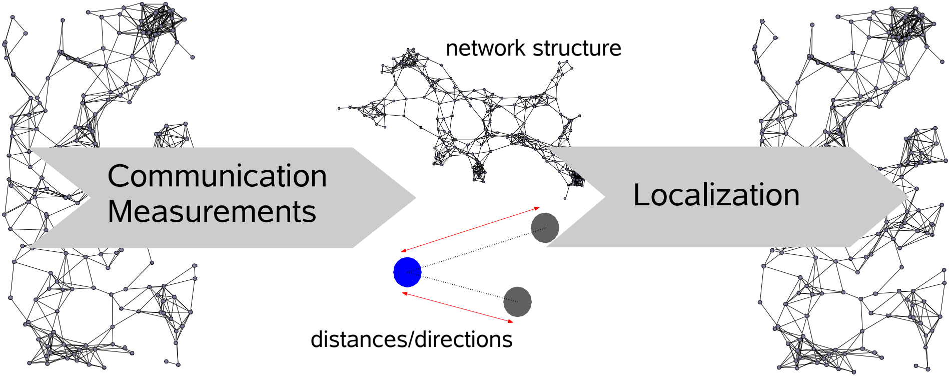
Maximum Rigid Components as Means for Direction-based Sensor Network Localization

Bastian Katz, Marco Gaertler, and Dorothea Wagner

Universität Karlsruhe (TH)
Forschungsuniversität · gegründet 1825

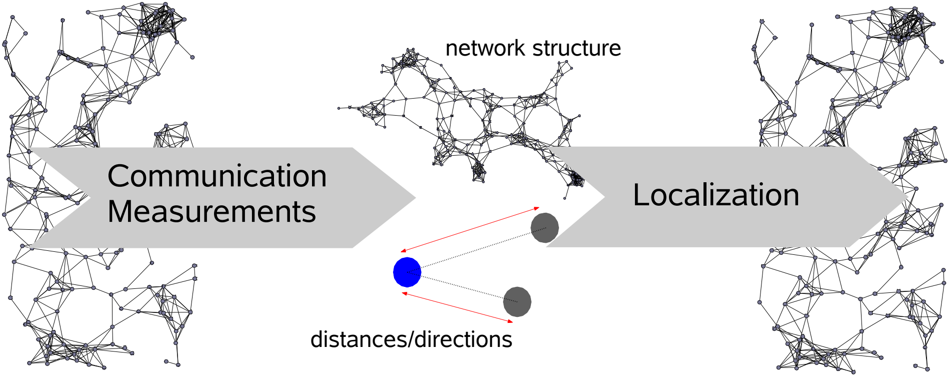
33^{rd} Int. Conference on Current Trends in Theory and Practice of Computer Science, Harachov, January 22, 2007

Sensor Network Localization



- Given a network, a communication model and measurements
  - is there a consistent embedding? (realizability)
  - is it unique? (uniqueness)
  - what is it? (reconstruction)

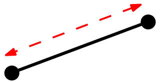
Sensor Network Localization



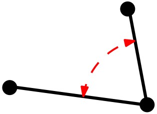
- Given a network, a communication model and measurements
  - is there a consistent embedding? (realizability)
  - is it unique? (uniqueness)
  - what is it? (reconstruction)

Possible Inputs of Localization Problems

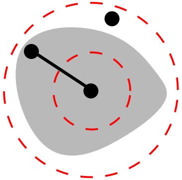
- Distances, e.g. from signal strength
  - uses common hardware capability
  - realization problem is $\mathcal{NP}$-hard



- Directions, e.g. from antenna arrays
  - needs special hardware
  - realization problem is in $\mathcal{P}$



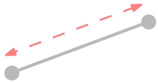
- Radio model, e.g. (quasi-)unit-disk-graph
  - relies on assumptions on radio propagation
  - $\mathcal{NP}$-hard even in combination with distances or directions

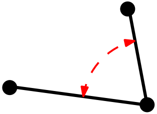


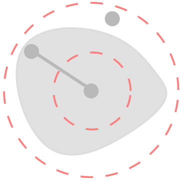
- We take a closer look on the algorithmics of the direction-based localization problem

Possible Inputs of Localization Problems

- Distances, e.g. from signal strength
  - uses common hardware capability
  - realization problem is $\mathcal{NP}$-hard
- Directions, e.g. from antenna arrays
  - needs special hardware
  - realization problem is in $\mathcal{P}$
- Radio model, e.g. (quasi-)unit-disk-graph
  - relies on assumptions on radio propagation
  - $\mathcal{NP}$-hard even in combination with distances or directions
- We take a closer look on the algorithmics of the direction-based localization problem







Localization Agenda

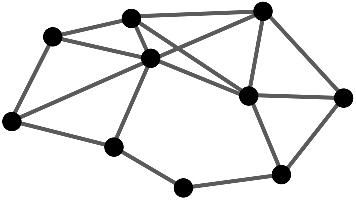
Localization Agenda

Given the communication graph with edge directions,

- What parts of the network can uniquely be reconstructed?
- What is an efficient way to identify these parts?
  - Can we take advantage of the sparsity and/or high locality?
  - To what extent can we use distributed techniques?
- How can we get the layout then?

Uniqueness

- Possible up to scaling/rotation
- Known problem in *rigidity theory*
  - uniqueness coincides with (parallel) rigidity
  - we are looking for maximum rigid components



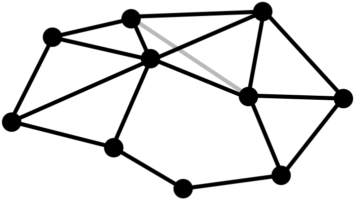
Parallel Rigidity [Laman 70 / Whiteley 96]

A graph $G = (V, E)$ is (parallelly) rigid in the plane iff it contains edges $E' \subseteq E$ with $|E'| = 2|V| - 3$ such that for all $E'' \subset E'$

$$|E''| \leq 2|V(E'')| - 3$$

Uniqueness

- Possible up to scaling/rotation
- Known problem in *rigidity theory*
  - uniqueness coincides with (parallel) rigidity
  - we are looking for maximum rigid components



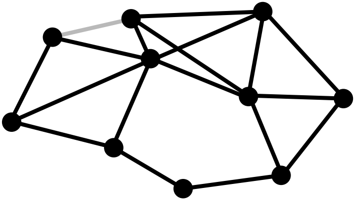
Parallel Rigidity [Laman 70 / Whiteley 96]

A graph $G = (V, E)$ is (parallelly) rigid in the plane iff it contains edges $E' \subseteq E$ with $|E'| = 2|V| - 3$ such that for all $E'' \subset E'$

$$|E''| \leq 2|V(E'')| - 3$$

Uniqueness

- Possible up to scaling/rotation
- Known problem in *rigidity theory*
  - uniqueness coincides with (parallel) rigidity
  - we are looking for maximum rigid components



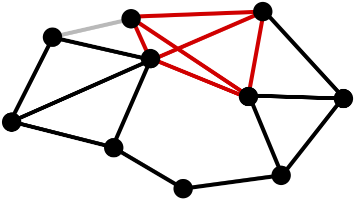
Parallel Rigidity [Laman 70 / Whiteley 96]

A graph $G = (V, E)$ is (parallelly) rigid in the plane iff it contains edges $E' \subseteq E$ with $|E'| = 2|V| - 3$ such that for all $E'' \subset E'$

$$|E''| \leq 2|V(E'')| - 3$$

Uniqueness

- Possible up to scaling/rotation
- Known problem in *rigidity theory*
  - uniqueness coincides with (parallel) rigidity
  - we are looking for maximum rigid components



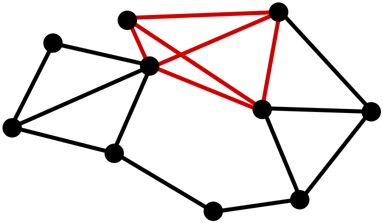
Parallel Rigidity [Laman 70 / Whiteley 96]

A graph $G = (V, E)$ is (parallelly) rigid in the plane iff it contains edges $E' \subseteq E$ with $|E'| = 2|V| - 3$ such that for all $E'' \subset E'$

$$|E''| \leq 2|V(E'')| - 3$$

Uniqueness

- Possible up to scaling/rotation
- Known problem in *rigidity theory*
  - uniqueness coincides with (parallel) rigidity
  - we are looking for maximum rigid components



Parallel Rigidity [Laman 70 / Whiteley 96]

A graph $G = (V, E)$ is (parallelly) rigid in the plane iff it contains edges $E' \subseteq E$ with $|E'| = 2|V| - 3$ such that for all $E'' \subset E'$

$$|E''| \leq 2|V(E'')| - 3$$

Rigid Components

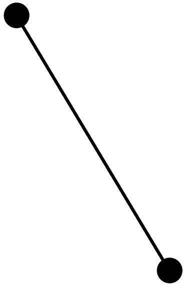
- Known test for rigidity: *pebble game*
  - could be adapted to identify rigid components
  - hard to distribute, runtime in $\mathcal{O}(n^2)$
- More intuitive techniques for graphs with high locality:
  - edges (trivial)
  - triangulation
  - edge overlapping
  - node overlapping
  - all combinations
- faster, easier to distribute, but...
- they do not end with *maximum* rigid components.

Rigid Components

- Known test for rigidity: *pebble game*
  - could be adapted to identify rigid components
  - hard to distribute, runtime in $\mathcal{O}(n^2)$
- More intuitive techniques for graphs with high locality:
  - edges (trivial)
  - triangulation
  - edge-overlapping
  - node-overlapping
  - all combinations
- faster, easier to distribute, but…
- they do not end with *maximum* rigid components.

Rigid Components

- Known test for rigidity: *pebble game*
  - could be adapted to identify rigid components
  - hard to distribute, runtime in $\mathcal{O}(n^2)$
- More intuitive techniques for graphs with high locality:
  - edges (trivial)
  - triangulation
  - edge-overlapping
  - node-overlapping
  - all combinations
- faster, easier to distribute, but…
- they do not end with *maximum* rigid components.

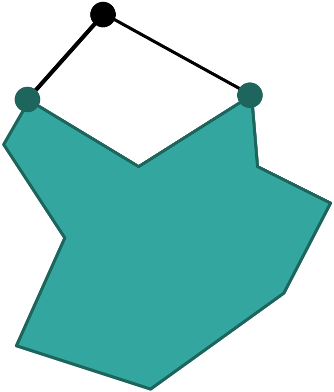


Rigid Components

- Known test for rigidity: *pebble game*
  - could be adapted to identify rigid components
  - hard to distribute, runtime in $\mathcal{O}(n^2)$
- More intuitive techniques for graphs with high locality:
  - edges (trivial)
  - triangulation
  - edge-overlapping
  - node-overlapping
  - all combinations
- faster, easier to distribute, but…
- they do not end with *maximum* rigid components.

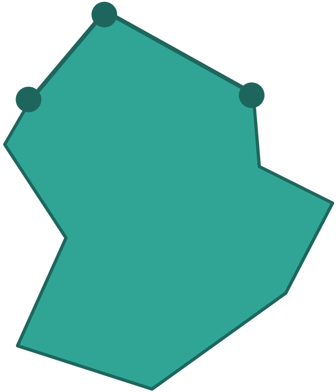
Rigid Components

- Known test for rigidity: *pebble game*
  - could be adapted to identify rigid components
  - hard to distribute, runtime in $\mathcal{O}(n^2)$
- More intuitive techniques for graphs with high locality:
  - edges (trivial)
  - triangulation
  - edge-overlapping
  - node-overlapping
  - all combinations
- faster, easier to distribute, but. . .
- they do not end with *maximum* rigid components.



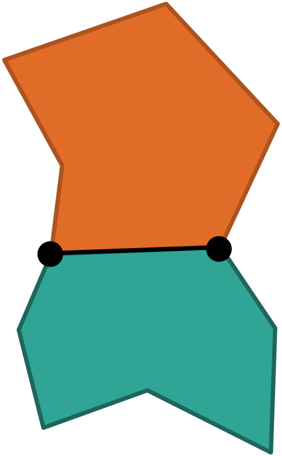
Rigid Components

- Known test for rigidity: *pebble game*
  - could be adapted to identify rigid components
  - hard to distribute, runtime in $\mathcal{O}(n^2)$
- More intuitive techniques for graphs with high locality:
  - edges (trivial)
  - triangulation
  - edge-overlapping
  - node-overlapping
  - all combinations
- faster, easier to distribute, but...
- they do not end with *maximum* rigid components.



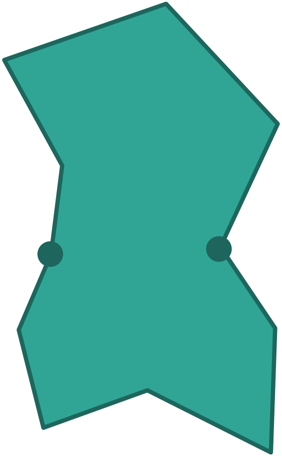
Rigid Components

- Known test for rigidity: *pebble game*
  - could be adapted to identify rigid components
  - hard to distribute, runtime in $\mathcal{O}(n^2)$
- More intuitive techniques for graphs with high locality:
  - edges (trivial)
  - triangulation
  - edge-overlapping
  - node-overlapping
  - all combinations
- faster, easier to distribute, but...
- they do not end with *maximum* rigid components.



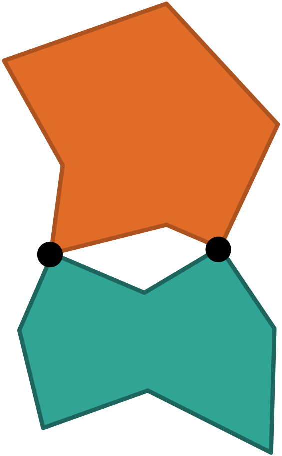
Rigid Components

- Known test for rigidity: *pebble game*
  - could be adapted to identify rigid components
  - hard to distribute, runtime in $\mathcal{O}(n^2)$
- More intuitive techniques for graphs with high locality:
  - edges (trivial)
  - triangulation
  - edge-overlapping
  - node-overlapping
  - all combinations
- faster, easier to distribute, but...
- they do not end with *maximum* rigid components.



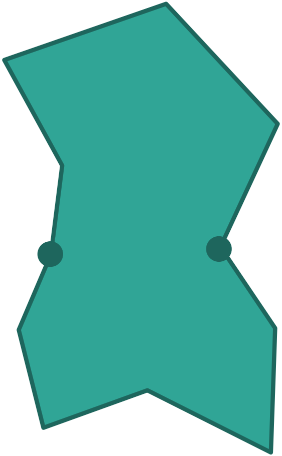
Rigid Components

- Known test for rigidity: *pebble game*
  - could be adapted to identify rigid components
  - hard to distribute, runtime in $\mathcal{O}(n^2)$
- More intuitive techniques for graphs with high locality:
  - edges (trivial)
  - triangulation
  - edge-overlapping
  - node-overlapping
  - all combinations
- faster, easier to distribute, but...
- they do not end with *maximum* rigid components.



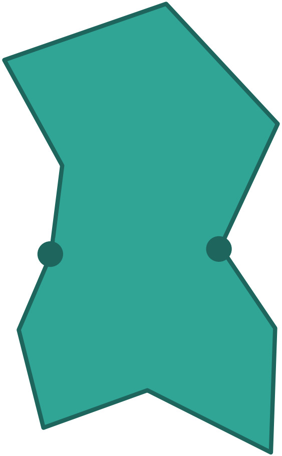
Rigid Components

- Known test for rigidity: *pebble game*
  - could be adapted to identify rigid components
  - hard to distribute, runtime in $\mathcal{O}(n^2)$
- More intuitive techniques for graphs with high locality:
  - edges (trivial)
  - triangulation
  - edge-overlapping
  - node-overlapping
  - all combinations
- faster, easier to distribute, but...
- they do not end with *maximum* rigid components.



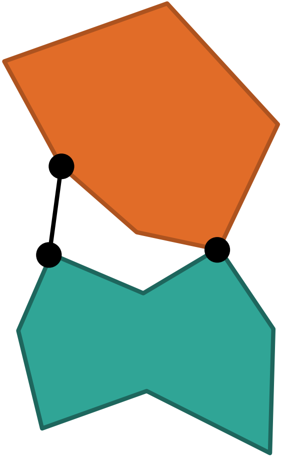
Rigid Components

- Known test for rigidity: *pebble game*
  - could be adapted to identify rigid components
  - hard to distribute, runtime in $\mathcal{O}(n^2)$
- More intuitive techniques for graphs with high locality:
  - edges (trivial)
  - triangulation
  - edge-overlapping
  - node-overlapping
  - all combinations
- faster, easier to distribute, but...
- they do not end with *maximum* rigid components.



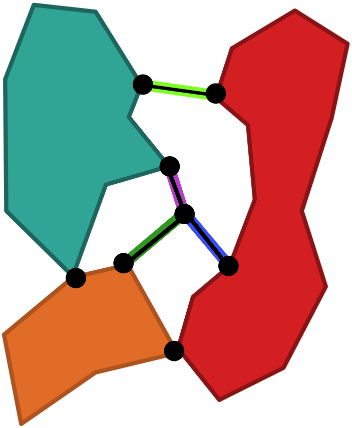
Rigid Components

- Known test for rigidity: *pebble game*
  - could be adapted to identify rigid components
  - hard to distribute, runtime in $\mathcal{O}(n^2)$
- More intuitive techniques for graphs with high locality:
  - edges (trivial)
  - triangulation
  - edge-overlapping
  - node-overlapping
  - all combinations
- faster, easier to distribute, but...
- they do not end with *maximum* rigid components.



Stuck?

- Is the price of being greedy to end here?
- Can we find maximum rigid components in such *body-joint frameworks*?
  - previous work [Moukarzel 96] solves only special cases



Rigidity in Body-Joint Frameworks

- Intuition in graphs:

  A *minimal* subgraph with enough edges is rigid.

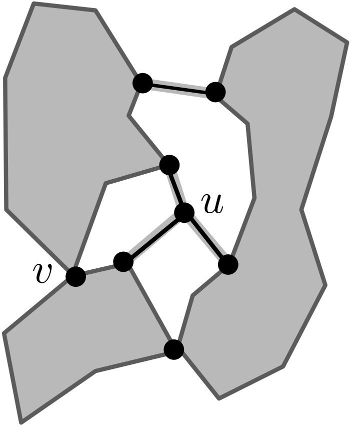
- Intuition with bodies and joints:

  A *minimal* set of bodies using enough nodes redundantly is rigid.

- Node redundancy wrt. a set of bodies $\mathcal{S}$:

$$\mathrm{rd}_{\mathcal{S}}(v) := \sharp\{S \in \mathcal{S} \mid v \in S\} - 1$$

- Example: $\mathrm{rd}_v(\mathcal{S}) = 1$, $\mathrm{rd}_u(\mathcal{S}) = 2$



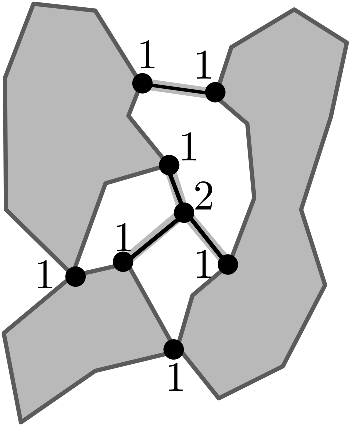
Rigidity in body-joint frameworks cont'd

Theorem [Our paper]

Let $\mathcal{S}$ be a set of bodies. If $\mathcal{S}$ is minimal with

$$2\sum_{v \in V} \mathrm{rd}_{\mathcal{S}}(v) \geq 3(|\mathcal{S}| - 1) \ ,$$

the bodies from $\mathcal{S}$ together are rigid.



≫ Example:
$2\sum \mathrm{rd}_{\mathcal{S}}(v) = 18 = 3(|\mathcal{S}| - 1)$

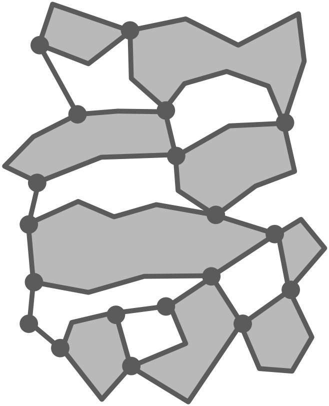
Algorithm Outline

Algorithm

Iteratively grow a set of maximally rigid bodies!

- repeatedly add a body to a set $\mathcal{S}^\star$
- check if $\mathcal{S}^\star$ contains a rigid subset
- merge rigid subsets as soon as possible

- How can we efficiently test for rigid subsets?



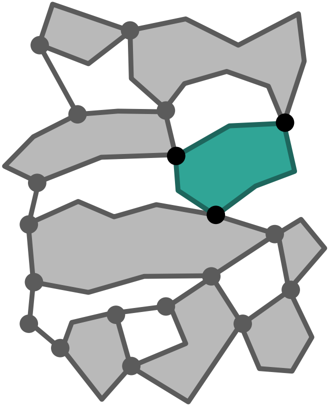
Algorithm Outline

Algorithm

Iteratively grow a set of maximally rigid bodies!

- repeatedly add a body to a set $\mathcal{S}^\star$
- check if $\mathcal{S}^\star$ contains a rigid subset
- merge rigid subsets as soon as possible

- How can we efficiently test for rigid subsets?



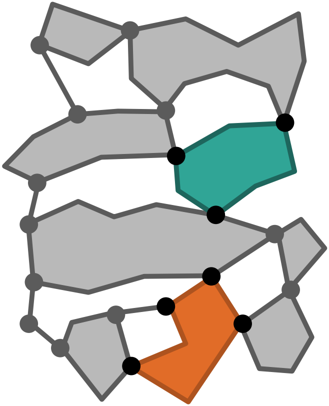
Algorithm Outline

Algorithm

Iteratively grow a set of maximally rigid bodies!

- ≫ repeatedly add a body to a set $\mathcal{S}^\star$
- ≫ check if $\mathcal{S}^\star$ contains a rigid subset
- ≫ merge rigid subsets as soon as possible

- ≫ How can we efficiently test for rigid subsets?



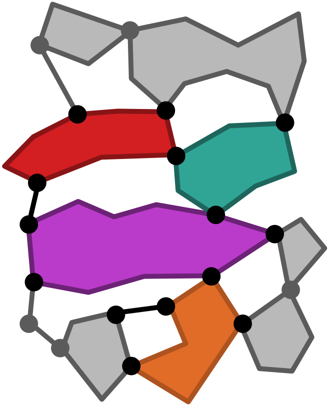
Algorithm Outline

Algorithm

Iteratively grow a set of maximally rigid bodies!

- ≫ repeatedly add a body to a set $\mathcal{S}^\star$
- ≫ check if $\mathcal{S}^\star$ contains a rigid subset
- ≫ merge rigid subsets as soon as possible

- ≫ How can we efficiently test for rigid subsets?



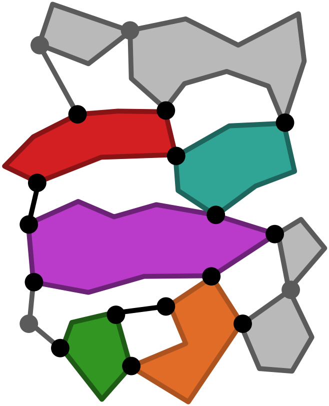
Algorithm Outline

Algorithm

Iteratively grow a set of maximally rigid bodies!

- ≫ repeatedly add a body to a set $\mathcal{S}^\star$
- ≫ check if $\mathcal{S}^\star$ contains a rigid subset
- ≫ merge rigid subsets as soon as possible

≫ How can we efficiently test for rigid subsets?



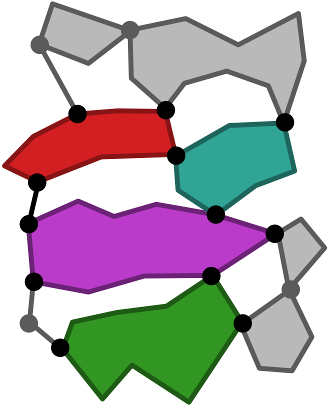
Algorithm Outline

Algorithm

Iteratively grow a set of maximally rigid bodies!

- ≫ repeatedly add a body to a set $\mathcal{S}^\star$
- ≫ check if $\mathcal{S}^\star$ contains a rigid subset
- ≫ merge rigid subsets as soon as possible

≫ How can we efficiently test for rigid subsets?



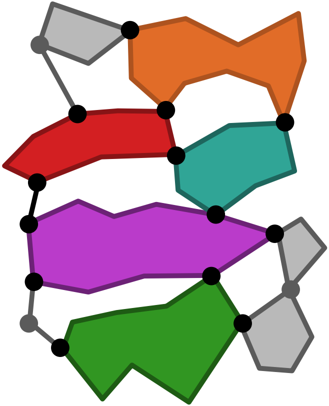
Algorithm Outline

Algorithm

Iteratively grow a set of maximally rigid bodies!

- ≫ repeatedly add a body to a set $\mathcal{S}^\star$
- ≫ check if $\mathcal{S}^\star$ contains a rigid subset
- ≫ merge rigid subsets as soon as possible

≫ How can we efficiently test for rigid subsets?

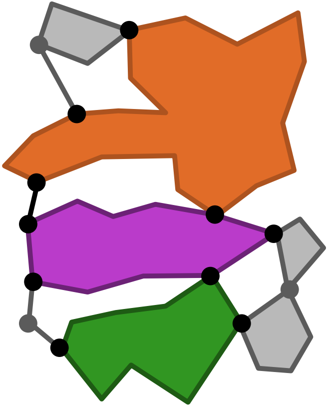


Algorithm Outline

Algorithm

Iteratively grow a set of maximally rigid bodies!

- repeatedly add a body to a set $\mathcal{S}^{\star}$
- check if $\mathcal{S}^{\star}$ contains a rigid subset
- merge rigid subsets as soon as possible



- How can we efficiently test for rigid subsets?

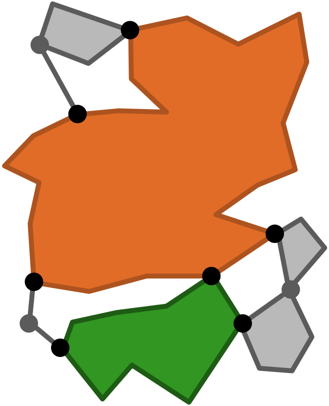
Algorithm Outline

Algorithm

Iteratively grow a set of maximally rigid bodies!

- repeatedly add a body to a set $\mathcal{S}^\star$
- check if $\mathcal{S}^\star$ contains a rigid subset
- merge rigid subsets as soon as possible

- How can we efficiently test for rigid subsets?



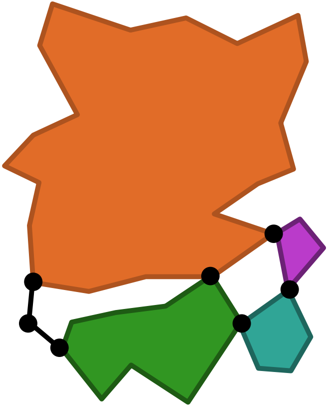
Algorithm Outline

Algorithm

Iteratively grow a set of maximally rigid bodies!

- repeatedly add a body to a set $\mathcal{S}^\star$
- check if $\mathcal{S}^\star$ contains a rigid subset
- merge rigid subsets as soon as possible

- How can we efficiently test for rigid subsets?

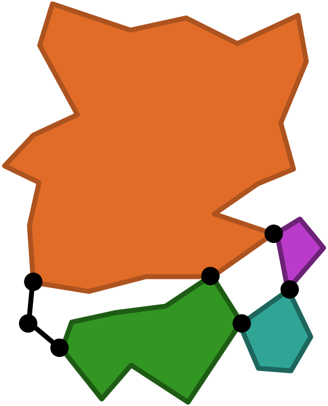


Algorithm Outline

Algorithm

Iteratively grow a set of maximally rigid bodies!

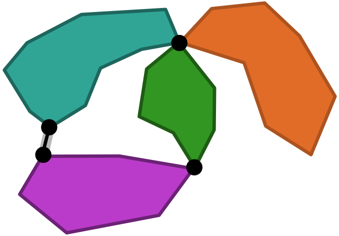
- repeatedly add a body to a set $\mathcal{S}^\star$
- check if $\mathcal{S}^\star$ contains a rigid subset
- merge rigid subsets as soon as possible

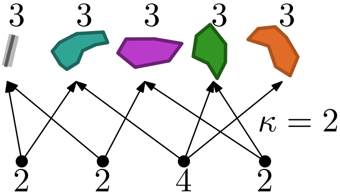


- How can we efficiently test for rigid subsets?

Bipartite Rigidity Flow Network

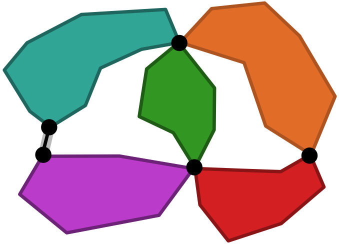
- Maintain a flow network:
  - nodes are bodies and joints
  - arcs are inclusions
  - support is twice the redundancy
  - demand and capacities 3 resp. 2
- Flow network is updated after adding a body to $\mathcal{S}^\star$
- maximum flows reveal rigid sets
  - Candidates are *closures* over each neighbored body in the residual graph
- Flows can be reused, bounding the time to $\mathcal{O}(n + l \log l + k^2)$ for k bodies and l joints (overall)

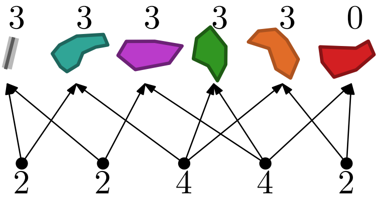




Bipartite Rigidity Flow Network

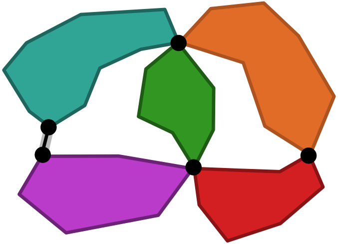
- Maintain a flow network:
  - nodes are bodies and joints
  - arcs are inclusions
  - support is twice the redundancy
  - demand and capacities 3 resp. 2
- Flow network is updated after adding a body to $\mathcal{S}^\star$
- maximum flows reveal rigid sets
  - Candidates are *closures* over each neighbored body in the residual graph
- Flows can be reused, bounding the time to $\mathcal{O}(n + l \log l + k^2)$ for k bodies and l joints (overall)

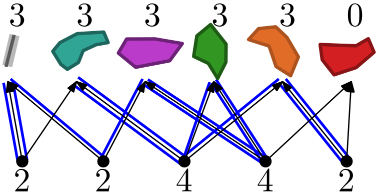




Bipartite Rigidity Flow Network

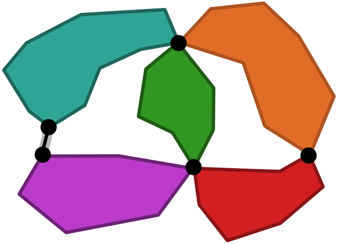
- Maintain a flow network:
  - nodes are bodies and joints
  - arcs are inclusions
  - support is twice the redundancy
  - demand and capacities 3 resp. 2
- Flow network is updated after adding a body to $\mathcal{S}^\star$
- maximum flows reveal rigid sets
  - Candidates are *closures* over each neighbored body in the residual graph
- Flows can be reused, bounding the time to $\mathcal{O}(n + l \log l + k^2)$ for k bodies and l joints (overall)

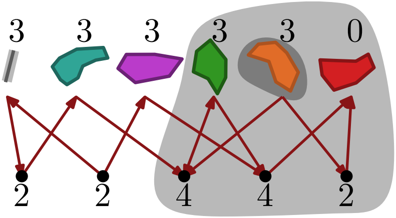




Bipartite Rigidity Flow Network

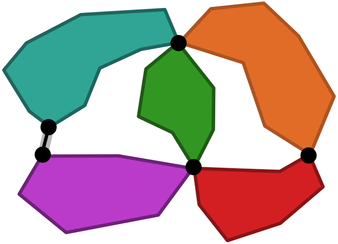
- Maintain a flow network:
  - nodes are bodies and joints
  - arcs are inclusions
  - support is twice the redundancy
  - demand and capacities 3 resp. 2
- Flow network is updated after adding a body to $\mathcal{S}^\star$
- maximum flows reveal rigid sets
  - Candidates are *closures* over each neighbored body in the residual graph
- Flows can be reused, bounding the time to $\mathcal{O}(n + l \log l + k^2)$ for k bodies and l joints (overall)

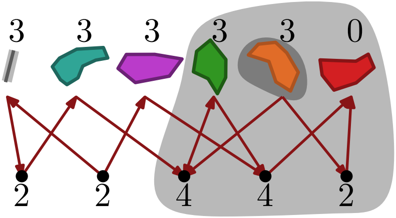




Bipartite Rigidity Flow Network

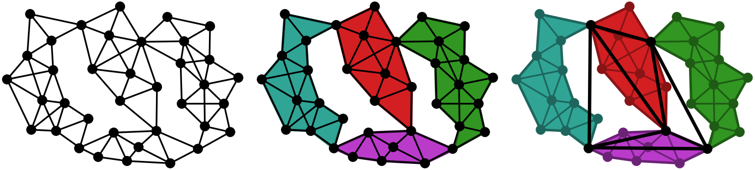
- Maintain a flow network:
  - nodes are bodies and joints
  - arcs are inclusions
  - support is twice the redundancy
  - demand and capacities 3 resp. 2
- Flow network is updated after adding a body to $\mathcal{S}^\star$
- maximum flows reveal rigid sets
  - Candidates are *closures* over each neighbored body in the residual graph
- Flows can be reused, bounding the time to $\mathcal{O}(n + l \log l + k^2)$ for k bodies and l joints (overall)





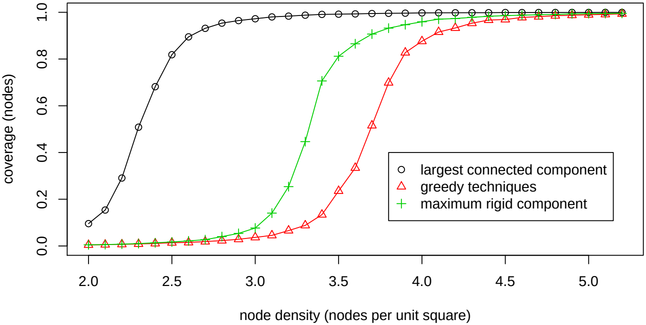
Layout

- Finding a consistent realization can be formulated as an LP
- For rigid subgraphs, solving a system of linear equations suffices
- Solving independend subproblems first reduces costs dramatically



- $\Rightarrow$ always keep a realization for all rigid bodies!

Experimental Results on Geometric Graphs



- Maximum rigid components demand less density.
- Greedy techniques really reduce the number of bodies.
- Layout subproblems almost always have constant size.
  - costs of layout become negligible

Conclusion

- Direction-based localization is the only case with
  - tight characterization of uniqueness
  - polynomial-time realization
- Our approach
  - solves the problem of finding rigid components in body-joint frameworks
  - allows to use fast & intuitive techniques first
  - is adapted to geometric graphs (at asymptotically no cost)
  - removes the bottleneck by iteratively solving the layout problem

Thank you for your attention.