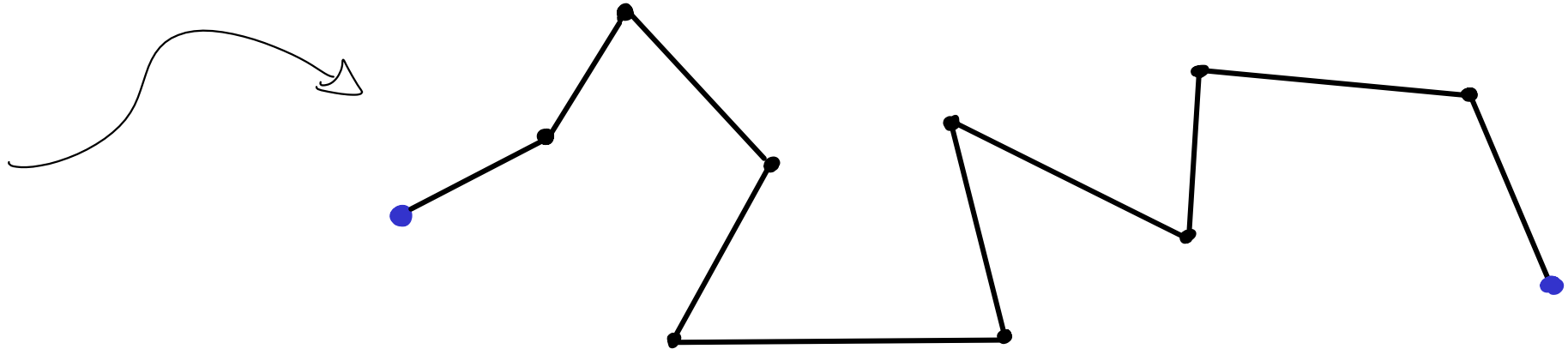
CHAIN SIMPLIFICATION

input:

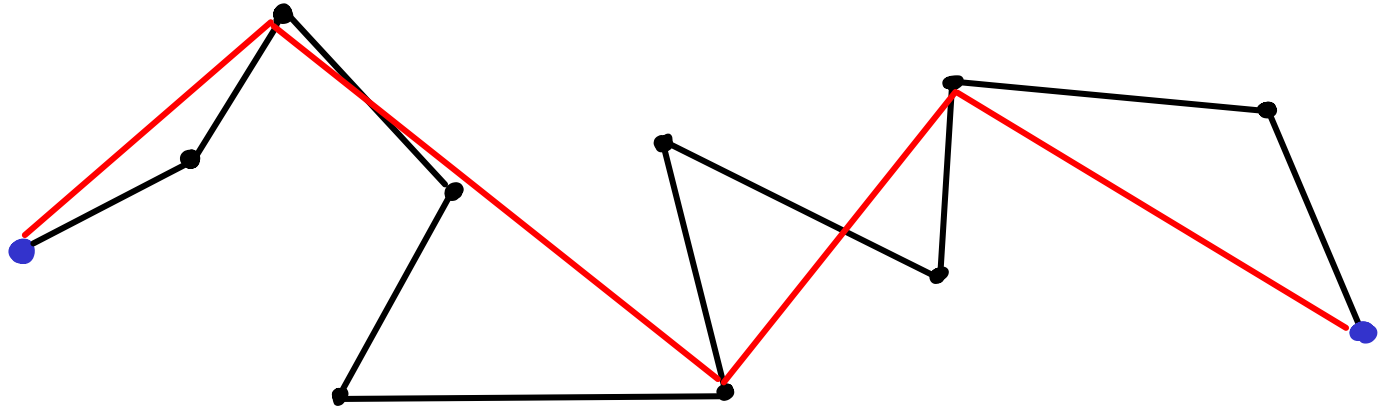
a chain



CHAIN SIMPLIFICATION

input:

a chain

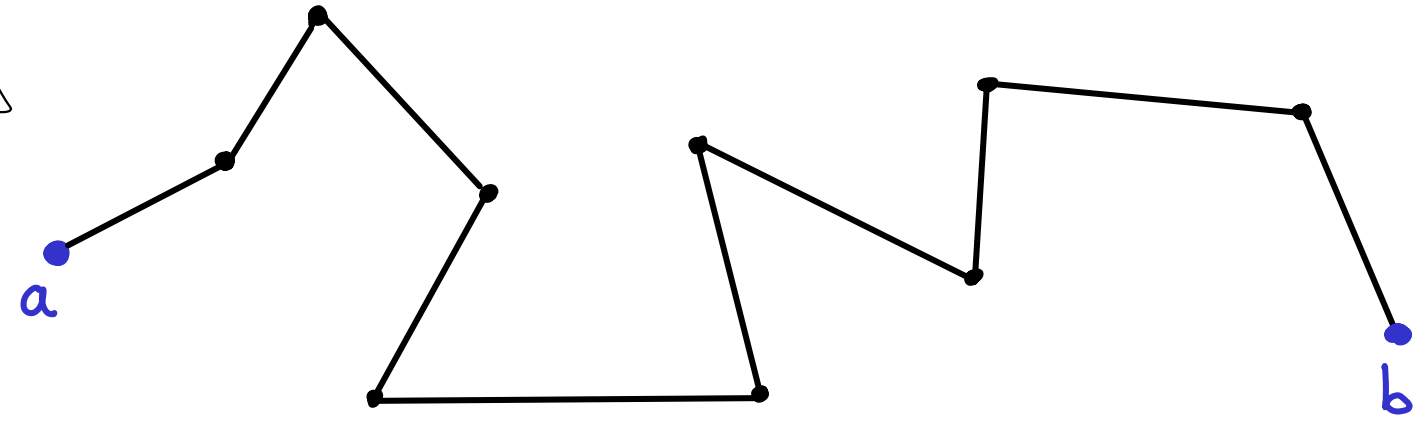


output : "something simpler"

CHAIN SIMPLIFICATION

input:

a chain



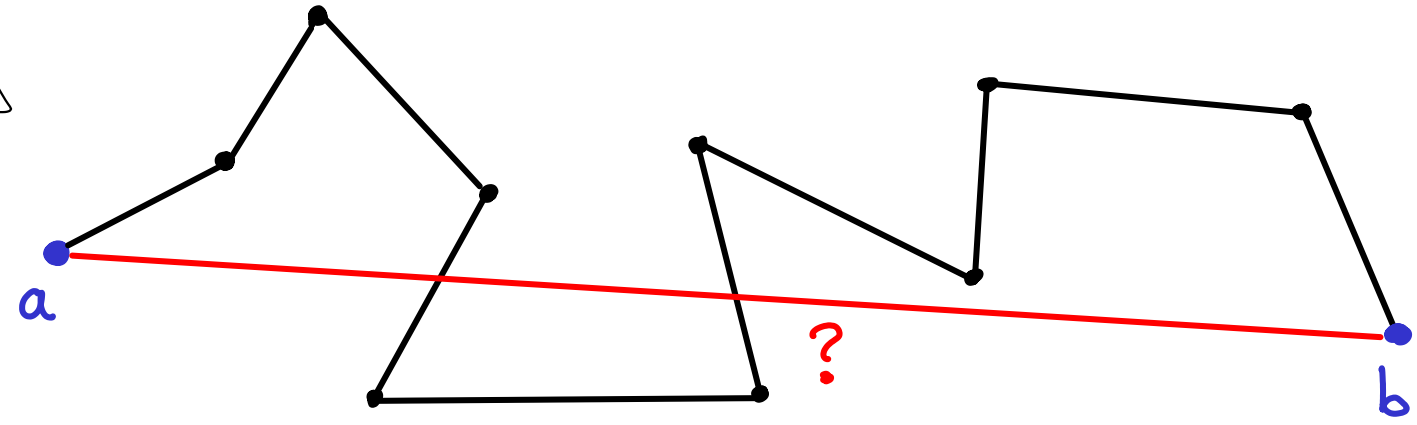
Algorithm:

```
simplify(a,b)   // chain from a to b
```

CHAIN SIMPLIFICATION : "ITERATIVE ENDPOINTS FIT"

input:

a chain



Algorithm:

simplify(a,b) // chain from a to b

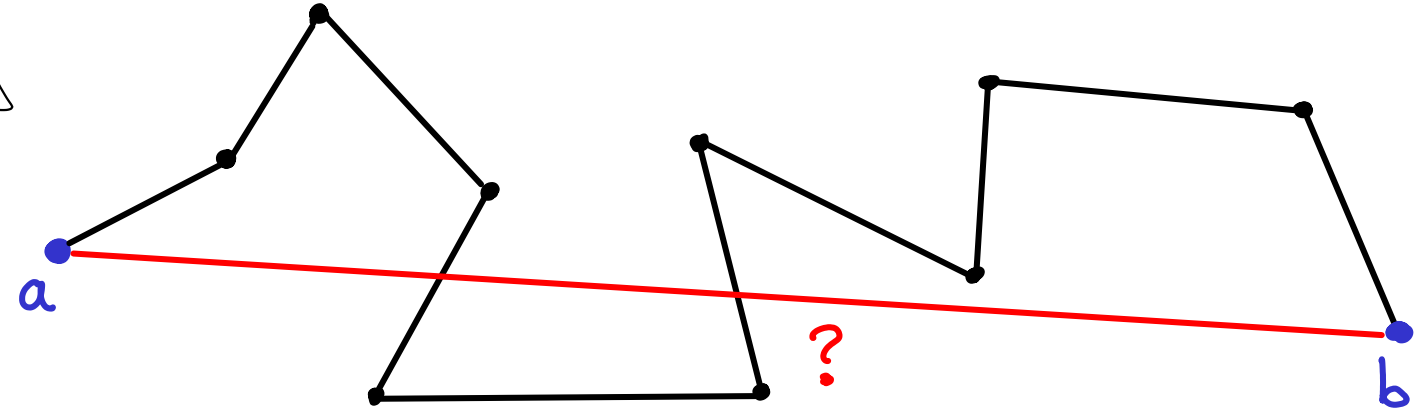
if $\overline{ab}$ doesn't approximate chain(a,b) "well" ?

CHAIN SIMPLIFICATION : "ITERATIVE ENDPOINTS FIT"

input:

① a chain

② an error tolerance



Algorithm:

simplify(a,b) // chain from a to b

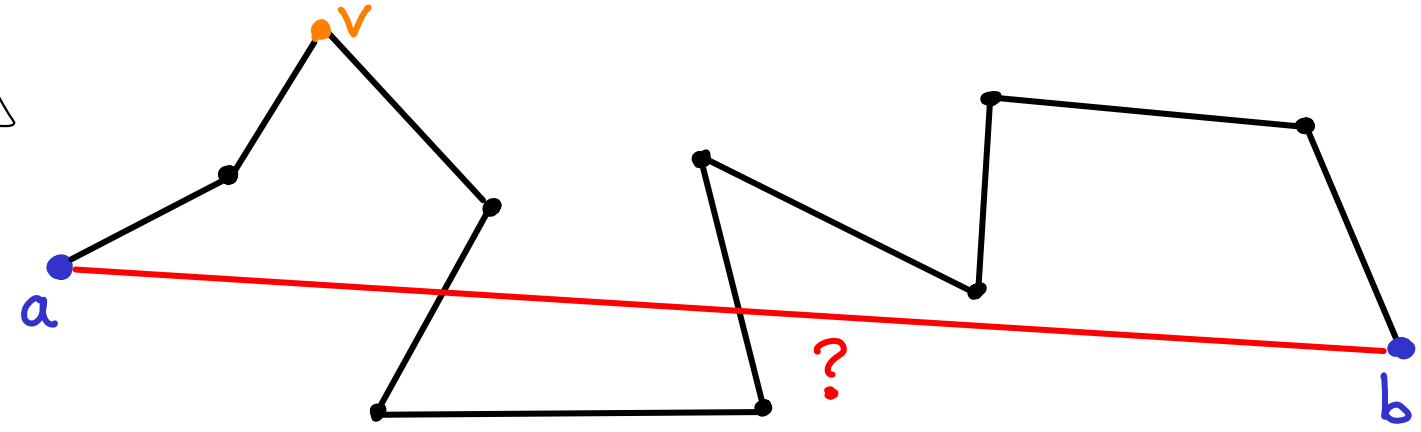
if $\overline{ab}$ doesn't approximate chain(a,b) "well"

CHAIN SIMPLIFICATION : "ITERATIVE ENDPOINTS FIT"

input:

① a chain

② an error tolerance



Algorithm:

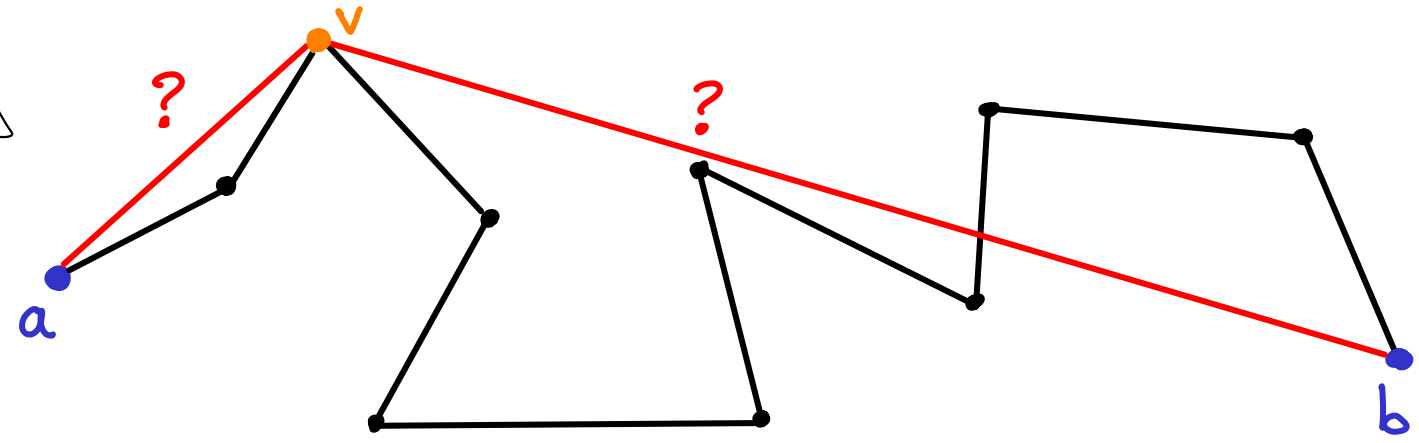
```
simplify(a,b)   // chain from a to b
  if ab doesn't approximate chain(a,b) "well"
    find v ∈ chain(a,b) w/ MAX dist to ab
```

CHAIN SIMPLIFICATION : "ITERATIVE ENDPOINTS FIT"

input:

① a chain

② an error tolerance



Algorithm:

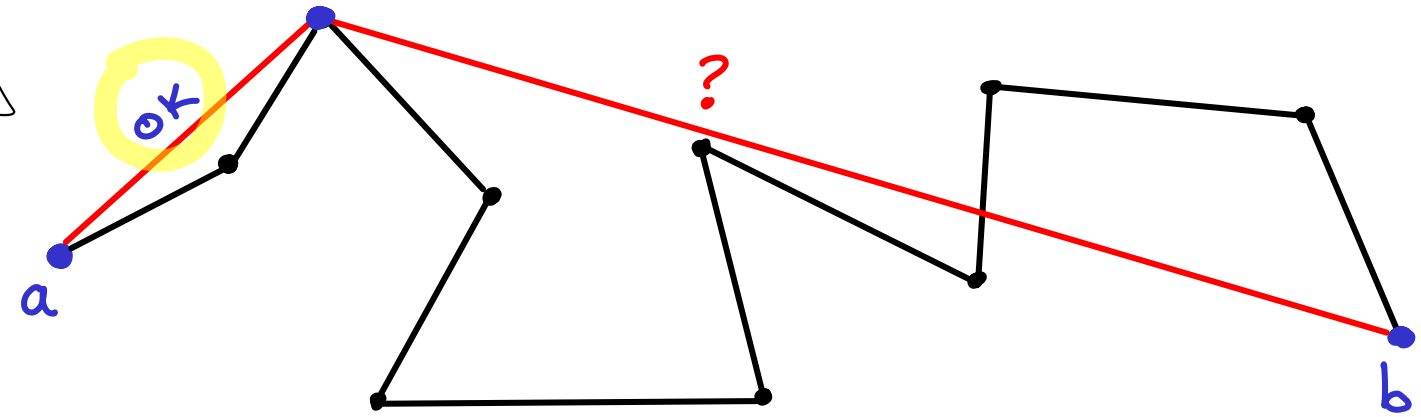
```
simplify(a,b)   // chain from a to b
    if ab doesn't approximate chain(a,b) "well"
        find v ∈ chain(a,b) w/ MAX dist to ab
        simplify(a,v)
        simplify(v,b)
```

CHAIN SIMPLIFICATION : "ITERATIVE ENDPOINTS FIT"

input:

① a chain

② an error tolerance



Algorithm:

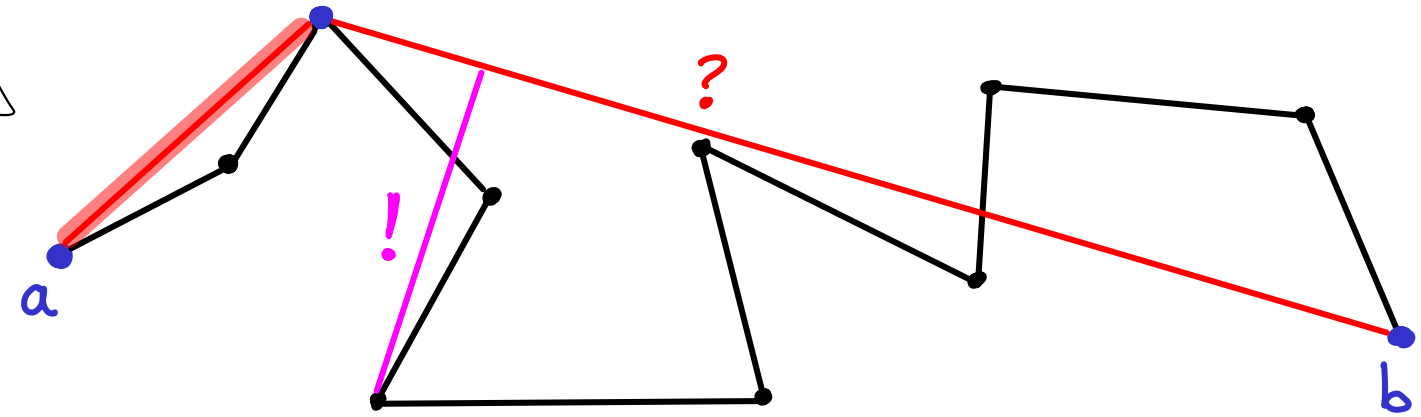
```
simplify(a,b)   // chain from a to b
    if ab doesn't approximate chain(a,b) "well"
        find v ∈ chain(a,b) w/ MAX dist to ab
        simplify(a,v)
        simplify(v,b)
    else output ab
```

CHAIN SIMPLIFICATION : "ITERATIVE ENDPOINTS FIT"

input:

① a chain

② an error tolerance



Algorithm:

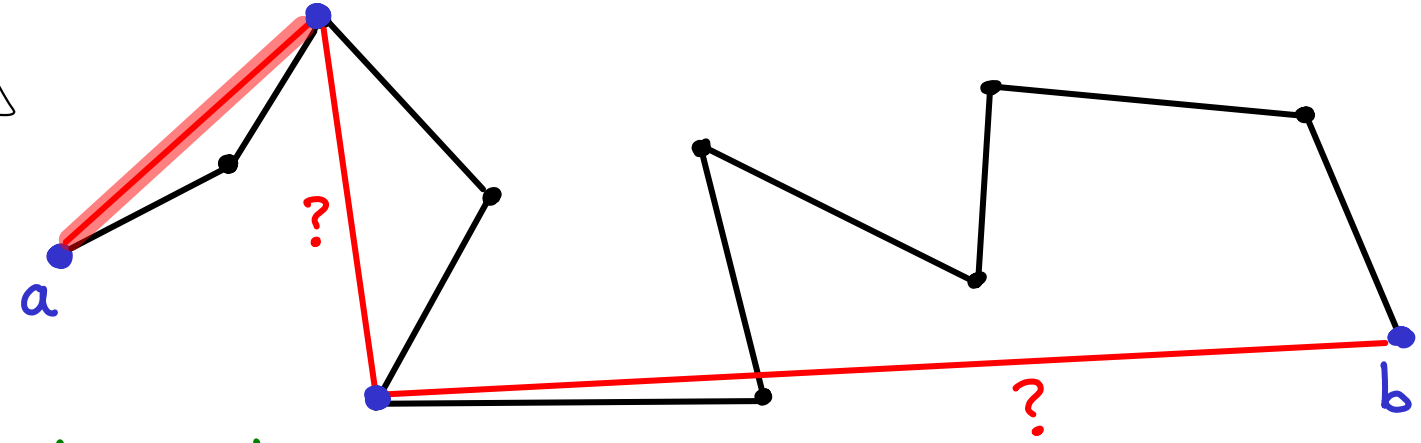
```
simplify(a,b)   // chain from a to b
  if ab doesn't approximate chain(a,b) "well"
    find v ∈ chain(a,b) w/ MAX dist to ab
    simplify(a,v)
    simplify(v,b)
  else output ab
```

CHAIN SIMPLIFICATION : "ITERATIVE ENDPOINTS FIT"

input:

① a chain

② an error tolerance



Algorithm:

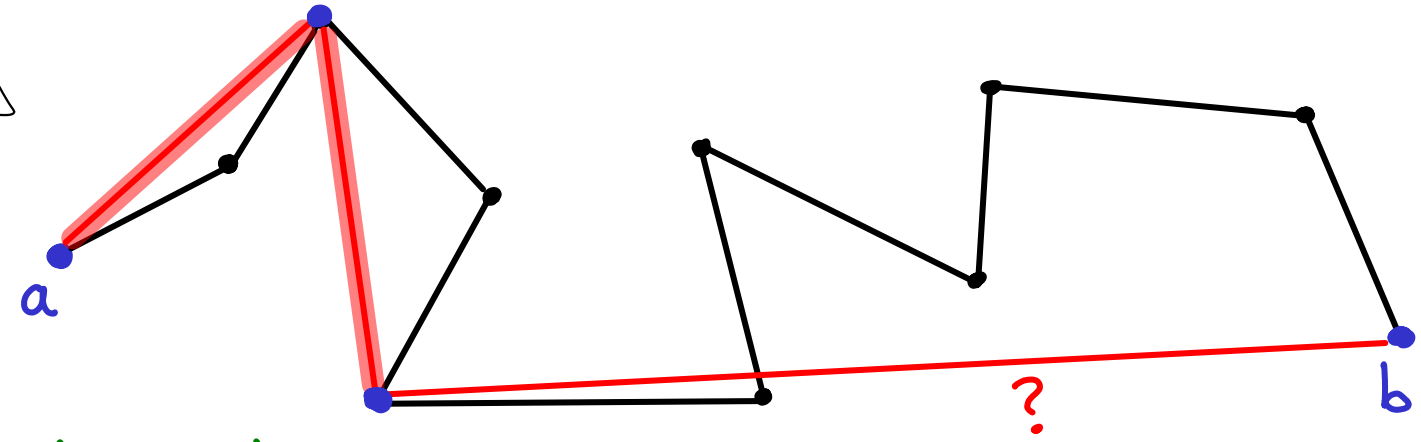
```
simplify(a,b)   // chain from a to b
  if ab doesn't approximate chain(a,b) "well"
    find v ∈ chain(a,b) w/ MAX dist to ab
    simplify(a,v)
    simplify(v,b)
  else output ab
```

CHAIN SIMPLIFICATION : "ITERATIVE ENDPOINTS FIT"

input:

① a chain

② an error tolerance



Algorithm:

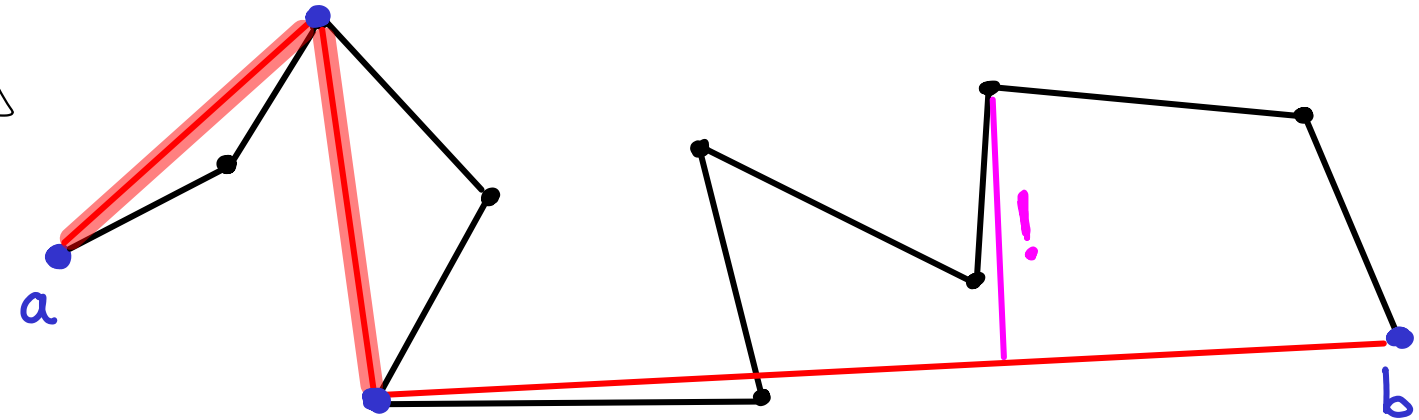
```
simplify(a,b)   // chain from a to b
  if ab doesn't approximate chain(a,b) "well"
    find v ∈ chain(a,b) w/ MAX dist to ab
    simplify(a,v)
    simplify(v,b)
  else output ab
```

CHAIN SIMPLIFICATION : "ITERATIVE ENDPOINTS FIT"

input:

① a chain

② an error tolerance



Algorithm:

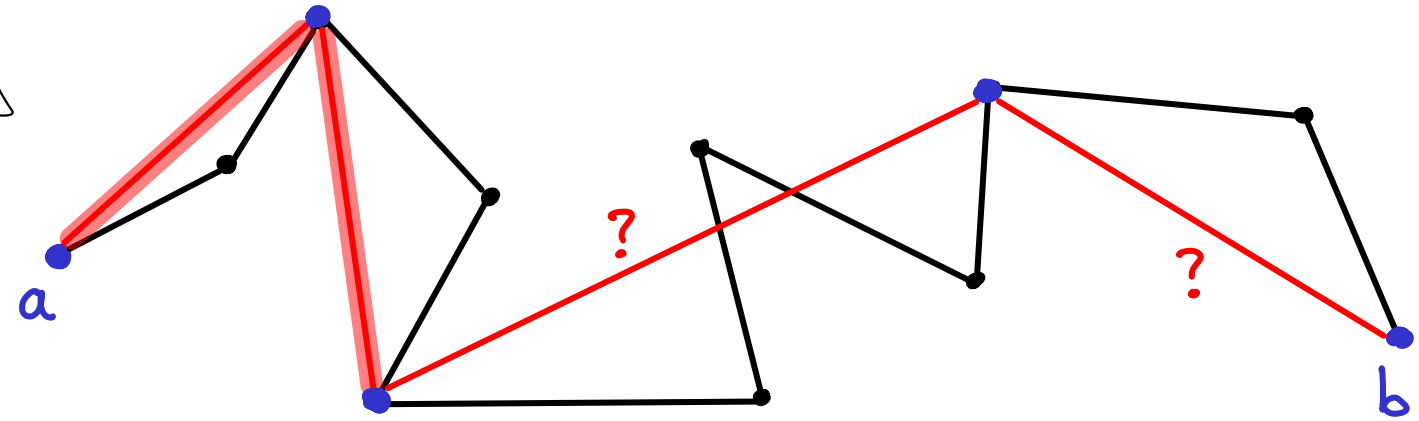
```
simplify(a,b)   // chain from a to b
  if ab doesn't approximate chain(a,b) "well"
    find v ∈ chain(a,b) w/ MAX dist to ab
    simplify(a,v)
    simplify(v,b)
  else output ab
```

CHAIN SIMPLIFICATION : "ITERATIVE ENDPOINTS FIT"

input:

① a chain

② an error tolerance



Algorithm:

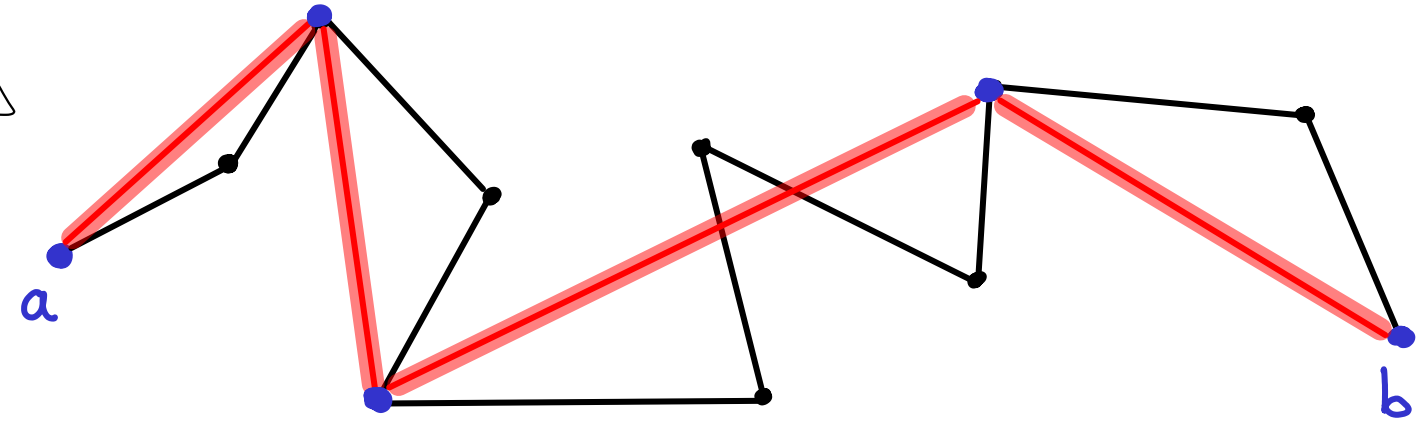
```
simplify(a,b)   // chain from a to b
    if ab doesn't approximate chain(a,b) "well"
        find v ∈ chain(a,b) w/ MAX dist to ab
        simplify(a,v)
        simplify(v,b)
    else output ab
```

CHAIN SIMPLIFICATION : "ITERATIVE ENDPOINTS FIT"

input:

① a chain

② an error tolerance



one detail missing

Algorithm:

```
simplify(a,b)   // chain from a to b
  if ab doesn't approximate chain(a,b) "well"
    | find v ∈ chain(a,b) w/ MAX dist to ab
    | simplify(a,v)
    | simplify(v,b)
  else output ab
```

CHAIN SIMPLIFICATION : "ITERATIVE ENDPOINTS FIT"

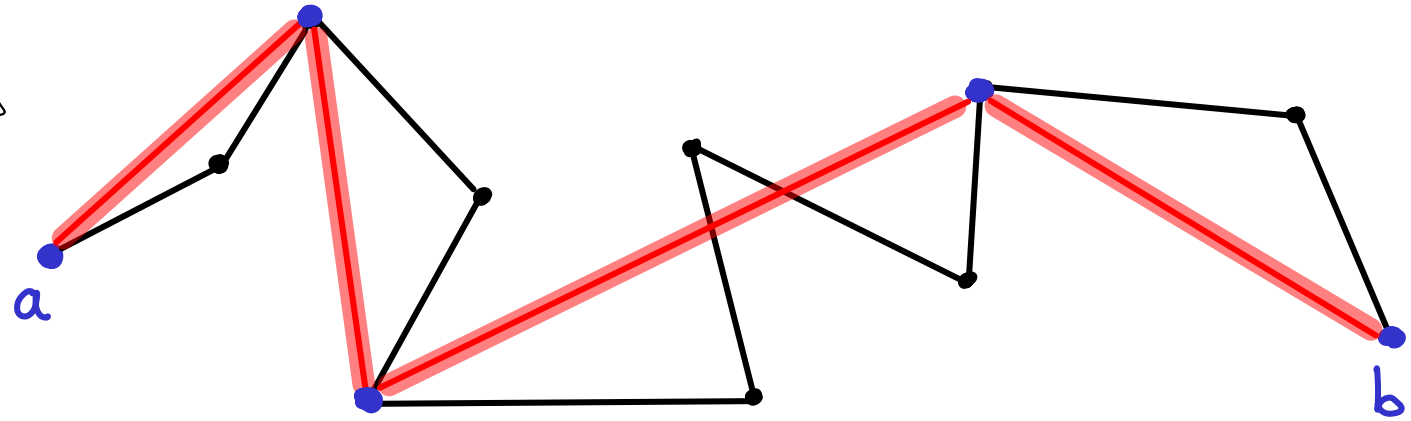
input:

① a chain

② an error tolerance

③ a distance definition

(between points & segments)



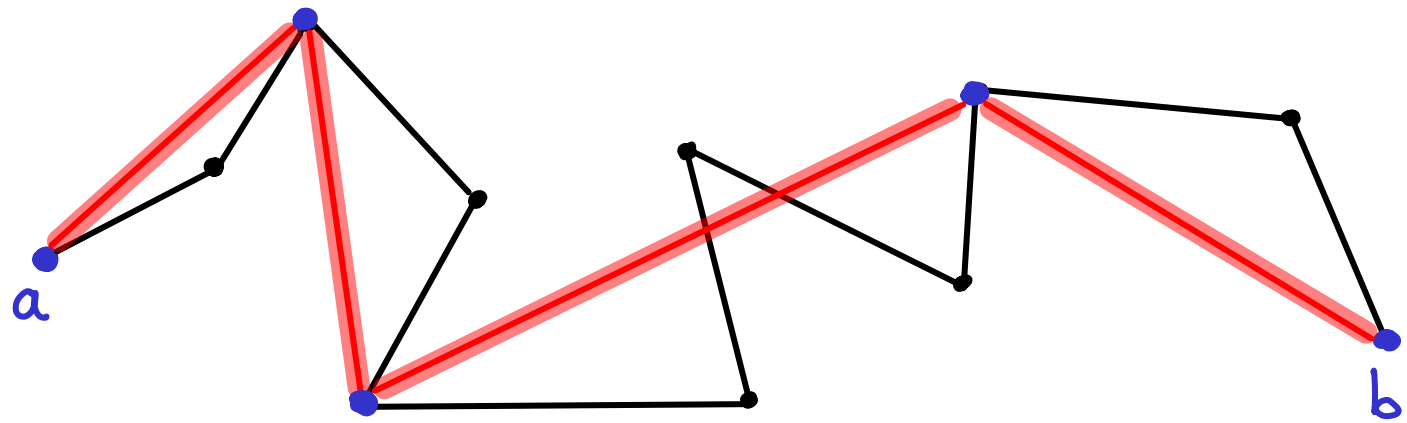
Algorithm:

```
simplify(a,b)   // chain from a to b
  if ab doesn't approximate chain(a,b) "well"
    find v ∈ chain(a,b) w/ MAX dist to ab
    simplify(a,v)
    simplify(v,b)
  else output ab
```

CHAIN SIMPLIFICATION : "ITERATIVE ENDPOINTS FIT"

Time (chain length: n)

?



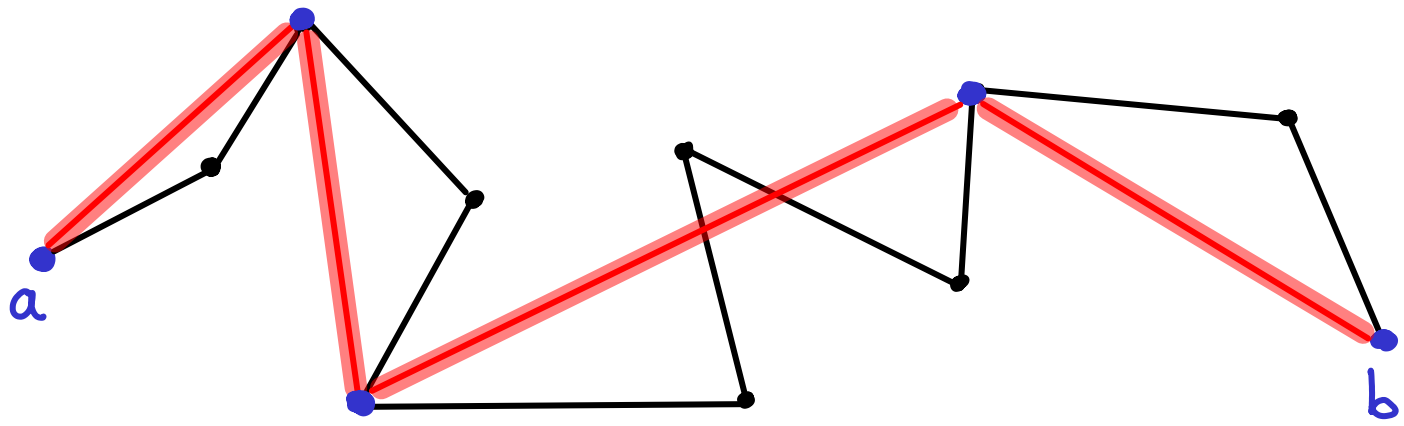
Algorithm:

```
simplify(a,b)   // chain from a to b
    if ab doesn't approximate chain(a,b) "well"
        find v ∈ chain(a,b) w/ MAX dist to ab
        simplify(a,v)
        simplify(v,b)
    else output ab
```

CHAIN SIMPLIFICATION : "ITERATIVE ENDPOINTS FIT"

Time (chain length: n)

- testing $\overline{ab}$: O(n)
  (for reasonable dist)



Algorithm:

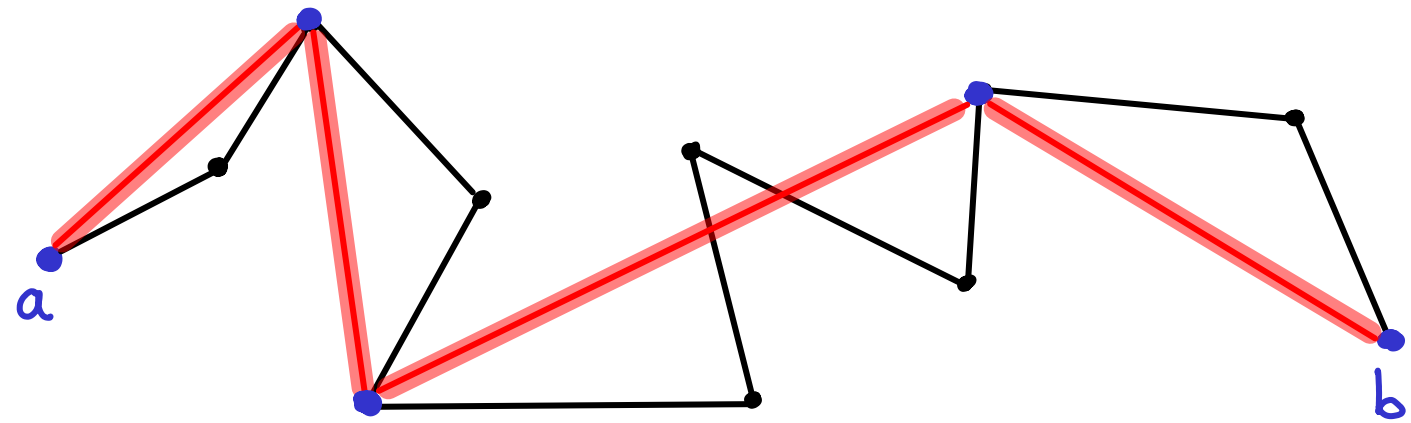
```
simplify(a,b)   // chain from a to b
  if ab doesn't approximate chain(a,b) "well"
    find v ∈ chain(a,b) w/ MAX dist to ab
    simplify(a,v)
    simplify(v,b)
  else output ab
```

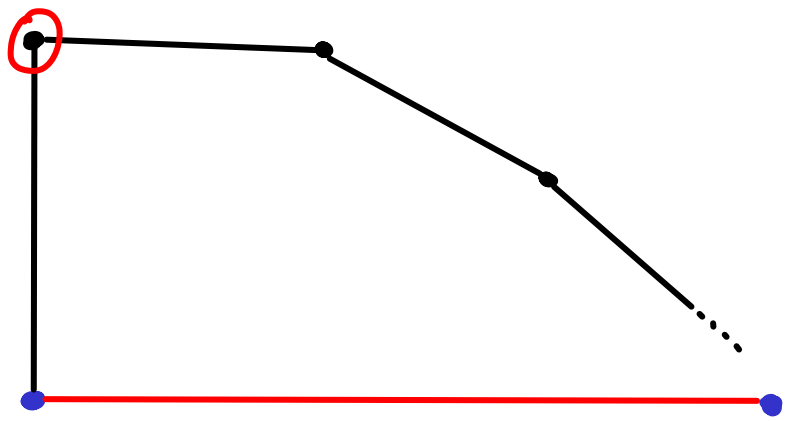
CHAIN SIMPLIFICATION : "ITERATIVE ENDPOINTS FIT"

Time (chain length: n)

- testing $\overline{ab}$: O(n)
  (for reasonable dist)

- worst case:
  unbalanced recursion





Algorithm:

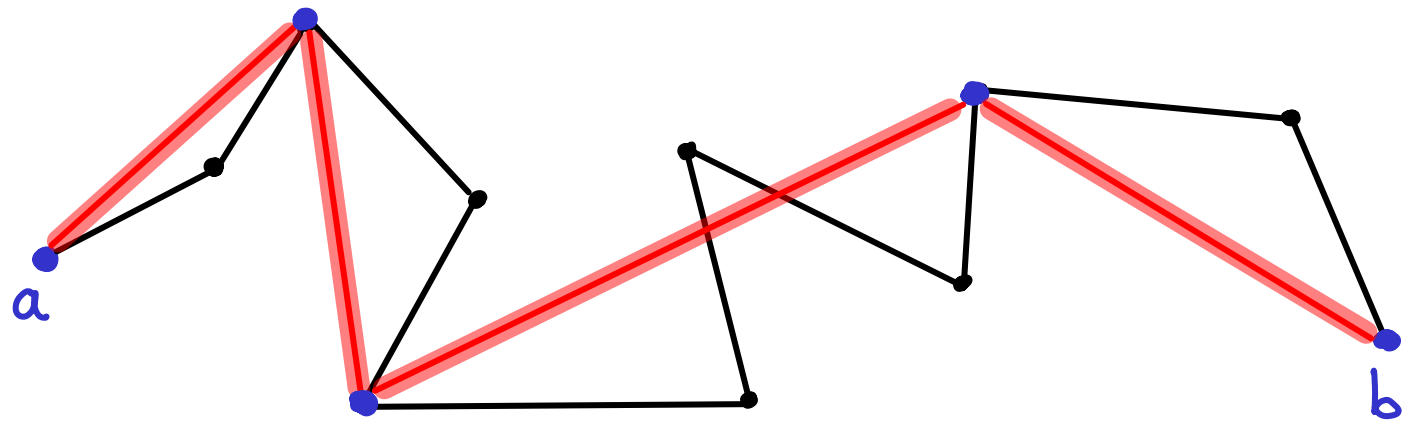
```
simplify(a,b)   // chain from a to b
  if ab doesn't approximate chain(a,b) "well"
    | find v ∈ chain(a,b) w/ MAX dist to ab
    | simplify(a,v)
    | simplify(v,b)
  else output ab
```

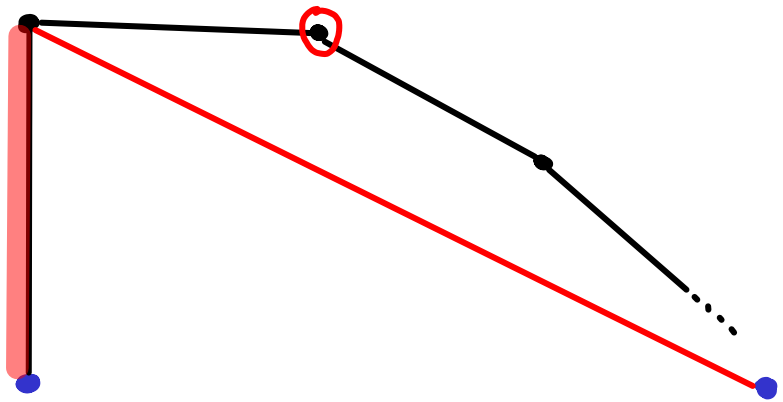
CHAIN SIMPLIFICATION : "ITERATIVE ENDPOINTS FIT"

Time (chain length: n)

- testing $\overline{ab}$: O(n)
  (for reasonable dist)

- worst case:
  unbalanced recursion





Algorithm:

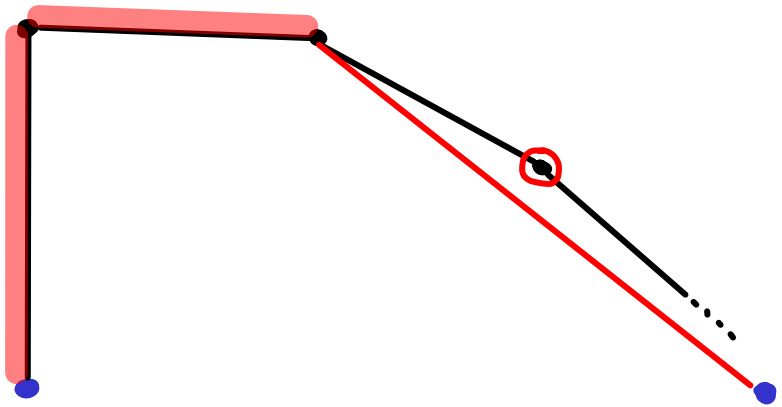
```
simplify(a,b)   // chain from a to b
  if ab doesn't approximate chain(a,b) "well"
    | find v ∈ chain(a,b) w/ MAX dist to ab
    | simplify(a,v)
    | simplify(v,b)
  else output ab
```

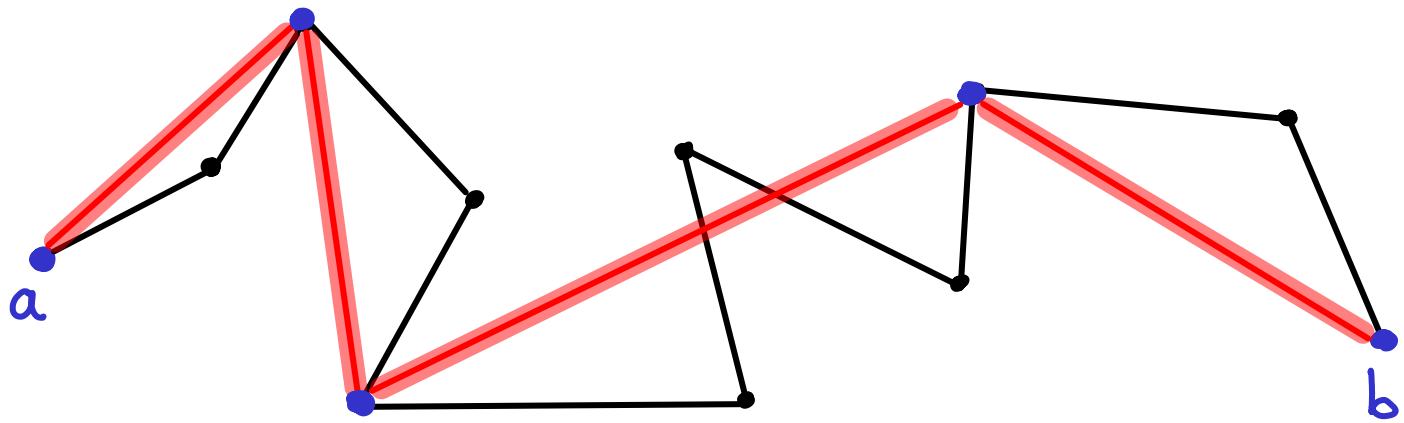
CHAIN SIMPLIFICATION : "ITERATIVE ENDPOINTS FIT"

Time (chain length: n)

- testing $\overline{ab}$: O(n)
  (for reasonable dist)

- worst case:
  unbalanced recursion
  ... $O(n^2)$



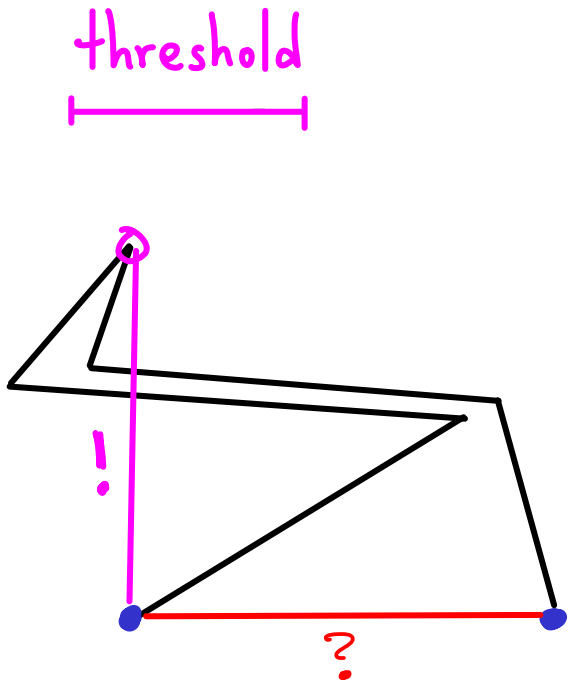


Algorithm:

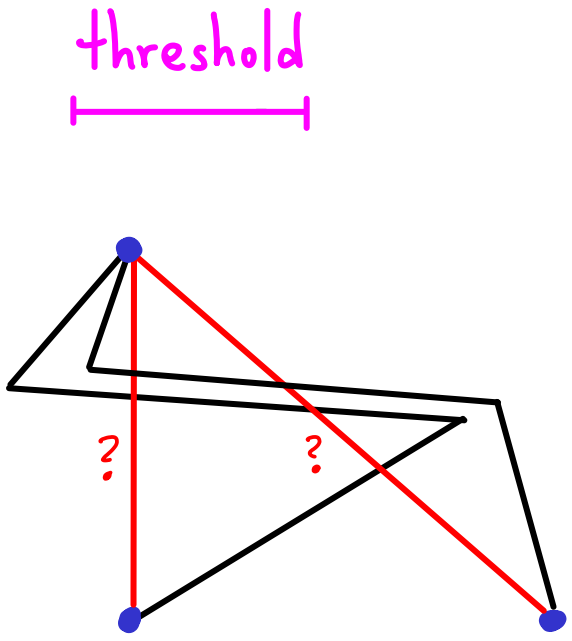
```
simplify(a,b)   // chain from a to b
  if ab doesn't approximate chain(a,b) "well"
    | find v ∈ chain(a,b) w/ MAX dist to ab
    | simplify(a,v)
    | simplify(v,b)
  else output ab
```

do we always get a simple (non-crossing) chain?

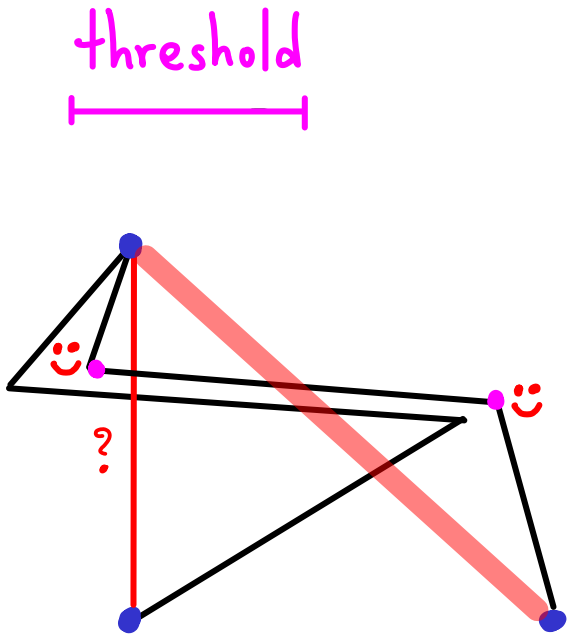
do we always get a simple (non-crossing) chain?



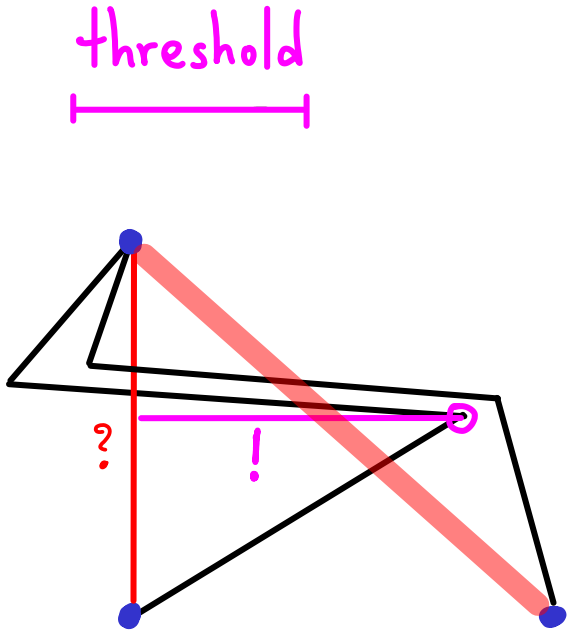
do we always get a simple (non-crossing) chain?



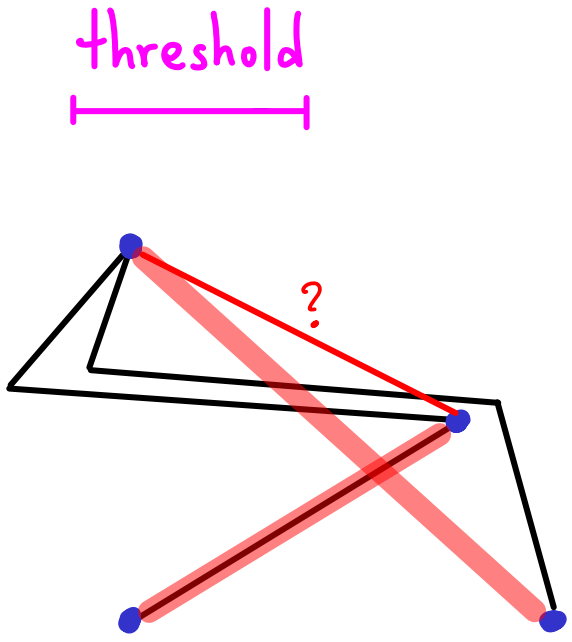
do we always get a simple (non-crossing) chain?



do we always get a simple (non-crossing) chain?

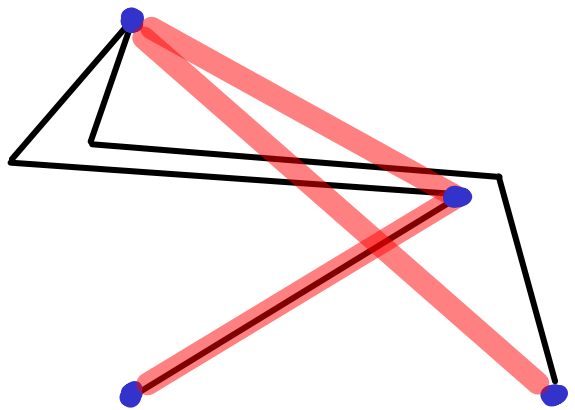


do we always get a simple (non-crossing) chain?



do we always get a simple (non-crossing) chain? NO

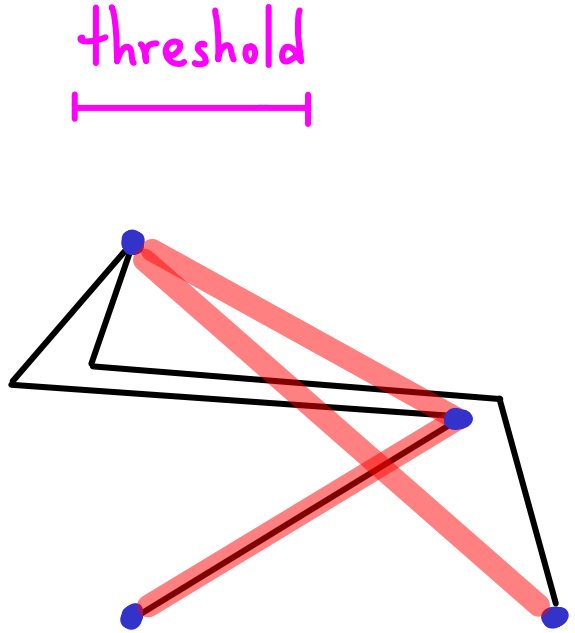
threshold



variants (possibly unexplored)

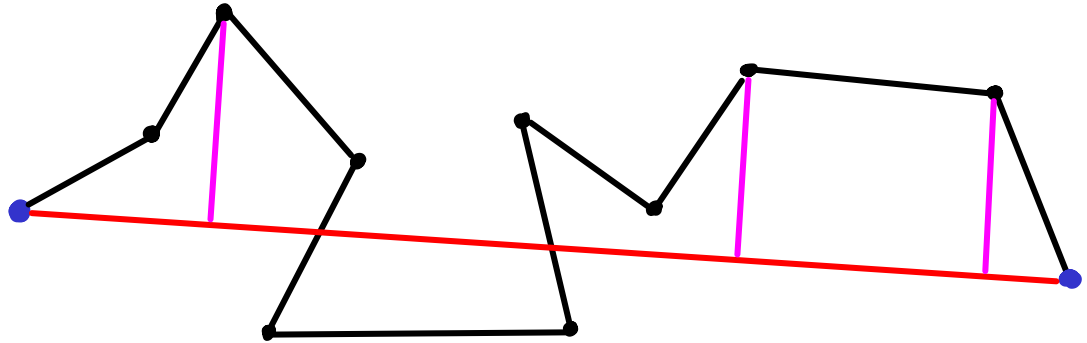
- require non-crossing output

do we always get a simple (non-crossing) chain? NO



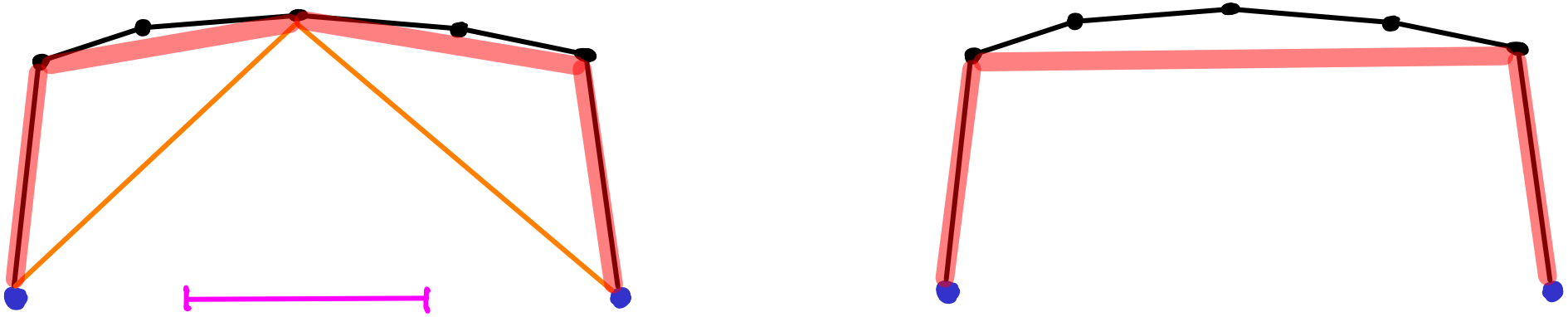
variants (possibly unexplored)

- require non-crossing output
- deal with multiple error violations
  - ↳ use median "offender"?
  - ↳ ≥2 recursive calls?



Does the algorithm minimize the number of segments used ?

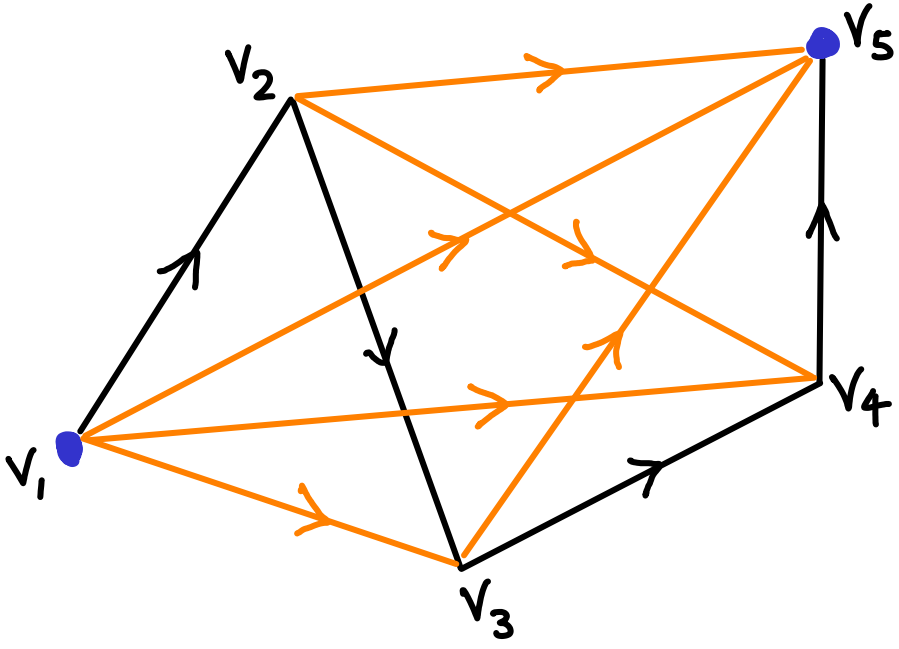
Does the algorithm minimize the number of segments used?



No. can be made arbitrarily worse

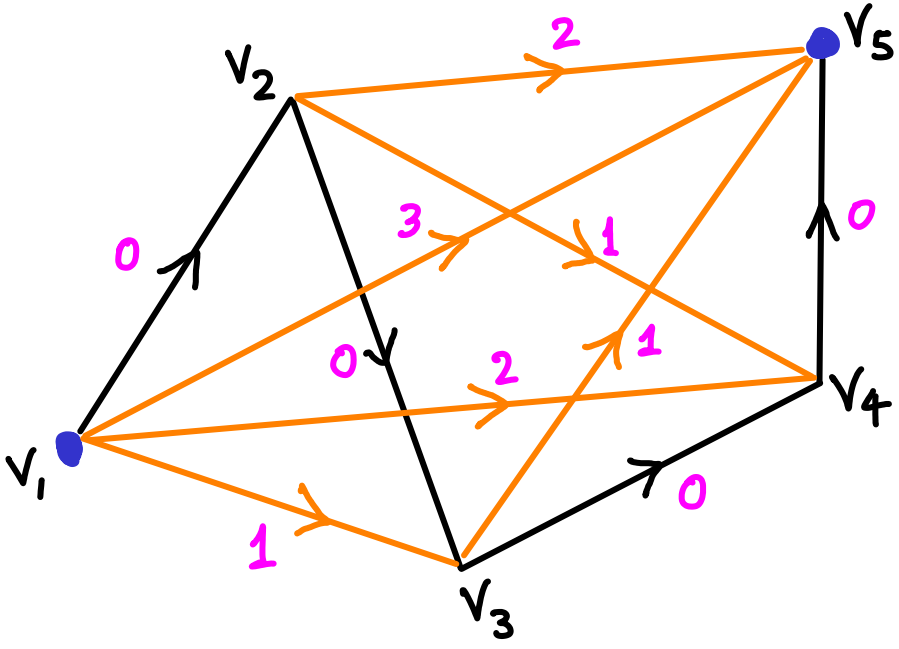
IRI-IMAI algorithm to minimize segments (given path & tolerance)

IRI-IMAI algorithm to minimize segments (given path & tolerance)



- form directed graph:
  - ↳ make $\overrightarrow{v_i v_j}$ iff $i < j$

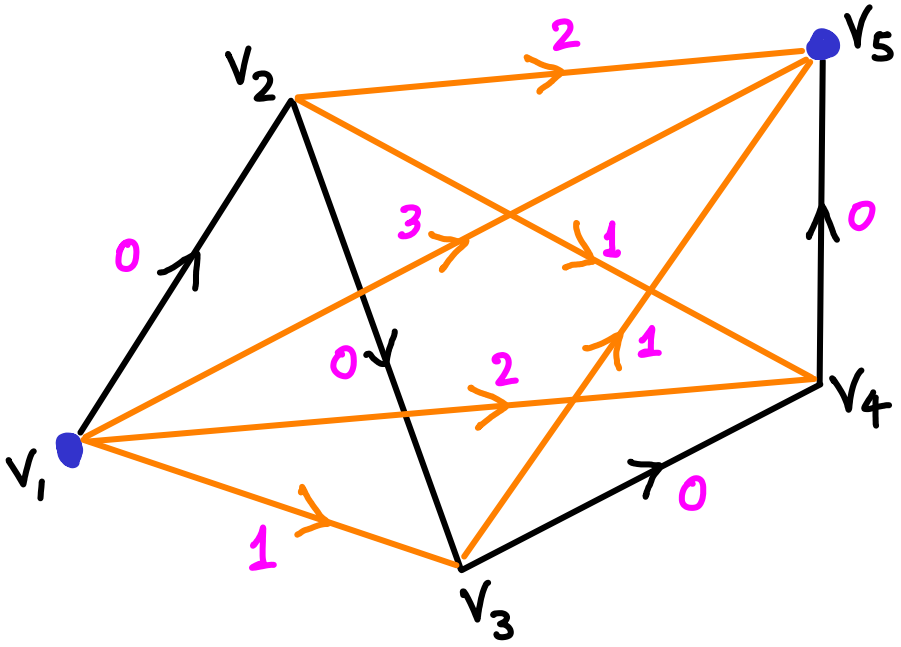
IRI-IMAI algorithm to minimize segments (given path & tolerance)



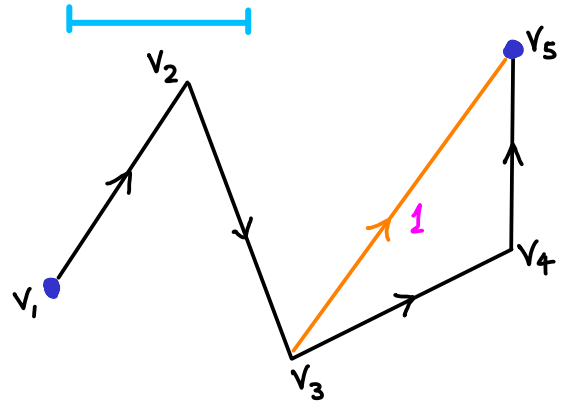
- form directed graph:
  - ↳ make $\overrightarrow{v_i v_j}$ iff $i<j$

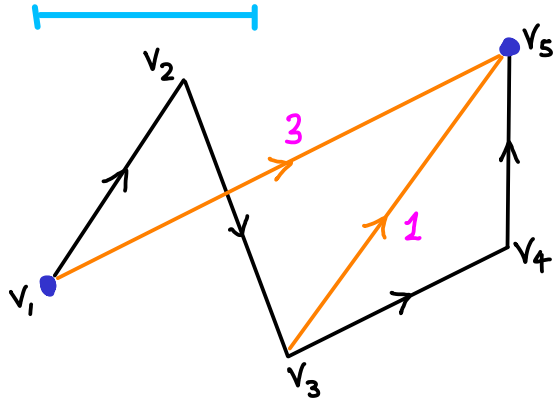
- give a score to each edge:
  - ↳ # path edges skipped
    - ↓
    - i.e. avoided

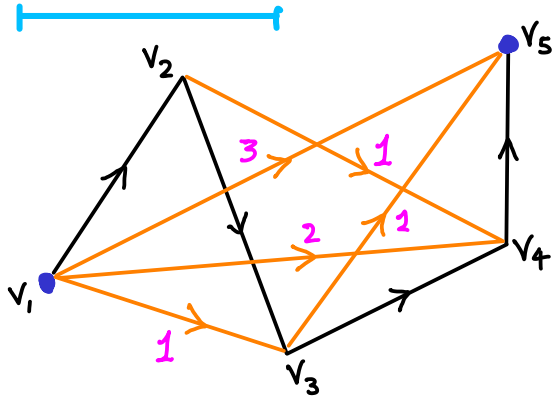
IRI-IMAI algorithm to minimize segments (given path & tolerance)



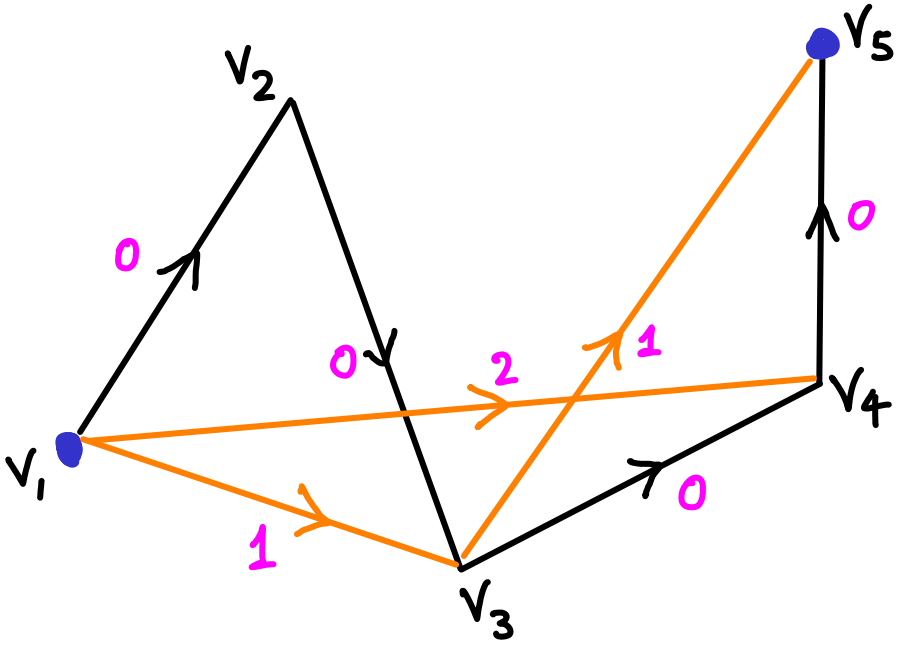
- form directed graph:
  - ↳ make $\overrightarrow{v_i v_j}$ iff $i<j$
- give a score to each edge:
  - ↳ # path edges skipped
- keep only edges satisfying tolerance







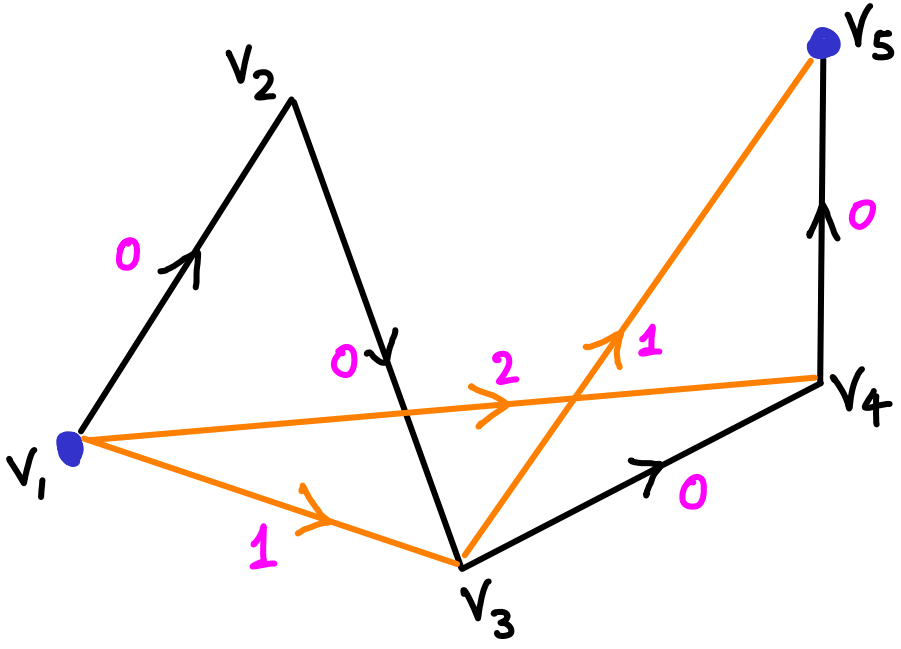
IRI-IMAI algorithm to minimize segments (given path & tolerance)



- form directed graph:
  - ↳ make $\overrightarrow{v_i v_j}$ iff $i < j$
- give a score to each edge:
  - ↳ # path edges skipped
- keep only edges satisfying tolerance

time so far?

IRI-IMAI algorithm to minimize segments (given path & tolerance)



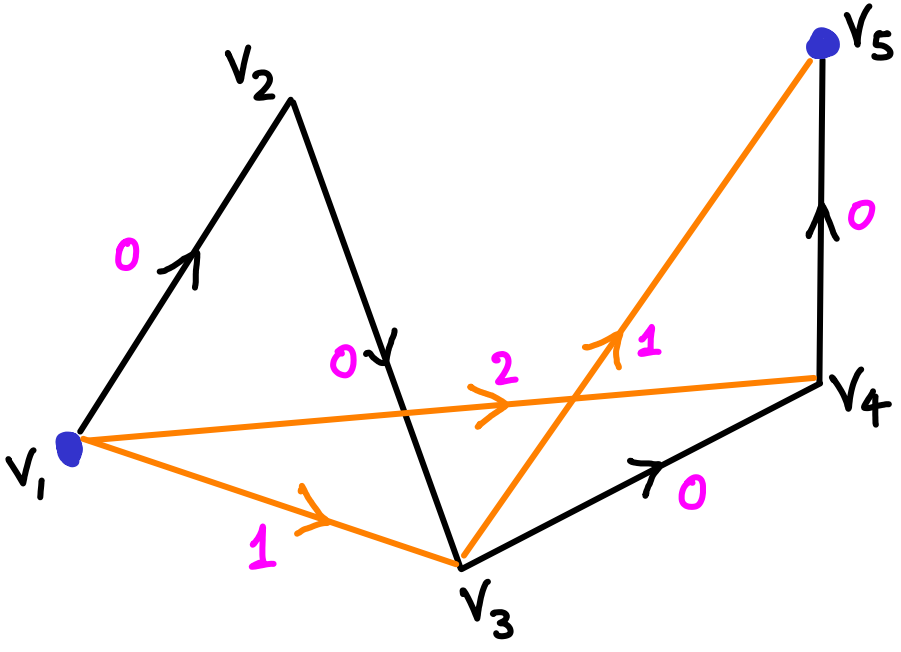
1 - form directed graph:
↳ make $\overrightarrow{v_i v_j}$ iff $i<j$

2 - give a score to each edge:
↳ # path edges skipped

3 - keep only edges satisfying tolerance

$\Theta(v^2)$ edges $\begin{cases} 1 & \Theta(v^2) \\ 2 & >> \\ 3 & \Theta(v^2)\cdot O(v) \end{cases}$

IRI-IMAI algorithm to minimize segments (given path & tolerance)



1 - form directed graph:
↳ make $\overrightarrow{v_i v_j}$ iff $i<j$

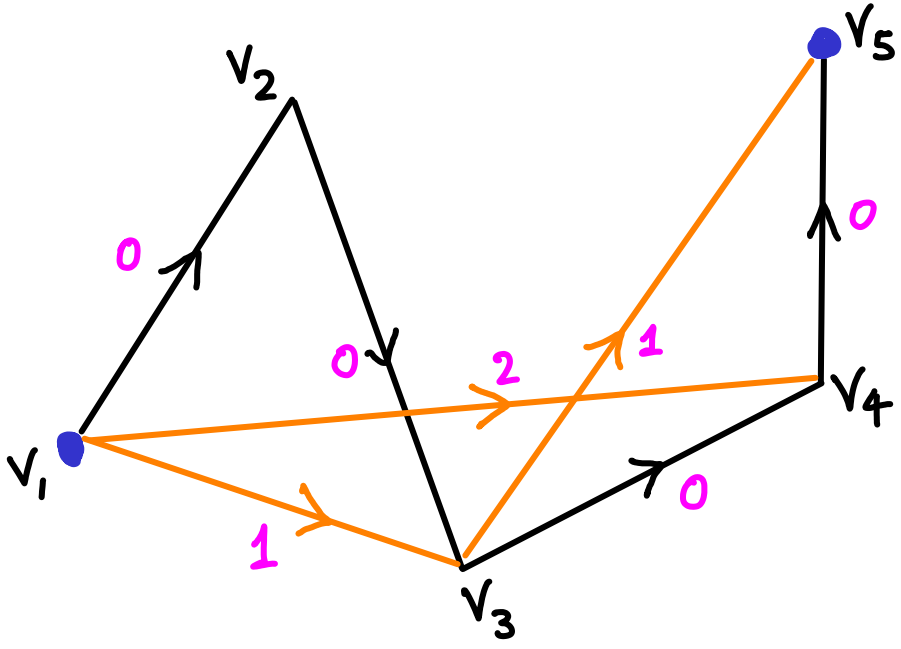
2 - give a score to each edge:
↳ # path edges skipped

3 - keep only edges satisfying tolerance

$\Theta(v^2)$ edges $\begin{cases} 1 \\ 2 \\ 3 \end{cases}$ $\Theta(v^2)$ >> $\Theta(v^2)\cdot O(v)$

Next?

IRI-IMAI algorithm to minimize segments (given path & tolerance)



1 - form directed graph:
↳ make $\overrightarrow{v_i v_j}$ iff $i<j$

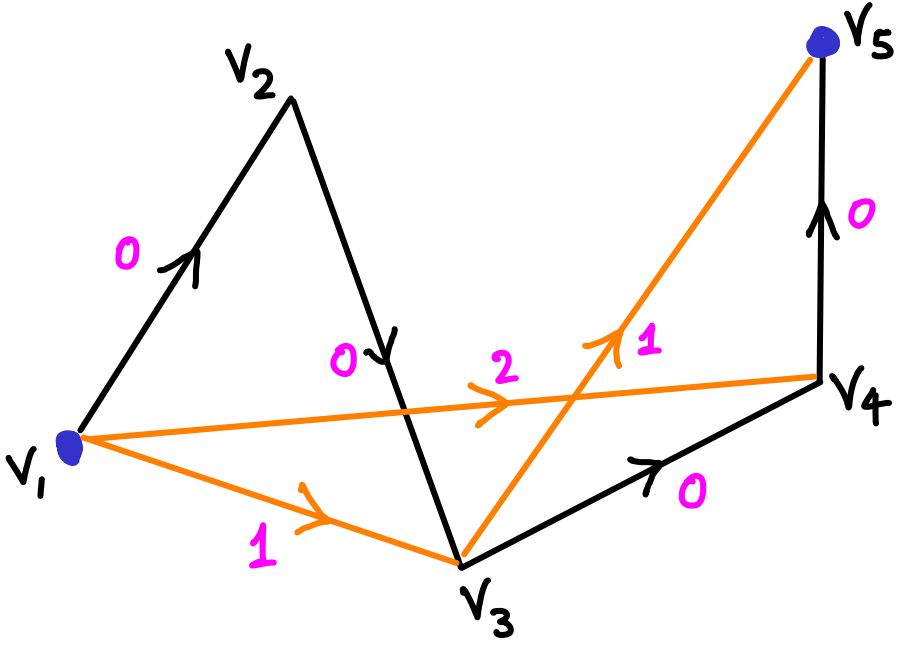
2 - give a score to each edge:
↳ # path edges skipped

3 - keep only edges satisfying tolerance

- max-weight path in a DAG
  ↳ min-weight w/ scores inverted

$\Theta(V^2)$ edges $\begin{cases} 1 \\ 2 \\ 3 \end{cases}$ $\Theta(V^2) \gg \Theta(V^2) \cdot O(V)$

IRI-IMAI algorithm to minimize segments (given path & tolerance)



1 - form directed graph:
↳ make $\overrightarrow{v_i v_j}$ iff $i<j$

2 - give a score to each edge:
↳ # path edges skipped

3 - keep only edges satisfying tolerance

- max-weight path in a DAG
  ↳ min-weight w/ scores inverted

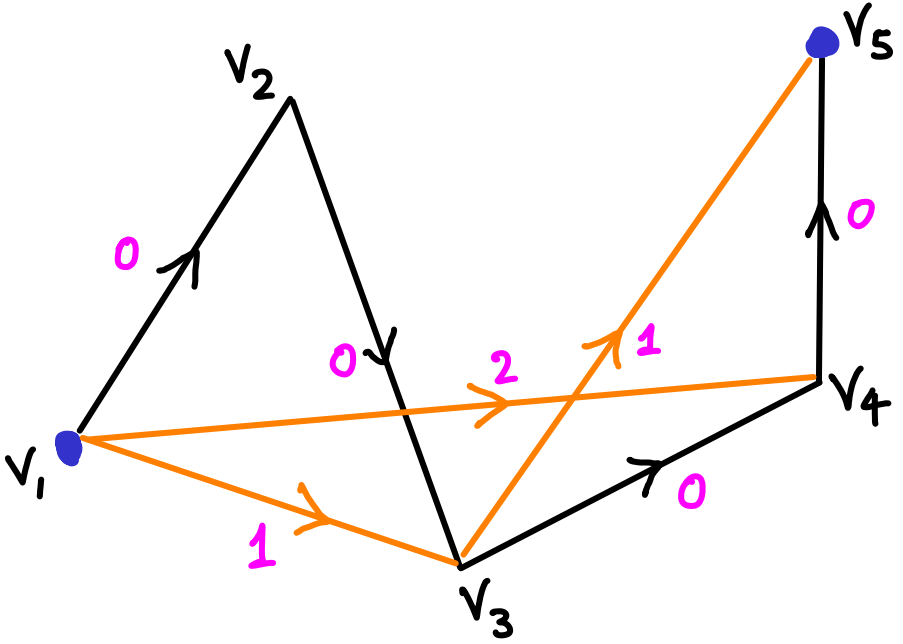
OR

- simple (unweighted) BFS

$O(V^2)$

$\Theta(V^2)$ edges $\begin{cases} 1 \\ 2 \\ 3 \end{cases}$ $\Theta(V^2)$ >> $\Theta(V^2) \cdot O(V)$

IRI-IMAI algorithm to minimize segments (given path & tolerance)



1 - form directed graph:
↳ make $\overrightarrow{v_i v_j}$ iff $i<j$

2 - give a score to each edge:
↳ # path edges skipped

3 - keep only edges satisfying tolerance

- max-weight path in a DAG
  ↳ min-weight w/ scores inverted

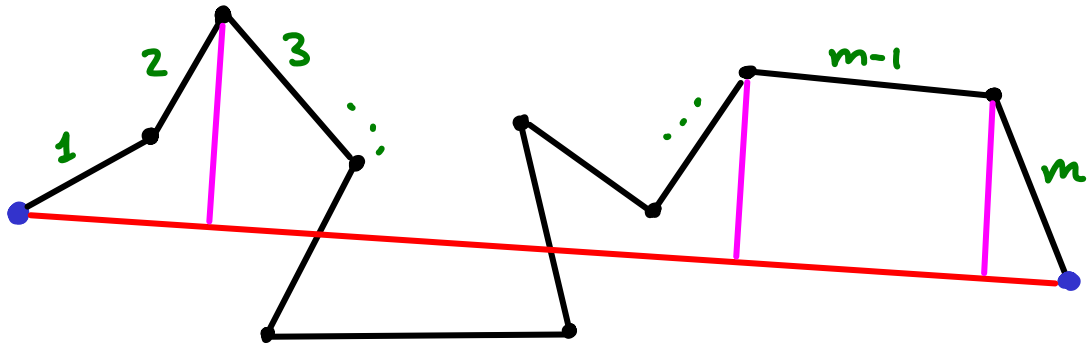
OR

- simple (unweighted) BFS

} $O(V^2)$

$\Theta(V^2)$ edges $\Theta(V^2)$ 1 2 3 >> $\Theta(V^2) \cdot O(V)$

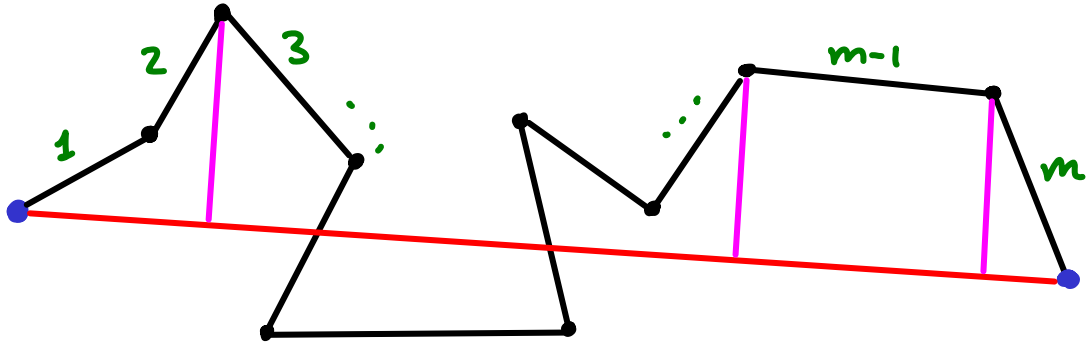
IRI-IMAI algorithm to minimize segments (given path & tolerance)



Testing an edge that connects endpoints of a path w/ m segments

↳ brute force : $\Theta(m)$ time

IRI-IMAI algorithm to minimize segments (given path & tolerance)



Testing an edge that connects endpoints of a path w/ m segments

↳ brute force : $\Theta(m)$ time

↳ under certain conditions (e.g. parallel strip distance)
& w/ preprocessing, we can do better

data structure?

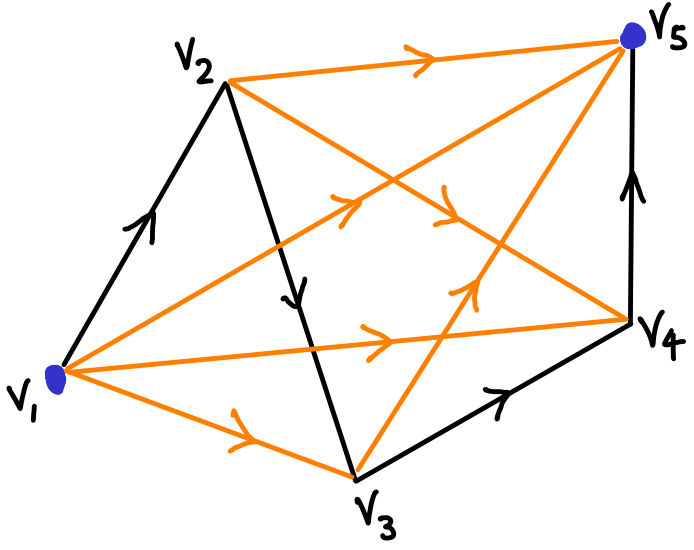
(and thus get $o(V^3)$ overall)

PROJECT

Suppose we are willing to use up to k edges.

↳ goal: minimize error

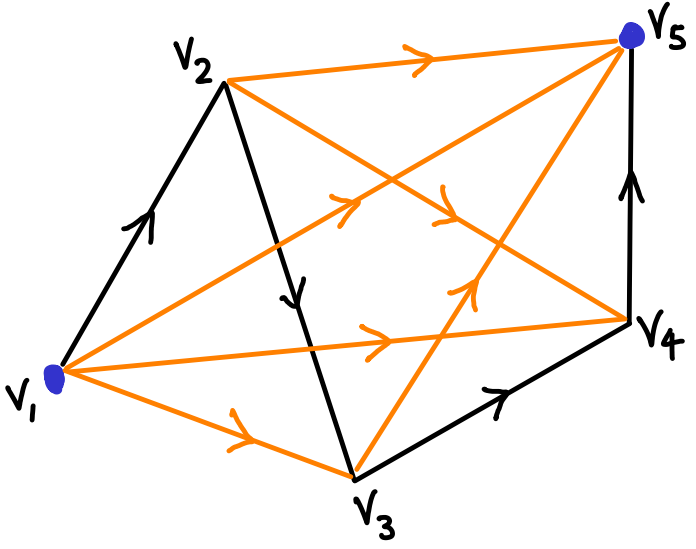
↘ & get a chain for output



Suppose we are willing to use up to k edges.

↳ goal: minimize error

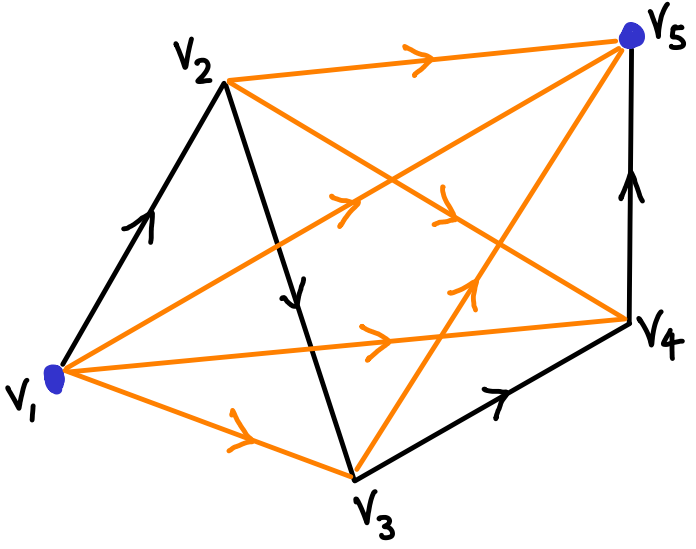
- build the full DAG $\Theta(V^2)$
- compute error of each edge $O(V^2 \cdot V)$



Suppose we are willing to use up to k edges.

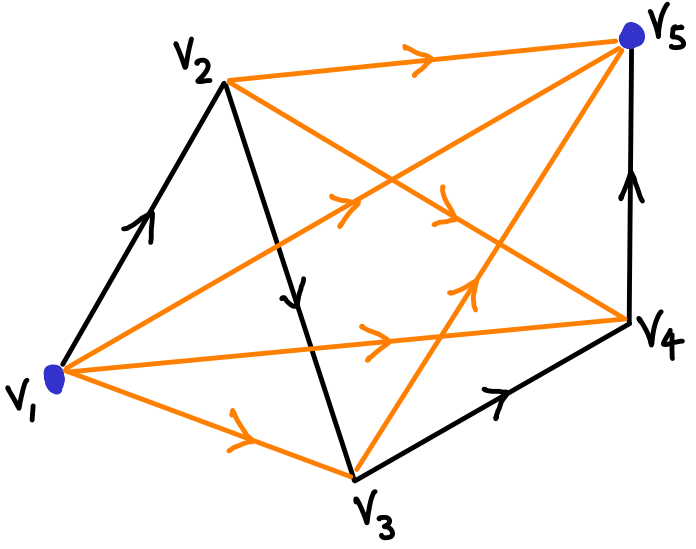
↳ goal: minimize error

- build the full DAG $\Theta(V^2)$
- compute error of each edge $O(V^2 \cdot V)$
- sort all error values $\Theta(V^2 \log V)$



Suppose we are willing to use up to k edges.

↳ goal: minimize error



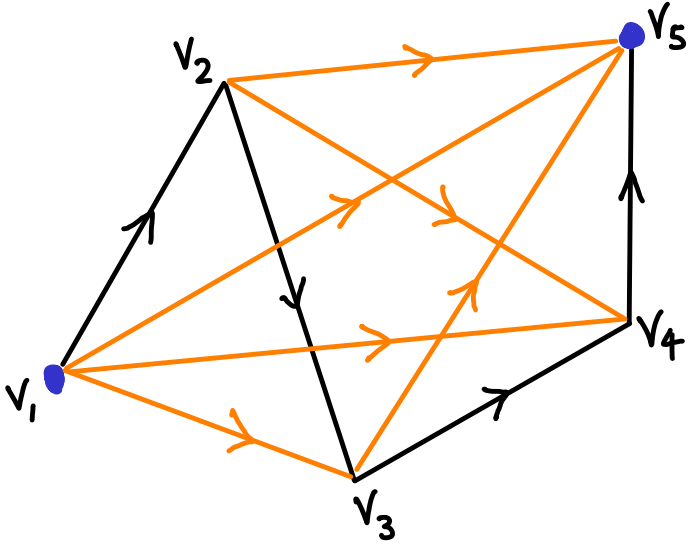
- build the full DAG $\Theta(V^2)$
- compute error of each edge $O(V^2 \cdot V)$
- sort all error values $\Theta(V^2 \log V)$
- binary search on error

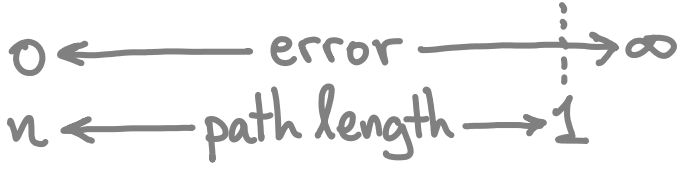
...?

Suppose we are willing to use up to k edges.

↳ goal: minimize error

- build the full DAG $\Theta(V^2)$
- compute error of each edge $O(V^2 \cdot V)$
- sort all error values $\Theta(V^2 \log V)$
- binary search on error

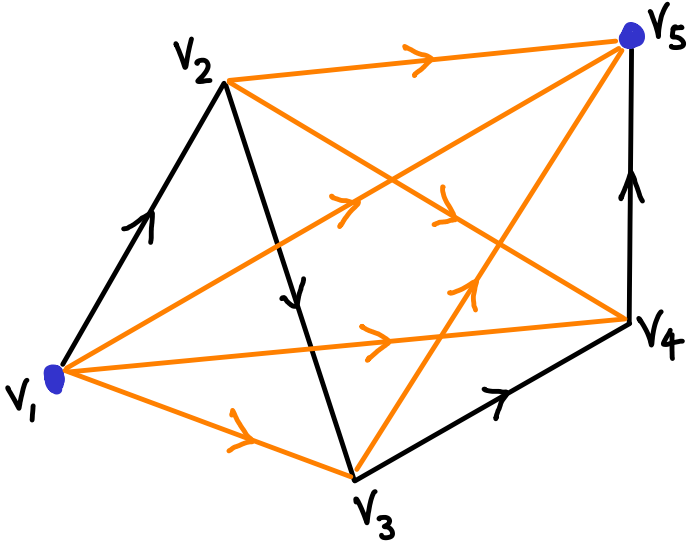


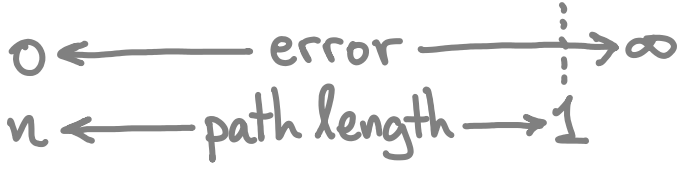


Suppose we are willing to use up to k edges.

↳ goal: minimize error

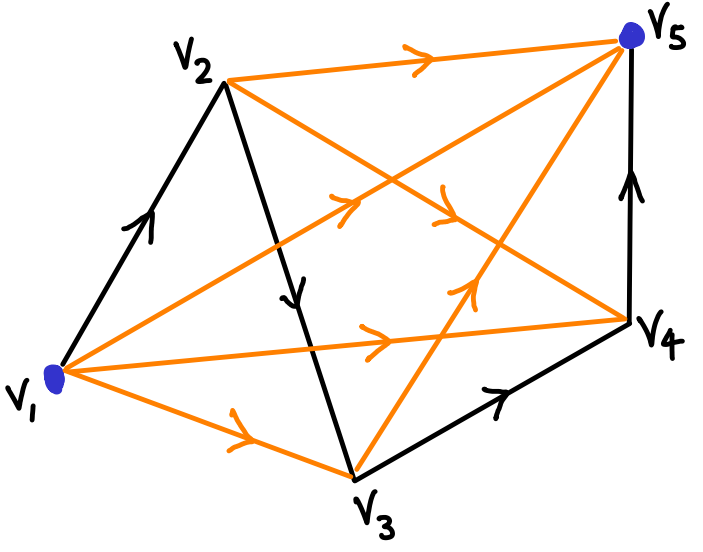
- build the full DAG $\Theta(V^2)$
- compute error of each edge $O(V^2 \cdot V)$
- sort all error values $\Theta(V^2 \log V)$
- binary search on error
  - ↳ for each error value
    - trim graph $O(V^2)$
    - find shortest path length $O(V^2)$



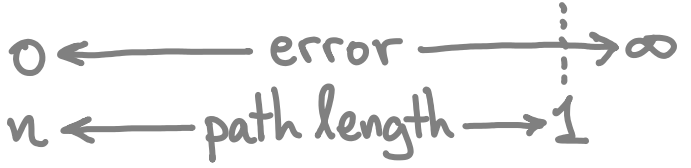


Suppose we are willing to use up to k edges.

↳ goal: minimize error



- build the full DAG $\Theta(V^2)$
- compute error of each edge $O(V^2 \cdot V)$
- sort all error values $\Theta(V^2 \log V)$
- binary search on error $\log(V^2)$
  - ↳ for each error value
    - trim graph $O(V^2)$
    - find shortest path length $O(V^2)$ → compare to k, adjust error



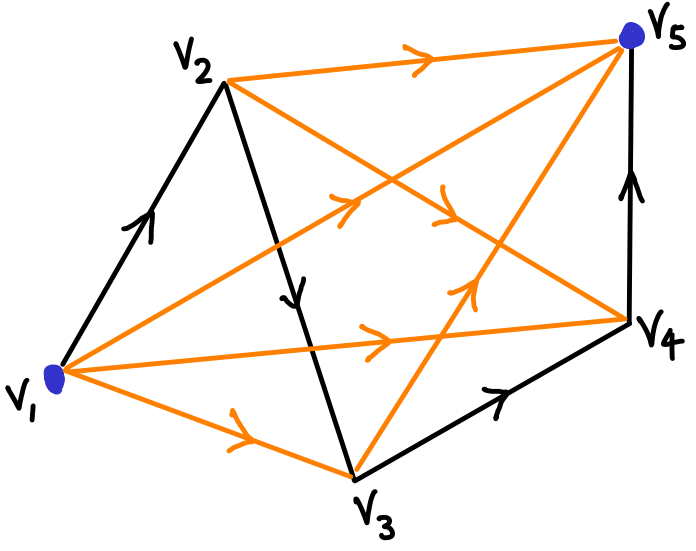
Suppose we are willing to use up to k edges.

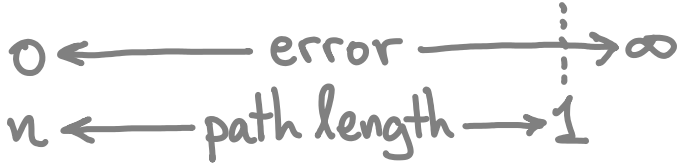
↳ goal: minimize error

- build the full DAG $\Theta(V^2)$
- compute error of each edge $O(V^2 \cdot V)$
- sort all error values $\Theta(V^2 \log V)$
- binary search on error $\log(V^2)$

  ↳ for each error value
  - trim graph $O(V^2)$
  - find shortest path length $O(V^2)$ → compare to k, adjust error

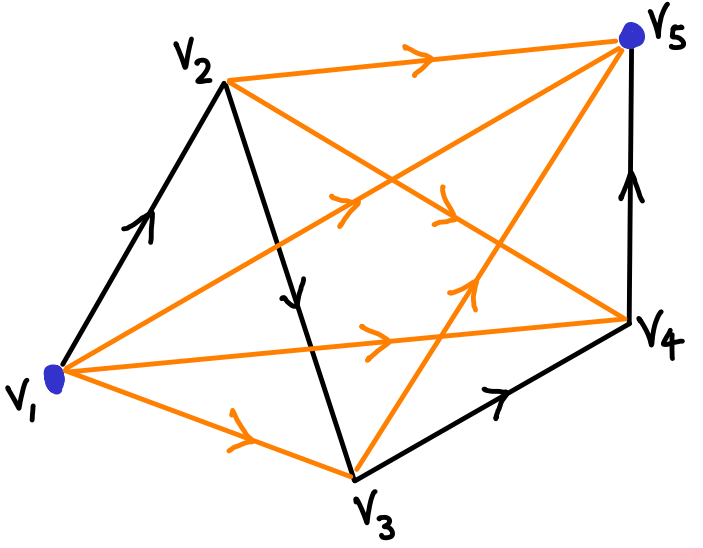
$O(V^2 \log V)$



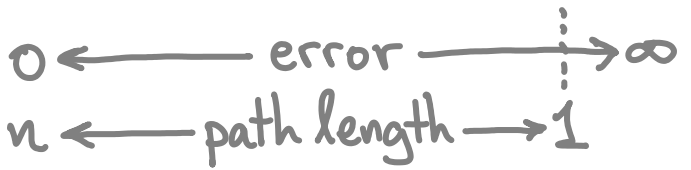


Suppose we are willing to use up to k edges.

↳ goal: minimize error



- build the full DAG $\Theta(V^2)$
- compute error of each edge $O(V^2 \cdot V)$ $O(V^2 \log V)$
- sort all error values $\Theta(V^2 \log V)$
- binary search on error $\log(V^2)$
  - ↳ for each error value
    - trim graph $O(V^2)$
    - find shortest path length $O(V^2)$ → compare to k, adjust error



Dynamic programming improvement?

Recap:

- lots of definitions for "similarity"
  - ↳ & for comparing curves

Recap:

- lots of definitions for "similarity"
  - ↳ & for comparing curves

- time complexity of Iri-Imai graph approach has been improved
  - ↳ stronger techniques, pre-processing, model, assumptions

Recap:

- lots of definitions for "similarity"
  - ↳ & for comparing curves
- time complexity of Iri-Imai graph approach has been improved
  - ↳ stronger techniques, pre-processing, model, assumptions
- the problem can also be made harder
  - ↳ e.g. require non-crossing output

Recap:

- lots of definitions for "similarity"
  - ↳ & for comparing curves
- time complexity of Iri-Imai graph approach has been improved
  - ↳ stronger techniques, pre-processing, model, assumptions
- the problem can also be made harder
  - ↳ e.g. require non-crossing output
- at least two formulations: MIN #edges or error tolerance

Recap:

- lots of definitions for "similarity"
  - ↳ & for comparing curves
- time complexity of Iri-Imai graph approach has been improved
  - ↳ stronger techniques, pre-processing, model, assumptions
- the problem can also be made harder
  - ↳ e.g. require non-crossing output
- at least two formulations: MIN #edges or error tolerance

Lots of project possibilities