Shortest Paths Algorithms

Giacomo Nannicini

LIX, École Polytechnique
giacomon@lix.polytechnique.fr

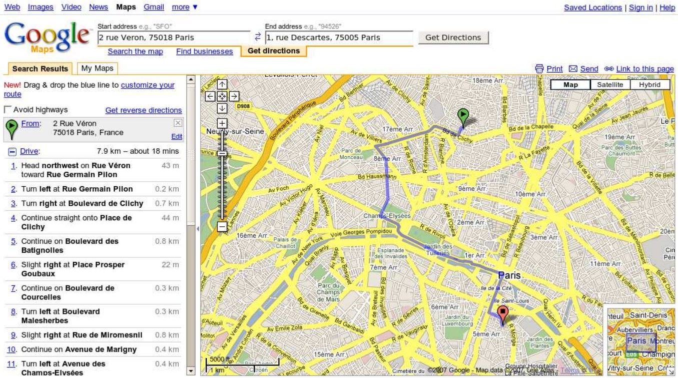
15/11/2007

Why shortest paths?

- Several real-life situation can be modeled as networks
  - Road networks
  - Telecommunications networks
  - Logistics
  - Etc...

Why shortest paths?

- Computing point-to-point shortest paths is of great interest to many users:
  - GPS devices with path computing capabilities
  - Many web sites provide users with route planners



Formulation

- We can formulate the problems as follows:

$$
\begin{aligned}
&(SP): \\
&z = \min \sum_{(i,j)\in A} c_{ij} x_{ij} \\
\sum_{k\in\delta^+(i)} x_{ik} - \sum_{k\in\delta^-(i)} x_{ki} &= 1 \text{ for } i = s \\
\sum_{k\in\delta^+(i)} x_{ik} - \sum_{k\in\delta^-(i)} x_{ki} &= 0 \text{ for } i \in V \setminus \{s,t\} \\
\sum_{k\in\delta^+(i)} x_{ik} - \sum_{k\in\delta^-(i)} x_{ki} &= -1 \text{ for } i = t \\
x_{ij} \geq 0 \text{ for } (i,j) &\in A \\
x &\in \mathbb{Z}^{|A|}
\end{aligned}
$$

where $x_{ij} = 1$ if (i,j) is in the shortest $s \to t$ path.

Complexity

- (SP) is an integer program
- Should be *very* difficult to solve, but we know that it is very easy in practice
- This is not the only case where we are "lucky"
- Let us investigate the reason

Easy Integer Programs

- Consider the problem (IP):

$$\min\{cx : Ax \leq b, x \in \mathbb{Z}^n_+\}$$

  with integral data A, b

- We know that a BFS will have the form $x = (x_B, x_N) = (B^{-1}b, 0)$ where B is an $m \times m$ nonsingular submatrix of (A, I) and I is an $m \times m$ identity matrix.

Easy Integer Programs

Observation:

If the optimal basis B has $det(B) = \pm 1$, then the linear programming relaxation solves (IP)

Proof: From Cramer's rule, $B^{-1} = adj(B)/det(B)$ where $adj(B)$ is the adjugate matrix $B_{ij} = (-1^{i+j})M_{ij}$. $adj(B)$ is integral, and as $det(B) = \pm 1$ we have B^{-1} integral $\Rightarrow B^{-1}b$ is integral for all integral b.

Totally Unimodular Matrices

Definition:

A matrix A is *totally unimodular* (TU) if every square submatrix of A has determinant $+1, -1$ or 0.

- If A is TU, $a_{ij} \in \{+1, -1, -\}\ \forall i, j$.
- Examples:

$$\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & -1 & -1 & 0 \\ -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Totally Unimodular Matrices

Proposition:

A is TU $\Leftrightarrow A^T$ is TU $\Leftrightarrow (A, I)$ is TU.

Sufficient Condition:

A matrix A is TU if:

1. $a_{ij} \in \{+1, -1, 0\}\ \forall i, j$.
2. Each column contains at most two nonzero coefficients.
3. There exists a partition (M_1, M_2) of the set M of rows such that each column j containing two nonzero coefficients satisfies $\sum_{i \in M_1} a_{ij} - \sum_{i \in M_2} a_{ij} = 0$.

Minimum Cost Network Flows

- Consider a digraph $G = (V, A)$ with arc capacities h_{ij} $\forall (i,j) \in A$, demands b_i (positive inflows or negative outflows) at each node $i \in V$, unit flow costs c_{ij} $\forall (i,j) \in A$.
- The minimum cost network flow problem is to find a feasible flow that satisfies all the demands at minimum cost.

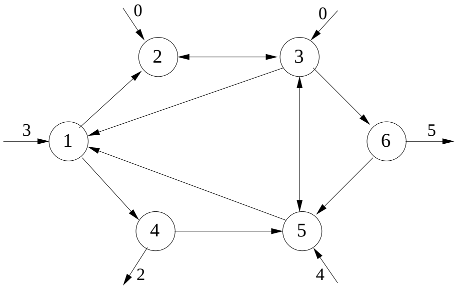
$$
\begin{array}{c}
(MCNF): \\
z = \min \sum_{(i,j) \in A} c_{ij} x_{ij}
\end{array}
$$

$$
\begin{aligned}
\sum_{k \in \delta^+(i)} x_{ik} - \sum_{k \in \delta^-(i)} x_{ki} &= b_i \text{ for } i \in V \\
0 \leq x_{ij} \leq h_{ij} \text{ for } (i,j) &\in A
\end{aligned}
$$

where x_{ij} denotes the flow in arc (i,j).

- The problem is feasible only if $\sum_{i \in V} b_i = 0$

Example



x_{12}	x_{14}	x_{23}	x_{31}	x_{32}	x_{35}	x_{36}	x_{45}	x_{51}	x_{53}	x_{65}		
1	1	0	-1	0	0	0	0	-1	0	0	=	3
-1	0	1	0	-1	0	0	0	0	0	0	=	0
0	0	-1	1	1	1	1	0	0	-1	0	=	0
0	-1	0	0	0	0	0	1	0	0	0	=	-2
0	0	0	0	0	-1	0	-1	1	1	-1	=	4
0	0	0	0	0	0	-1	0	0	0	1	=	-5

$$0 \leq x_{ij} \leq h_{ij}.$$

Minimum Cost Network Flows

Proposition:

The constraint matrix A arising in a minimum cost network flow problem is totally unimodular.

Proof: The matrix A is of the form $\begin{pmatrix} C \\ I \end{pmatrix}$, where C comes from the flow conservation constraints, and I from the capacity constraints. Therefore we only have to show that C is TU. This follows from the sufficient condition above, with the partition $M_1 = M$ and $M_2 = \emptyset$.

($MCNF$) Is An Easy Problem

Corollary:

In a ($MCNF$) problem, if b_i and h_{ij} are integral, then each extreme point is integral.

- Each time that we have a network flow problem with the constraints in the form above, we know that the solution is integral.
- It is a situation that is frequently found when modeling problems on networks.

(SP) Is An Easy Problem

$$
\begin{aligned}
& (SP): \\
z = \min \sum_{(i,j)\in A} c_{ij} x_{ij} & \\
\sum_{k\in\delta^+(i)} x_{ik} - \sum_{k\in\delta^-(i)} x_{ki} &= 1 \text{ for } i = s \\
\sum_{k\in\delta^+(i)} x_{ik} - \sum_{k\in\delta^-(i)} x_{ki} &= 0 \text{ for } i \in V \setminus \{s,t\} \\
\sum_{k\in\delta^+(i)} x_{ik} - \sum_{k\in\delta^-(i)} x_{ki} &= -1 \text{ for } i = t \\
x_{ij} \geq 0 \text{ for } (i,j) &\in A \\
x \in \mathbb{Z}^{|A|} &
\end{aligned}
$$

- It is clearly a $(MCNF) \Rightarrow$ integral solution!

The Shortest Path Tree Problem

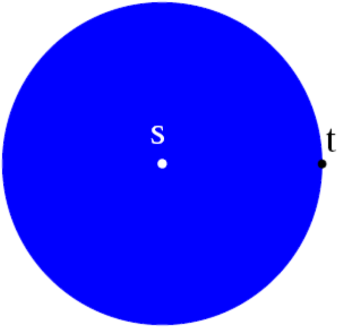
- Suppose we want to compute the shortest path from a source node s to all other nodes $v \in V$.
- Formulation:

$$
\begin{aligned}
& (SPT): \\
z = \min & \sum_{(i,j)\in A} c_{ij} x_{ij} \\
\sum_{k\in\delta^+(i)} x_{ik} - \sum_{k\in\delta^-(i)} x_{ki} &= |V| - 1 \text{ for } i = s \\
\sum_{k\in\delta^+(i)} x_{ik} - \sum_{k\in\delta^-(i)} x_{ki} &= -1 \text{ for } i \in V \setminus \{s\} \\
x_{ij} \geq 0 \text{ for } (i,j) &\in A \\
x &\in \mathbb{Z}^{|A|}
\end{aligned}
$$

where $x_{ij} \geq 0$ if (i,j) is an arc of the SPT rooted at s.

Dijkstra's Algorithm

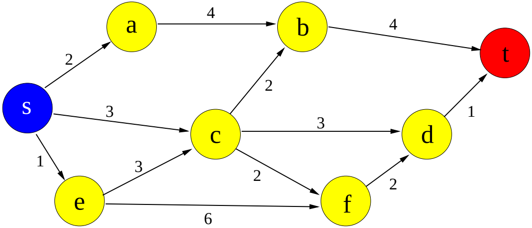
- The Shortest Path Problem can be solved with purely combinatorial algorithms.
- The most famous one: Dijkstra's algorithm.
- Idea: explore nodes, starting from the nearest to the source node s, in a "ball" centered at s.



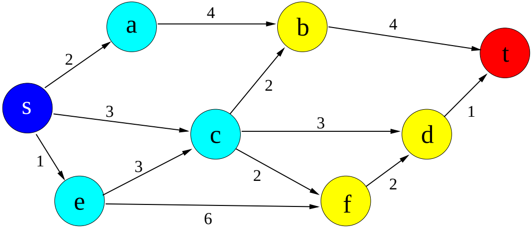
Dijkstra's Algorithm

- Let s be the source node, Q be the queue of explored nodes, $d[v]$ be the tentative distance of v from s, $p[v]$ be the tentative parent node of v on the shortest $s \rightarrow v$ path.
- Initialize: $Q \leftarrow \emptyset, d[v] \leftarrow \infty \; \forall v \in V \setminus \{s\}, p[v] \leftarrow NIL \; \forall v \in V \setminus \{s\}, d[s] \leftarrow 0, p[s] \leftarrow s$. We say that nodes in Q are `explored`.
- Algorithm:
  1. Extract $i \leftarrow \arg\min_{v \in Q}\{d[v]\}$ (we say that i is `settled`).
  2. For each $j \in \delta^+(i) : d[i] + c_{ij} < d[j]$ set $Q \leftarrow Q \cup \{j\}, d[j] \leftarrow d[i] + c_{ij}, p[j] \leftarrow i$.
  3. Repeat until a stopping criterion is met.
- Commonly used stopping criteria:
  - As soon as a target node t is settled.
  - When Q is empty.

Example



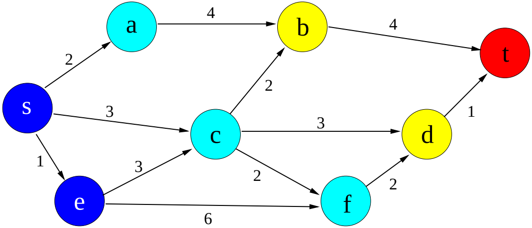
Example



Priority Queue:

- e ← 1
- a ← 2
- c ← 3

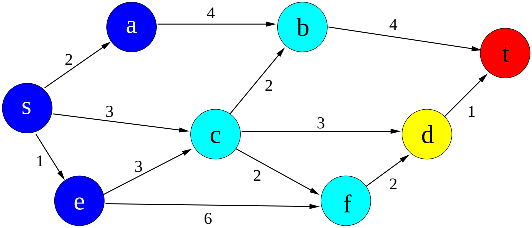
Example



Priority Queue:

- a $\leftarrow$ 2
- c $\leftarrow$ 3
- f $\leftarrow$ 7

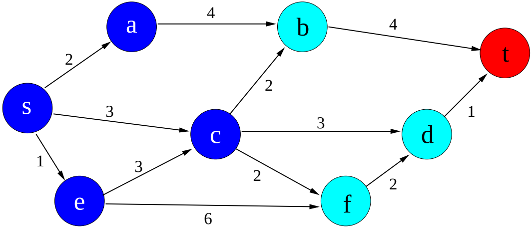
Example



Priority Queue:

- c ← 3
- b ← 6
- f ← 7

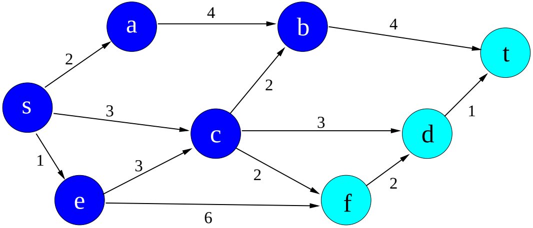
Example



Priority Queue:

- b ← 5
- f ← 5
- d ← 6

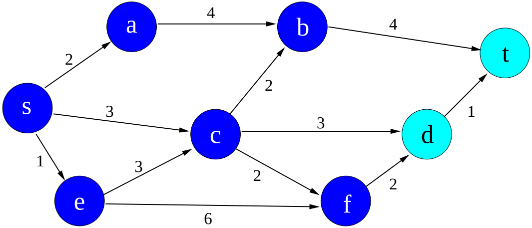
Example



Priority Queue:

- f ← 5
- d ← 6
- t ← 9

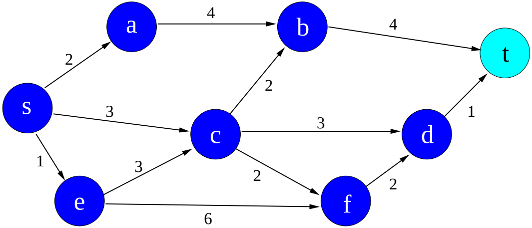
Example



Priority Queue:

- d ← 6
- t ← 9

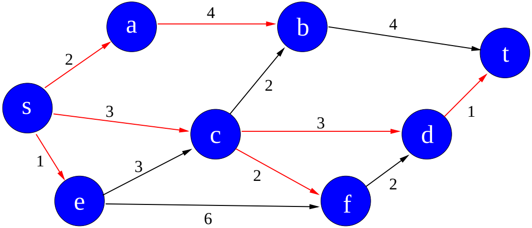
Example



Priority Queue:

- t ← 7

Example

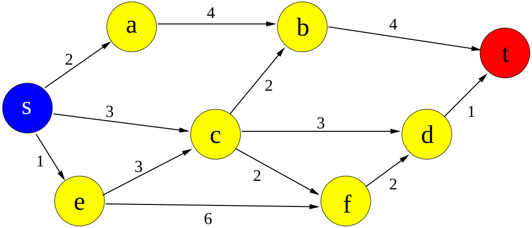


Priority Queue: $\emptyset$

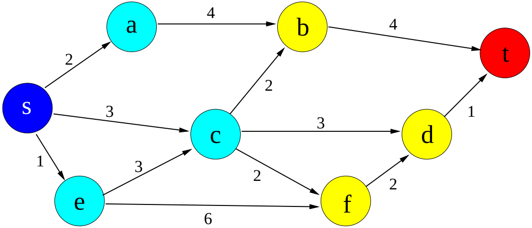
Goal Directed Search: A^*

- Same principle as Dijkstra's algorithm: extract minimum from a queue, explore adjacent nodes, update labels, repeat.
- Main difference: add to the key of the priority queue a potential function $\pi(v)$ which estimates $d(v,t)$.
- If $\pi(v) \leq d(v,t) \quad \forall v$ then A^* computes shortest paths.
- If $\pi(v)$ is a good estimation of $d(v,t)$, A^* explores considerably fewer nodes than Dijkstra's algorithm.

Goal Directed Search: A^*



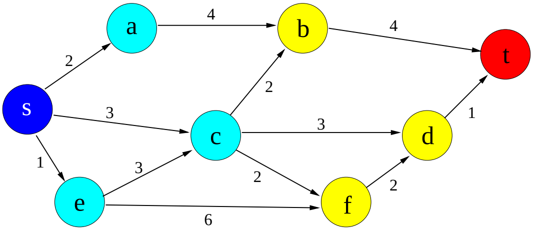
Goal Directed Search: A^*



Priority Queue:

- e ← 1
- a ← 2
- c ← 3

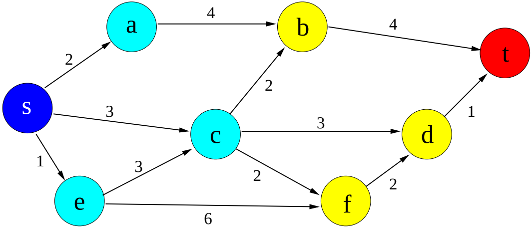
Goal Directed Search: A^*



Priority Queue:

- e $\leftarrow 1 + \pi(e)$
- a $\leftarrow 2 + \pi(a)$
- c $\leftarrow 3 + \pi(c)$

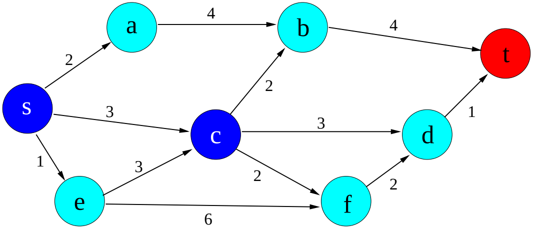
Goal Directed Search: A^*



Priority Queue:

- c $\leftarrow$ 7
- e $\leftarrow$ 8
- a $\leftarrow$ 10

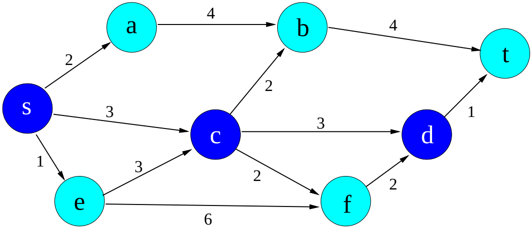
Goal Directed Search: A^*



Priority Queue:

- d $\leftarrow$ 7
- e $\leftarrow$ 8
- f $\leftarrow$ 8
- b $\leftarrow$ 9
- a $\leftarrow$ 10

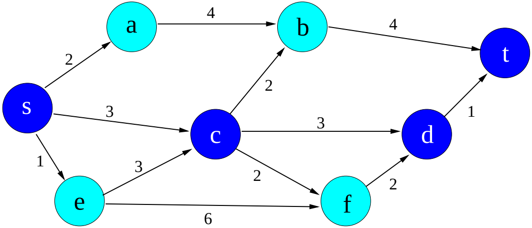
Goal Directed Search: A^*



Priority Queue:

- t ← 7
- e ← 8
- f ← 8
- b ← 9
- a ← 10

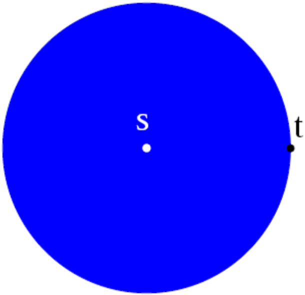
Goal Directed Search: A^*



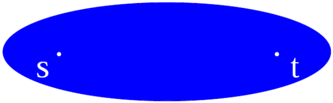
Priority Queue:

- e $\leftarrow$ 8
- f $\leftarrow$ 8
- b $\leftarrow$ 9
- a $\leftarrow$ 10

Goal Directed Search: A^*



Dijkstra's algorithm

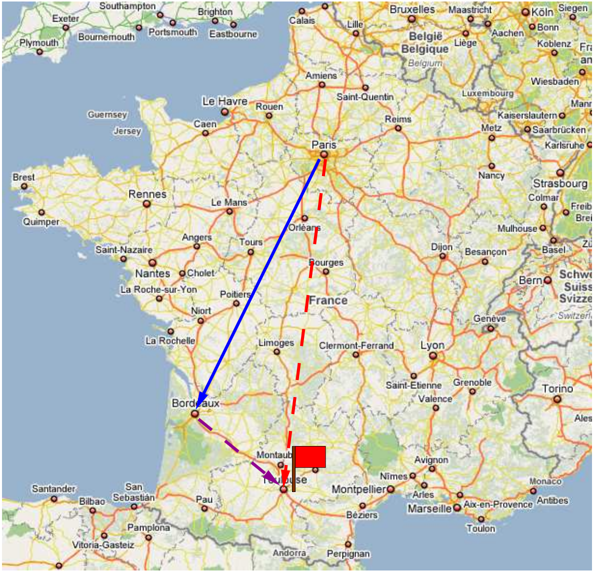


A^*

A Good Lower Bound

- The quality of $\pi(v)$ is critic for performances: the closer to $d(v, t)$, the better.
- On an Euclidean plane, we can use the standard Euclidean distance to compute potentials.
- Idea ([Goldberg and Harrelson, 2004]): use a few nodes as landmarks to compute distances within the graph.
- Then triangle inequality comes to our help.
  - ALT algorithm: A^*, Landmarks, Triangle inequality.

A Good Lower Bound



A Good Lower Bound

- Suppose we have a set $L \subset V$ of landmarks, i.e. we know $d(v,\ell), d(\ell,v) \quad \forall v \in V, \ell \in L$.
- Then we have $d(v,\ell) \le d(v,t) + d(t,\ell)$ and $d(\ell,t) \le d(\ell,v) + d(v,t) \quad \forall v \in V, \ell \in L$.

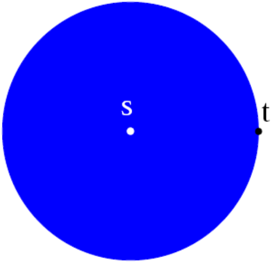
Lower bounding function:

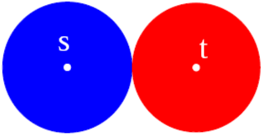
$$\pi(v) = \max_{\ell \in L} \max\{d(v,\ell) - d(t,\ell), d(\ell,t) - d(\ell,v)\}.$$

is a lower bound to $d(v,t) \forall v,t \in V$.

Bidirectional Search

- Suppose we want to compute a point-to-point shortest path.
- Main idea: explore nodes not only from the source, but also from target node, using the reverse graph $\bar{G} = (V, \bar{A})$ where $(i,j) \in \bar{A} \Leftrightarrow (j,i) \in A$.
- This will reduce the search space.





Balancing the search

- At each iteration, how do we choose between the forward and the backward search?
- Simple idea: alternate between the two searches at each iteration.
- This works very well in practice.
- Stopping criterion: stop as soon as there is a node v which has been settled by both searches.

Theorem:

During bidirectional Dijkstra's algorithm, suppose that v is the first node that is settled by both searches. Then the shortest path from s to t passes through v.

Bidirectional A^*

- In principle, we could bidirectionalize the A^* algorithm, and it should still work.
- We can't use the same stopping criterion! (Try to prove it)
- Conservative idea:
  - Keep the value β of the shortest $s \rightarrow t$ path found so far.
  - This may be updated each time that we obtain a new meeting point.
  - Suppose v_f is the minimum element of the forward search queue, and v_b is the minimum element of the backward search queue. If $\beta \leq d(s, v_f) + d(v_b, t)$ then we can stop the search, and β is optimal.
- We have to work on the potentials: we need $\pi_f(v) + \pi_b(v)$ to be constant $\forall v \in V$.

Transit Node Routing

- We can make the following two observations:
  - The set of nodes such that at least one node appears on any sufficiently long shortest path (*transit nodes*) is very small.
  - For any s, t pair, the number of these "important" nodes that are involved in a shortest path computation (*access nodes*) is very small.
- Using these ideas, we can develop a very efficient algorithm.

Transit Node Routing

- Consider a set $\mathcal{T} \subset V$ of transit nodes, and an access mapping $\mathcal{A} : V \to 2^{\mathcal{T}}$ that maps a vertex to its access nodes set.
- Consider a locality filter $\mathcal{L} : V \times V \to \{\texttt{true}, \texttt{false}\}$ that decides whether an $s \to t$ query is local or not.

Property:

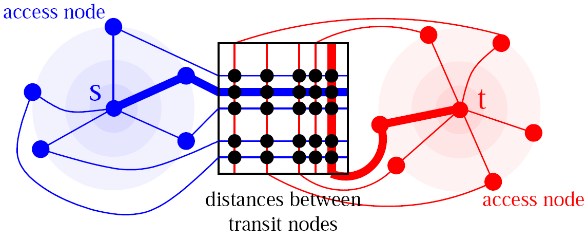
$$\neg\mathcal{L}(s,t) \Rightarrow d(s,t) = \min_{u \in \mathcal{A}(s), v \in \mathcal{A}(t)} \{d(s,u) + d(u,v) + d(v,t)\}.$$

Transit Node Routing

- Assume we have precomputed $d(u,v) : u, v \in \mathcal{T}$.
- Algorithm:
  - If $\neg\mathcal{L}(s,t)$, compute $d(s,t)$ as

$$d(s,t) = \min\{d(s,u) + d(u,v) + d(v,t) | u \in \mathcal{A}(s), v \in \mathcal{A}(t)\}.$$

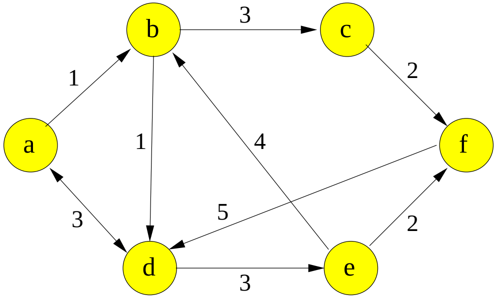
  - Otherwise, use any other shortest paths algorithm.



Transit Node Routing

- A very efficient implementation [Sanders and Schultes, 2007] has been presented at the 9th DIMACS Computational Challenge (late 2006).
- It is based on the Highways Hierarchies algorithm [Sanders and Schultes, 2005].
- Average query times for the european road network: 5.6 microseconds, no more than a few hundreds microseconds in the worst case.

Exercises: AMPL



- Write model and data file for the SP problem for this network, with source node: *a* and target node: *f* (use CPLEX: `option solver cplex;`).
- Write a run file that uses the model and data file to compute and display the shortest path for each node pair in the network.
- Modify those files to compute the SP tree rooted at each node.