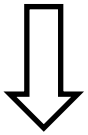
Detecting Superbubbles in Assembly Graphs

Taku Onodera (U. Tokyo)
Kunihiko Sadakane (NII)
Tetsuo Shibuya (U. Tokyo)

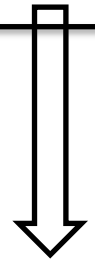
de Bruijn Graph-based Assembly

Reads (substrings of original DNA sequence)



de Bruijn graph / Unipath graph

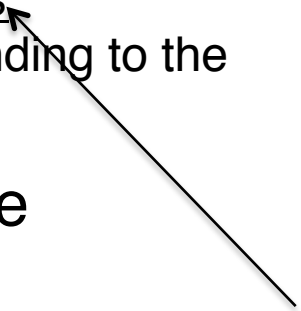
Well-studied
Elegant theories exist



1. remove motifs of errors
2. find the path corresponding to the original sequence

Relatively open
Existing solutions are ad-hoc

Whole genome sequence

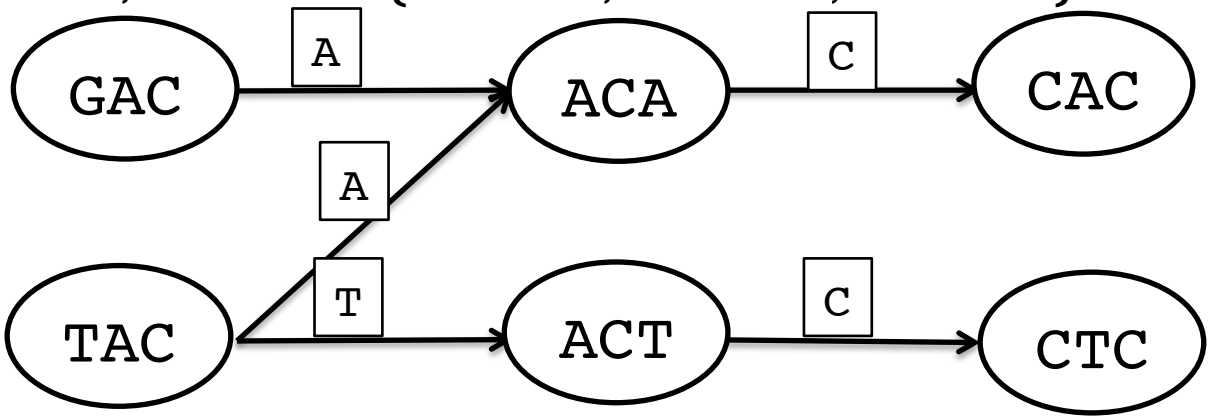


The current work revisits here

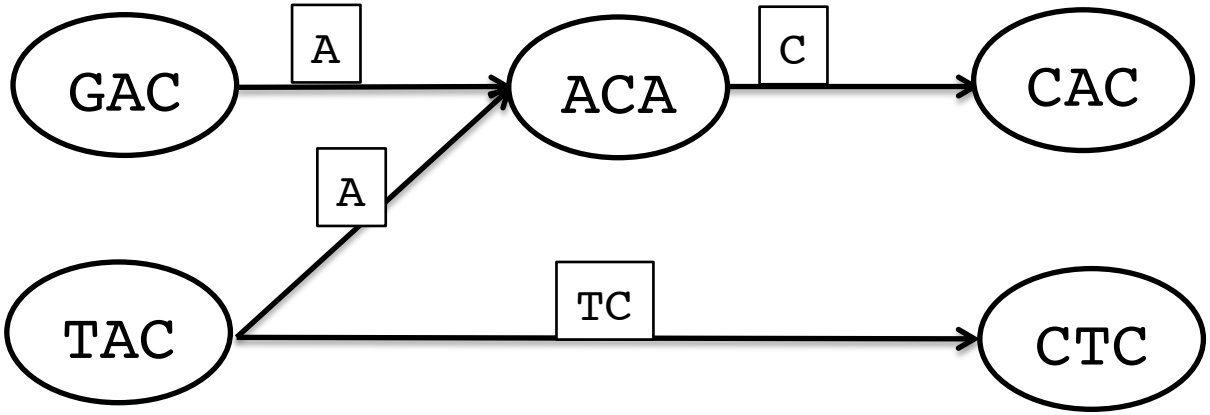
de Bruijn Graph & Unipath Graph

- de Bruijn Graph

  k=3, reads = {TACAC, TACTC, GACAC}



- Unipath Graph

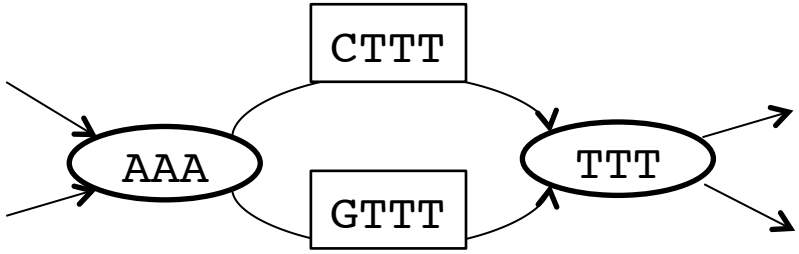


Existing Error Detection Methods

Bubbles are the most fundamental motif of errors.

reads

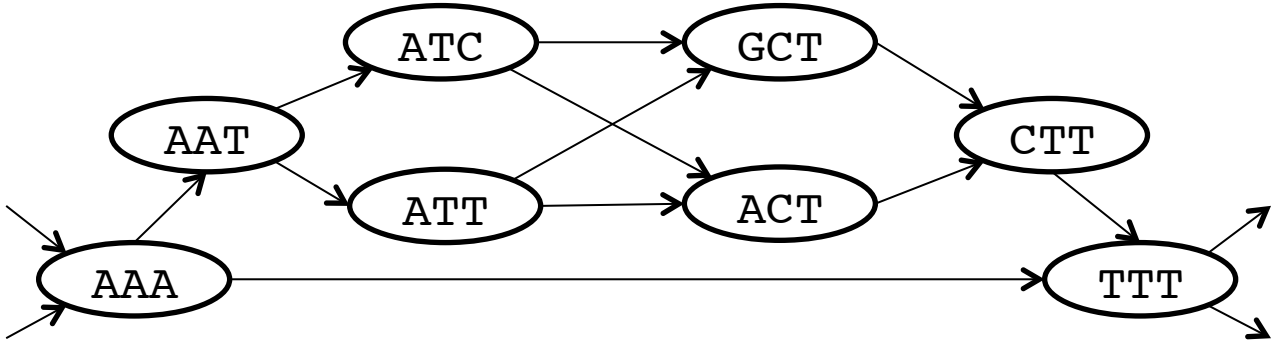
AAA**C**TTT
AAA**G**TTT



Bubbles are easy to detect but sometimes more complex structures do appear.

reads

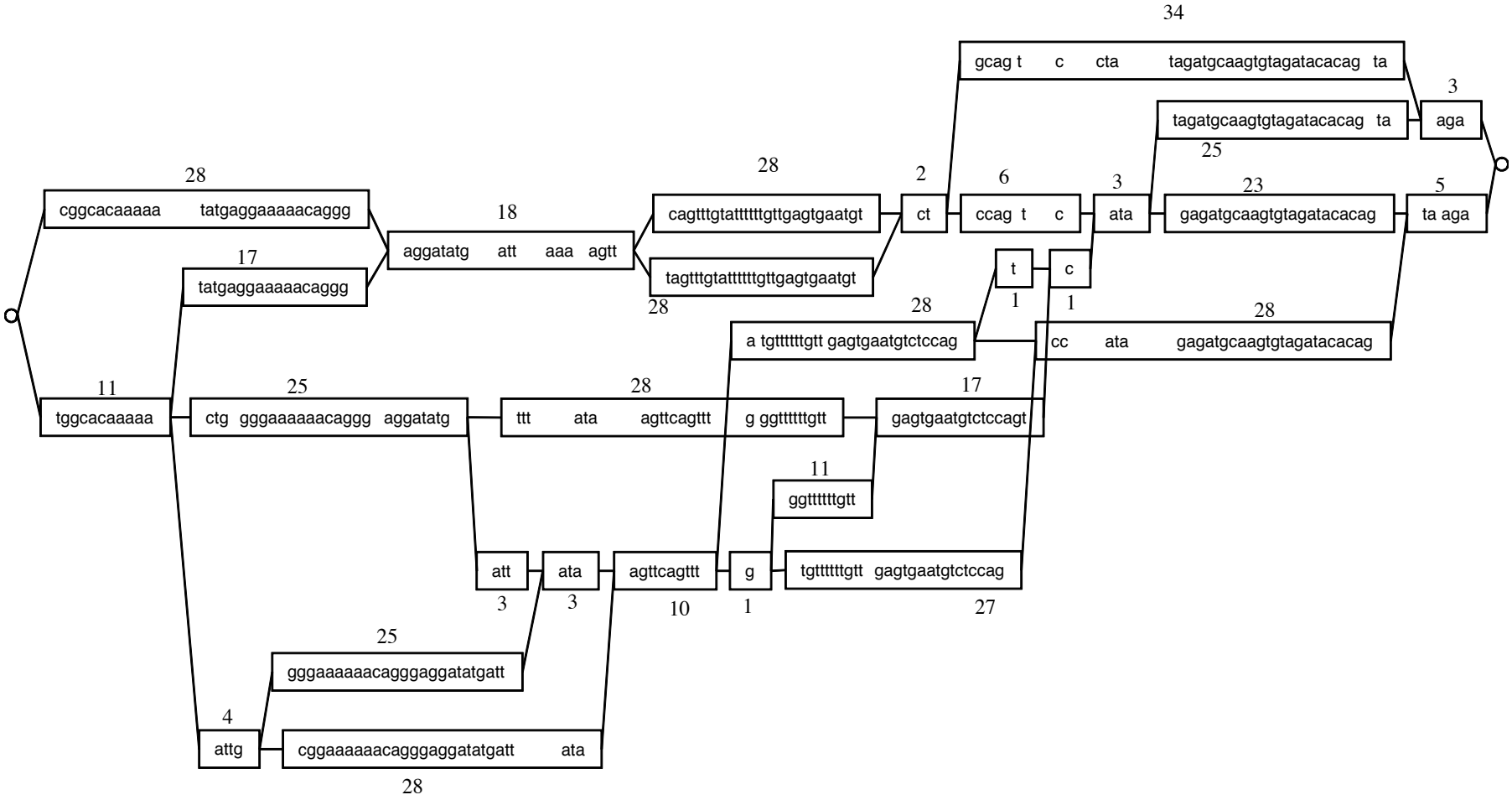
AAACCGGTTT
AAATCGCTTT
AAATCACTTT
AAATTGCTTT
AAATTACTTT



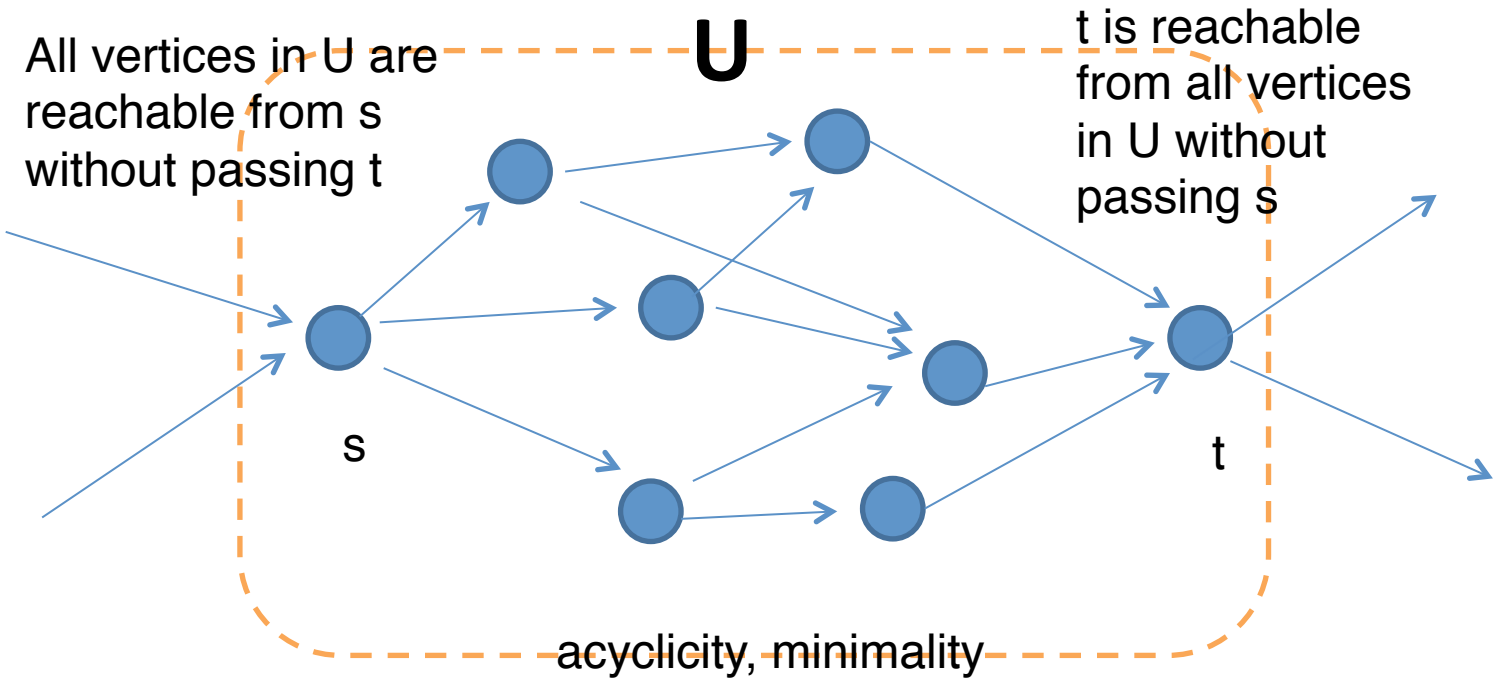
How to find such structures?

Our Results

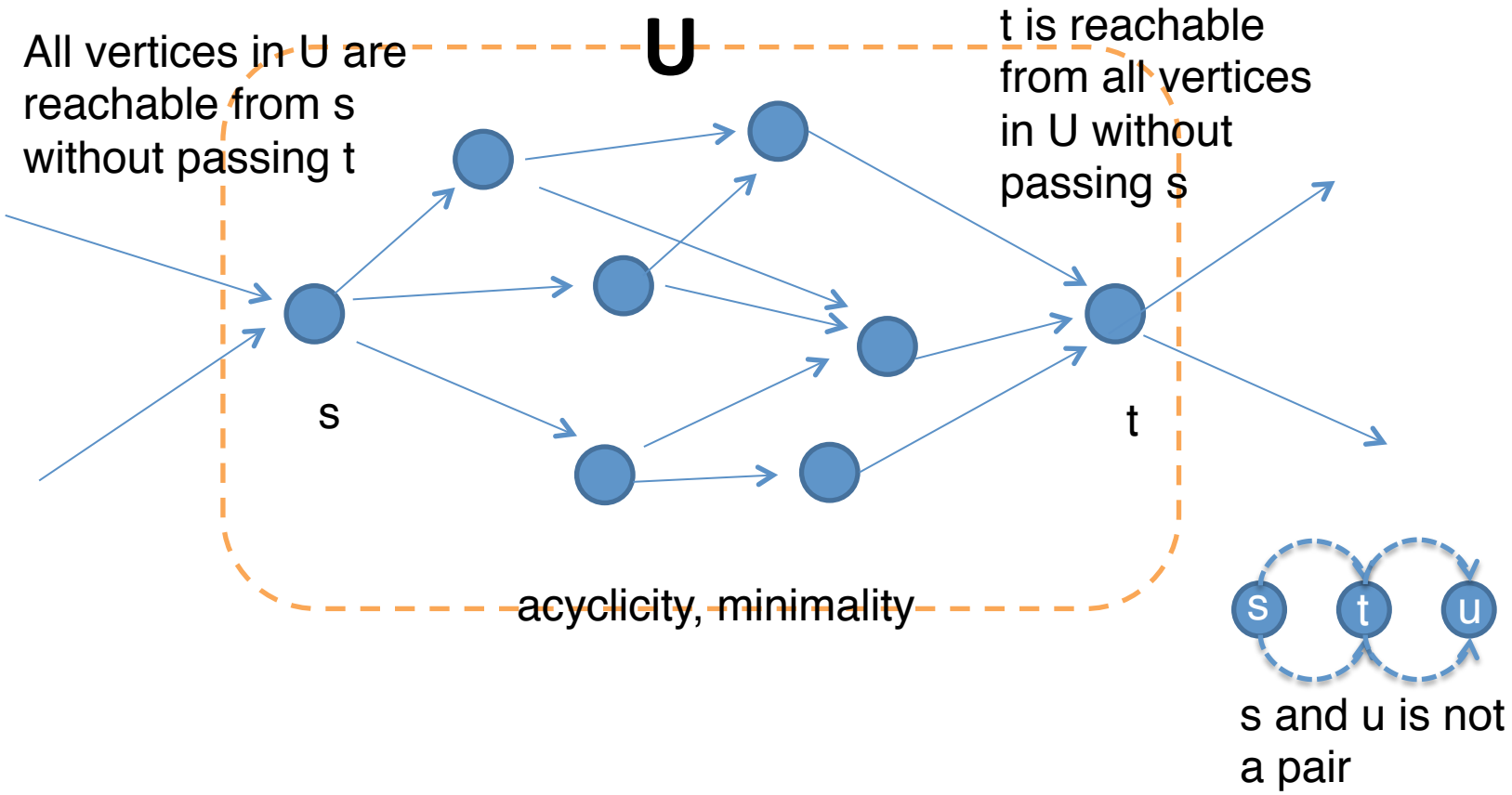
- Defined a new class of substructures **superbubbles**
- Expected linear (worst case quadratic)-time enumeration



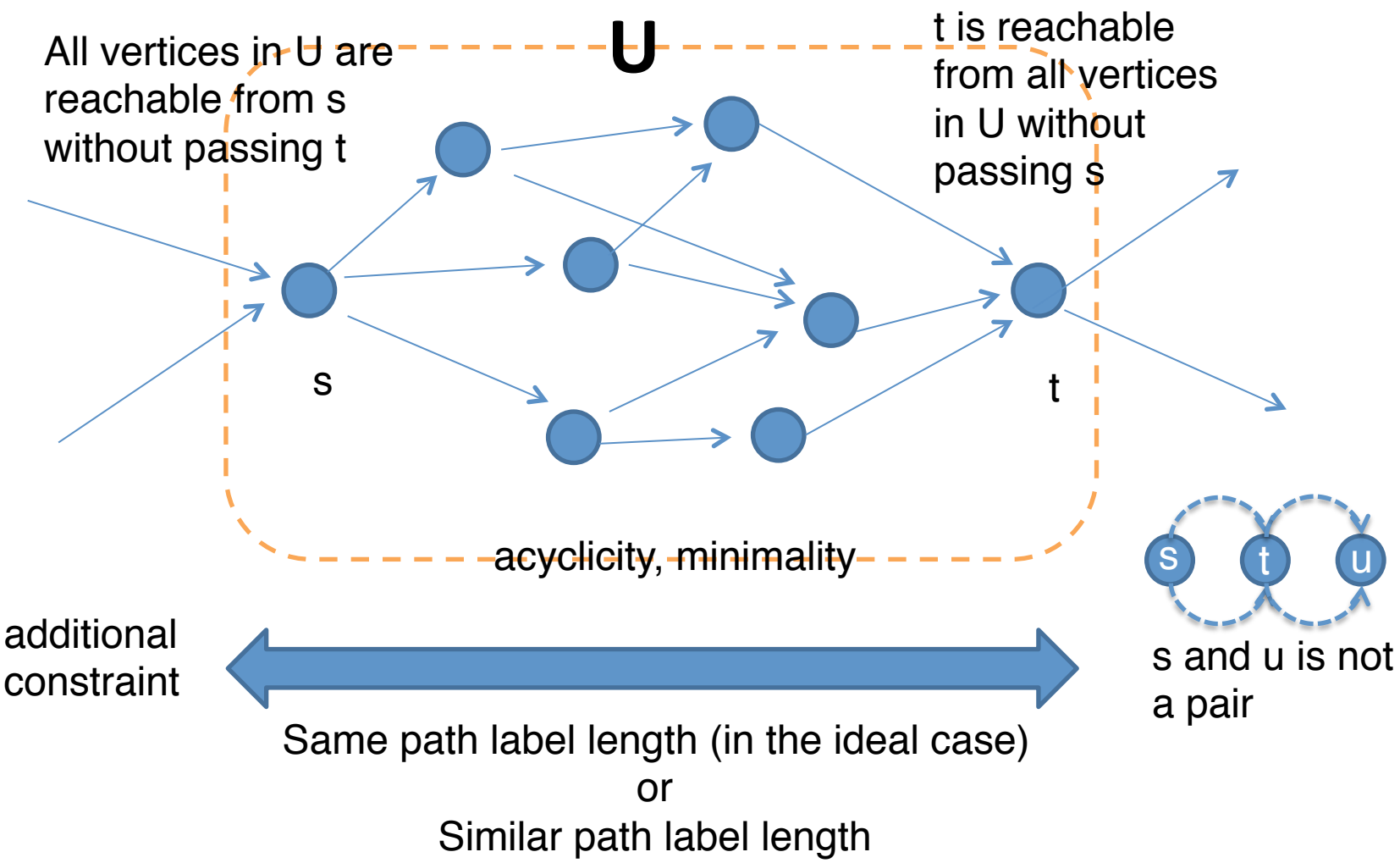
Superbubble



Superbubble



Superbubble



Fundamental Properties

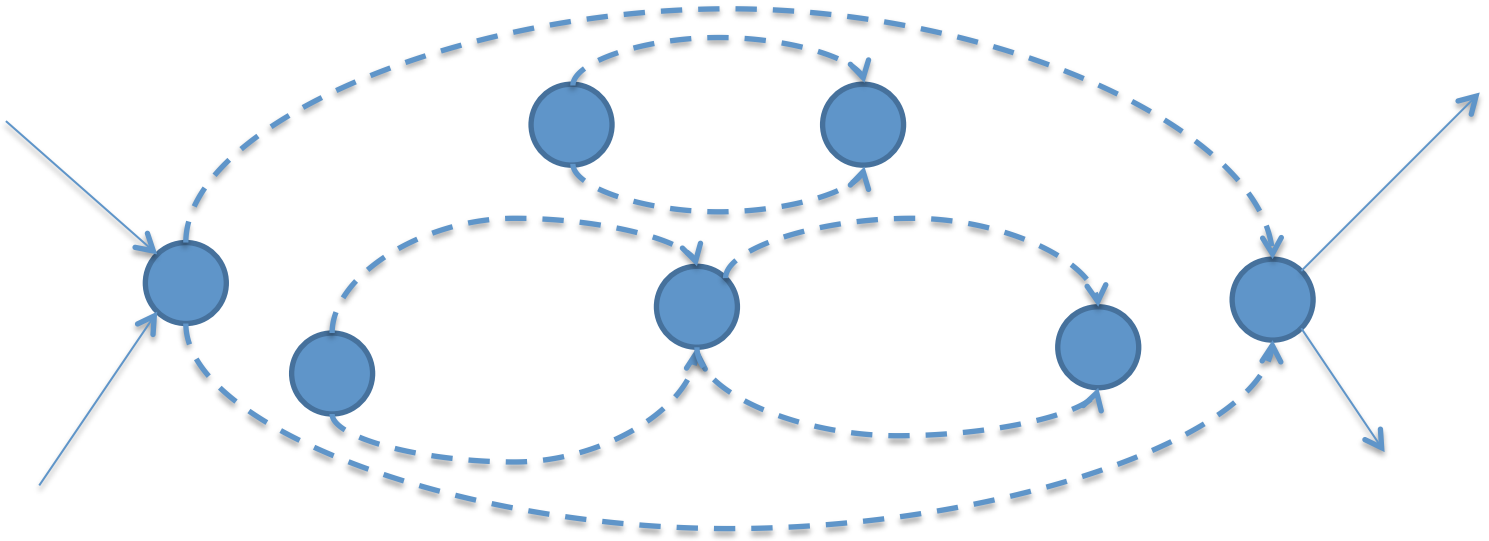
A superbubble can be specified by (s,t) (if (s,t) is known, it can be traversed in linear time)

∵ By topological sort

#Superbubbles = O(n)

∵ A vertex cannot be the entrance of >1 superbubbles

Distinct superbubbles are either separated or nested



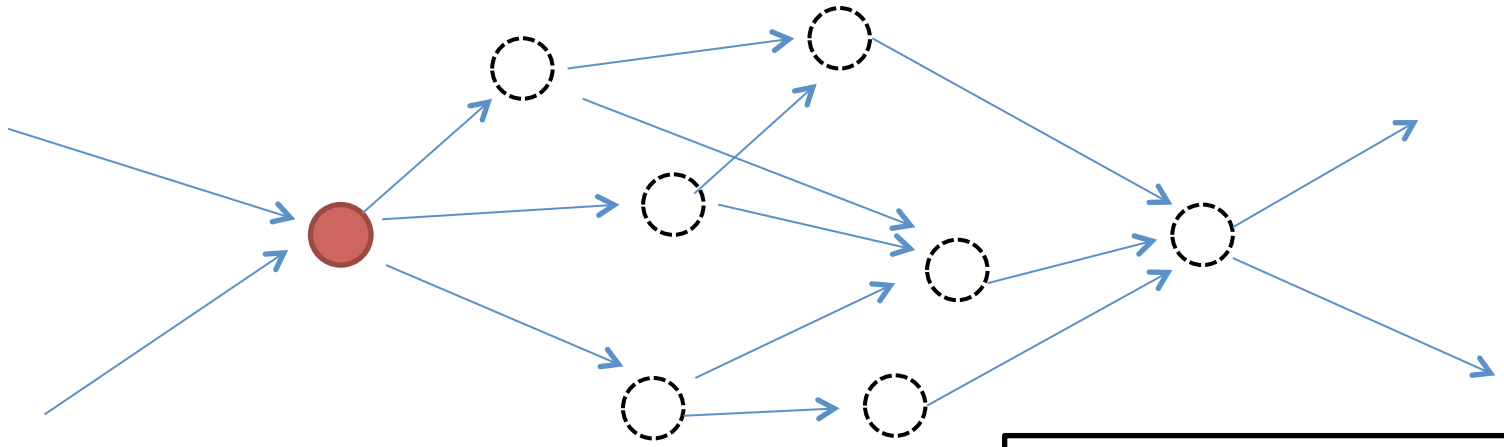
Enumeration

If you can check weather a vertex is the entrance of a superbubble (and find the corresponding exit) or not, check all vertices and you are done.

To check a vertex v is the entrance of a superbubble, topological sort from v.

Entrance Checking

Topological sorting from an entrance



unknown

some parents are visited, others are not

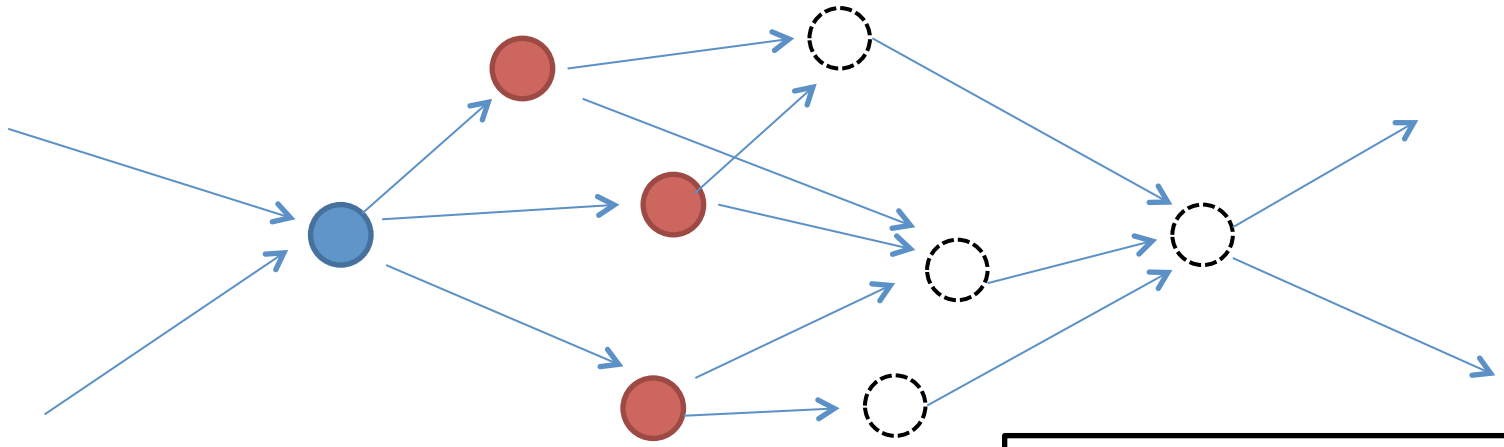
in queue

visited

```
tsort(u):
  enqueue u
  while queue is not empty
    v <- dequeue
    label v as visited
    for w in v's children
      if w's parents are all visited
        enqueue w
```

Entrance Checking

Topological sorting from an entrance



unknown

some parents are visited, others are not

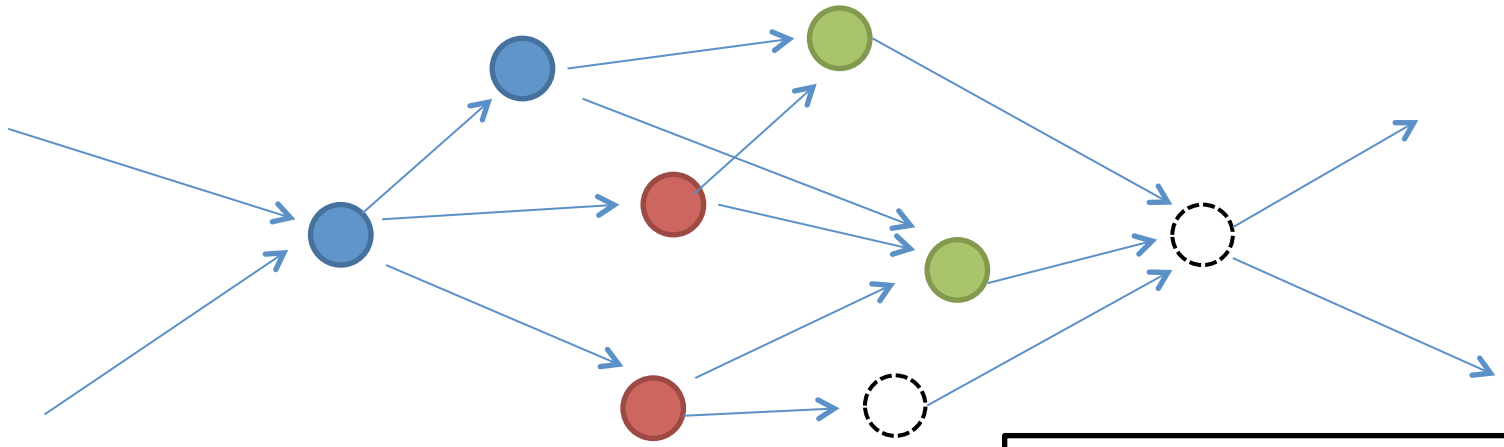
in queue

visited

```
tsort(u):
  enqueue u
  while queue is not empty
    v <- dequeue
    label v as visited
    for w in v's children
      if w's parents are all visited
        enqueue w
```

Entrance Checking

Topological sorting from an entrance



unknown

some parents are visited, others are not

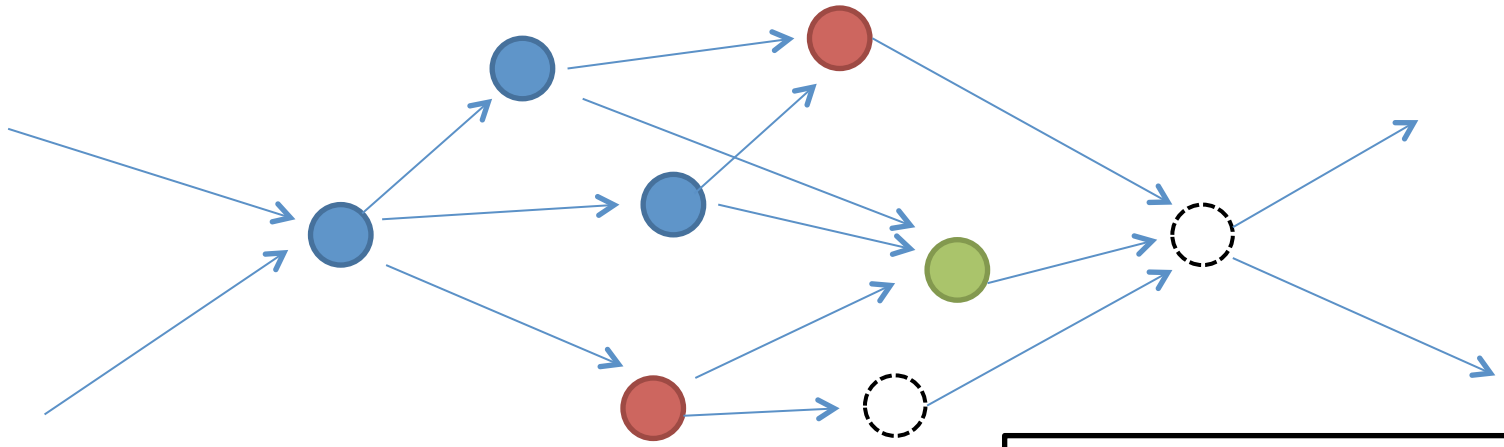
in queue

visited

```
tsort(u):
  enqueue u
  while queue is not empty
    v <- dequeue
    label v as visited
    for w in v's children
      if w's parents are all visited
        enqueue w
```

Entrance Checking

Topological sorting from an entrance



unknown

some parents are visited, others are not

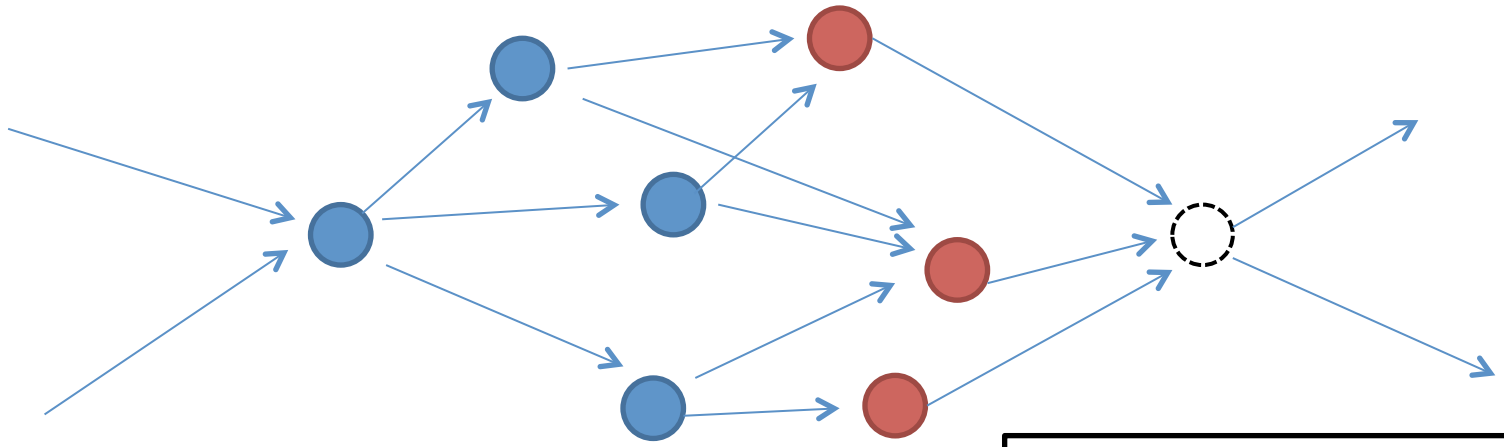
in queue

visited

```
tsort(u):
  enqueue u
  while queue is not empty
    v <- dequeue
    label v as visited
    for w in v's children
      if w's parents are all visited
        enqueue w
```

Entrance Checking

Topological sorting from an entrance



unknown

some parents are visited, others are not

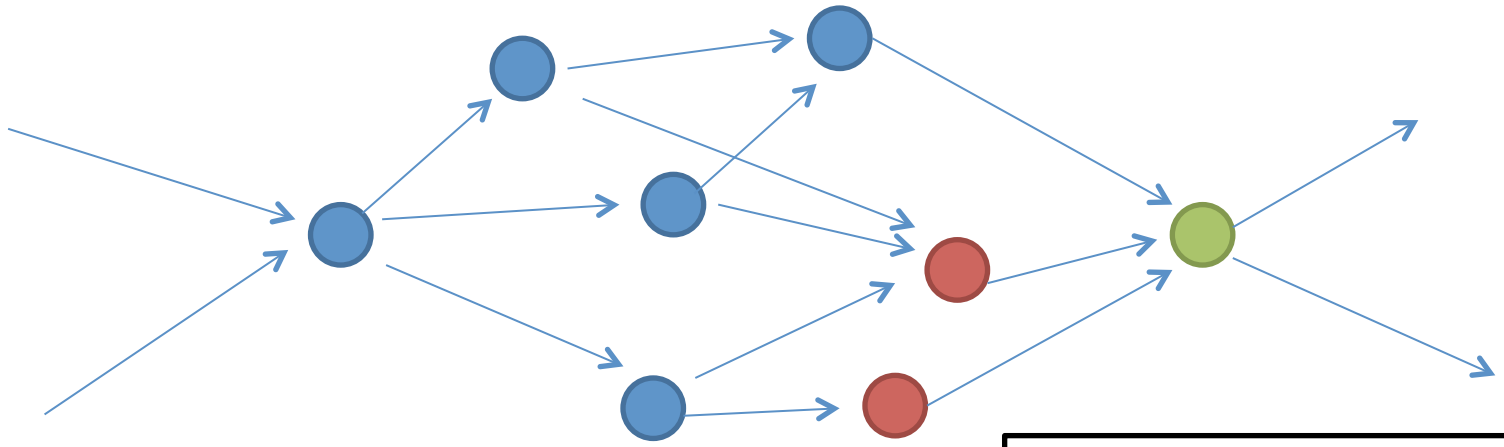
in queue

visited

```
tsort(u):
  enqueue u
  while queue is not empty
    v <- dequeue
    label v as visited
    for w in v's children
      if w's parents are all visited
        enqueue w
```

Entrance Checking

Topological sorting from an entrance



unknown

some parents are visited, others are not

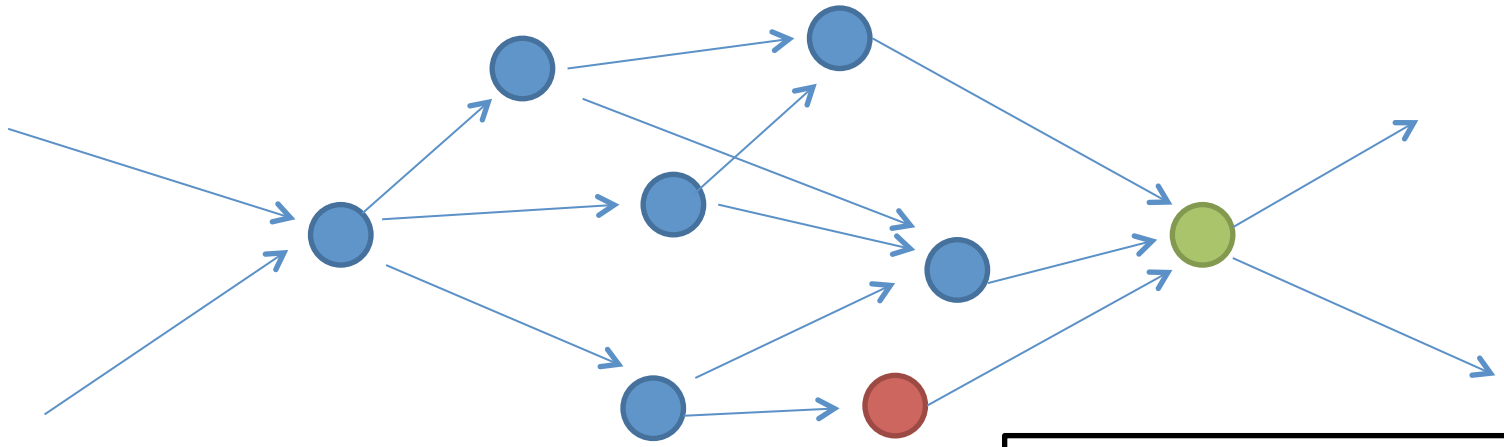
in queue

visited

```
tsort(u):
  enqueue u
  while queue is not empty
    v <- dequeue
    label v as visited
    for w in v's children
      if w's parents are all visited
        enqueue w
```

Entrance Checking

Topological sorting from an entrance



unknown

some parents are visited, others are not

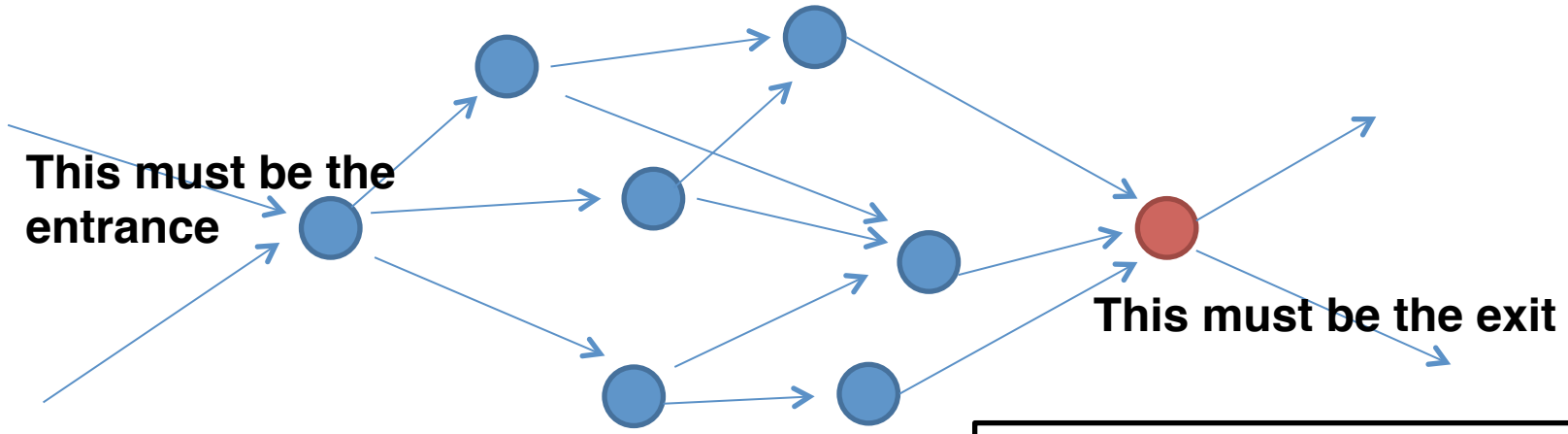
in queue

visited

```
tsort(u):
  enqueue u
  while queue is not empty
    v <- dequeue
    label v as visited
    for w in v's children
      if w's parents are all visited
        enqueue w
```

Entrance Checking

Topological sorting from an entrance



unknown

some parents are visited, others are not

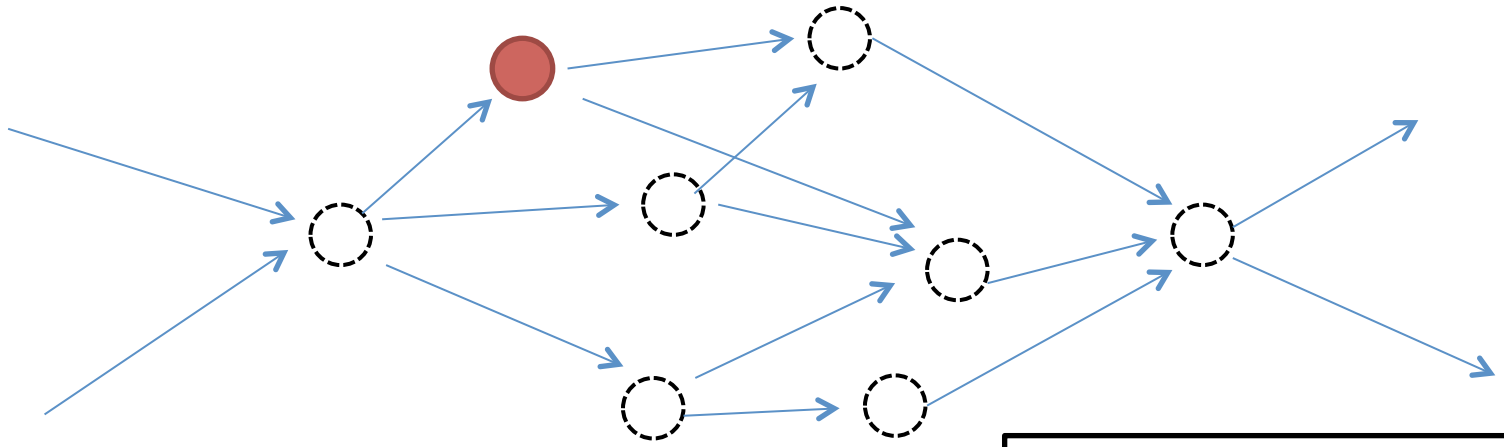
in queue

visited

```
tsort(u):
  enqueue u
  while queue is not empty
    v <- dequeue
    label v as visited
    for w in v's children
      if w's parents are all visited
        enqueue w
```

Entrance Checking

Topological sorting from a non-entrance



unknown

some parents are visited, others are not

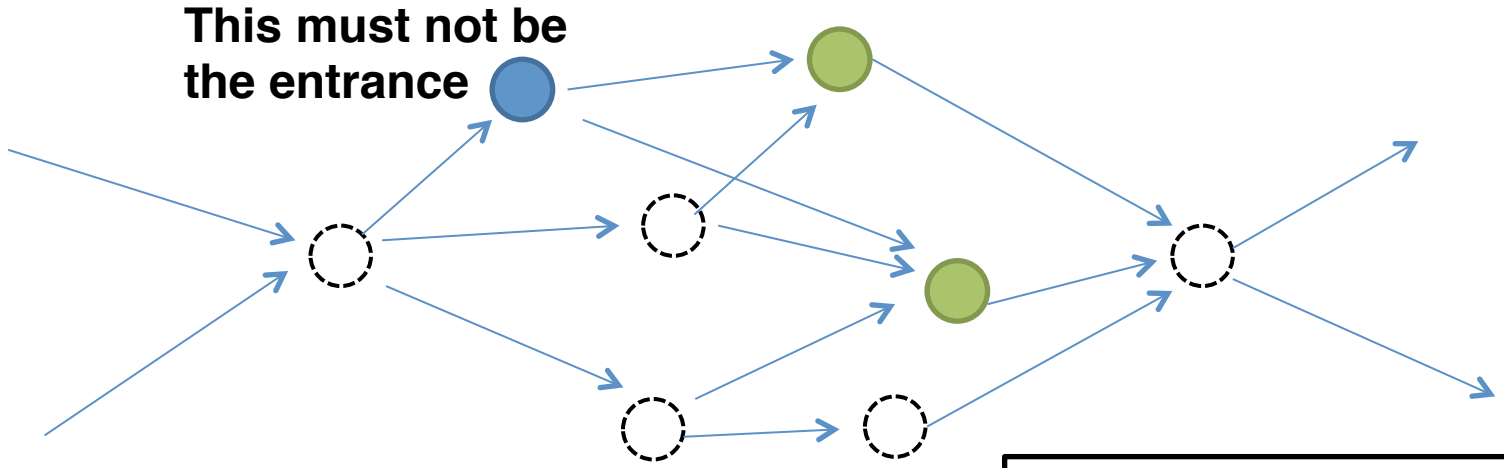
in queue

visited

```
tsort(u):
  enqueue u
  while queue is not empty
    v <- dequeue
    label v as visited
    for w in v's children
      if w's parents are all visited
        enqueue w
```

Entrance Checking

Topological sorting from a non-entrance



unknown

some parents are visited, others are not

in queue

visited

```
tsort(u):
  enqueue u
  while queue is not empty
    v <- dequeue
    label v as visited
    for w in v's children
      if w's parents are all visited
        enqueue w
```

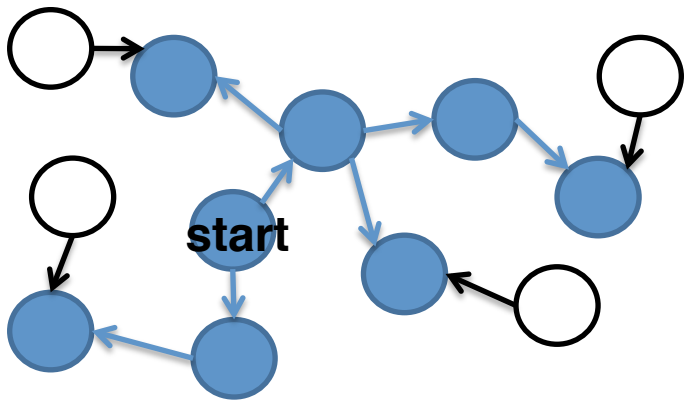
Runtime

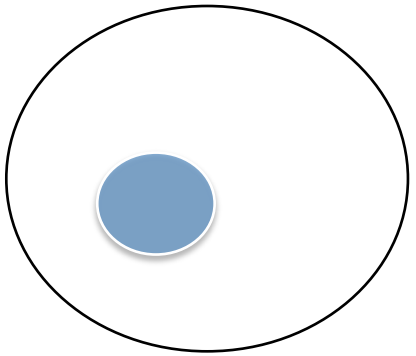
Topological sorting O(n+m)-time from every vertex

⇒ O(n(n+m))-time in the worst case

O(n)-time in expectation under a reasonable model

∵ topological sort from most vertices halts instantly





E[size of search tree] < const.
if E[#children of each vertex] < 1

Experiment

Procedures

1. Constructed a unipath graph from Human genome reads of 40× coverage (via the succinct de Bruijn graph [Bowe et al. WABI'12])
2. Enumerated all superbubbles in the unipath graph

Results

Procedure 2 took 12min. by a single core machine.

The histogram of the size of superbubbles

size	3-9	10-19	20-29	30-39	40-49	50-59	60-
#S.B.	71663	4295	347	69	21	8	5

For 86.3% of 23,078 superbubbles of size >=5, ratio between the longest/shortest path label length< 1.05

Summary

- Defined superbubbles
- An efficient (both theoretically and practically) algorithm to enumerate all superbubbles in graphs
  - Found many superbubbles in data from Human genome reads

Future work

- Error/Variation separation
- Find the right path in superbubbles
- Worst-case O(n)-time algorithm