

Local Distributed Decision

Pierre Fraigniaud Amos Korman David Peleg

CNRS and University Paris Diderot

Workshop on Sublinear Algorithms, Bertinoro, May 23-27, 2011

Decision problems

Does randomization helps?

Nondeterminism

Power of oracles

Further works

Outline

Decision problems

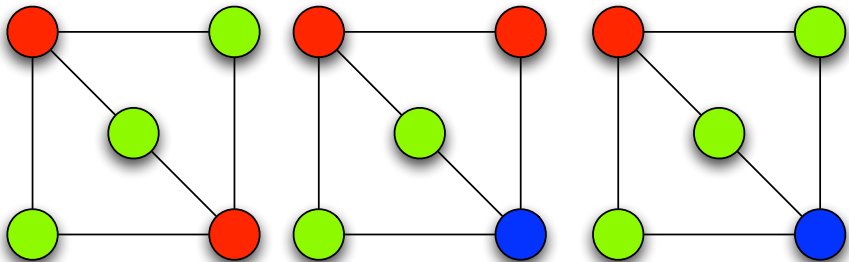
Does randomization helps?

Nondeterminism

Power of oracles

Further works

Decide coloring



Computational model

LOCAL model

In each round during the execution of a distributed algorithm, every processor:

1. **sends** messages to its neighbors,
2. **receives** messages from its neighbors, and
3. **computes**, i.e., performs individual computations.

Computational model

LOCAL model

In each round during the execution of a distributed algorithm, every processor:

1. **sends** messages to its neighbors,
2. **receives** messages from its neighbors, and
3. **computes**, i.e., performs individual computations.

Input

An input configuration is a pair (G, x) where G is a connected graph, and every node $v \in V(G)$ is assigned as its **local** input a binary string $x(v) \in \{0, 1\}^*$.

Computational model

LOCAL model

In each round during the execution of a distributed algorithm, every processor:

1. **sends** messages to its neighbors,
2. **receives** messages from its neighbors, and
3. **computes**, i.e., performs individual computations.

Input

An input configuration is a pair (G, x) where G is a connected graph, and every node $v \in V(G)$ is assigned as its **local** input a binary string $x(v) \in \{0, 1\}^*$.

Output

The output of node v performing Algorithm $\mathcal{A}$ running in G with input x and identity assignment Id :

$$out_{\mathcal{A}}(G, x, Id, v)$$

Languages

A **distributed language** is a decidable collection of configurations.

Languages

A **distributed language** is a decidable collection of configurations.

- Coloring =
 $\{(G, x) \text{ s.t. } \forall v \in V(G), \forall w \in N(v), x(v) \neq x(w)\}.$

Languages

A **distributed language** is a decidable collection of configurations.

- ▶ Coloring =
 $\{(G, x) \text{ s.t. } \forall v \in V(G), \forall w \in N(v), x(v) \neq x(w)\}.$
- ▶ At-Most-One-Marked = $\{(G, x) \text{ s.t. } \|x\|_1 \leq 1\}.$

Languages

A **distributed language** is a decidable collection of configurations.

- ▶ Coloring =
 $\{(G, x) \text{ s.t. } \forall v \in V(G), \forall w \in N(v), x(v) \neq x(w)\}.$
- ▶ At-Most-One-Marked = $\{(G, x) \text{ s.t. } \|x\|_1 \leq 1\}.$
- ▶ Consensus =
 $\{(G, (x_1, x_2)) \text{ s.t. } \exists u \in V(G), \forall v \in V(G), x_2(v) = x_1(u)\}.$

Languages

A **distributed language** is a decidable collection of configurations.

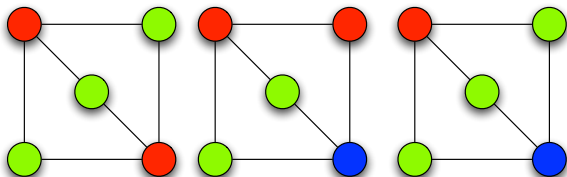
- ▶ Coloring =
 $\{(G, x) \text{ s.t. } \forall v \in V(G), \forall w \in N(v), x(v) \neq x(w)\}.$
- ▶ At-Most-One-Marked = $\{(G, x) \text{ s.t. } \|x\|_1 \leq 1\}.$
- ▶ Consensus =
 $\{(G, (x_1, x_2)) \text{ s.t. } \exists u \in V(G), \forall v \in V(G), x_2(v) = x_1(u)\}.$
- ▶ MIS = $\{(G, x) \text{ s.t. } S = \{v \in V(G) \mid x(v) = 1\} \text{ is a MIS}\}.$

Decision

Let $\mathcal{L}$ be a distributed language.

Algorithm $\mathcal{A}$ decides $\mathcal{L} \iff$ for every configuration (G, x) :

- ▶ If $(G, x) \in \mathcal{L}$, then for every identity assignment Id ,
 $\text{out}_{\mathcal{A}}(G, x, \text{Id}, v) = \text{"yes"}$ for every node $v \in V(G)$;
- ▶ If $(G, x) \notin \mathcal{L}$, then for every identity assignment Id ,
 $\text{out}_{\mathcal{A}}(G, x, \text{Id}, v) = \text{"no"}$ for at least one node $v \in V(G)$.



Local decision

Let t be a function of triplets (G, x, Id) .

Definition

$LD(t)$ is the class of all distributed languages that can be decided by a distributed algorithm that runs in at most t communication rounds.

Local decision

Let t be a function of triplets (G, x, Id) .

Definition

$LD(t)$ is the class of all distributed languages that can be decided by a distributed algorithm that runs in at most t communication rounds.

- ▶ $Coloring \in LD(1)$ and $MIS \in LD(1)$.
- ▶ $AMOM$, $Consensus$, and $SpanningTree$ are not in $LD(t)$, for any $t = o(n)$.

Outline

Decision problems

Does randomization helps?

Nondeterminism

Power of oracles

Further works

Related work

What can be computed locally?

Define **LCL** as **LD($O(1)$)** involving

- ▶ solely graphs of constant maximum degree
- ▶ inputs taken from a set of constant size

Theorem (Naor and Stockmeyer [STOC '93])

*If there exists a **randomized** algorithm that constructs a solution for a problem in **LCL** in $O(1)$ rounds, then there is also a **deterministic** algorithm constructing a solution for that problem in $O(1)$ rounds.*

Proof uses Ramsey theory.

Not clearly extendable to languages in **LD($O(1)$)** $\setminus$ **LCL**.

$(\Delta + 1)$ -coloring

Arbitrary graphs

- ▶ can be randomly computed in expected #rounds $O(\log n)$
(Alon, Babai, Itai [J. Alg. 1986]) (Luby [SIAM J. Comput. 1986])
- ▶ best known deterministic algorithm performs in $2^{O(\sqrt{\log n})}$ rounds (Panconesi, Srinivasan [J. Algorithms, 1996])

Bounded degree graphs

- ▶ Randomization does not help for 3-coloring the ring
(Naor [SIAM Disc. Maths 1991])
- ▶ can be randomly computed in expected #rounds $O(\log \Delta + \sqrt{\log n})$ (Schneider, Wattenhofer [PODC 2010])
- ▶ best known deterministic algorithm performs in $O(\Delta + \log^* n)$ rounds
(Barenboim, Elkin [STOC 2009]) (Kuhn [SPAA 2009])

2-sided error Monte Carlo algorithms

Focus on distributed algorithms that use randomization but whose running time are deterministic.

2-sided error Monte Carlo algorithms

Focus on distributed algorithms that use randomization but whose running time are deterministic.

(p, q) -decider

- ▶ If $(G, x) \in \mathcal{L}$ then, for every identity assignment Id ,
 $\Pr[\text{out}_{\mathcal{A}}(G, x, \text{Id}, v) = \text{"yes"} \text{ for every node } v \in V(G)] \geq p$
- ▶ If $(G, x) \notin \mathcal{L}$ then, for every identity assignment Id ,
 $\Pr[\text{out}_{\mathcal{A}}(G, x, \text{Id}, v) = \text{"no"} \text{ for at least one node } v \in V(G)] \geq q$

Example: AMOM



Example: AMOM



Randomized algorithm

- ▶ every unmarked node says “yes” with probability 1;
- ▶ every marked node says “yes” with probability p .

Example: AMOM



Randomized algorithm

- ▶ every unmarked node says “yes” with probability 1;
- ▶ every marked node says “yes” with probability p .

Remarks:

- ▶ Runs in zero time;
- ▶ If the configuration has at most one marked node then correct with probability at least p .
- ▶ If there are at least $k \geq 2$ marked nodes, correct with probability at least $1 - p^k \geq 1 - p^2$
- ▶ Thus there exists a (p, q) -decider for $q + p^2 \leq 1$.

Bounded-probability error local decision

Definition

$\text{BPLD}(t, p, q)$ is the class of all distributed languages that have a randomized distributed (p, q) -decider running in time at most t .

I.e., can be decided in time at most t by a randomized distributed algorithm with “yes” success probability p and “no” success probability q .

Bounded-probability error local decision

Definition

$\text{BPLD}(t, p, q)$ is the class of all distributed languages that have a randomized distributed (p, q) -decider running in time at most t .

I.e., can be decided in time at most t by a randomized distributed algorithm with “yes” success probability p and “no” success probability q .

Remark

For p and q such that $p^2 + q \leq 1$, there exists a language $\mathcal{L} \in \text{BPLD}(0, p, q)$, such that $\mathcal{L} \notin \text{LD}(t)$, for any $t = o(n)$.

A sharp threshold for hereditary languages

A **prefix** of a configuration (G, x) is a configuration $(G[U], x[U])$, where $U \subseteq V(G)$

Hereditary languages

A language $\mathcal{L}$ is *hereditary* if every prefix of every configuration $(G, x) \in \mathcal{L}$ is also in $\mathcal{L}$.

- ▶ **Coloring** and **AMOM** are hereditary languages.
- ▶ Every language $\{(G, \epsilon) \mid G \in \mathcal{G}\}$ where $\mathcal{G}$ is hereditary is... hereditary. (Examples of hereditary graph families are planar graphs, interval graphs, forests, chordal graphs, cographs, perfect graphs, etc.)

A sharp threshold for hereditary languages

A **prefix** of a configuration (G, x) is a configuration $(G[U], x[U])$, where $U \subseteq V(G)$

Hereditary languages

A language $\mathcal{L}$ is *hereditary* if every prefix of every configuration $(G, x) \in \mathcal{L}$ is also in $\mathcal{L}$.

- ▶ Coloring and AMOM are hereditary languages.
- ▶ Every language $\{(G, \epsilon) \mid G \in \mathcal{G}\}$ where $\mathcal{G}$ is hereditary is... hereditary. (Examples of hereditary graph families are planar graphs, interval graphs, forests, chordal graphs, cographs, perfect graphs, etc.)

Theorem

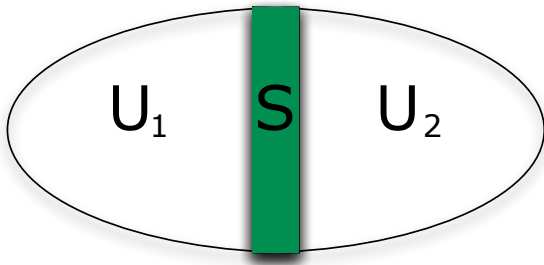
Let $\mathcal{L}$ be an hereditary language and let t be a function of triples (G, x, Id) . If $\mathcal{L} \in BPLD(t, p, q)$ for constants $p, q \in (0, 1]$ such that $p^2 + q > 1$, then $\mathcal{L} \in LD(O(t))$.

One ingredient in the proof

Let $0 < \delta < p^2 + q - 1$, and define $\lambda = 11 \cdot \lceil \log p / \log(1 - \delta) \rceil$.

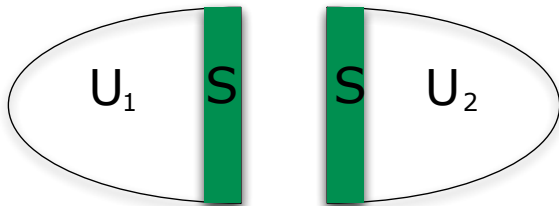
Separating partition

A *separating partition* of (G, x, Id) is a triplet (S, U_1, U_2) of pairwise disjoint subsets of nodes such that $S \cup U_1 \cup U_2 = V$, and $\text{dist}_G(U_1, U_2) \geq \lambda \cdot t$.



Glueing lemma

Given a separating partition (S, U_1, U_2) of (G, x, Id) , let $G_k = G[U_k \cup S]$, and let x_k be the input x restricted to nodes in G_k , for $k = 1, 2$.



Lemma ($\star$)

For every instance (G, x) with identity assignment Id , and every separating partition (S, U_1, U_2) of (G, x, Id) , we have:

$$\left((G_1, x_1) \in \mathcal{L} \text{ and } (G_2, x_2) \in \mathcal{L} \right) \Rightarrow (G, x) \in \mathcal{L}.$$

Remark. Lemma ($\star$) does not use the fact that $\mathcal{L}$ is hereditary, but uses $p^2 + q > 1$.

Derandomization

Deterministic Algorithm $\mathcal{D}$, applied at a node u

Given an instance (G, x) and an id-assignment Id :

If $(B_G(u, 2\lambda t), x[B_G(u, 2\lambda t)]) \in \mathcal{L}$
then $\text{out}(u) = \text{"yes"}$
else $\text{out}(u) = \text{"no"}$

Remark

Nodes do not know t , but this can be fixed.

Correctness (1/4)

Assume $(G, x) \in \mathcal{L}$.

Since $\mathcal{L}$ is hereditary, every prefix of (G, x) is also in $\mathcal{L}$.

Thus, every node u outputs $\text{out}(u) = \text{"yes"}$.

Correctness (1/4)

Assume $(G, x) \in \mathcal{L}$.

Since $\mathcal{L}$ is hereditary, every prefix of (G, x) is also in $\mathcal{L}$.

Thus, every node u outputs $\text{out}(u) = \text{"yes"}$.

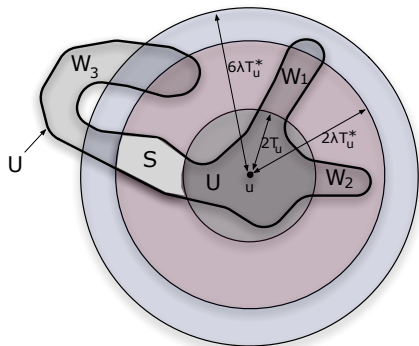
Conversely, assume $(G, x) \notin \mathcal{L}$.

Assume towards contradiction that by applying $\mathcal{D}$ on (G, x, Id) , every node u outputs $\text{out}(u) = \text{"yes"}$.

Let $U \subseteq V(G)$ be a maximal set of vertices such that $G[U]$ is connected and $(G[U], x[U]) \in \mathcal{L}$.

- ▶ U is not empty, as $(B_G(u, 2\lambda t), x[B_G(u, 2\lambda t)]) \in \mathcal{L}$ for every node u .
- ▶ $|U| < |V(G)|$, because $(G, x) \notin \mathcal{L}$.

Correctness (2/4)



Let $u \in U$ be a node such that $B_G(u, 2t)$ contains a node outside U .

Let $G' = G[U \cup V(B_G(u, 2t))]$.

Observe that G' is connected and that G' strictly contains U .

Our goal is to show that $(G', x[G']) \in \mathcal{L}$, for contradiction.

Correctness (3/4)

Let $W^1, W^2, \dots, W^\ell$ be the ℓ connected components of $G[U] \setminus B_G(u, 2t)$, ordered arbitrarily.

Let H denote the maximal graph such that H is connected and

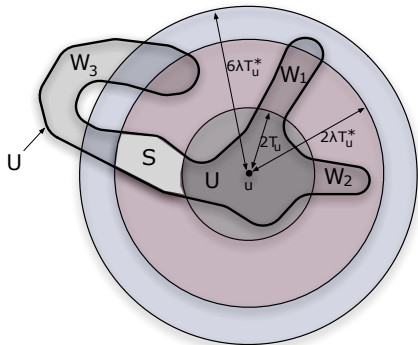
$$B_G(u, 2t) \subset V(H) \subseteq B_G(u, 2t) \cup (U \cap B_G(u, 2\lambda t))$$

Let W^0 be the empty graph, and for $k = 0, 1, 2, \dots, \ell$, define the graph $Z^k = H \cup W^0 \cup W^1 \cup W^2 \cup \dots \cup W^k$.

Observe that Z^k is connected for each $k = 0, 1, 2, \dots, \ell$, and that $Z^\ell = G'$.

We prove by induction on k that $(Z^k, x[Z^k]) \in \mathcal{L}$ for every $k = 0, 1, 2, \dots, \ell$. (This will establish the contradiction since, as we mentioned before, $Z^\ell = G'$).

Correctness (4/4)



Define the sets of nodes

- ▶ $S = V(Z^k) \cap V(W^{k+1})$
- ▶ $U_1 = V(Z^k) \setminus S$
- ▶ $U_2 = V(W^{k+1}) \setminus S$

A crucial observation is that (S, U_1, U_2) is a separating partition of Z^{k+1} .

By induction, we have $(G_1, x[G_1]) \in \mathcal{L}$, because $G_1 = G[U_1 \cup S] = Z^k$.

In addition, we have $(G_2, x[G_2]) \in \mathcal{L}$, because $G_2 = G[U_2 \cup S] = W^{k+1}$, and W^{k+1} is a prefix of $G[U]$.

We can now apply Lemma $(\star)$ and conclude that $(Z^{k+1}, x[Z^{k+1}]) \in \mathcal{L}$.

Outline

Decision problems

Does randomization helps?

Nondeterminism

Power of oracles

Further works

Distributed certification

One motivation

Settings in which one must perform local verifications repeatedly.

- ▶ Proof Labeling Scheme (Korman, Kutten, Peleg [PODC 2007])
- ▶ Distributed verification (Das Sarma et al. [STOC 2011])

Distributed certification

One motivation

Settings in which one must perform local verifications repeatedly.

- ▶ Proof Labeling Scheme (Korman, Kutten, Peleg [PODC 2007])
- ▶ Distributed verification (Das Sarma et al. [STOC 2011])

Definition

An algorithm $\mathcal{A}$ verifies $\mathcal{L}$ if and only if for every configuration (G, x) , the following hold:

- ▶ If $(G, x) \in \mathcal{L}$, then there exists a certificate y such that, for every id-assignment Id , $out_{\mathcal{A}}(G, (x, y), Id, v) = \text{"yes"}$ for all $v \in V(G)$;
- ▶ If $(G, x) \notin \mathcal{L}$, then for every certificate y , and for every id-assignment Id , $out_{\mathcal{A}}(G, (x, y), Id, v) = \text{"no"}$ for at least one node $v \in V(G)$.

Non-determinism helps

Definition

Let t be a function of triplets (G, x, Id) . $\text{NLD}(t)$ is the class of all distributed languages that can be verified in at most t communication rounds.

Example

$\text{Tree} = \{(G, \epsilon) \mid G \text{ is a tree}\} \in \text{NLD}(1)$.

Certificate given at node v is $y(v) = \text{dist}_G(v, \hat{v})$, where $\hat{v} \in V(G)$ is an arbitrary fixed node.

Verification procedure verifies the following:

- ▶ $y(v)$ is a non-negative integer,
- ▶ if $y(v) = 0$, then $y(w) = 1$ for every neighbor w of v , and
- ▶ if $y(v) > 0$, then there exists a neighbor w of v such that $y(w) = y(v) - 1$, and, for all other neighbors w' of v , we have $y(w') = y(v) + 1$.

NLD-complete problem

Reduction

$\mathcal{L}_1$ is **locally reducible** to $\mathcal{L}_2$, denoted by $\mathcal{L}_1 \preceq \mathcal{L}_2$, if there exists a constant time local algorithm $\mathcal{A}$ such that, for every configuration (G, x) and every id-assignment Id , $\mathcal{A}$ produces $\text{out}(v) \in \{0, 1\}^*$ as output at every node $v \in V(G)$ so that

$$(G, x) \in \mathcal{L}_1 \iff (G, \text{out}) \in \mathcal{L}_2 .$$

NLD-complete problem

Reduction

$\mathcal{L}_1$ is **locally reducible** to $\mathcal{L}_2$, denoted by $\mathcal{L}_1 \preceq \mathcal{L}_2$, if there exists a constant time local algorithm $\mathcal{A}$ such that, for every configuration (G, x) and every id-assignment Id , $\mathcal{A}$ produces $\text{out}(v) \in \{0, 1\}^*$ as output at every node $v \in V(G)$ so that

$$(G, x) \in \mathcal{L}_1 \iff (G, \text{out}) \in \mathcal{L}_2 .$$

The language Containment

$x(v) = (\mathcal{E}(v), \mathcal{S}(v))$ where:

- ▶ $\mathcal{E}(v)$ is an element
- ▶ $\mathcal{S}(v)$ is a finite collection of sets

$\{(G, (\mathcal{E}, \mathcal{S})) \mid \exists v \in V, \exists \mathcal{S} \in \mathcal{S}(v) \text{ s.t. } \mathcal{S} \supseteq \{\mathcal{E}(u) \mid u \in V\}\}.$

NLD-complete problem

Reduction

$\mathcal{L}_1$ is **locally reducible** to $\mathcal{L}_2$, denoted by $\mathcal{L}_1 \preceq \mathcal{L}_2$, if there exists a constant time local algorithm $\mathcal{A}$ such that, for every configuration (G, x) and every id-assignment Id , $\mathcal{A}$ produces $\text{out}(v) \in \{0, 1\}^*$ as output at every node $v \in V(G)$ so that

$$(G, x) \in \mathcal{L}_1 \iff (G, \text{out}) \in \mathcal{L}_2 .$$

The language Containment

$x(v) = (\mathcal{E}(v), \mathcal{S}(v))$ where:

- ▶ $\mathcal{E}(v)$ is an element
- ▶ $\mathcal{S}(v)$ is a finite collection of sets

$\{(G, (\mathcal{E}, \mathcal{S})) \mid \exists v \in V, \exists S \in \mathcal{S}(v) \text{ s.t. } S \supseteq \{\mathcal{E}(u) \mid u \in V\}\}$.

Theorem

Containment is **NLD($O(1)$)-complete**.

Combining non-determinism with randomization

Let $\text{BPNLD}(t) = \cup_{p^2+q \leq 1} \text{BPNLD}(t, p, q)$.

Combining non-determinism with randomization

Let $\text{BPNLD}(t) = \cup_{p^2+q \leq 1} \text{BPNLD}(t, p, q)$.

Theorem

$\text{BPNLD}(O(1))$ contains all languages.

Combining non-determinism with randomization

Let $\text{BPNLD}(t) = \cup_{p^2+q \leq 1} \text{BPNLD}(t, p, q)$.

Theorem

$\text{BPNLD}(O(1))$ contains all languages.

Proof

The certificate is a map of the graph, i.e., an isomorphic copy H of G , with nodes labeled from 1 to n .

Each node v is also given its label $\ell(v)$ in H .

The proof that nodes can probabilistically check $H \sim G$ relies on two facts:

- ▶ To be “cheated”, a wrong map must be a lift of G .
- ▶ One can check whether H is a lift of G by having node(s) labeled 1 acting as in AMOM.



The “most difficult” decision problem

The problem **Cover**

$\{(G, (\mathcal{E}, \mathcal{S})) \mid \exists v \in V, \exists S \in \mathcal{S}(v) \text{ s.t. } S = \{\mathcal{E}(u) \mid u \in V\}\}.$

Theorem

Cover is ***BPNLD**($O(1)$)-complete*.

Outline

Decision problems

Does randomization helps?

Nondeterminism

Power of oracles

Further works

Numerous examples in the literature for which the knowledge of the size of the network is required to efficiently compute solutions.

The oracle GraphSize

Numerous examples in the literature for which the knowledge of the size of the network is required to efficiently compute solutions.

$$\text{GraphSize} = \{(G, k) \text{ s.t. } |V(G)| = k\}.$$

Theorem

For every language $\mathcal{L}$, we have $\mathcal{L} \in \text{NLD}^{\text{GraphSize}}$.

Proof

Certificate is the **map** of G . (Cannot be “cheated” whenever the nodes know the number of nodes).

Outline

Decision problems

Does randomization helps?

Nondeterminism

Power of oracles

Further works

Further works

- ▶ Connections between classical computational complexity theory and the local complexity one, e.g., by putting **constraints on individual computation time**.

Further works

- ▶ Connections between classical computational complexity theory and the local complexity one, e.g., by putting **constraints on individual computation time**.
- ▶ Also, one could restrict the **memory** used by a node, in addition to, or instead of, bounding the sequential time.

Further works

- ▶ Connections between classical computational complexity theory and the local complexity one, e.g., by putting **constraints on individual computation time**.
- ▶ Also, one could restrict the **memory** used by a node, in addition to, or instead of, bounding the sequential time.
- ▶ Complexity framework taking also **traffic congestion** into account. (This can be done by, e.g., considering the *CONGEST* model).

Thank You!