
2. An edge is inserted to the graph.

... in less time than re-evaluating it from scratch.

The dynamic complexity of DMST is not known.

For MST see Holm, de Lichtenberg, Thorup, JACM 2001

Is it Possible?

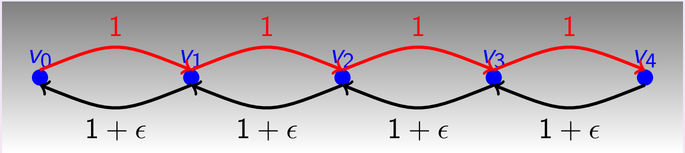
DMST on a Directed Acyclic Graph:

For each $v \in V$ choose the min-cost incoming edge.
Cycles will not occur.

Verification Time on a DAG

- $cost(T)$ minimum iff $\forall(u, v) \in T$ min-cost edge entering v.
- This amounts testing $O(m)$ inequalities.
- $\Omega(m)$ time is required (Rabin 1972).

A weak lower bound

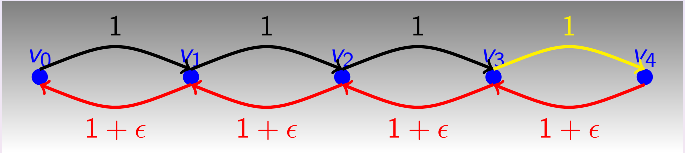


Sensitivity to edge operations

Removal of (v_3, v_4) from the digraph and the DMST...

Theorem

A weak lower bound



Sensitivity to edge operations

... causes the DMST to change entirely! At least verification time is required to update: $\Omega(n^2)$ for dense digraphs.

Theorem

A dynamic algorithm retaining and using only the DMST itself requires $\Omega(m)$ time, which is $\Omega(n^2)$ for dense digraphs.

Note: $O(\log n)$ Update Time on DAGs

- For each vertex v store its incoming edges in a FHeap $H(v)$.
- Deletion of $(u, v) \in E$ from $G(V, E)$:
    1. $(u, v) \in T$: *extract* least-cost e from $H(v)$ to replace (u, v).
    2. $(u, v) \notin T$: *decrease-cost* to $-\infty$ of (u, v) in $H(v)$ and extract (u, v) from $H(v)$.
- Insertion of (u, v) to E:
    1. $\exists (x, v) \in T$ with $cost(x, v) \leq cost(u, v)$: *insert* (u, v) in $H(v)$.
    2. $\exists (x, v) \in T$ with $cost(x, v) > cost(u, v)$: replace (x, v) with (u, v) and *insert* (x, v) in $H(v)$.

An algorithm for dynamic updates

A *dynamization* of Edmonds' algorithm:

- Encode output into an Augmented Tree (ATree) structure.
- Modify the data structure per edge operation.
- Identify maximal portion of *unaffected* output to maintain.
- Execute a modified implementation to augment the maintained output.

Edmonds' Algorithm

1. Set $v = v_0$.
2. Repeat until a single vertex remains:
   1. Select the min-cost edge entering v, say (u, v).
   2. For each edge (x, v) set $cost(x, v) \leftarrow cost(x, v) - cost(u, v)$.
   3. Contract dicycle (if any) due to (u, v), to vertex u.
   4. Set $v = u$.
3. Remove redundant selected edges.

Edmonds' Algorithm

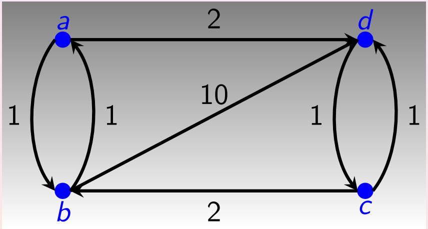
1. Set $v = v_0$.
2. Repeat until a single vertex remains: [$O(m + n \log n)$ Time]
   1. Select the min-cost edge entering v, say (u, v).
   2. For each edge (x, v) set $cost(x, v) \leftarrow cost(x, v) - cost(u, v)$.
   3. Contract dicycle (if any) due to (u, v), to vertex u.
   4. Set $v = u$.
3. Remove redundant selected edges. [$O(n)$ Time]

We *dynamize* the loop

Example on Edmonds' Algorithm

Next Step

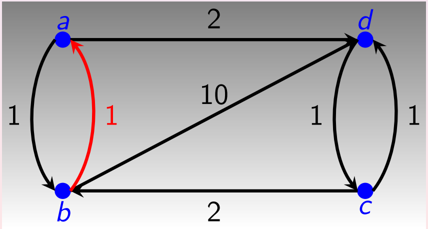
Take the least cost edge entering vertex a.
That is edge (b, a).



Example on Edmonds' Algorithm

Next Step

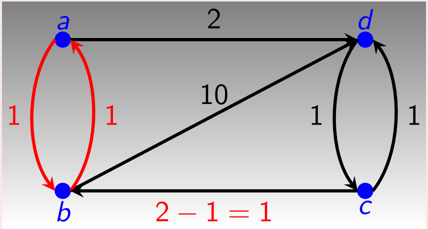
Take the least cost edge entering vertex b.
That is edge (a, b).
Reduce the cost of (c, b) to $2 - cost(a, b) = 2 - 1 = 1$.



Example on Edmonds' Algorithm

Next Step

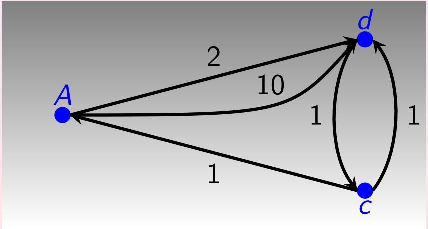
Contract cycle $a \to b \to a$ into a vertex A.



Example on Edmonds' Algorithm

Next Step

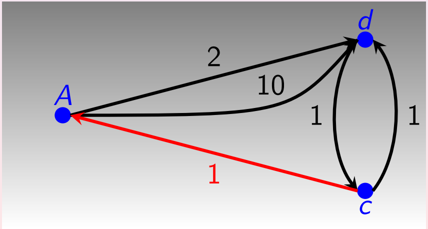
Take the least cost edge entering vertex A.
That is edge (c, A) (which was edge (c, b)).



Example on Edmonds' Algorithm

Next Step

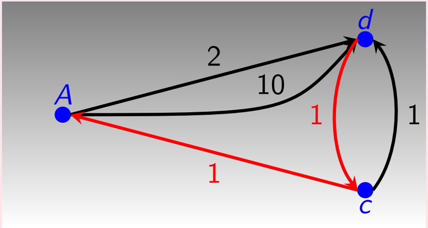
Take the least cost edge entering c.
That is edge (d, c).



Example on Edmonds' Algorithm

Next Step

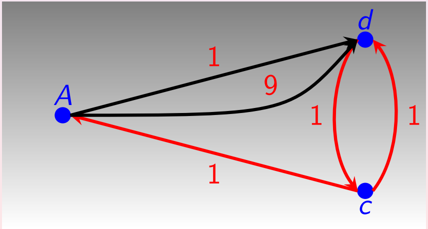
Take the least cost edge entering d.
That is edge (c, d).
Reduce the cost of (a, d) to $2 - cost(c, d) = 2 - 1 = 1$.
Reduce the cost of (b, d) to $10 - cost(c, d) = 10 - 1 = 9$.



Example on Edmonds' Algorithm

Next Step

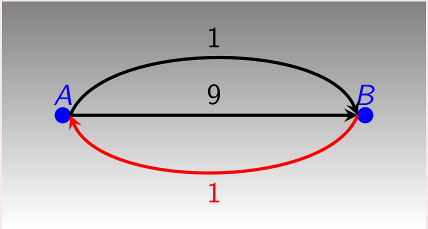
Contract cycle $c \rightarrow d \rightarrow c$ into a vertex B.



Example on Edmonds' Algorithm

Next Step

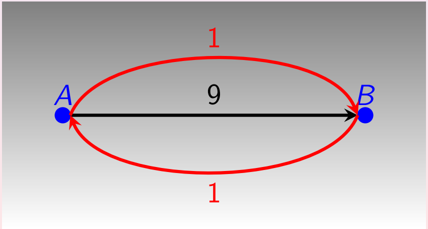
Take the least cost edge entering B.
That is edge (A, B) (which was edge (a, d)).



Example on Edmonds' Algorithm

Next Step

Contract cycle $A \rightarrow B \rightarrow A$ into a vertex C.

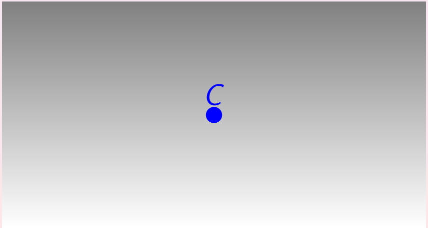


Example on Edmonds' Algorithm

Next Step

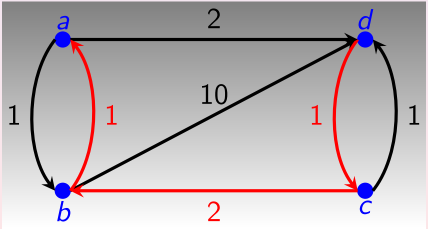
End of loop.

Removal of redundant edges produces the DMST.



Example on Edmonds' Algorithm

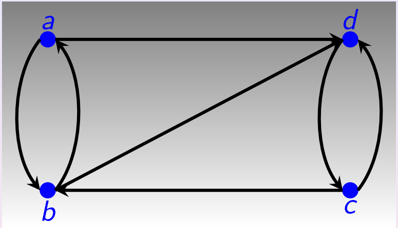
Next Step



Encode the output of the loop into a hierarchical structure (ATree):

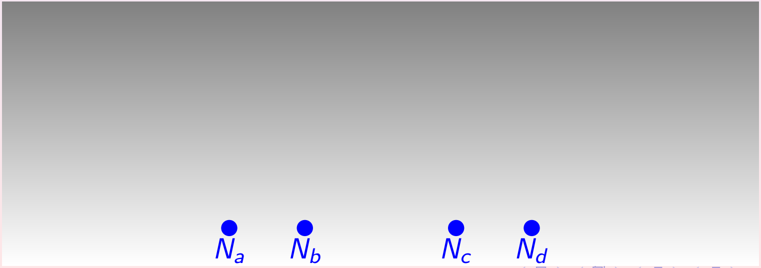
1. *Leaf Nodes*: one for each vertex of the digraph.
2. *Internal Nodes*: Represent vertices emerging from contractions.
3. Each node is labelled with the edge selected for the represented vertex.
4. Each internal node has an associated list of edges absorbed due to contraction.

Bottom-Up Construction of an ATree (Example)

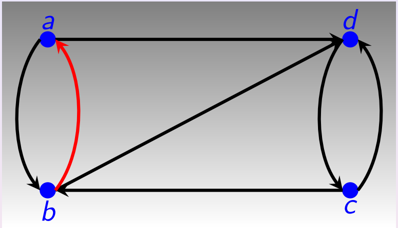


Next Step

Label N_a with (b, a).

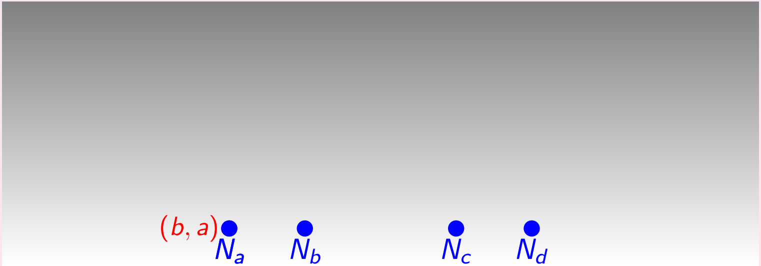


Bottom-Up Construction of an ATree (Example)

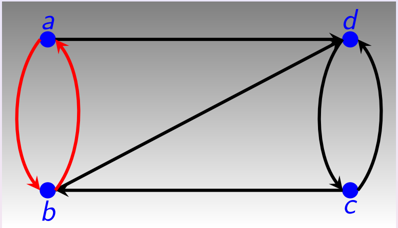


Next Step

Label N_b with (a, b).

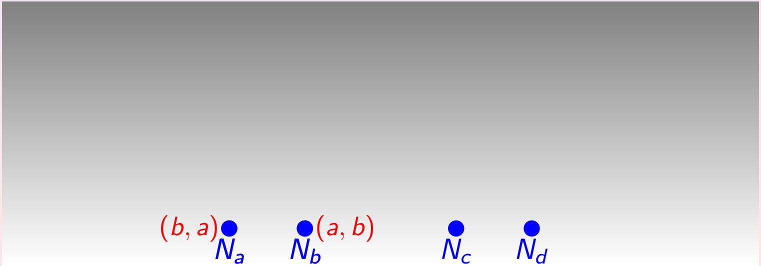


Bottom-Up Construction of an ATree (Example)

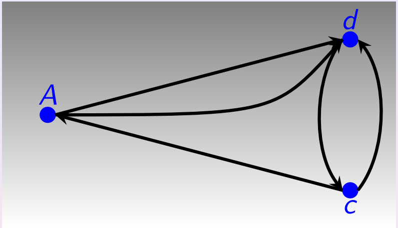


Next Step

Create N_A, parent of N_a, N_b.

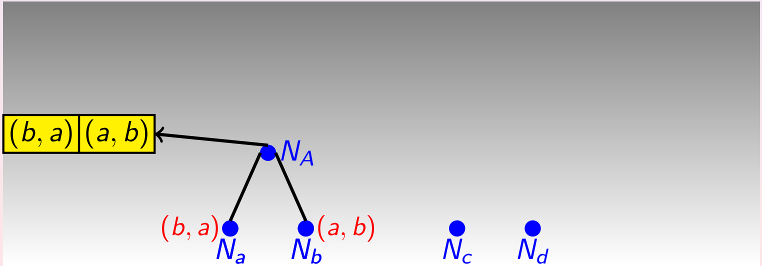


Bottom-Up Construction of an ATree (Example)

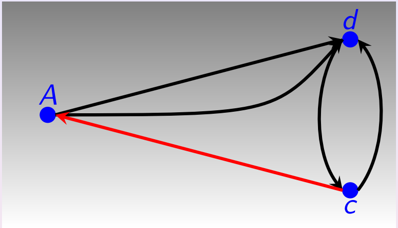


Next Step

Label N_A with (c, b).

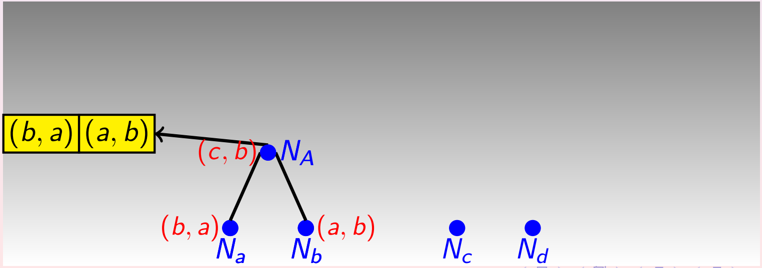


Bottom-Up Construction of an ATree (Example)

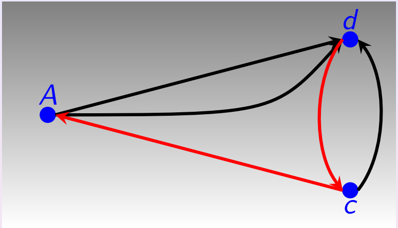


Next Step

Label c with (d, c).

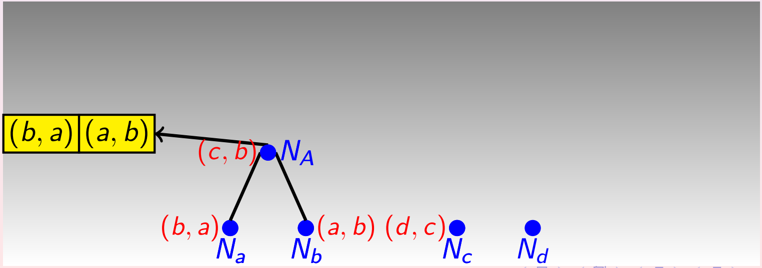


Bottom-Up Construction of an ATree (Example)

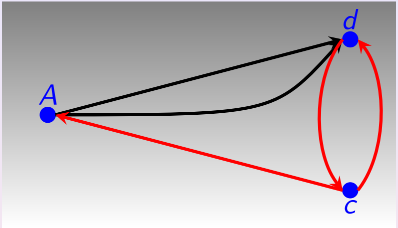


Next Step

Label d with (c, d).

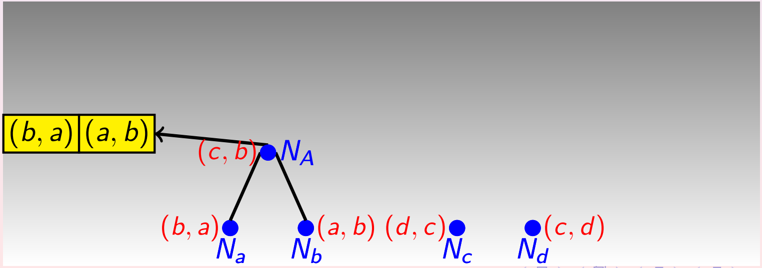


Bottom-Up Construction of an ATree (Example)

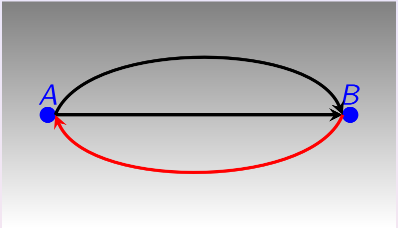


Next Step

Create N_B, parent of N_c, N_d.

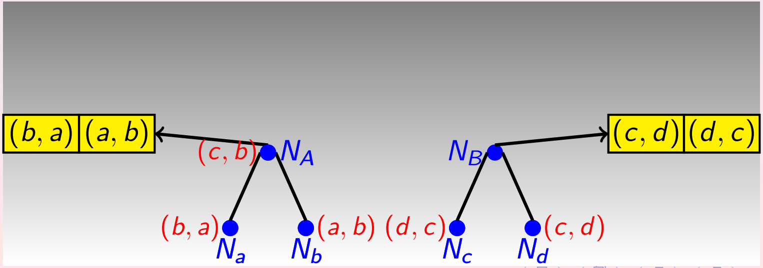


Bottom-Up Construction of an ATree (Example)

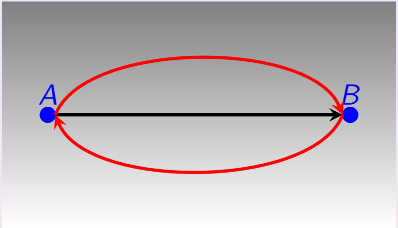


Next Step

Label N_B with (a, d).

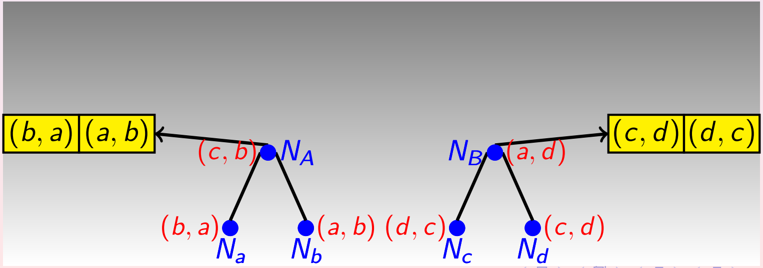


Bottom-Up Construction of an ATree (Example)

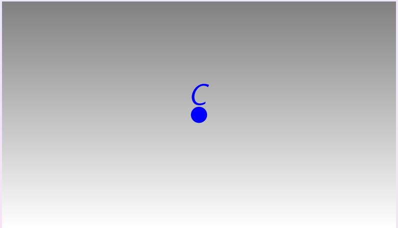


Next Step

Create N_C, parent of N_A, N_B.

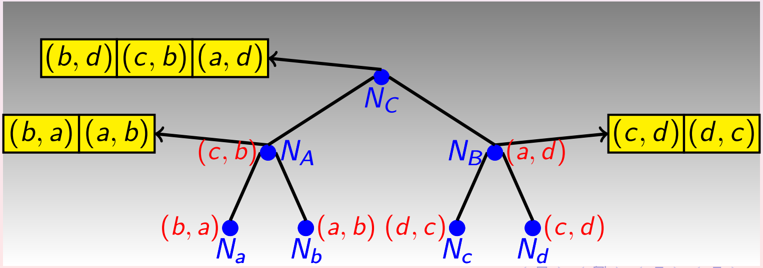


Bottom-Up Construction of an ATree (Example)



Next Step

The *labelset* $\mathcal{L}$ of the ATree is the set of edges label its nodes.

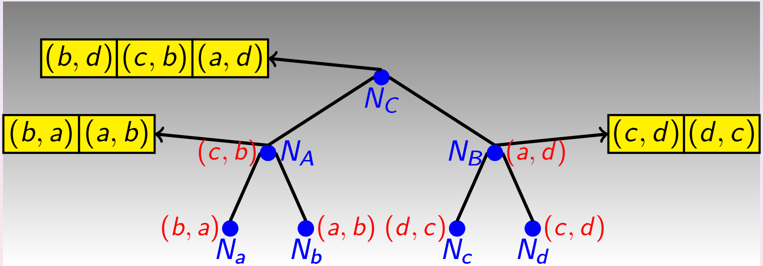


Deleting an edge

Let e be the deleted edge. Two cases must be considered:

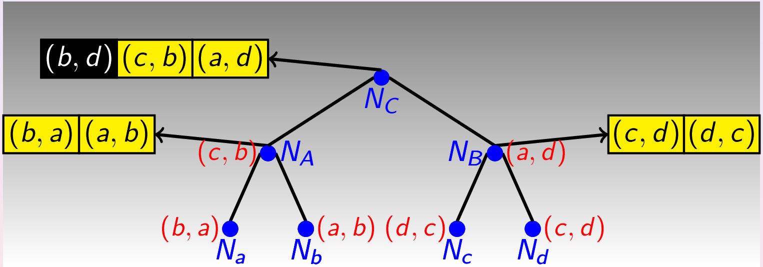
- $e \notin \mathcal{L}$: Update the ATree and Stop.
- $e \in \mathcal{L}$:
  1. Maintain a maximal portion of the ATree, unaffected by the deletion.
  2. This represents a maximally contracted digraph G'.
  3. Initialize and execute Edmonds' algorithm on G'.

Edge Deletion Example



Case 1: Deleting (b, d) not in the labelset $\mathcal{L}$

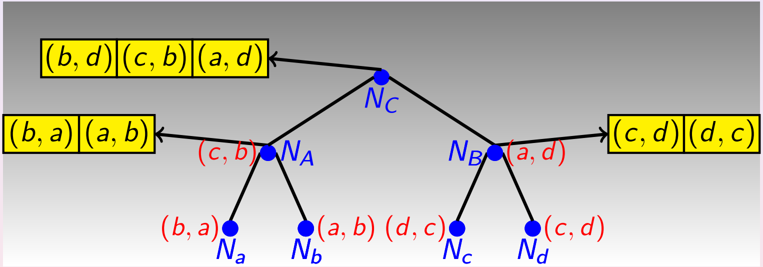
Edge Deletion Example



Case 1: Deleting (b, d) not in the labelset $\mathcal{L}$

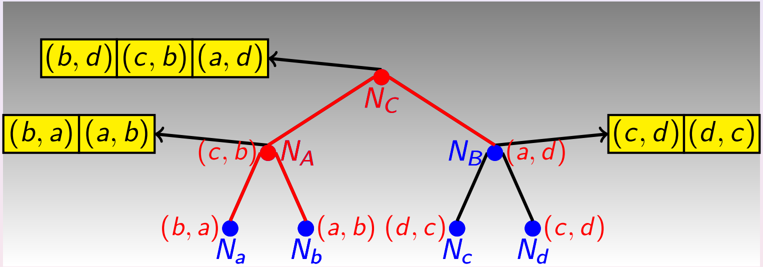
Remove (b, d) from the list containing it.

Edge Deletion Example



Case 2: Deleting (a, b) from the labelset $\mathcal{L}$

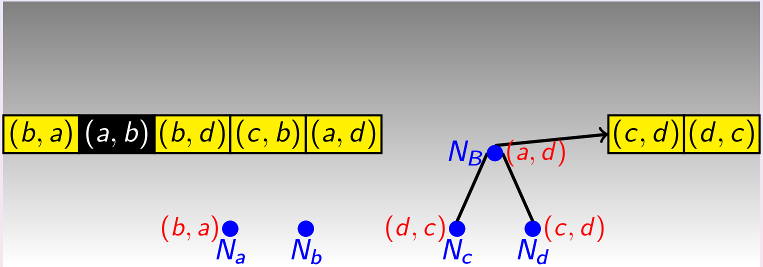
Edge Deletion Example



Case 2: Deleting (a, b) from the labelset $\mathcal{L}$

Remove the path from N_b to the root, N_C, and unite the orphaned associated lists.

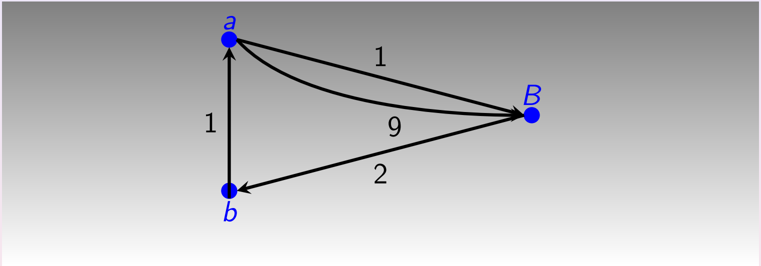
Edge Deletion Example



Case 2: Deleting (a, b) from the labelset $\mathcal{L}$

- Roots $\{N_a, N_b, N_B\}$ are vertex set of maintained contracted digraph.
- List union of removed nodes (but (a, b)) is the edge set.

Edge Deletion Example



A maximally contracted digraph unaffected by deletion

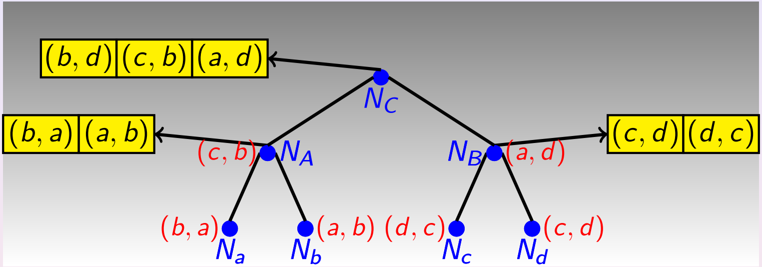
- Cost assignment on edges can be done in $O(n)$ time.
- Execute Edmonds' algorithm on this digraph.

Inserting an Edge

Two cases must be considered:

- $cost(T)$ is not affected: update the ATree and stop.
- The newly inserted edge reduces $cost(T)$.
  1. e substitues some $e' \in \mathcal{L}$.
  2. Engage *virtual* deletion of e'.
  3. Initialize and execute Edmonds' algorithm on $G'(V', E' \cup \{e\})$.

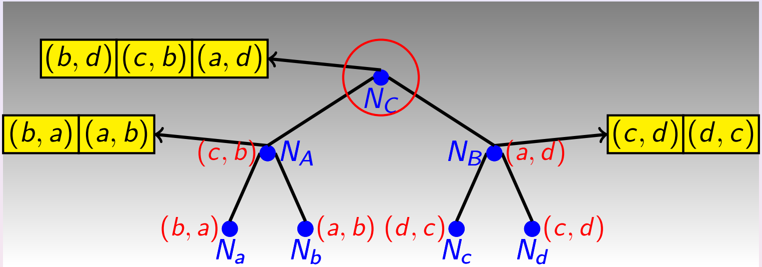
Edge Insertion Example



Insertion of edge (a, c) at cost 10.

1. Find *Least Common Ancestor* (LCA) of N_a and N_c.

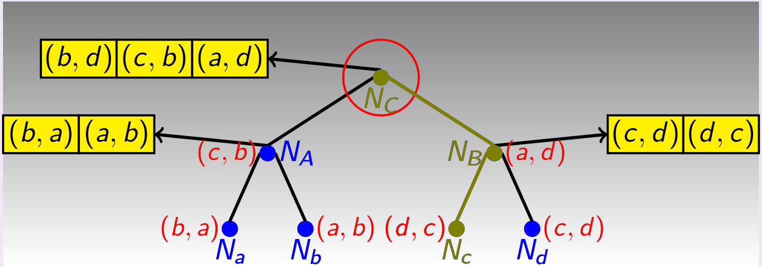
Edge Insertion Example



Insertion of edge (a, c) at cost 10.

1.
2. Take the subtree rooted at the found LCA (N_C).

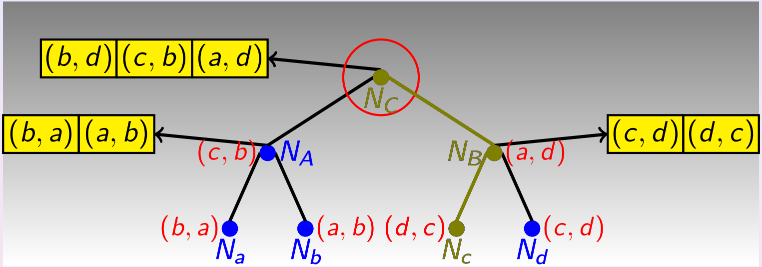
Edge Insertion Example



Insertion of edge (a, c) at cost 10.

1.
2.
3. Follow the path $N_c \cdots N_C$, comparing $cost(a, c)$ with the cost of the labelling edges of the nodes in the order visited.

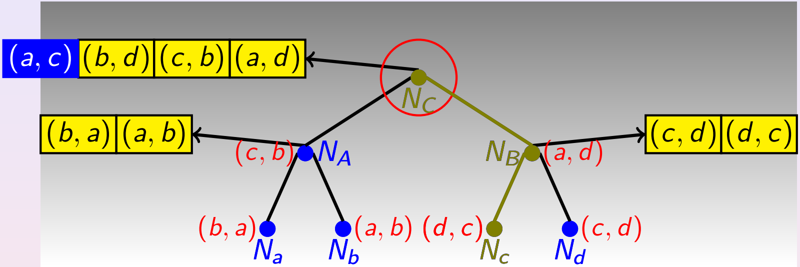
Edge Insertion Example



Case 1: (a, c) costs less than (d, c) or than (a, d).

1. Engage a *virtual* deletion of (d, c) or (a, d) respectively.

Edge Insertion Example



Case 2: (a, c) cannot replace an edge from labelset

1.
2. Insert (a, c) into the list of the LCA

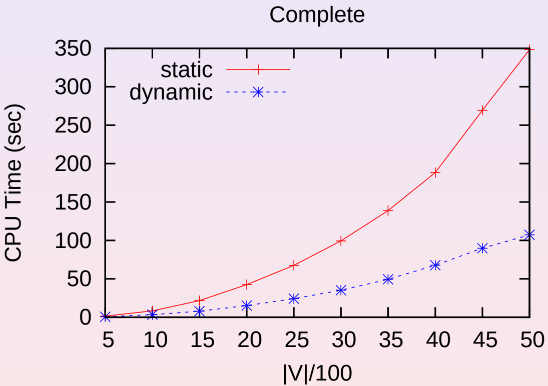
Technical Details

- All operations on ATree structure take $O(n)$ time.
- Implementation relies on ideas of Gabow et al., 1986.
- For sparse digraphs use heaps of Mendelson et al. 2004.
- Output Complexity (in terms of minimal output subset needing update):
  - ρ denotes vertex set of maximally contracted digraph.
  - Can be shown to be $O(n + ||\rho|| + |\rho| \log |\rho|)$.
  - $||\rho||$ denotes edge set entering vertices of $\rho|$.
  - Worst case equal to complexity of re-evaluation.

Experimental Evaluation

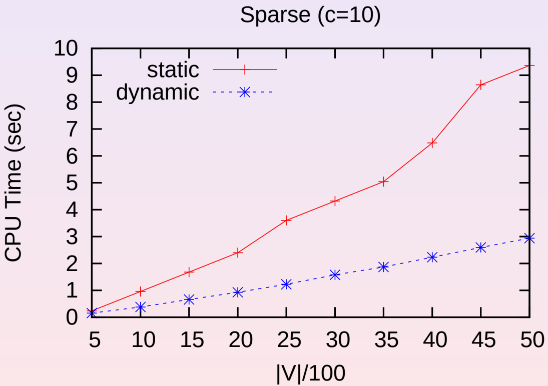
- Random Digraphs of orders $500 - 5000$, step 500.
  1. Dense Digraphs: $d \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$.
  2. Sparse Digraphs: $d = \frac{c}{n-1}$, $c \in \{10, 20, 30, 40, 50\}$.
- Sparse digraphs with embedded clique of increasing order.
- Evaluating updates in comparison to static DMST re-evaluation.
- Average time over 10^4 deletions/insertions decided with 0.5.

Performance on Complete Digraphs



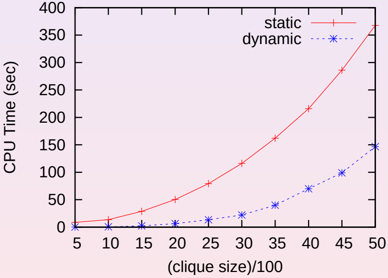
Over 60% relative gain in execution time in dense digraphs.

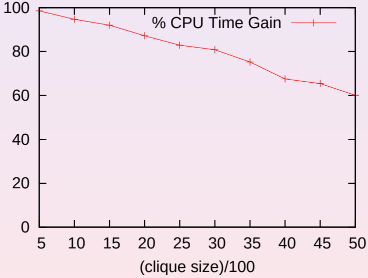
Performance on Sparse Digraphs



Increasing gain with digraph order in sparse digraphs: 60% to 80%

Embedded Cliques





As the clique grows the performance falls from a 95% to a 60%.

Conclusions

- A (weak) lower bound on the complexity of DMST updates.
- A dynamic algorithm studied in terms of output complexity and experimentally.
- Achieving a factor > 2 speedup on dense digraphs.
- Achieving huge speedup on sparse digraphs:
  - For DAGs our scheme reduces to the $O(\log n)$ update time
  - What about average case complexity for sparse digraphs?
- The complexity of updates is still open.

Platform of Experimentation

- Standard C++ implementation, gcc 3.2
- P4 2.6 GHz, 512 MB Main memory
- Linux Kernel 2.6