Fully Dynamic Maintenance of Arc-Flags in Road Networks

Gianlorenzo D'Angelo[1]

Mattia D'Emidio[2]
Daniele Frigioni[2]
Camillo Vitale[2]

[1]MASCOTTE project, INRIA, France.

[2]Università degli Studi dell'Aquila, Italy.

Static point-to-point routing in road networks

Problem

Find the shortest or quickest route from a source to a target location in a road network

Using Dijkstra's algorithm for point-to-point shortest paths results to slow in large networks

Solution

In the last years many speed-up techniques for Dijkstra's algorithm for point-to-point shortest paths have been developed
When applied to road networks they are millions of times faster than the basic Dijkstra's algorithm

Static point-to-point routing in road networks

Most of the speed-up techniques divide the computation into two steps:

Preprocessing where some information are precomputed in advance before knowing the source and target locations

- performed off-line only once
- may require long computational time and large storage

Query where the above computed information are used to accelerate a source-target query

- performed on-line, every time that a source-target query is asked
- fast and space efficient

Dynamic point-to-point routing in road networks

- When a change occurs in the network, the preprocessed information need to be computed again
- We cannot perform on-line the preprocessing from-scratch as it is too slow
- There is a need for dynamic algorithms which efficiently update preprocessed information

Dynamic point-to-point routing in road networks – Related work

- Many algorithms have been developed in the latest years to efficiently update the preprocessed information of specific speed-up technique
  - Geometric Containers [Wagner et al. 2005]
  - Highway hierarchies and overlay graphs [Sanders and Schultes 2007]
  - ALT [Delling and Wagner 2007]
  - Arc-Flags in Time dependent graphs [Berger et al. 2010]
  - Overlay graphs [Delling et al. 2011]
- Some of the dynamic solutions known only find relaxed preprocessed information (i.e. the queries remain correct but the speed-up gets worse compared to a from-scratch approach)

In this work

We present a dynamic algorithm for a speed-up technique called Arc-Flags

- it is fast
- the updated information are equivalent to the from-scratch approach

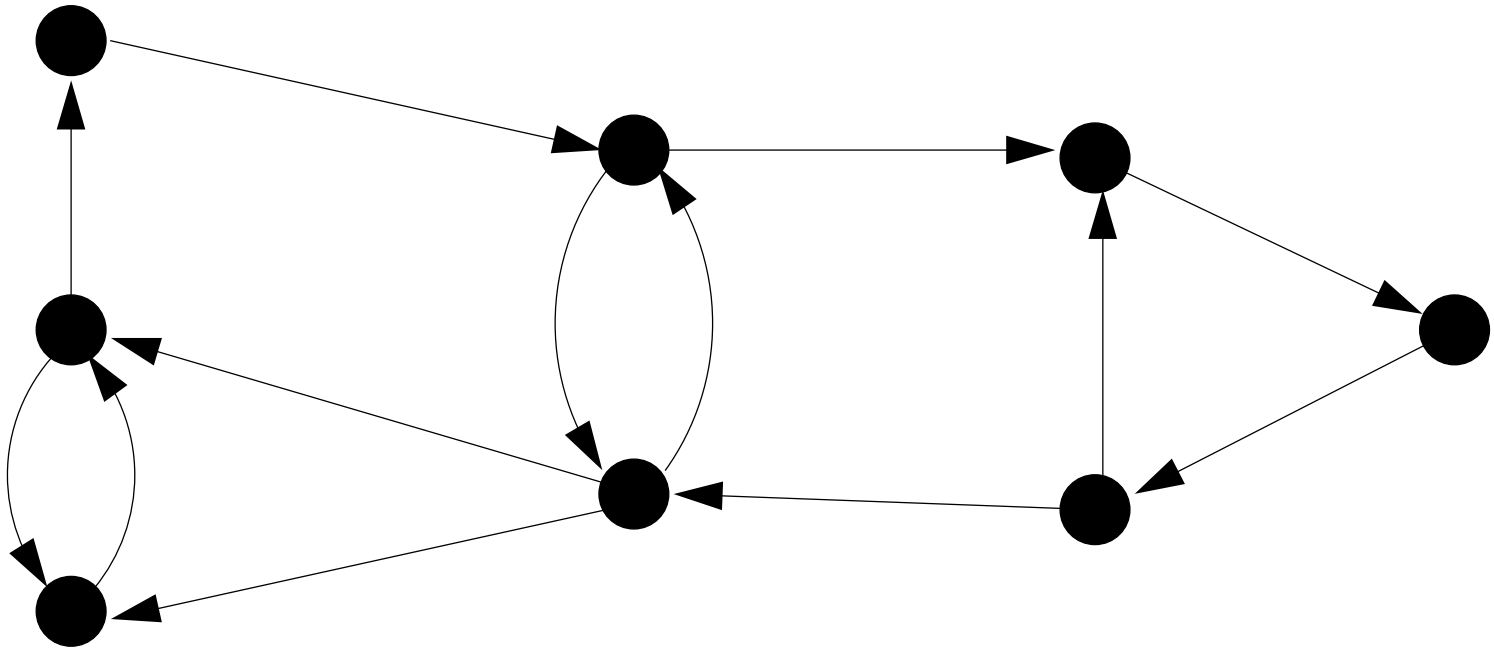
Arc-Flags is one of the most important speed-up technique because it can be combined with other ones to get even more speed-up
(see Bauer et al. *Combining hierarchical and goal-directed speed-up techniques for Dijkstra's algorithm* ACM JEA 2010)

Outline

Outline

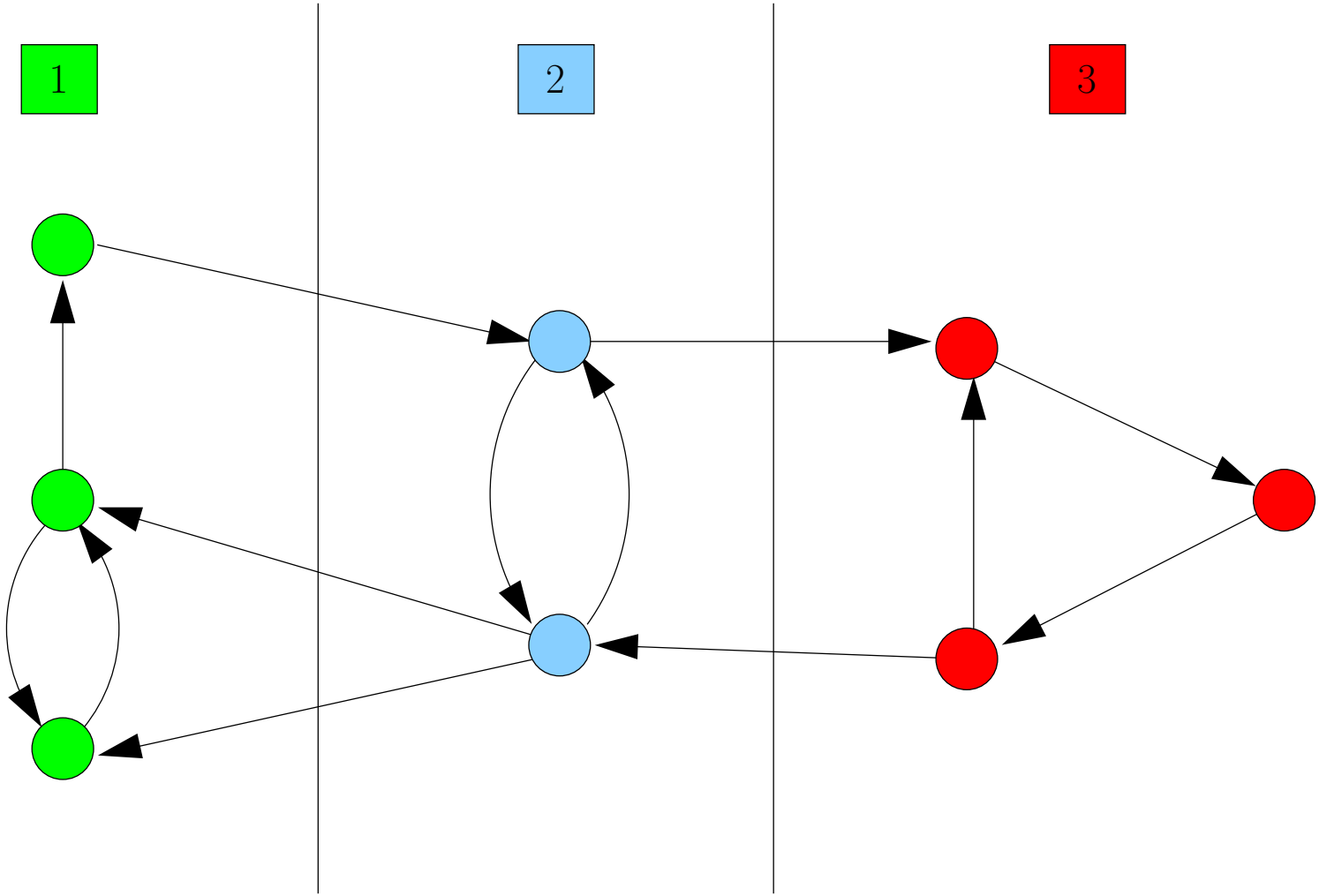
Static Arc-Flags

Take a weighted graph



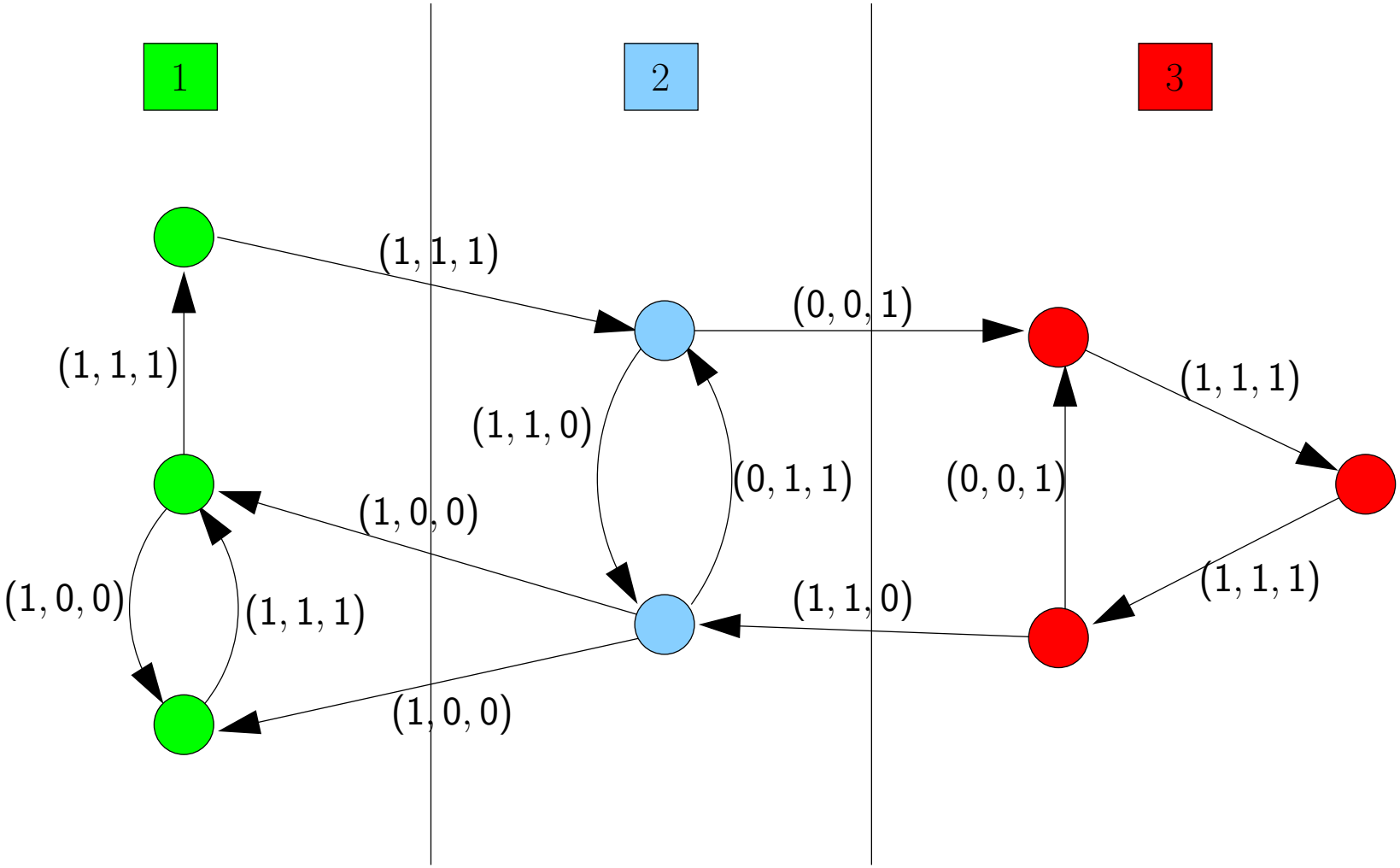
Static Arc-Flags

Preprocessing: Partition the graph into regions



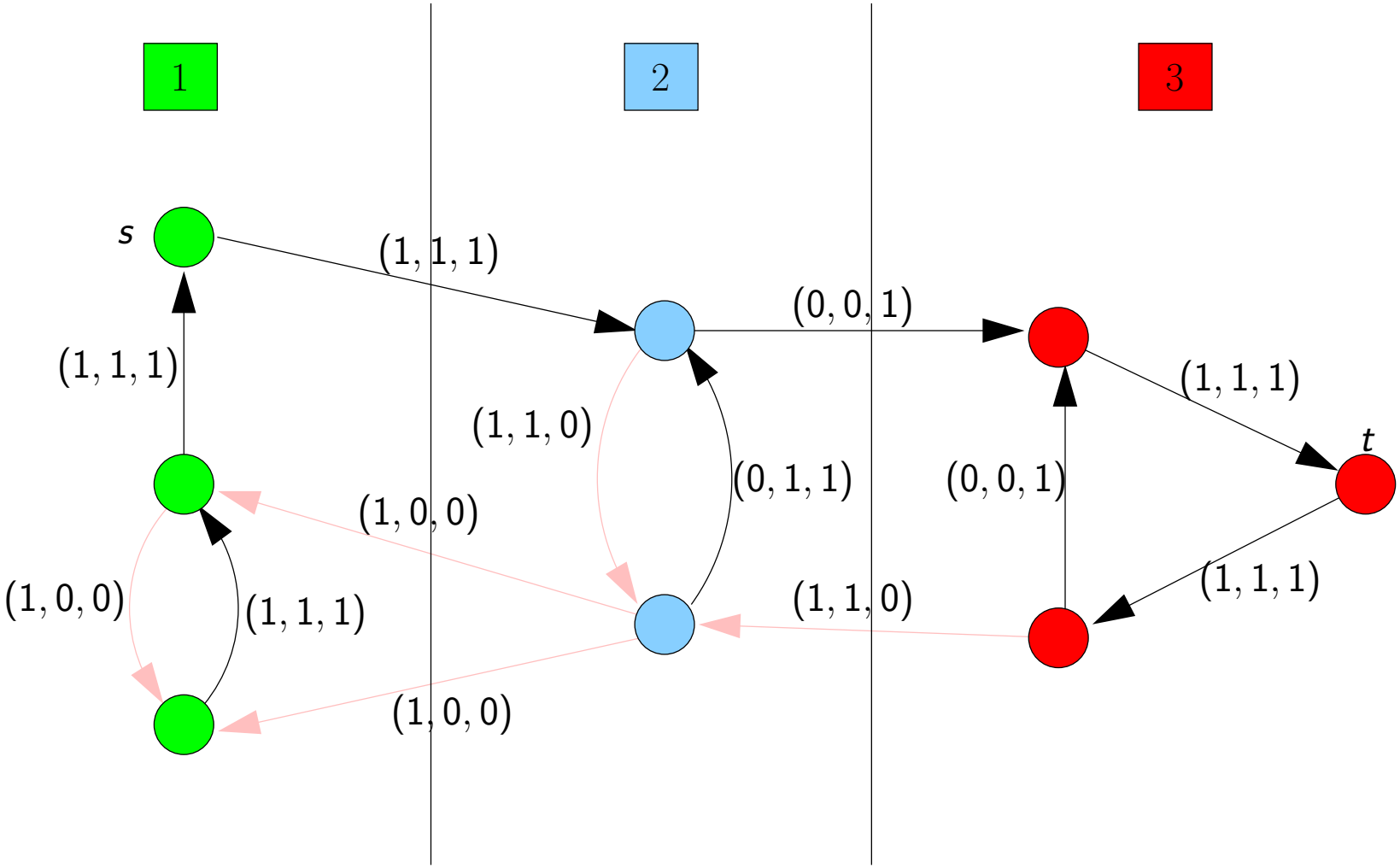
Static Arc-Flags

Preprocessing: For each edge e and for each region R associate a flag which is true if and only if e is on a shortest path towards R



Static Arc-Flags

s-*t* **query:** Use Dijkstra's algorithm only on edges with a flag set to true for the region where *t* belongs to

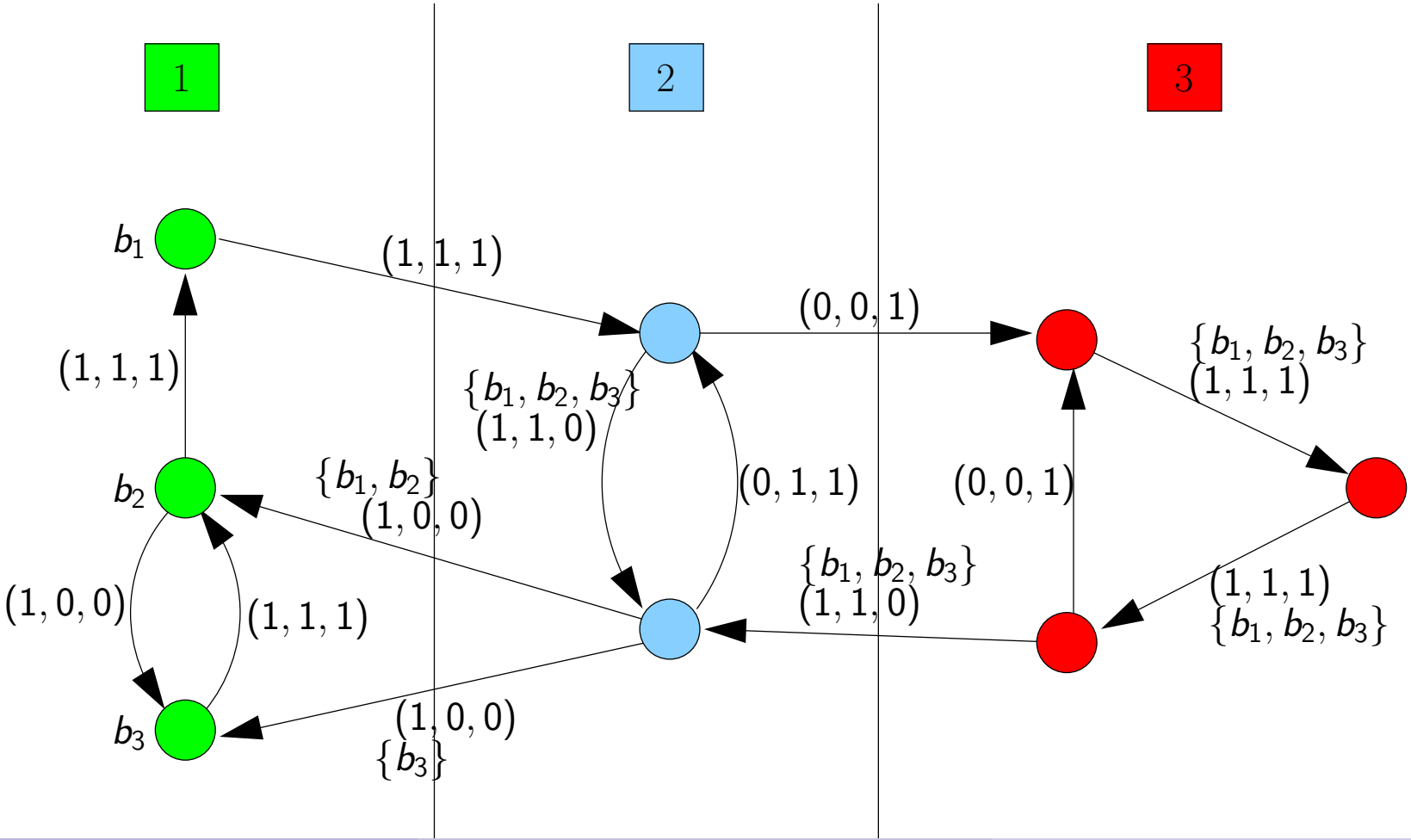


Outline

- It is difficult to update Arc-Flags because a bit set to 1 encodes many shortest paths
- In a previous work we showed how to update sub-optimal Arc-Flags: we only switch flags from 0 to 1 but this decreases the speed-up
- In another previous work (SEA2011) we gave an algorithm which correctly update Arc-Flags when a weight increase occurs
- We now give an algorithm that uses the same data structure as the previous algorithm to correctly update Arc-Flags when a weight decrease occurs
- The two algorithms are combined to obtain a fully dynamic algorithm

Data Structure

For each edge $e = (u, v)$ and region R_k, the Road-Sign $S_k(u, v)$ of e to R_k is the set of boundary nodes b of R_k such that a shortest path from u to b contains e



Data Structure

- For each edge $e = (u, v)$ and region R_k, the Arc-Flag of e towards R_K is true if and only if both u and v belong to R_k or $S_k(u, v) \not\equiv \emptyset$
- Road signs can be computed during the preprocessing phase of Arc-Flags
- Road signs require $O(|E||B|)$ space, where B is the number of boundary nodes, but we will show later how to store them in almost linear space

Incremental algorithm

The incremental algorithm is performed for each region R_k and works in two phases:

1. EdgeUpdate
2. ForwardUpdate

Incremental algorithm – EdgeUpdate

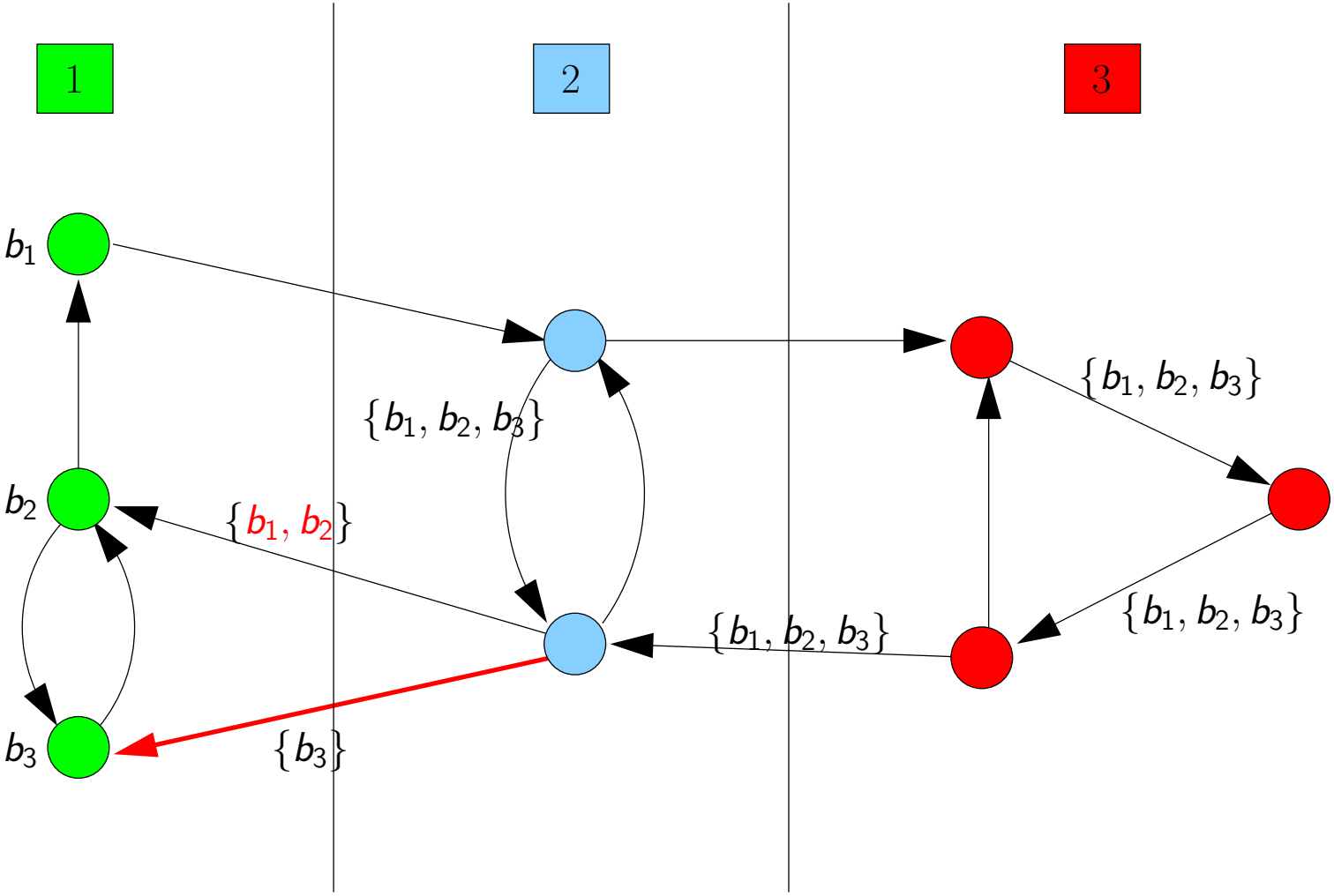
If a weight decrease operation occurs on the edge (x_i, y_i):

1. compute the set $\bar{B}_k(x_i)$ as:

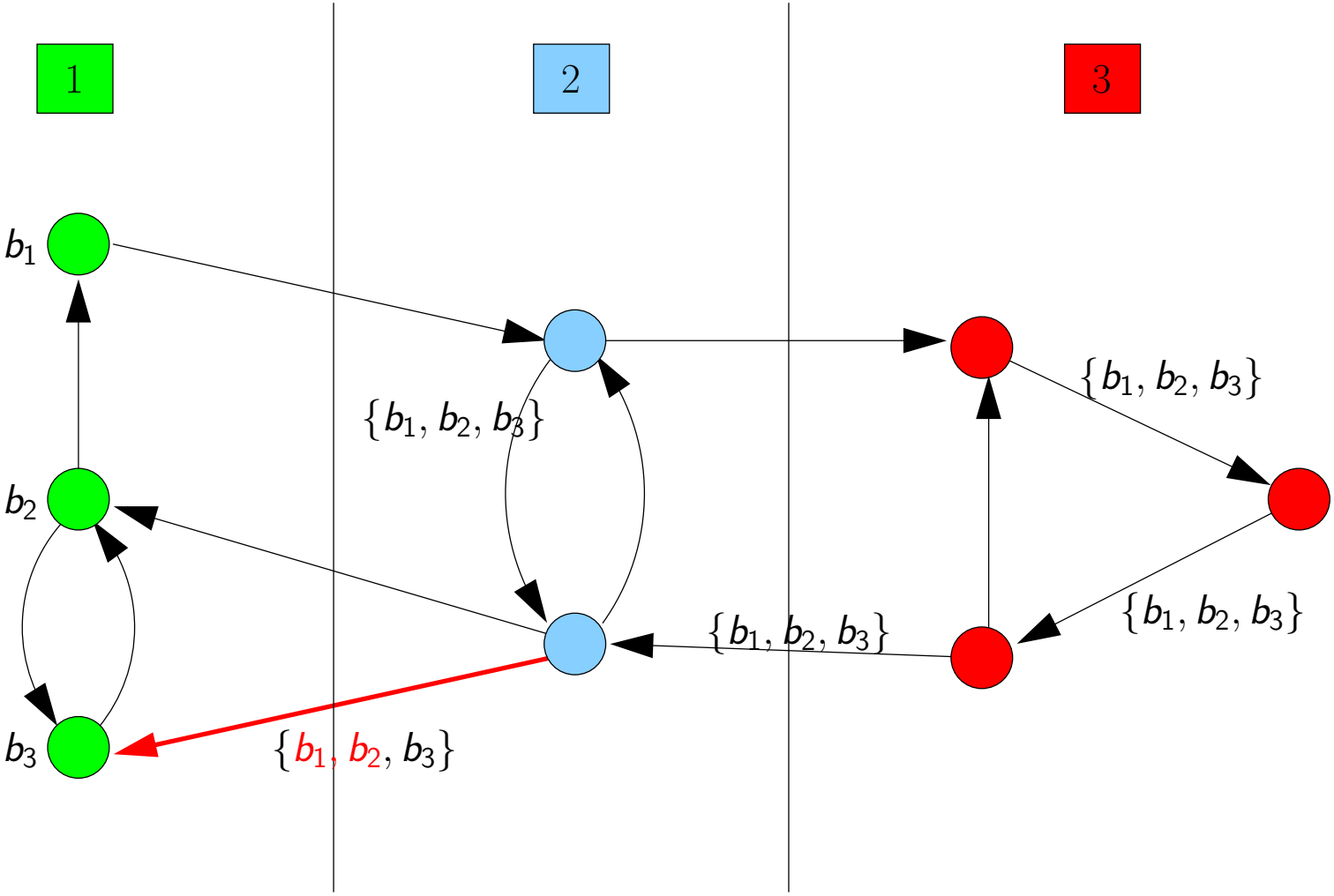
$$\bar{B}_k(x_i) = B(R_k) \setminus S_k(x_i, y_i)$$

2. for each $b \in \bar{B}_k(x_i)$:
   - let $u \in N_o(x_i)$ the node such that $b \in S_k(x_i, u)$
   - compute the shortest path distance from u to b
   - compute the shortest path distance from y_i to b
   - if the shortest path to b passing through y_i is better, move b from $S_k(x_i, u)$ to $S_k(x_i, y_i)$

Incremental algorithm – EdgeUpdate



Incremental algorithm – EdgeUpdate



Incremental algorithm – ForwardUpdate

For each $b_i \in S_k(x_i, y_i)$:

1. compute the shortest path distance from x_i to b_i
2. insert x_i into a priority queue
3. greedily extract min-priority nodes and relax ingoing edges
4. switch b_i, where needed

Compacting Road-Signs

$B(R_k)$: the set of boundary nodes of R_k
For each node u:

1. $B(R_k) = \bigcup_{(u,v)\in E} S_k(u,v)$
2. $S_k(u,v_1) \cap S_k(u,v_2) = \emptyset$, for each $v_1 \neq v_2$ such that $(u,v_1) \in E$ and $(u,v_2) \in E$

For each node u, take and arbitrary edge (u,v),

$$S_k(u,v) = B(R_k) \setminus \bigcup_{(u,v')\in E, v'\neq v} S_k(u,v')$$

We store only $|E| - |V|$ Road-Signs and the space occupancy is $O((|E| - |V|) \cdot |B|)$

Outline

Input instances: Graphs

graph	n. of nodes	n. of edges	%mot	%nat	%reg	%urb
LUX	30 647	75 576	0.55	1.95	14.77	82.71
DNK	469 110	1 090 148	24.02	3.06	0.48	72.45
BEL	458 403	1 164 046	22.90	2.92	0.52	73.62
AUT	722 512	1 697 902	27.60	5.33	1.71	65.21
ESP	695 399	1 868 838	33.22	6.34	1.51	58.87
NED	892 027	2 278 824	0.40	0.56	5.16	93.86
SWE	1 546 705	3 484 378	19.54	2.86	0.45	77.10

Input instances: Partitioning

G. Karypis. METIS - A Family of Multilevel Partitioning Algorithms, 2007.

graph	nodes	edges	regions	boundaries
LUX	30 647	75 576	64	1215
DNK	469 110	1 090 148	128	5898
BEL	458 403	1 164 046	128	6595
AUT	722 512	1 697 902	128	6294
ESP	695 399	1 868 838	128	5260
NED	892 027	2 278 824	128	6427
SWE	1 546 705	3 484 378	128	7962

Input instances: Sequences of operations

Fully dynamic sequences of updates simulating real world road networks behavior, as, for example:

Disruption: most significant operation on a road segment: the increase of the weight, which simulates a delay in the travel time on that segment due, for instance, to a traffic jam

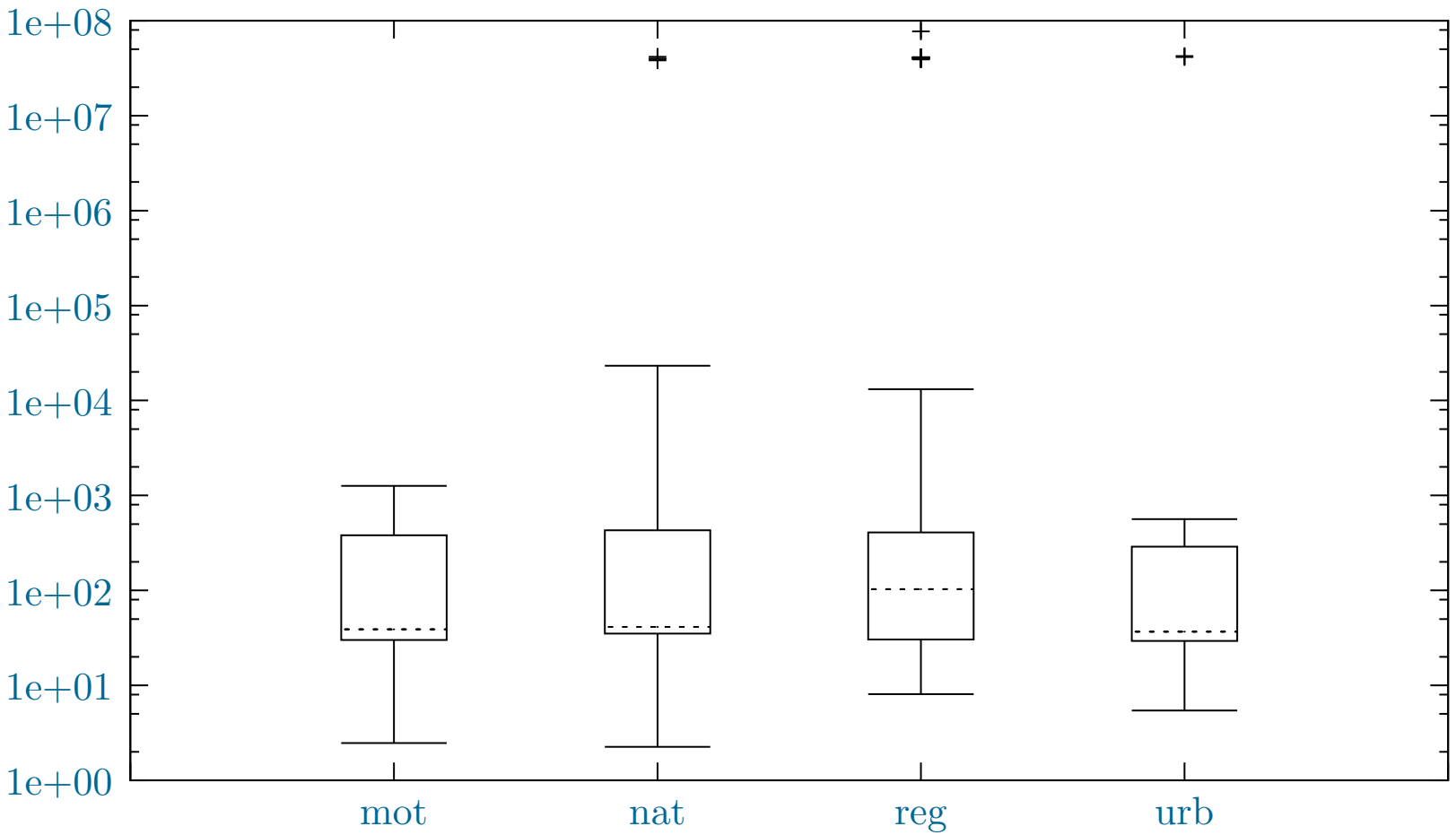
Recovery: usually this operation is followed by a weight decrease on the same road segment which simulates the restore from the delay

Input instances: Sequences of operations

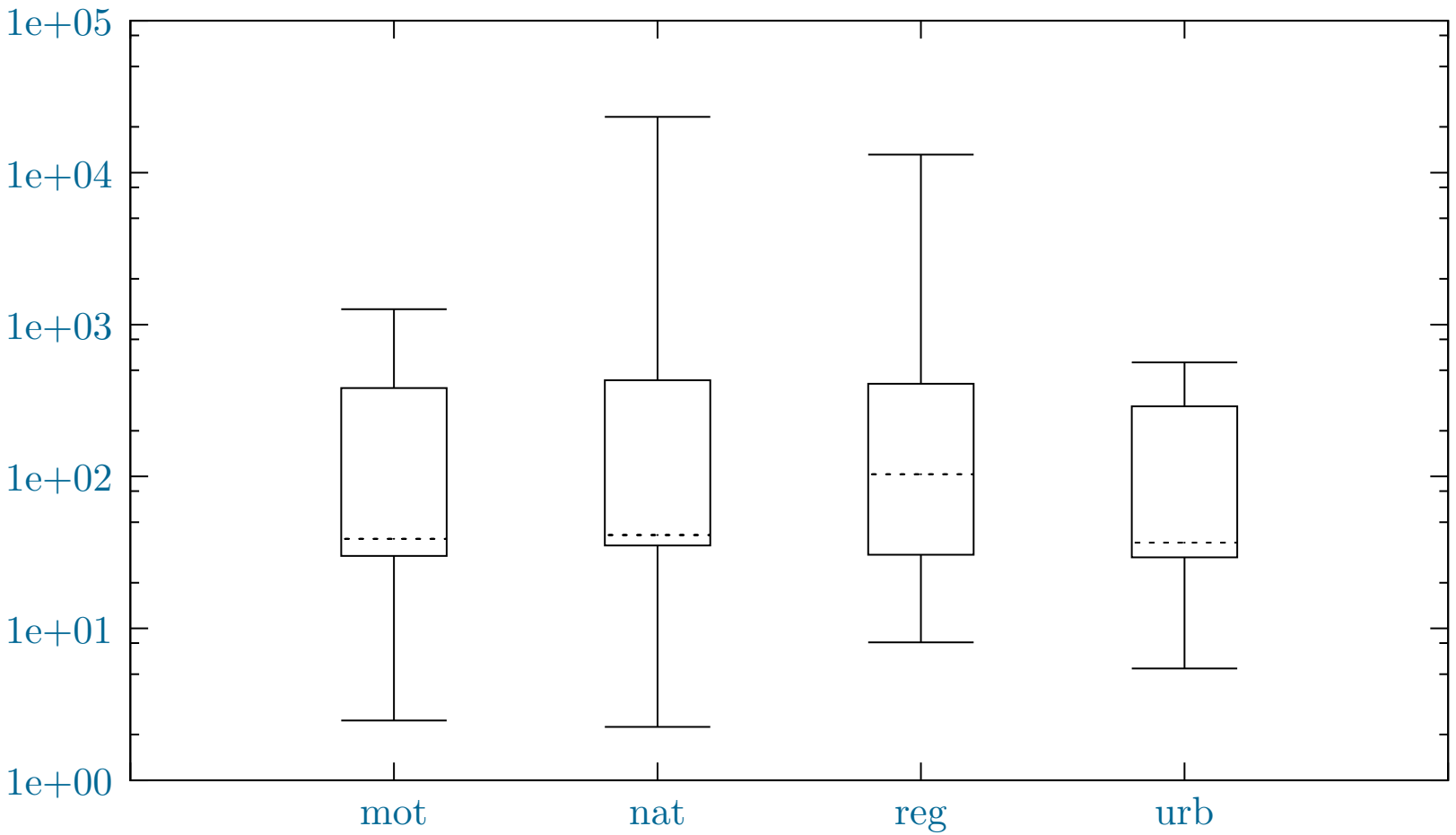
We performed simulation on sequences of 100 weight change operations as follows:

- for each operation in the sequence that increases the weight of an edge (x_i, y_i) of a quantity γ_i, there is a corresponding subsequent operation which decreases the weight of edge (x_i, y_i) of the same amount γ_i.
- $\gamma_i = x \cdot w(x_i, y_i)$, where $x \in [25\%, 75\%]$
- we measured the execution time of the dynamic algorithm
- we computed the ratio between the execution time of a from-scratch approach and the dynamic algorithm (speed-up)

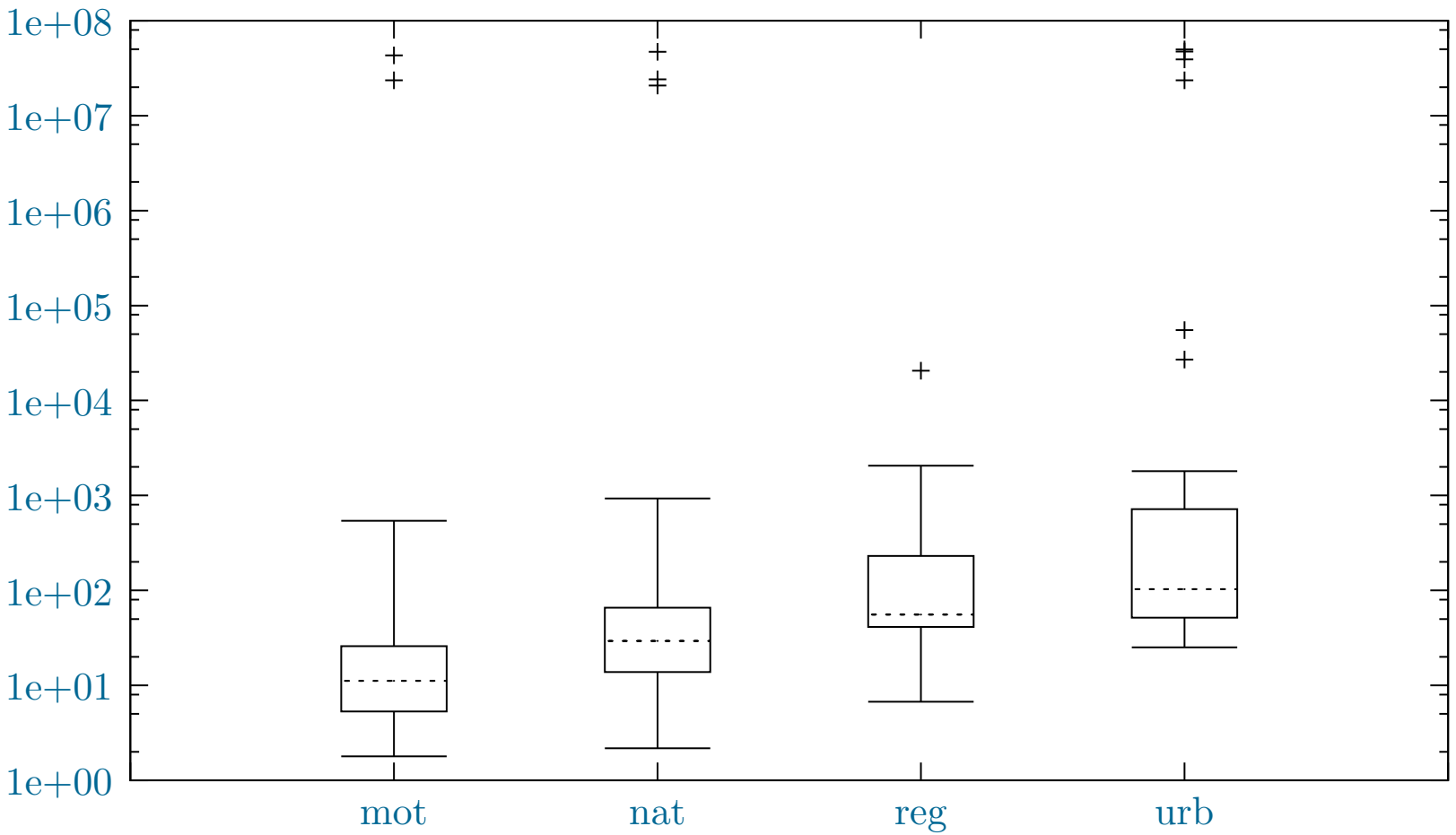
DynRS – swe



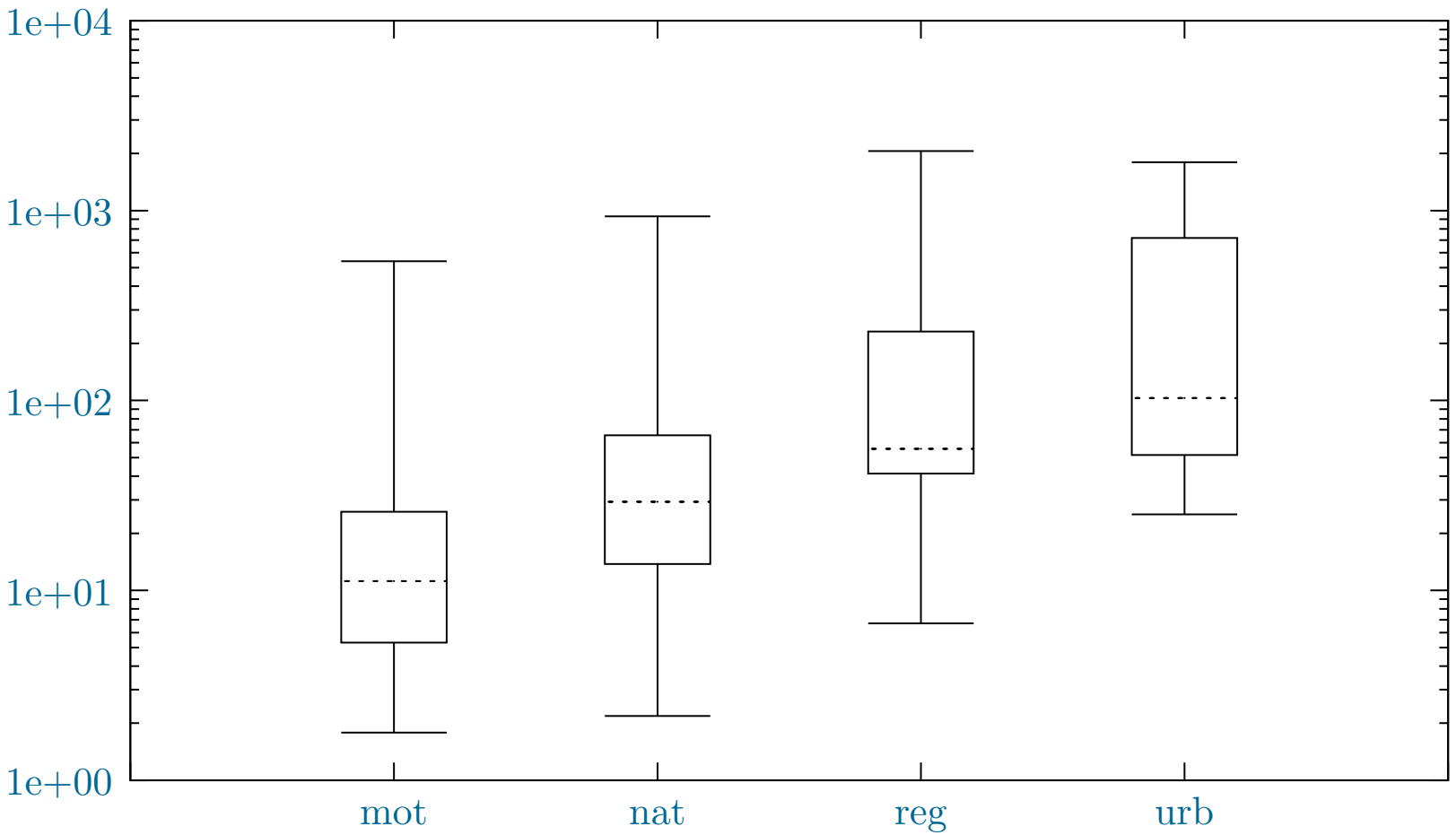
DynRS – swe



DynRS – ned



DynRS – ned



Average time and speed-up

graph	cat.	avg. **Arc-Flags** (sec)		avg. time DynRS(sec)		**speed-up**	
LUX	mot	5.50	5.52	2.28	1.57	11.09	40.32
	nat	5.51		2.64		32.81	
	reg	5.56		0.99		24.79	
	urb	5.52		0.38		92.60	
DNK	mot	542.55	542.46	17.05	14.90	449.10	431.41
	nat	542.83		11.24		430.16	
	reg	542.22		16.16		330.64	
	urb	542.25		15.13		515.72	
BEL	mot	644.48	644.34	23.16	23.73	195.53	238.23
	nat	644.30		28.93		195.81	
	reg	644.31		28.00		421.83	
	urb	644.28		14.85		139.73	
AUT	mot	935.16	940.83	39.31	26.25	108.12	166.10
	nat	934.16		19.08		185.48	
	reg	956.62		27.70		183.09	
	urb	937.38		18.89		187.71	
ESP	mot	736.81	737.31	22.21	21.41	392.19	373.98
	nat	737.30		21.44		562.27	
	reg	736.50		24.46		264.78	
	urb	736.32		17.54		276.66	
NED	mot	1 607.36	1 606.29	206.36	90.55	31.45	169.79
	nat	1 606.22		107.27		107.20	
	reg	1 609.60		30.85		181.84	
	urb	1 601.96		17.71		358.67	
SWE	mot	2 681.54	2 681.16	113.90	76.12	180.36	316.64
	nat	2 681.94		68.98		519.96	
	reg	2 678.81		52.79		394.99	
	urb	2 682.34		68.82		171.24	

Preprocessing time

graph	#Regions	AF (sec)	AF+RS (sec)	Ratio
LUX	64	5.52	11.79	2.14
DNK	128	542.46	1 104.16	2.04
BEL	128	644.34	1 373.11	2.13
AUT	128	940.83	1 995.93	2.12
ESP	128	737.31	1 446.65	1.96
NED	128	1 606.29	3 214.34	2.00
SWE	128	2 681.16	5 986.62	2.23

Preprocessing space

graph	#Regions	AF (B)	AF+RS (B)	Ratio
LUX	64	1 209 216	1 744 531	1.44
DNK	128	17 442 368	43 932 508	2.52
BEL	128	18 624 736	64 759 010	3.48
AUT	128	27 166 432	78 720 055	2.90
ESP	128	29 901 408	67 664 666	2.26
NED	128	36 461 184	64 612 836	1.77
SWE	128	55 750 048	163 151 588	2.93

Outline

Conclusions and future Work

In this paper we introduced a new algorithm which:

- Allows us to efficiently update the Arc-Flags in a fully dynamic scenario
- Does not introduce loss in query performance

Future work:

- Combine with other dynamic speed-up techniques
- Implement parallel algorithms
- Compare with other preprocessing for arc flags which use parallel processing

Thank you for your attention