Formal Proof of a Gathering Algorithm in the Pactole Framework

Mathis Bouverot-Dupuis

Boss : Pierre Courtieu (CNAM)

June - July 2022

Talk outline

Suzuki & Yamashita's Model

Weber Point Properties

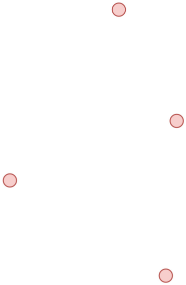
Alignment

Gathering

Suzuki & Yamashita's Model

Very simple (dumb) robots :

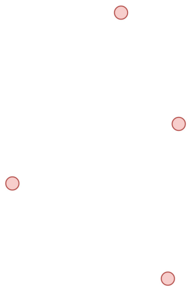
- Points in $\mathbb{R}^2$ (can overlap)



Suzuki & Yamashita's Model

Very simple (dumb) robots :

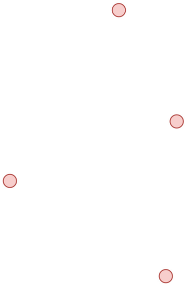
- Points in $\mathbb{R}^2$ (can overlap)
- Anonymous



Suzuki & Yamashita's Model

Very simple (dumb) robots :

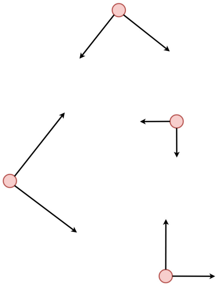
- Points in $\mathbb{R}^2$ (can overlap)
- Anonymous
- No direct communication



Suzuki & Yamashita's Model

Very simple (dumb) robots :

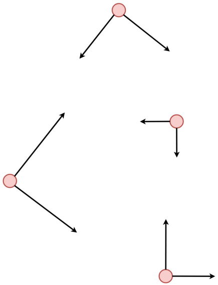
- Points in $\mathbb{R}^2$ (can overlap)
- Anonymous
- No direct communication
- No common direction/scale



Suzuki & Yamashita's Model

Very simple (dumb) robots :

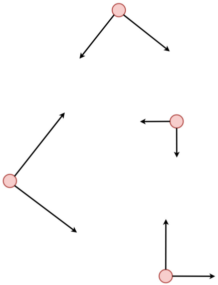
- Points in $\mathbb{R}^2$ (can overlap)
- Anonymous
- No direct communication
- No common direction/scale
- Strong multiplicity detection



Suzuki & Yamashita's Model

Very simple (dumb) robots :

- Points in $\mathbb{R}^2$ (can overlap)
- Anonymous
- No direct communication
- No common direction/scale
- Strong multiplicity detection
- Same robogram



Weber point : Definition

- Let X : multiset of points in $\mathbb{R}^2$

Weber point : Definition

- Let X : multiset of points in $\mathbb{R}^2$
- Sum of distances to X :

$$D_X(p) := \sum_{x \in X} \|x - p\|$$

Weber point : Definition

- Let X : multiset of points in $\mathbb{R}^2$
- Sum of distances to X :

$$D_X(p) := \sum_{x \in X} \|x - p\|$$

- Set of weber points of X :

$$\{p \mid p \text{ minimizes } D_X\}$$

Weber point : Definition

- Let X : multiset of points in $\mathbb{R}^2$
- Sum of distances to X :

$$D_X(p) := \sum_{x \in X} \|x - p\|$$

- Set of weber points of X :

$$\{p \mid p \, \text{minimizes} \, D_X\}$$

- Similar to barycenter (sum of distances v.s. sum of distances squared).

Weber point v.s Barycenter

	barycenter	weber point
exists		
unique		
computable		

Weber point v.s Barycenter

	barycenter	weber point
exists	Yes	Yes
unique		
computable		

Weber point v.s Barycenter

	barycenter	weber point
exists	Yes	Yes
unique	Yes	No
computable		

Weber point v.s Barycenter

	barycenter	weber point
exists	Yes	Yes
unique	Yes	No
computable	Yes	No

Weber point v.s Barycenter

	barycenter	weber point
exists	Yes	Yes
unique	Yes	No
computable	Yes	No

When is the weber point unique ?

Weber point v.s Barycenter

	barycenter	weber point
exists	Yes	Yes
unique	Yes	No
computable	Yes	No

When is the weber point unique ?

- Points not aligned.

Weber point v.s Barycenter

	barycenter	weber point
exists	Yes	Yes
unique	Yes	No
computable	Yes	No

When is the weber point unique ?

- Points not aligned.
- Odd number of points.

Weber point v.s Barycenter

	barycenter	weber point
exists	Yes	Yes
unique	Yes	No
computable	Yes	No

When is the weber point unique ?

- Points not aligned.
- Odd number of points.
- Otherwise : sometimes.

Weber point v.s Barycenter

	barycenter	weber point
exists	Yes	Yes
unique	Yes	No
computable	Yes	No

When is the weber point unique ?

- Points not aligned.
- Odd number of points.
- Otherwise : sometimes.

How about computability ?

Weber point v.s Barycenter

	barycenter	weber point
exists	Yes	Yes
unique	Yes	No
computable	Yes	No

When is the weber point unique ?

- Points not aligned.
- Odd number of points.
- Otherwise : sometimes.

How about computability ?

- We don't care (toy example).

Weber point v.s Barycenter

	barycenter	weber point
exists	Yes	Yes
unique	Yes	No
computable	Yes	No

When is the weber point unique ?

- Points not aligned.
- Odd number of points.
- Otherwise : sometimes.

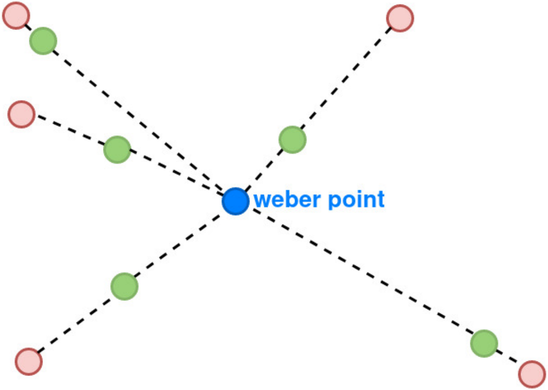
How about computability ?

- We don't care (toy example).

Why use the weber point ?

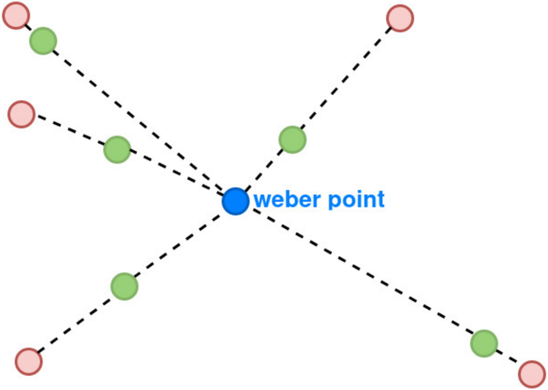
Contraction Lemma

- Let $\mathbf{X}$, $\mathbf{Y}$: multiset of points (cf. figure).



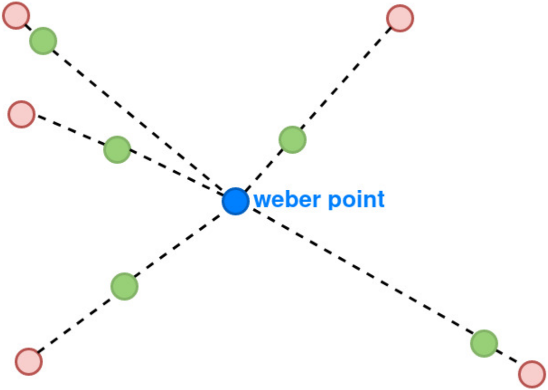
Contraction Lemma

- Let $\mathbf{X}$, $\mathbf{Y}$: multiset of points (cf. figure).
- Suppose $w \in WP(X)$.



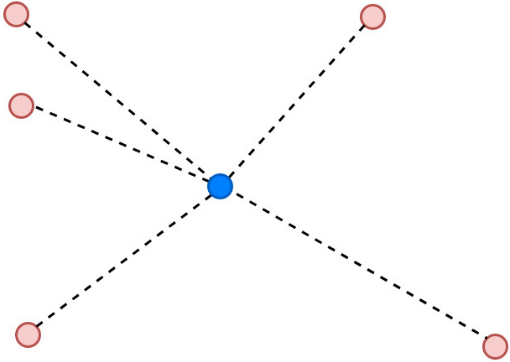
Contraction Lemma

- Let **X**, **Y** : multiset of points (cf. figure).
- Suppose $w \in WP(X)$.
- Then $w \in WP(Y)$.



Contraction Lemma (proof idea)

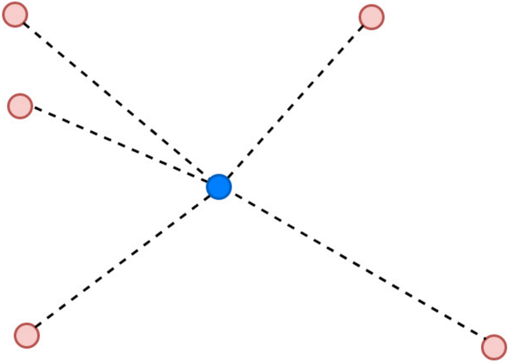
Alternate characterization of weber points :



Contraction Lemma (proof idea)

Alternate characterization of weber points :

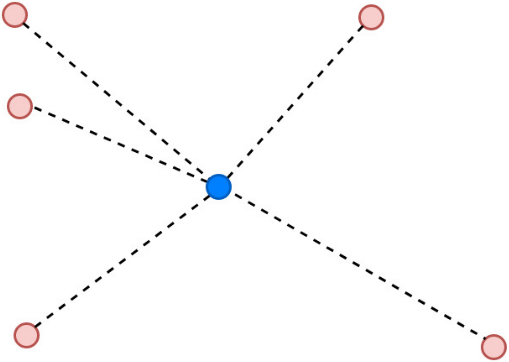
- Recall $D_X(p) := \sum_{x \in X} \|x - p\|$.



Contraction Lemma (proof idea)

Alternate characterization of weber points :

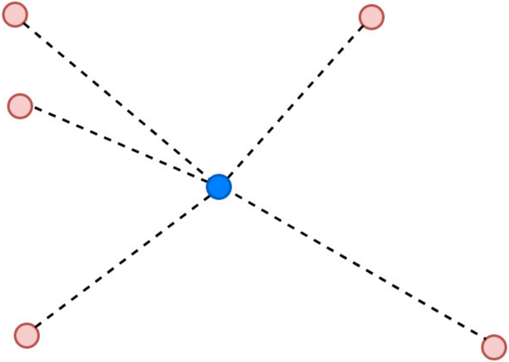
- Recall $D_X(p) := \sum_{x \in X} \|x - p\|$.
- D_X is convex & differentiable (almost) everywhere.



Contraction Lemma (proof idea)

Alternate characterization of weber points :

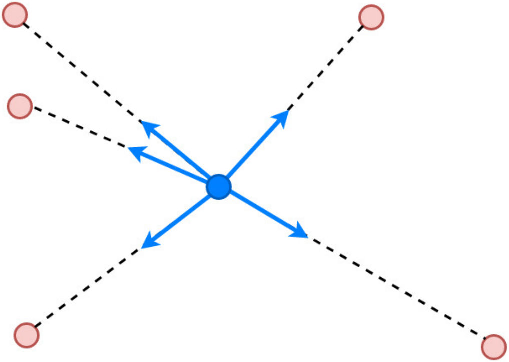
- Recall $D_X(p) := \sum_{x \in X} \|x - p\|$.
- D_X is convex & differentiable (almost) everywhere.
- Minimizing $D_X \iff$ gradient of D_x is 0.



Contraction Lemma (proof idea)

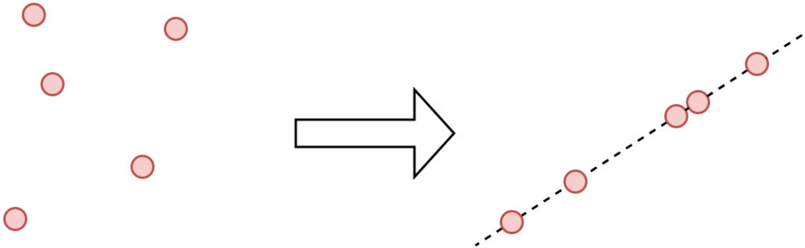
Alternate characterization of weber points :

- Recall $D_X(p) := \sum_{x \in X} \|x - p\|$.
- D_X is convex & differentiable (almost) everywhere.
- Minimizing $D_X \iff$ gradient of D_x is 0.
- Gradient : $\nabla D_X(p) = \sum_{x \in X} \frac{p-x}{\|p-x\|}$



Alignment

Goal : move robots to a common line, and make them stay on the line.



Alignment

Goal : move robots to a common line, and make them stay on the line.

Coq

```
Definition aligned : configuration → Prop.
```

Alignment

Goal : move robots to a common line, and make them stay on the line.

Coq
```
Definition aligned : configuration → Prop.
```

Coq
```
Definition eventually_aligned (c:configuration)
(d:demon) (r:robogram) :=
  Stream.eventually
    (Stream.forever (Stream.instant aligned))
    (execute r d c).
```

Alignment

Goal : move robots to a common line, and make them stay on the line.

Coq
```
Definition aligned : configuration → Prop.
```

Coq
```
Definition eventually_aligned (c:configuration)
(d:demon) (r:robogram) :=
  Stream.eventually
    (Stream.forever (Stream.instant aligned))
    (execute r d c).
```

Coq
```
Theorem alignment_correct :=
  ∀ (c:configuration) (d:demon),
    (* Hypotheses on d *) →
    eventually_aligned c d align_robogram.
```

Robogram

Very simple robogram : move towards the weber point (in a straight line) until aligned.

Robogram

Very simple robogram : move towards the weber point (in a straight line) until aligned.

Coq

```
Definition align_robogram (obs:observation) : R2 :=
  if aligned_dec obs
  then origin
  else weber_calc obs.
```

Robogram

Very simple robogram : move towards the weber point (in a straight line) until aligned.

```coq
Definition align_robogram (obs:observation) : R2 :=
  if aligned_dec obs
  then origin
  else weber_calc obs.
```

```coq
Definition observation := multiset R2.
```

Setting

Suzuki & Yamashita's Model : several versions.

- Fully-synchronous (FSYNC), semi-synchronous (SSYNC) or asynchronous (ASYNC) ?

Setting

Suzuki & Yamashita's Model : several versions.

- Fully-synchronous (FSYNC), semi-synchronous (SSYNC) or asynchronous (ASYNC) ?
- Rigid or flexible ?

Setting

Suzuki & Yamashita's Model : several versions.

- Fully-synchronous (FSYNC), semi-synchronous (SSYNC) or asynchronous (ASYNC) ?
- Rigid or flexible ?

My proofs (alignment) :

Setting

Suzuki & Yamashita's Model : several versions.

- Fully-synchronous (FSYNC), semi-synchronous (SSYNC) or asynchronous (ASYNC) ?
- Rigid or flexible ?

My proofs (alignment) :

- SSYNC & rigid.

Setting

Suzuki & Yamashita's Model : several versions.

- Fully-synchronous (FSYNC), semi-synchronous (SSYNC) or asynchronous (ASYNC) ?
- Rigid or flexible ?

My proofs (alignment) :

- SSYNC & rigid.
- SSYNC & flexible.

Setting

Suzuki & Yamashita's Model : several versions.

- Fully-synchronous (FSYNC), semi-synchronous (SSYNC) or asynchronous (ASYNC) ?
- Rigid or flexible ?

My proofs (alignment) :

- SSYNC & rigid.
- SSYNC & flexible.
- ASYNC (first time in Pactole) & flexible.

ASYNC model

How do we represent robots ?

ASYNC model

How do we represent robots ?

- SSYNC : current position only.

ASYNC model

How do we represent robots ?

- SSYNC : current position only.
- ASYNC : start, destination & current positions.

ASYNC model

How do we represent robots ?

- SSYNC : current position only.
- ASYNC : start, destination & current positions.

Coq

```
Definition info := (location * location * ratio)%type.
```

ASYNC model

How do we represent robots ?

- SSYNC : current position only.
- ASYNC : start, destination & current positions.

Coq

```
Definition info := (location * location * ratio)%type.
```

Coq

```
Instance St : State info :=
{ get_location := fun '(start, dest, r) ⇒
    straight_path start dest r }.
```

ASYNC model

How do we represent robots ?

- SSYNC : current position only.
- ASYNC : start, destination & current positions.

Coq
```
Definition info := (location * location * ratio)%type.
```

Coq
```
Instance St : State info :=
{ get_location := fun '(start, dest, r) ⇒
    straight_path start dest r }.
```

How to update robots each round ?

ASYNC model

How do we represent robots ?

- SSYNC : current position only.
- ASYNC : start, destination & current positions.

Coq
```
Definition info := (location * location * ratio)%type.
```

Coq
```
Instance St : State info :=
{ get_location := fun '(start, dest, r) ⇒
    straight_path start dest r }.
```

How to update robots each round ?

- Activated robots :

```
new_start <- straight_path start dest ratio
new_dest  <- robogram (obs_from_config config)
new_ratio <- 0
```

ASYNC model

How do we represent robots ?

- SSYNC : current position only.
- ASYNC : start, destination & current positions.

Coq
```
Definition info := (location * location * ratio)%type.
```

Coq
```
Instance St : State info :=
{ get_location := fun '(start, dest, r) ⇒
    straight_path start dest r }.
```

How to update robots each round ?

- Activated robots :

```
new_start <- straight_path start dest ratio
new_dest  <- robogram (obs_from_config config)
new_ratio <- 0
```

- Other robots :

```
new_ratio <- ratio + demon_ratio
```

ASYNC model : rigid & flexible

How to define rigid & flexible demons in ASYNC ?

ASYNC model : rigid & flexible

How to define rigid & flexible demons in ASYNC ?

Rigid : no longer the default setting.

```
Coq
Definition rigid_da_prop (da:demonic_action) :=
  ∀ (c:configuration) (id:ident),
    da.(activate) id = true →
    get_location (c id) ≡ get_destination (c id).
```

ASYNC model : rigid & flexible

How to define rigid & flexible demons in ASYNC ?

Rigid : no longer the default setting.

Coq
```
Definition rigid_da_prop (da:demonic_action) :=
  ∀ (c:configuration) (id:ident),
    da.(activate) id = true →
    get_location (c id) ≡ get_destination (c id).
```

Flexible :

Coq
```
Definition flex_da_prop (da:demonic_action) (δ:R) :=
  ∀ (c:configuration) (id:ident),
    da.(activate) id = true →
    get_location (c id) ≡ get_destination (c id)  ∨
    δ ≤ dist (get_start (c id)) (get_location (c id)).
```

Decreasing measure

We define a measure that decreases each time a robot that isn't on the weber point is activated.

- Simple case : SSYNC & rigid.

Decreasing measure

We define a measure that decreases each time a robot that isn't on the weber point is activated.

- Simple case : SSYNC & rigid.
- Count how many robots have not arrived yet.

Decreasing measure

We define a measure that decreases each time a robot that isn't on the weber point is activated.

- Simple case : SSYNC & rigid.
- Count how many robots have not arrived yet.

Coq

```
Definition measure (c:configuration) :=
  n - countA_occ _ _ (weber_calc c) c.
```

Decreasing measure

We define a measure that decreases each time a robot that isn't on the weber point is activated.

- Simple case : SSYNC & rigid.
- Count how many robots have not arrived yet.

Coq
```
Definition measure (c:configuration) :=
  n - countA_occ _ _ (weber_calc c) c.
```

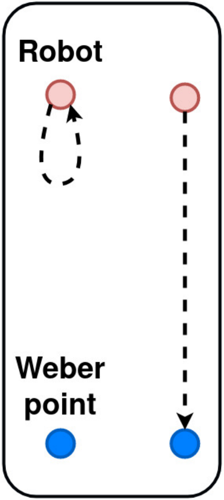
Coq
```
Lemma round_decreases_measure :
  ∀ (c:config) (da:demonic_action),
    (* Hypotheses on da *) →
    (∃ id, da.(activate) id = true  ∧
      get_location (c id) =/= weber_calc c) →
   aligned (round gatherW da c).
```

Decreasing measure (ASYNC & flex)

Lifecycle of a robot in ASYNC flex :

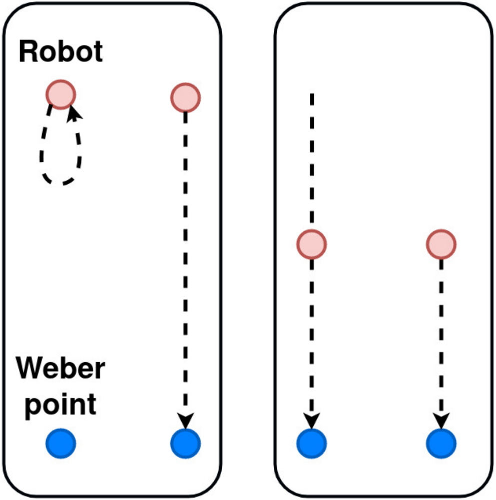
Decreasing measure (ASYNC & flex)

Lifecycle of a robot in ASYNC flex :



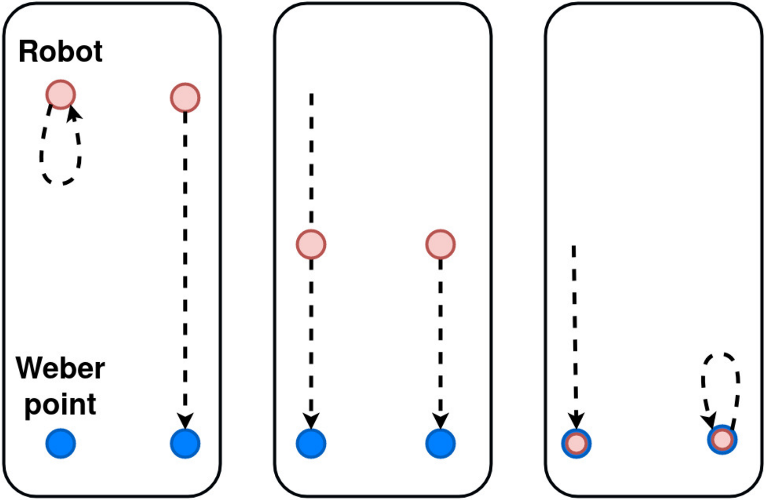
Decreasing measure (ASYNC & flex)

Lifecycle of a robot in ASYNC flex :



Decreasing measure (ASYNC & flex)

Lifecycle of a robot in ASYNC flex :



Decreasing measure (ASYNC & flex)

We need one measure for each type of activation.

Decreasing measure (ASYNC & flex)

We need one measure for each type of activation.

- $\mathbf{measure_1}$: count how many robots are looping but not on the weber point.

Decreasing measure (ASYNC & flex)

We need one measure for each type of activation.

- $\mathbf{measure_1}$: count how many robots are looping but not on the weber point.
- $\mathbf{measure_2}$: count the total distance from the start of each robot to the weber point.

Decreasing measure (ASYNC & flex)

We need one measure for each type of activation.

- **measure_1** : count how many robots are looping but not on the weber point.
- **measure_2** : count the total distance from the start of each robot to the weber point.
- **measure_3** : count how many robots are **not** looping on the weber point.

Decreasing measure (ASYNC & flex)

We need one measure for each type of activation.

- $\mathbf{measure_1}$: count how many robots are looping but not on the weber point.
- $\mathbf{measure_2}$: count the total distance from the start of each robot to the weber point.
- $\mathbf{measure_3}$: count how many robots are **not** looping on the weber point.

The final measure is $\mathbf{measure_1} + \mathbf{measure_2} + \mathbf{measure_3}$.

Decreasing measure (ASYNC & flex)

We need one measure for each type of activation.

- $\mathbf{measure_1}$: count how many robots are looping but not on the weber point.
- $\mathbf{measure_2}$: count the total distance from the start of each robot to the weber point.
- $\mathbf{measure_3}$: count how many robots are **not** looping on the weber point.

The final measure is $\mathbf{measure_1} + \mathbf{measure_2} + \mathbf{measure_3}$.
The proof is then a well-founded induction on the measure.

Decreasing measure (ASYNC & flex)

We need one measure for each type of activation.

- $\mathbf{measure_1}$: count how many robots are looping but not on the weber point.
- $\mathbf{measure_2}$: count the total distance from the start of each robot to the weber point.
- $\mathbf{measure_3}$: count how many robots are **not** looping on the weber point.

The final measure is $\mathbf{measure_1} + \mathbf{measure_2} + \mathbf{measure_3}$.
The proof is then a well-founded induction on the measure.

Coq
```
Definition lt_config (c1 c2:configuration) :=
  0 ≤ measure c1 ≤ measure c2 - min 1 δ.
```

Decreasing measure (ASYNC & flex)

We need one measure for each type of activation.

- **measure$_1$** : count how many robots are looping but not on the weber point.
- **measure$_2$** : count the total distance from the start of each robot to the weber point.
- **measure$_3$** : count how many robots are **not** looping on the weber point.

The final measure is **measure$_1$** + **measure$_2$** + **measure$_3$**.
The proof is then a well-founded induction on the measure.

Coq

```
Definition lt_config (c1 c2:configuration) :=
  0 ≤ measure c1 ≤ measure c2 - min 1 δ.
```

Coq

```
Lemma lt_config_wf : well_founded lt_config.
```

Gathering

Goal : move the robots to a common point, and make them stay there.

Gathering

Goal : move the robots to a common point, and make them stay there.
Does the same algorithm work for gathering ?

Gathering

Goal : move the robots to a common point, and make them stay there.

Does the same algorithm work for gathering ?

- Problem when aligned : weber point is not unique.

Gathering

Goal : move the robots to a common point, and make them stay there.

Does the same algorithm work for gathering ?

- Problem when aligned : weber point is not unique.

Switch to another algorithm once aligned ?

Gathering

Goal : move the robots to a common point, and make them stay there.

Does the same algorithm work for gathering ?

- Problem when aligned : weber point is not unique.

Switch to another algorithm once aligned ?

- Does not work in ASYNC.

Gathering

Goal : move the robots to a common point, and make them stay there.

Does the same algorithm work for gathering ?

- Problem when aligned : weber point is not unique.

Switch to another algorithm once aligned ?

- Does not work in ASYNC.

For which initial configurations does the weber algorithm work ?

Gathering

Goal : move the robots to a common point, and make them stay there.

Does the same algorithm work for gathering ?

- Problem when aligned : weber point is not unique.

Switch to another algorithm once aligned ?

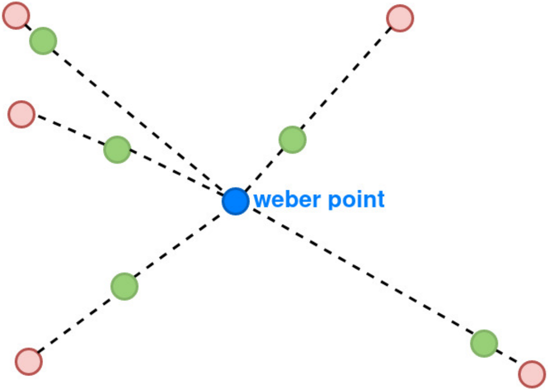
- Does not work in ASYNC.

For which initial configurations does the weber algorithm work ?

- When the weber point is unique (initially).

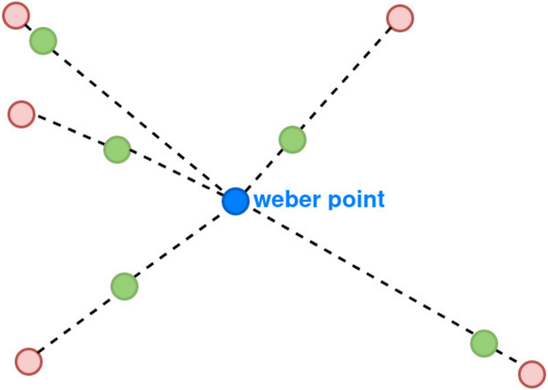
Strong Contraction Lemma (Zohir Bouzid)

- Let **X**, **Y** : multiset of points (cf. figure).



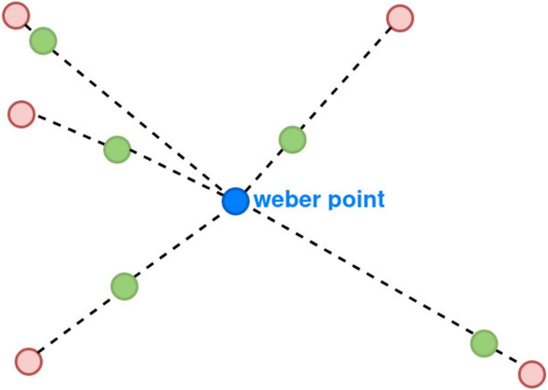
Strong Contraction Lemma (Zohir Bouzid)

- Let **X**, **Y** : multiset of points (cf. figure).
- Suppose w is the **unique** weber point of X.



Strong Contraction Lemma (Zohir Bouzid)

- Let **X**, **Y** : multiset of points (cf. figure).
- Suppose w is the **unique** weber point of X.
- Then w is the **unique** weber point of Y.



Recap

Recap

- Formal proof in Coq of a simple algorithm to solve the gathering problem.

Recap

- Formal proof in Coq of a simple algorithm to solve the gathering problem.
- Formal proof in Coq of the properties of the weber point.

Recap

- Formal proof in Coq of a simple algorithm to solve the gathering problem.
- Formal proof in Coq of the properties of the weber point.
- Assumes the weber point is computable.

Recap

- Formal proof in Coq of a simple algorithm to solve the gathering problem.
- Formal proof in Coq of the properties of the weber point.
- Assumes the weber point is computable.
- Asynchronous and flexible setting.

Recap

- Formal proof in Coq of a simple algorithm to solve the gathering problem.
- Formal proof in Coq of the properties of the weber point.
- Assumes the weber point is computable.
- Asynchronous and flexible setting.
- Not universal : initial position must have a unique weber point.

Recap

- Formal proof in Coq of a simple algorithm to solve the gathering problem.
- Formal proof in Coq of the properties of the weber point.
- Assumes the weber point is computable.
- Asynchronous and flexible setting.
- Not universal : initial position must have a unique weber point.

Thank you !