Placing Arrows in Directed Graph Drawings

Carla Binucci[1], Markus Chimani[2], Walter Didimo[1], Giuseppe Liotta[1], Fabrizio Montecchiani[1]

[1] University of Perugia [2]Osnabrück University

Some preliminary considerations

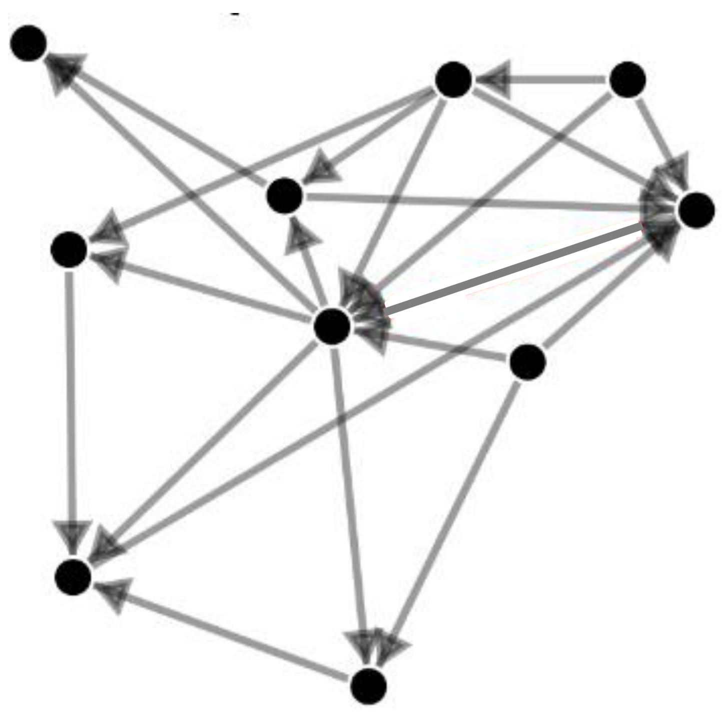
- Directed graphs are used in many application domain.

- Usually a directed edge is represented as a line with an arrow head at its target.

- This is the prevailing model in software systems.

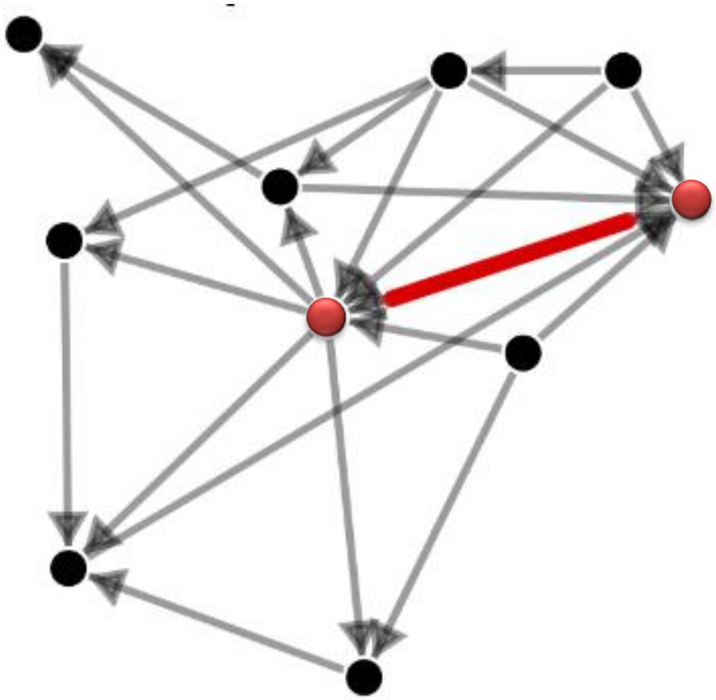
The Problem

This simple model becomes problematic when several edges attach to a vertex on a similar trajectory



The Problem

This simple model becomes problematic when several edges attach to a vertex on a similar trajectory



Our goals

Computing a placement of the arrow heads such that:

(a) They do not overlap other edges or arrow heads.

(b) They are as close as possible to the target vertices of the edges.

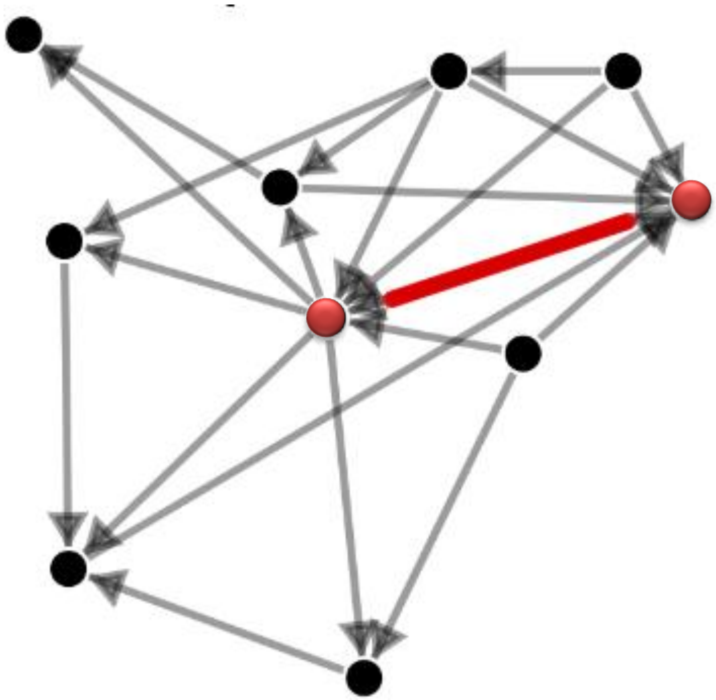
Our Contribution

- Problem formulation & NP-hardness.

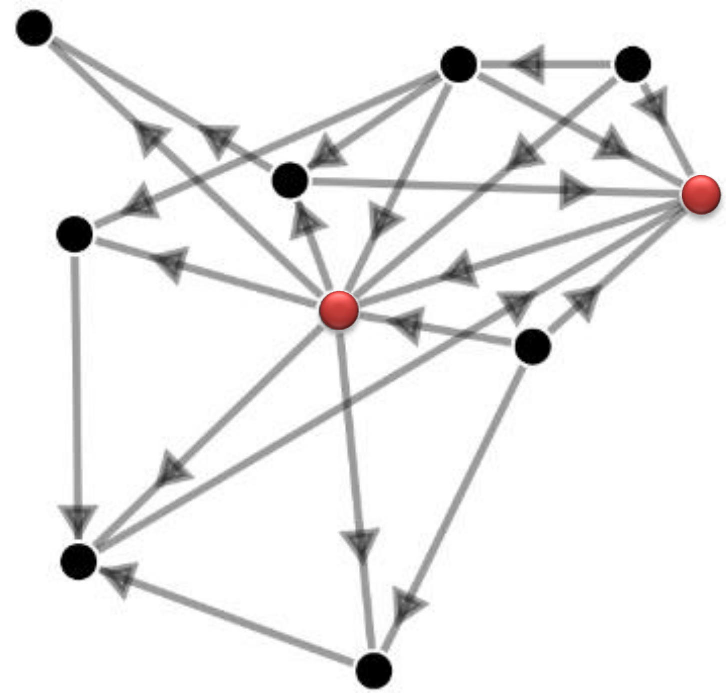
- Exact and heuristic algorithms for a discretized version of the problem.

- A preliminary experimental study.

Example of drawings

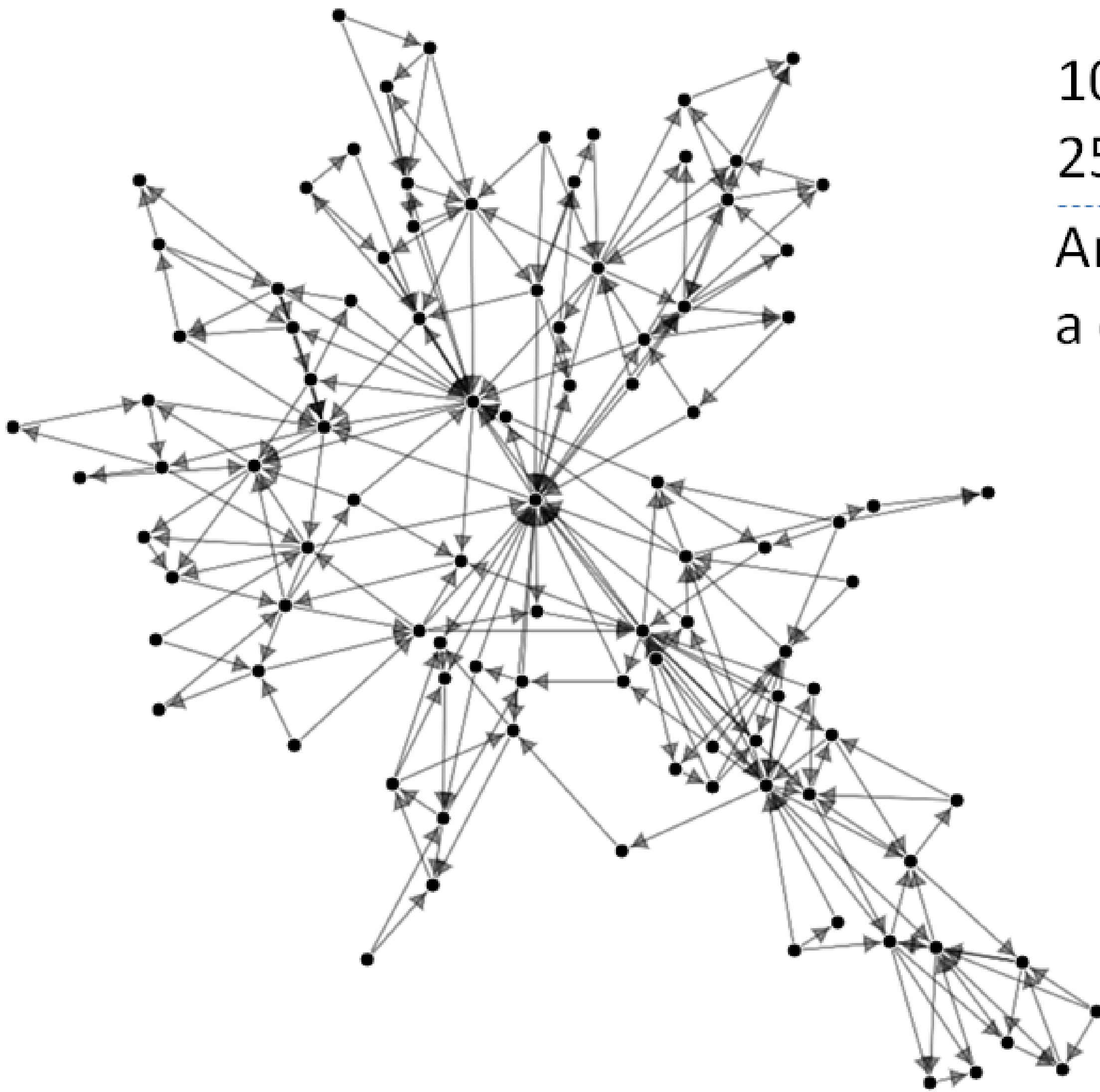


Arrows placed by
a common editor



Arrows placed by
our exact method

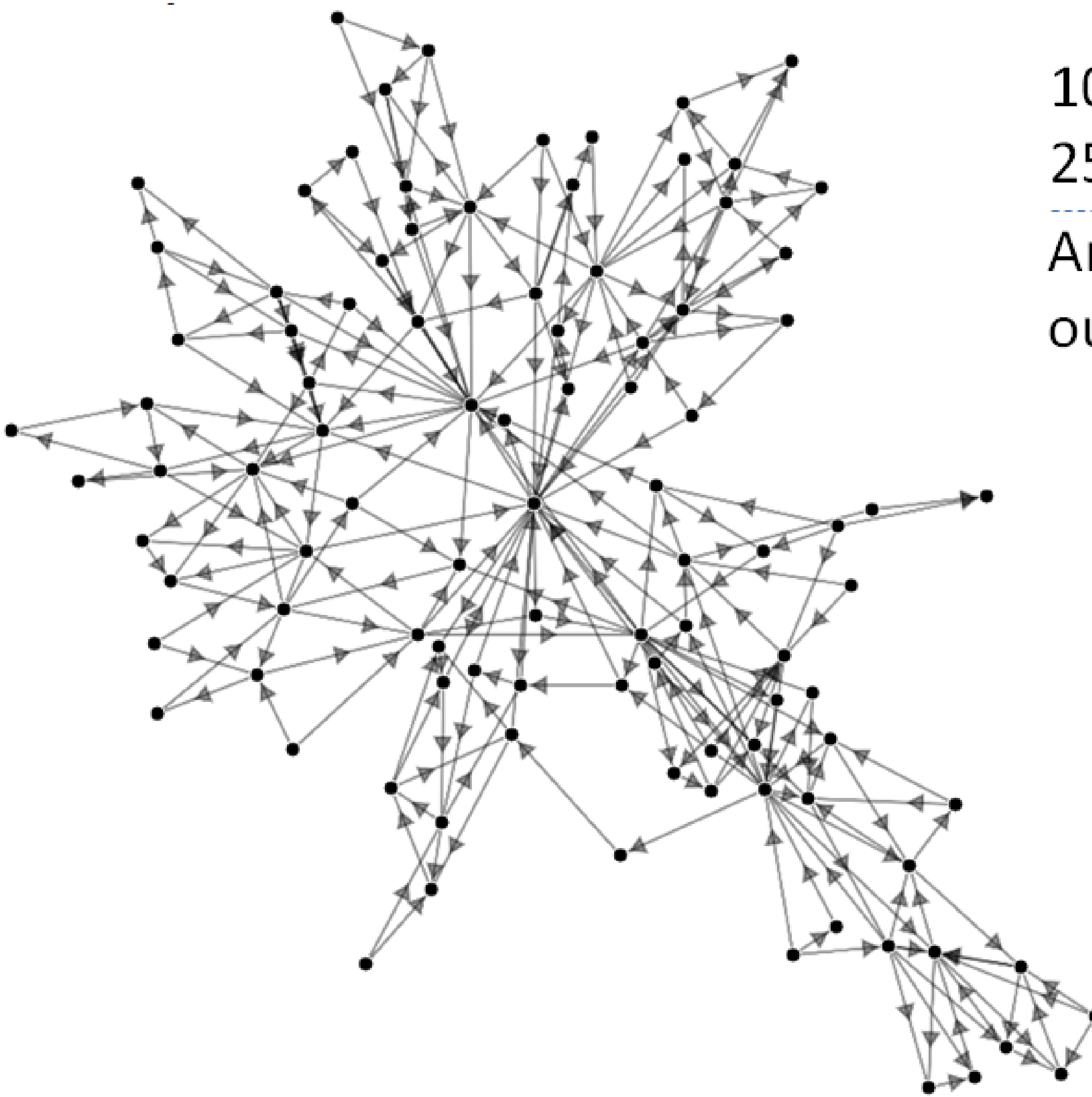
Larger examples



100 vertices
250 edges

Arrows placed by
a common editor

Larger examples



100 vertices
250 edges

Arrows placed by our exact method

Related Works

- User studies on the readability of directed-edge representations
  - [*Holten and van Wijk*, 2009] [*Holten et al.*, 2011].

- Map labeling problems and in particularly edge labeling problems
  - [*Kakoulis and Tollis*, 2001, 2003, 2006, 2013], [*Gemsa et al.*, 2013], [*Gemsa et al.*, 2014], [*van Kreveld et al.*, 1999], [*Marks and Shieber*, 1991], [*Strijk and van Kreveld*, 2002], [*Strijk and Wolff*, 2001], [*Wagner et al.*, 2001]…….

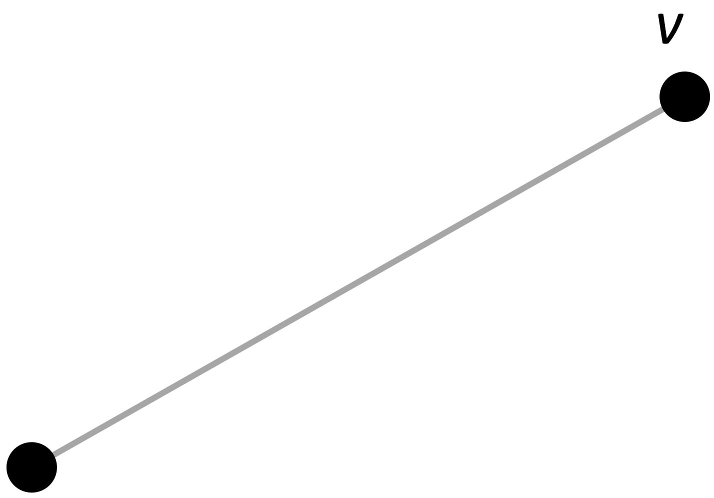
- Research on this topic started at Dagstuhl with the valuable contribution of Michael Kaufmann and Dorothea Wagner
  - [*Dagstuhl seminar 15052*, 2015].

....In what follows....

- **Problem formulation & NP-hardness**
- Algorithms
- Experiments

Modeling arrow heads

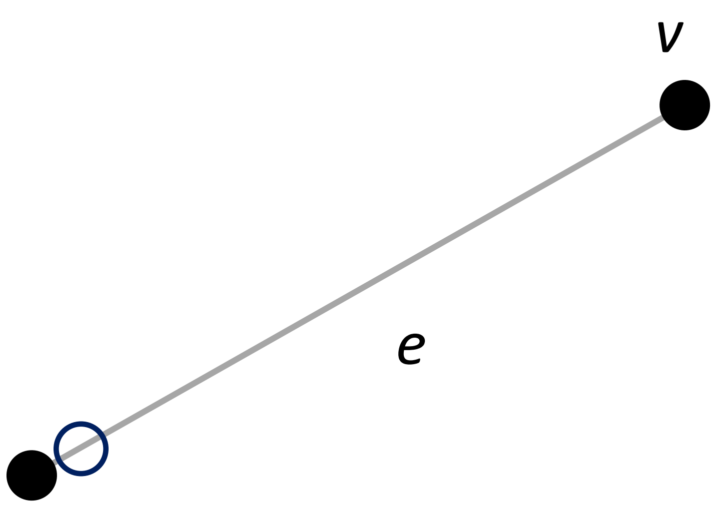
Consider a straight-line drawing Γ of a digraph $G = (V, E)$:



- Each vertex v is drawn as a circle (possibly a point).

Modeling arrow heads

Consider a straight-line drawing *Γ* of a digraph *G* = (*V*, *E*):

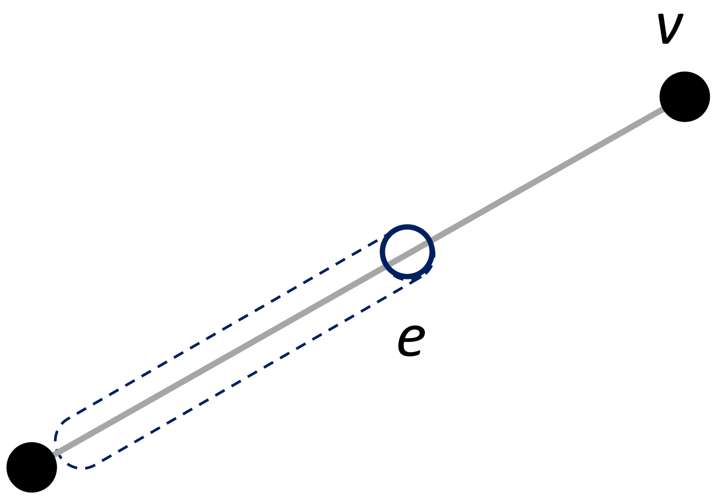


- Each vertex *v* is drawn as a circle (possibly a point).

- We model an arrow of an edge *e* as a circle of radius r_E centered in a point along *e*

Modeling arrow heads

Consider a straight-line drawing *Γ* of a digraph *G* = (*V*, *E*):

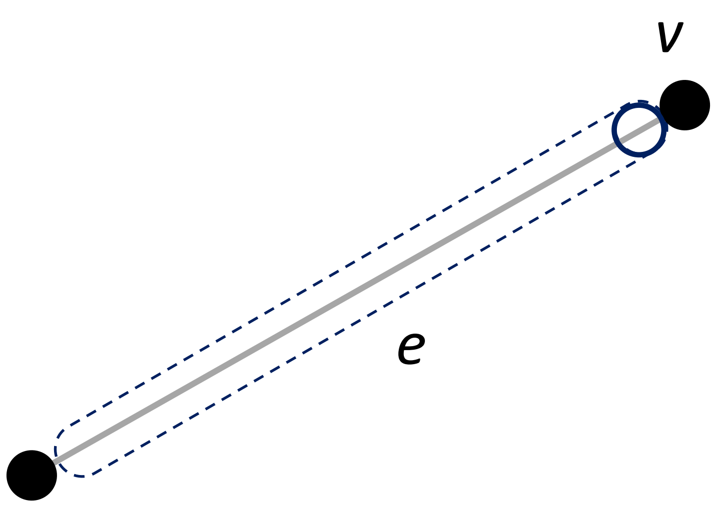


- Each vertex *v* is drawn as a circle (possibly a point).

- We model an arrow of an edge *e* as a circle of radius r_E centered in a point along *e*

Modeling arrow heads

Consider a straight-line drawing *Γ* of a digraph *G* = (*V*, *E*):

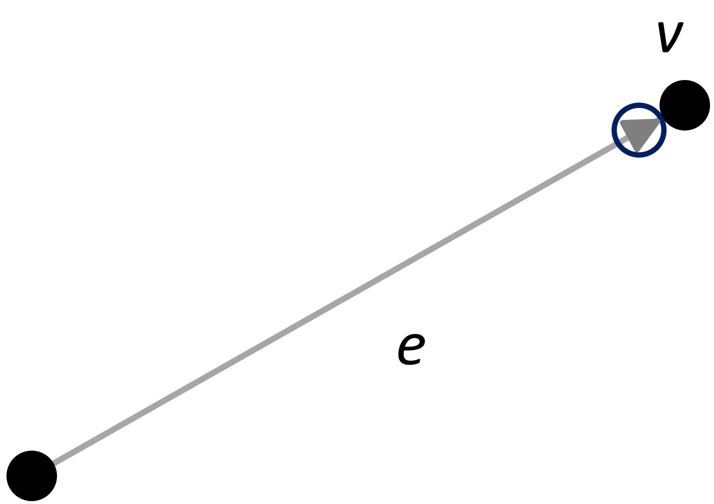


- Each vertex *v* is drawn as a circle (possibly a point).

- We model an arrow of an edge *e* as a circle of radius r_E centered in a point along *e*

Modeling arrow heads

Consider a straight-line drawing *Γ* of a digraph *G* = (*V*, *E*):



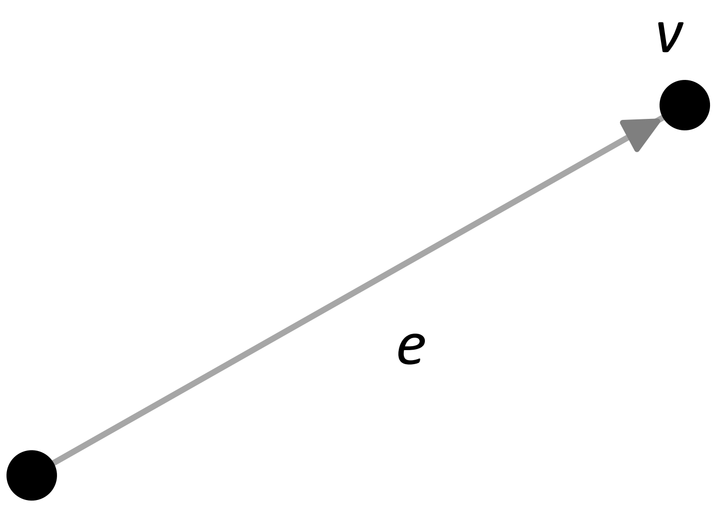
- Each vertex *v* is drawn as a circle (possibly a point).

- We model an arrow of an edge *e* as a circle of radius r_E centered in a point along *e*

- When Γ is displayed, the arrow of *e* is drawn as a triangle inscribed in the circle.

Modeling arrow heads

Consider a straight-line drawing Γ of a digraph $G = (V, E)$:



- Each vertex v is drawn as a circle (possibly a point).

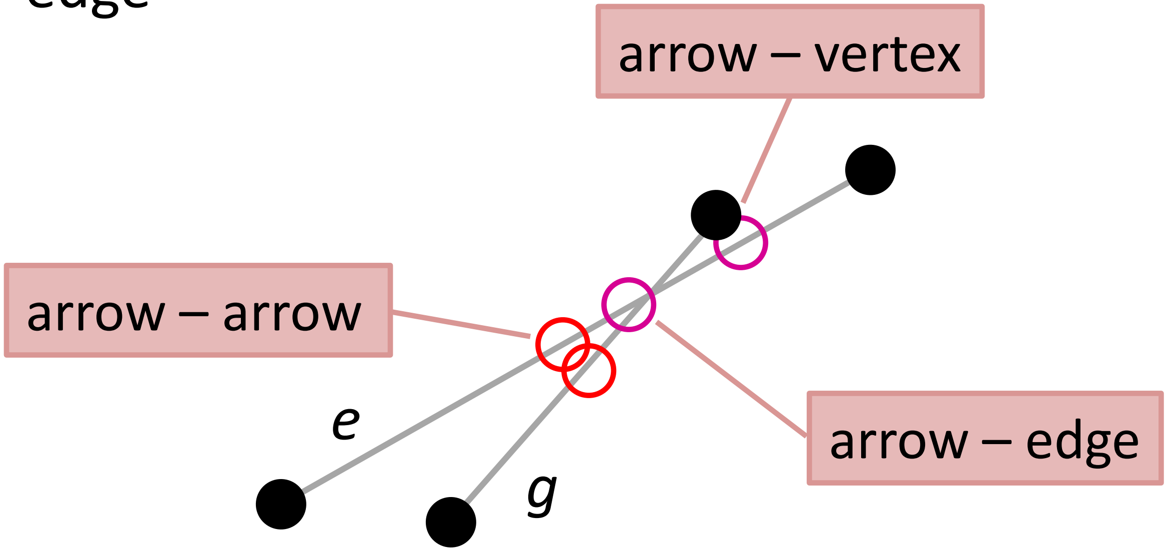
- We model an arrow of an edge e as a circle of radius r_E centered in a point along e

- When Γ is displayed, the arrow of e is drawn as a triangle inscribed in the circle.

Overlap between objects

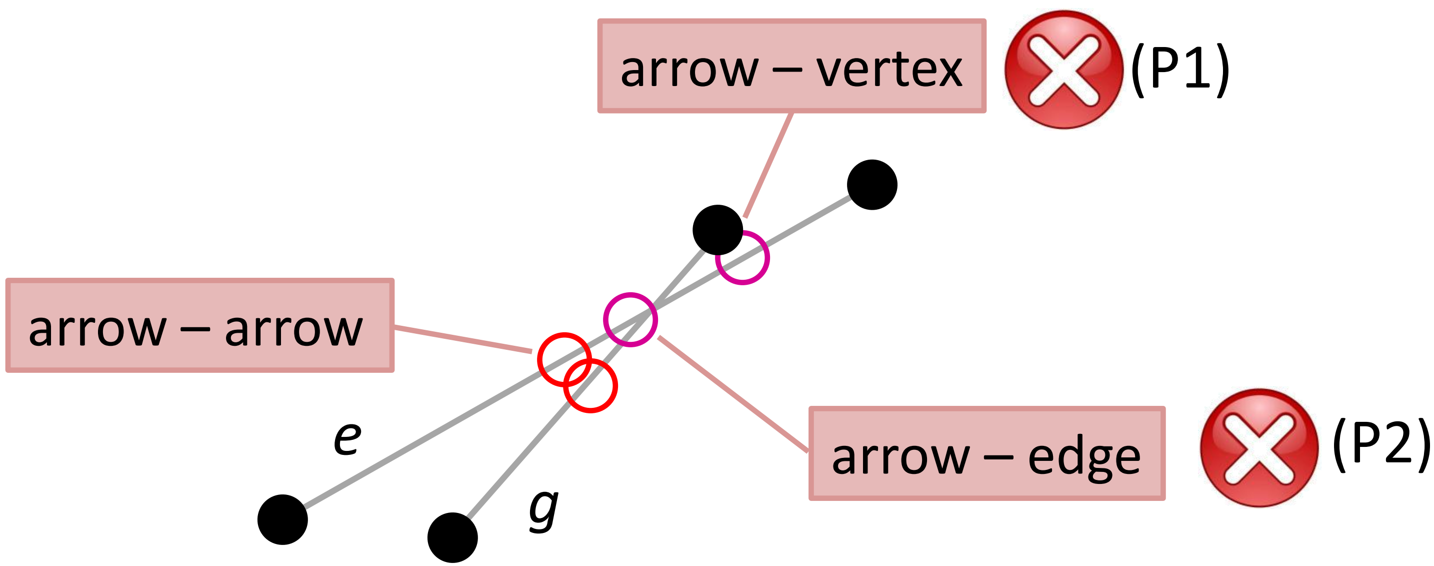
Three types of *overlap*:

- arrow – arrow
- arrow – vertex
- arrow – edge



A valid position

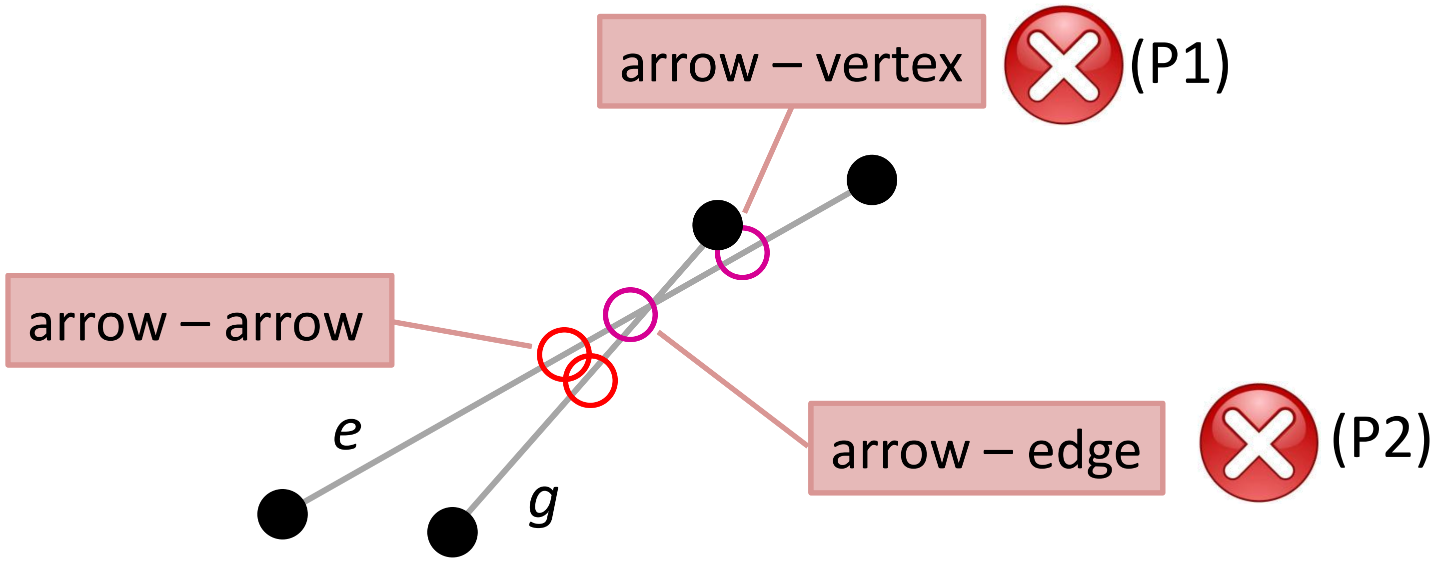
A position of an arrow of an edge *e* is a *valid position* if it does not overlap: (P1) any vertex v; (P2) any edge $g \neq e$.



A valid placement

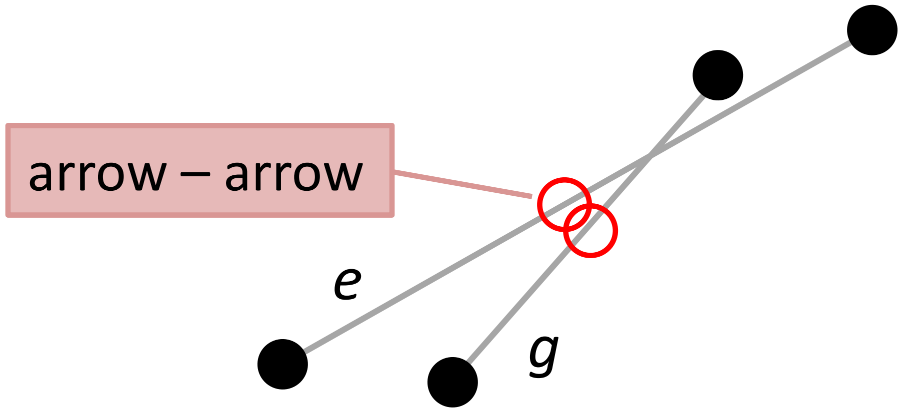
A position of an arrow of an edge *e* is a *valid position* if it does not overlap: (P1) any vertex v; (P2) any edge *g* ≠ *e.*

An assignment of a valid position to each arrow is called a *valid placement* of the arrows.



Overlap number

Given a valid placement, the *overlap number* is the number of pairs of overlapping arrows.



Arrow Placement problem

Assume that all circles representing a vertex and an arrow have a common radius r_V and r_E, respectively.

Given a straight-line drawing Γ of a digraph $G = (V, E)$, and two constants r_V and r_E compute a valid placement of the arrows (if one exists) such that the ***overlap number*** **is minimum**

NP-hardness

Theorem. The Arrow-Placement problem is NP-hard.

The proof uses a reduction from Planar 3-SAT; the technique is similar to those used in the context of edge and map labeling [*Kakoulis and Tollis*, 2001], [*Wolff*,2000], [*Strijk and Wolff*, 2001].

Discrete-Arrow-Placement problem

- Arrow-Placement remains NP-hard even if we fix a finite set of valid positions for each arrow.

- We call this variant *Discrete-Arrow-Placement* problem.

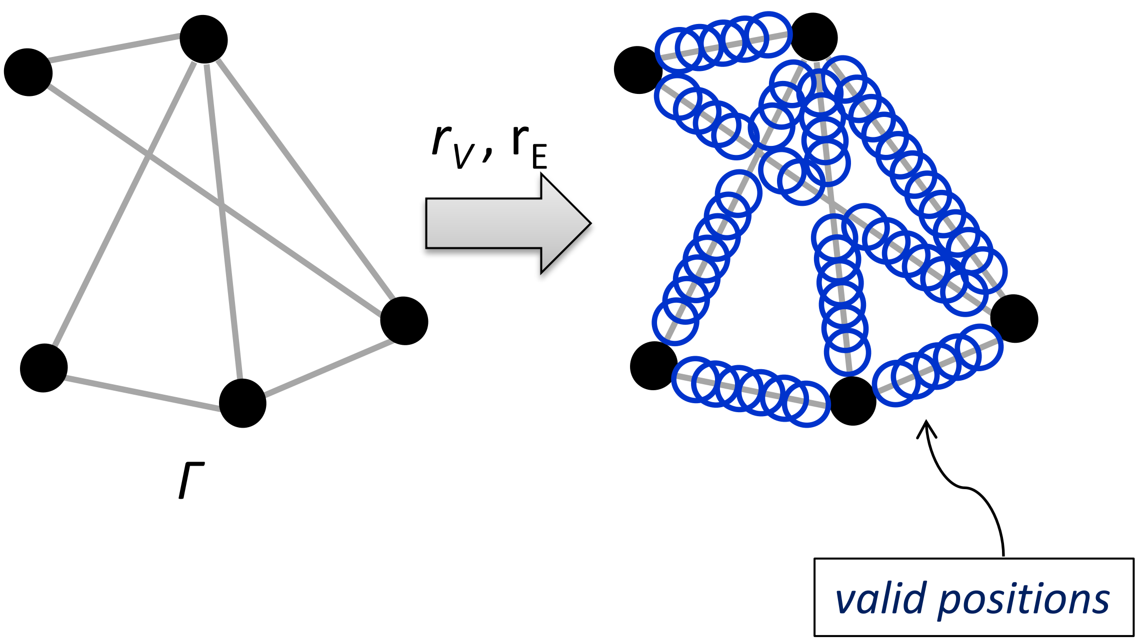
- Our algorithms are designed for this variant of the Arrow-Placement problem.

....In what follows....

- Problem formulation & NP-hardness
- **Algorithms**
- Experiments

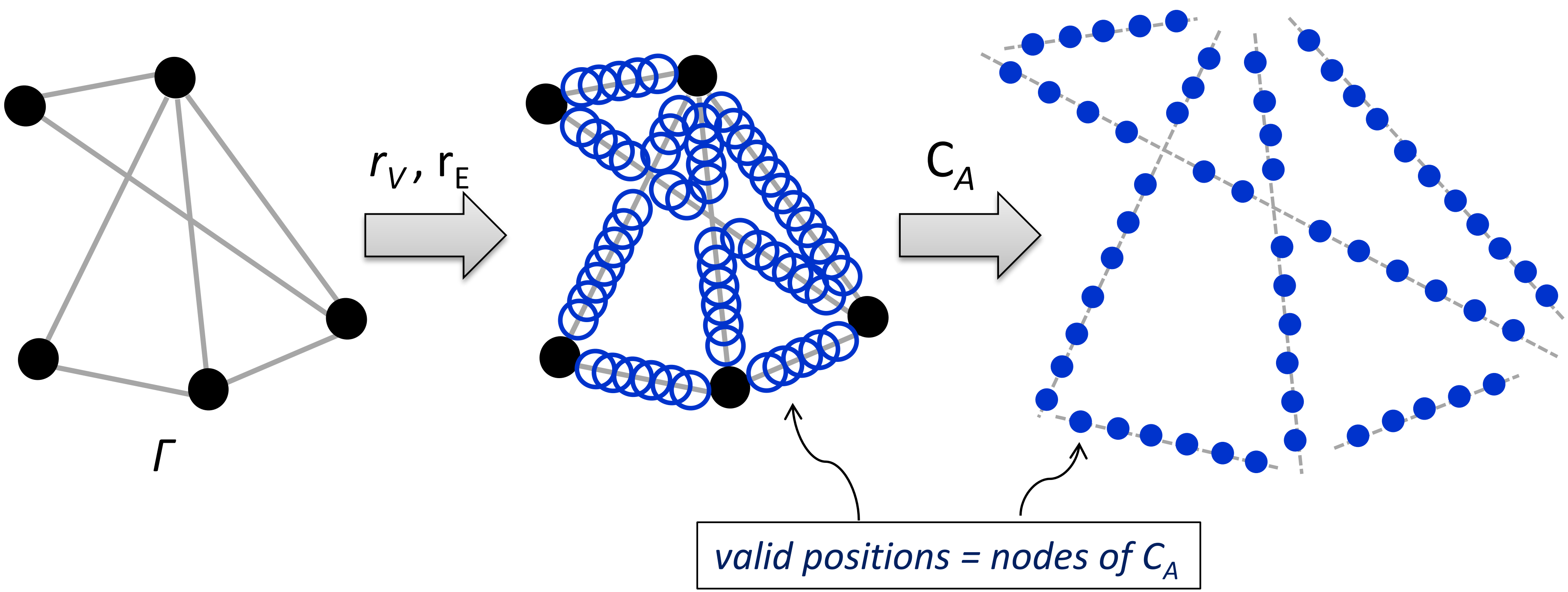
Algorithms – basic idea

- Our algorithms are based on an *arrow conflict graph C_A*.



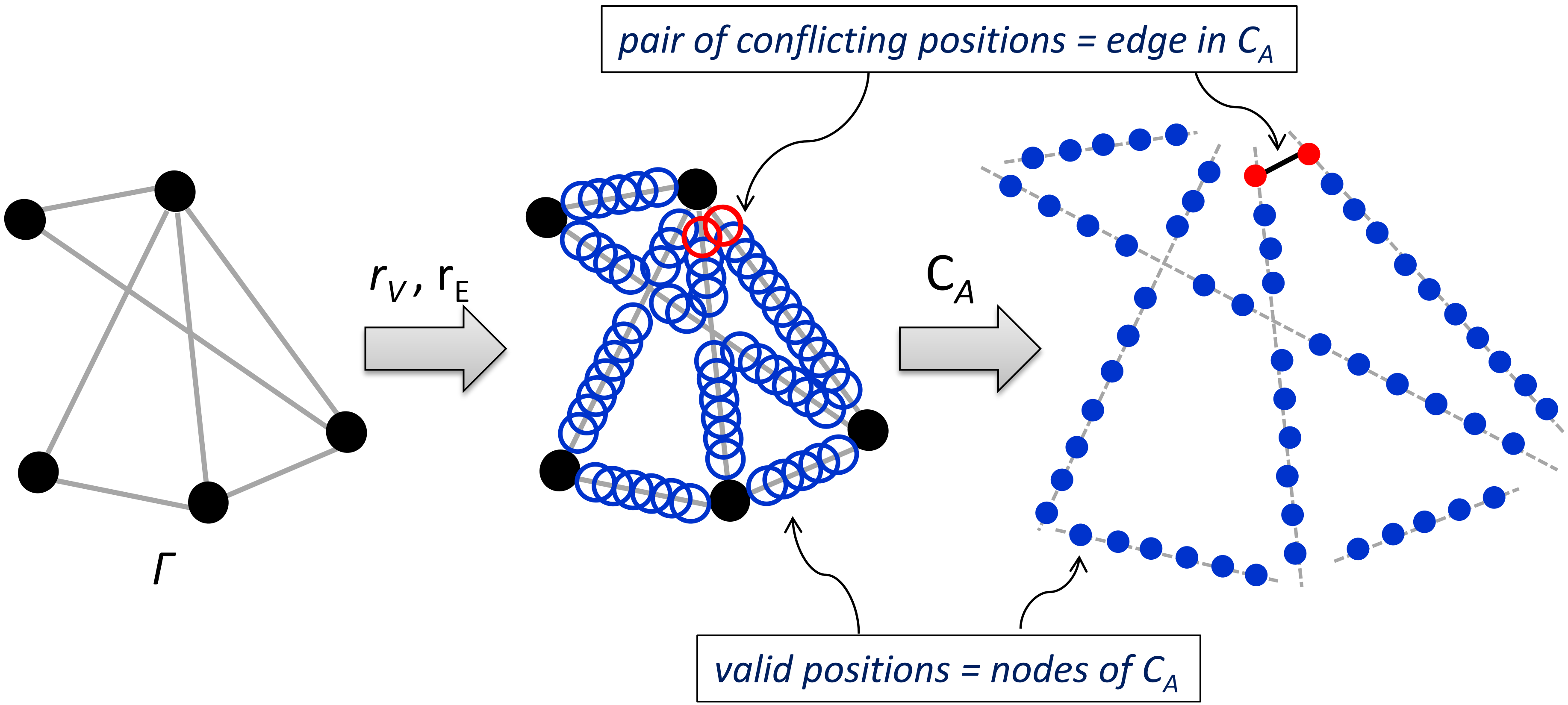
Algorithms – basic idea

- Our algorithms are based on an *arrow conflict graph C_A*.



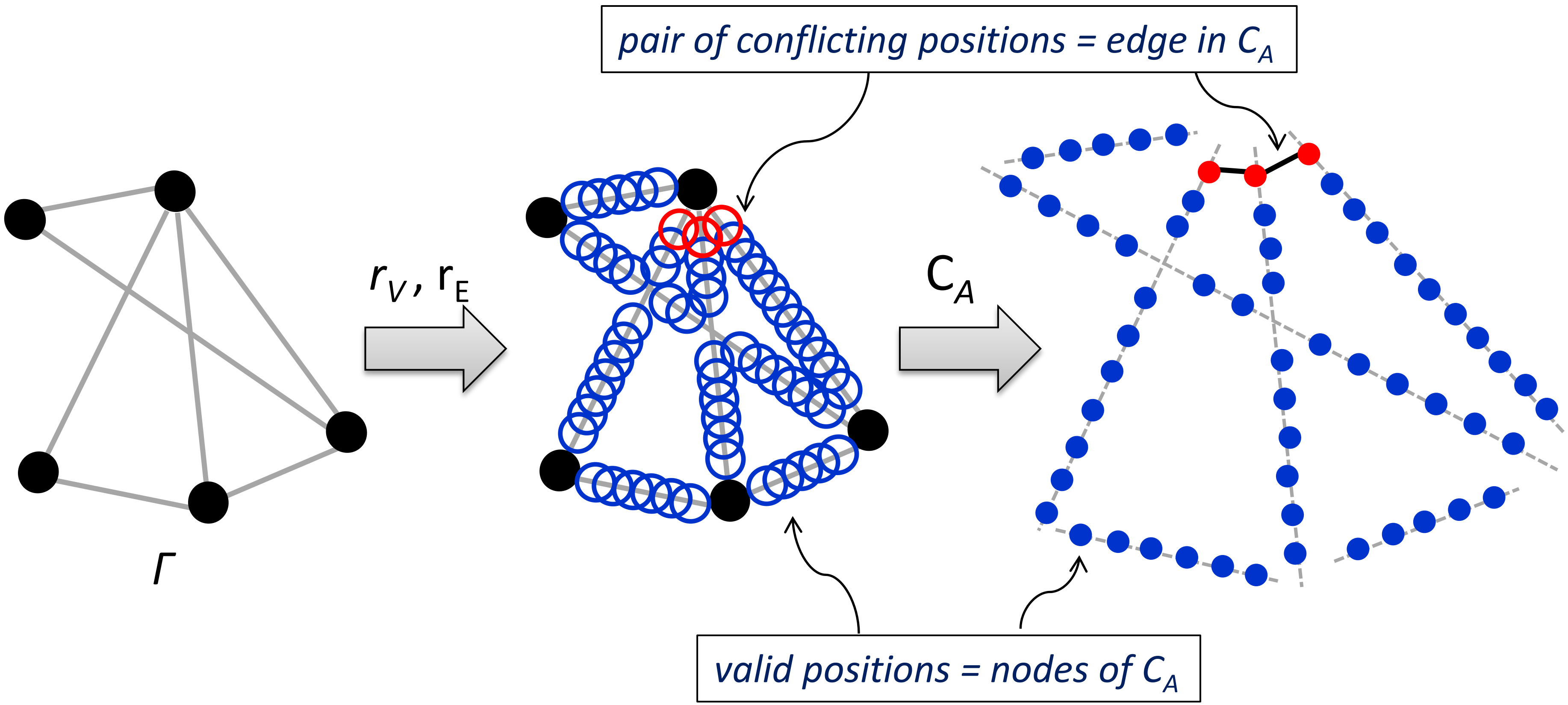
Algorithms – basic idea

- Our algorithms are based on an *arrow conflict graph C_A*.



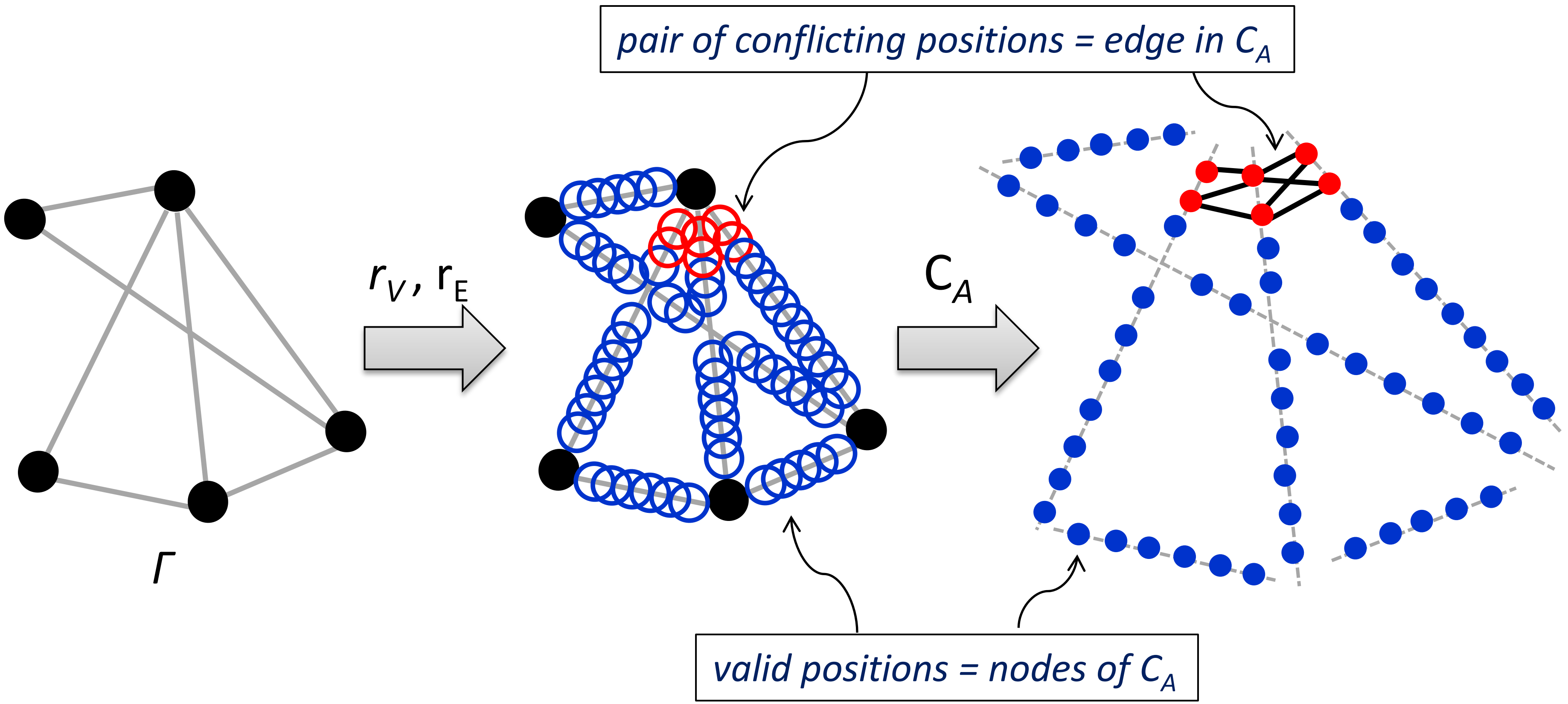
Algorithms – basic idea

- Our algorithms are based on an *arrow conflict graph C_A*.



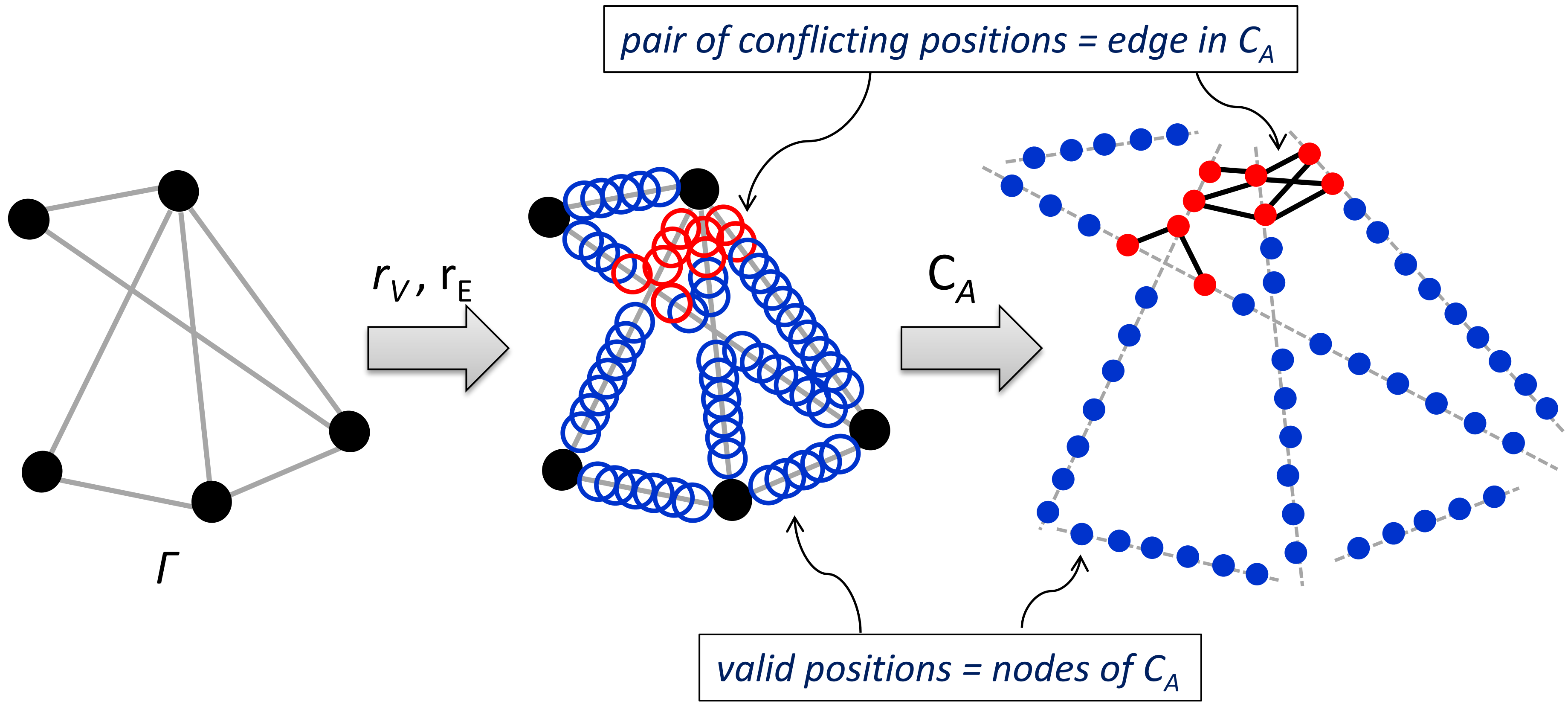
Algorithms – basic idea

- Our algorithms are based on an *arrow conflict graph C_A*.



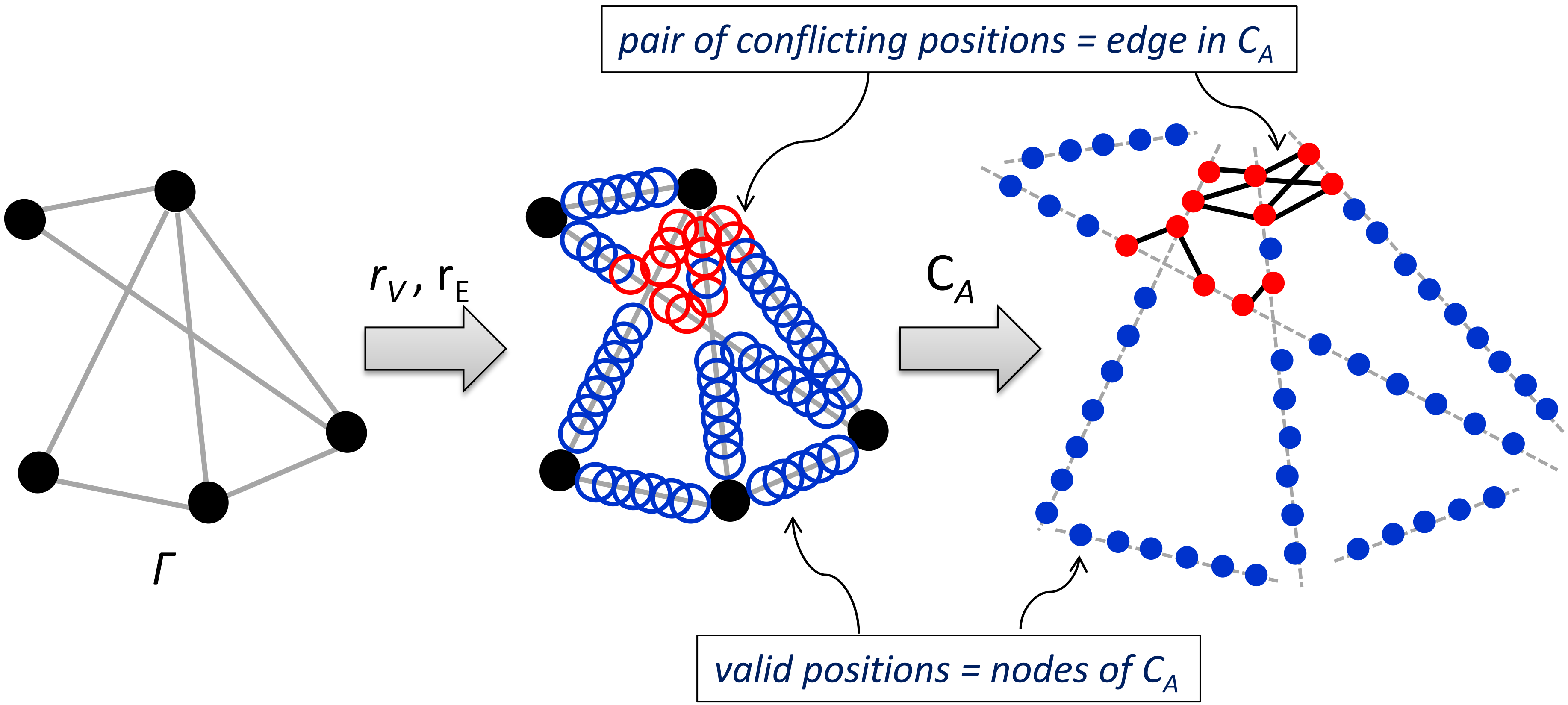
Algorithms – basic idea

- Our algorithms are based on an *arrow conflict graph C_A*.



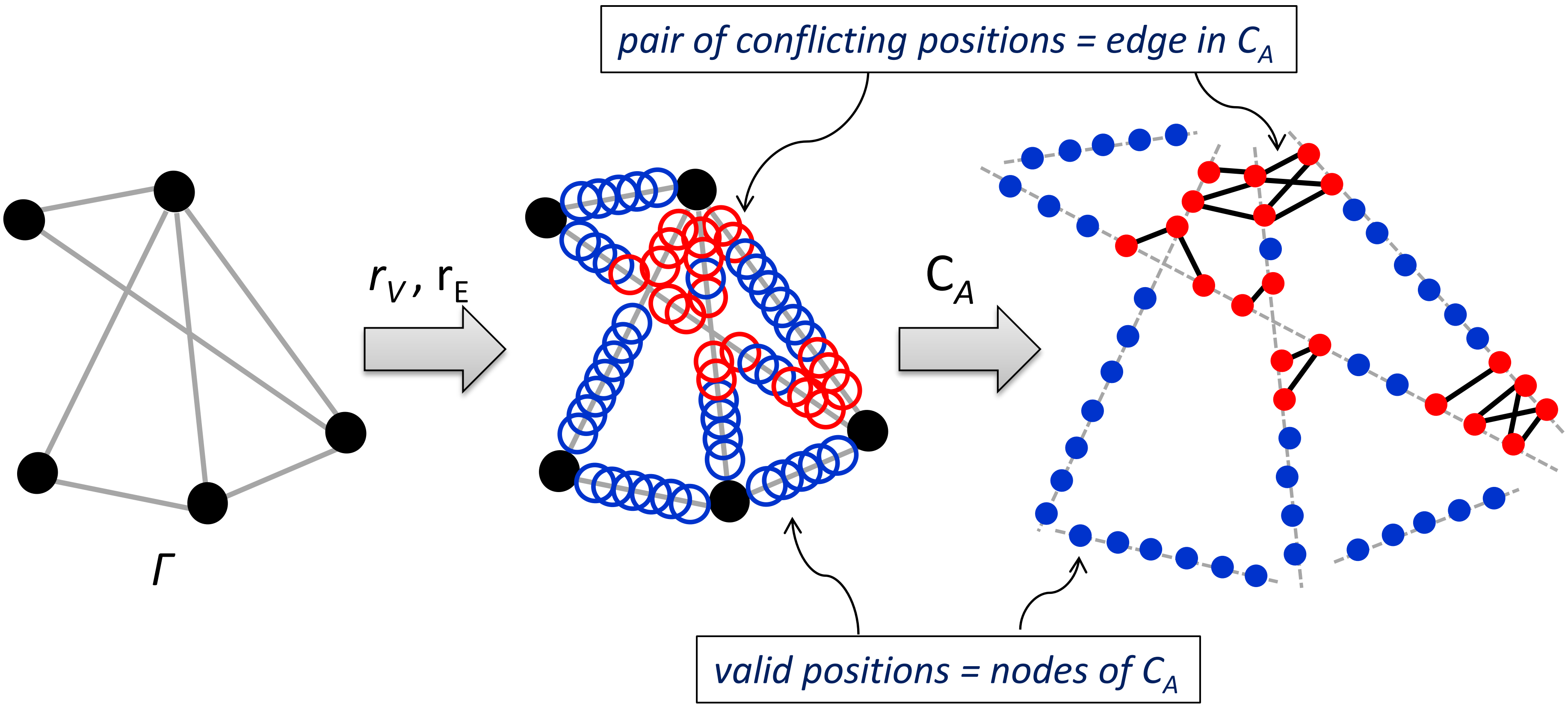
Algorithms – basic idea

- Our algorithms are based on an *arrow conflict graph* C_A.

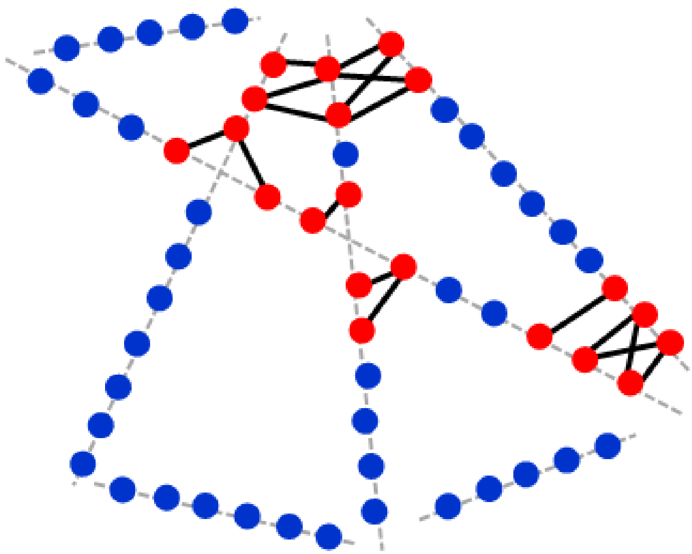


Algorithms – basic idea

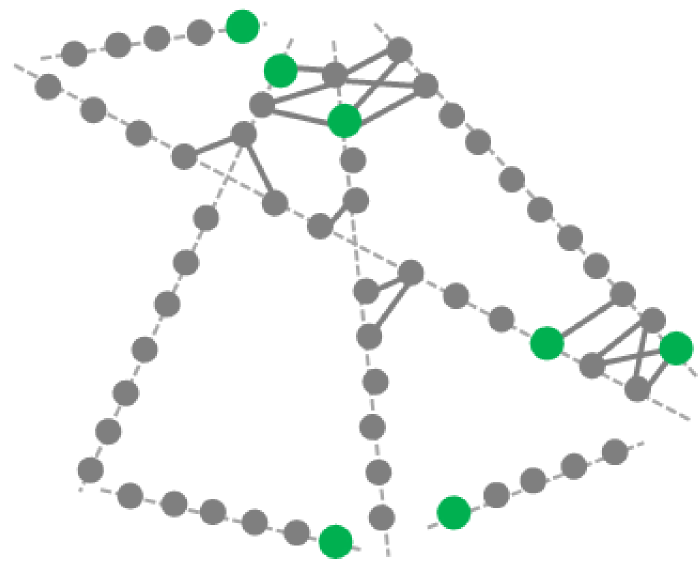
- Our algorithms are based on an *arrow conflict graph C_A*.

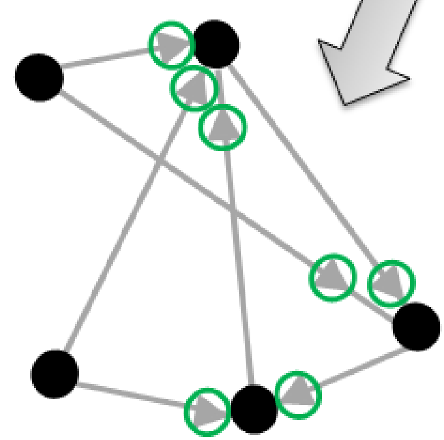


Algorithms – basic idea



select a valid position for each arrow





A valid placement with overlap number equal to zero

Algorithms

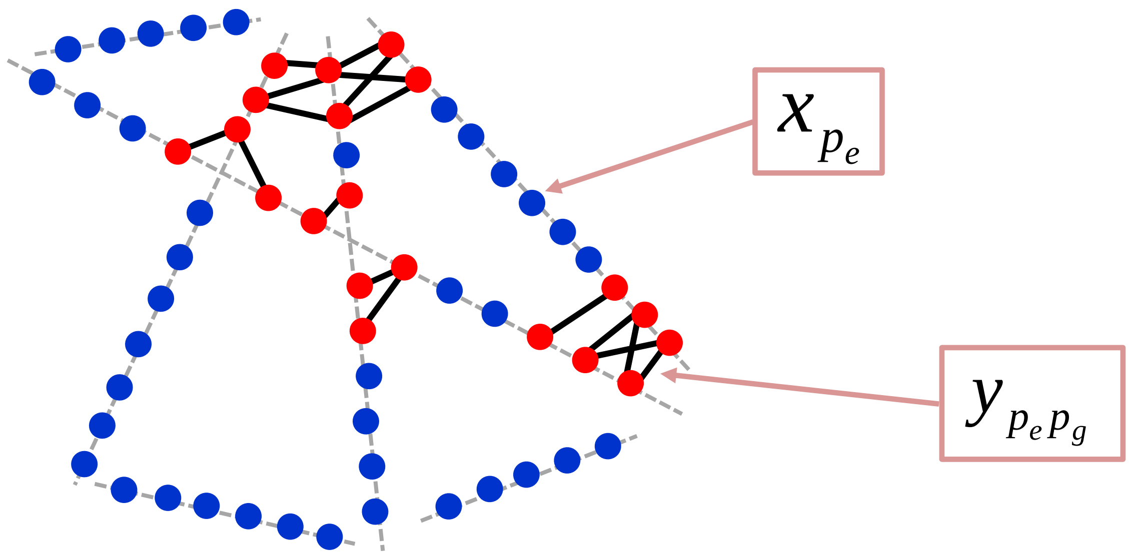
In general, we compute a valid placement with minimum number of overlaps.

- Our exact algorithm uses an ILP formulation.
- Our heuristic adopts a greedy strategy.

  - Both techniques try to minimize the distance of each arrow from its target vertex as a secondary objective.

ILP formulation - variables

- A binary variable x_{p_e} for each valid position p_e
- A binary variable $y_{p_e p_g}$ for each edge (p_e, p_g) of C_A



- The total number of variables is $O(|A|^2)$

ILP formulation

$$\min \sum_{(p_e, p_g) \in E(C_A)} y_{p_e p_g} + \frac{1}{M} \cdot \sum_{e \in E} \sum_{p_e \in A_e} d(p_e) x_{p_e}$$

distance of p_e from the target

$$\sum_{p_e \in A_e} x_{p_e} = 1 \qquad \forall e \in E$$

$$x_{p_e} + x_{p_g} \leq y_{p_e p_g} + 1 \qquad \forall (p_e, p_g) \in E(C_A)$$

Heuristic

Our heuristic follows a greedy strategy, based on C_A.

- We associate a cost $c(p_e)$ with each position p_e, and then execute $|E|$ iterations.

- In each iteration:
  - select a position p_e of minimum cost and place the arrow of the corresponding edge there;
  - remove all positions of edge e from C_A and update the costs of the remaining positions.

Heuristic – cost function

$$c(p_e) = \delta(p_e) + \frac{1}{M} \cdot d(p_e) + T \cdot \sigma_{p_e}$$

Number of positions conflicting with p_e

Distance of p_e from the target.

Number of already chosen positions that conflict with p_e

Heuristic – cost function

$$c(p_e) = \delta(p_e) + \frac{1}{M} \cdot d(p_e) + T \cdot \sigma_{p_e}$$

Number of positions conflicting with p_e

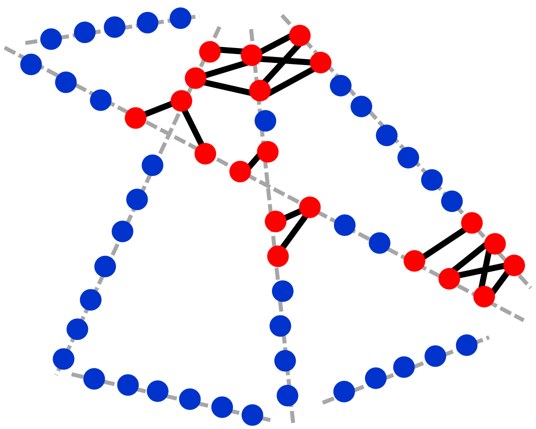
Distance of p_e from the target.

Number of already chosen positions that conflict with p_e

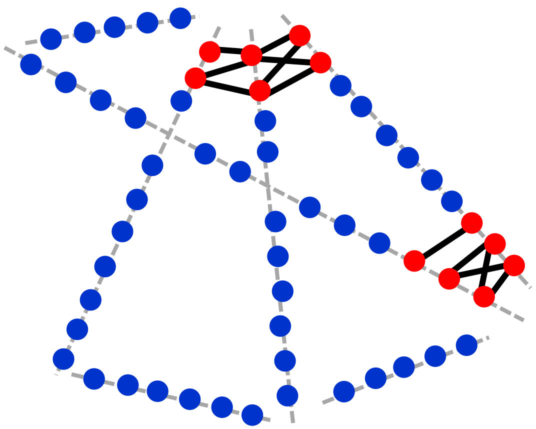
- Positions with minimum number of conflicts and closer to the target vertex, are preferred.
- Positions conflicting with already placed arrows are chosen only if necessary.

Heuristic - two variants

• Constructing C_A may be time-consuming in practice. We also considered a simplified version of C_A.



full C_A



simplified C_A

➢ **HeurGlobal** is the heuristic that considers full C_A.

➢ **HeurLocal** is the variant based on the simplified version of C_A.

….In what follows….

- Problem formulation & NP-hardness
- Algorithms
- **Experiments**

Experimental settings - Test suite

- **Planar**: biconnected planar digraphs with edge density 1.5–2.5
- **Random**: digraphs with edge density 1.4–1.6 (generated with uniform probability distribution).

- 30 instances each;
- 6 graphs for each number of vertices $n \in \{100, 200, \ldots, 500\}$.

- **North**: a set of 1,275 real-world digraphs with 10–100 vertices and average density 1.4.

- Drawing algorithm: OGDF's FM^3 algorithm [*Hachul and Jūnger*,2004].

Invalid positions

- If an edge has no valid positions we enforce it to have a unique (*invalid*) position for the arrow, the position closest to its target vertex.

- In the final placement there might be some crossings between an arrow and a vertex or an edge.

Measures

- **Running time**.
- **Placement time** (the time spent to find a placement after C_A has been computed).
- **Overlap number**.
- **Number of crossings** (due to invalid positions).

We compared our algorithms also with a trivial algorithm **EDITOR** which simply places each arrow close to its target vertex.

Major findings

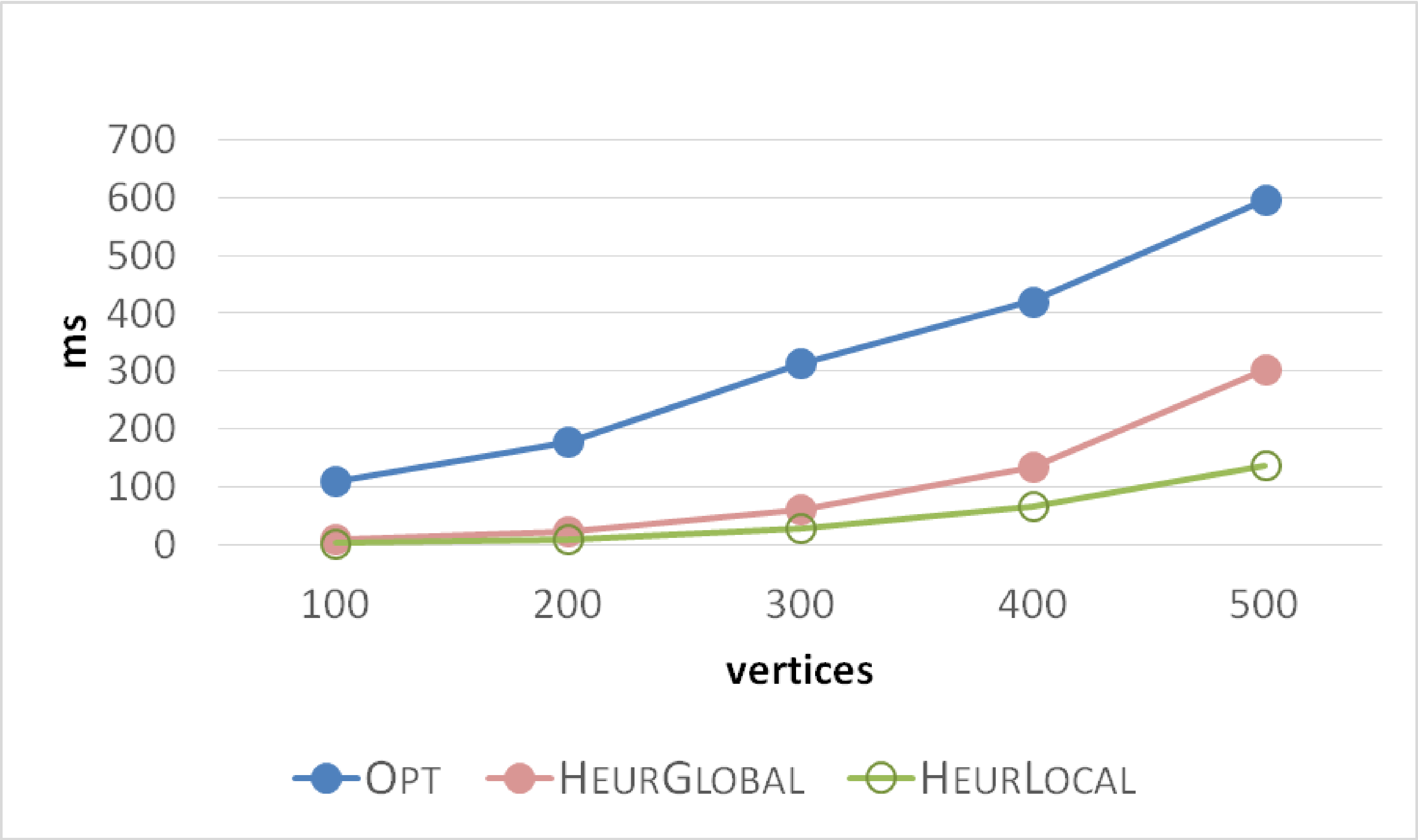
- The algorithms are efficient in practice (less than one second); the optimum (OPT) is the slowest.

- The placement time of HEURGLOBAL and HEURLOCAL are similar. 1/3 of the overall running time is taken from the construction of C_A. The construction of the simplified version of C_A is negligible.

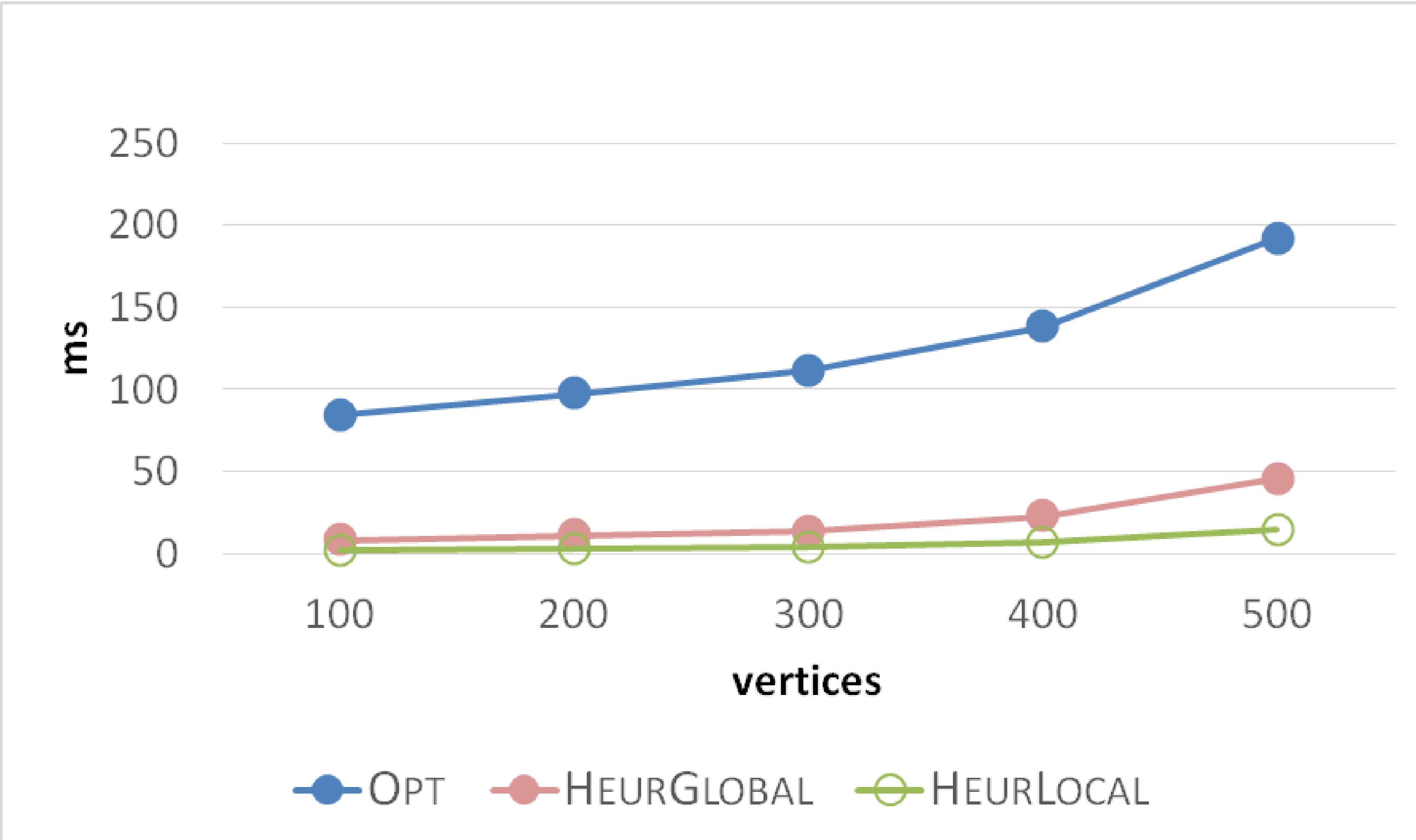
- HEURGLOBAL almost coincides with the optimum in terms of overlaps. HEURLOCAL also gives very good solutions.

- Our algorithms reduce the number of invalid positions and produce significantly less crossings than EDITOR.

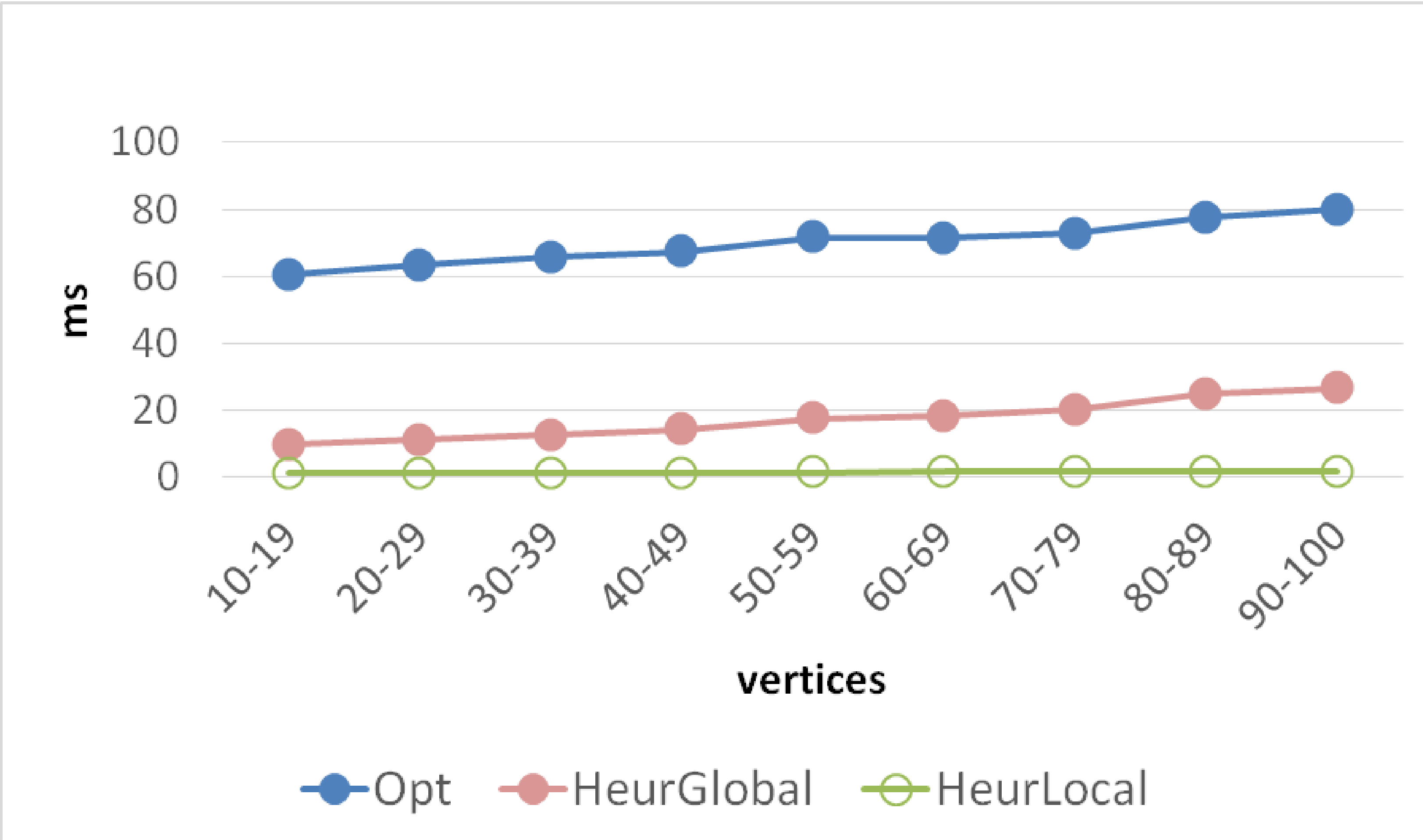
PLANAR – Running Time



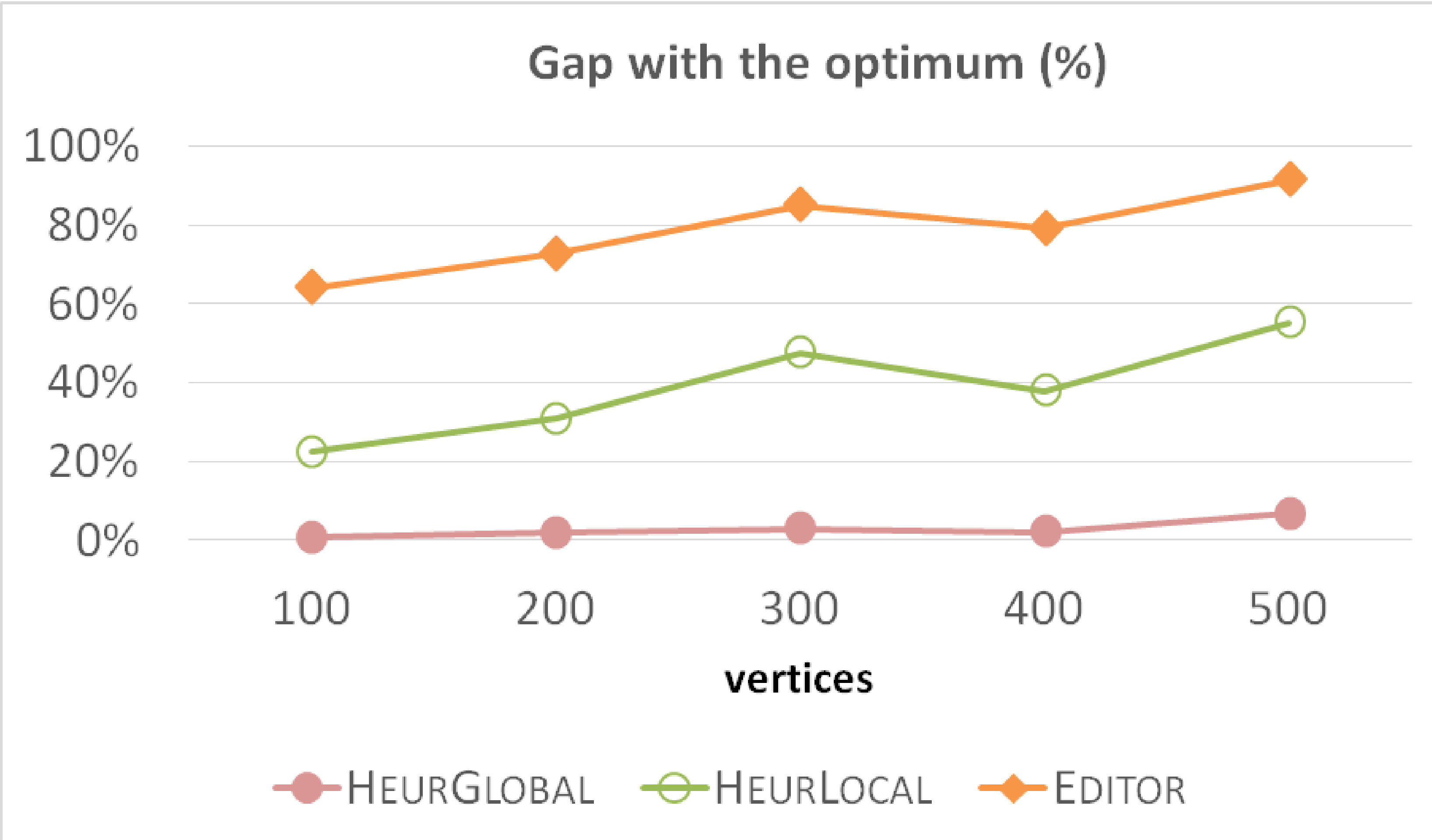
RANDOM – Running Time



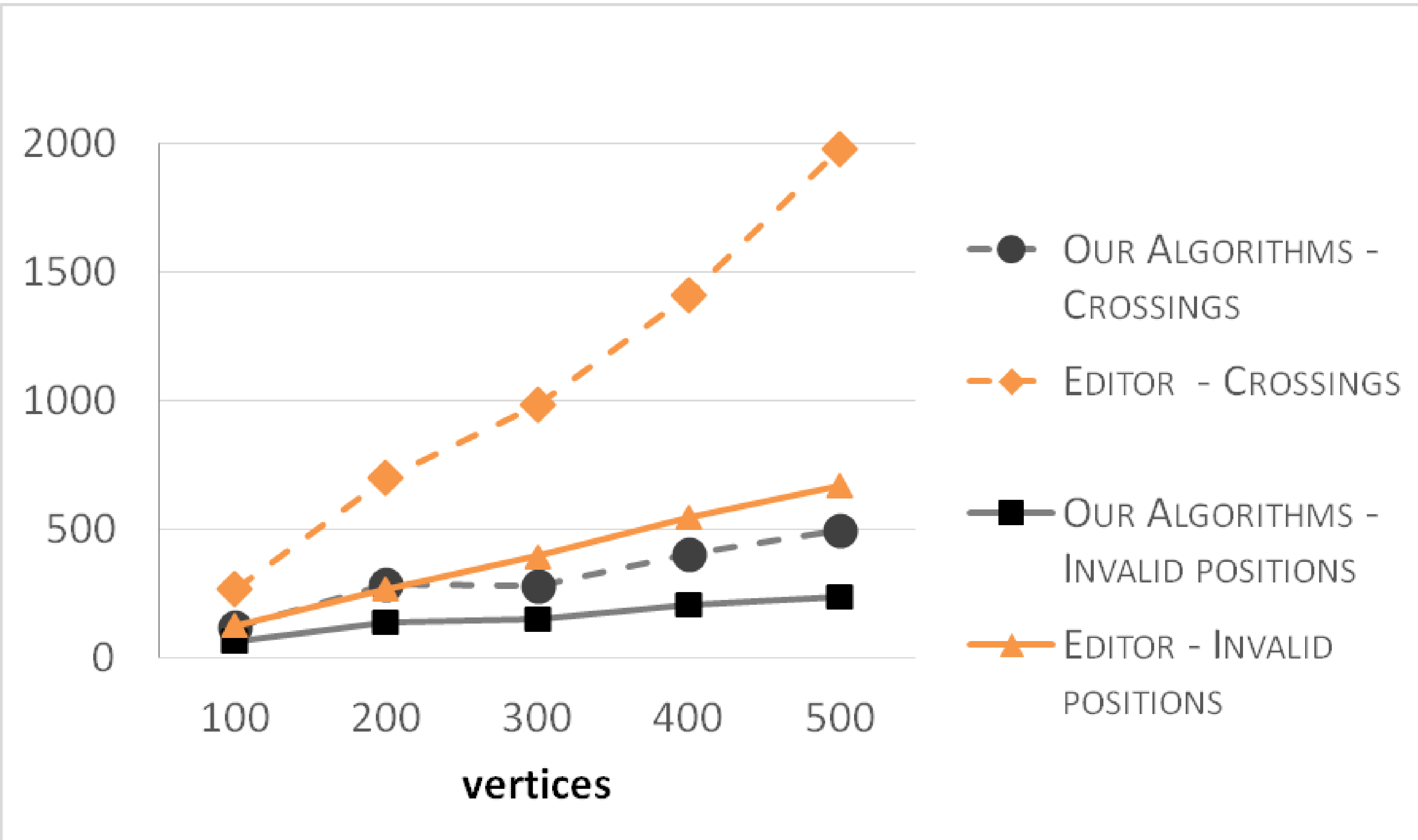
NORTH – Running Time



PLANAR – Overlap number



PLANAR – Crossings & invalid positions



Scalability of our techniques

- We extended both Planar and Random sets with 30 larger instances each (6 graphs for each number of vertices n ∈ {600,700,...,1000}).

- The behavior of our algorithms is similar to that reported for smaller instances:

  - The algorithms are still fast (less than two second).

  - HEURGLOBAL almost coincides with the optimum in terms of overlaps.

  - Our algorithms still generate significatively less crossings than EDITOR.

  - Constructing C_A remains the most expensive step.

Future work

- Speed-up our techniques for constructing C_A using a sweepline or the labeling techniques in [*Wagner et al.*, 2001].

- Validate the effectiveness of our approach through a user study (e.g. for tasks that involve path recognition).

- Consider both placing labels and arrow heads.

- Investigate the non-discretized problem variant, both from a practical and theoretical point of view.

Thank you!