An Efficient Algorithm for the Fast Delivery Problem

Iago A. Carvalho[1] Thomas Erlebach[2]
Kleitos Papadopoulos[2]

[1]Dept. of Computer Science, Universidada Federal de Minas Gerais, Brazil

[2]School of Informatics, University of Leicester, England

FCT 2019, Copenhagen, Denmark, 14 August 2019

Motivation: Delivery of Packages by Drones

Motivation: Delivery of Packages by Drones

What if drones (or agents) with different speeds need to collaborate to deliver a package as quickly as possible?

Problem Definition: FastDelivery

Input:

- Undirected graph $G = (V, E)$ with edge lengths $\ell_e > 0$.
  Convention: $|V| = n, |E| = m$
- $k \leq n$ agents. For $1 \leq i \leq k$, agent i is located at node $a_i \in V$ at time 0 and has velocity $v_i > 0$.
- A package that needs to be delivered from source $s \in V$ to destination $y \in V$

Output:

- Schedule of agent movements to collaboratively deliver the package from s to y.

Objective:

- Minimize the time when the package reaches y.

Problem Definition: FastDelivery

Input:

- Undirected graph $G = (V, E)$ with edge lengths $\ell_e > 0$.
  Convention: $|V| = n, |E| = m$
- $k \leq n$ agents. For $1 \leq i \leq k$, agent i is located at node $a_i \in V$ at time 0 and has velocity $v_i > 0$.
- A package that needs to be delivered from source $s \in V$ to destination $y \in V$

Output:

- Schedule of agent movements to collaboratively deliver the package from s to y.

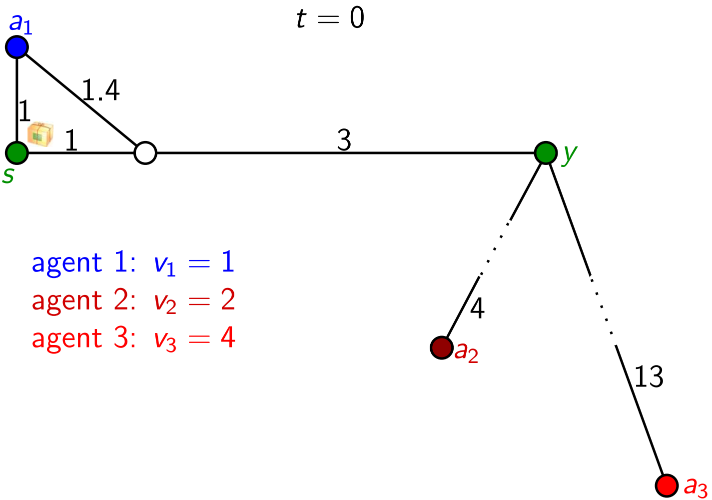
Objective:

- Minimize the time when the package reaches y.

Remark:

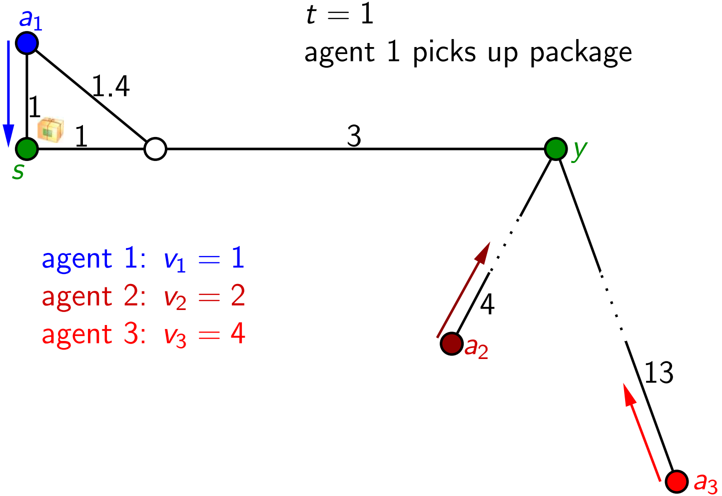
- Package handovers are instantaneous and can happen at a node or at any point on an edge.

Example



agent 1: $v_1 = 1$
agent 2: $v_2 = 2$
agent 3: $v_3 = 4$

Example



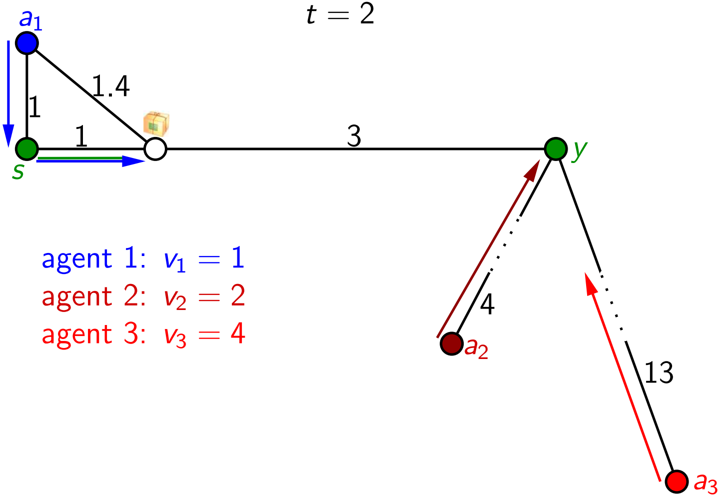
$t = 1$

agent 1 picks up package

agent 1: $v_1 = 1$
agent 2: $v_2 = 2$
agent 3: $v_3 = 4$

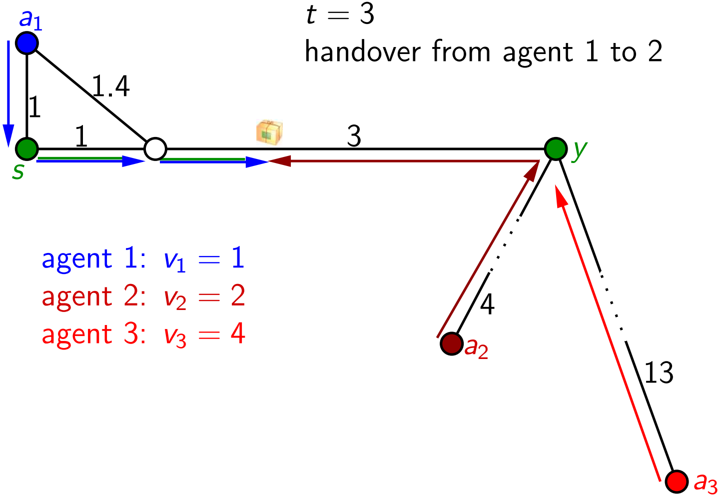
Example

$t = 2$



agent 1: $v_1 = 1$
agent 2: $v_2 = 2$
agent 3: $v_3 = 4$

Example



$t = 3$

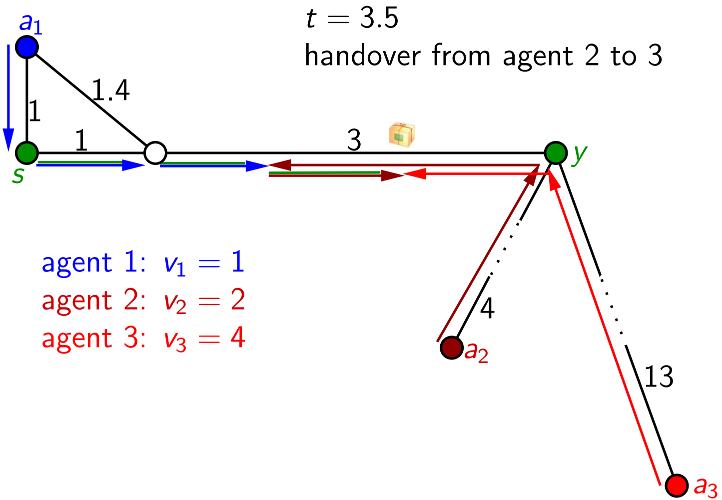
handover from agent 1 to 2

agent 1: $v_1 = 1$

agent 2: $v_2 = 2$

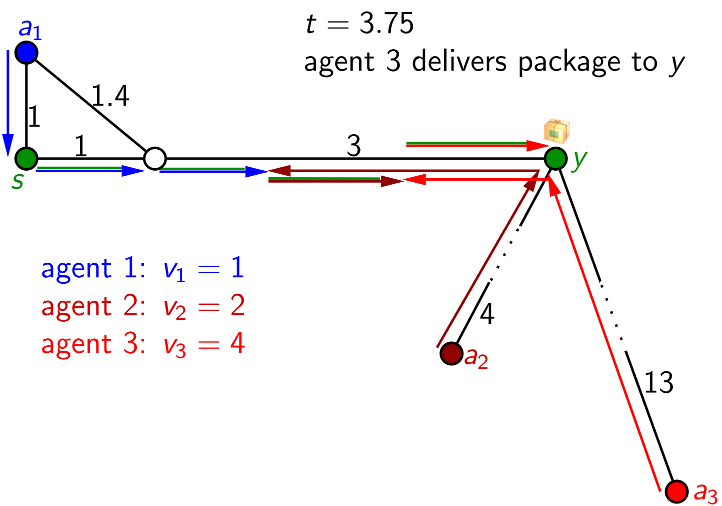
agent 3: $v_3 = 4$

Example



$t = 3.5$
handover from agent 2 to 3
a_1
1.4
1
1
3
s
y
4
a_2
13
a_3
agent 1: $v_1 = 1$
agent 2: $v_2 = 2$
agent 3: $v_3 = 4$

Example

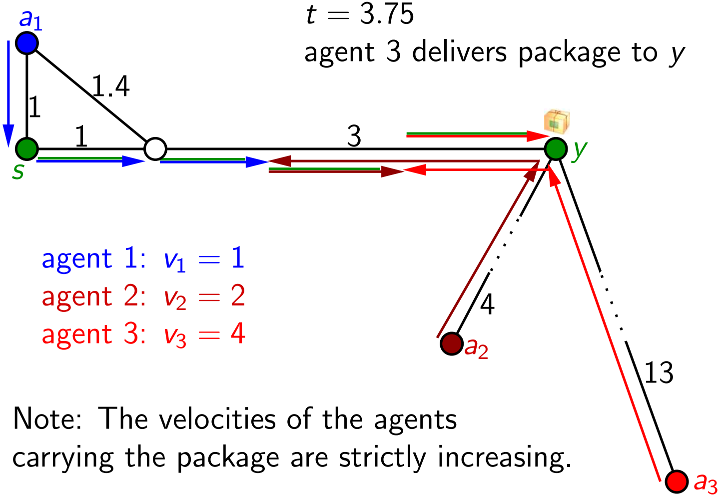


$t = 3.75$
agent 3 delivers package to y
a_1
1
1.4
1
3
s
y
4
a_2
13
a_3
agent 1: $v_1 = 1$
agent 2: $v_2 = 2$
agent 3: $v_3 = 4$

Example

$t = 3.75$

agent 3 delivers package to y



agent 1: $v_1 = 1$
agent 2: $v_2 = 2$
agent 3: $v_3 = 4$

Note: The velocities of the agents carrying the package are strictly increasing.

Previous Work

- **Bärtschi, Graf, Mihalák 2018:**
  - $O(k^2 m + kn^2 + \mathrm{APSP})$ time algorithm for FastDelivery based on dynamic programming
  - For minimizing the energy consumption among all fastest delivery schedules: NP-hardness for planar graphs, polynomial algorithms for paths and for equal velocities
- **Bärtschi et al. 2017:**
  - Energy-efficient delivery by agents with equal speed: NP-hard for multiple packages, polynomial for a single package
- **Chalopin et al. 2013, 2014; Bärtschi et al. 2017:**
  - Energy-constrained collaborative delivery

Our Result

Theorem

FastDelivery *can be solved in* $O(kn \log n + km)$ *time*

- Improvement over $O(k^2 m + kn^2 + \mathrm{APSP})$ by Bärtschi et al. 2018:
  - $O(n^4)$ to $O(n^3)$ for dense graphs and $k = \Omega(n)$
  - $O(n^3)$ to $O(n^2 \log n)$ for sparse graphs and $k = \Omega(n)$

Our Result

Theorem

FASTDELIVERY *can be solved in* $O(kn \log n + km)$ *time*

- Improvement over $O(k^2 m + kn^2 + \mathrm{APSP})$ by Bärtschi et al. 2018:
  - $O(n^4)$ to $O(n^3)$ for dense graphs and $k = \Omega(n)$
  - $O(n^3)$ to $O(n^2 \log n)$ for sparse graphs and $k = \Omega(n)$
- **Main idea:** Apply Dijkstra's algorithm for graphs with edges with **time-dependent transit times** (cf. **Cooke and Halsey, 1966; Delling and Wagner, 2009**)

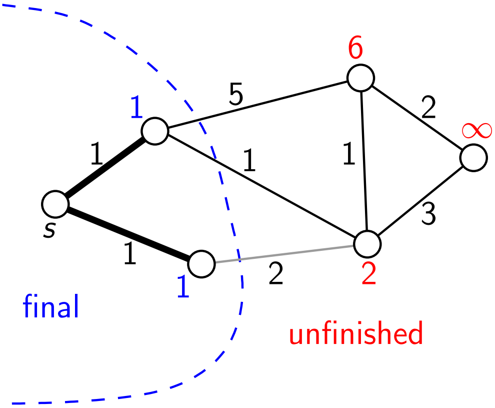
Our Result

Theorem

FASTDELIVERY *can be solved in* $O(kn \log n + km)$ *time*

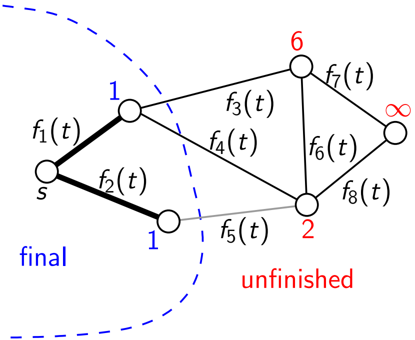
- Improvement over $O(k^2 m + kn^2 + \mathrm{APSP})$ by Bärtschi et al. 2018:
  - $O(n^4)$ to $O(n^3)$ for dense graphs and $k = \Omega(n)$
  - $O(n^3)$ to $O(n^2 \log n)$ for sparse graphs and $k = \Omega(n)$
- **Main idea:** Apply Dijkstra's algorithm for graphs with edges with **time-dependent transit times** (cf. **Cooke and Halsey, 1966; Delling and Wagner, 2009**)
- **Key Ingredient:** Transport package over an edge as quickly as possible (FASTLINEDELIVERY problem).

Reminder: Standard Dijkstra Algorithm



- In each step:
  - find the unfinished node v with smallest tentative distance
  - make v final and update the tentative distances of its unfinished neighbors (“relax” edges)

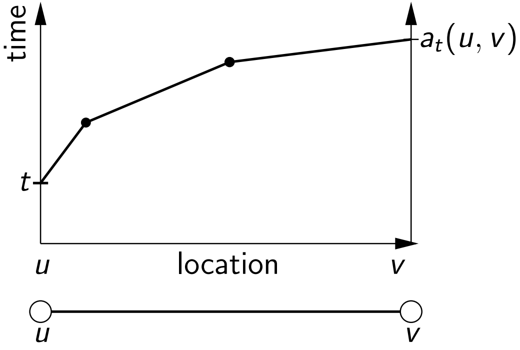
Dijkstra with Time-Dependent Transit Times



- In each step:
  - find the unfinished node v with smallest tentative earliest arrival time (EAT)
  - make v final and update the tentative EAT of its unfinished neighbors, using current transit times
- Correct if transit times satisfy FIFO property (no overtaking).

Diagram for Package Transport Over One Edge

- For any edge $uv \in E$, let $\boldsymbol{a_t(u,v)}$ be the earliest time when a package present at u at time t can reach v over edge uv
- The transport of the package from u to v can be visualised in a time-space diagram:



Package Transport Satisfies FIFO

Claim

For $t < t'$, $a_t(u, v) \leq a_{t'}(u, v)$.

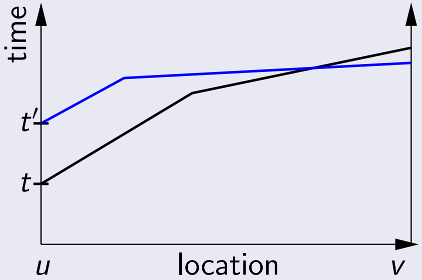
Package Transport Satisfies FIFO

Claim

For $t < t'$, $a_t(u,v) \leq a_{t'}(u,v)$.

Proof.

Assume otherwise:



□

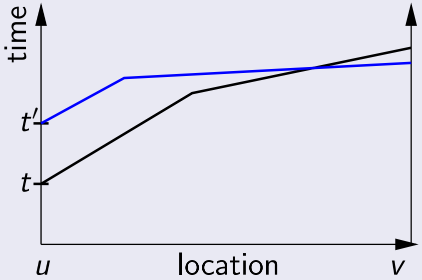
Package Transport Satisfies FIFO

Claim

For $t < t'$, $a_t(u, v) \le a_{t'}(u, v)$.

Proof.

Assume otherwise:



At the crossover point, the faster agent could take over from one of the agents starting at time t, so the package could be transported to reach v before $a_t(u, v)$. Contradiction! □

Time-Dependent Dijkstra for FastDelivery

```
d(s) ← t_s;              /* time when first agent reaches s */
d(v) ← ∞ for all v ∈ V \ {s};
final(v) ← false for all v ∈ V;
insert s into priority queue Q with priority d(s);
while Q not empty do
    u ← node with minimum d value in Q;
    delete u from Q; final(u) ← true;
    if u = y then break;
    t ← d(u);              /* time when package reaches u */
    forall neighbors v of u with final(v) = false do
        a_t(u, v) ← FastLineDelivery(u, v, t);
        if a_t(u, v) < d(v) then
            d(v) ← a_t(u, v);
            if v ∈ Q then decrease priority of v to d(v);
            else insert v into Q with priority d(v);
```

Running-Time for Whole Algorithm

- Run standard Dijkstra from each of the k agent nodes a_i to find the earliest arrival time for each agent at each node in V: $O(k(n \log n + m))$ time.

Running-Time for Whole Algorithm

- Run standard Dijkstra from each of the k agent nodes a_i to find the earliest arrival time for each agent at each node in V: $O(k(n \log n + m))$ time.
- Sort agent arrivals at each node (and discard slower agents that arrive after faster agents): $O(nk \log k)$ time.

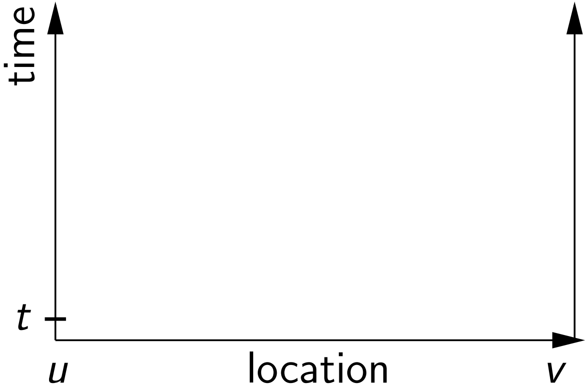
Running-Time for Whole Algorithm

- Run standard Dijkstra from each of the k agent nodes a_i to find the earliest arrival time for each agent at each node in V: $O(k(n \log n + m))$ time.
- Sort agent arrivals at each node (and discard slower agents that arrive after faster agents): $O(nk \log k)$ time.
- Time-dependent Dijkstra framework: $O(n \log n + T)$, where T is the time for m calls of FastLineDelivery (including preprocessing)

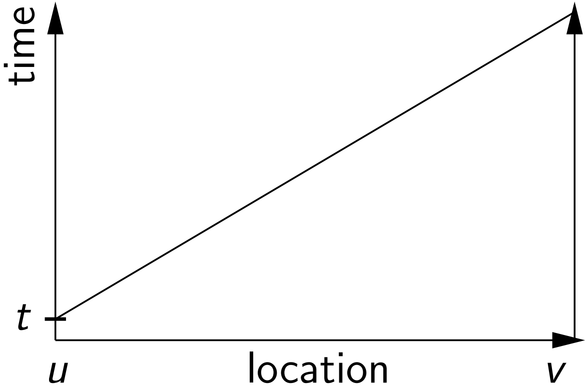
Running-Time for Whole Algorithm

- Run standard Dijkstra from each of the k agent nodes a_i to find the earliest arrival time for each agent at each node in V: $O(k(n \log n + m))$ time.
- Sort agent arrivals at each node (and discard slower agents that arrive after faster agents): $O(nk \log k)$ time.
- Time-dependent Dijkstra framework: $O(n \log n + T)$, where T is the time for m calls of FastLineDelivery (including preprocessing)
- Components of T:
  - $O(nk \log k)$ for preprocessing each node in $O(k \log k)$ time
  - $O(mk)$ for executing FastLineDelivery(u, v, t) in $O(k)$ time for m edges

$\Rightarrow$ Total: $O(kn \log n + km)$

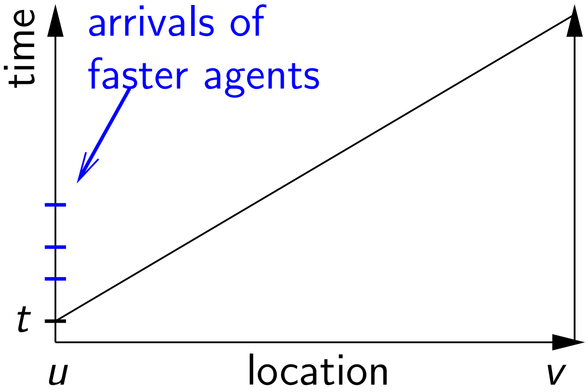


- Agent brings package to u at time t

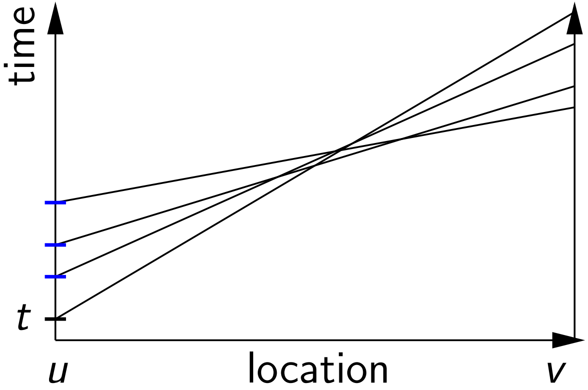


- Same agent could carry package to v

Preprocessing for FastLineDelivery(u, v, t)

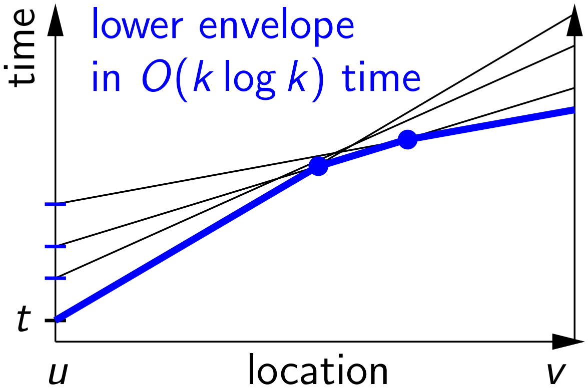


- Faster agents may help



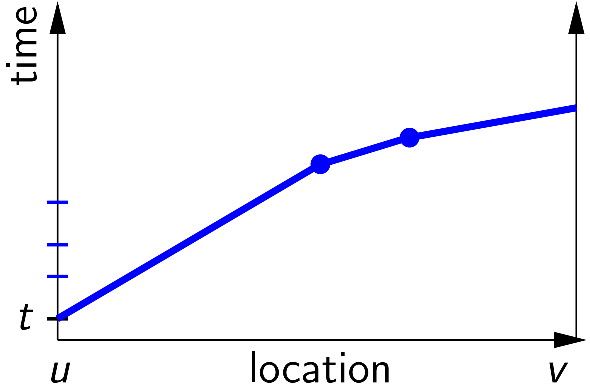
- Trajectories of faster agents

Preprocessing for FastLineDelivery(u, v, t)

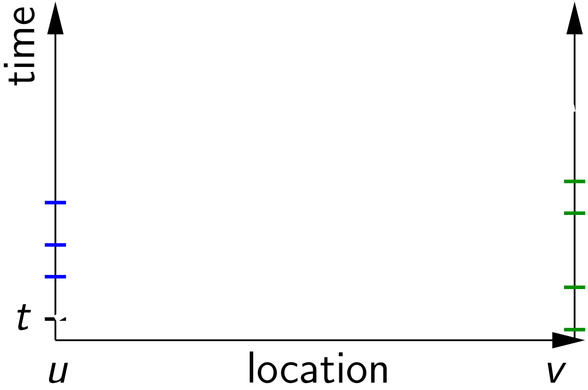


- Use sweepline algorithm (Bentley and Ottmann 1979)

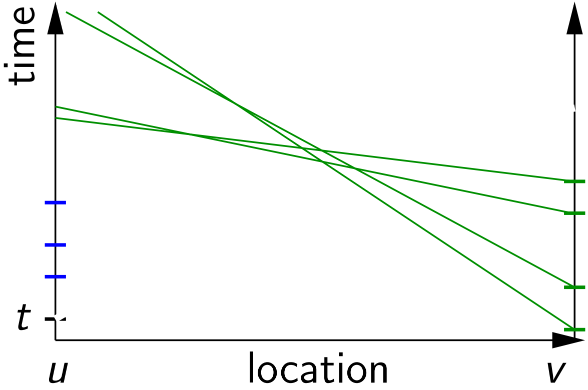
Preprocessing for FastLineDelivery(u, v, t)



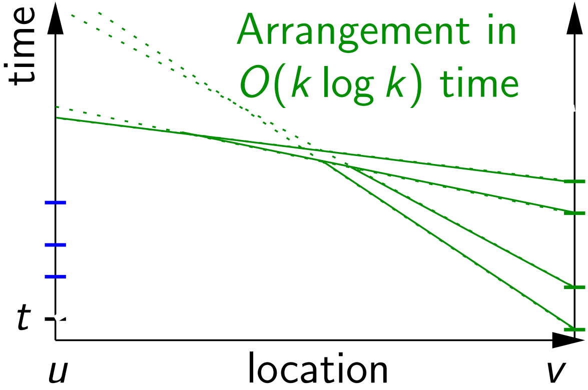
- Fastest way for agents coming from u to deliver package to v



- Agents coming from v may help

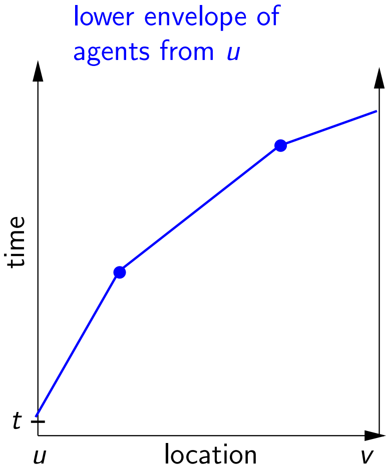


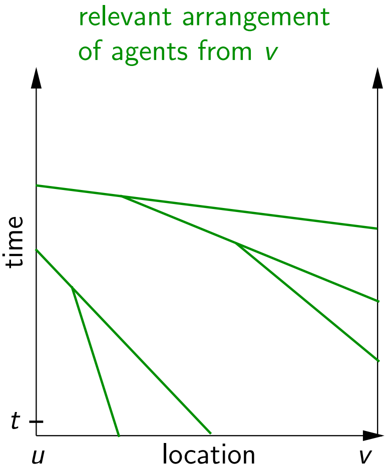
- Trajectories of agents coming from v



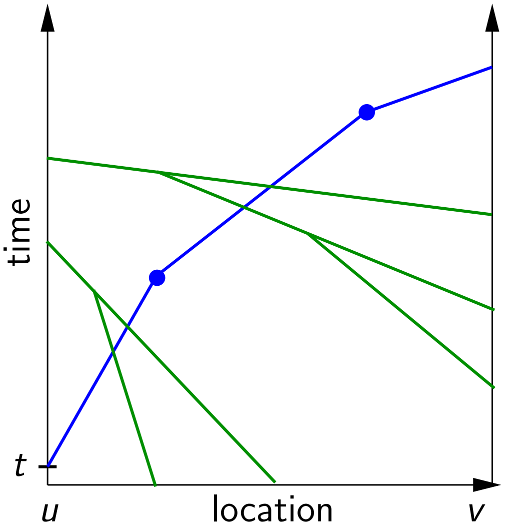
- Relevant arrangement of agents coming from v

Result of preprocessing for FASTLINEDELIVERY(u, v, t)



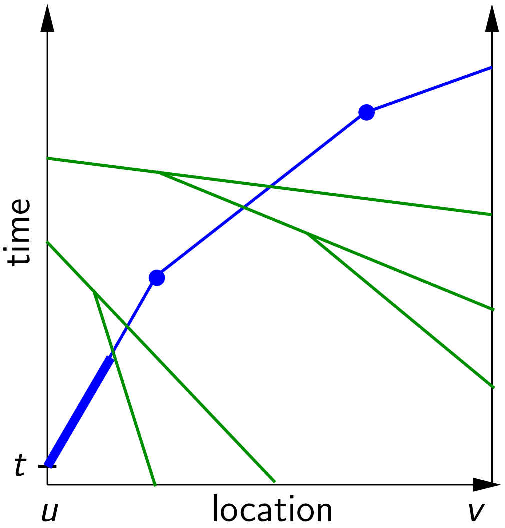


Computing FASTLINEDELIVERY(u, v, t)

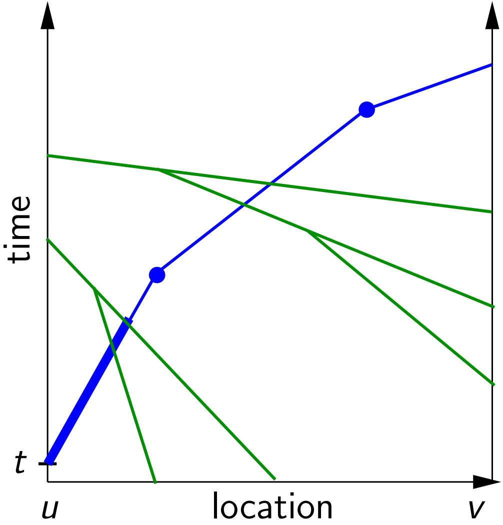


- Trace the lower envelope from u to v

Computing FastLineDelivery(u, v, t)

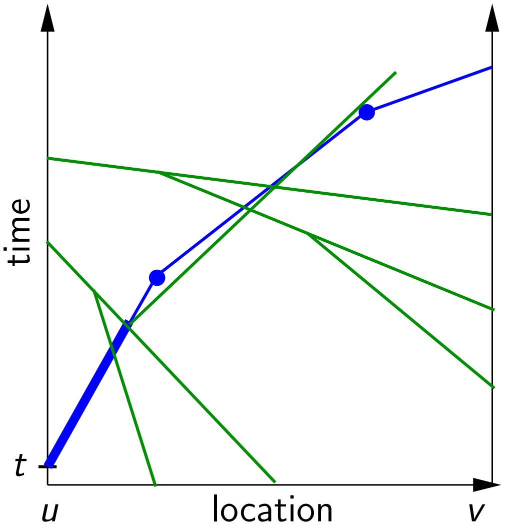


- Intersect slower agent, do nothing



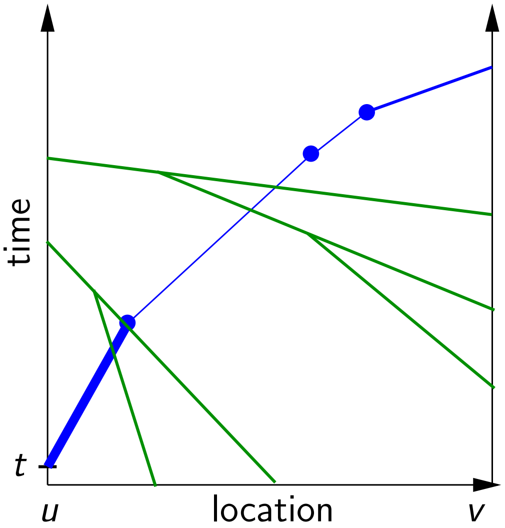
- Intersect faster agent, hand over

Computing FastLineDelivery(u, v, t)

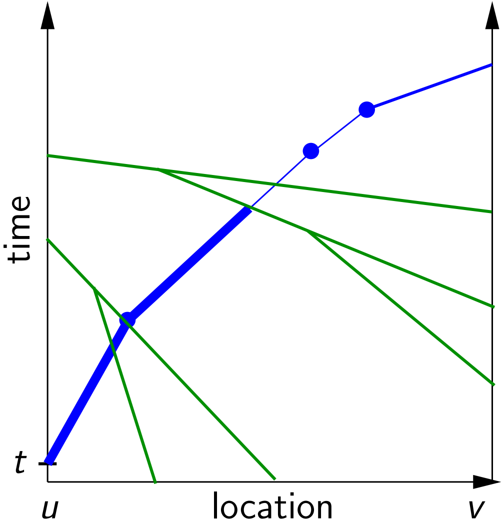


- Intersect faster agent, hand over

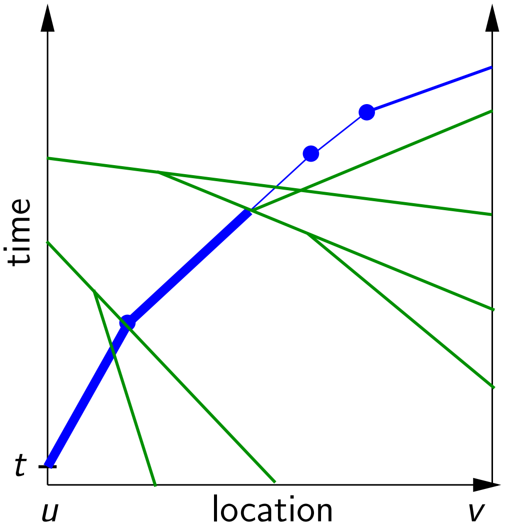
Computing FastLineDelivery(u, v, t)



- Intersect faster agent, hand over, update lower envelope

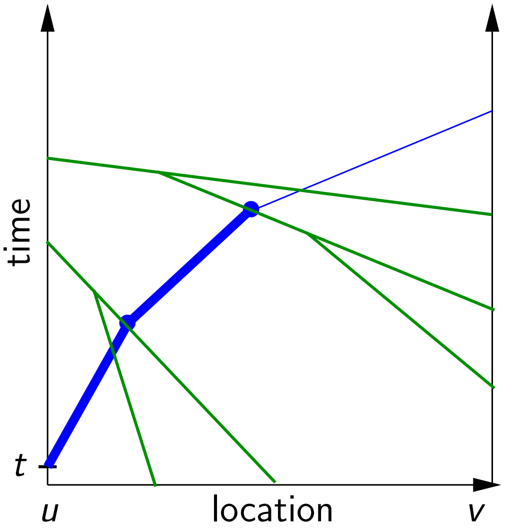


- Intersect faster agent, hand over



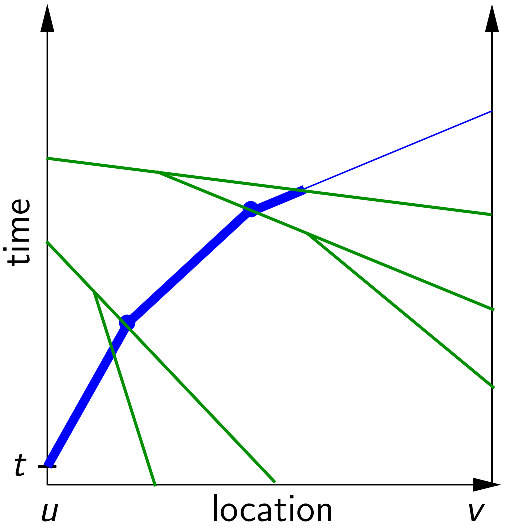
- Intersect faster agent, hand over

Computing FASTLINEDELIVERY(u, v, t)



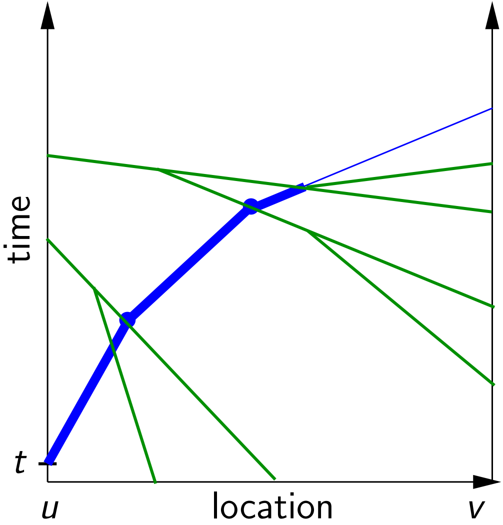
- Intersect faster agent, hand over, update lower envelope

Computing FastLineDelivery(u, v, t)



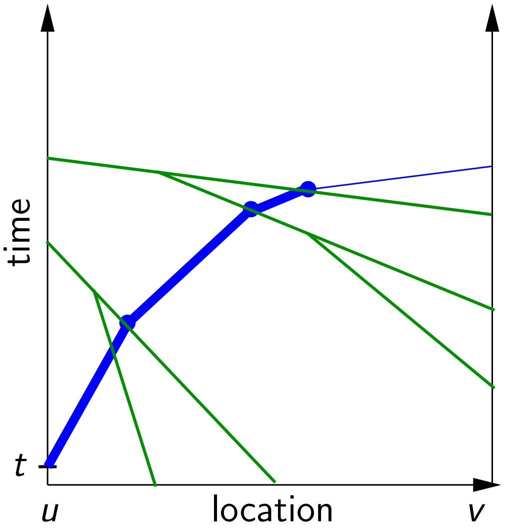
- Intersect faster agent, hand over

Computing FASTLINEDELIVERY(u, v, t)



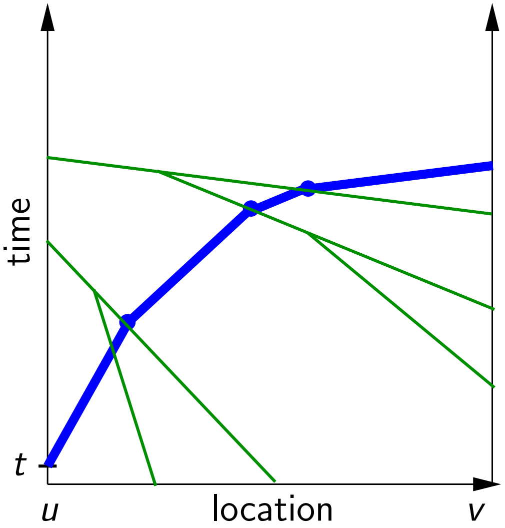
- Intersect faster agent, hand over

Computing FastLineDelivery(u, v, t)



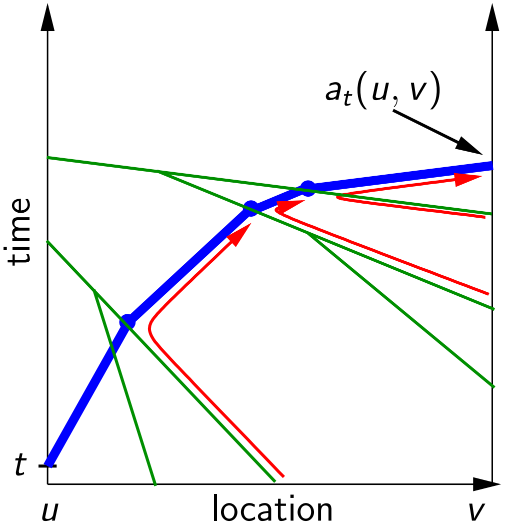
- Intersect faster agent, hand over, update lower envelope

Computing FastLineDelivery(u, v, t)



- Reach v

Computing FASTLINEDELIVERY(u, v, t)



- Solution to FASTLINEDELIVERY(u, v, t)

Summary of Solution to FastLineDelivery

- Compute relevant arrangement once for every node: $O(k \log k)$ time per node
- Compute lower envelope for each node when it is made final: $O(k \log k)$ time per node
- Compute $a_t(u, v)$ in $O(k)$ time (once for each edge):
  - trace lower envelope of agents coming from u, in the direction from u to v
  - update lower envelope whenever a faster agent of the relevant arrangement of v is met
- Correctness can be proved by induction (the current lower envelope is always a fastest and foremost solution using only the agents from u and those from v that could have reached the package by now)

Conclusion

Our Result

- FastDelivery can be solved in $O(kn \log n + km)$ time
- Key ideas:
  - Use Dijkstra for time-dependent transit times
  - Solve FastLineDelivery using geometric representation of agent movements

Future Work

- Can the running-time be improved further?
- Consider FastDelivery in the Euclidean plane?

Thank you!

Questions?