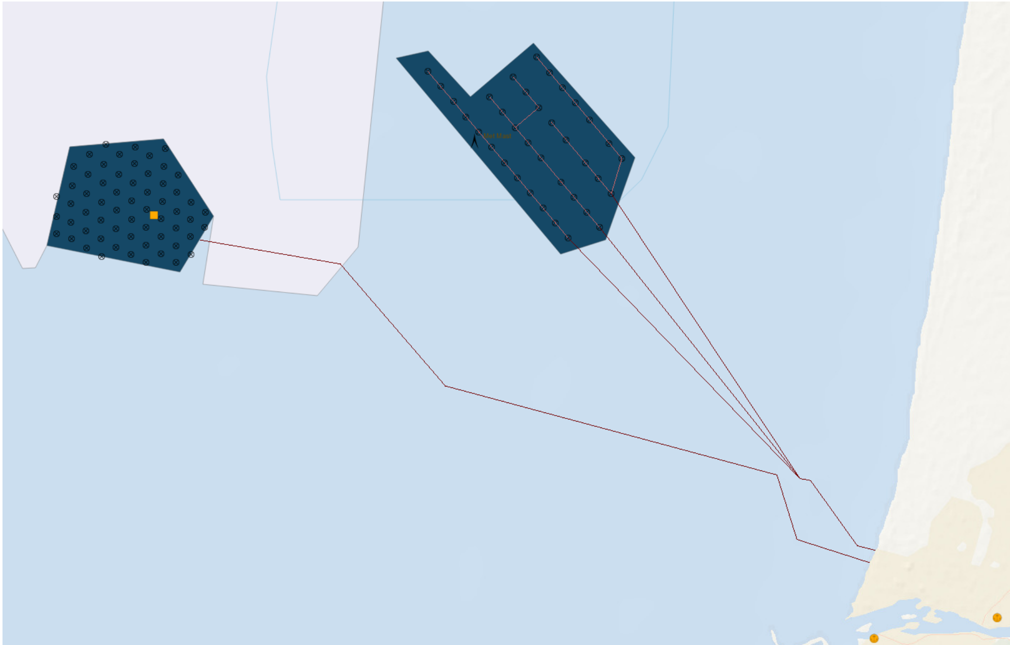
Flexible and efficient site constraint handling for wind farm layout optimization

Erik Quaeghebeur

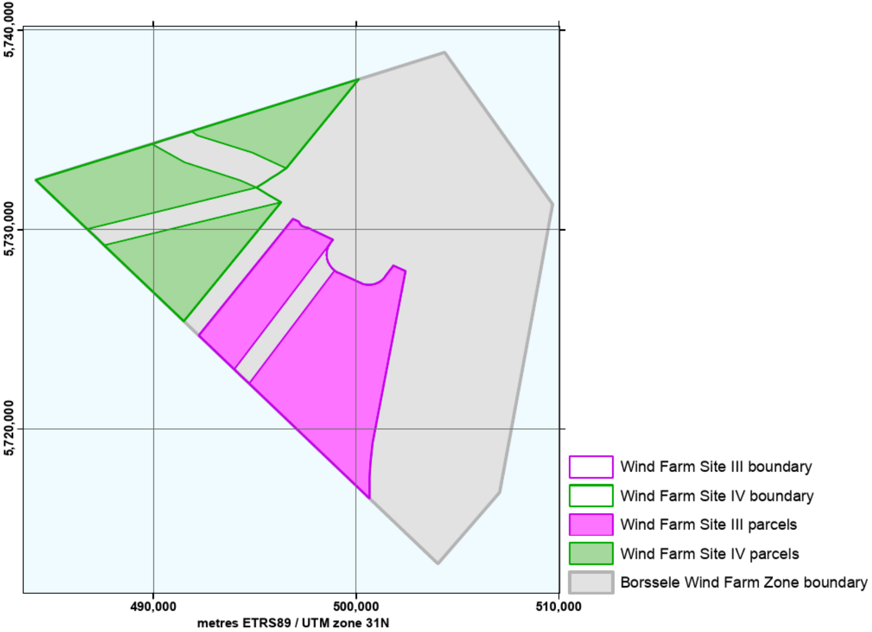
Wind Energy Group — Delft University of Technology

WESC 2019
20 June 2019

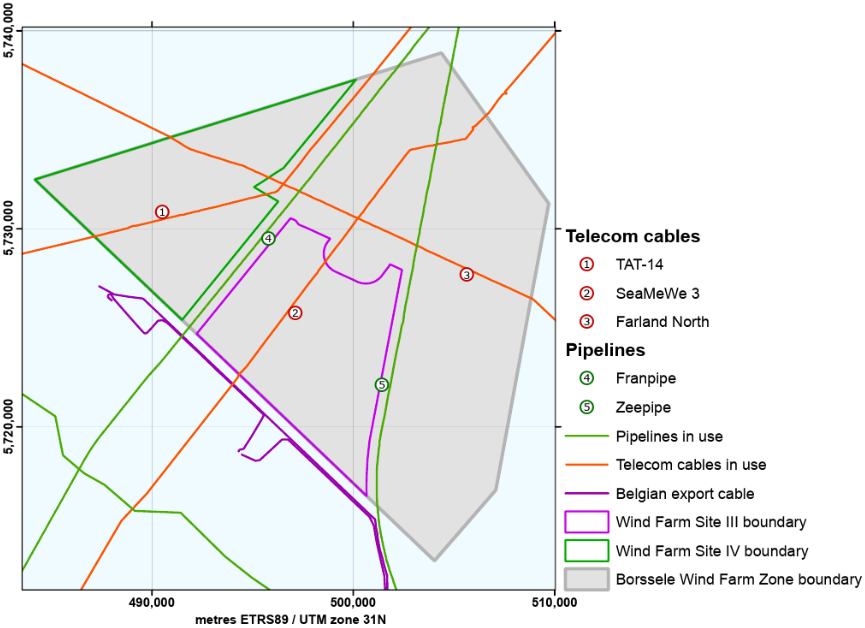
TU Delft



[4C Offshore: https://www.4coffshore.com/]

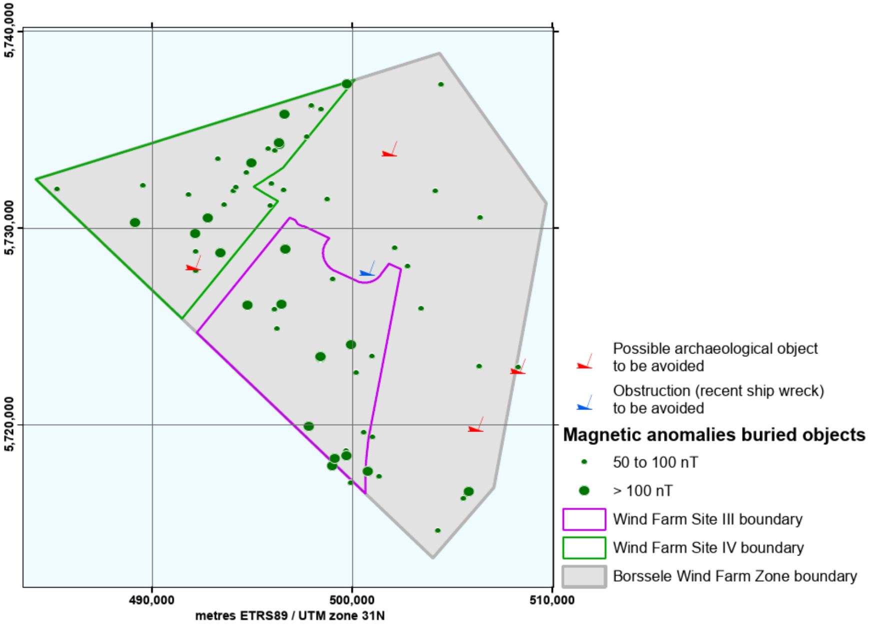


[Netherlands Enterprise Agency (RVO.nl) Borssele Wind Farm Zone: Project and Site Description Wind Farm Sites III and IV (2016-08)]



[Netherlands Enterprise Agency (RVO.nl) Borssele Wind Farm Zone: Project and Site Description Wind Farm Sites III and IV (2016-08)]

TU Delft



[Netherlands Enterprise Agency (RVO.nl) Borssele Wind Farm Zone: Project and Site Description Wind Farm Sites III and IV (2016-08)]

So what do real (offshore) sites look like?

A plate of irregularly-cut pieces of Emmental cheese...

- multiple non-connected parts
- non-convex, with concavities of various sizes
- circular exclusion zones strewn around.

How can we handle constraints for complex sites?

How can we handle constraints for complex sites?

1. Discretize the possible turbine positions
(computationally efficient, but limits optimization approaches)

How can we handle constraints for complex sites?

1. Discretize the possible turbine positions
(computationally efficient, but limits optimization approaches)

2. Divide the site into quadrilaterals and transform those to rectangles
(straightforward, but working in transformed space may be inconvenient)

How can we handle constraints for complex sites?

1. Discretize the possible turbine positions
(computationally efficient, but limits optimization approaches)

2. Divide the site into quadrilaterals and transform those to rectangles
(straightforward, but working in transformed space may be inconvenient)

3. Describe the site as a set of polygonal curves and use a ray shooting algorithm
(flexible, but limited for correcting violations)

How can we handle constraints for complex sites?

1. Discretize the possible turbine positions
(computationally efficient, but limits optimization approaches)

2. Divide the site into quadrilaterals and transform those to rectangles
(straightforward, but working in transformed space may be inconvenient)

3. Describe the site as a set of polygonal curves and use a ray shooting algorithm
(flexible, but limited for correcting violations)

4. *Various approaches I'm not aware of, but which you'll tell me about later*

How can we handle constraints for complex sites?

1. Discretize the possible turbine positions
   (computationally efficient, but limits optimization approaches)
2. Divide the site into quadrilaterals and transform those to rectangles
   (straightforward, but working in transformed space may be inconvenient)
3. Describe the site as a set of polygonal curves and use a ray shooting algorithm
   (flexible, but limited for correcting violations)
4. *Various approaches I'm not aware of, but which you'll tell me about later*
5. Decomposition into nested convex polygons and calculating closest border point
   (both flexible and efficient?)

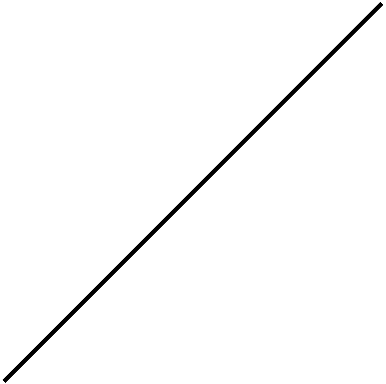
TUDelft

How can we handle constraints for complex sites?

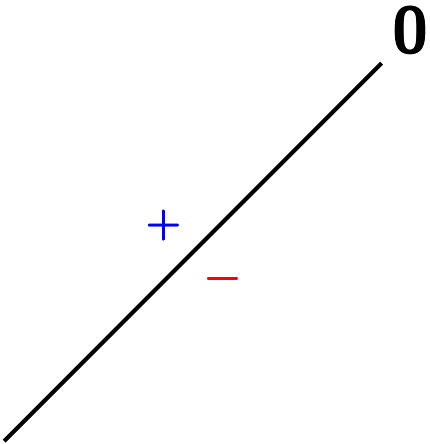
1. Discretize the possible turbine positions
(computationally efficient, but limits optimization approaches)
2. Divide the site into quadrilaterals and transform those to rectangles
(straightforward, but working in transformed space may be inconvenient)
3. Describe the site as a set of polygonal curves and use a ray shooting algorithm
(flexible, but limited for correcting violations)
4. *Various approaches I'm not aware of, but which you'll tell me about later*
5. Decomposition into nested convex polygons and calculating closest border point
(both flexible and efficient?)

Circular constraints need to be added separately to 2, 3, 5!

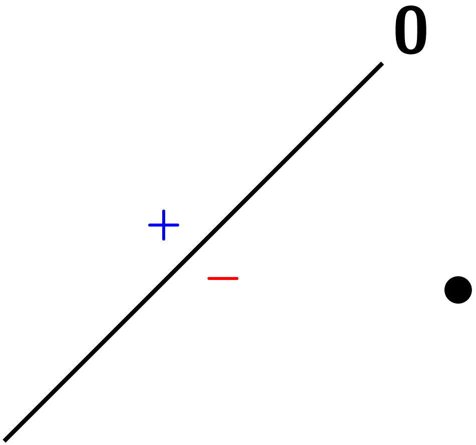
Linear constraints as the basis



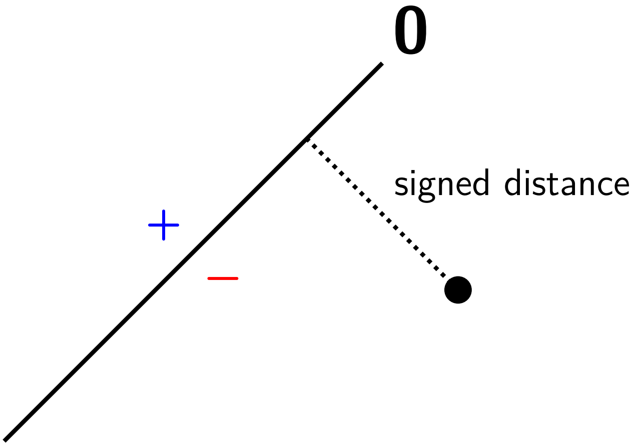
Linear constraints as the basis



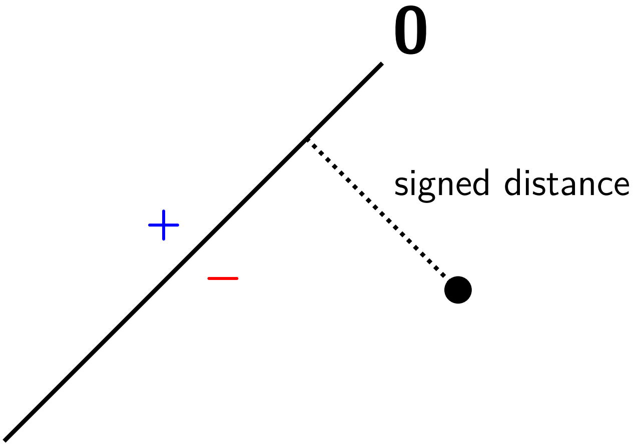
Linear constraints as the basis



Linear constraints as the basis



Linear constraints as the basis



Convex polygons are sets of linear constraints

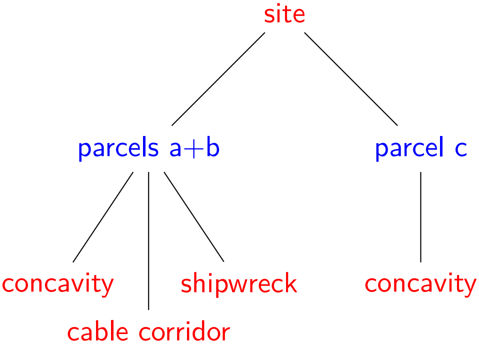
Site decomposition in terms of convex polygons and discs

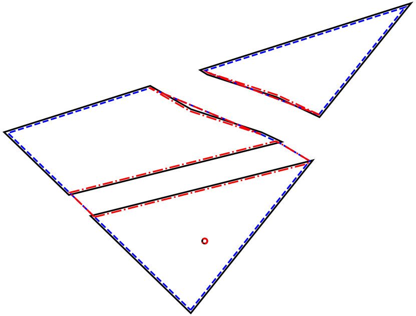
- Sites are described as a tree of convex polygons and discs.
- Levels alternate between included and excluded.
- Needs to be done just once, starting from the parcels' vertex lists.

Site decomposition in terms of convex polygons and discs

- Sites are described as a tree of convex polygons and discs.
- Levels alternate between included and excluded.
- Needs to be done just once, starting from the parcels' vertex lists.

Example for Borssele IV:

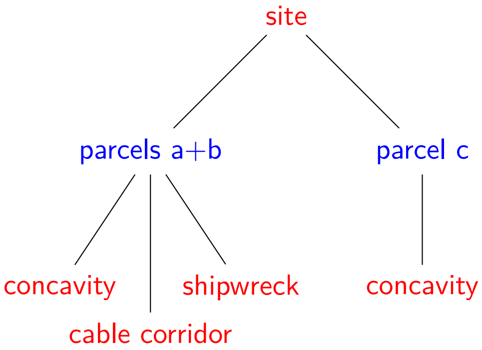


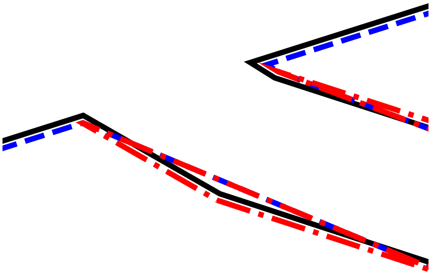


Site decomposition in terms of convex polygons and discs

- Sites are described as a tree of convex polygons and discs.
- Levels alternate between included and excluded.
- Needs to be done just once, starting from the parcels' vertex lists.

Example for Borssele IV:

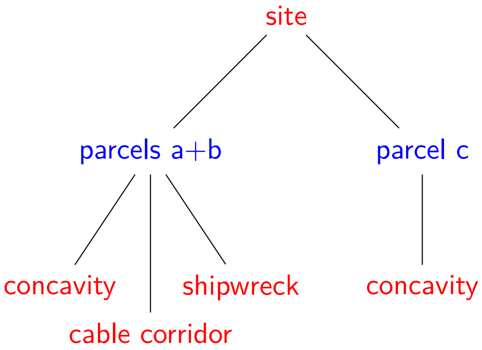


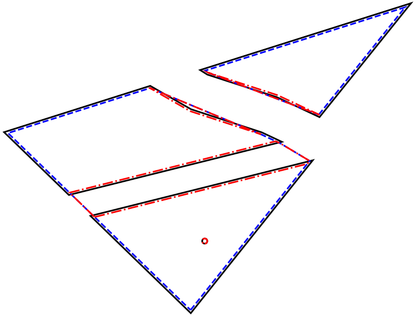


Site decomposition in terms of convex polygons and discs

- Sites are described as a tree of convex polygons and discs.
- Levels alternate between included and excluded.
- Needs to be done just once, starting from the parcels' vertex lists.

Example for Borssele IV:

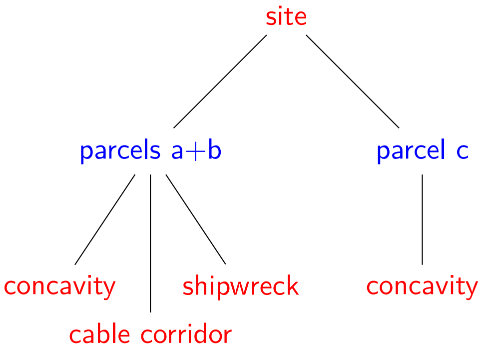


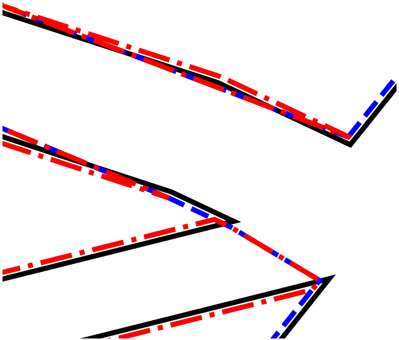


Site decomposition in terms of convex polygons and discs

- Sites are described as a tree of convex polygons and discs.
- Levels alternate between included and excluded.
- Needs to be done just once, starting from the parcels' vertex lists.

Example for Borssele IV:

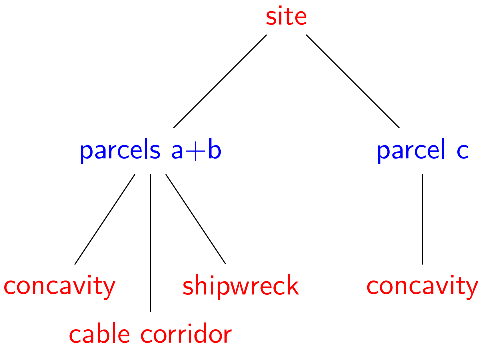


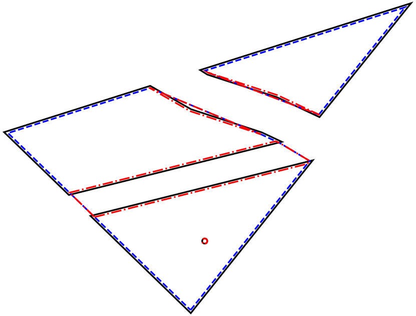


Site decomposition in terms of convex polygons and discs

- Sites are described as a tree of convex polygons and discs.
- Levels alternate between included and excluded.
- Needs to be done just once, starting from the parcels' vertex lists.

Example for Borssele IV:



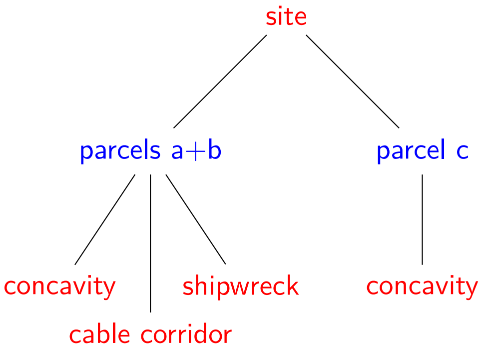


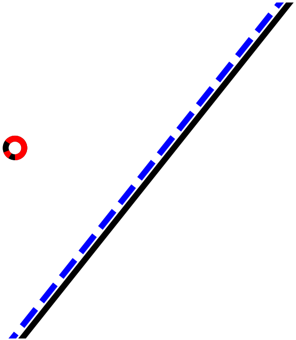
TU Delft

Site decomposition in terms of convex polygons and discs

- Sites are described as a tree of convex polygons and discs.
- Levels alternate between included and excluded.
- Needs to be done just once, starting from the parcels' vertex lists.

Example for Borssele IV:

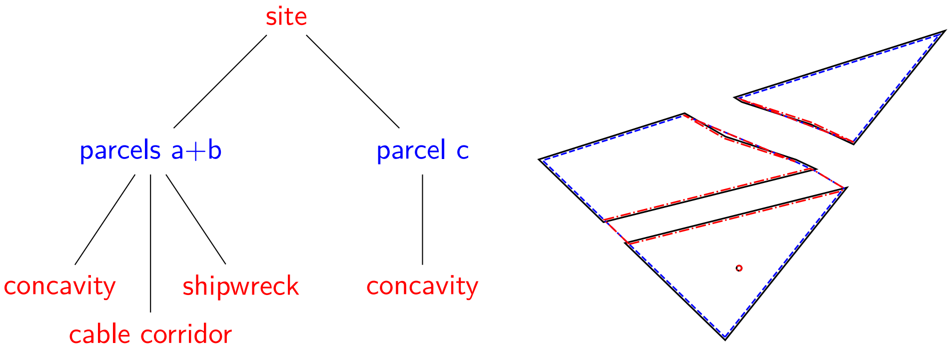




Site decomposition in terms of convex polygons and discs

- Sites are described as a tree of convex polygons and discs.
- Levels alternate between included and excluded.
- Needs to be done just once, starting from the parcels' vertex lists.

Example for Borssele IV:



Site decomposition in terms of convex polygons and discs

```yaml
radius: 8.4628
location:
  - reference_system: ETRS89
  - utm_zone: 31
  - utm: [492539.33, 5733792.49]
roughness: 0.0001
parcels:
  - constraints: # IVa + IVb
      - {"x": -0.871282, "y": -0.899564, "1": -1., rotor_constraint: true} # WFZ_4, P_75
      - {"x":  1., "y": -0.798527, "1": -0.669320, rotor_constraint: true} # P_75, P_74
      - {"x":  0.598875, "y":  1., "1":  0.020557, rotor_constraint: true} # P_74, P_73
      - {"x":  0.470030, "y":  1., "1":  0.059057, rotor_constraint: true} # P_73, P_72
      - {"x":  0.416783, "y":  1., "1":  0.069992, rotor_constraint: true} # P_72, P_70
      - {"x": -0.316693, "y":  1., "1": -0.158119, rotor_constraint: true} # P_70, WFZ_4
    exclusions:
      - constraints:
          - {"x":  0.249945, "y": -1., "1": -0.395467, rotor_constraint: true}
          - {"x": -0.249356, "y":  1., "1":  0.274018, rotor_constraint: true}
      - constraints:
          - {"x": -0.573028, "y": -1., "1": -0.118584, rotor_constraint: true}
          - {"x": -0.326646, "y": -1., "1": -0.088503, rotor_constraint: true}
      - circle: # 100_m buffer around archeological shipwreck
          {center: [-0.058554, -0.658767], radius: 0.011816}
  - constraints: # IVc
      - {"x": -0.316691, "y":  1., "1": -0.158109, rotor_constraint: true} # P_82, P_78
      - {"x":  1., "y": -0.803943, "1": -0.541463, rotor_constraint: true} # P_78, P_79
      - {"x": -0.379160, "y": -1., "1":  0.094044, rotor_constraint: true} # P_79, P_81
      - {"x": -0.625480, "y": -1., "1":  0.082657, rotor_constraint: true} # P_81, P_82
    exclusions:
      - constraints:
          - {"x": 0.324121, "y": 1., "1": -0.096589, rotor_constraint: true}
          - {"x": 0.468405, "y": 1., "1": -0.136250, rotor_constraint: true}
```

TU Delft

Deduced from the vertex lists for the parcel boundaries

```
boundaries:
  - polygon: # IVa
      - [-0.98795236, -0.15475832] # WFZ_4
      - [-0.68775564, -0.4455137 ] # P_77 ← not in vertices of IVa+b
      - [ 0.29881097, -0.19950719] # P_73
      - [ 0.20536654, -0.15558547] # P_72
      - [-0.12209072, -0.04862303] # P_71 ← not in vertices of IVa and IVa+b
      - [-0.31100018,  0.0596274 ] # P_70
  - polygon: # IVb
      - [-0.58775308, -0.54237308] # P_76 ← not in vertices of IVa+b
      - [-0.12313056, -0.99239048] # P_75
      - [ 0.44168357, -0.28507005] # P_74
    exclusions:
      - circle: # 100 m buffer around archeological shipwreck
          {center: [-0.058554, -0.658767], radius: 0.011816}
  - polygon: # IVc
      - [-0.08008326,  0.13274754] # P_82
      - [-0.04622915,  0.11157248] # P_81
      - [ 0.27488267,  0.00749325] # P_80 ← not in vertices of IVc
      - [ 0.47291443, -0.08526582] # P_79
      - [ 0.89693565,  0.44216109] # P_78
```

Checking site constraints efficiently

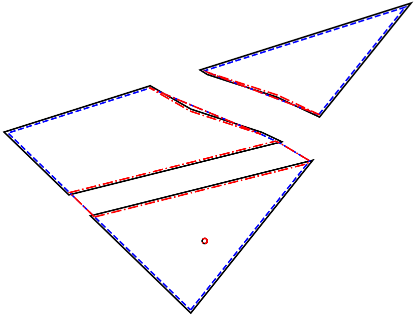
- Walk the tree from root to leaves.
- Only check the turbines inside the parent.

Checking site constraints efficiently

- Walk the tree from root to leaves.
- Only check the turbines inside the parent.

Example for Borssele IV:

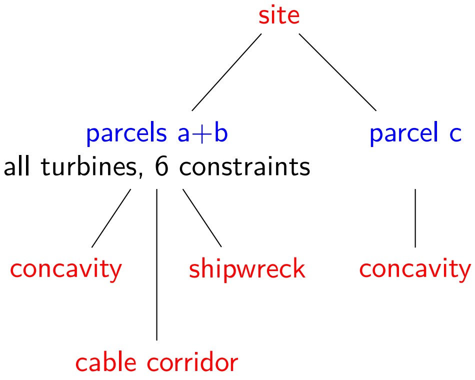
- site
  - parcels a+b
    - concavity
    - cable corridor
    - shipwreck
  - parcel c
    - concavity

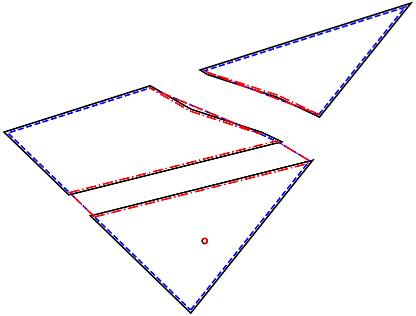


Checking site constraints efficiently

- Walk the tree from root to leaves.
- Only check the turbines inside the parent.

Example for Borssele IV:





Checking site constraints efficiently

- Walk the tree from root to leaves.
- Only check the turbines inside the parent.

Example for Borssele IV:

site

parcels a+b
all turbines, 6 constraints

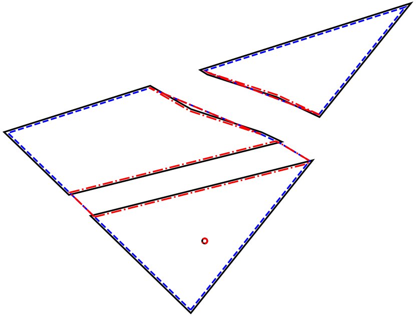
parcel c
all t's, 3 c's

concavity

cable corridor

shipwreck

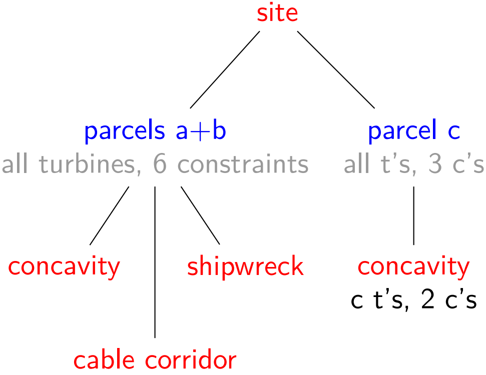
concavity

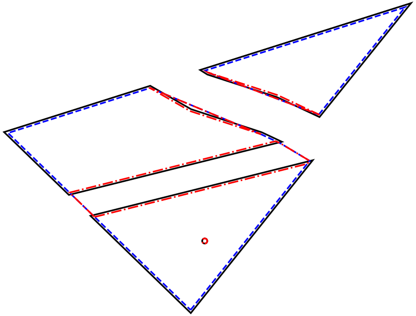


Checking site constraints efficiently

- Walk the tree from root to leaves.
- Only check the turbines inside the parent.

Example for Borssele IV:

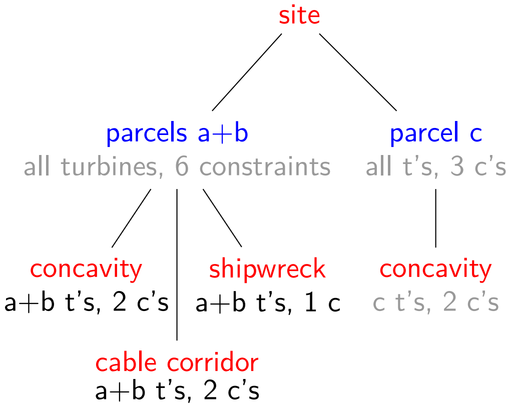


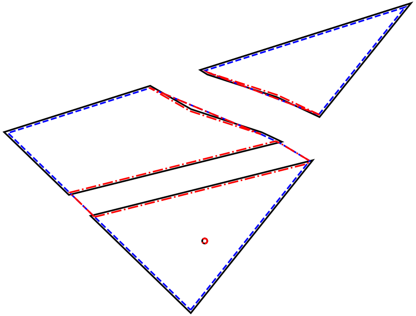


Checking site constraints efficiently

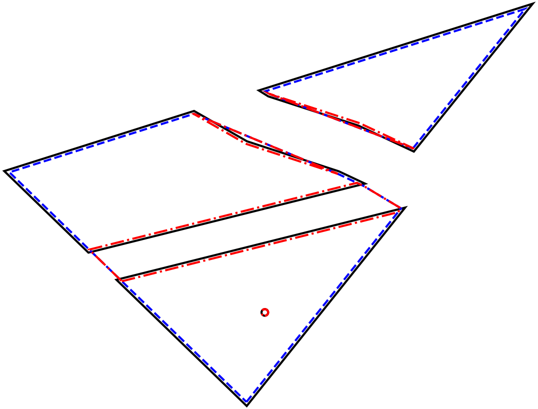
- Walk the tree from root to leaves.
- Only check the turbines inside the parent.

Example for Borssele IV:



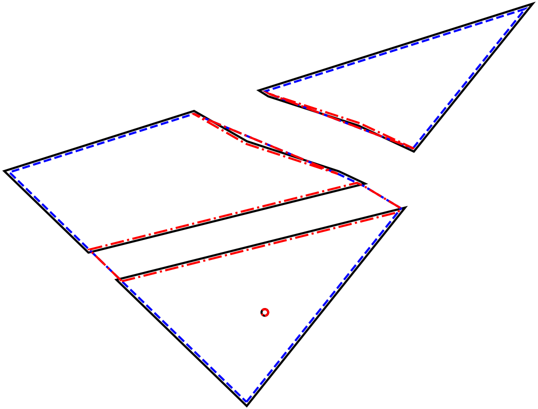


Correcting constraint violations: *move to the closest border point*



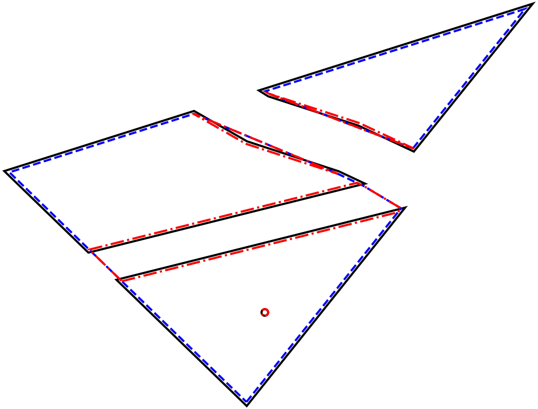
Correcting constraint violations: *move to the closest border point*

1. Assume constraint check done;
   only consider turbines violating site constraints.



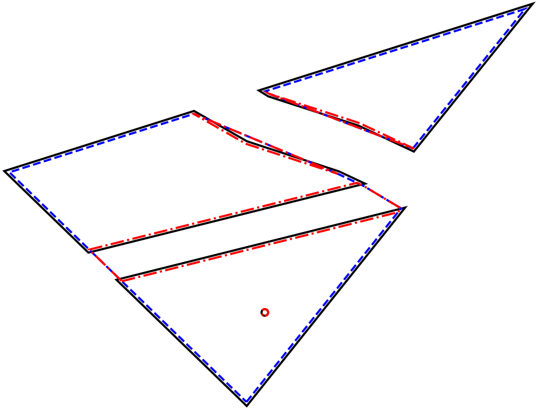
Correcting constraint violations: *move to the closest border point*

1. Assume constraint check done;
   only consider turbines violating site constraints.
2. Determine closest point on *all* violated constraints (the *candidates*).



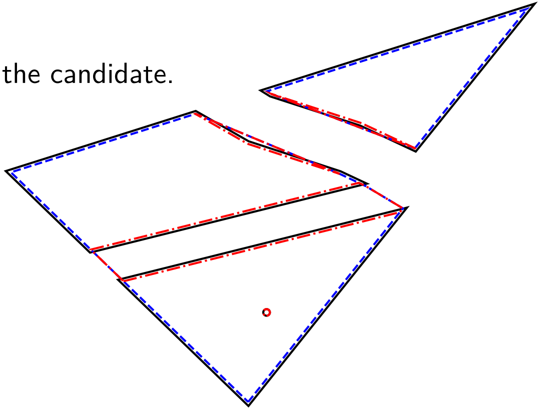
Correcting constraint violations: *move to the closest border point*

1. Assume constraint check done;
   only consider turbines violating site constraints.
2. Determine closest point on *all* violated constraints (the *candidates*).
3. Remove candidates that fall outside of the site.



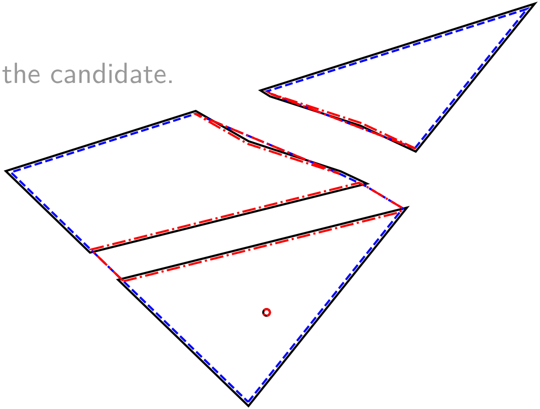
Correcting constraint violations: *move to the closest border point*

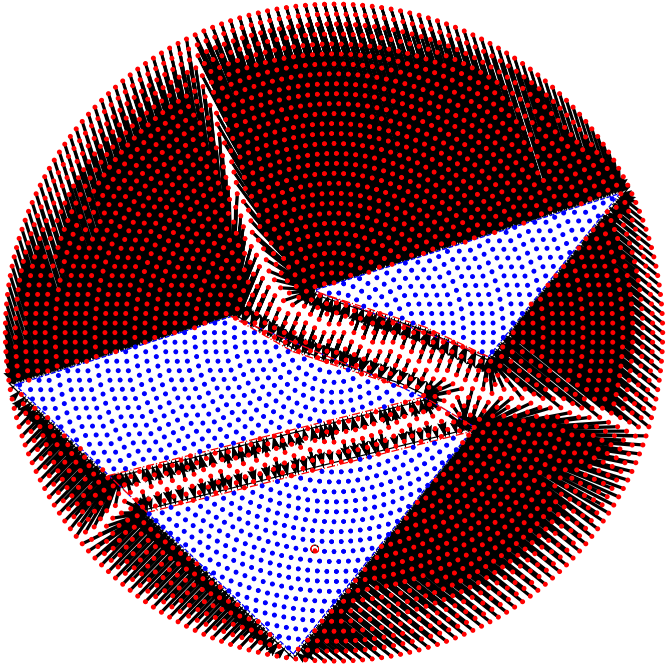
1. Assume constraint check done;
   only consider turbines violating site constraints.
2. Determine closest point on *all* violated constraints (the *candidates*).
3. Remove candidates that fall outside of the site.
4. For parcels without a candidate,
   take the closest parcel vertex as the candidate.

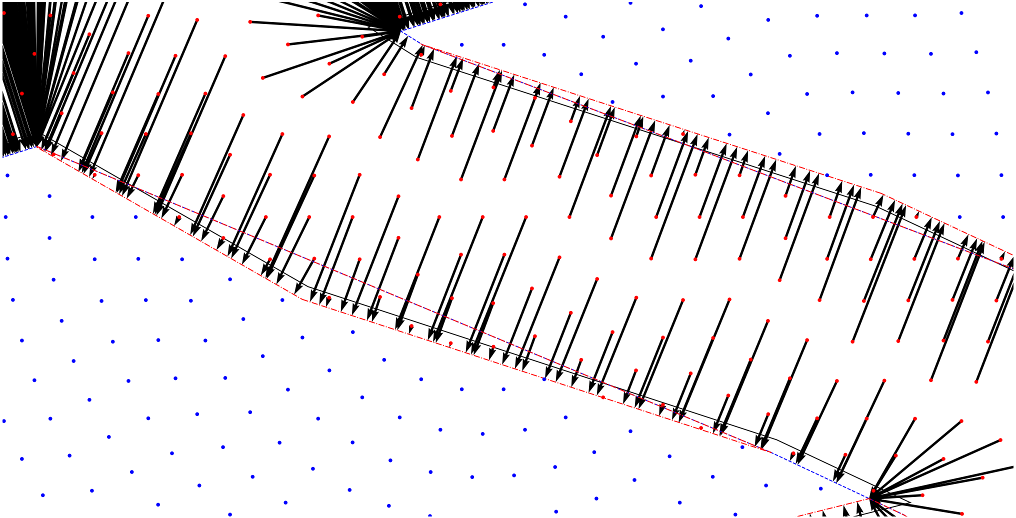


Correcting constraint violations: *move to the closest border point*

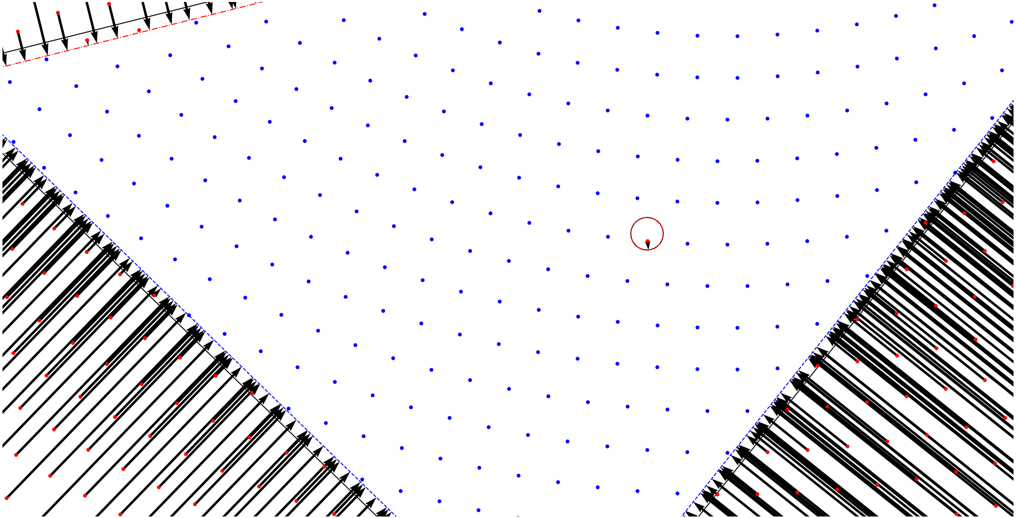
1. Assume constraint check done;
   only consider turbines violating site constraints.
2. Determine closest point on *all* violated constraints (the *candidates*).
3. Remove candidates that fall outside of the site.
4. For parcels without a candidate,
   take the closest parcel vertex as the candidate.
5. Take the closest candidate
   as the correction.







(My current implementation differs from the algorithm sketched.)



Conclusions

- Site constraint handling deserves more attention.

- Efficient approaches are possible.

To do

- Decent overview of constraint handling approaches. (*Your input is appreciated!*)
- Efficiency relative to other approaches.
- Actual characterization of computational complexity.
- Better implementation of correction algorithm.
- Restriction to ‘simple’ decompositions?

Thanks! Questions?