The Shortest Path problem in graphs with selfish edges

Review

- VCG-mechanism: pair $M=\langle g,p\rangle$ where
  - $g(r) = \arg\max_{y\in X} \sum_i v_i(r_i,y)$
  - $p_i(g(r)) = -\sum_{j\neq i} v_j(r_j,g(r_{-i})) + \sum_{j\neq i} v_j(r_j,g(r))$
- VCG-mechanisms are truthful for utilitarian problems (i.e., problems in which the SCF is given by the sum of players' valuation functions)

Buying a path in a network

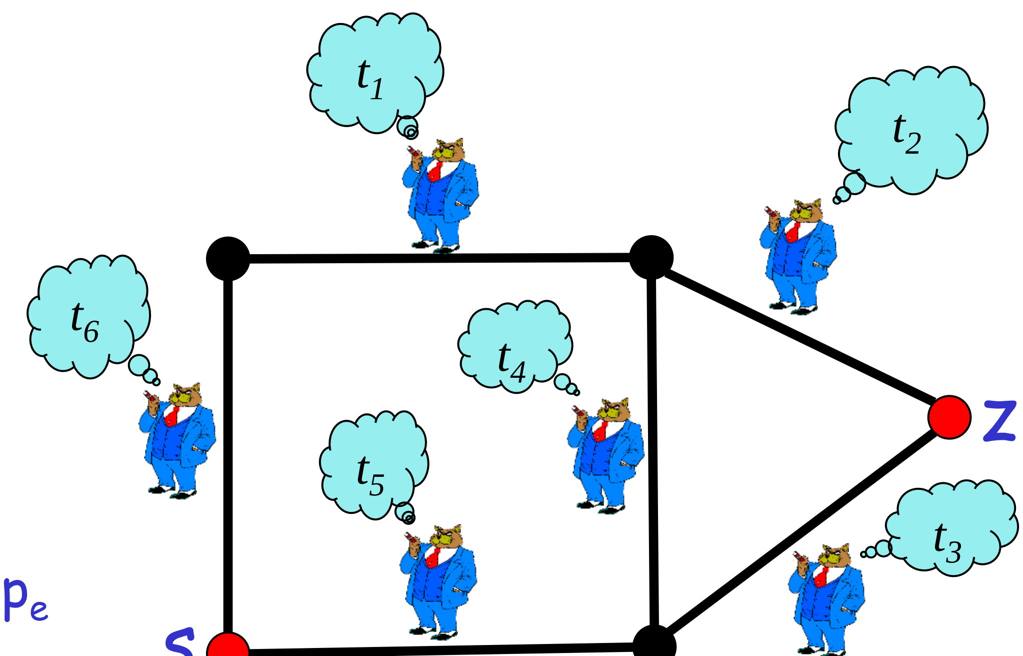
$f(t,x)$:
The length of a path w.r.t. the true edge costs

X: set of all paths between s and z

Mechanism

decides the path and the payments

t_e: cost of edge e



if edge e is selected
and receives a payment of p_e
e's utility:

$$p_e - t_e$$

The private-edge SP problem

- **Given**: an undirected graph $G=(V,E)$ such that each edge is owned by a distinct player, a source node s and a destination node z; we assume that a player's private type is the positive cost (length) of the edge, and her valuation function is equal to her negated type if edge is selected in the solution, and 0 otherwise.
- **Question**: design a truthful mechanism in order to find a shortest path in $G_t=(V,E,t)$ between s and z.

Notation and assumptions

- $n=|V|$, $m=|E|$
- $d_G(s,z)$: distance in $G=(V,E,r)$ between s ans z (sum of reported costs of edges on a shortest path $P_G(s,z)$ in G)
- Nodes s and z are 2-edge-connected in G, i.e., there exists in G at least 2 edge-disjoint paths between s and z $\Rightarrow$ for any edge of $P_G(s,z)$ removed from the graph there exists at least one replacement path in G-e between s and z (this will bound the problem, since otherwise a bridge-edge might have an unbounded marginal utility)

VCG mechanism

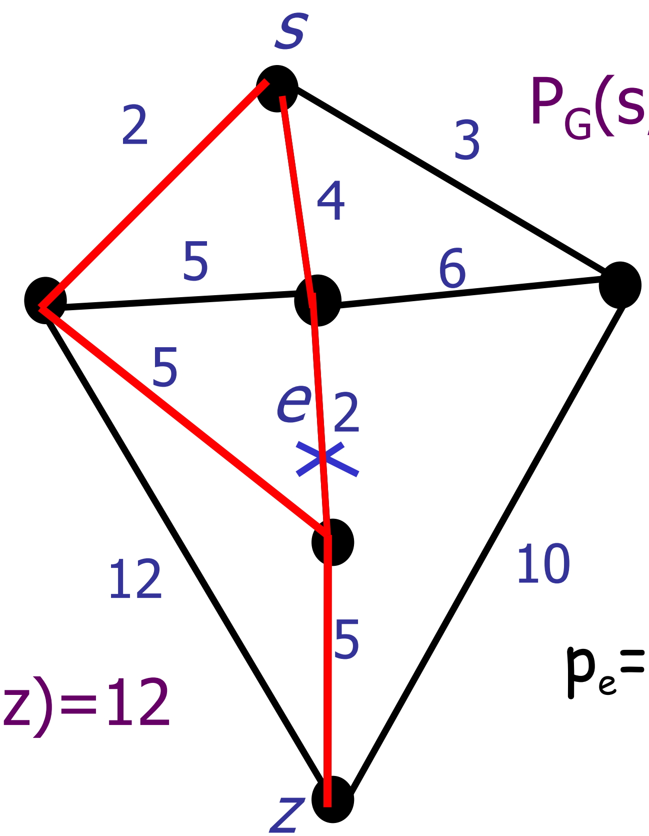
- The problem is utilitarian (indeed, the (negated) cost of a solution is given by the sum of valuations) $\Rightarrow$ **VCG-mechanism** $M=\langle g,p\rangle$:
  - g: computes $\arg\max_{y\in X} \sum_{e\in E} v_e(r(e),y)$, i.e., $P_G(s,z)$ in $G=(V,E,r)$, where $r(e)$ denotes the reported cost of e; indeed, valuation functions are negative, so maximizing their sum means to compute a cheapest path;
  - p (Clarke payments): for each $e\in E$:

$$p_e = \underbrace{-\sum_{j\neq e} v_j(r(j),g(r_{-e}))}\ \underbrace{+\sum_{j\neq e} v_j(r(j),g(r))}, \text{ namely}$$

$$p_e=\begin{cases} d_{G-e}(s,z)-[d_G(s,z)-r(e)] = d_{G-e}(s,z)-d_G(s,z) + r(e) & \text{if } e\in P_G(s,z) \\ d_G(s,z)-d_G(s,z) = 0 & \text{otherwise} \end{cases}$$

$\Rightarrow$ For each $e\in P_G(s,z)$, we have to compute $d_{G-e}(s,z)$, namely the length of a shortest path in $G-e=(V,E\setminus\{e\},r_{-e})$ between s and z.

The replacement shortest path



$P_G(s,z) \Rightarrow d_G(s,z)=11$

$P_{G-e}(s,z) \Rightarrow d_{G-e}(s,z)=12$

$p_e=d_{G-e}(s,z)-d_G(s,z) + r(e) =$
$12-11+2=3$

Remark: $u_e = p_e+v_e= p_e- t_e = p_e- r(e) = d_{G-e}(s,z)-d_G(s,z)+ \cancel{r(e)} - \cancel{r(e)}$, and since $d_{G-e}(s,z) \geq d_G(s,z) \Rightarrow u_e \geq 0$

A trivial but costly implementation

- Step 1: First of all, apply Dijkstra to compute $P_G(s,z)$ $\Rightarrow$ this costs $O(m + n \log n)$ time by using Fibonacci heaps.
- Step 2: Then, $\forall e \in P_G(s,z)$ apply Dijkstra in G-e to compute $P_{G-e}(s,z)$ $\Rightarrow$ we spend $O(m + n \log n)$ time for each of the $O(n)$ edges in $P_G(s,z)$, i.e., $O(mn + n^2 \log n)$ time

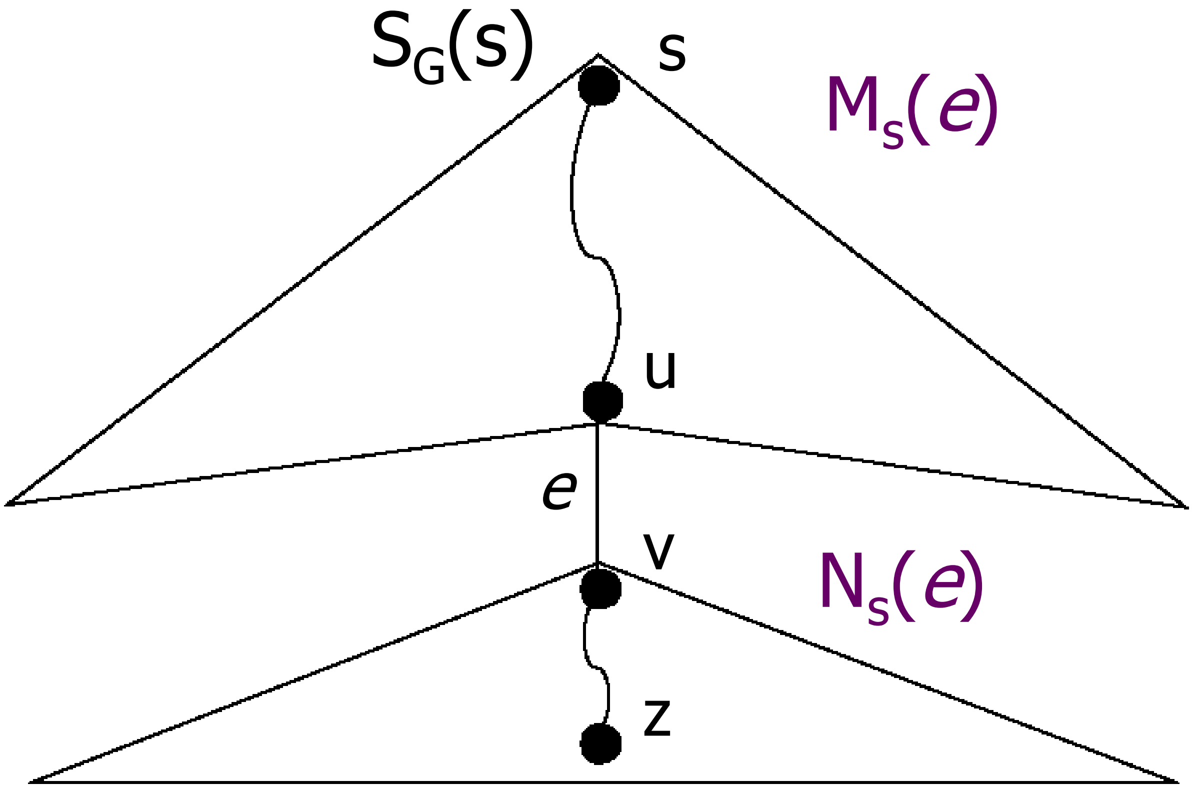
$\Rightarrow$Overall complexity: $O(mn + n^2 \log n)$ time

- We will see an efficient solution costing $O(m + n \log n)$ time

Notation

- $S_G(s)$, $S_G(z)$: single-source shortest-path trees rooted at s and z
- $M_s(e)$: set of nodes in $S_G(s)$ not descending from edge e (i.e., the set of nodes whose shortest path from s does not use e)
- $N_s(e)=V/M_s(e)$
- $M_z(e)$, $N_z(e)$ defined analogously

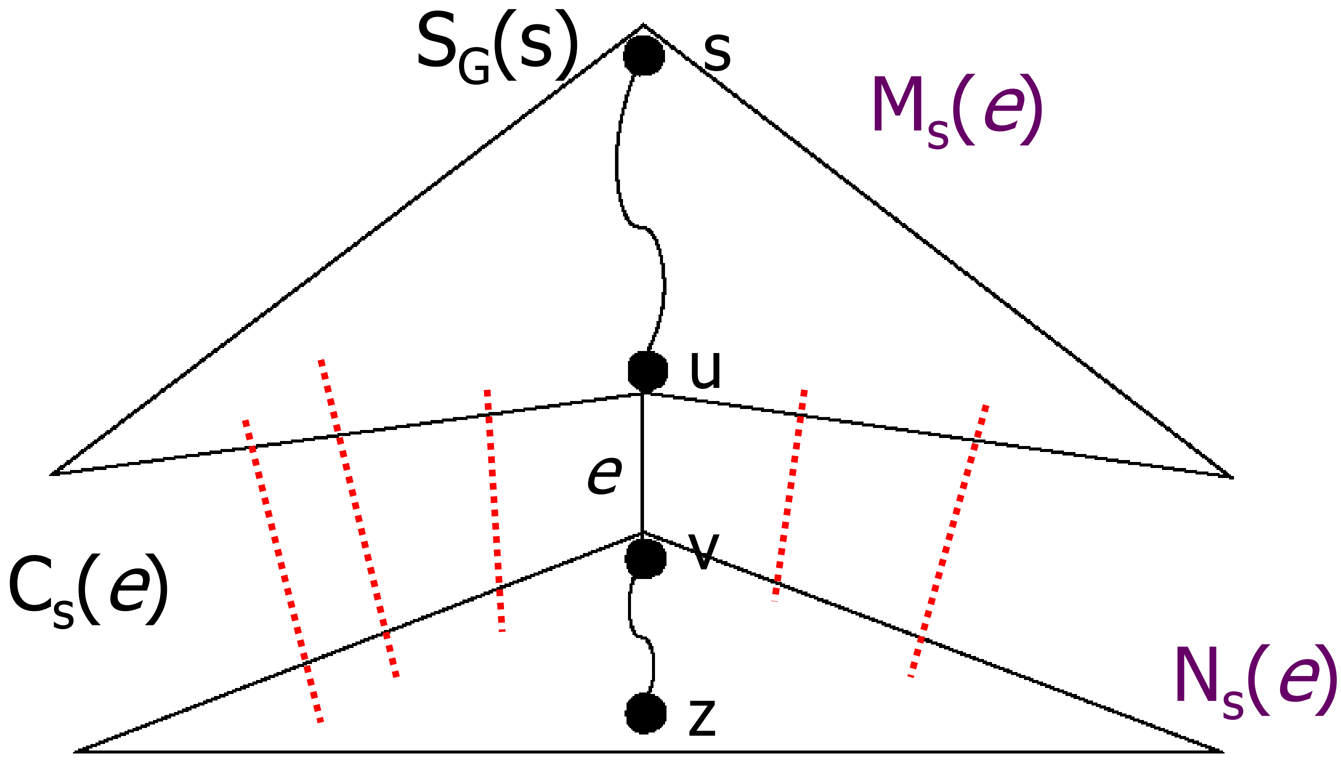
A picture



Crossing edges

- $(M_s(e), N_s(e))$ is a cut in G
- $C_s(e)=\{(x,y) \in E \setminus \{e\}: x \in M_s(e), y \in N_s(e)\}$ edges "belonging" to the cut: crossing edges

Crossing edges



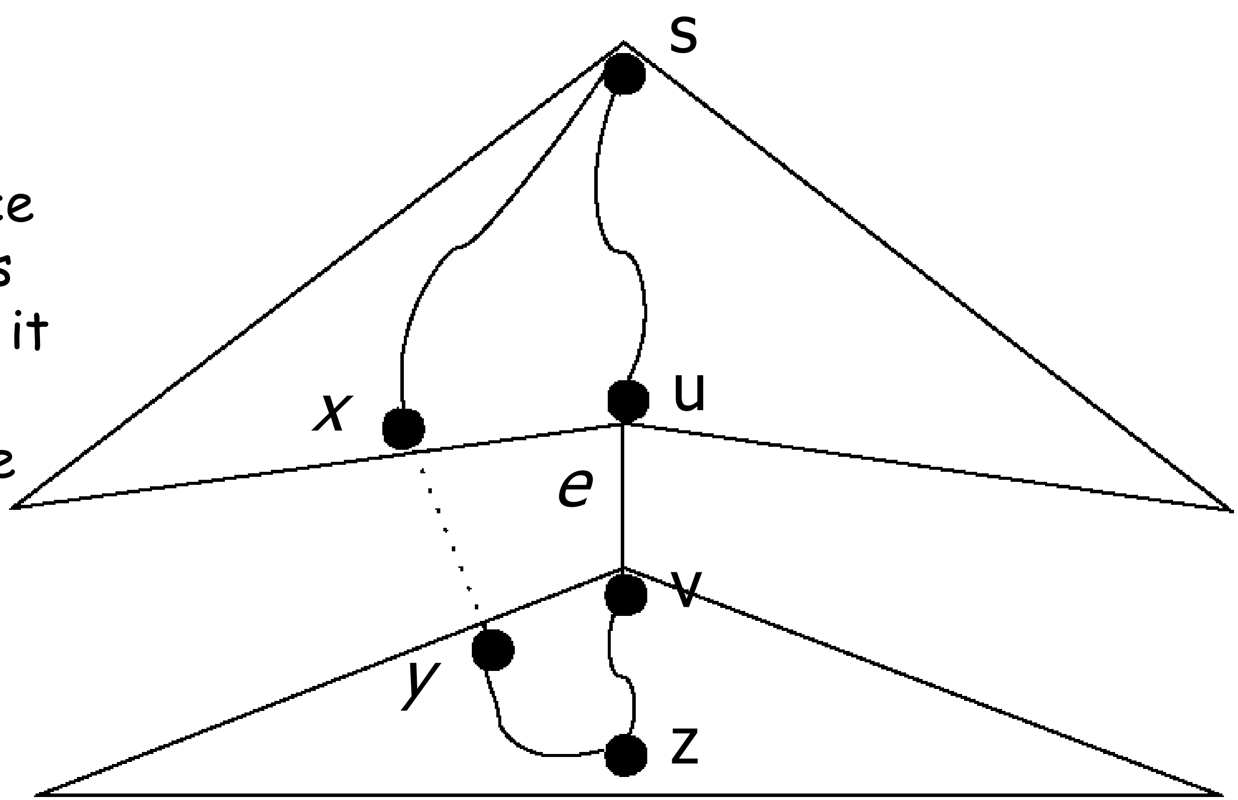
What about $P_{G-e}(s,z)$?

- Trivial: it does not use e, and it is shortest among all paths between s and z not using e
- There can be many replacement (shortest) paths between s and z not using e, but each one of them must cross at least once the cut $C_s(e)$
- Thus, the length of a replacement shortest path can be written as follows:

$$d_{G-e}(s,z)=\min_{f=(x,y)\in C_s(e)}\{d_{G-e}(s,x)+r(f)+d_{G-e}(y,z)\}$$

A replacement shortest path for *e*

Remark: since edge weights are positive, it must cross only once the cut. Can you see why?



$$d_{G-e}(s,z)=\min_{f=(x,y)\,\in\, C_s(e)}\{d_{G-e}(s,x)+r(f)+d_{G-e}(y,z)\}$$

How to compute $d_{G-e}(s,z)$

Let $f=(x,y) \in C_s(e)$; we will show that

$$d_{G-e}(s,x)+r(f)+d_{G-e}(y,z)=d_G(s,x)+r(f)+d_G(y,z)$$

Remark: $d_{G-e}(s,x)=d_G(s,x)$, since $x \in M_s(e)$

Lemma: Let $f=(x,y) \in C_s(e)$ be a crossing edge ($x \in M_s(e)$). Then $y \in M_z(e)$ (from which it follows that $d_{G-e}(y,z)=d_G(y,z)$).

A simple lemma

Proof (by contr.) Assume $y \notin M_z(e)$, then $y \in N_z(e)$. Hence, y is a descendant of u in $S_G(z)$, i.e., $P_G(z,y)$ uses e. Notice that v is closer to z than u in $S_G(z)$, and so $P_G(v,y)$ is a subpath of $P_G(z,y)$ and (recall that r(e) is positive):

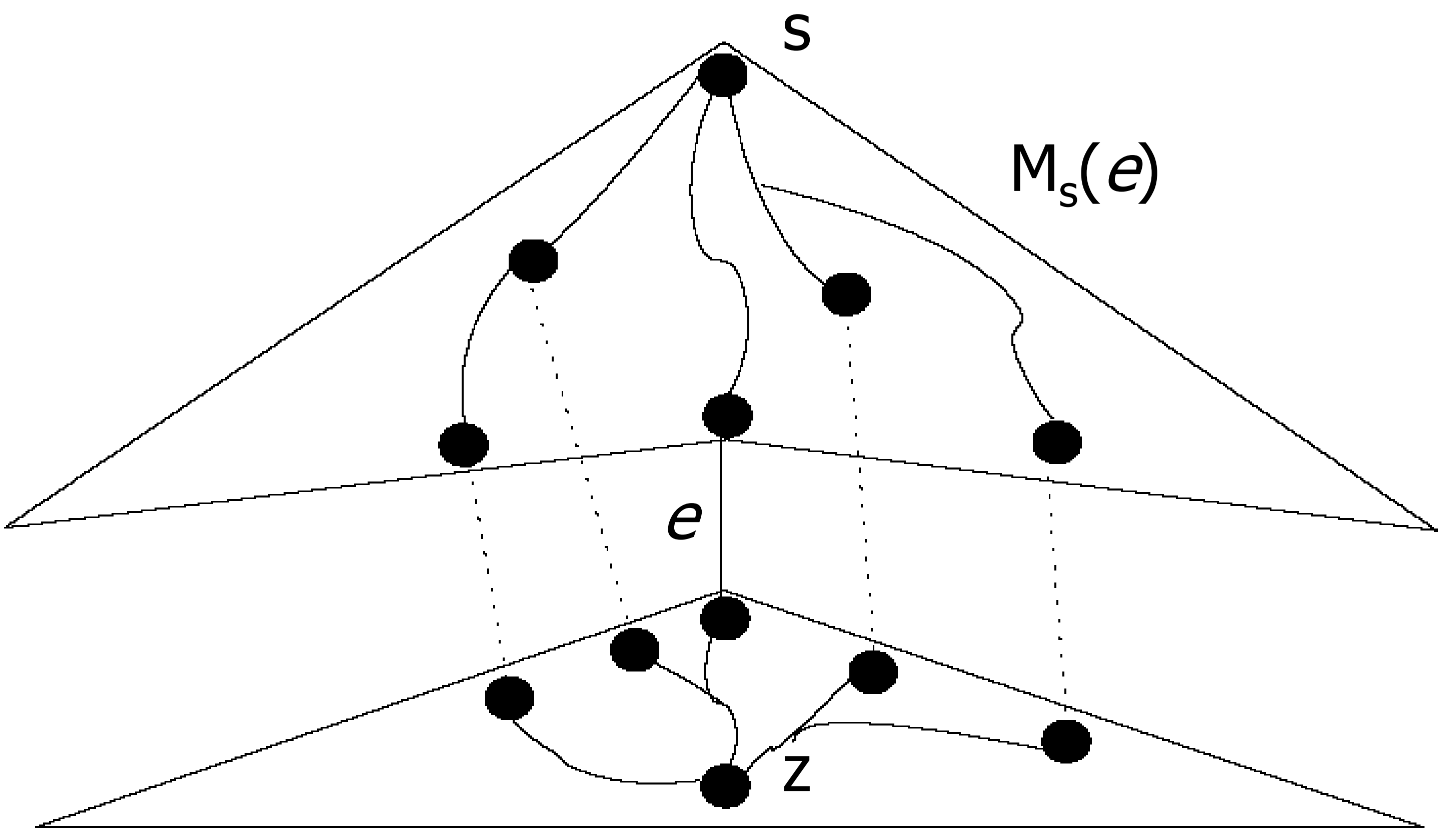
$$\boxed{d_G(v,y)=r(e) + d_G(u,y) > d_G(u,y).}$$

But $y \in N_s(e)$, and so $P_G(s,y)$ uses e. However, u is closer to s than v in $S_G(s)$, and so $P_G(u,y)$ is a subpath of $P_G(s,y)$ and:

$$\boxed{d_G(u,y)=r(e) + d_G(v,y) > d_G(v,y).}$$

■

A picture



$N_s(e) \subseteq M_z(e)$

Computing the length of replacement paths

Given $S_G(s)$ and $S_G(z)$, in $O(1)$ time we can compute the length of a shortest path between s and z passing through f and avoiding e as follows:

$$k(f) := \underbrace{d_{G-e}(s,x)}_{\substack{d_G(s,x) \\ \text{given by } S_G(s)}} + r(f) + \underbrace{d_{G-e}(y,z)}_{\substack{d_G(y,z) \\ \text{given by } S_G(z)}}$$

A corresponding algorithm

Step 1: Compute $S_G(s)$ and $S_G(z)$

Step 2: $\forall e \in P_G(s,z)$ check all the crossing edges in $C_s(e)$, and take the minimum w.r.t. the key k.

Time complexity

Step 1: $O(m + n \log n)$ time

Step 2: $O(m)$ crossing edges for each of the $O(n)$ edges on $P_G(s,z)$: since in $O(1)$ we can establish whether an edge of G is currently a crossing edge (can you guess how?), Step 2 costs $O(mn)$ time

$\Rightarrow$ Overall complexity: $O(mn)$ time

☺ Improves on $O(mn + n^2 \log n)$ if $m=o(n \log n)$

A more efficient solution: the Malik, Mittal and Gupta algorithm (1989)

- MMG have solved in $O(m + n \log n)$ time the following related problem: given a SP $P_G(s,z)$, compute its most vital edge, namely an edge whose removal induces the worst (i.e., longest) replacement shortest path between s and z.
- Their approach computes efficiently all the replacement shortest paths between s and z…

  …but this is exactly what we are looking for in our VCG-mechanism!

The MMG algorithm at work

The basic idea of the algorithm is that when an edge e on $P_G(s,z)$ is considered, then we have a priority queue H containing the set of nodes in $N_s(e)$; with each node $y \in H$ remains associated a key $k(y)$ and a corresponding crossing edge, defined as follows:

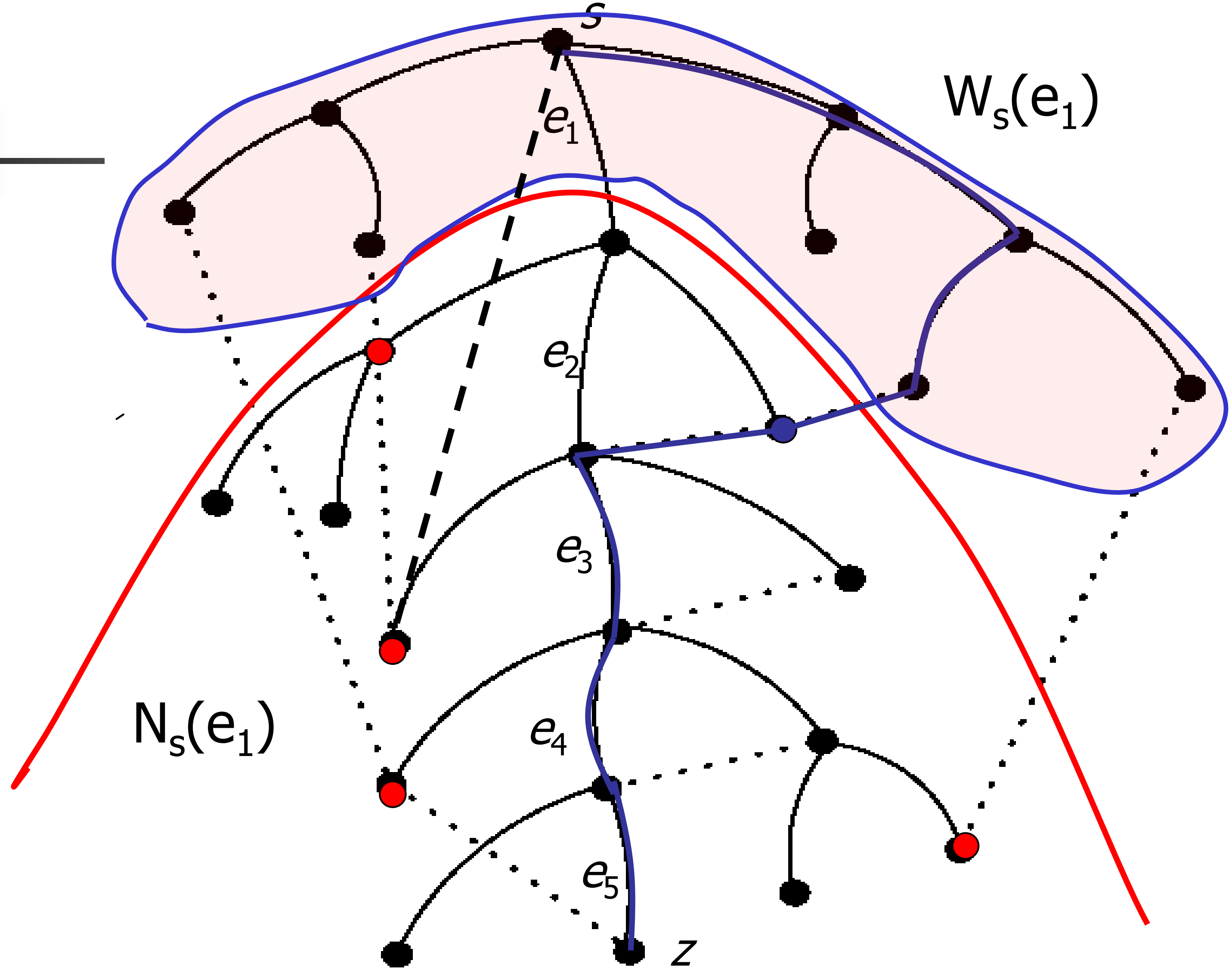
$$k(y) = \min_{(x,y)\in E,\ x\in M_s(e)} \{d_G(s,x)+r(x,y)+d_G(y,z)\}$$

$\Rightarrow$ $k(y)$ is the length of a SP in $G-e$ from s to z passing through the node y, and so the minimum key is associated with a replacement shortest path for e

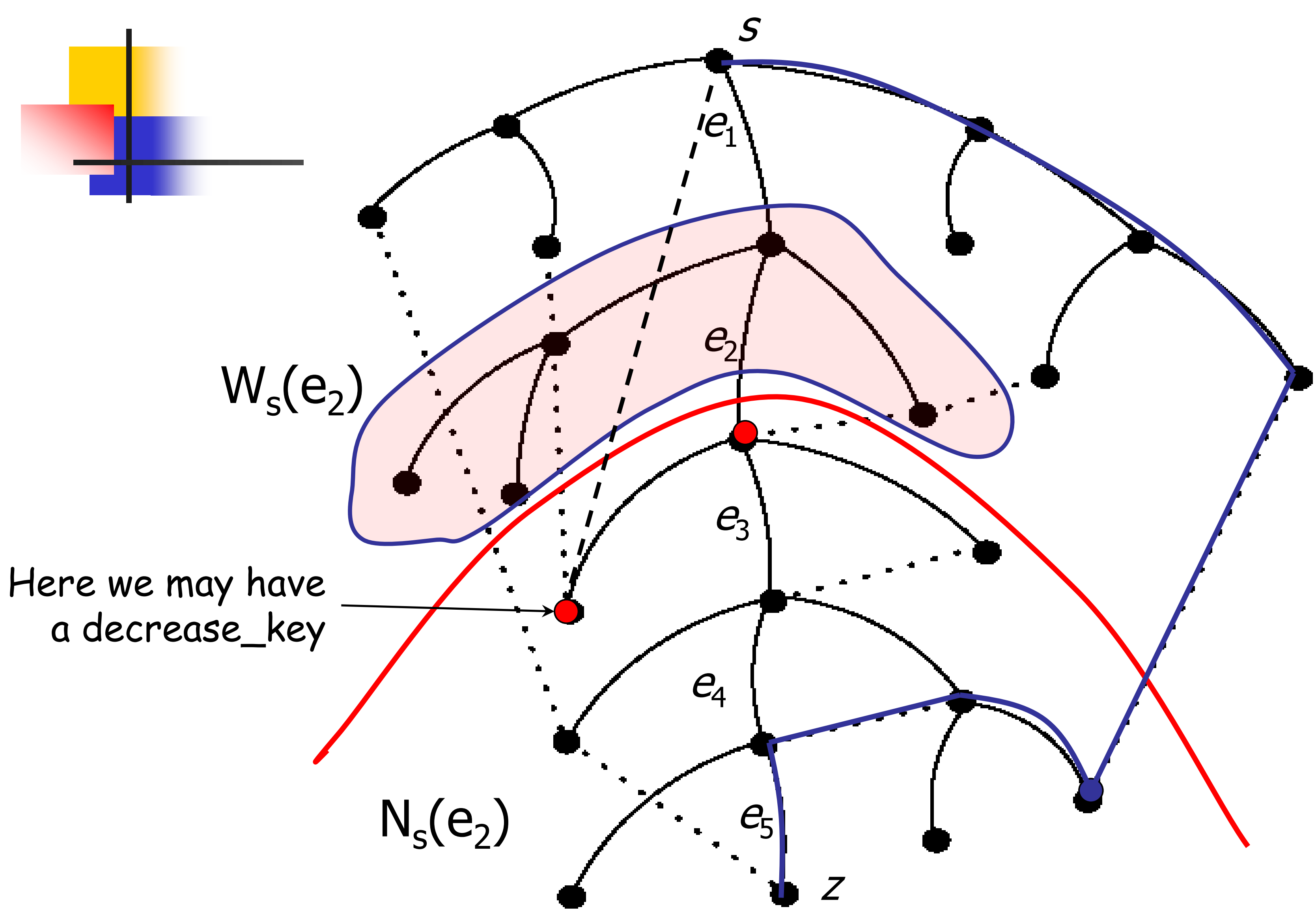
The MMG algorithm at work (2)

- Initially, $H = V$, and $k(y)=+\infty$ for each $y \in V$
- Let $P_G(s,z) = \{e_1, e_2, \dots, e_q\}$, and consider these edges one after the other. When edge e_i is considered, modify H as follows:
  - Remove from H all the nodes in $W_s(e_i)=N_s(e_{i-1})\setminus N_s(e_i)$ (for $i=1$, set $N_s(e_{i-1})=V$)
  - Consider all the edges (x,y) s.t. $x \in W_s(e_i)$ and $y \in H$, and compute $k'(y)=d_G(s,x)+r(x,y)+d_G(y,z)$. If $k'(y)<k(y)$, decrease $k(y)$ to $k'(y)$, and update the corresponding crossing edge to (x,y)
  - Then, find the minimum in H w.r.t. k, which returns the length of a replacement shortest path for e_i (i.e., $d_{G-e_i}(s,z)$), along with the selected crossing edge

An example



An example (2)



Time complexity of MMG

Theorem:

Given a shortest path between two nodes s and z in a graph G with n vertices and m edges, all the replacement shortest paths between s and z can be computed in $O(m + n \log n)$ time.

Time complexity of MMG

Proof: Compute $S_G(s)$ and $S_G(z)$ in $O(m + n \log n)$ time. Then, use a Fibonacci heap to maintain H (observe that $W_s(e_i)$ can be computed in $O(|W_s(e_i)|)$ time), on which the following operations are executed:

- A single make_heap
- n insert
- $q=O(n)$ find_min
- $O(n)$ delete
- $O(m)$ decrease_key

$\Rightarrow$ $O(m + n \log n)$ total time

In a Fibonacci heap, the amortized cost of a delete is $O(\log n)$, the amortized cost of a decrease_key is $O(1)$, while insert, find_min, and make_heap cost $O(1)$, so

Plugging-in the MMG algorithm into the VCG-mechanism

Corollary

There exists a VCG-mechanism for the private-edge SP problem running in $O(m + n \log n)$ time.

Proof.

Running time for the mechanism's algorithm: $O(m + n \log n)$ (Dijkstra).

Running time for computing the payments: $O(m + n \log n)$, by applying MMG to compute all the distances $d_{G-e}(s,z)$. ■