CH介绍

潘睿

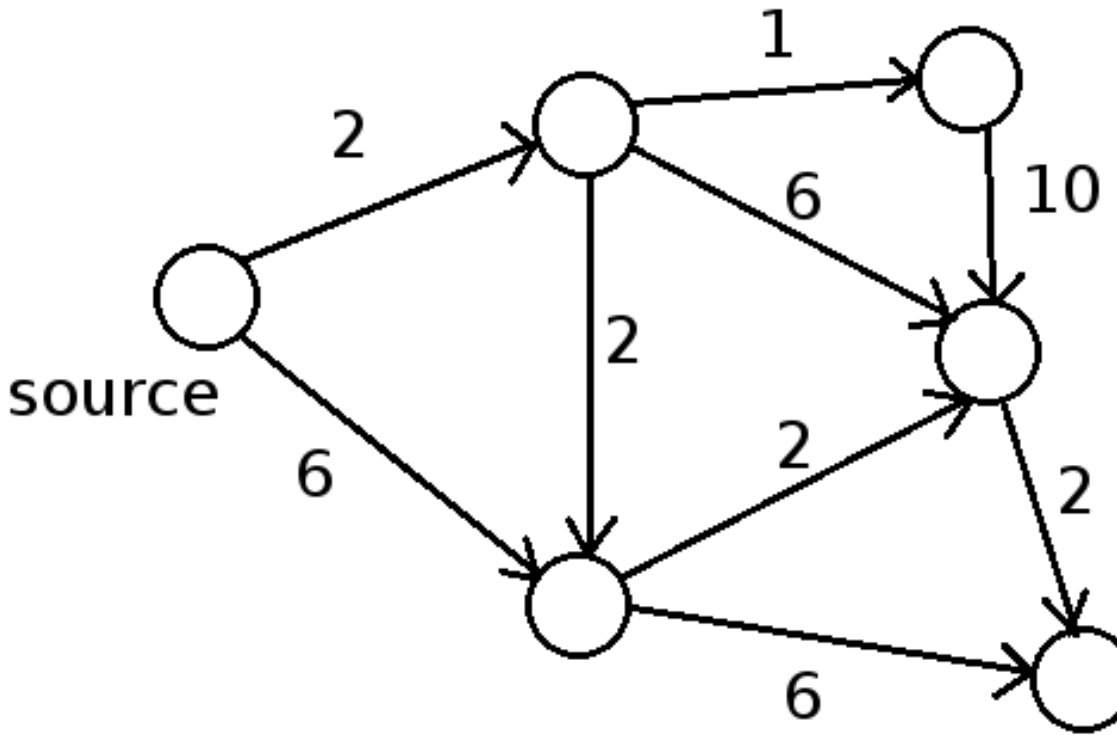
目录

- **概览**
- **历史& 基础知识**
- **简介**
- **算法细节 & 优化**
- 算法改进

概览

概览

- 问题：
  - 给定有向图G=<V,E>和图中两点s,t，求解s到t的最短路径



概览

- 挑战：
  - 实际中的巨大规模使得实时路径规划变得困难！

概览

- 2015年为止的大部分知名算法在West Europe道路网络数据集（18M个节点，42.5M条有向边）中的表现

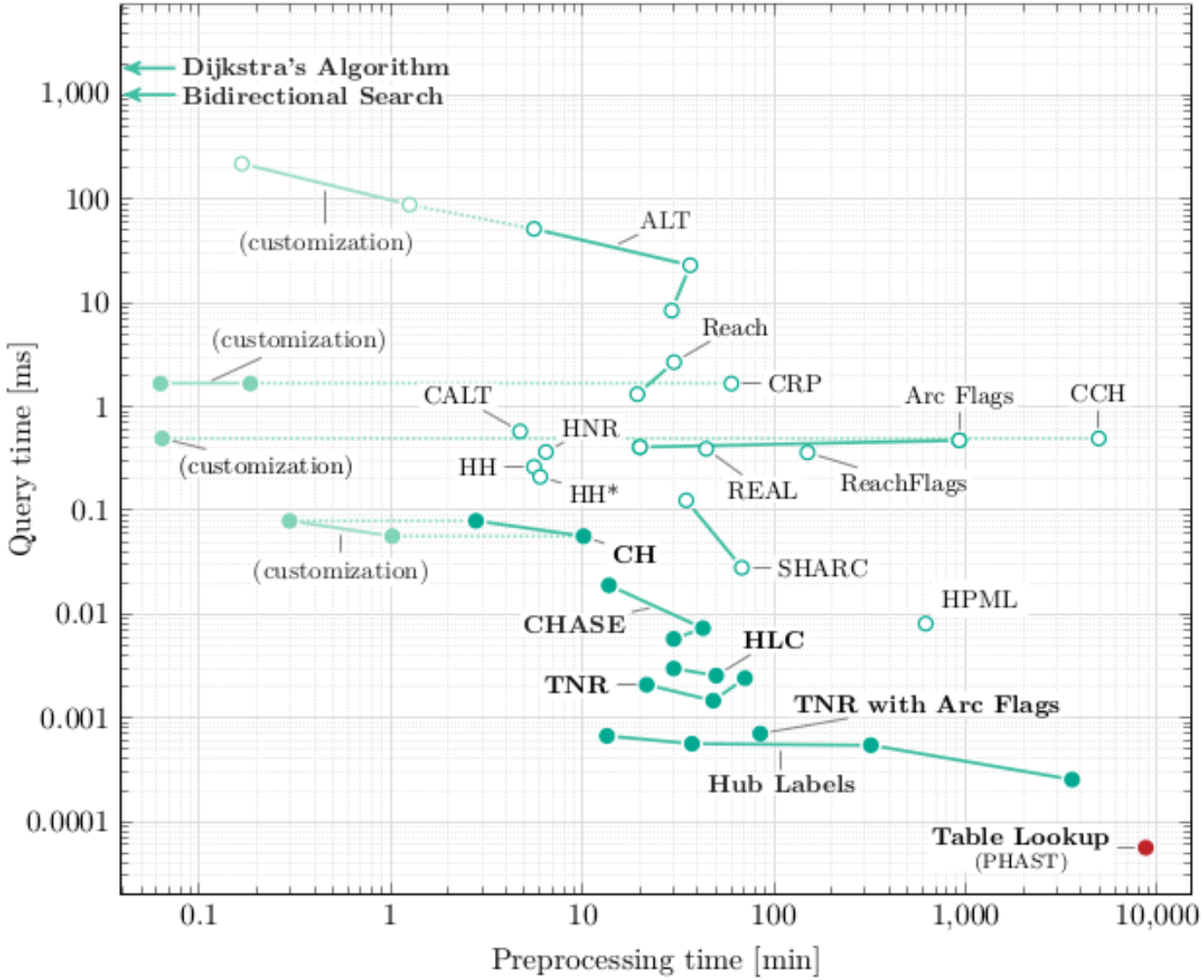


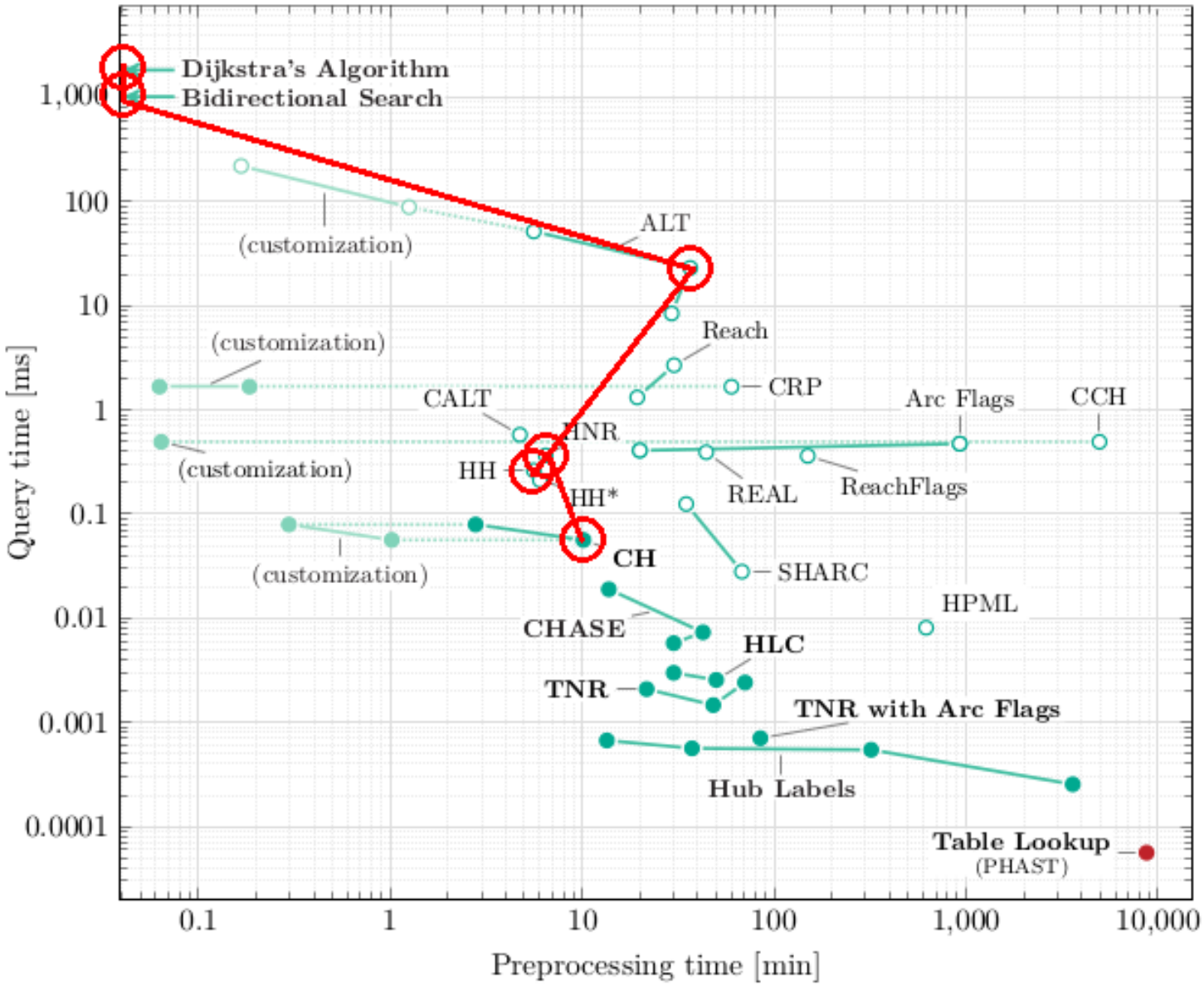
Figure 7. Preprocessing and average query time performance for algorithms with available experimental data on the road network of Western Europe, using travel times as edge weights. Connecting lines indicate different trade-offs for the same algorithm. The figure is inspired by [238].

历史 & 基础知识

历史 & 基础知识

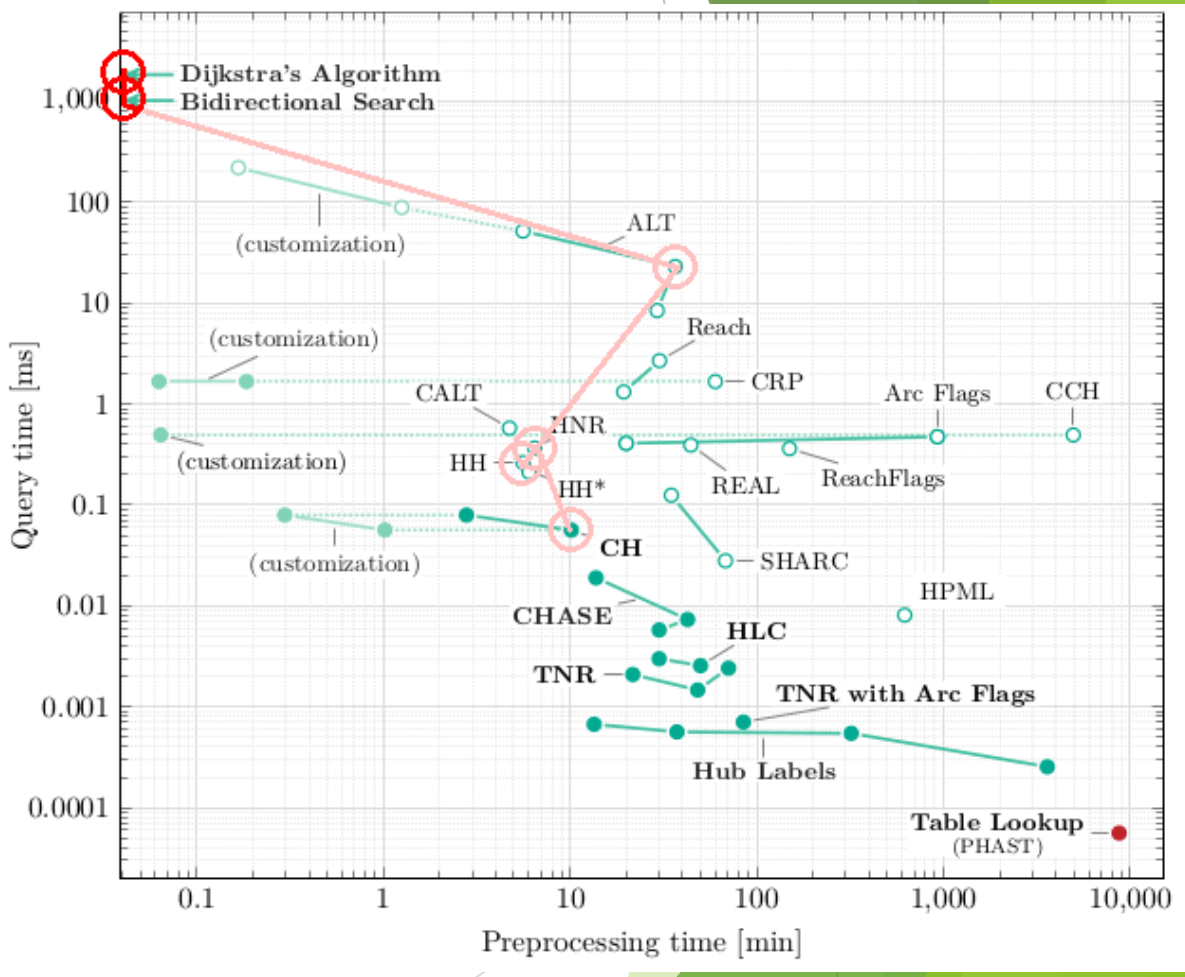
- **[1959] Dijkstra最短路算法**
  - **起源**
- [~1968] Heuristic A*搜索算法
  - 结合实际信息
- [1991, 2002] Heuristic Hierarchical Approaches
  - 层次化
- [2005] Highway Hierarchies (HH)
  - 层次化的精确解
- [2005] Highway-Node Routing (HNR)
  - 层次化的一般抽象
- **[2008] Contraction Hierarchies (CH)**

历史 & 基础知识


Dijkstra's Algorithm
Bidirectional Search
(customization)
ALT
(customization)
Reach
CRP
CALT
Arc Flags
CCH
HNR
(customization)
HH
REAL
ReachFlags
HH*
CH
(customization)
SHARC
HPML
CHASE
HLC
TNR
TNR with Arc Flags
Hub Labels
Table Lookup
(PHAST)
Query time [ms]
Preprocessing time [min]
1,000
100
10
1
0.1
0.01
0.001
0.0001
0.1
1
10
100
1,000
10,000

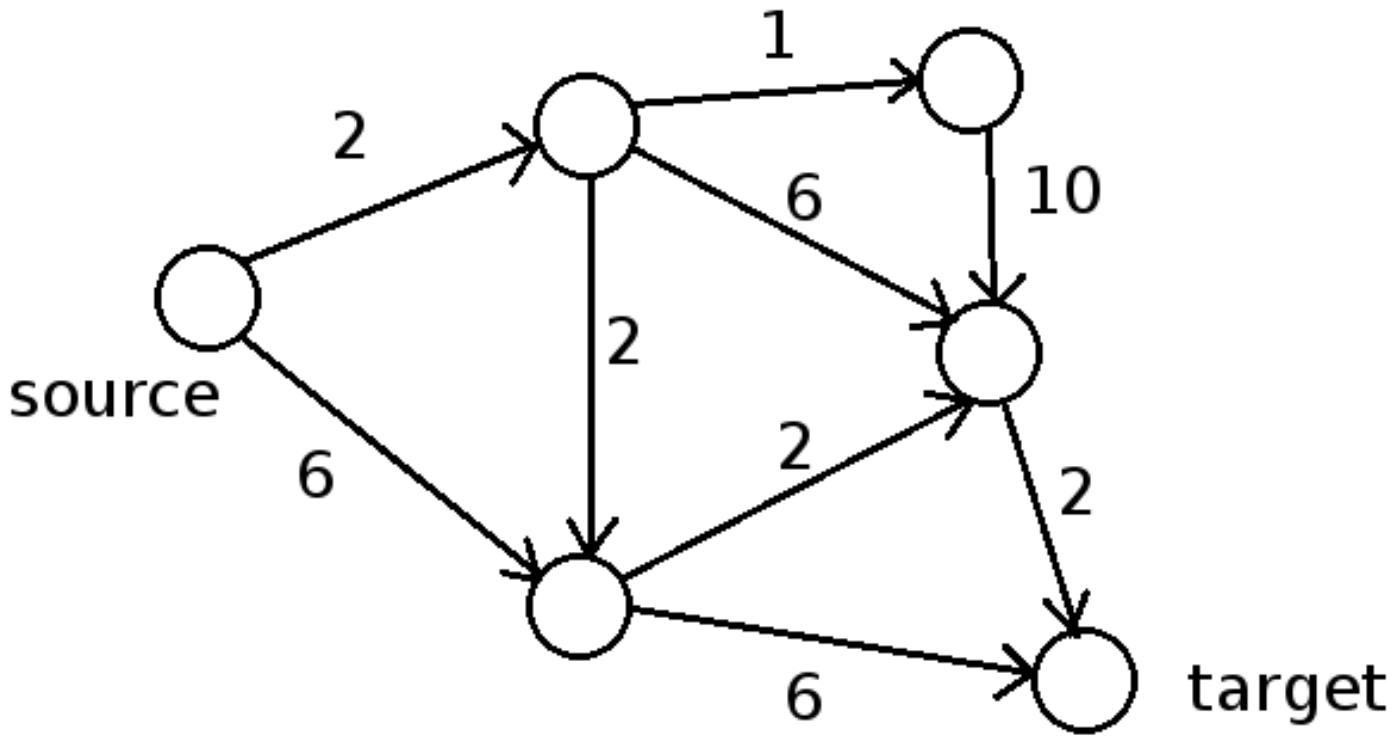
历史 & 基础知识

- [1959] Dijkstra最短路算法
  - 起源
- [~1968] Heuristic A*搜索算法
- [1991, 2002] Heuristic Hierarchical Approaches
- [2005] Highway Hierarchies (HH)
- [2005] Highway-Node Routing (HNR)
- [2008] Contraction Hierarchies (CH)



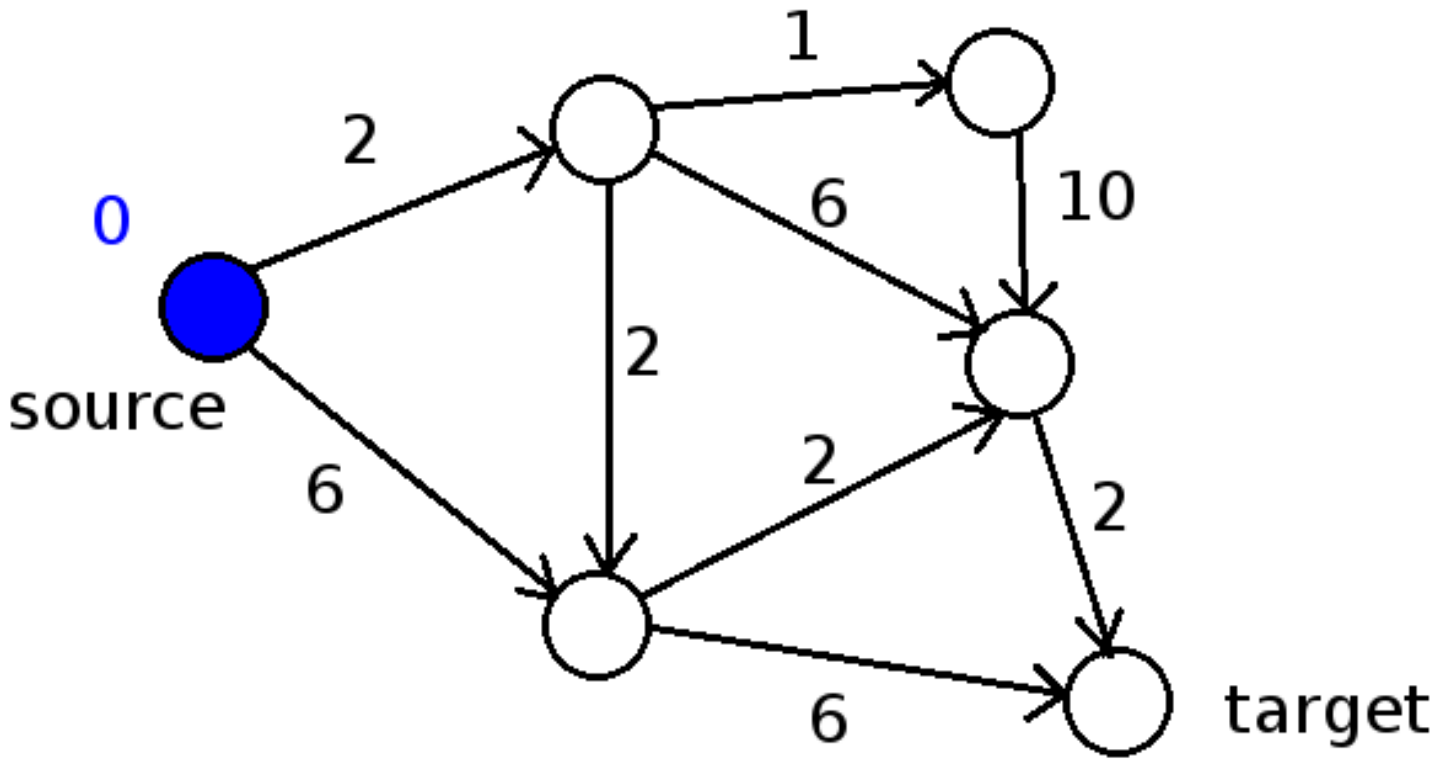
历史 & 基础知识

- Dijkstra（迪杰斯特拉）最短路算法



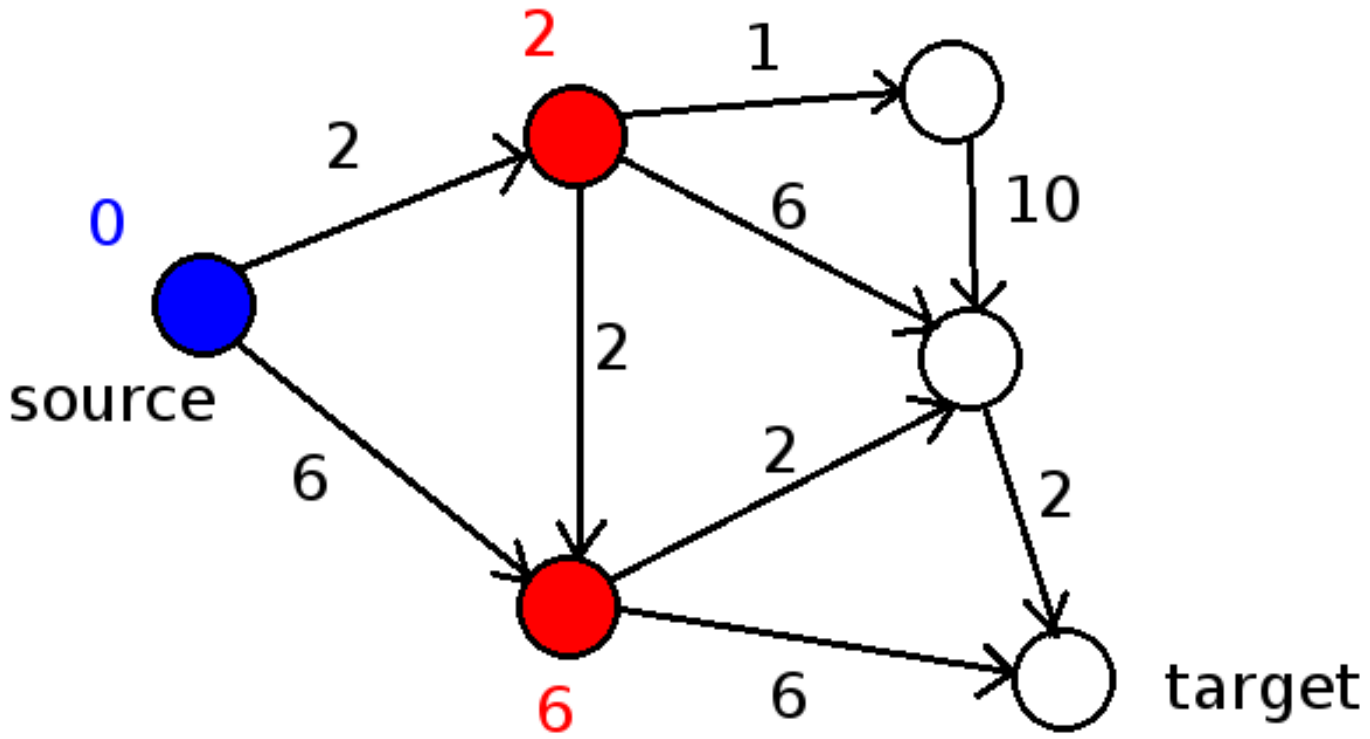
历史 & 基础知识

- Dijkstra（迪杰斯特拉）最短路算法



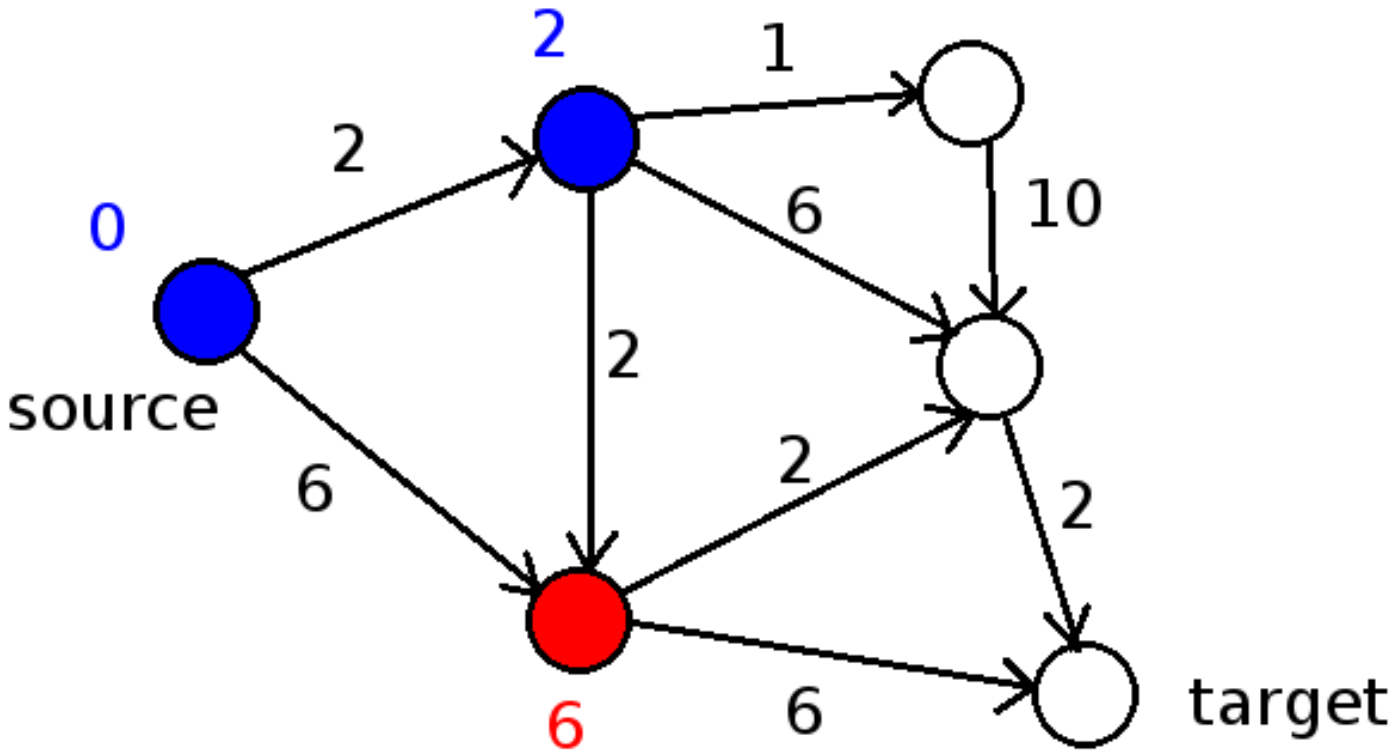
历史 & 基础知识

- Dijkstra（迪杰斯特拉）最短路算法



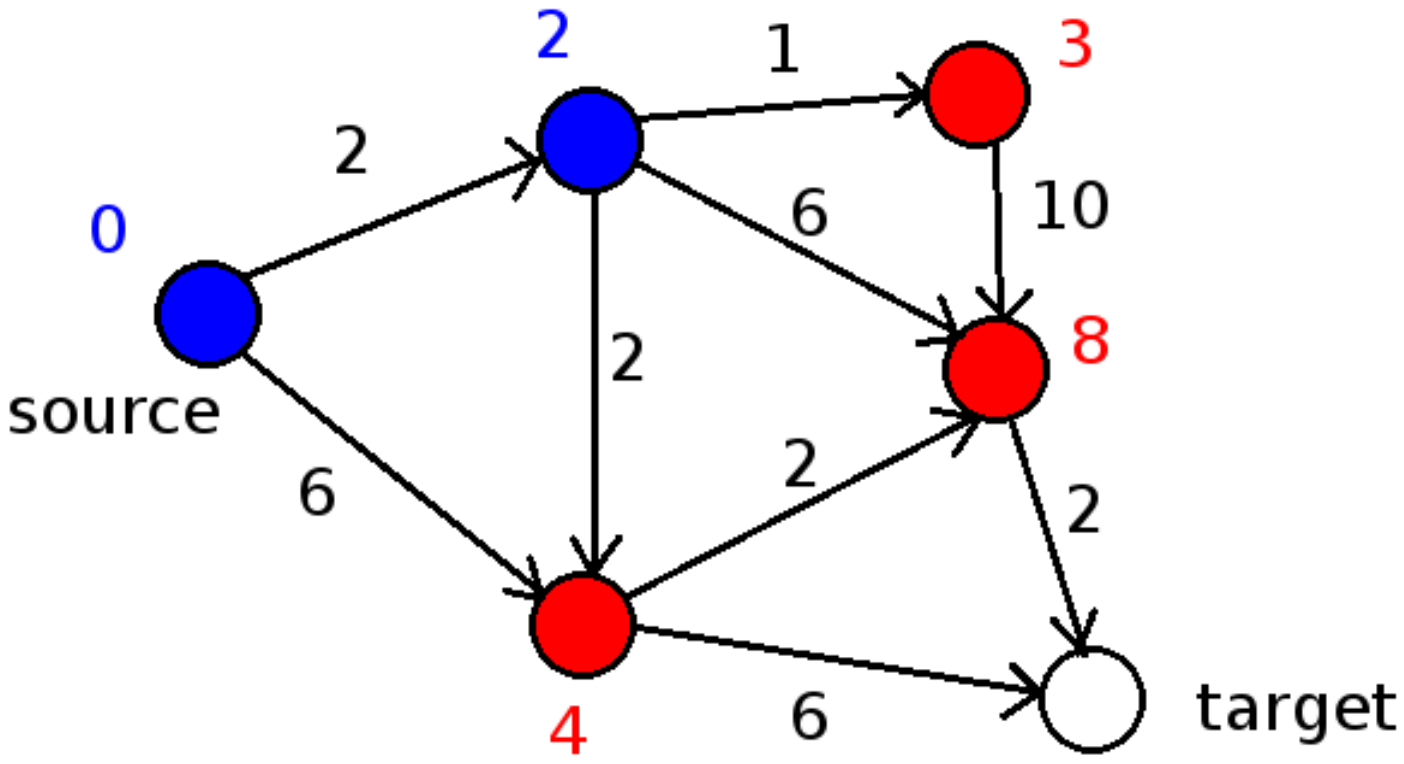
历史 & 基础知识

- Dijkstra（迪杰斯特拉）最短路算法



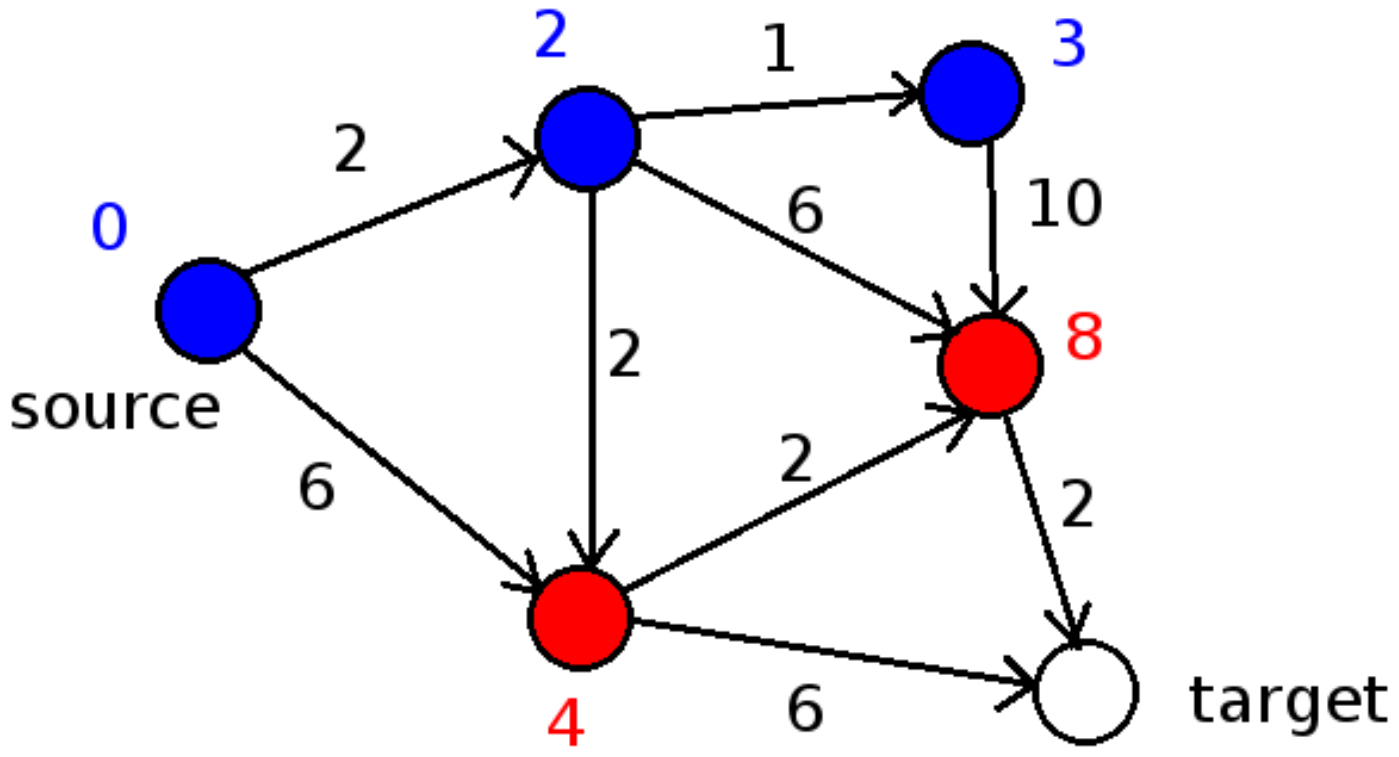
历史 & 基础知识

- Dijkstra（迪杰斯特拉）最短路算法



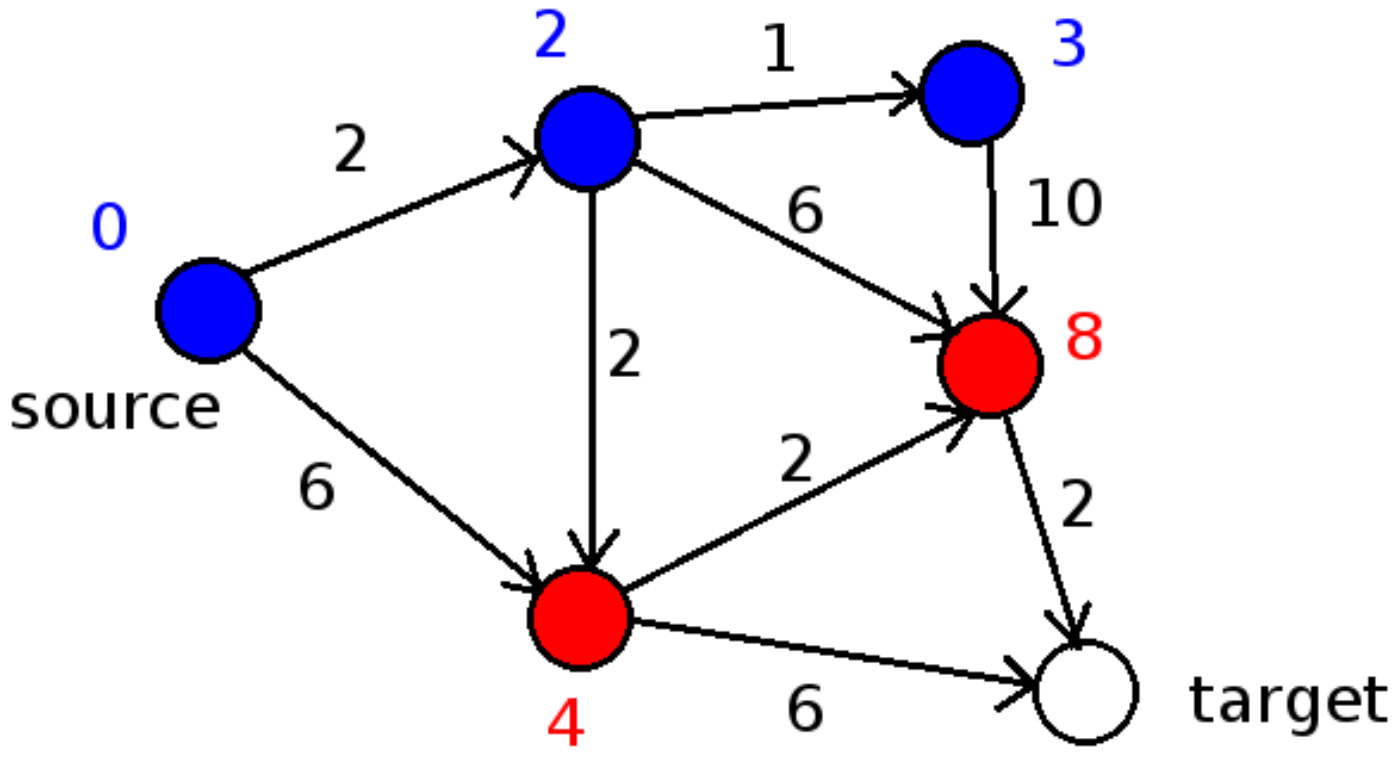
历史 & 基础知识

- Dijkstra（迪杰斯特拉）最短路算法



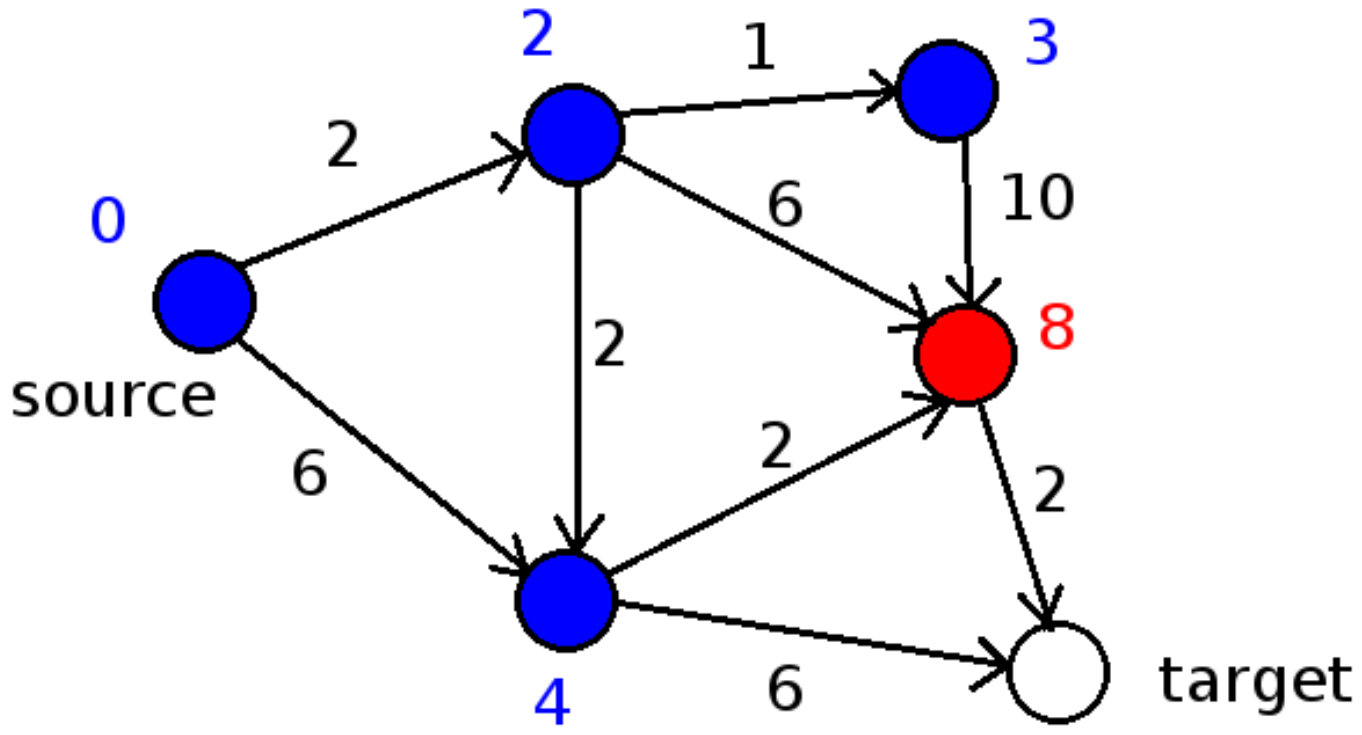
历史 & 基础知识

- Dijkstra（迪杰斯特拉）最短路算法



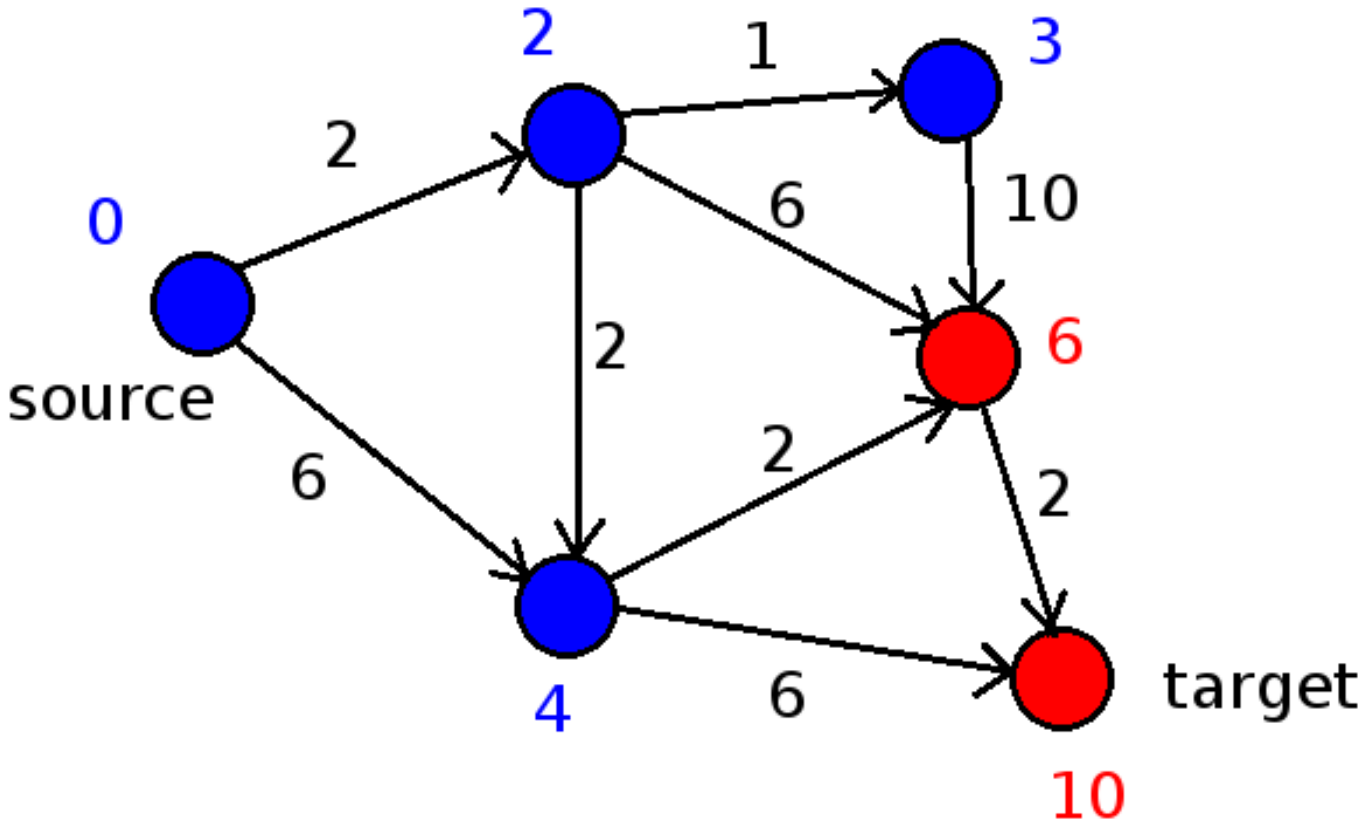
历史 & 基础知识

- Dijkstra（迪杰斯特拉）最短路算法



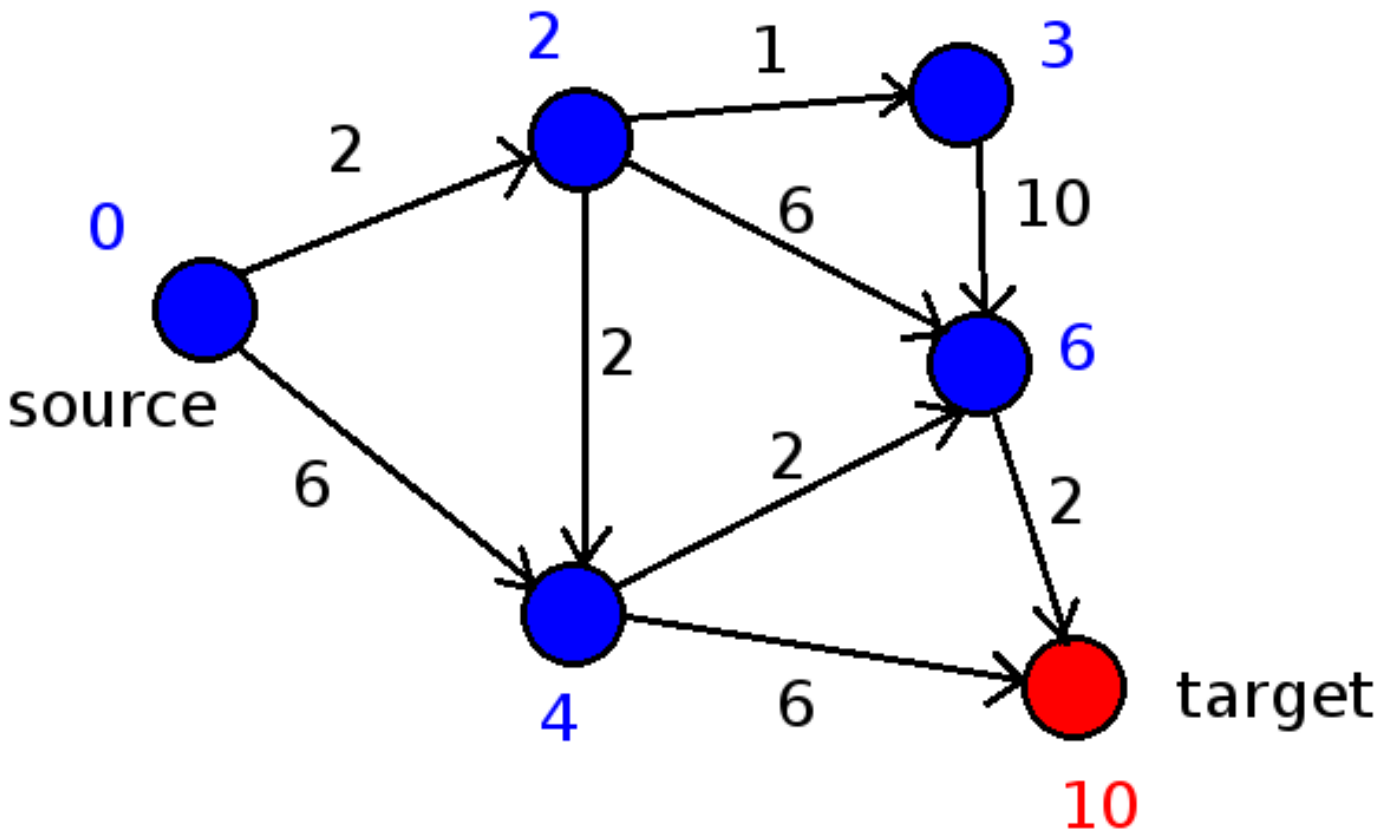
历史 & 基础知识

- Dijkstra（迪杰斯特拉）最短路算法



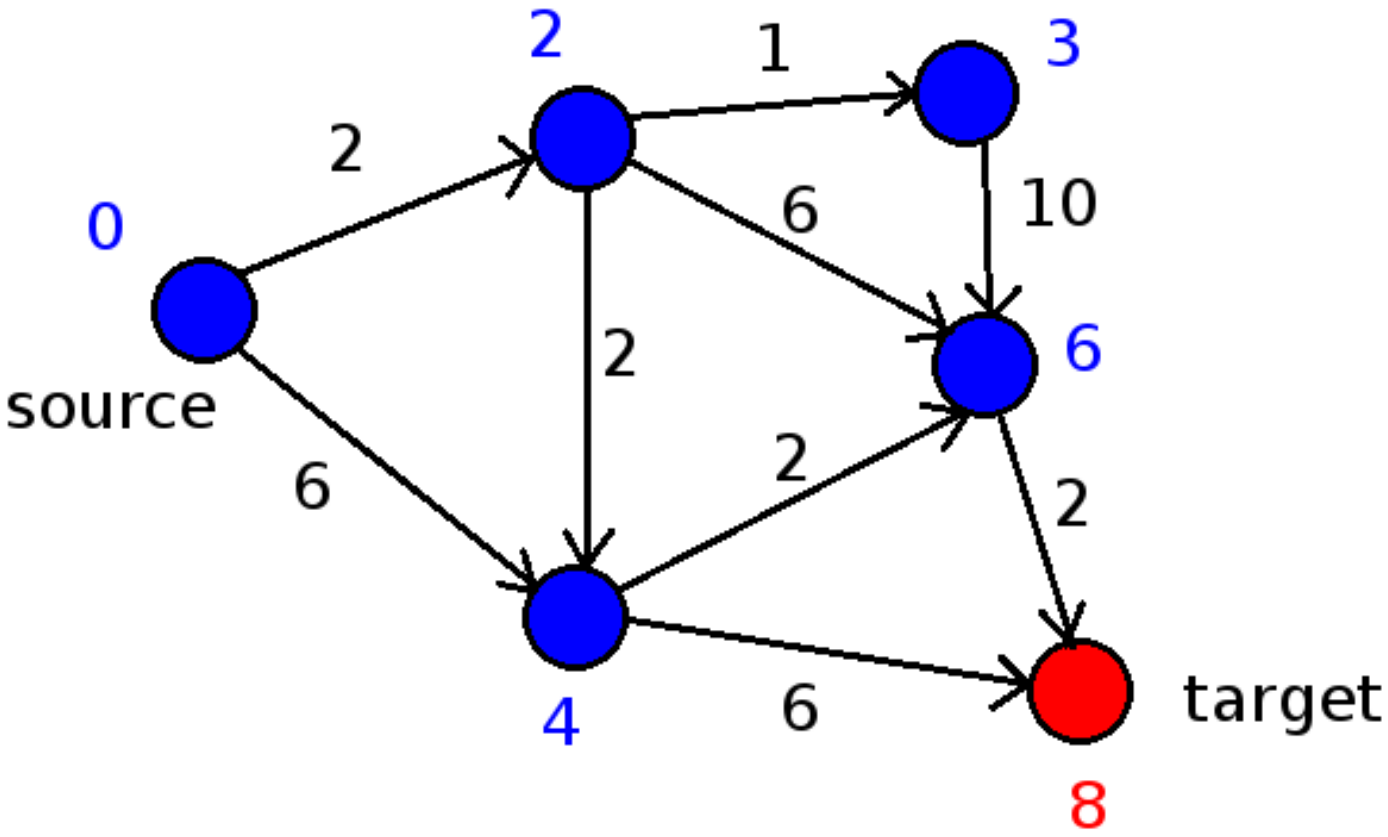
历史 & 基础知识

- Dijkstra（迪杰斯特拉）最短路算法



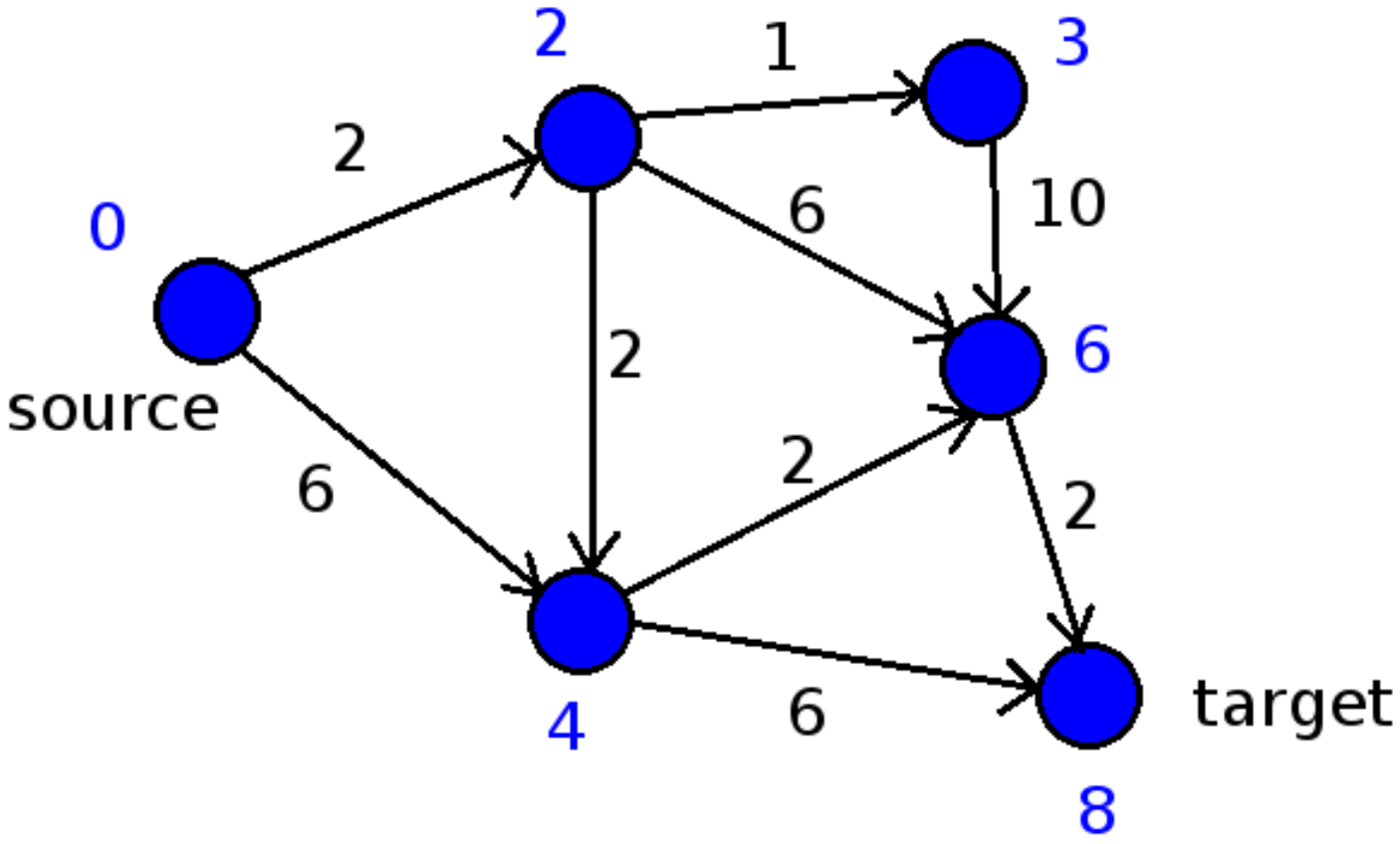
历史 & 基础知识

- Dijkstra（迪杰斯特拉）最短路算法



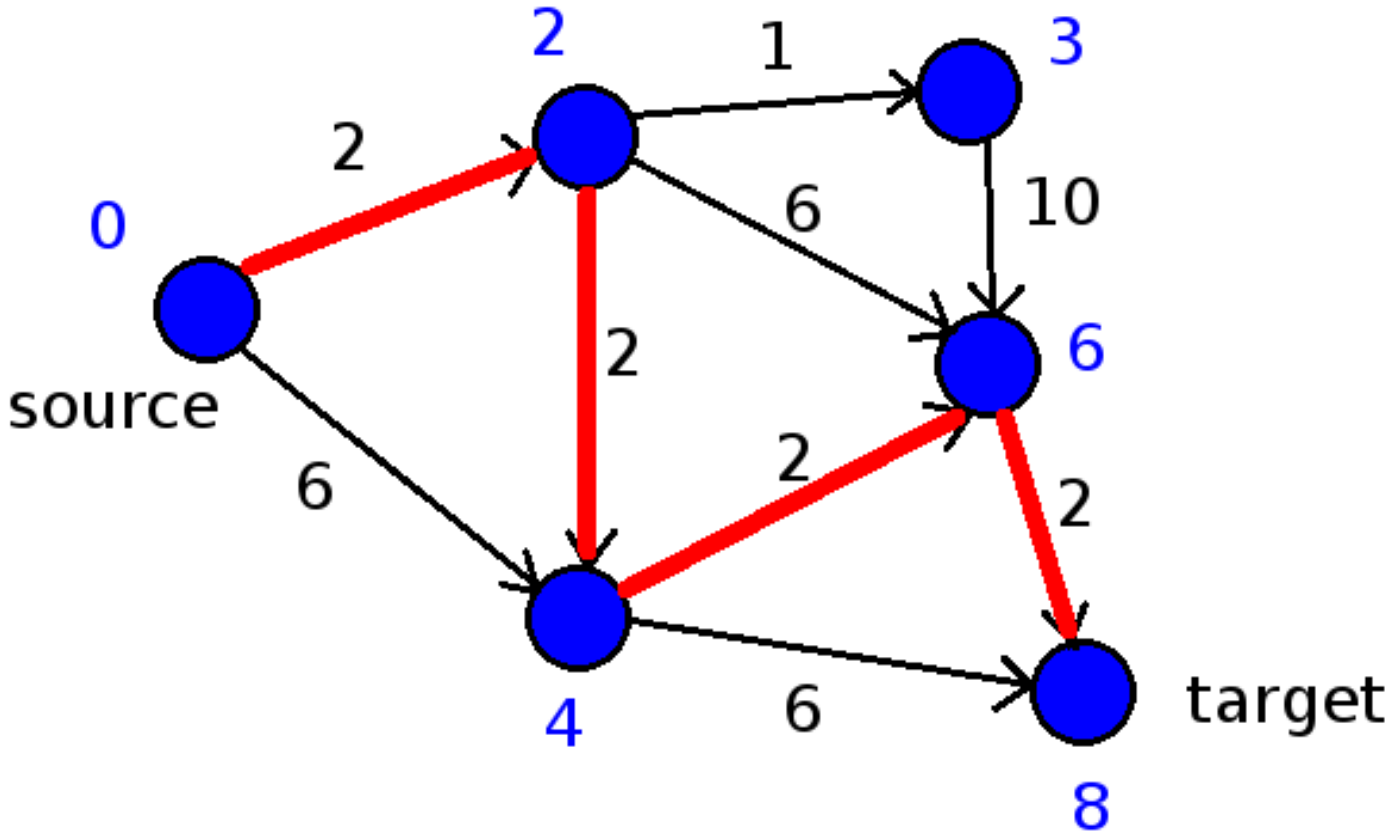
历史 & 基础知识

- Dijkstra（迪杰斯特拉）最短路算法



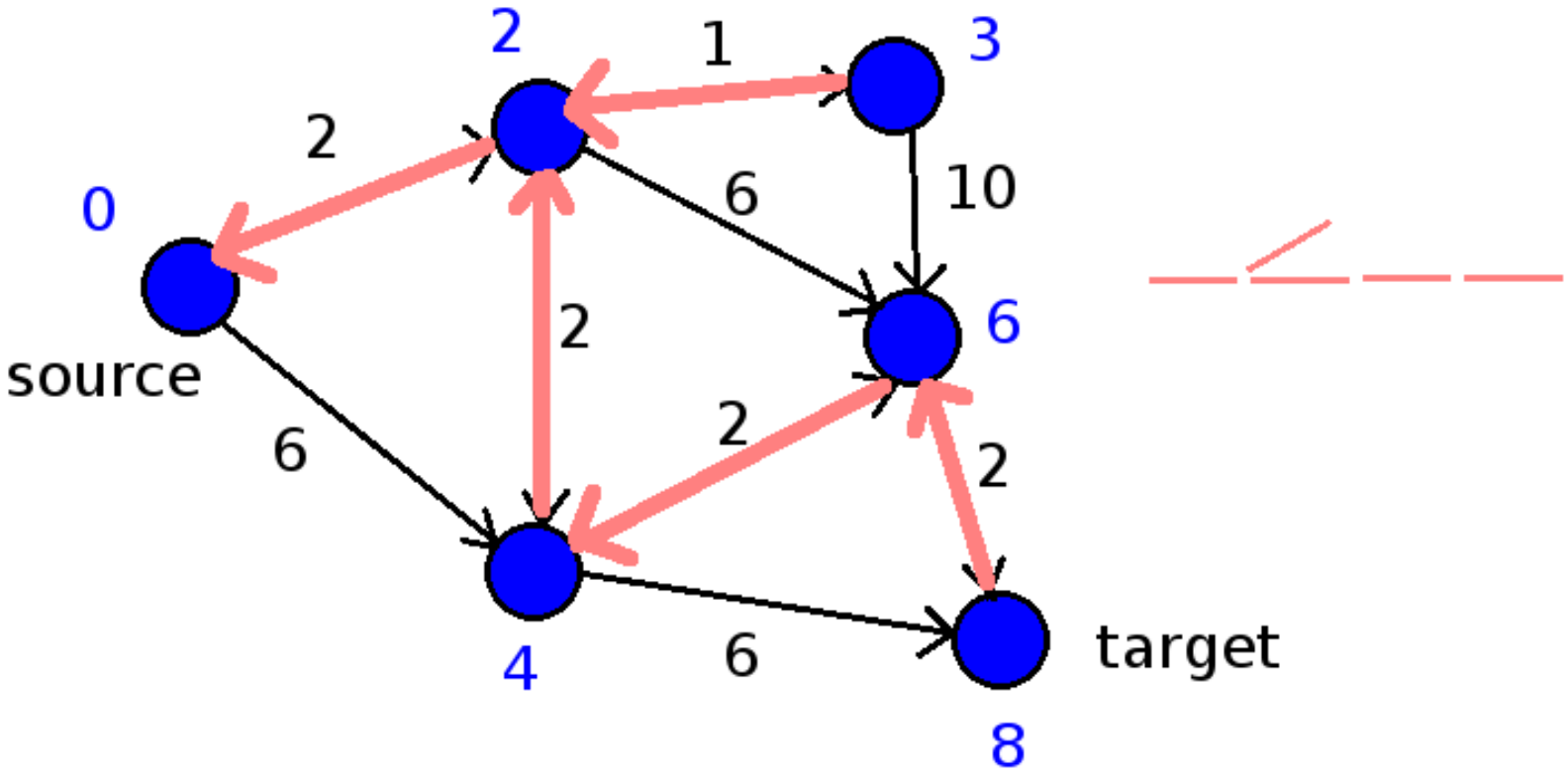
历史 & 基础知识

- Dijkstra（迪杰斯特拉）最短路算法



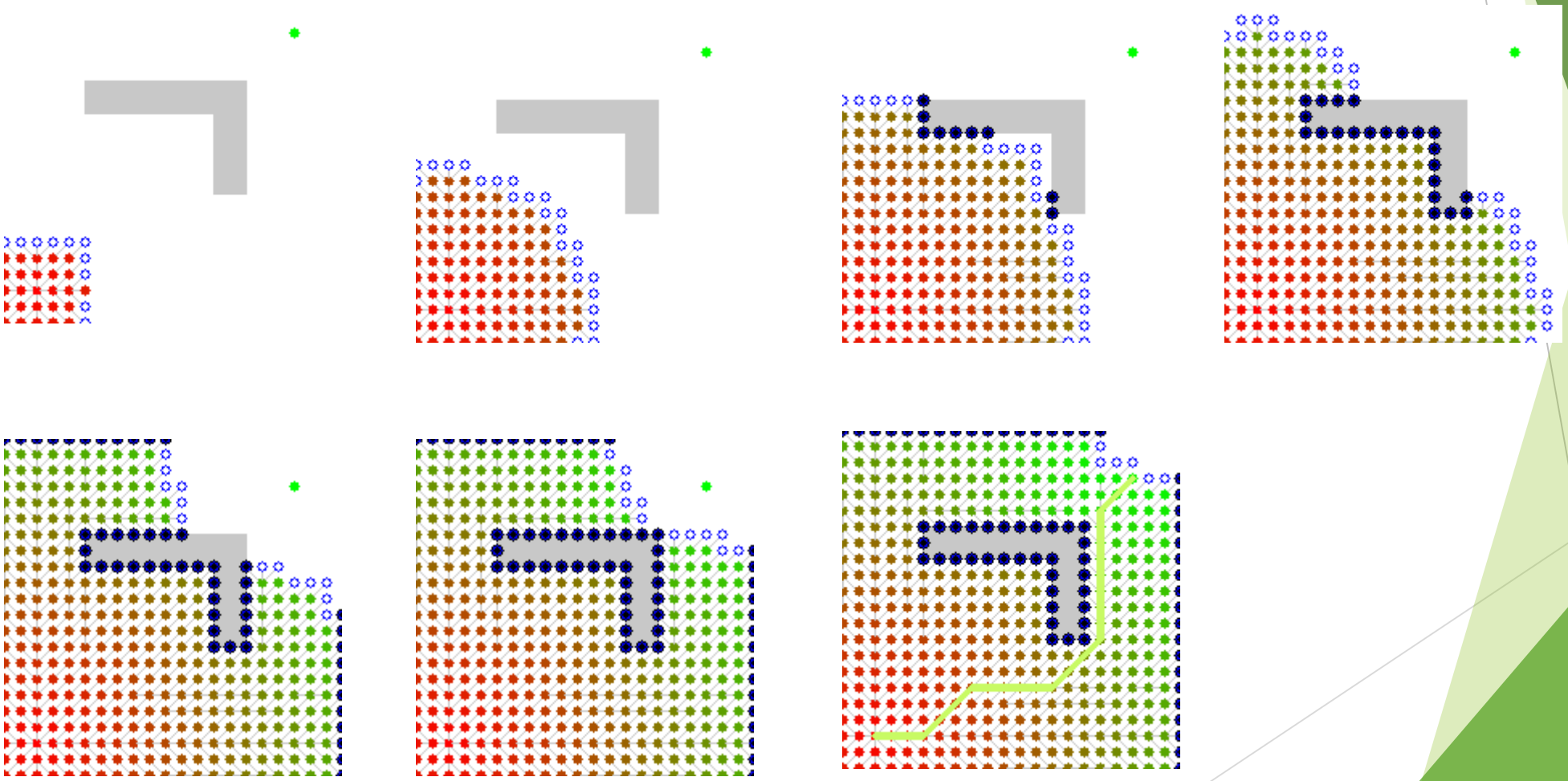
历史 & 基础知识

- Dijkstra（迪杰斯特拉）最短路算法
  - 如果一个点的前继为其父亲节点，则Dijkstra得到一棵最短路树，其中每个节点到根节点都是最短路



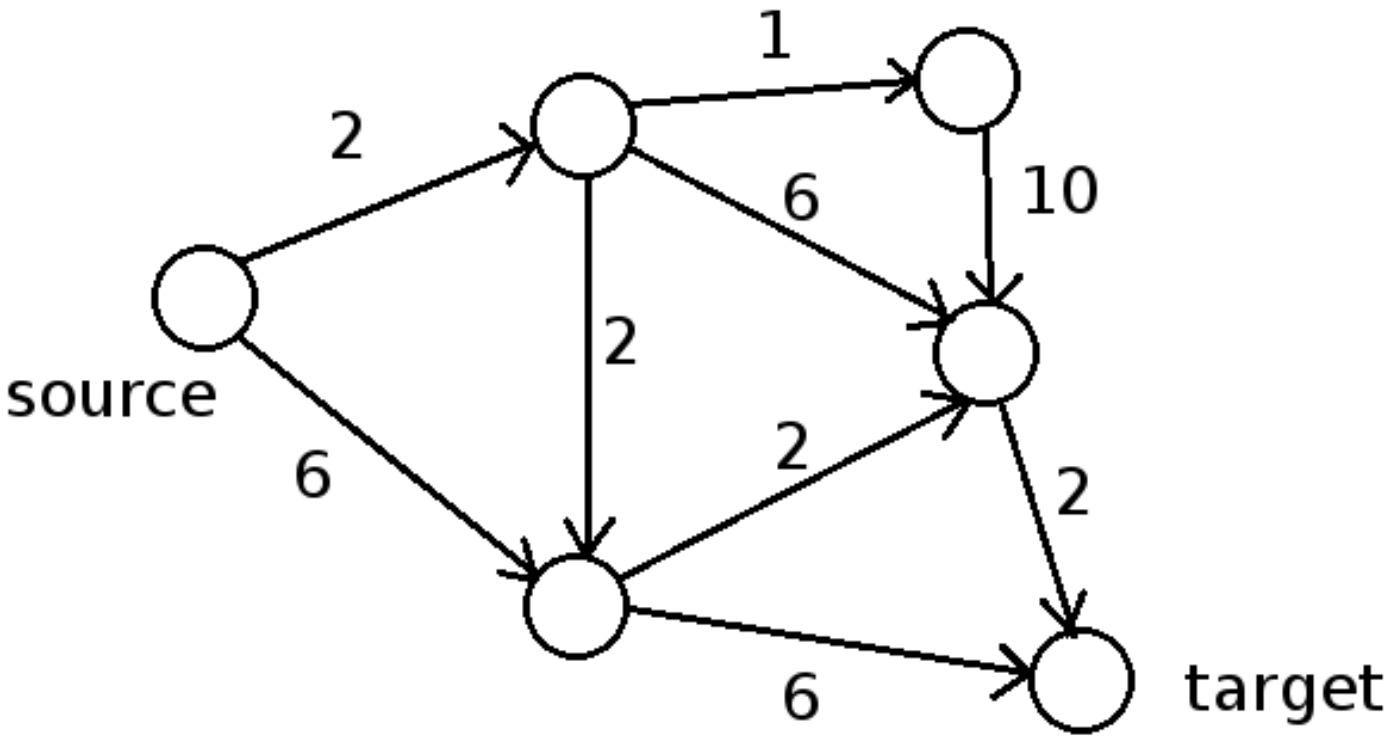
历史 & 基础知识

- Dijkstra（迪杰斯特拉）最短路算法



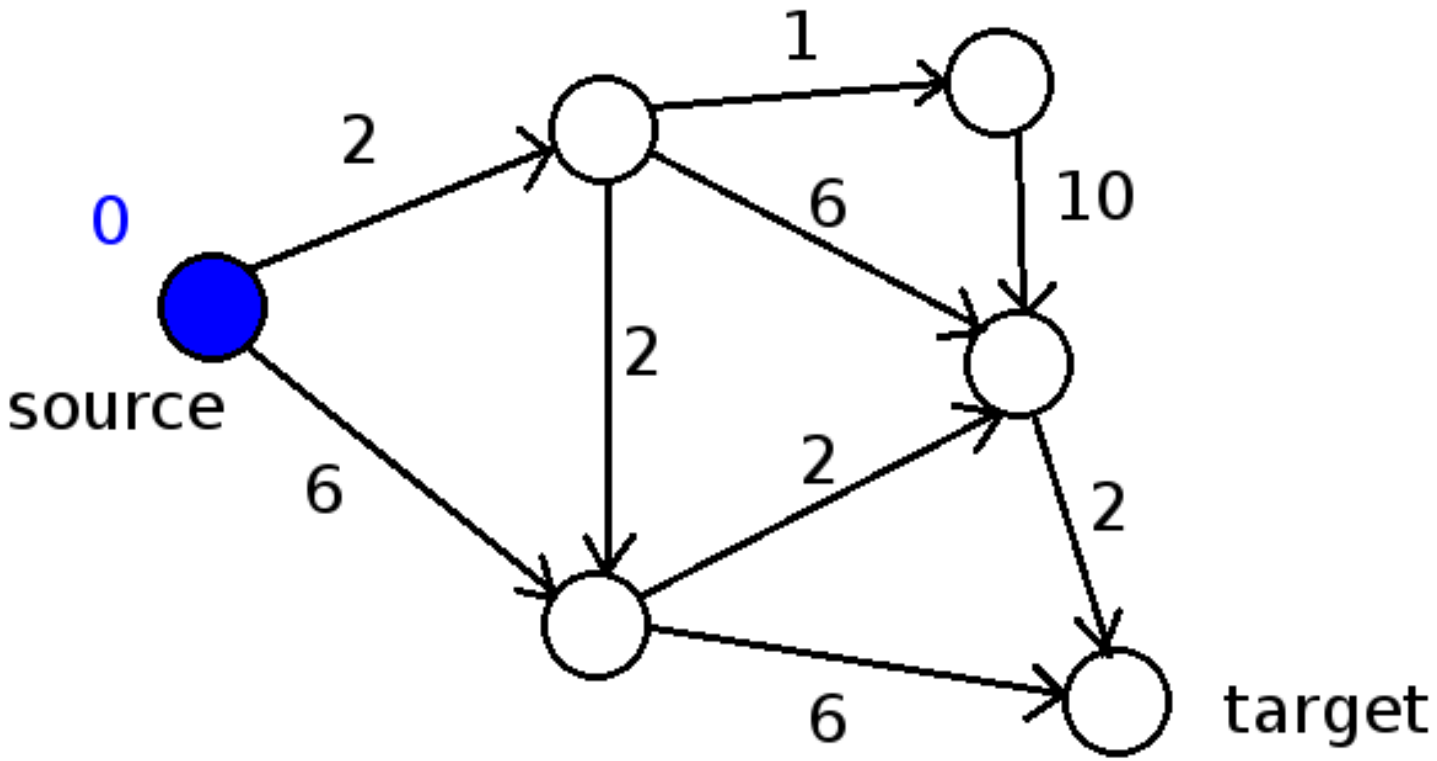
历史 & 基础知识

- Bidirectional Dijkstra （双向迪杰斯特拉）最短路算法



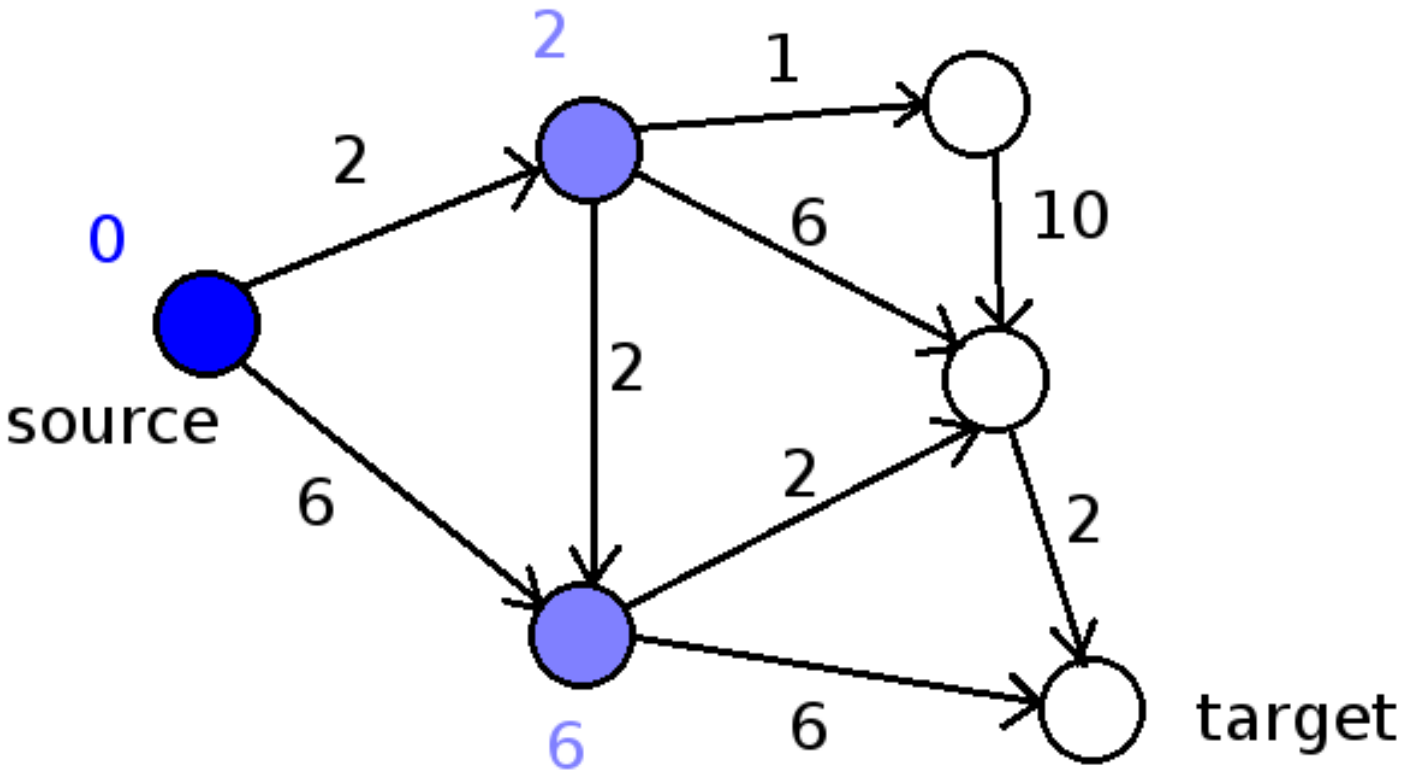
历史 & 基础知识

- Bidirectional Dijkstra （双向迪杰斯特拉）最短路算法



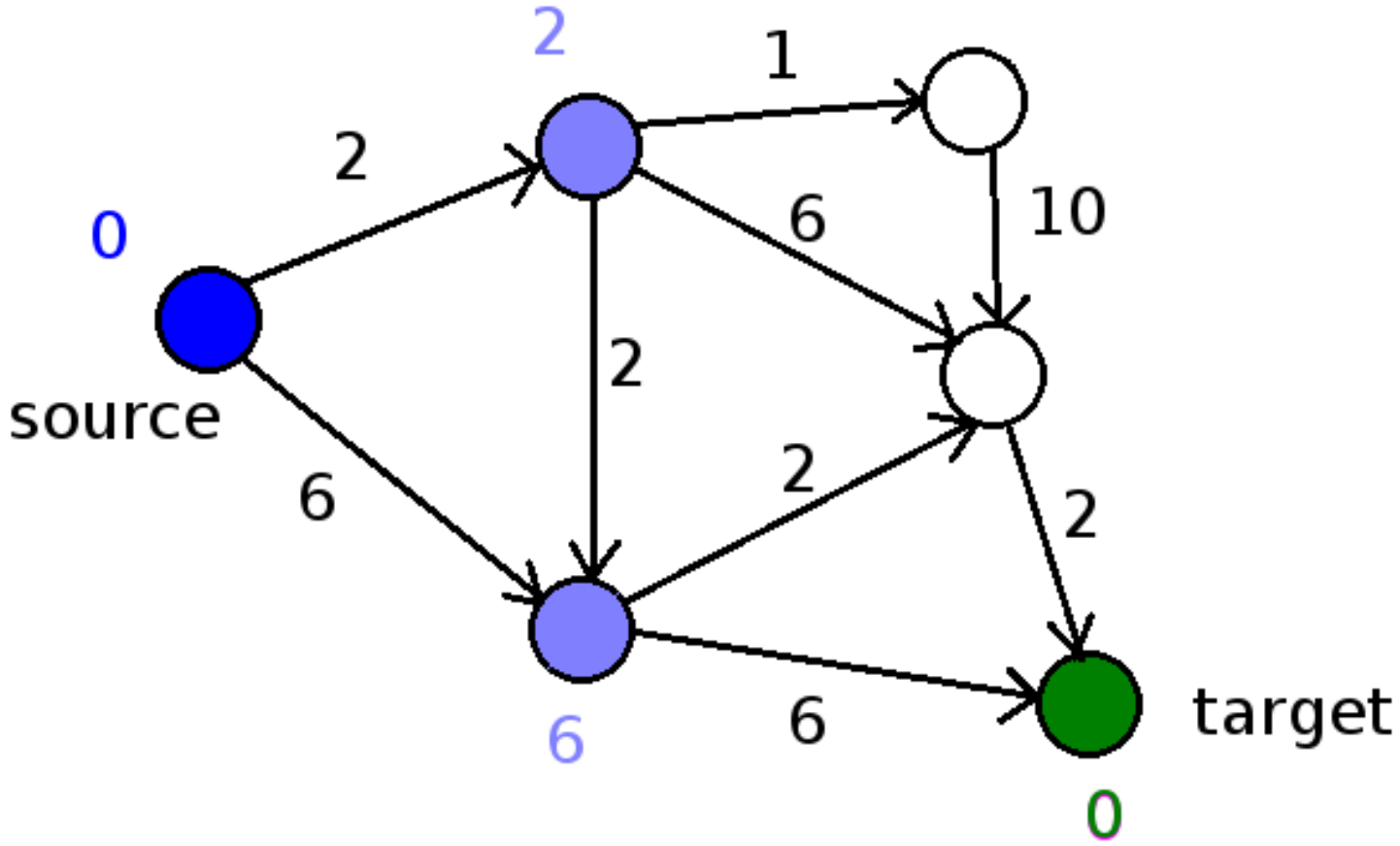
历史 & 基础知识

- Bidirectional Dijkstra （双向迪杰斯特拉）最短路算法



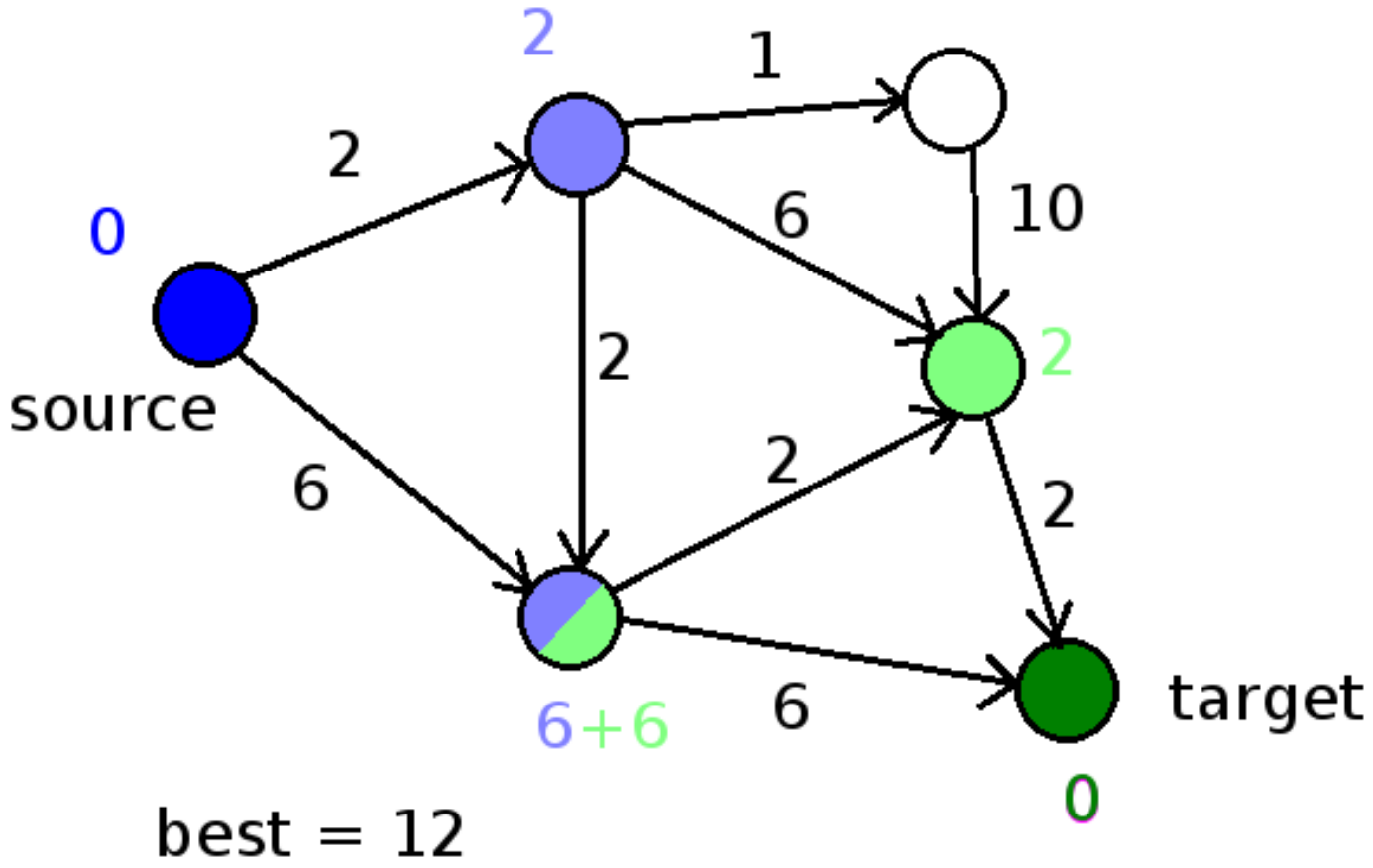
历史 & 基础知识

- Bidirectional Dijkstra （双向迪杰斯特拉）最短路算法



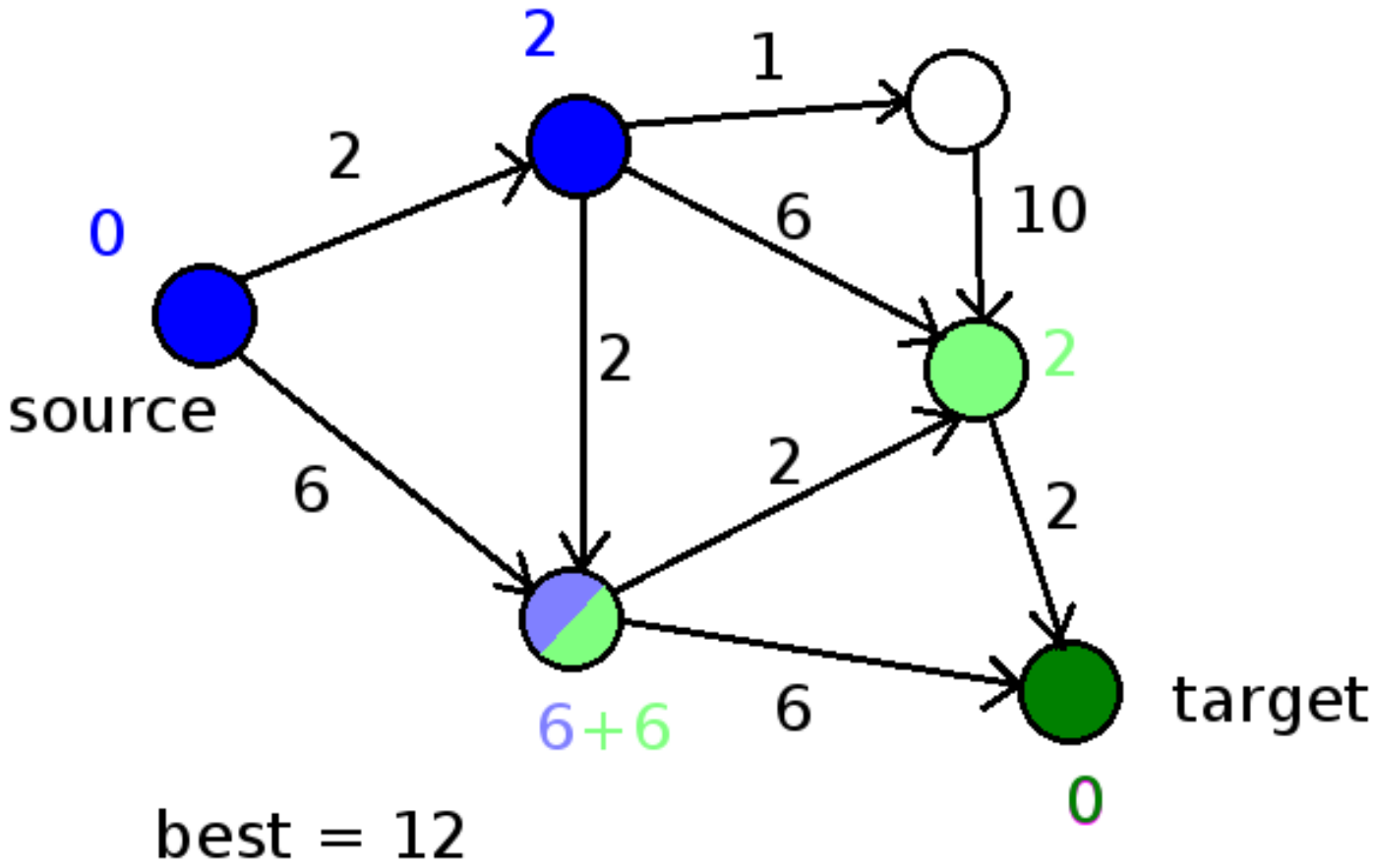
历史 & 基础知识

- Bidirectional Dijkstra （双向迪杰斯特拉）最短路算法



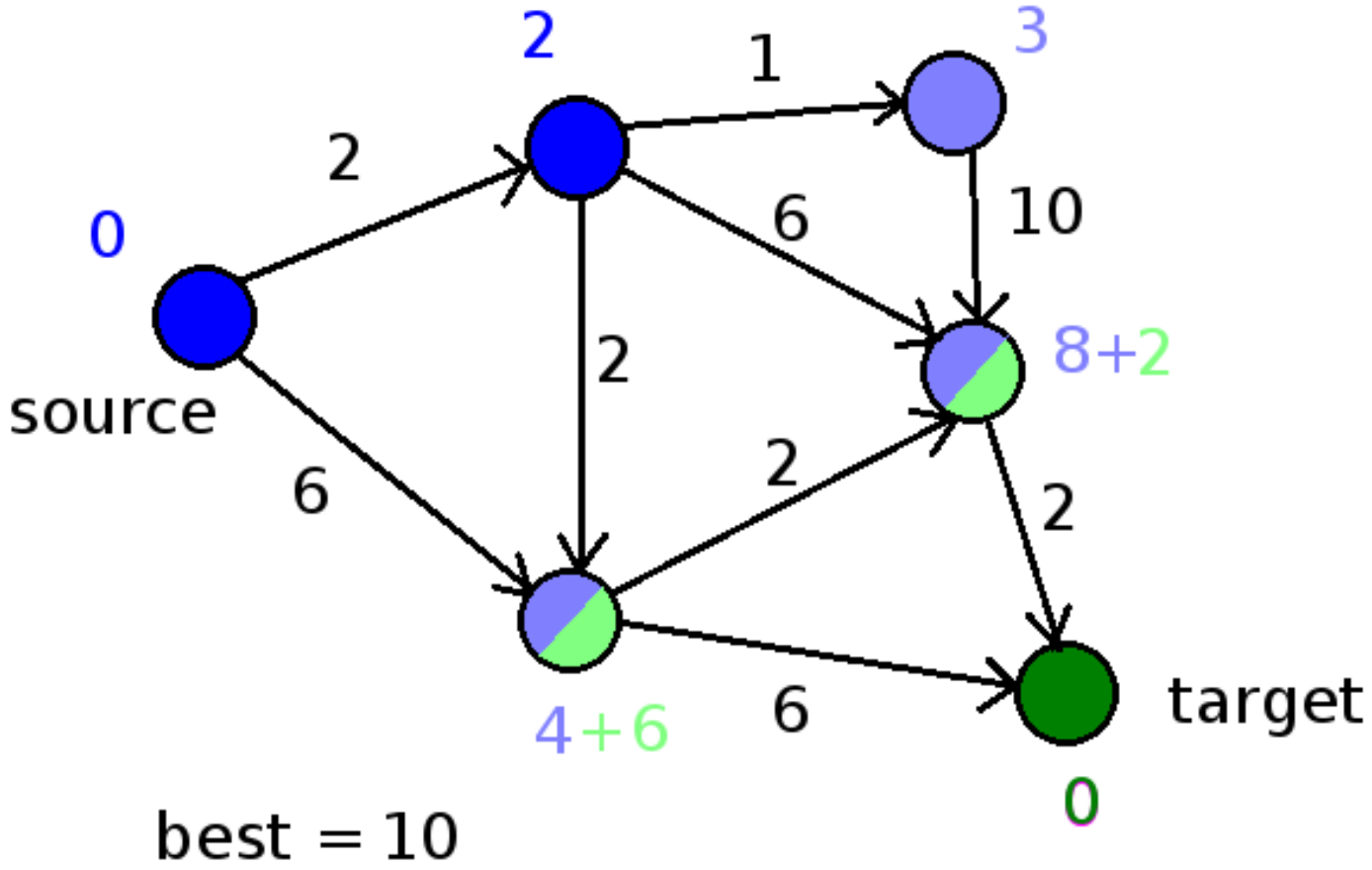
历史 & 基础知识

- Bidirectional Dijkstra （双向迪杰斯特拉）最短路算法



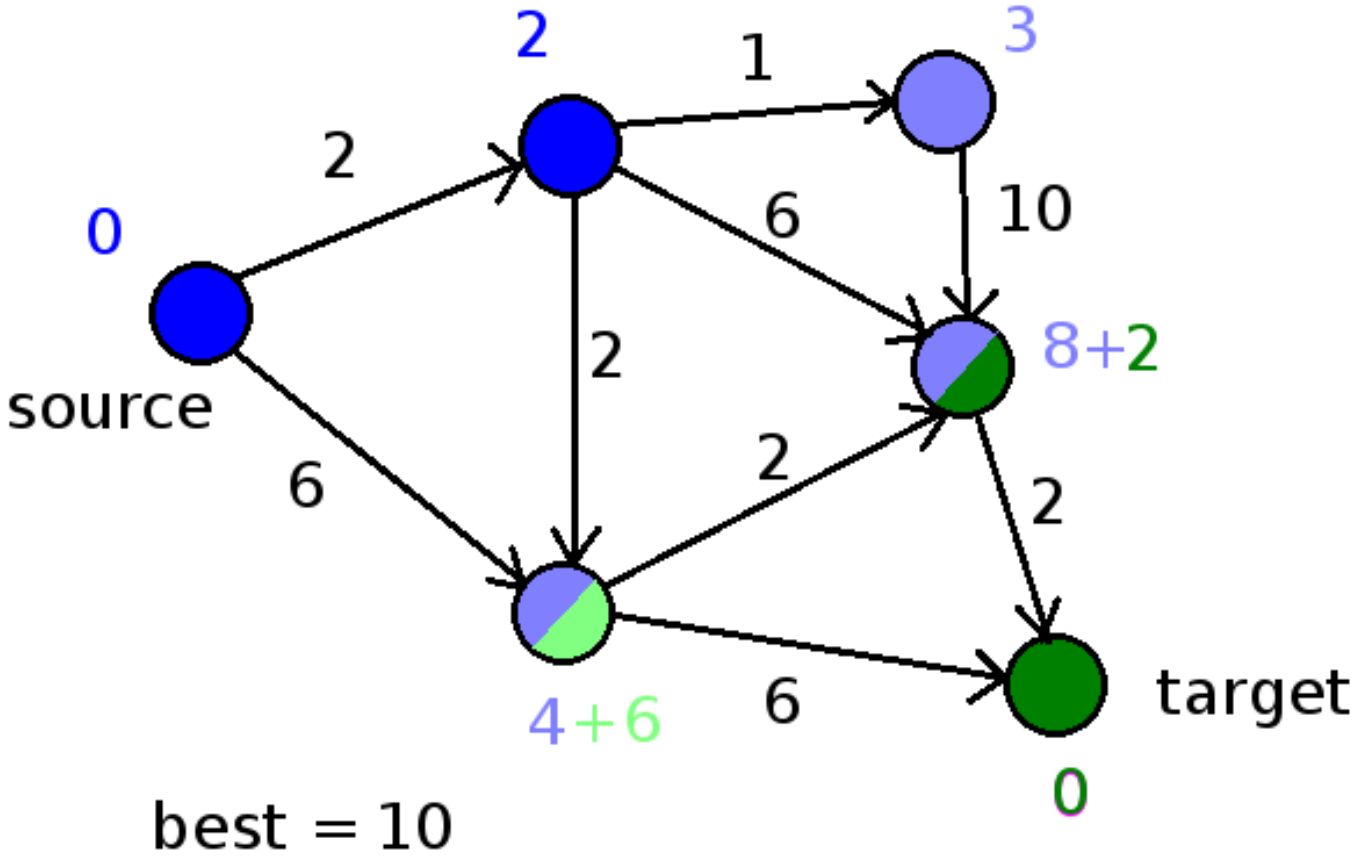
历史 & 基础知识

- Bidirectional Dijkstra （双向迪杰斯特拉）最短路算法



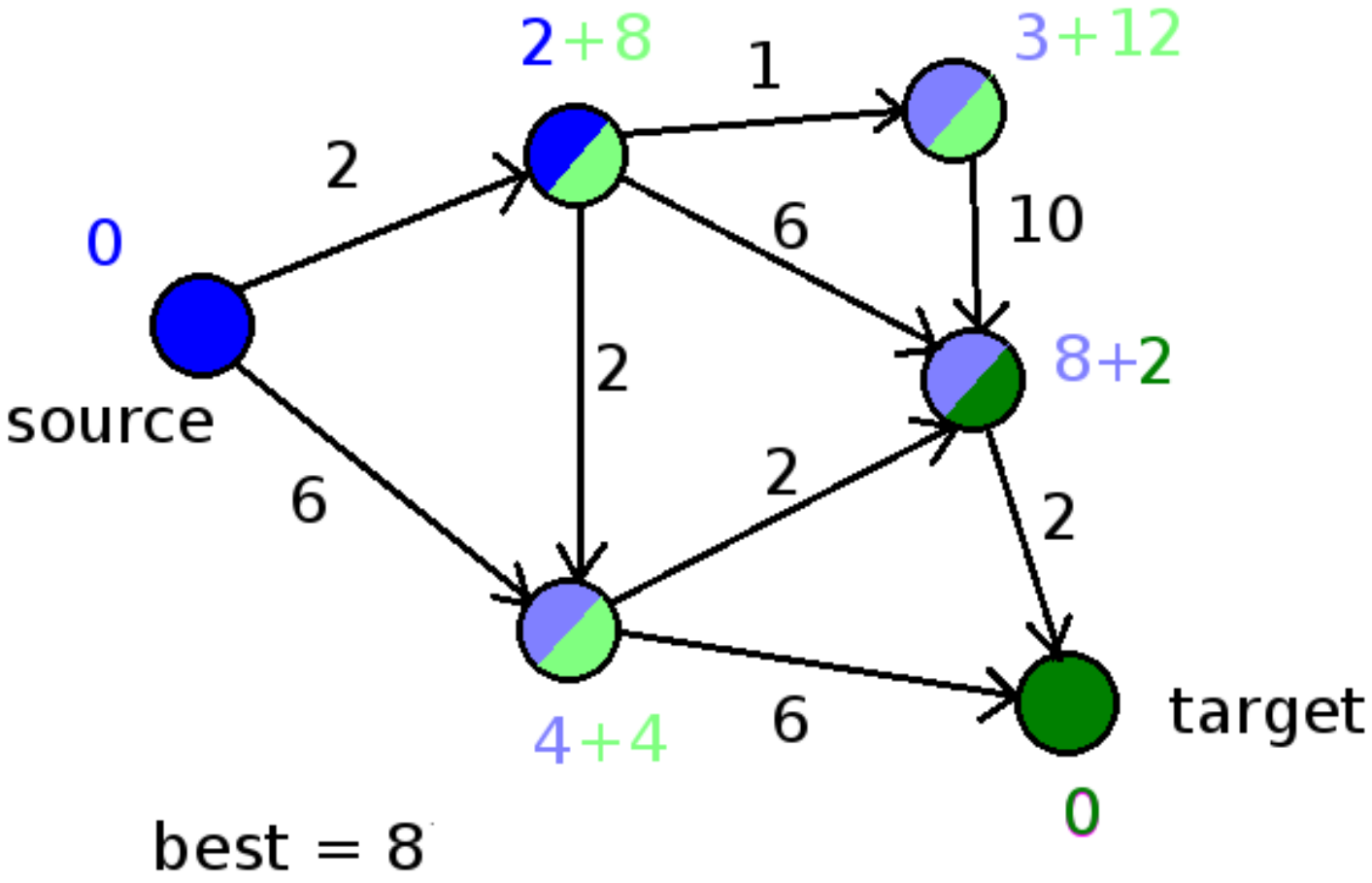
历史 & 基础知识

- Bidirectional Dijkstra （双向迪杰斯特拉）最短路算法



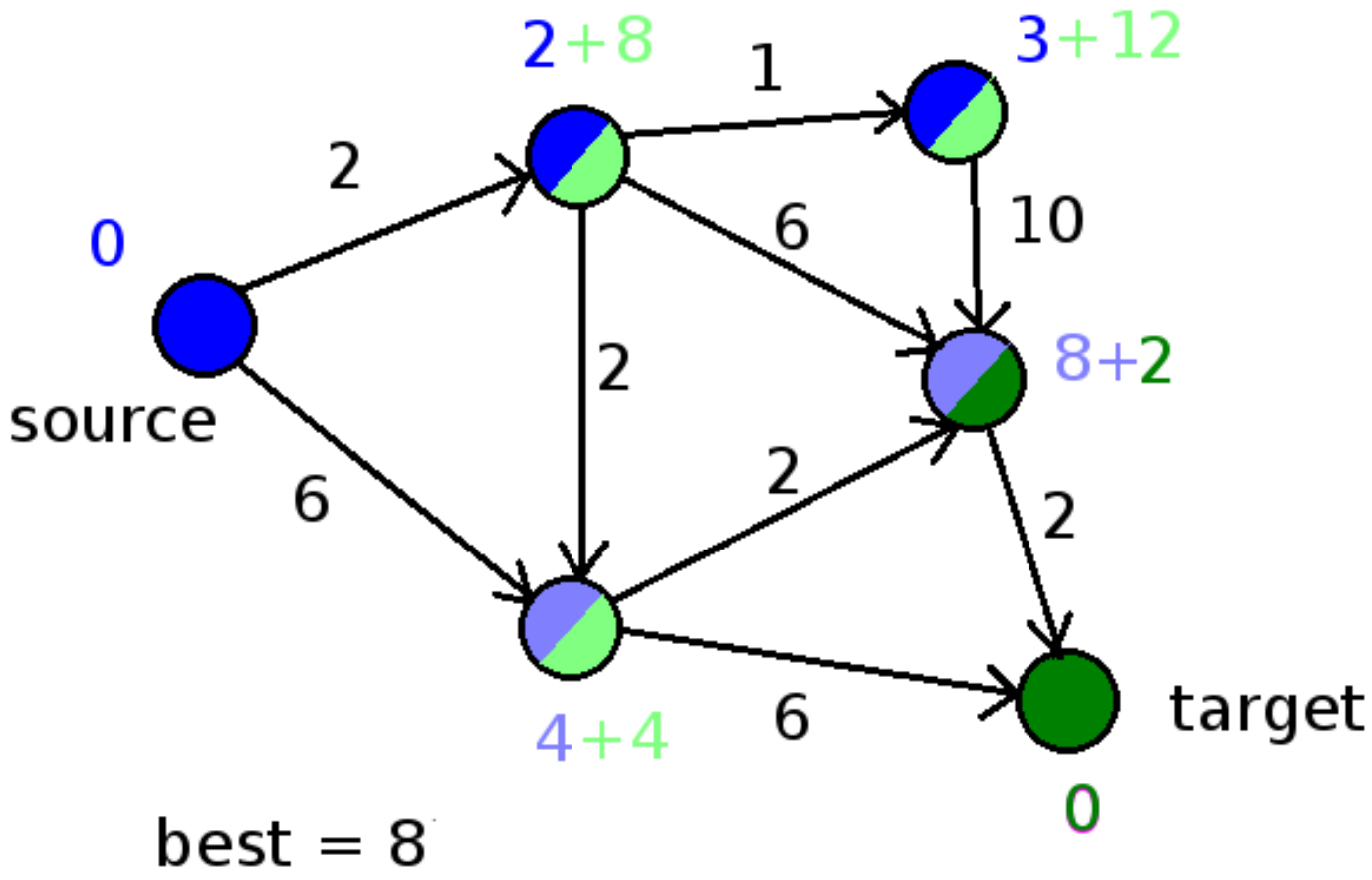
历史 & 基础知识

- Bidirectional Dijkstra （双向迪杰斯特拉）最短路算法



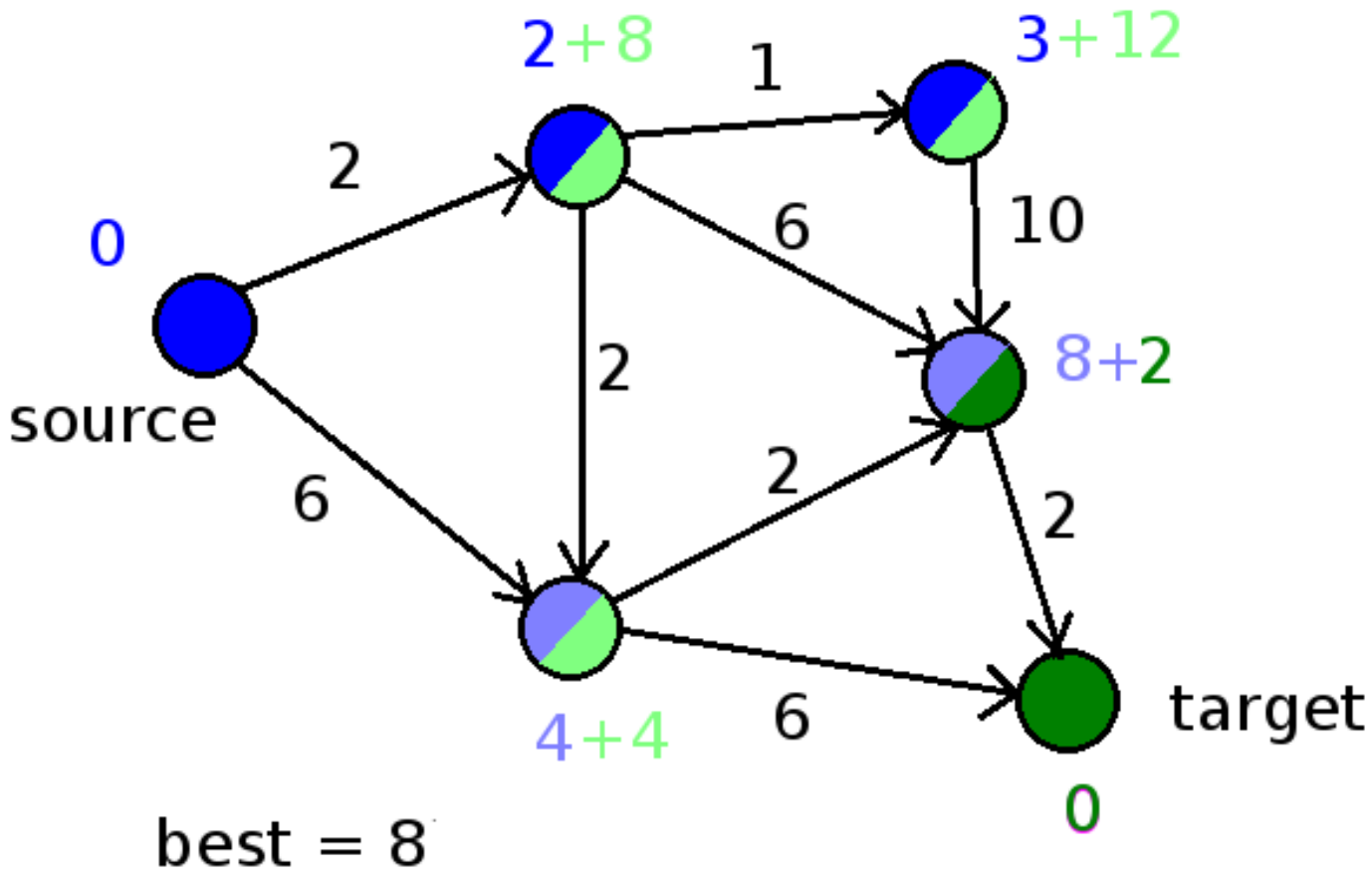
历史 & 基础知识

- Bidirectional Dijkstra （双向迪杰斯特拉）最短路算法



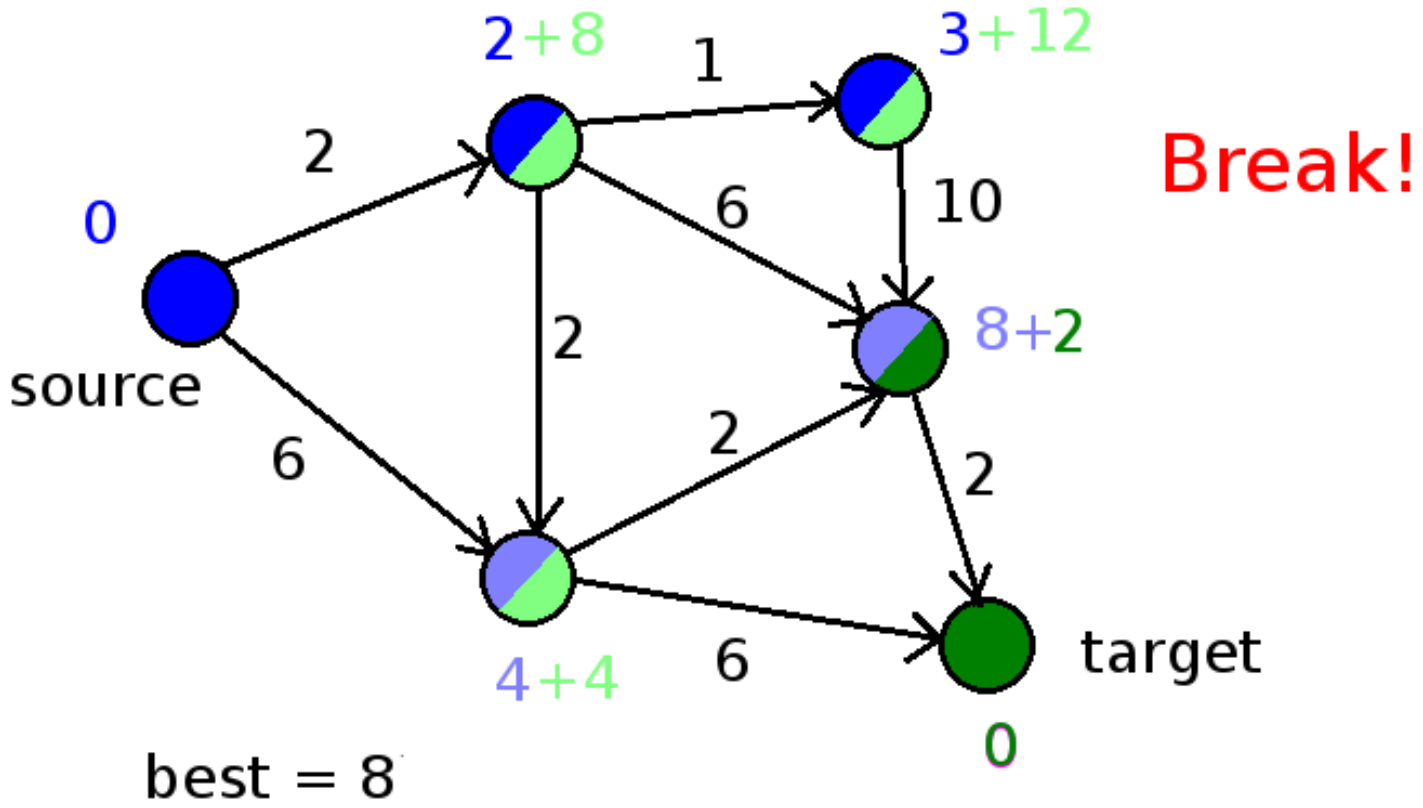
历史 & 基础知识

- Bidirectional Dijkstra （双向迪杰斯特拉）最短路算法



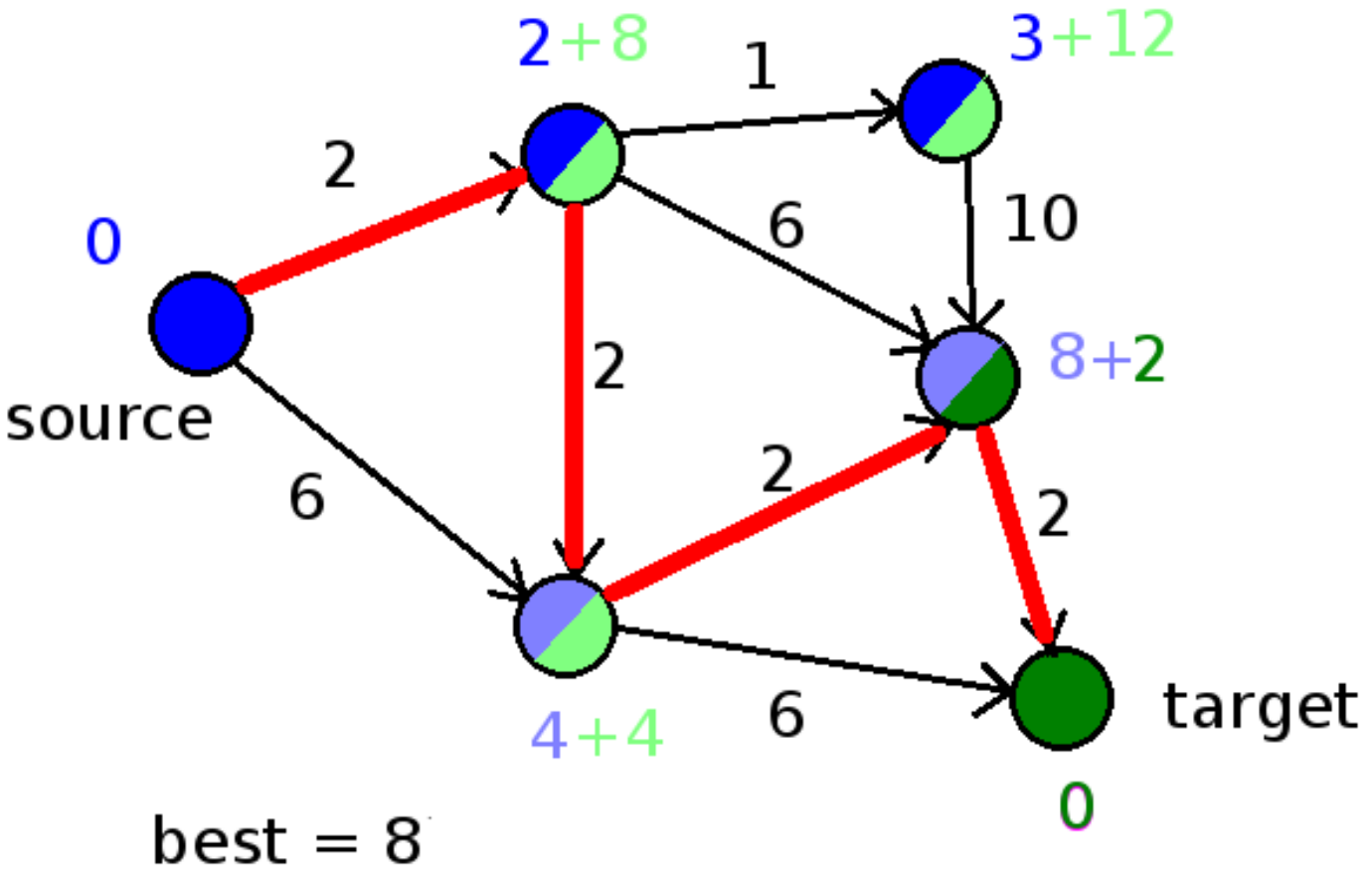
历史 & 基础知识

- Bidirectional Dijkstra （双向迪杰斯特拉）最短路算法



历史 & 基础知识

- Bidirectional Dijkstra （双向迪杰斯特拉）最短路算法



历史 & 基础知识

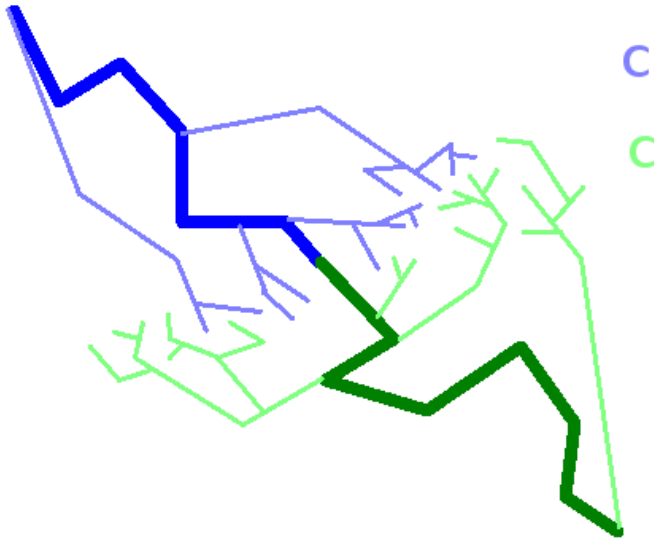
- Bidirectional Dijkstra （双向迪杰斯特拉）最短路算法
  - 两个问题：
    - 什么时候退出？
    - 第一个点被两边都固定的点是否一定在最短路上？

历史 & 基础知识

- Bidirectional Dijkstra （双向迪杰斯特拉）最短路算法
  - 两个问题：
    - 什么时候退出？
      - 一种选择：当前最优路径不大于两边可扩展点的最小和时
    - 第一个点被两边都固定的点是否一定在最短路上？

历史 & 基础知识

- Bidirectional Dijkstra （双向迪杰斯特拉）最短路算法
  - 两个问题：
    - 什么时候退出？
      - 一种选择：当前最优路径不大于两边可扩展点的最小和时
      - 为什么正确？



best =10

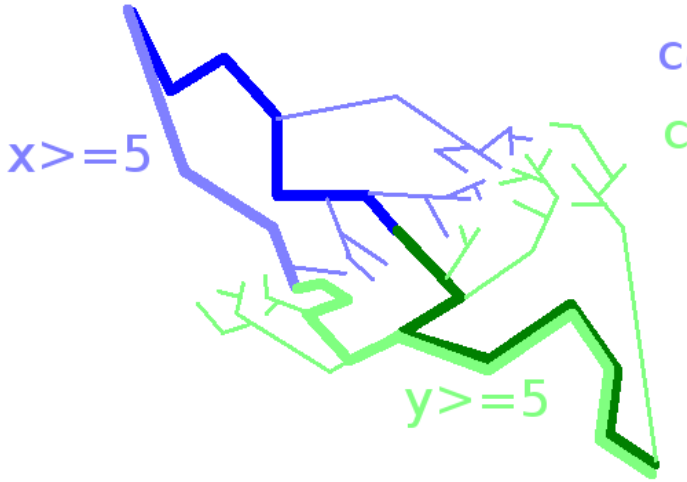
candidates: 9,8,10,7,5,6,8

candidates: 5,6,9,9,10,7,8

5+5>=10

历史 & 基础知识

- Bidirectional Dijkstra （双向迪杰斯特拉）最短路算法
  - 两个问题：
    - 什么时候退出？
      - 一种选择：当前最优路径不大于两边可扩展点的最小和时
      - 为什么正确？
        - 两边继续扩展的长度只会更长，不会更短，所以之后找到的路径只可能更长



best =10

candidates: 9,8,10,7,5,6,8

candidates: 5,6,9,9,10,7,8

5+5>=10

历史 & 基础知识

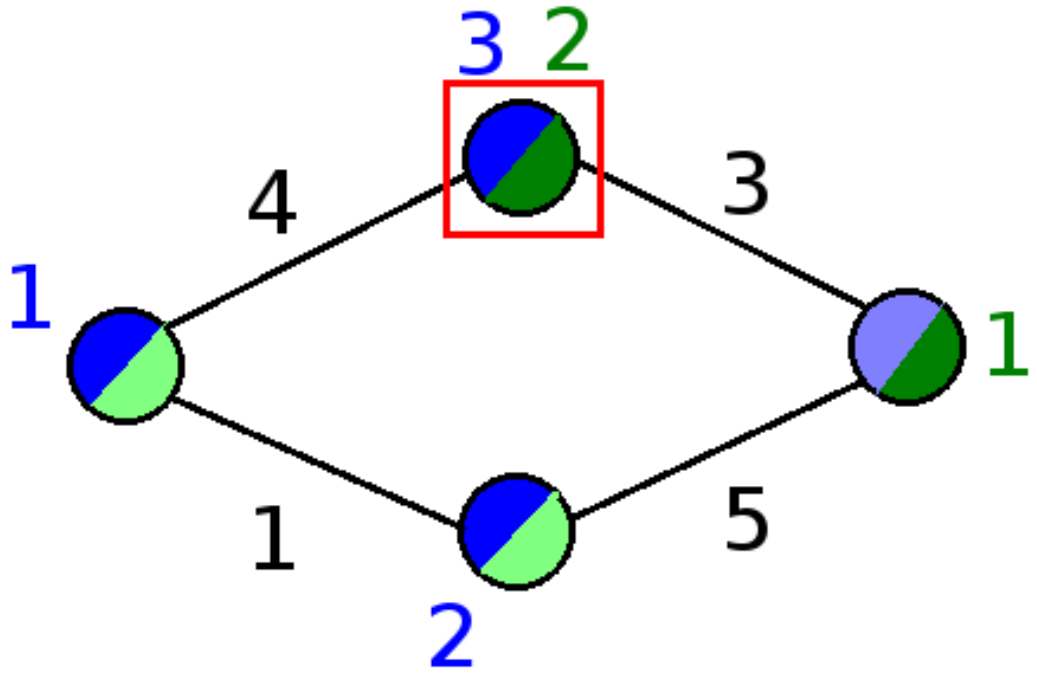
- Bidirectional Dijkstra （双向迪杰斯特拉）最短路算法
  - 两个问题：
    - 第一个点被两边都固定的点是否一定在最短路上？

历史 & 基础知识

- Bidirectional Dijkstra （双向迪杰斯特拉）最短路算法
  - 两个问题：
    - 第一个点被两边都固定的点是否一定在最短路上？
      - 不一定
      - 为什么？

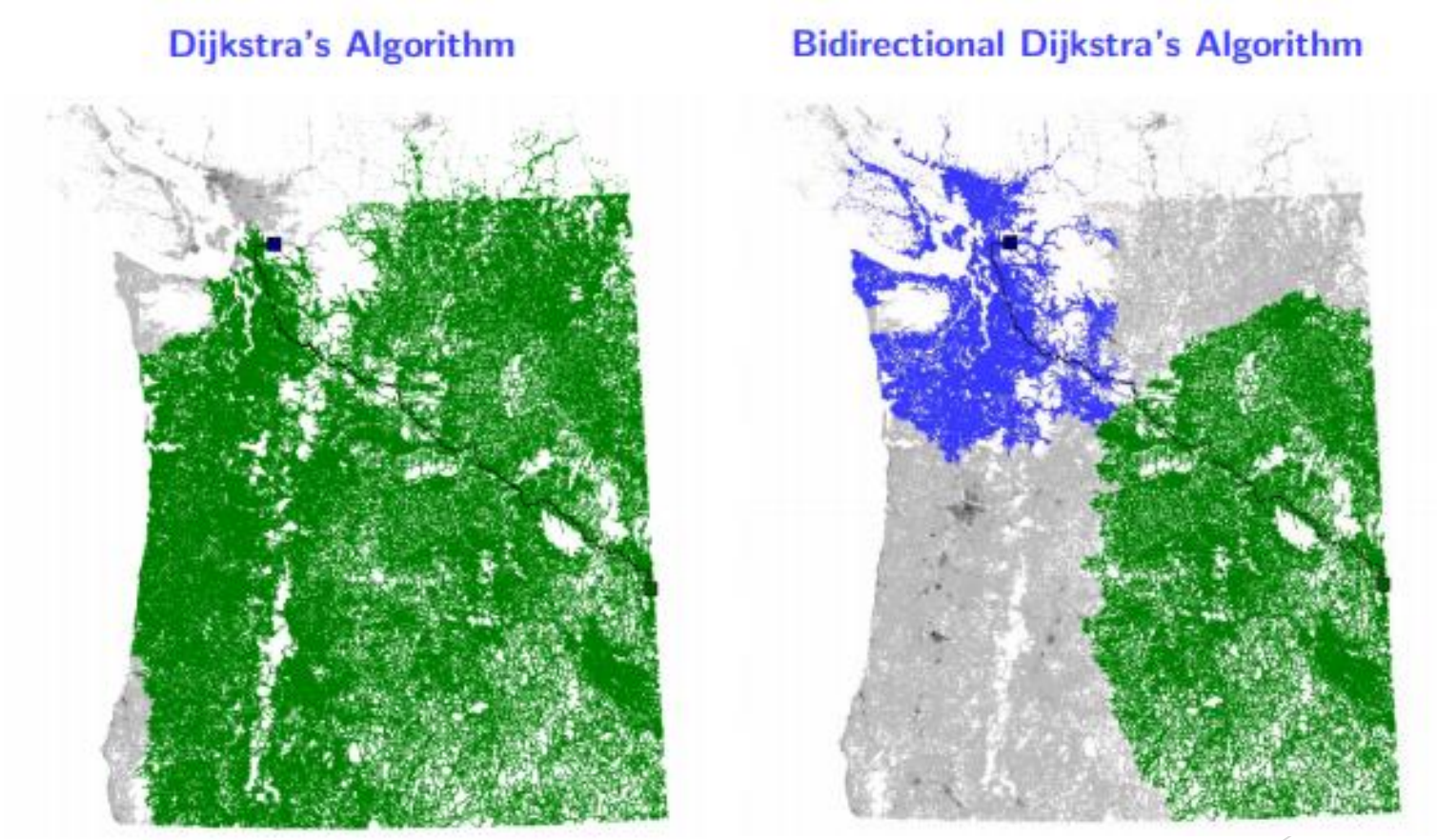
历史 & 基础知识

- Bidirectional Dijkstra （双向迪杰斯特拉）最短路算法
  - 两个问题：
    - 第一个点被两边都固定的点是否一定在最短路上？
      - 不一定
      - 为什么？



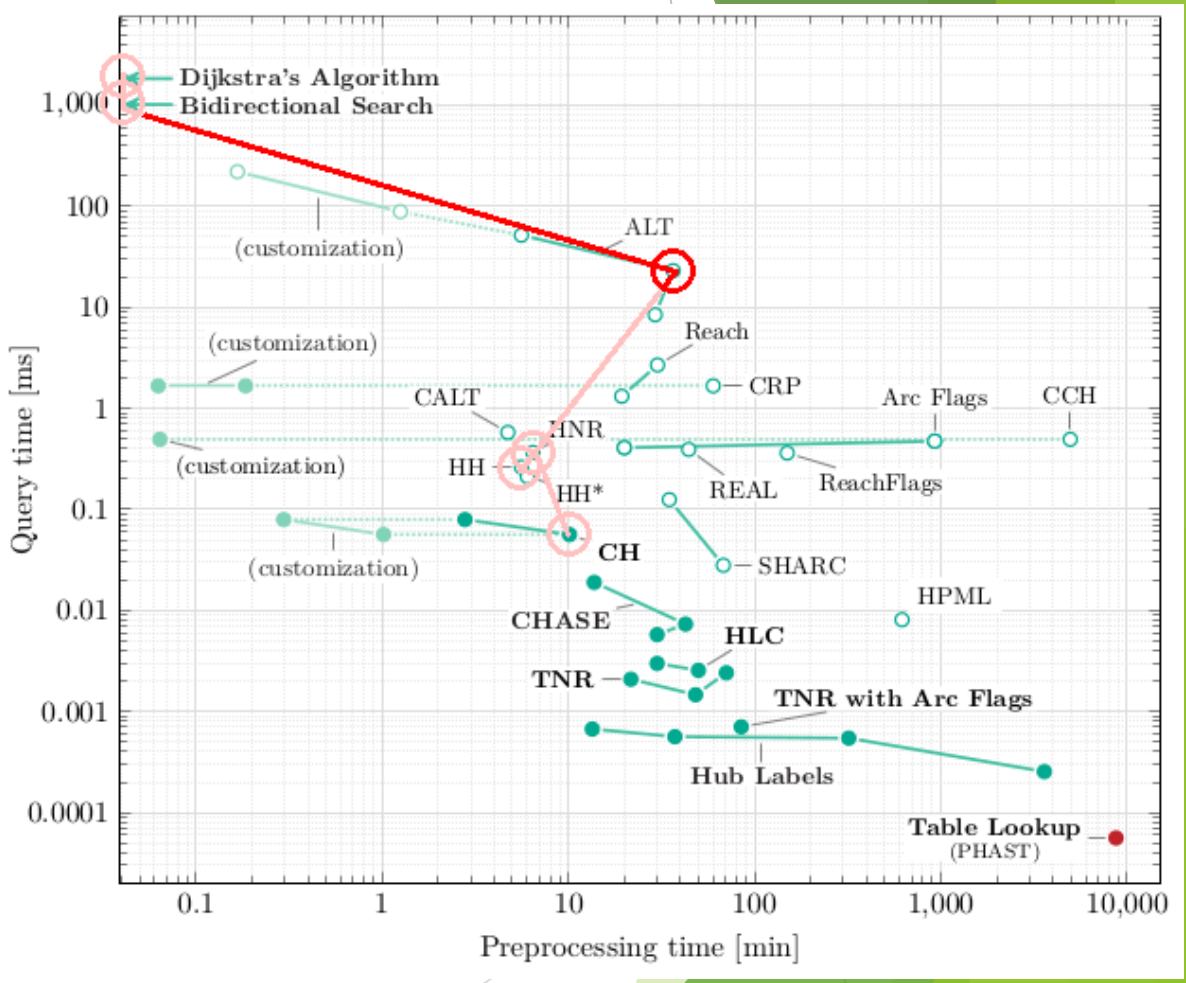
历史 & 基础知识

- Bidirectional Dijkstra （双向迪杰斯特拉）最短路算法
  - 与Dijkstra算法的对比
    - 如果大致成圆形扩展，则速度约快一倍



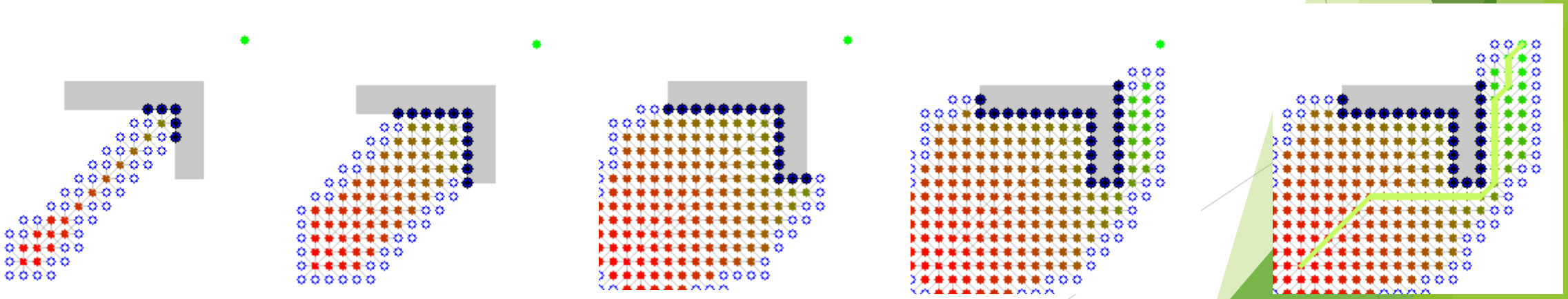
历史 & 基础知识

- [1959] Dijkstra最短路算法
- [~1968] Heuristic A*搜索算法
  - 结合实际信息
- [1991, 2002] Heuristic Hierarchical Approaches
- [2005] Highway Hierarchies (HH)
- [2005] Highway-Node Routing (HNR)
- [2008] Contraction Hierarchies (CH)



历史 & 基础知识

- Heuristic A*（启发式A*）搜索算法
  - 思想：增加对未来路径的预估，保证大致方向的正确，从而减小搜索空间大小
  - 具体方法：类似Dijkstra，但每次取f(x) = g(x) + h(x)最小的点
    - g(x)为起点到点x的实际最短距离
    - h(x)为预估点x到终点的最短距离，需要保证h(x)<=实际的最短距离
    - h(x) = 0即为Dijkstra算法
    - 实际情况中可取h(x)为x到终点的几何距离



历史 & 基础知识

- ALT (A*, Landmarks and Triangle inequality) 算法
  - 思想：预处理一些重要的点和所有点之间的距离，利用三角不等式更好地估计h(x)
  - 具体方法：
    - dist(x, t) >= dist(x, L) - dist(t, L)
    - dist(x, t) >= dist(L, t) - dist(L, x)

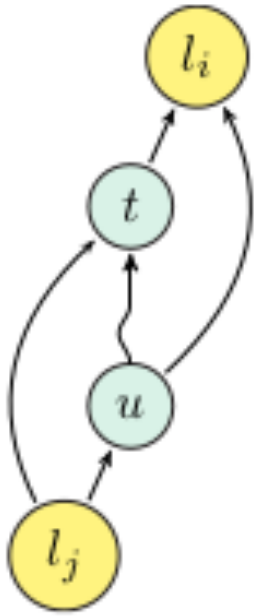
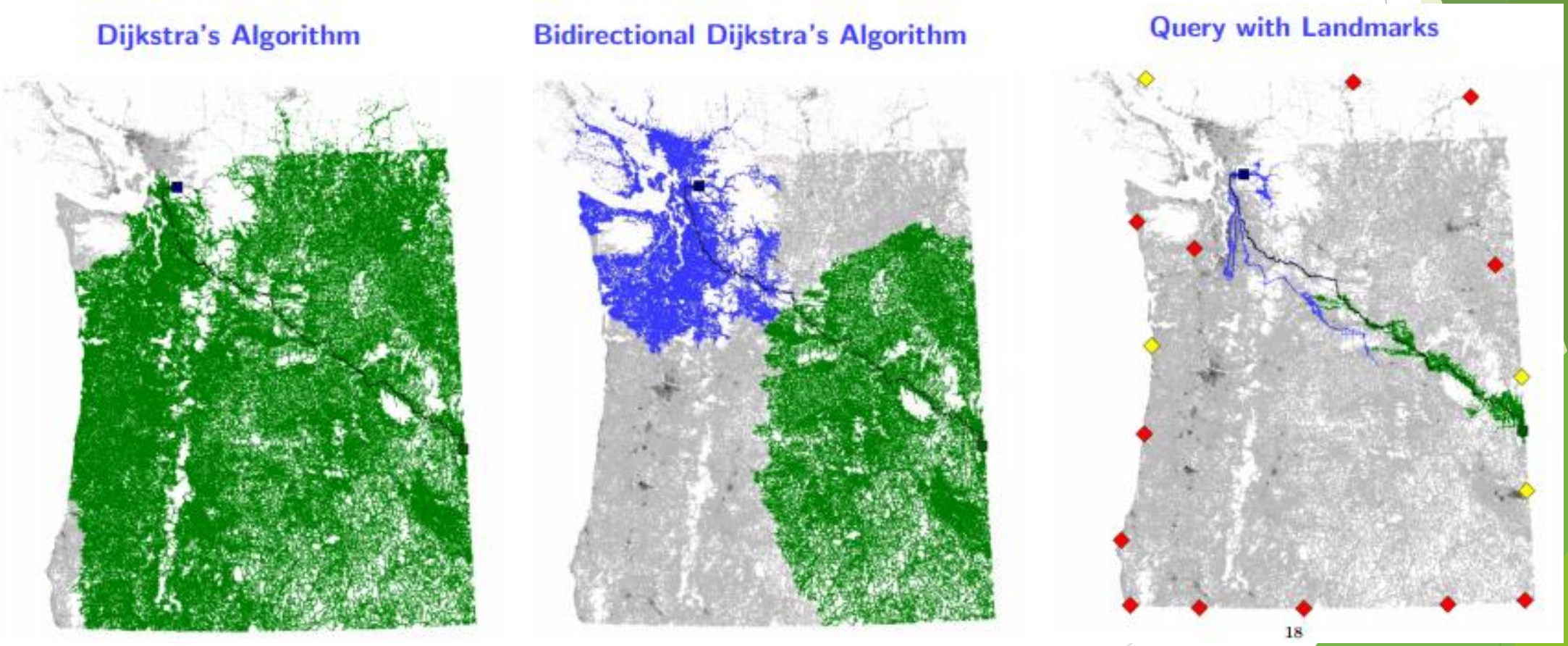


Figure 2. Triangle inequalities for ALT.

历史 & 基础知识

- ALT (A*, Landmarks and Triangle inequality) 算法
  - 和Dijkstra，bidirectional Dijkstra算法的对比

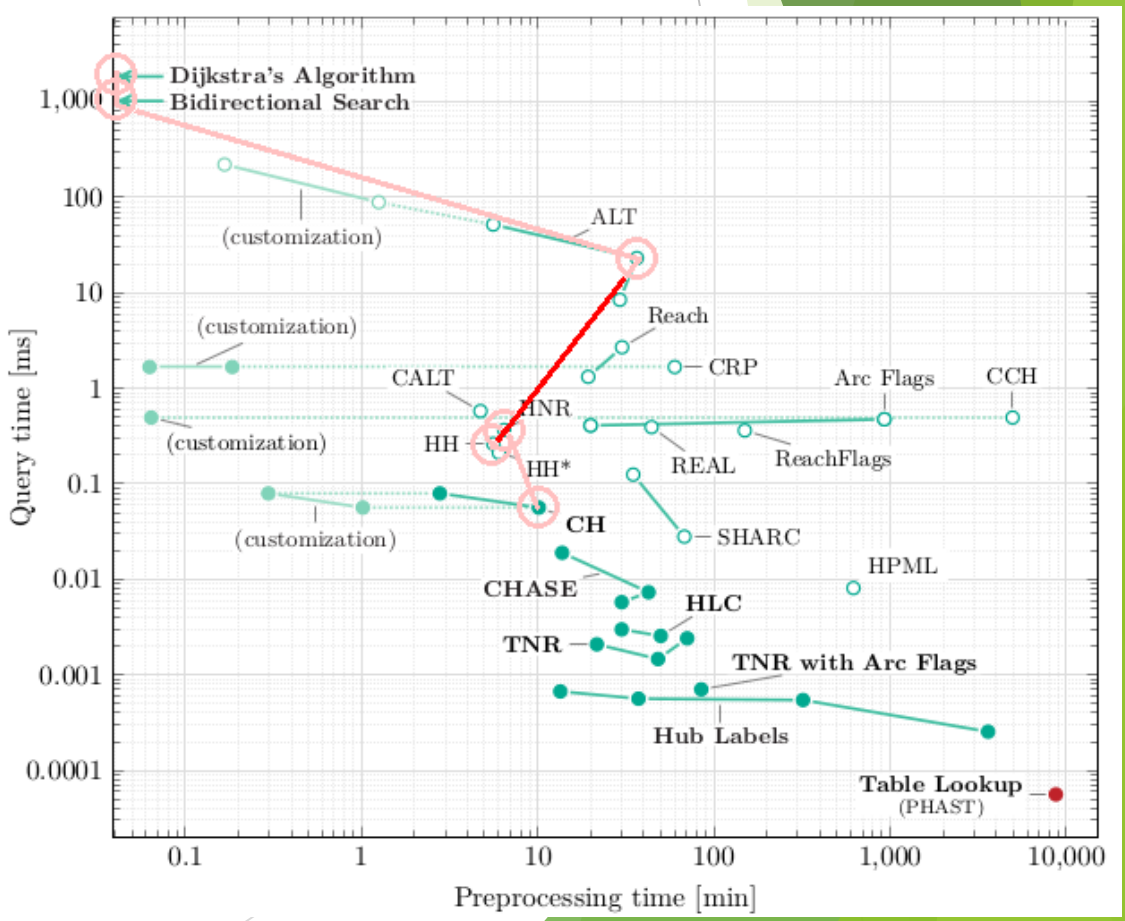


历史 & 基础知识

- 目前为止的算法都很难满足实际情况的需求

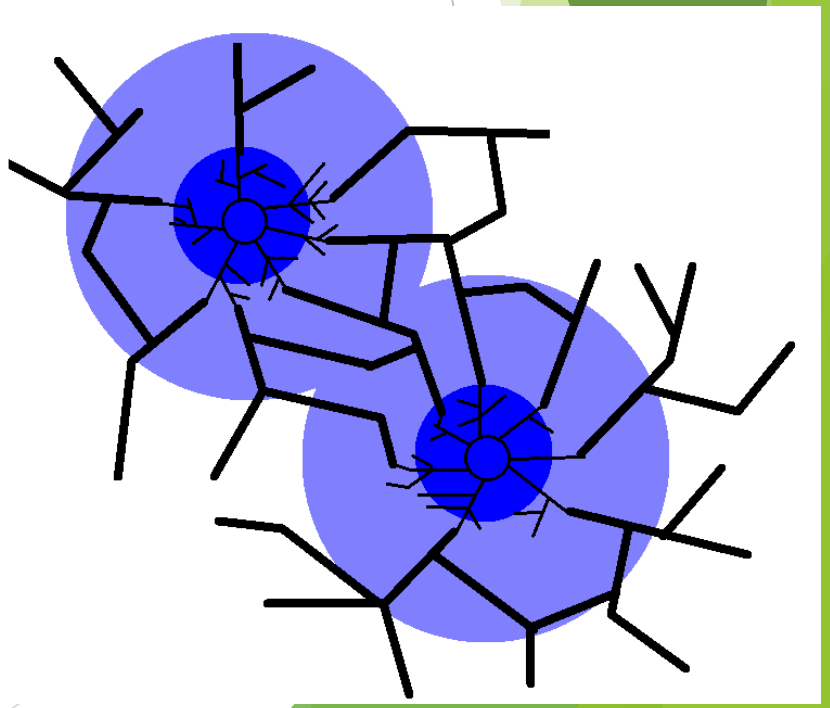
历史 & 基础知识

- [1959] Dijkstra最短路算法
- [~1968] Heuristic A*搜索算法
- [1991, 2002] Heuristic Hierarchical Approaches
  - 层次化
- [2005] Highway Hierarchies (HH)
- [2005] Highway-Node Routing (HNR)
- [2008] Contraction Hierarchies (CH)



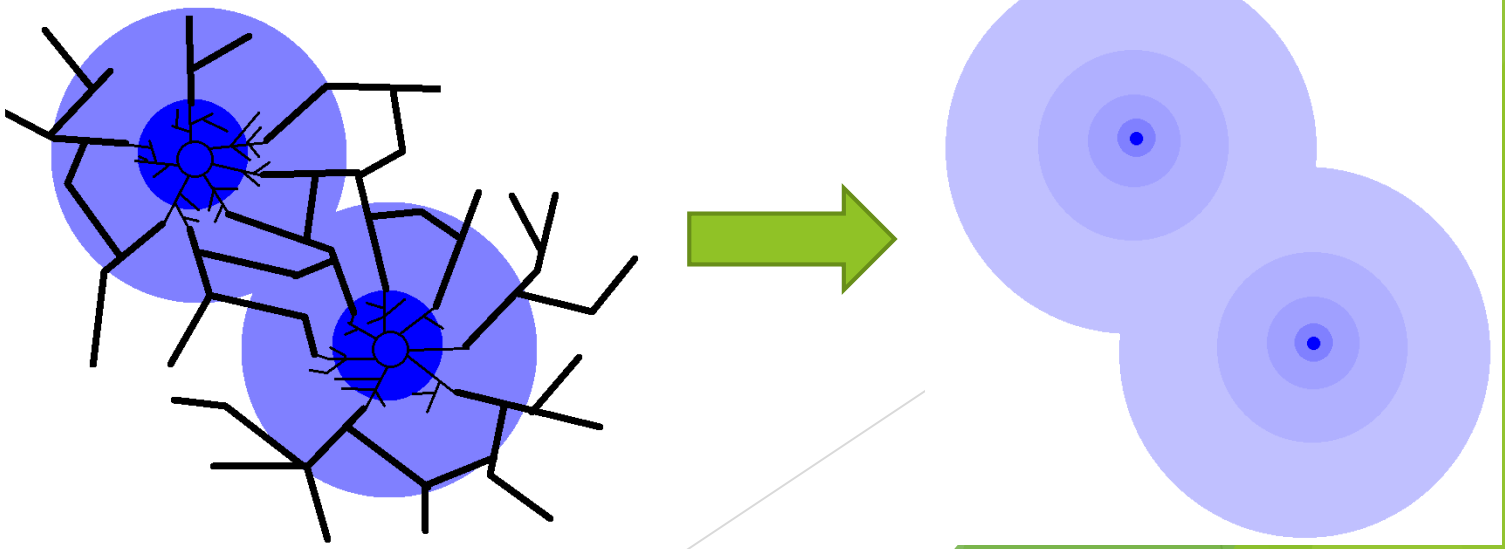
历史 & 基础知识

- Heuristic Hierarchical Approaches（启发式层次方法）
- 出现在实际系统中
  - [1991] Map Navigation Software of the Electro Multivision of the ’ 91 Toyota Soarer
  - [2002] Heuristic techniques for accelerating hierarchical routing on road networks
  - 思想：
    - 在局部考虑所有边，在局部以外只考虑”重要“的边
  - 具体方法：
    - 使用双向搜索
    - 直接对道路进行分类
      - 公路、国道、区域公路……



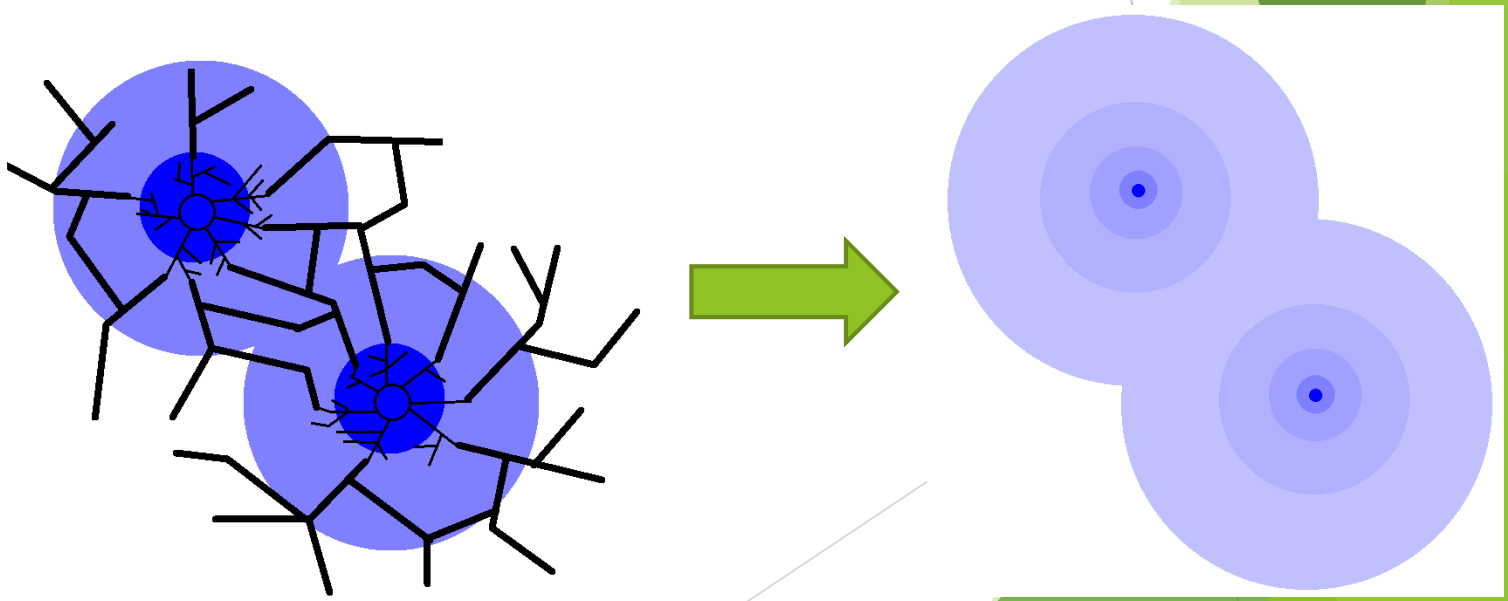
历史 & 基础知识

- Heuristic Hierarchical Approaches
- 优点：
  - 这种思想可以迭代，从而产生（层次）hierarchy
  - 层次化是很自然的想法
    - 一个村到相隔很远的另一个村
    - 先通过泥泞的小路到公路上，再从公路到高速公路上，再从高速公路回到公路，最后经过小路到达目的



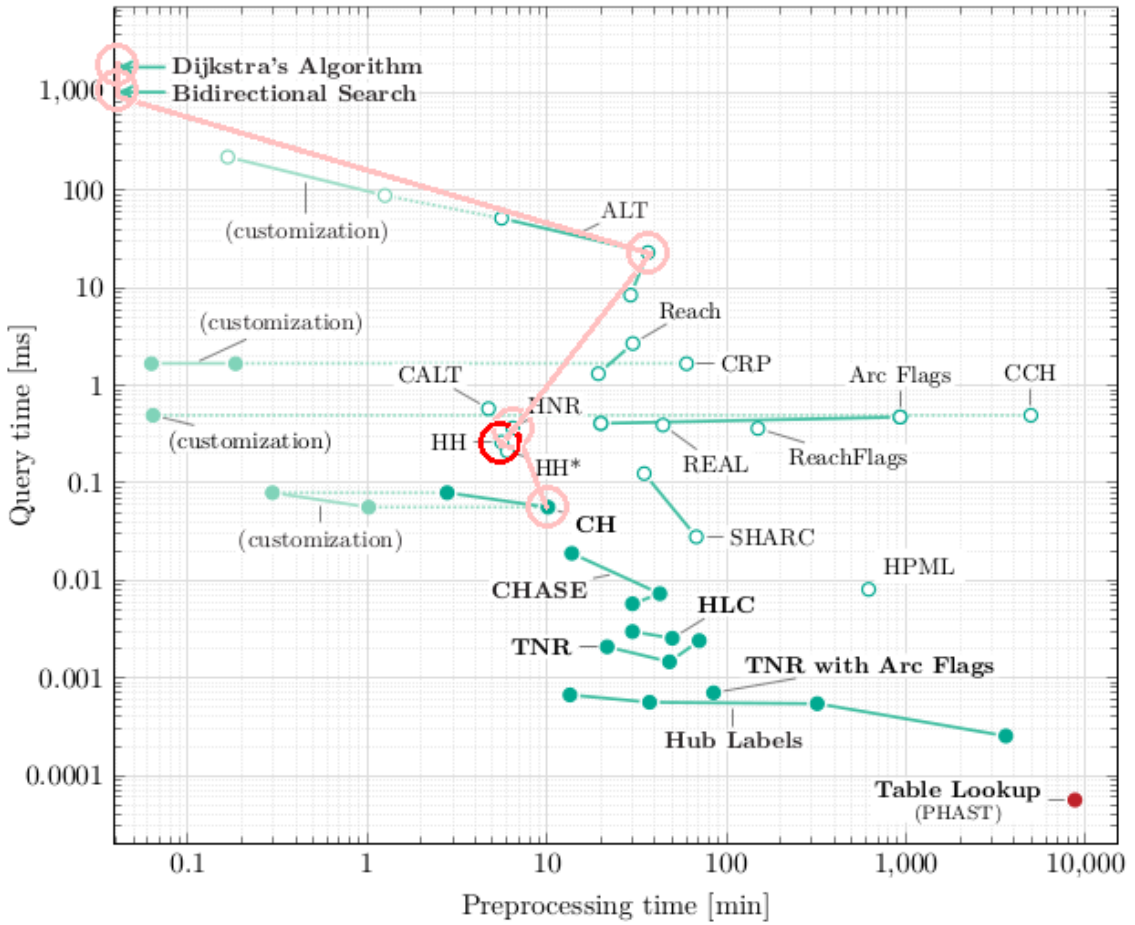
历史 & 基础知识

- Heuristic Hierarchical Approaches
- 缺点：
  - Dijkstra和A*得到的是精确的最短路，这种方法只能得到近似的最短路
  - 如何在不同层次下定义”重要“的边是个问题
    - 基本只能靠人工手调
    - 近似程度和速度之间的权衡



历史 & 基础知识

- [1959] Dijkstra最短路算法
- [~1968] Heuristic A*搜索算法
- [1991, 2002] Heuristic Hierarchical Approaches
- [2005] Highway Hierarchies (HH)
  - 层次化的精确解
- [2005] Highway-Node Routing (HNR)
- [2008] Contraction Hierarchies (CH)



历史 & 基础知识

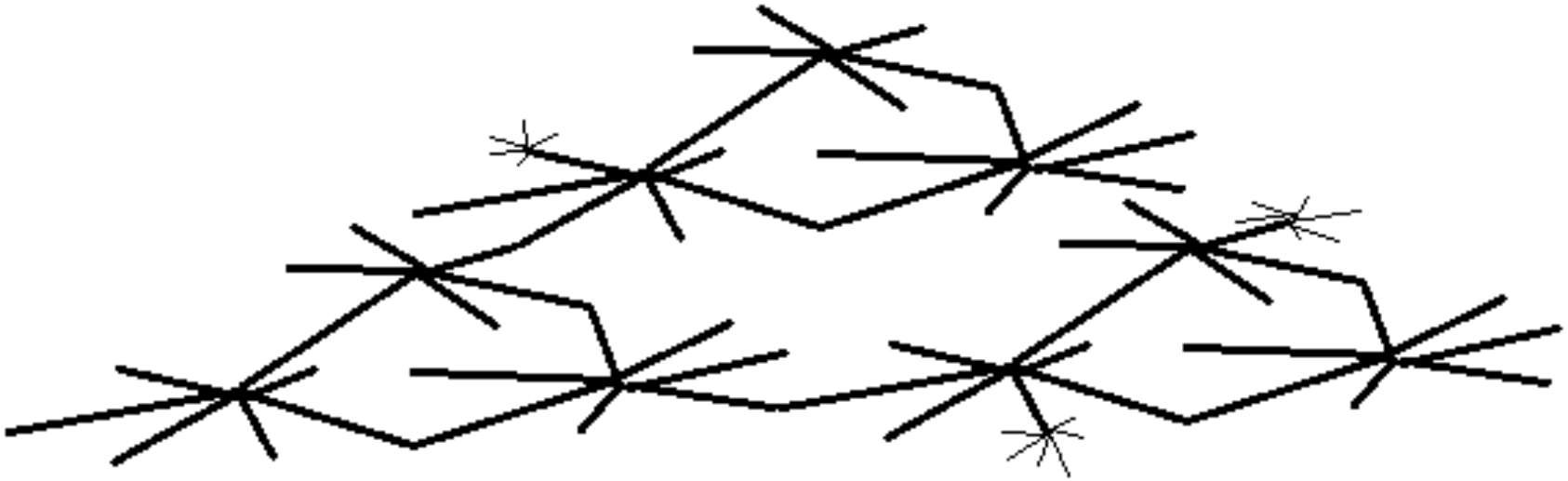
- Highway Hierarchies (HH)
- 思想：
  - Highway（高速公路）总是远离起点和终点的->自动检测高速公路
  - 保留所有点对的最短路->保证结果精确性

历史 & 基础知识

- Highway Hierarchies (HH)
- 具体方法：
  - Edge reduction（缩边）: 选取highway
    - 一条边(u,v)为highway当且仅当其在某个点对(s,t)的某条最短路上，且v不是离s前H近的点，u不是离t前H近的点
  - Node reduction（缩点）:压缩度数较低的点
    - 对得到的结果删除孤立点和冗余树
    - 压缩度数为2的点
    - 其他
  - 由此得到原图的一个上层图，再迭代处理

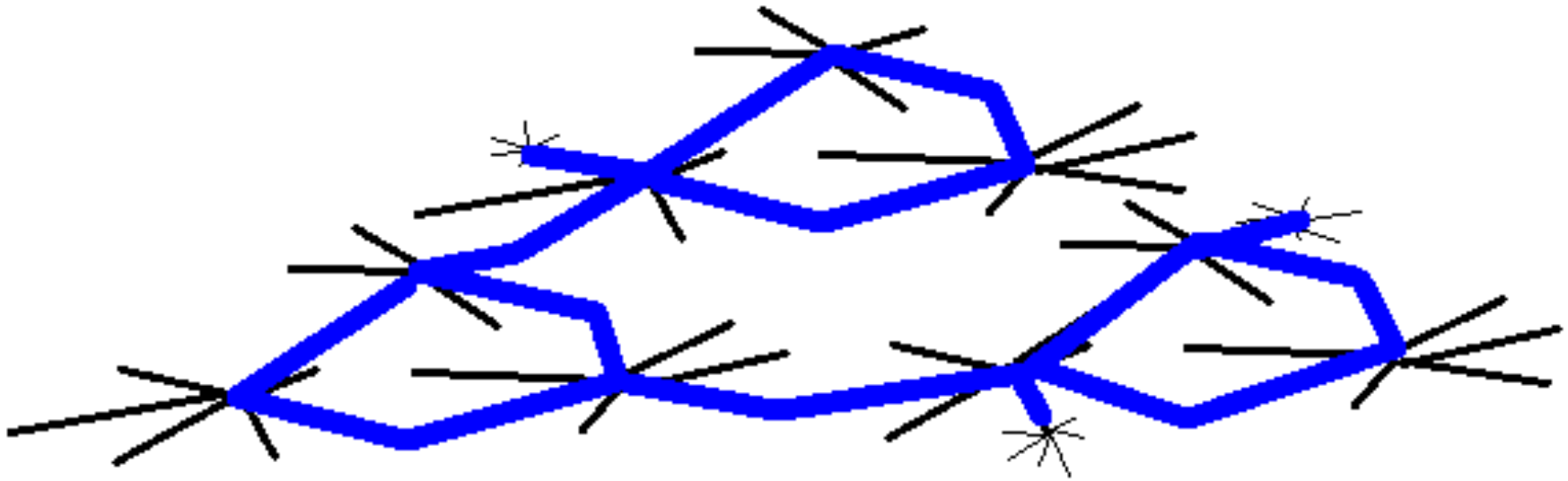
历史 & 基础知识

- Highway Hierarchies (HH)



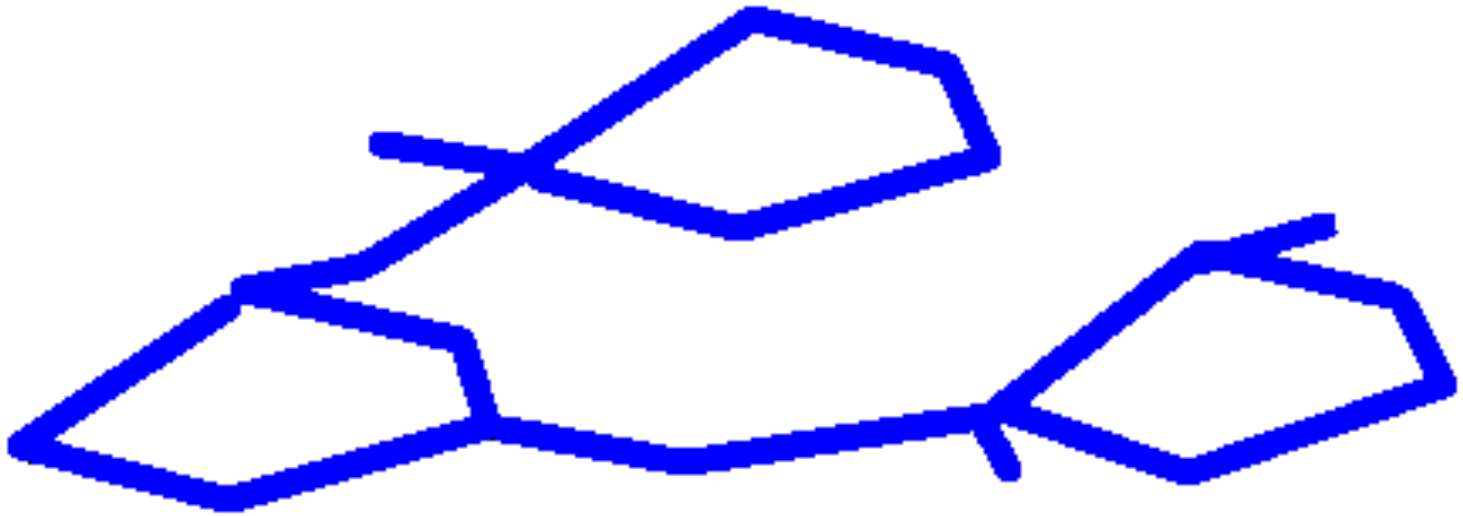
历史 & 基础知识

- Highway Hierarchies (HH)



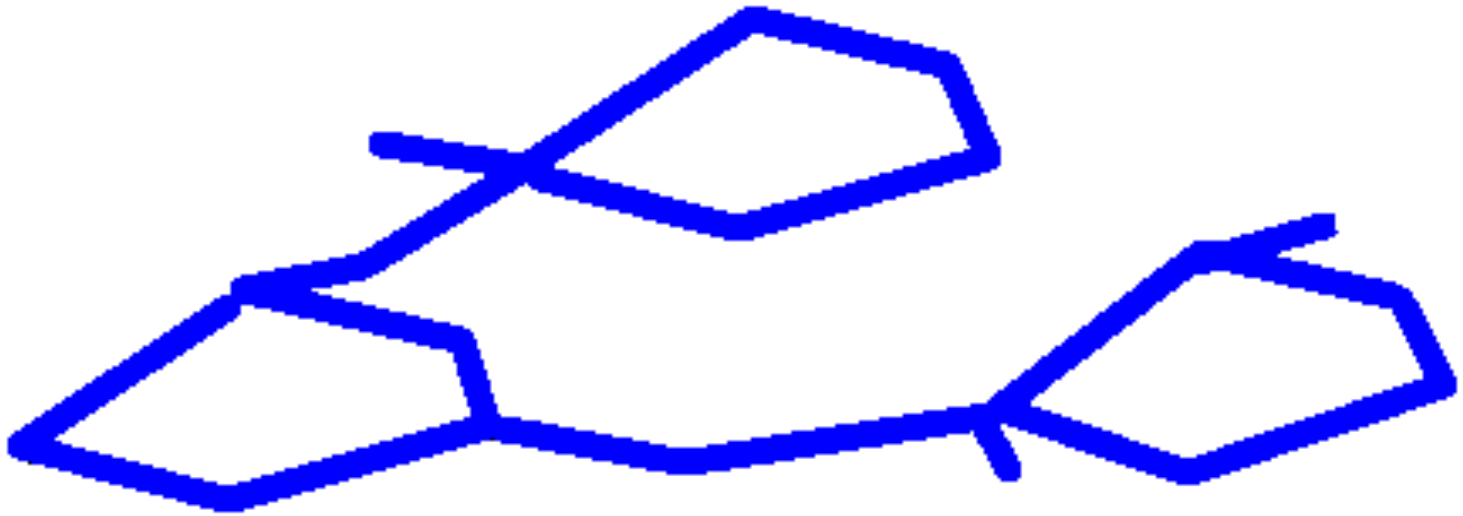
历史 & 基础知识

- Highway Hierarchies (HH)



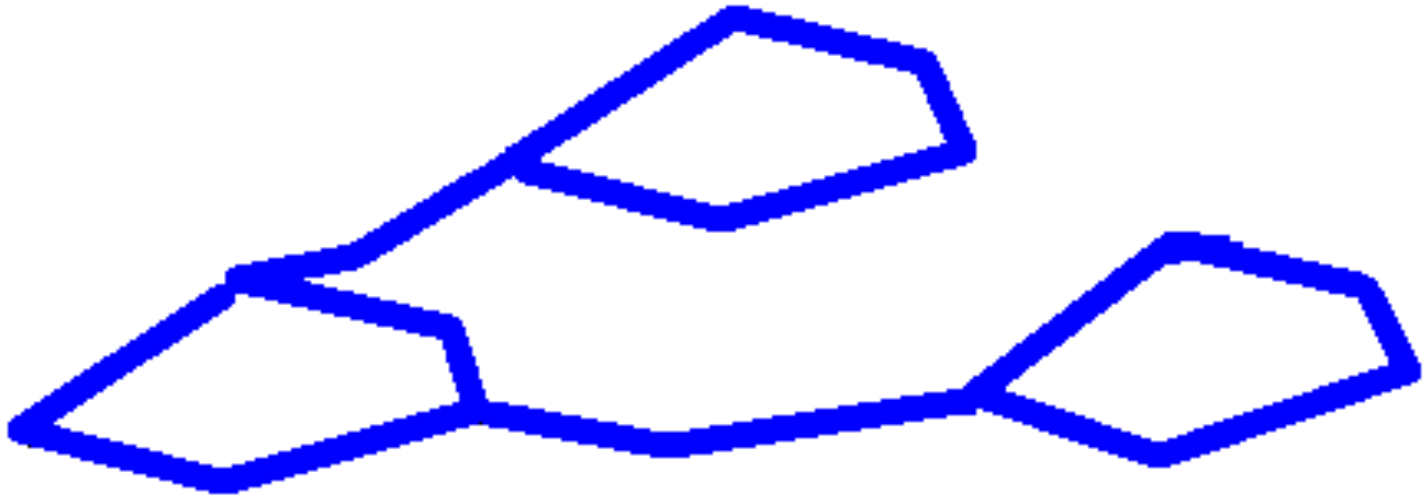
历史 & 基础知识

- Highway Hierarchies (HH)



历史 & 基础知识

- Highway Hierarchies (HH)

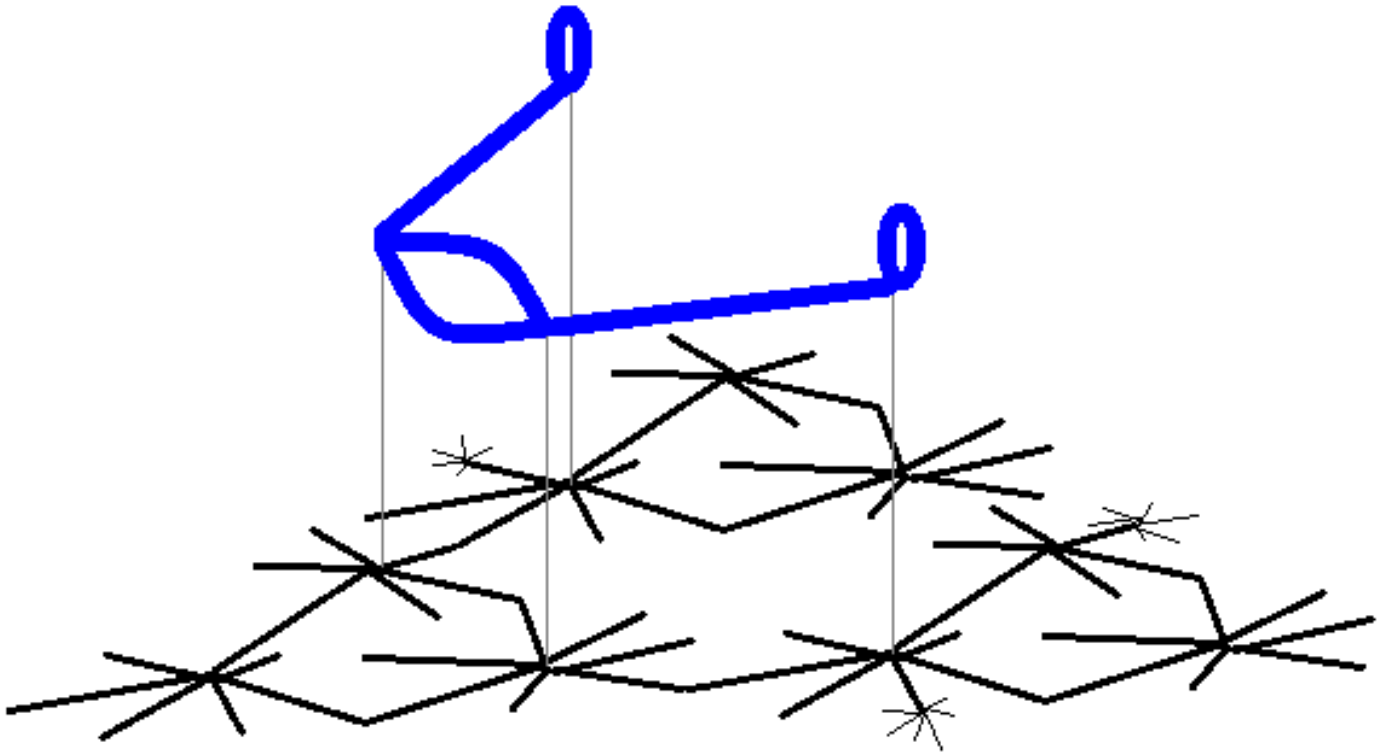


历史 & 基础知识

- Highway Hierarchies (HH)

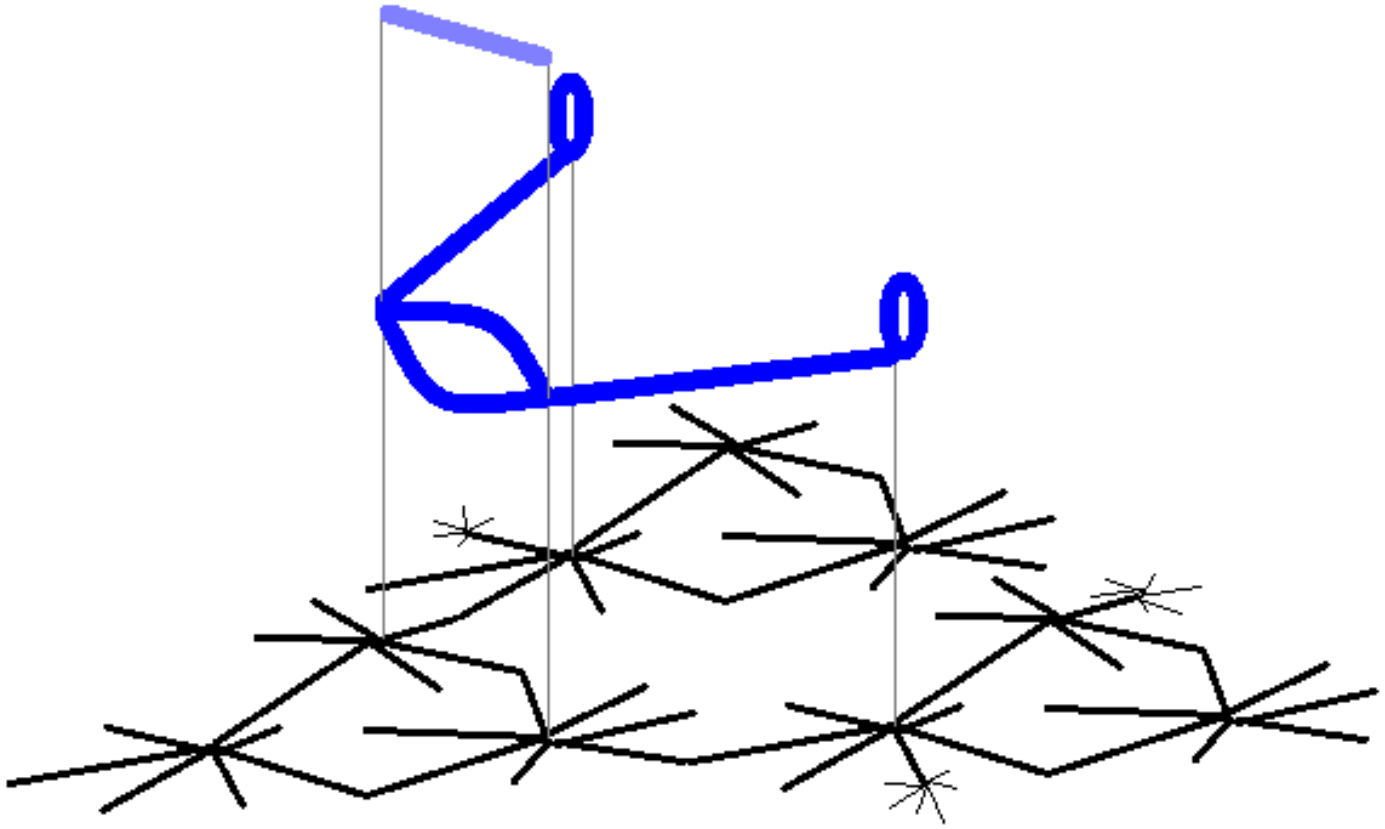
历史 & 基础知识

- Highway Hierarchies (HH)



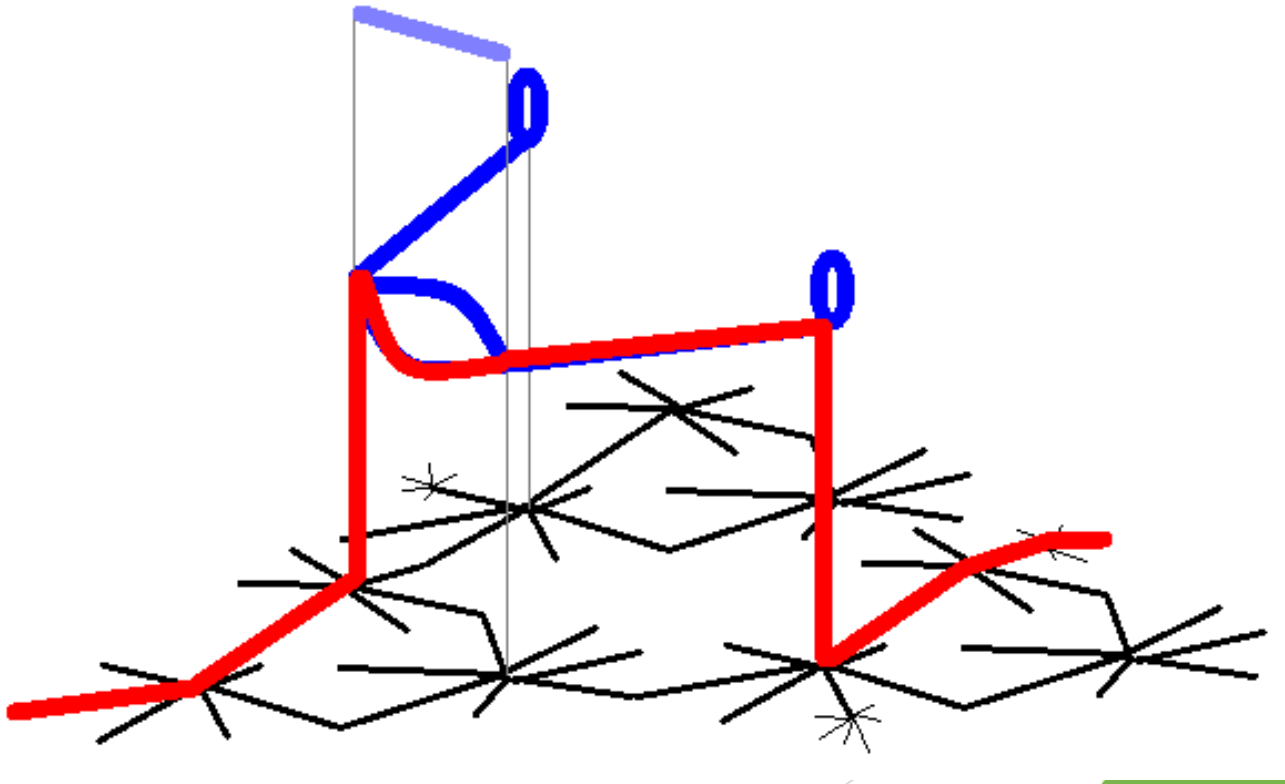
历史 & 基础知识

- Highway Hierarchies (HH)



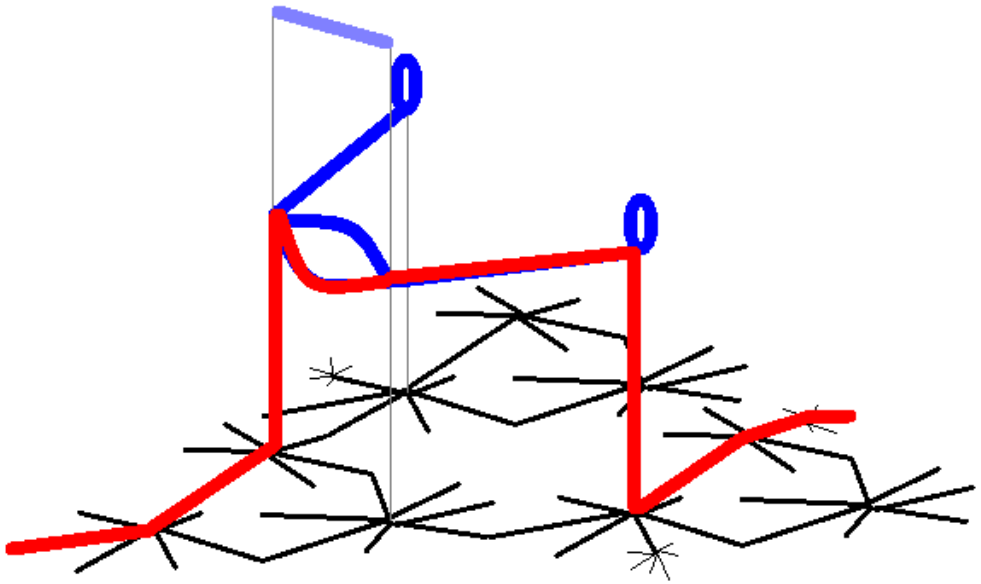
历史 & 基础知识

- Highway Hierarchies (HH)
- 高层和低层之间的边权为0
- 查询最短路时使用双向Dijkstra搜索



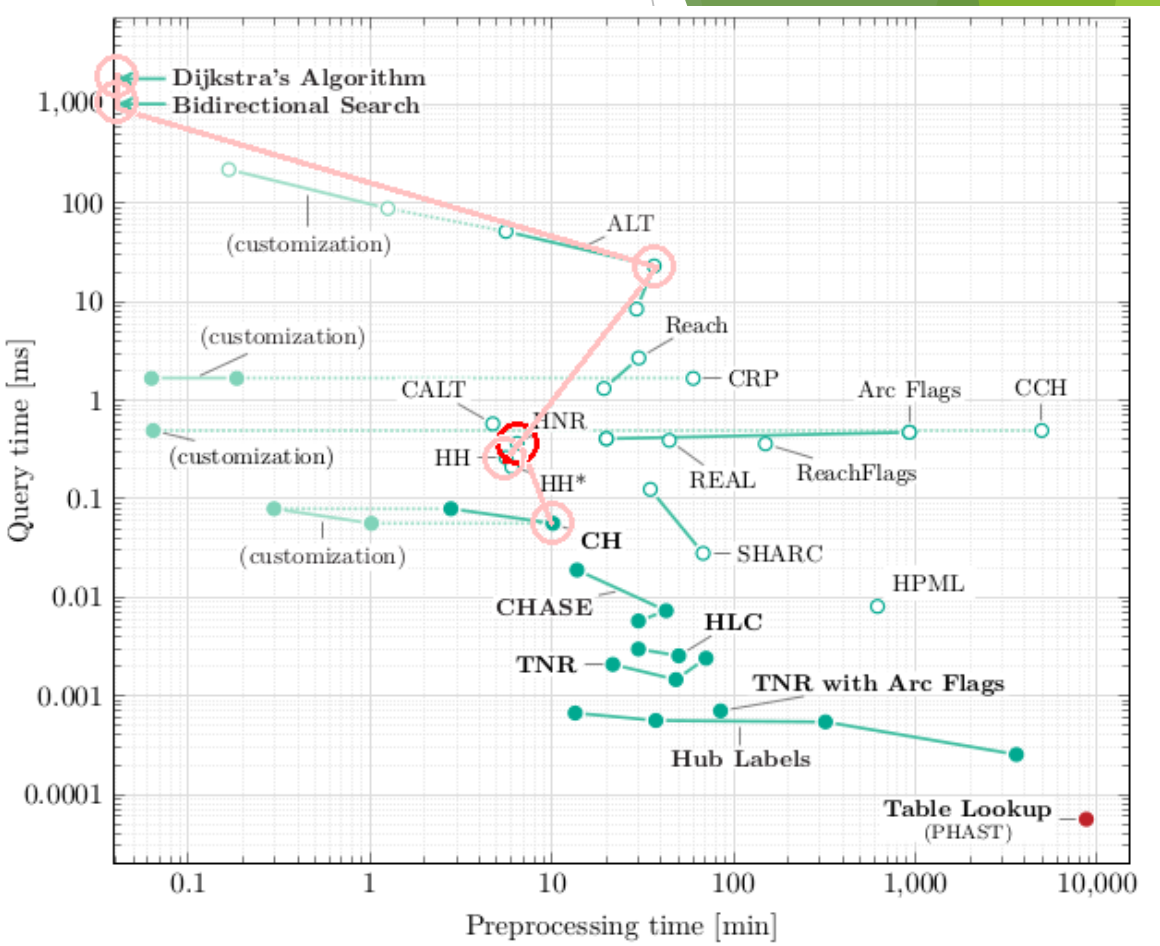
历史 & 基础知识

- Highway Hierarchies (HH)
- 缩点后原本需要数步或数十步的搜索可能一步就能达到，高层图是原图在查询最短路时的”高速公路“
- 第一个在庞大公路网络上query时间达到毫秒级别的算法
- 缺点：
  - 在局部铁路网络、高维格点网络等网络上表现不好
  - 较难应对动态权值的网络



历史 & 基础知识

- [1959] Dijkstra最短路算法
- [~1968] Heuristic A*搜索算法
- [1991, 2002] Heuristic Hierarchical Approaches
- [2005] Highway Hierarchies (HH)
- [2005] Highway-Node Routing (HNR)
  - 层次化的一般抽象
- [2008] Contraction Hierarchies (CH)



历史 & 基础知识

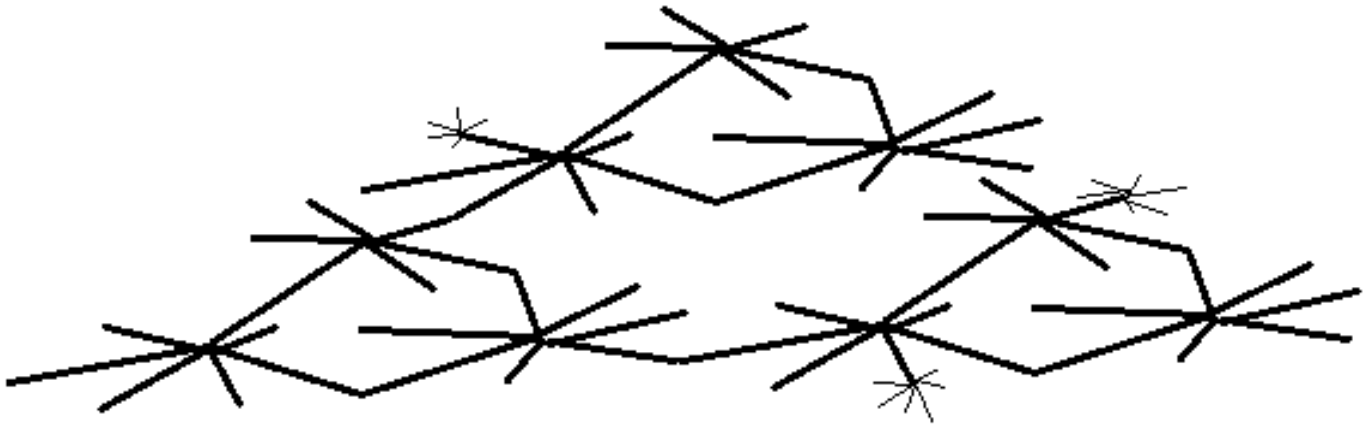
- Highway-Node Routing (HNR)
- 原因：
  - 为了更好地解决动态权值网络的最短路问题
  - 为了简化HH

历史 & 基础知识

- Highway-Node Routing (HNR)
- 思想：
  - 在原图的基础上挑选重要的点（highway node），通过原图上的局部搜索保留这些点间的最短路径
- 具体方法：
  - 通过某种方法决定重要的点集
    - 启发式赋权/图的割点集/访问频率
  - 局部搜索构建新的“高速公路”(shortcut)
    - 和哪些点连边是个问题，防止变成完全图
  - 反复迭代得到层次图

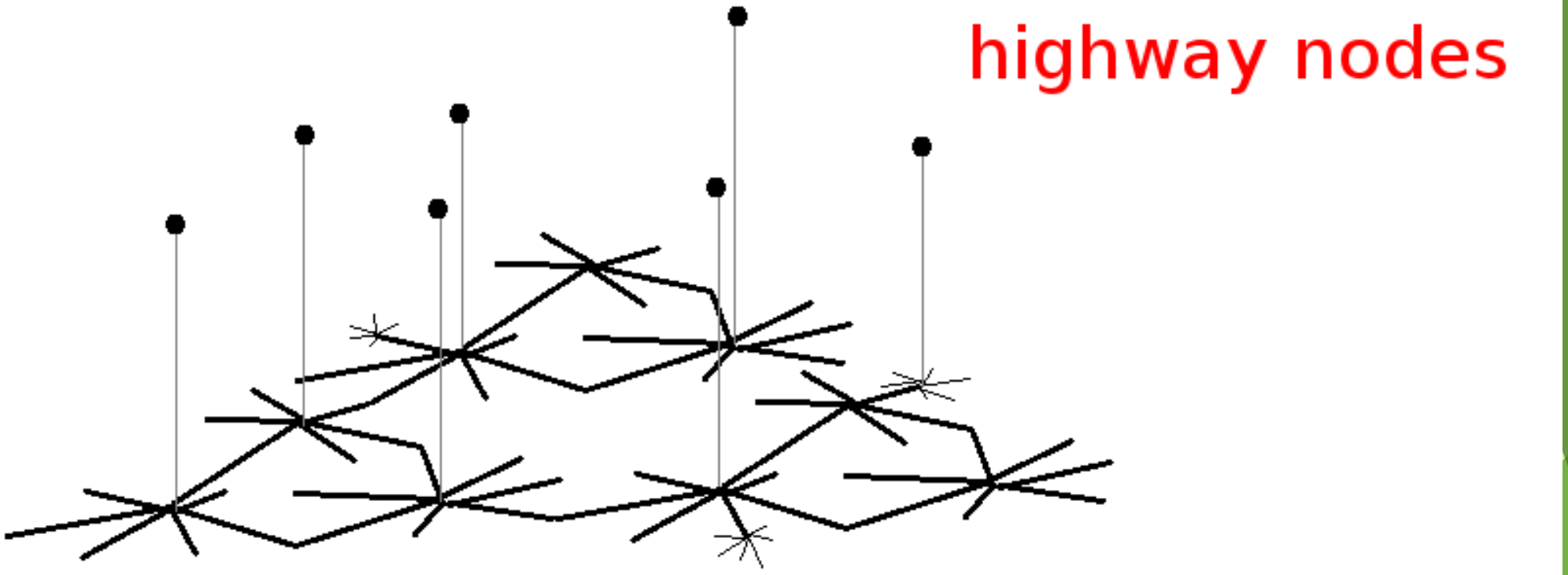
历史 & 基础知识

- Highway-Node Routing (HNR)



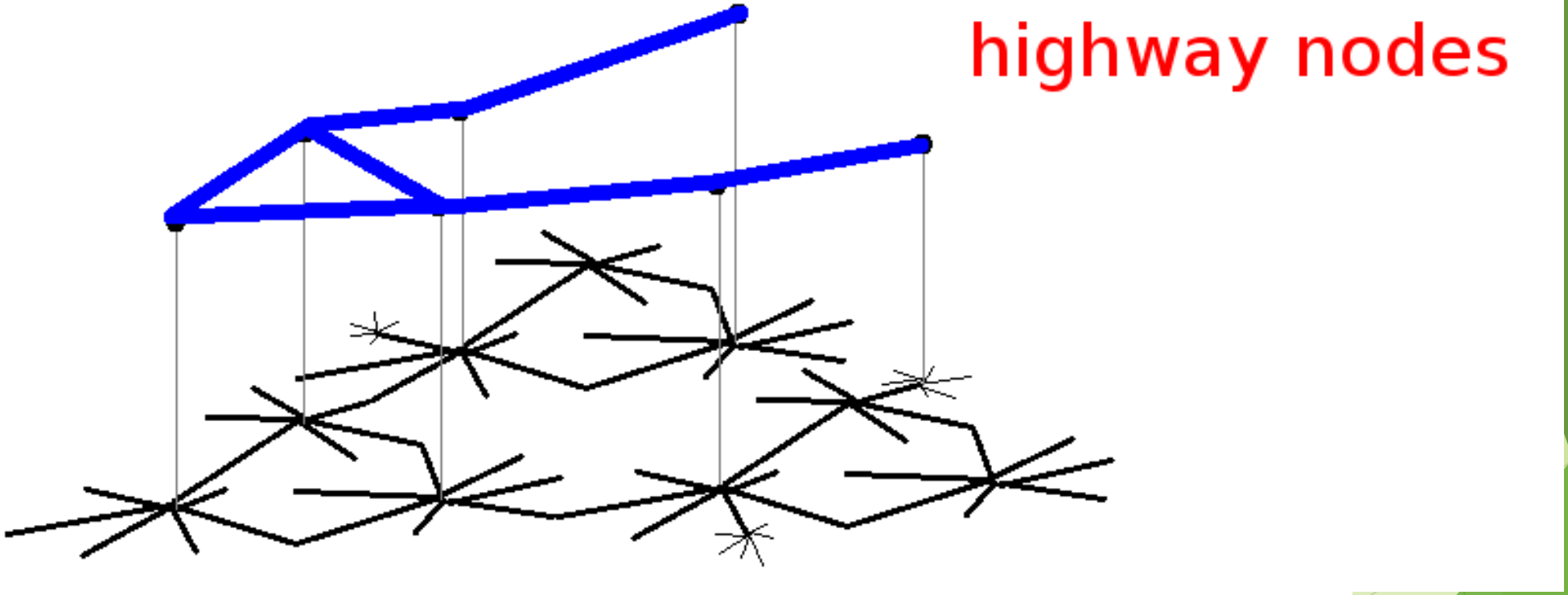
历史 & 基础知识

- Highway-Node Routing (HNR)



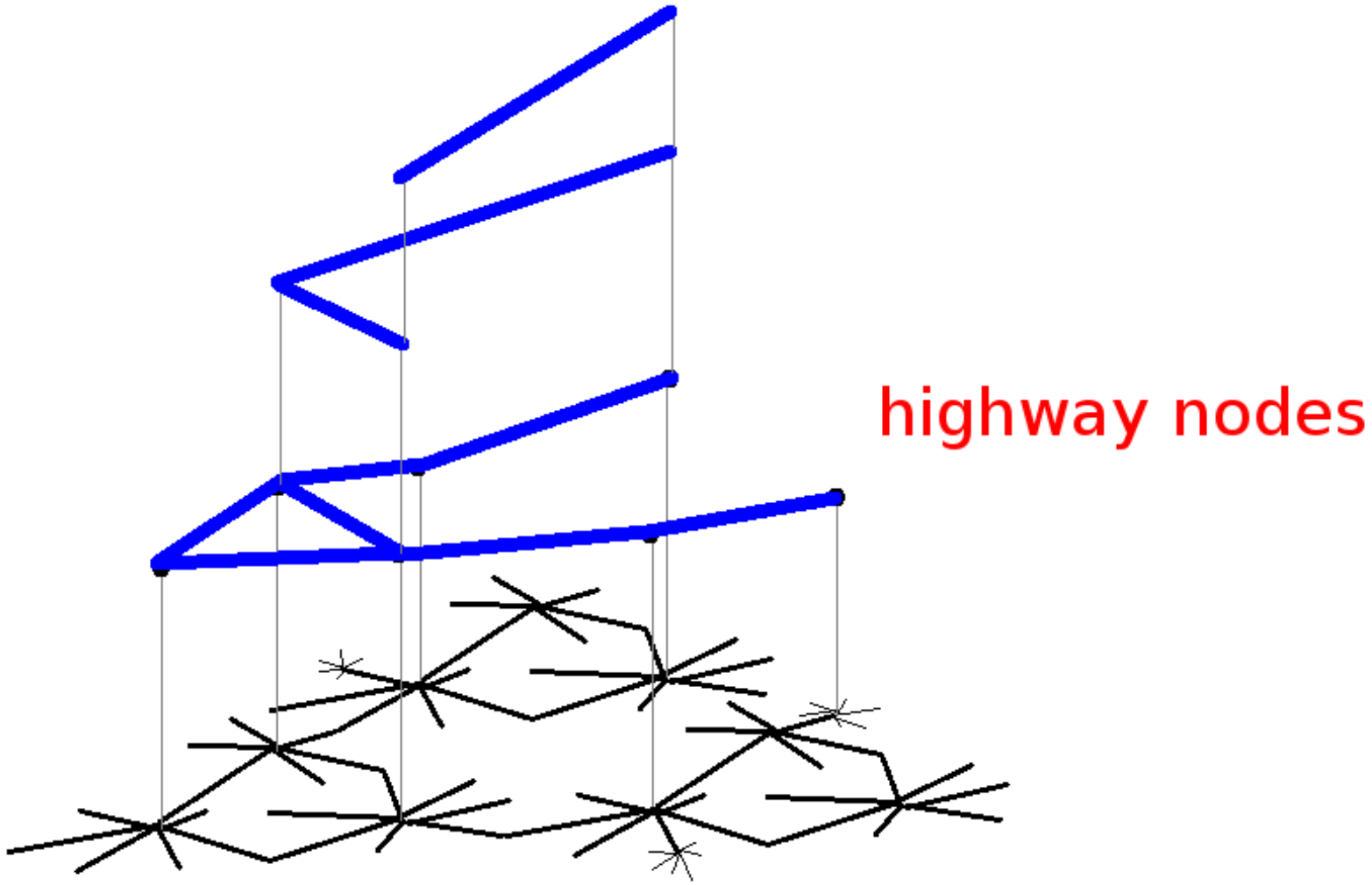
历史 & 基础知识

- Highway-Node Routing (HNR)



历史 & 基础知识

- Highway-Node Routing (HNR)



历史 & 基础知识

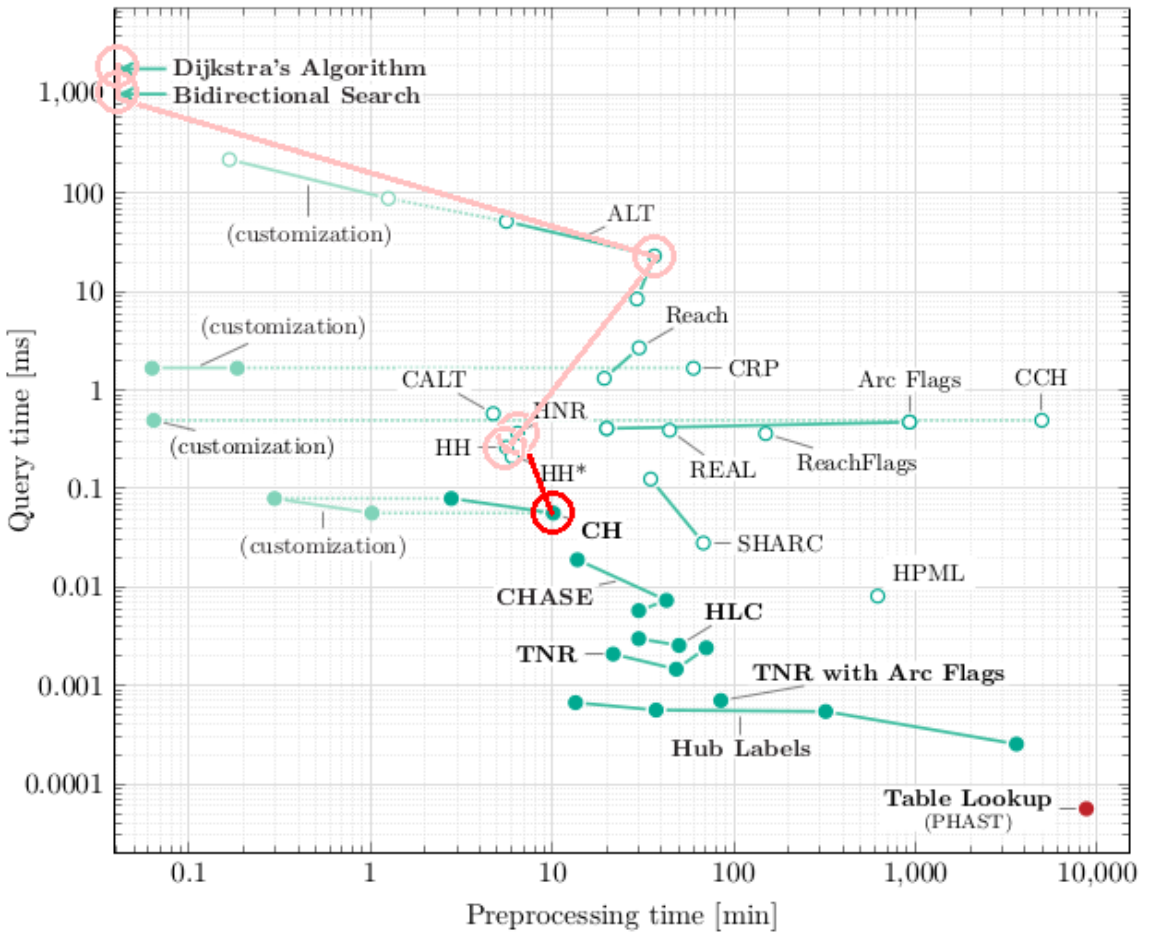
- Highway-Node Routing (HNR)
- 优点：
  - 相比HH，只有node reduction，没有edge reduction
  - 更好地应对动态网络：更新高层图只需要做局部搜索
- 缺点：
  - 怎么选取点的重要程度是个问题
    - 实践中常用HH得到的点序

历史 & 基础知识

- 当前算法在实践中已经比Dijkstra算法快数千倍

历史 & 基础知识

- [1959] Dijkstra最短路算法
- [~1968] Heuristic A*搜索算法
- [1991, 2002] Heuristic Hierarchical Approaches
- [2005] Highway Hierarchies (HH)
- [2005] Highway-Node Routing (HNR)
- [2008] Contraction Hierarchies (CH)



历史 & 基础知识

- Contraction Hierarchies (CH)
- Motivation
  - 我们一开始只想给HNR找除了HH外的一个更好的点序。HH是balabala这样工作的。在HH的原始论文中，缩点只是删除了一度点和二度点，也就是，把冗余的树和多跳路径给删了。

The node ordering in highway-node routing uses levels computed by *highway hierarchies* (HHs) [4 5 2]. Our original motivation for CHs was to simplify HNR by obviating the need for another (more complicated) speedup technique (HHs) for node ordering. HHs are constructed by alternating between two subroutines: *Edge reduction* is a sophisticated and relatively costly routine that only keeps edges required 'in the middle' of 'long-distance' paths. *Node reduction* contracts nodes. In the original paper for undirected HHs [5], node reduction only contracted nodes of degrees one and two, i.e., it removed attached trees and multihop

历史 & 基础知识

- Contraction Hierarchies (CH)
- Motivation
  - 我们最早认为缩点只是一个辅助，缩边才是主力。（所以我们想优化一下缩点看看有没有什么效果然后才好发paper毕业）balabala。我们发现edge difference在缩点中非常重要，也有人做过相关工作，但他们最终的高层图密集得变成完全图了，所以减少冗余边也挺重要的。

paths. We originally viewed node contraction as a mere helper for the main workhorse edge reduction. For directed graphs [5], we needed a more general criterion which nodes should be contracted away. It turned out that the edge difference is a good way to estimate the cost of contracting a node v. In [6, 7] this method is further refined to use a priority queue and to avoid parallel edges. All previous approaches to contraction had in common that the average degree of the nodes in the overlay graph would eventually explode. So it looked like an additional technique such as edge reduction or reaches would be a necessary ingredient of any high-performance hierarchical routing method. Perhaps the most important result of CHs is that using *only* (a more sophisticated) node contraction, we get very good performance.

历史 & 基础知识

- Contraction Hierarchies (CH)
- Motivation
  - 不过我们觉得CH最重要的结果是告诉我们
    - 只靠缩点就可以得到很好的层次图

paths. We originally viewed node contraction as a mere helper for the main workhorse edge reduction. For directed graphs [5], we needed a more general criterion which nodes should be contracted away. It turned out that the edge difference is a good way to estimate the cost of contracting a node v. In [6, 7] this method is further refined to use a priority queue and to avoid parallel edges. All previous approaches to contraction had in common that the average degree of the nodes in the overlay graph would eventually explode. So it looked like an additional technique such as edge reduction or reaches would be a necessary ingredient of any high-performance hierarchical routing method. Perhaps the most important result of CHs is that using *only* (a more sophisticated) node contraction, we get very good performance.

历史 & 基础知识

- Contraction Hierarchies (CH)
- Motivation
  - 现在最快的方法是TNR，查询时间比CH快40倍，但它内存消耗和预处理时间太大，而且不好应对动态网络。最要命的是它的点序也是靠别的算法产生的（比如HH）。
  - 我们发现CH产生的点序可以提升它的速度。

The fastest speedup technique so far, *transit-node routing* [8 2], offers a factor up to 40 times better query times than CHs. However, it needs considerably higher preprocessing time and space, is less amenable to dynamization, and, most importantly it relies on another hierarchical speedup technique for its preprocessing. We have preliminary evidence that using CHs for this purpose leads to improved performance.

历史 & 基础知识

- Contraction Hierarchies (CH)
- Motivation

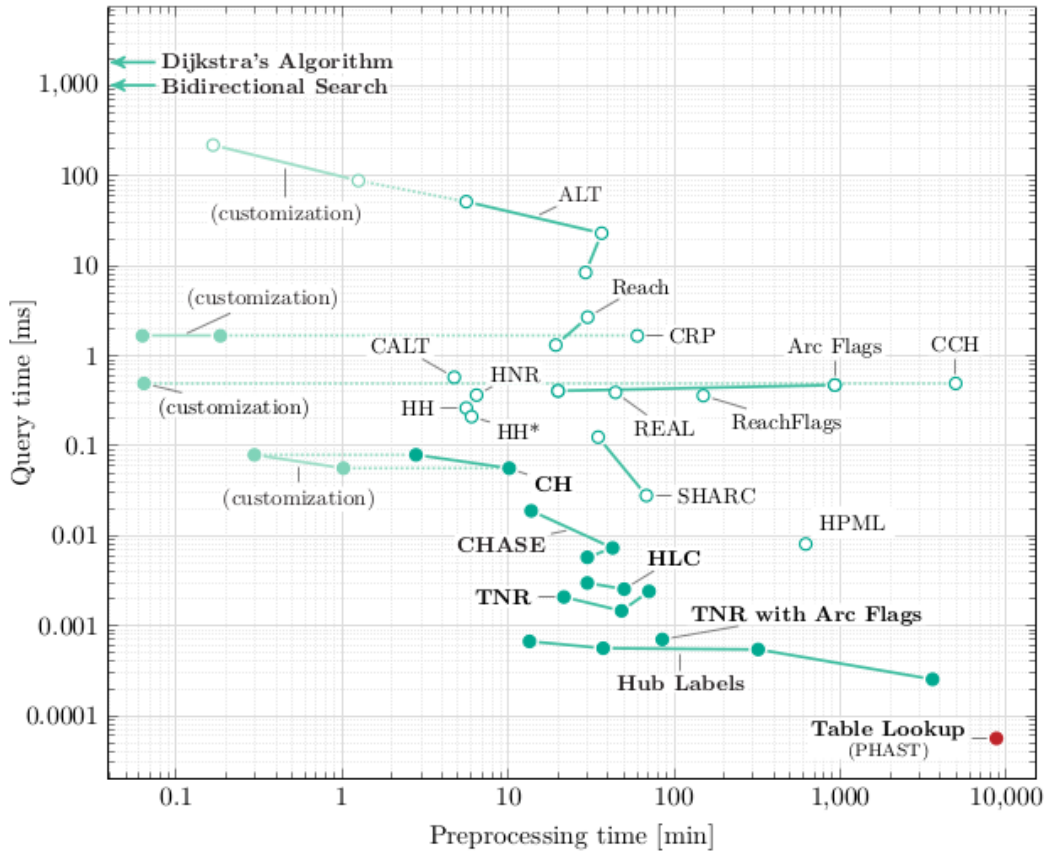


Figure 7. Preprocessing and average query time performance for algorithms with available experimental data on the road network of Western Europe, using travel times as edge weights. Connecting lines indicate different trade-offs for the same algorithm. The figure is inspired by [238].

Table 1. Performance of various speedup techniques on Western Europe. Column *source* indicates the implementation tested for this survey.

		DATA STRUCTURES		QUERIES	
algorithm	impl. source	space [GiB]	time [h:m]	scanned vertices	time [µs]
Dijkstra	[75]	0.4	–	9 326 696	2 195 080
Bidir. Dijkstra	[75]	0.4	–	4 914 804	1 205 660
CRP	[77]	0.9	1:00	2 766	1 650
Arc Flags	[75]	0.6	0:20	2 646	408
CH	[77]	0.4	0:05	280	110
CHASE	[75]	0.6	0:30	28	5.76
HLC	[82]	1.8	0:50	–	2.55
TNR	[15]	2.5	0:22	–	2.09
TNR+AF	[40]	5.4	1:24	–	0.70
HL	[82]	18.8	0:37	–	0.56
HL-∞	[5]	17.7	60:00	–	0.25
table lookup	[75]	1 208 358.7	145:30	–	0.06

简介

简介

- CH怎么做的？
- CH为什么是对的？
- CH为什么快？

简介

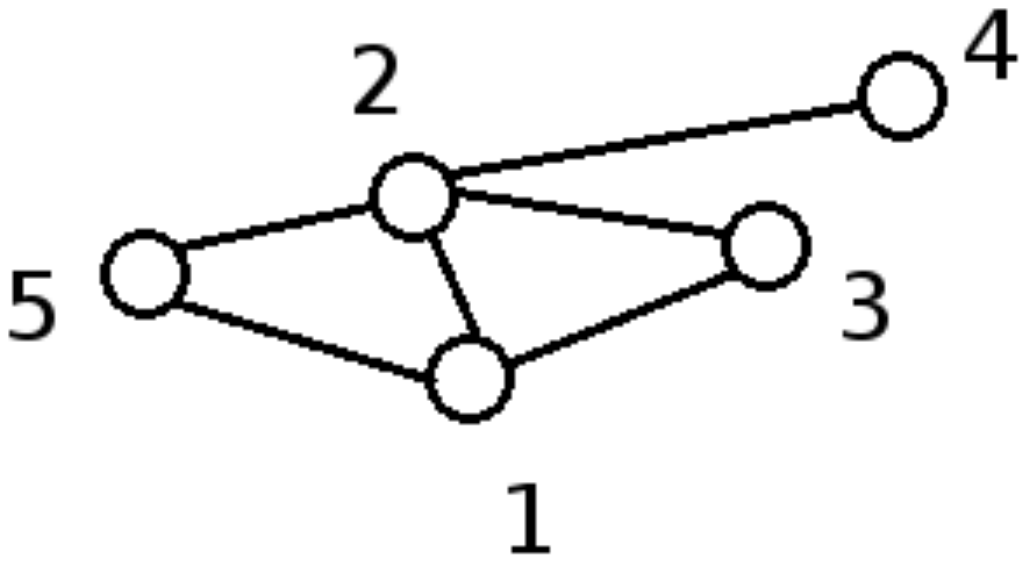
- CH怎么做的？
  - 预处理
    - 确定一个缩点的顺序
    - 每次按这个顺序选一个点u缩
      - 如果v->u->w可能是v到w的唯一的最短路，在原图中增加shortcut捷径(v,w)，权值为(v,u)和(u,w)的权值和
    - 删除点u

Algorithm 1: SimplifiedConstructionProcedure($G = (V, E), <$)

```
1 foreach u ∈ V ordered by < ascending do
2     foreach (v, u) ∈ E with v > u do
3         foreach (u, w) ∈ E with w > u do
4             if ⟨v, u, w⟩ "may be" the only shortest path from v to w then
5                 E := E ∪ {(v, w)} (use weight w(v, w) := w(v, u) + w(u, w))
```

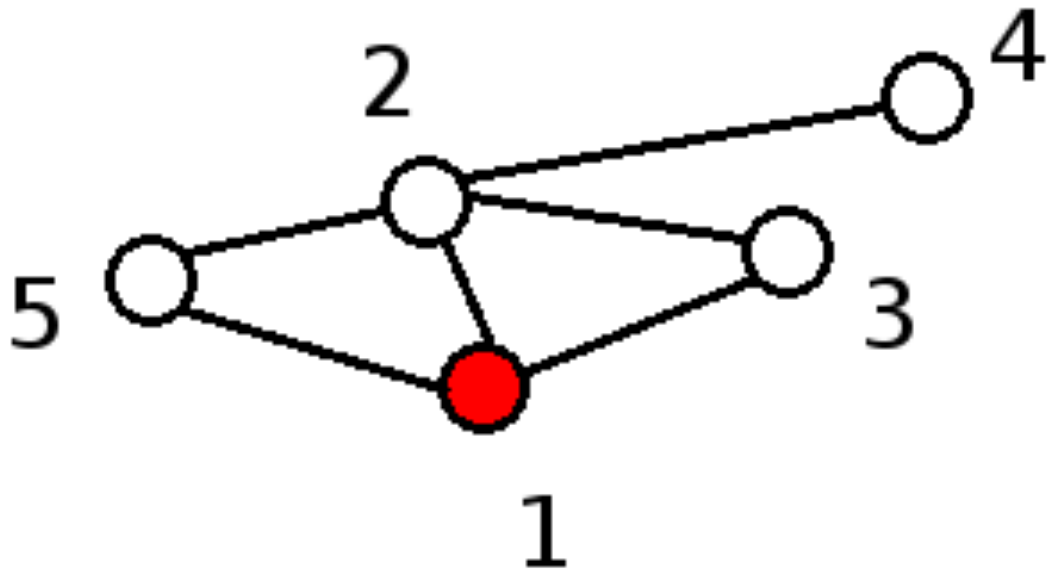
简介

- CH怎么做的？
  - 这里用一个无向无权图表示，实际情况是有向有权的，大家理解意思就好



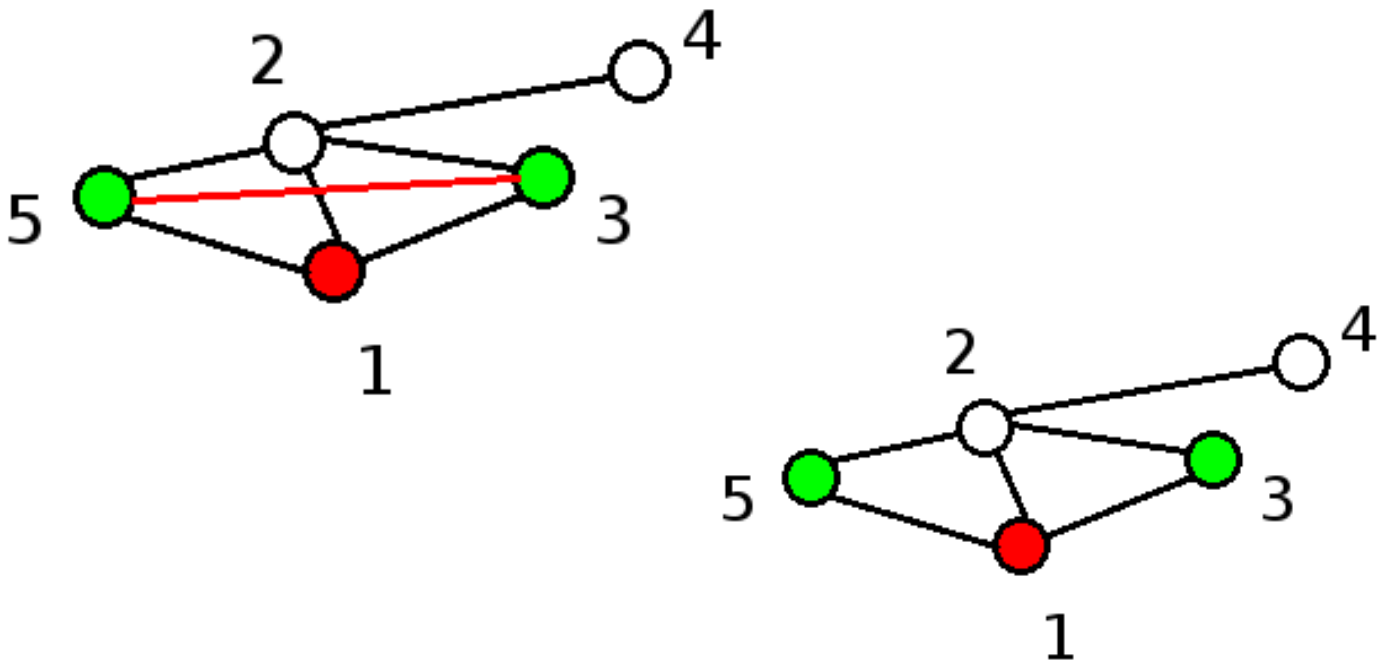
简介

- CH怎么做的？



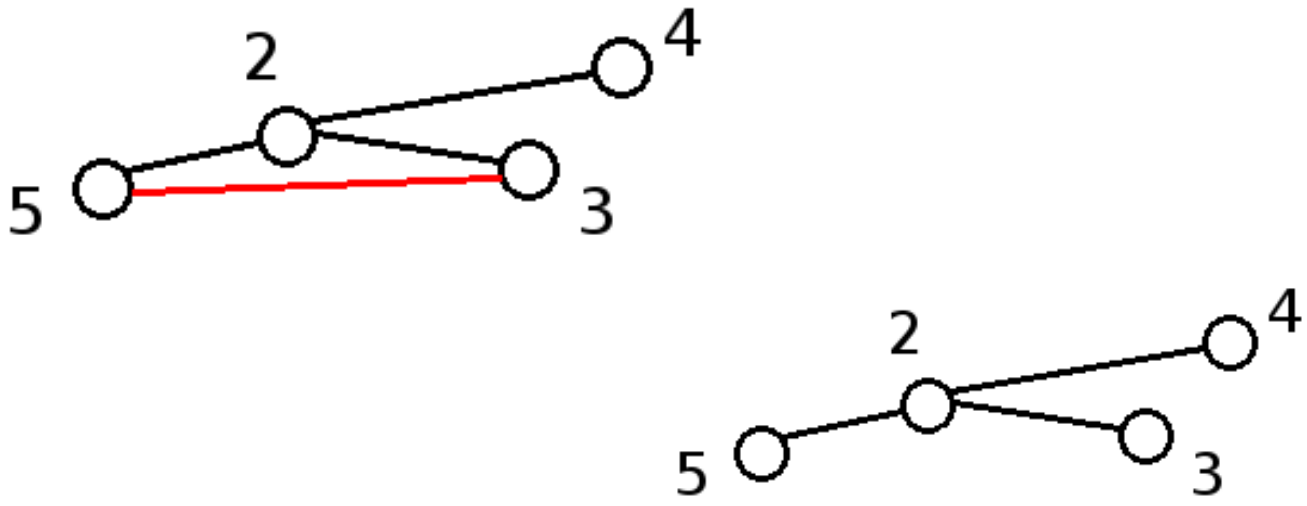
简介

- CH怎么做的？
  - 看情况加shortcut
    - 如果5到3的最短路一定要经过1就得加上，否则不加



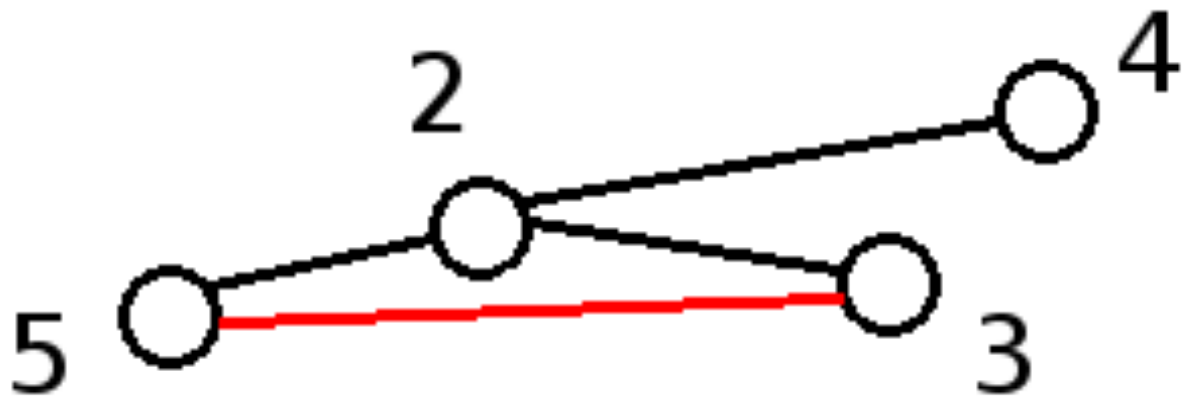
简介

- CH怎么做的？
  - 删点1



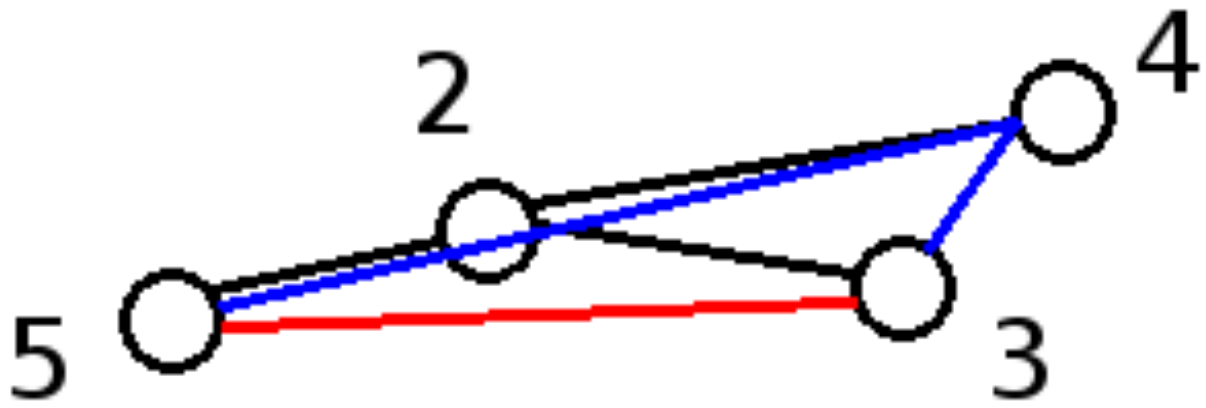
简介

- CH怎么做的？
  - 方便起见下面我们只展示第一种情况



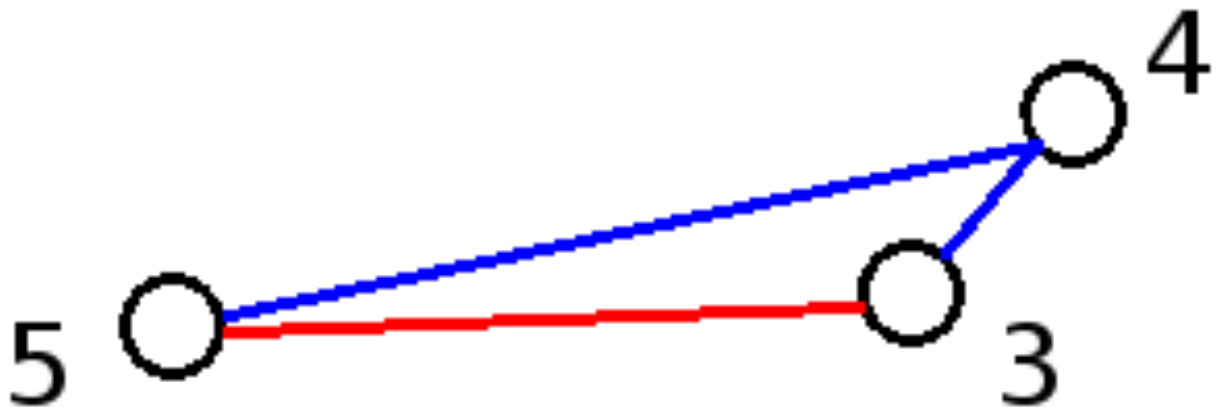
简介

- CH怎么做的？
  - 毫无疑问点5到点4,点3到点4的最短路肯定经过点2
  - 必须加上shortcut



简介

- CH怎么做的？
  - 删除点2



简介

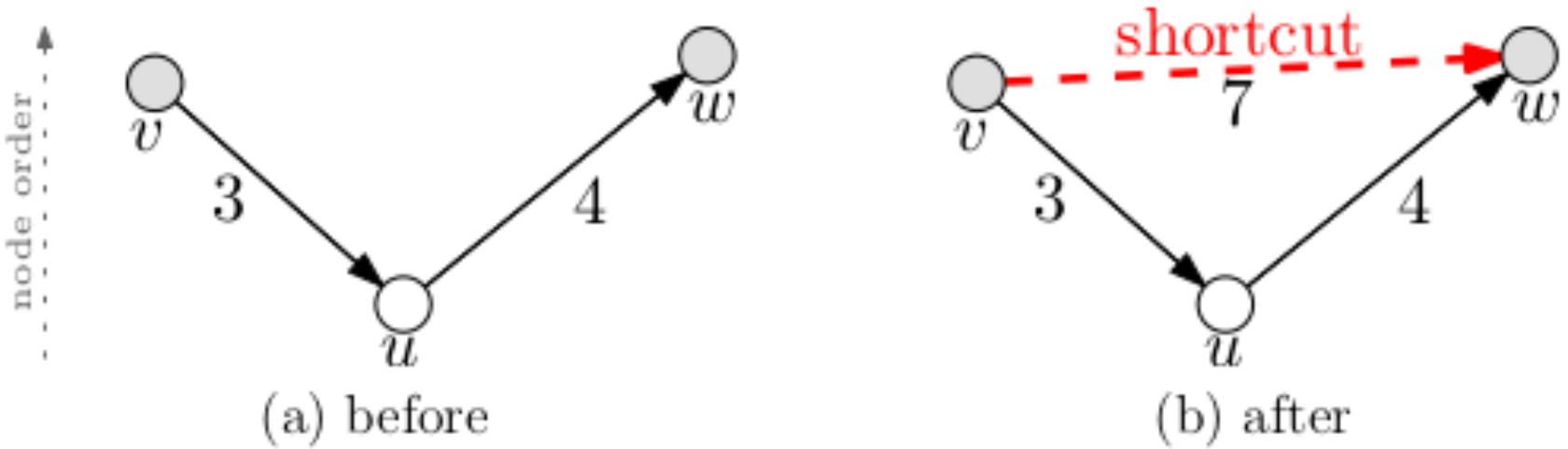
- CH怎么做的？
  - 类似地，contract点3

简介

- CH怎么做的？
  - contract点4

简介

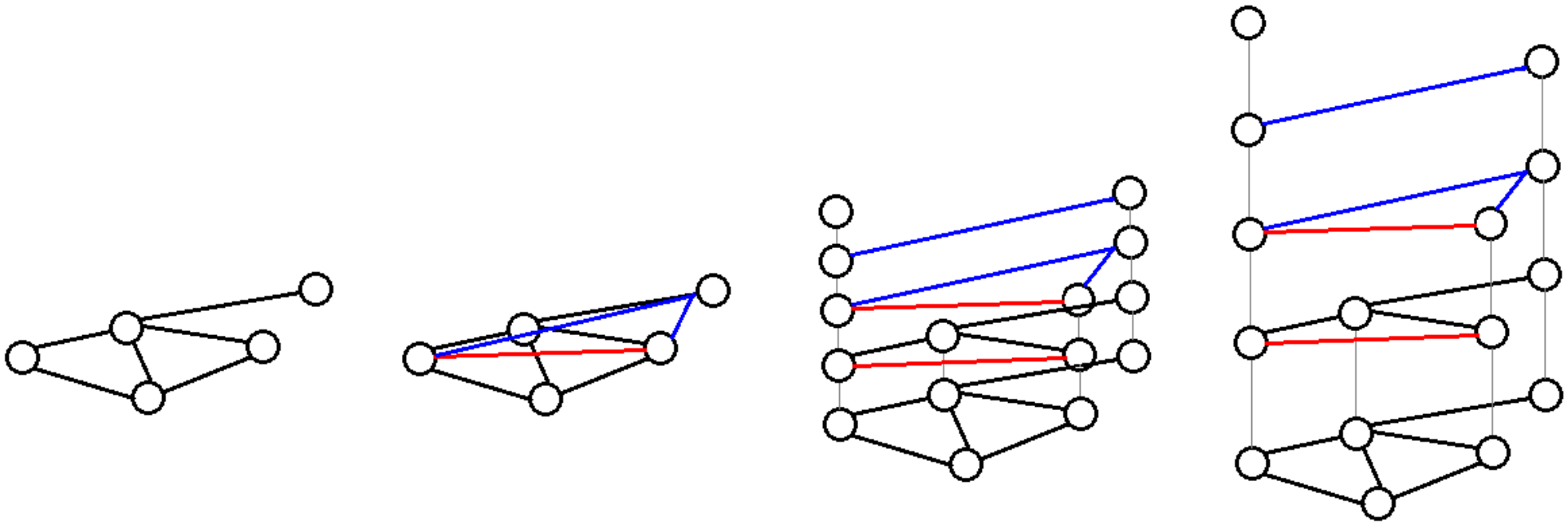
- CH怎么做的？
  - 预处理结束后所有添加进去的边就构成了层次图



简介

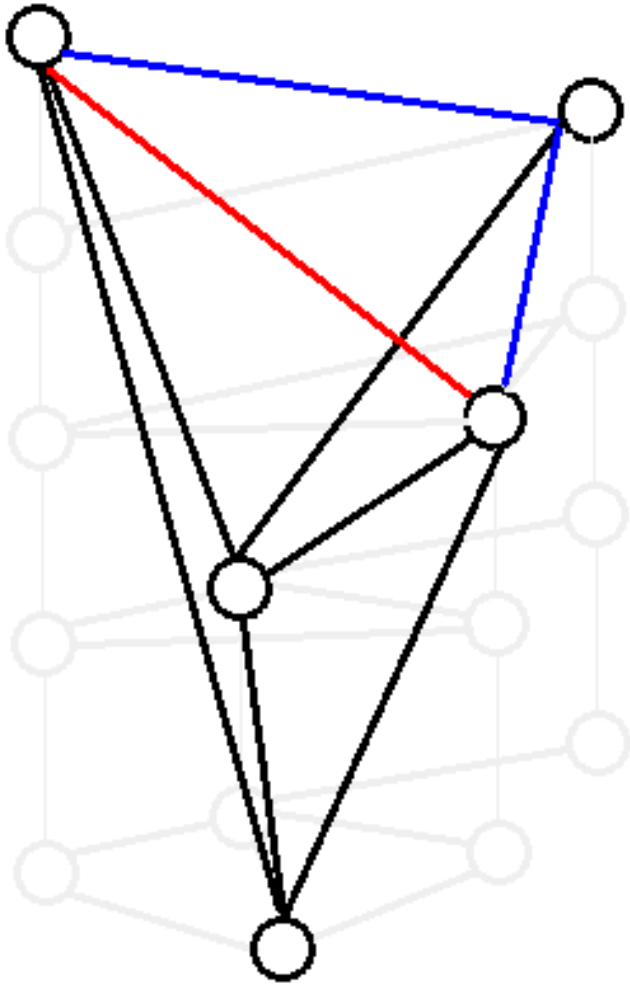
- CH怎么做的？
  - 层次在哪里体现？
  - 缩点的顺序就是层次，CH实际上构建了一个|V|层的层次图

edge weights change. Contraction Hierarchies are an extreme case of the hierarchies in HNR – every node defines its own level of the hierarchy.



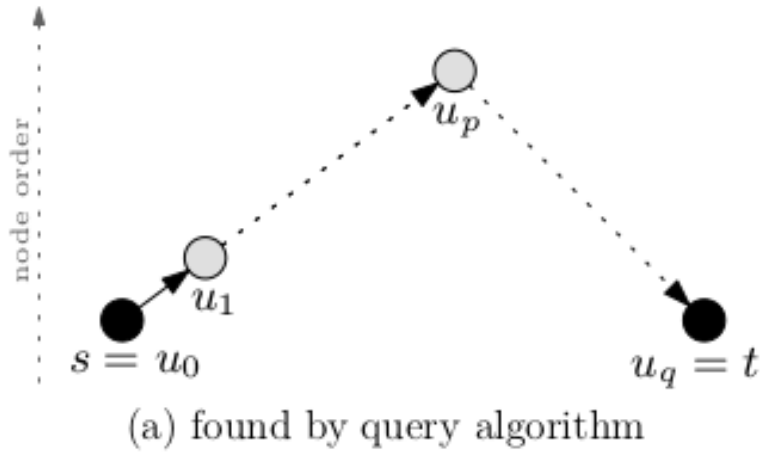
简介

- CH怎么做的？
  - 把点分层的另一种表示

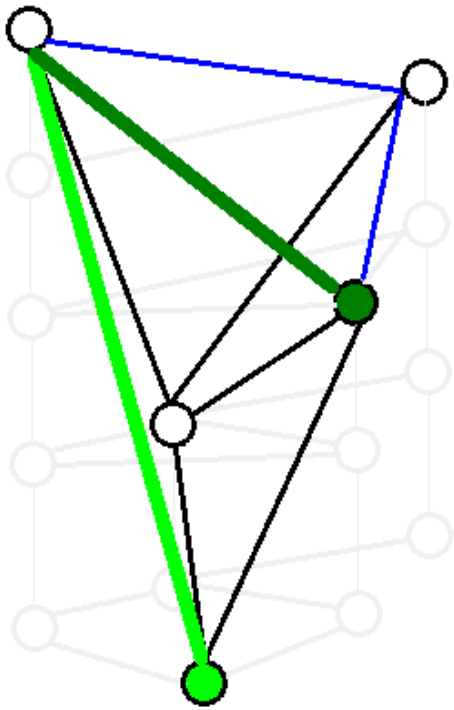


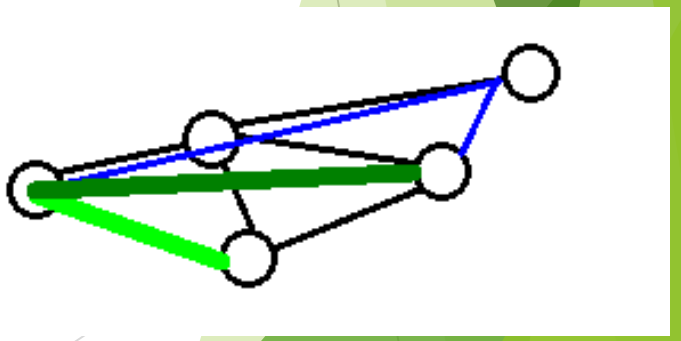
简介

- CH怎么做的？
  - 搜索时做双向Dijkstra搜索
  - 但有一个限制
    - 起点开始只往上层搜索，终点开始只往上层回溯



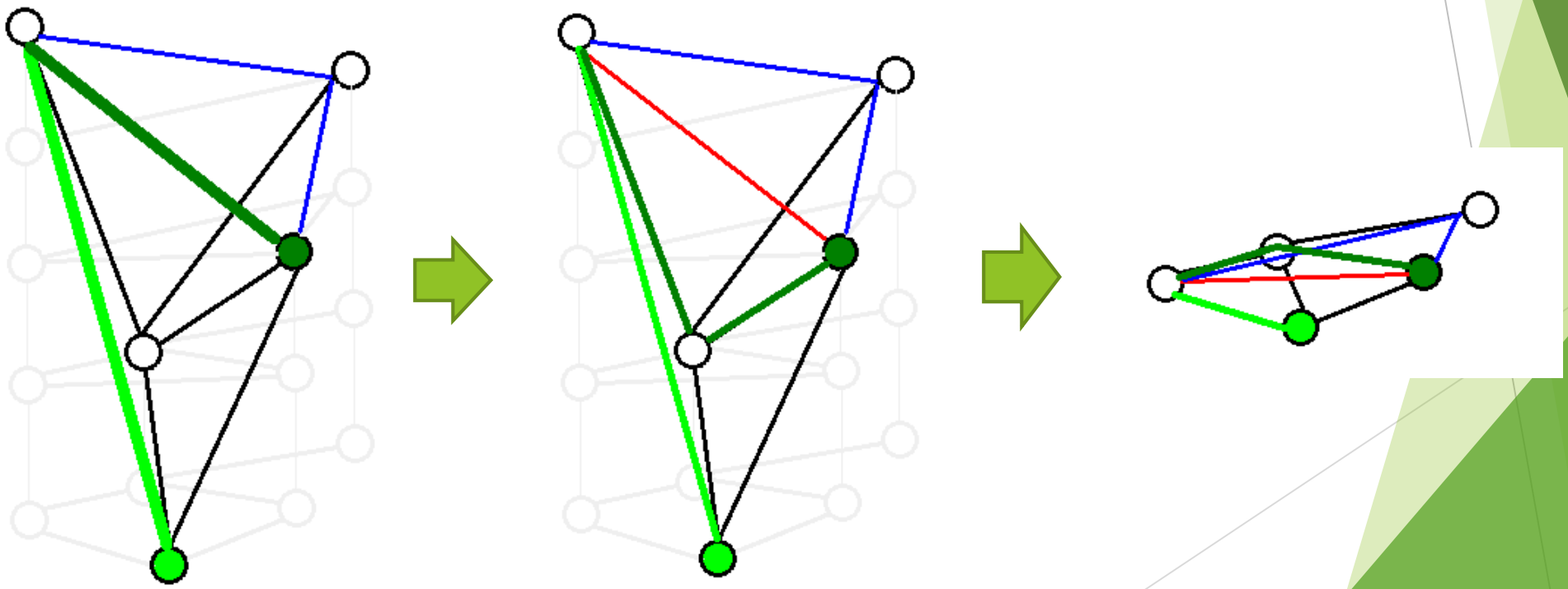
(a) found by query algorithm





简介

- CH怎么做的？
  - 得到最短路径后解压shortcuts得到对应原图的路径

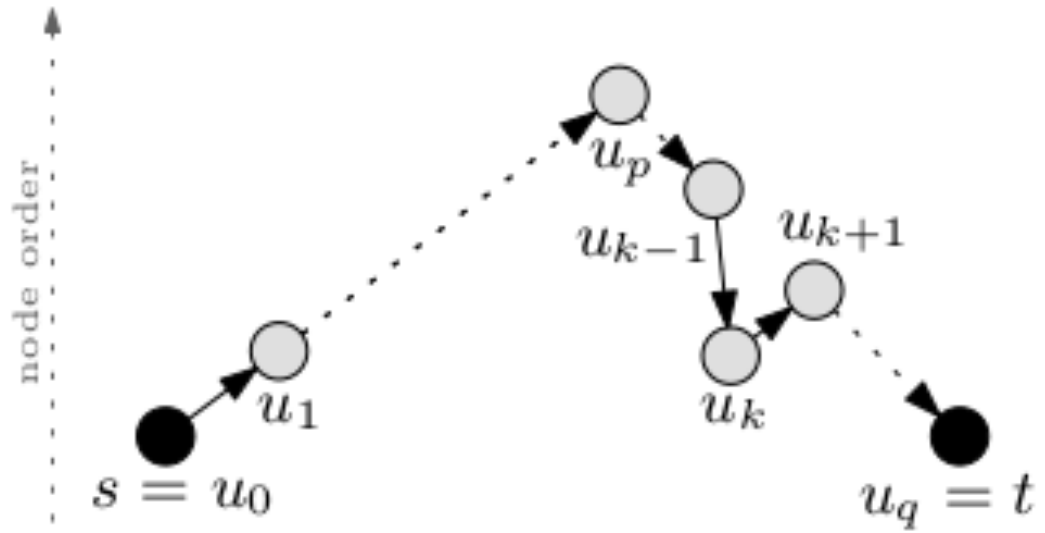


简介

- CH为什么是对的？
  - CH查询得到的是最短路的精确解
  - 为什么限定了搜索的方向还能够得到精确解？

简介

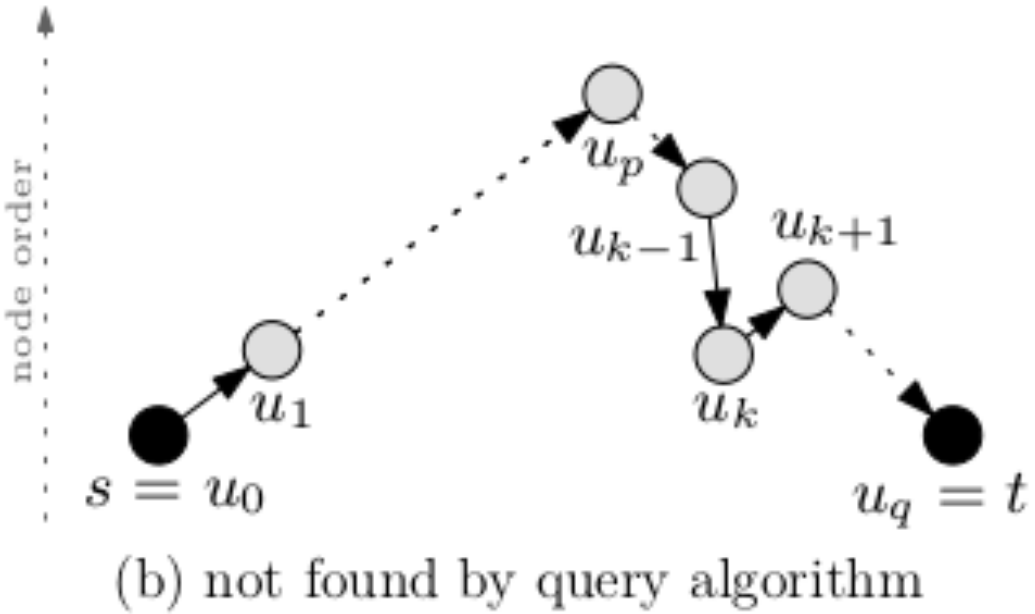
- CH为什么是对的？
  - 首先，因为CH保留了原图的所有边，加入的shortcut也不会使最短路变长，所以不考虑点序的搜索得到的一定是最优解
  - 我们把这种解画出来，把路径上的经过点的点序用高度表示



(b) not found by query algorithm

简介

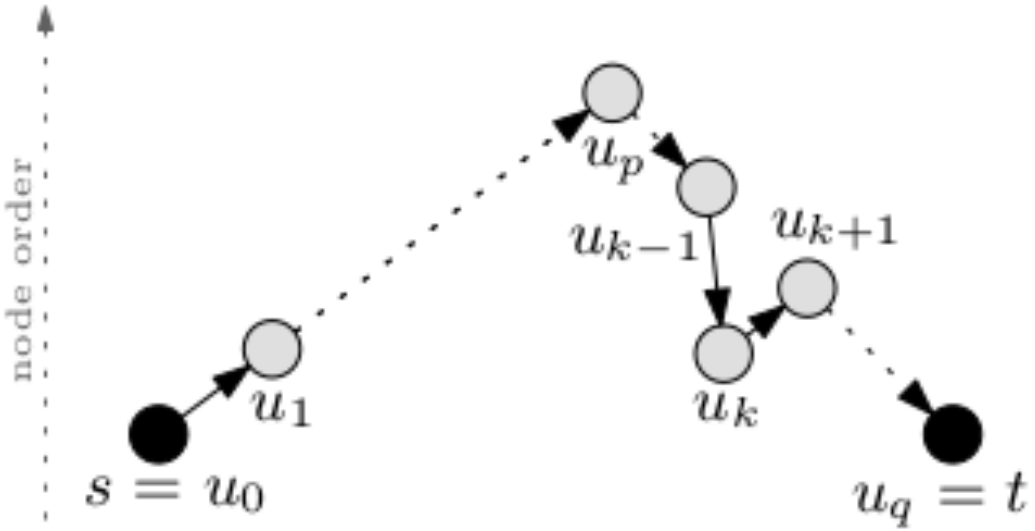
- CH为什么是对的？
  - 不失一般性的，我比较喜欢这么画



(b) not found by query algorithm

简介

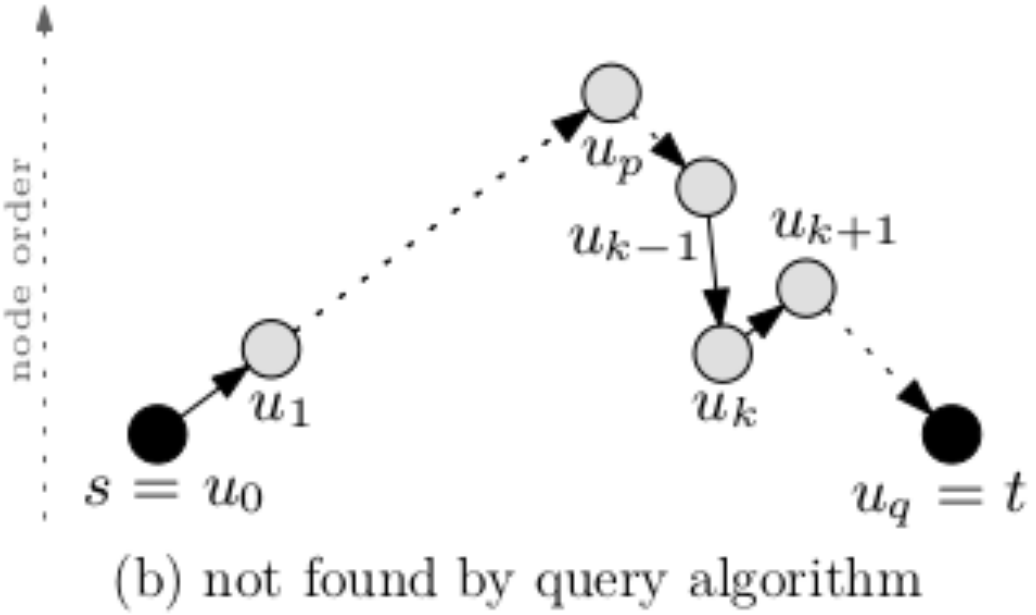
- CH为什么是对的？
  - 我们可以找到第一个不满足搜索条件的地方
  - 这种地方比较形象地，可以叫做“坑”



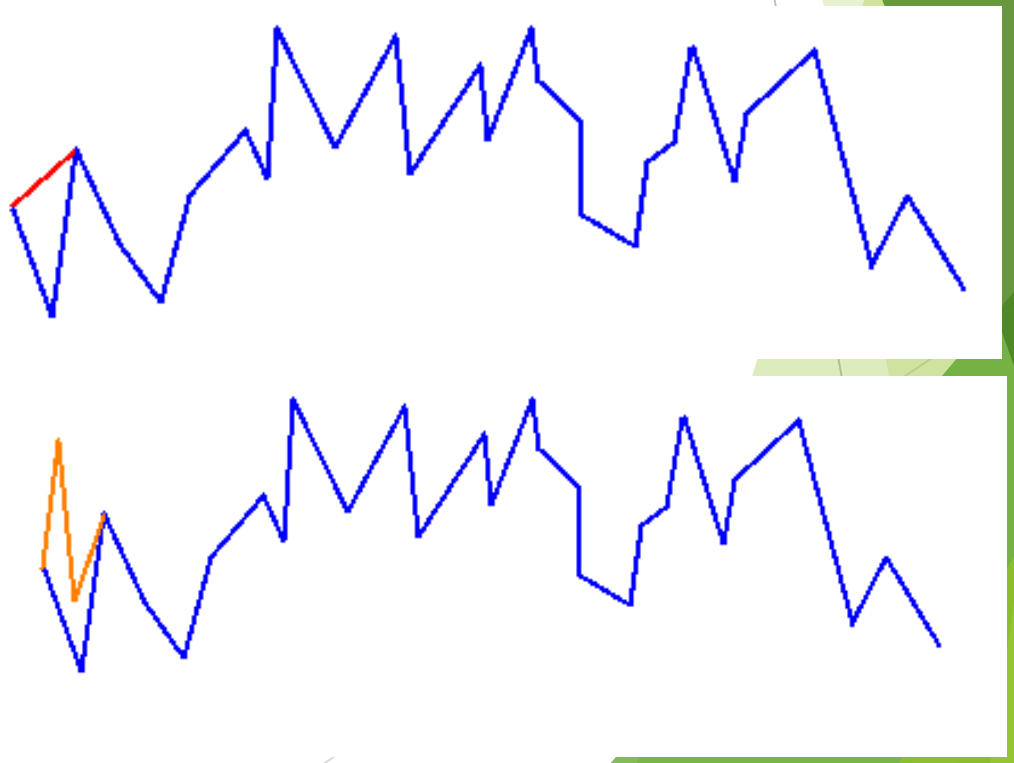
(b) not found by query algorithm

简介

- CH为什么是对的？
  - 加shortcut实际上就是“填坑”
  - “填坑”后的路径仍是一条最短路径
  - 如果最短路真的必须经过这个坑（v到w唯一的最短路），那么这个坑一定会被填上；否则这个坑就可以被绕开（其他路径）

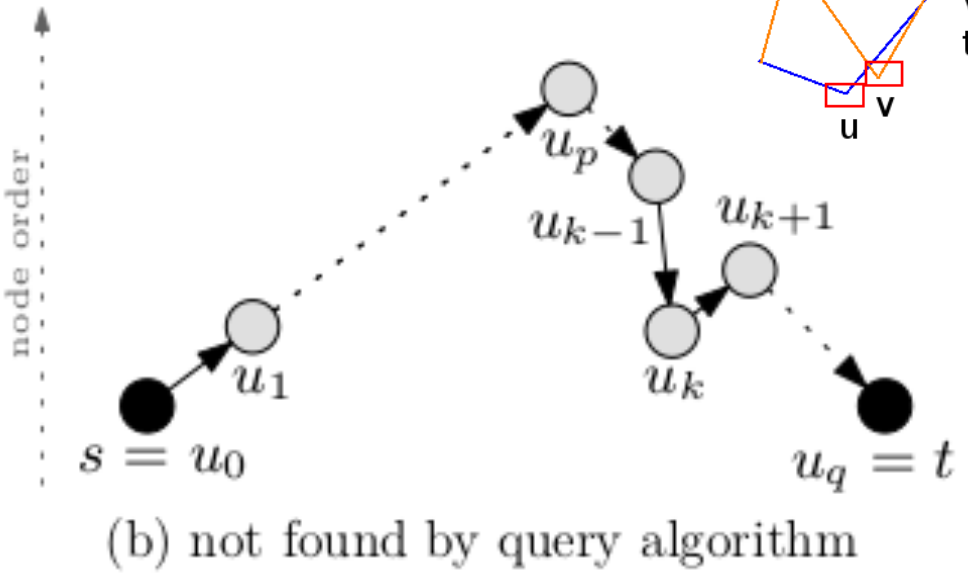


(b) not found by query algorithm



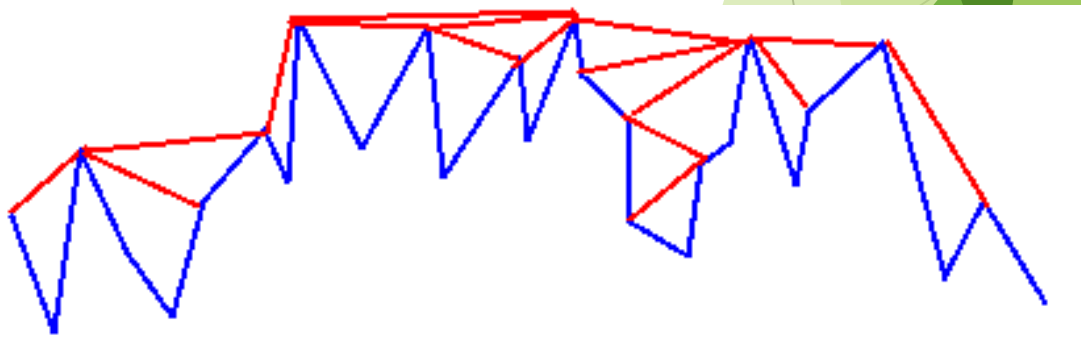
简介

- CH为什么是对的？
  - “绕开坑”后，路径上的最小点变大（CH不加shortcut的条件是有一条其他最短路，这条路径上的点都比点u大（因为它们都在点u之后缩(contract)）），所以能“绕开坑”的次数是有限的
  - 同理，“填坑”次数也是有限的
  - 等到再也绕不开的时候，“坑”肯定都能“填上”，就变成我们想要的形式了



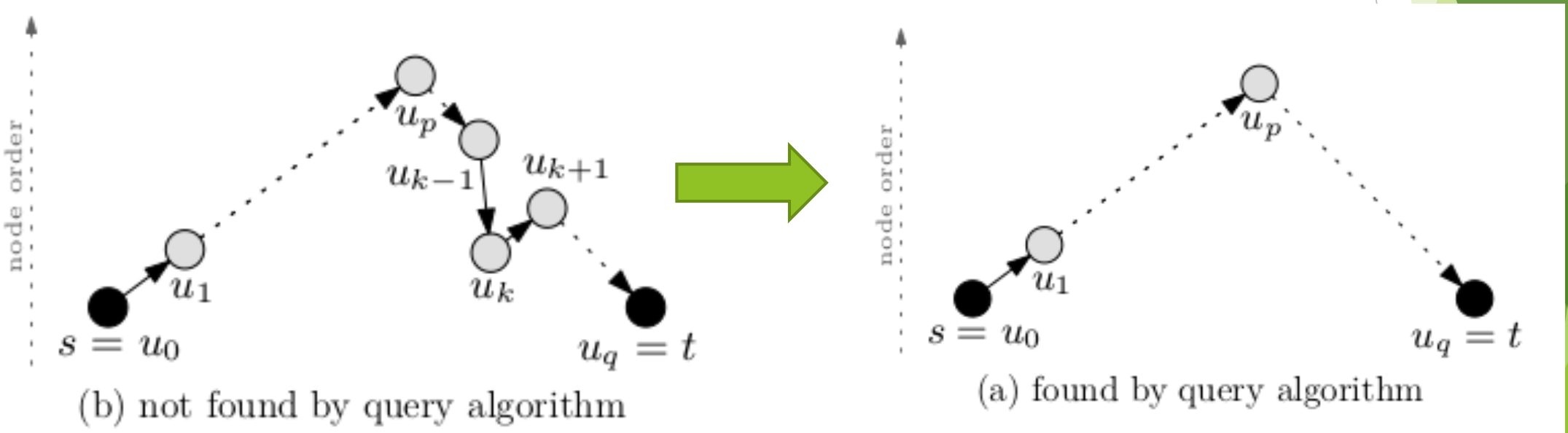
(b) not found by query algorithm

When node u is contracted, v still exists, so v is 'higher' than u.



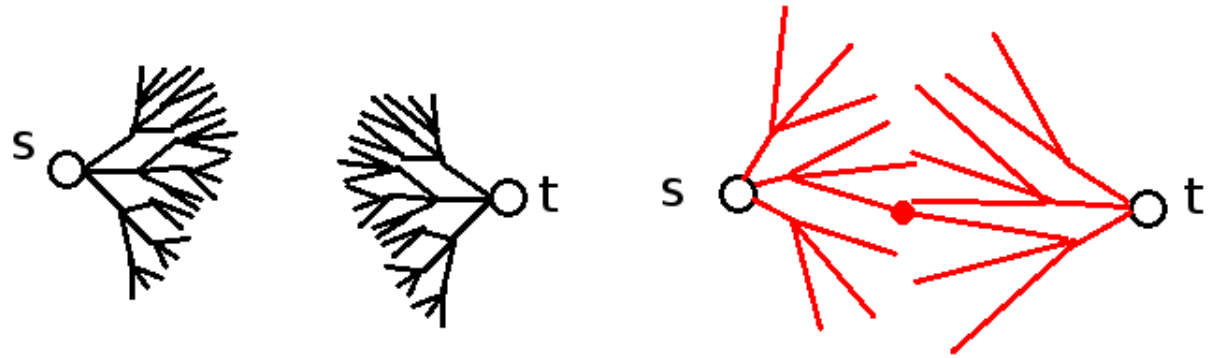
简介

- CH为什么是对的？
  - 所以存在一条从s到t的“从低到高到低”的最短路径
  - 这条路径会被查询算法找到，证毕



简介

- CH为什么快？
  - CH通过shortcut为Dijkstra算法提供了更“宽阔”的视野



  - 如果node order（点序）合适，搜索深度（从s到t经过的边数）有O(logn)的感觉，而非Dijkstra的O(n)

算法细节& 优化

算法细节&优化

- 回顾CH的预处理
  - 确定一个缩点的顺序
  - 每次按这个顺序选一个点u缩
    - 如果v->u->w可能是v到w的唯一的最短路，在原图中增加shortcut捷径(v,w)，权值为(v,u)和(u,w)的权值和
  - 删除点u

Algorithm 1: SimplifiedConstructionProcedure($G = (V, E), <$)

```
1 foreach u ∈ V ordered by < ascending do
2     foreach (v, u) ∈ E with v > u do
3         foreach (u, w) ∈ E with w > u do
4             if ⟨v, u, w⟩ "may be" the only shortest path from v to w then
5                 E := E ∪ {(v, w)} (use weight w(v, w) := w(v, u) + w(u, w))
```

算法细节&优化

- 回顾CH的预处理
  - 确定一个**缩点的顺序（顺序是什么？）**
  - 每次按这个顺序选一个点u缩
    - 如果v->u->w**可能是v到w的唯一的最短路（怎么确定？）**，在原图中增加shortcut捷径(v,w)，权值为(v,u)和(u,w)的权值和
  - 删除点u

Algorithm 1: SimplifiedConstructionProcedure($G = (V, E), <$)

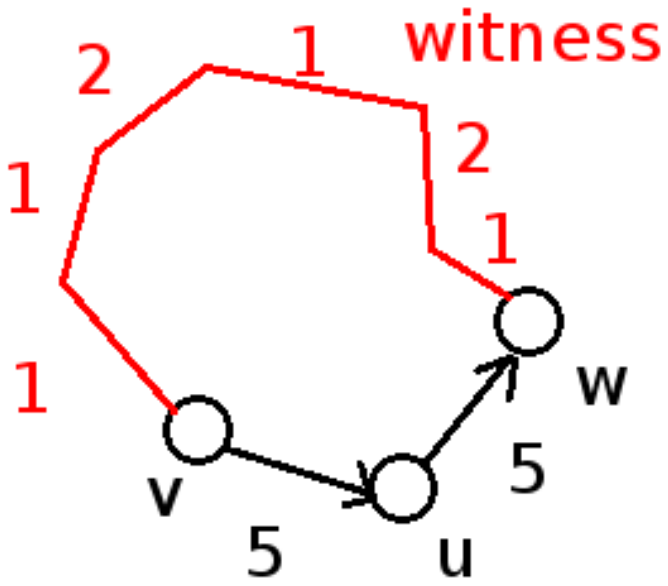
```
1 foreach u ∈ V ordered by < ascending do
2     foreach (v, u) ∈ E with v > u do
3         foreach (u, w) ∈ E with w > u do
4             if ⟨v, u, w⟩ "may be" the only shortest path from v to w then
5                 E:= E ∪ {(v, w)} (use weight w(v, w):= w(v, u) + w(u, w))
```

算法细节&优化

- Node order (缩点顺序)
- Witness path search (确定v->u->w是不是v到w惟一的最短路)

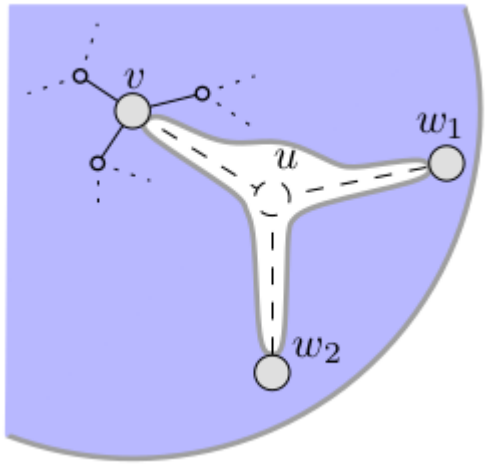
算法细节&优化

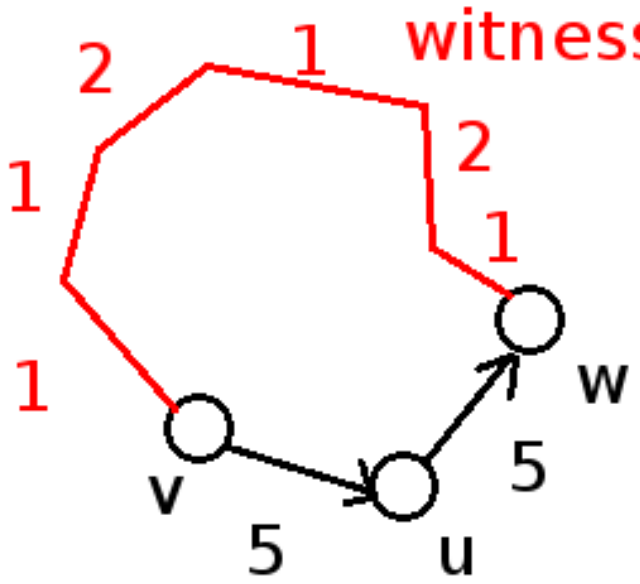
- Witness path search (确定v->u->w是不是v到w唯一的最短路)
  - 为什么叫witness path search ?
    - 如果找到一条v到w的其他的最短路或更短的路，那么v->u->w就不是v到w唯一的最短路了
    - 这条路经就是“v->u->w不是v到w唯一的最短路”这一命题的witness(证据)



算法细节&优化

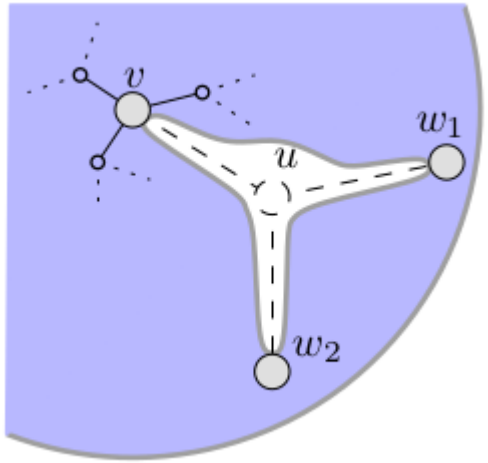
- Witness path search (确定v->u->w是不是v到w唯一的最短路)
  - 如何找witness path ?
  - 具体方法：
    - 在V/{u}构成的子图中运行最短路算法
    - 枚举v，w ，如果没有找到不长于v->u->w的路径，则增加边(v,w)
    - 删除点u

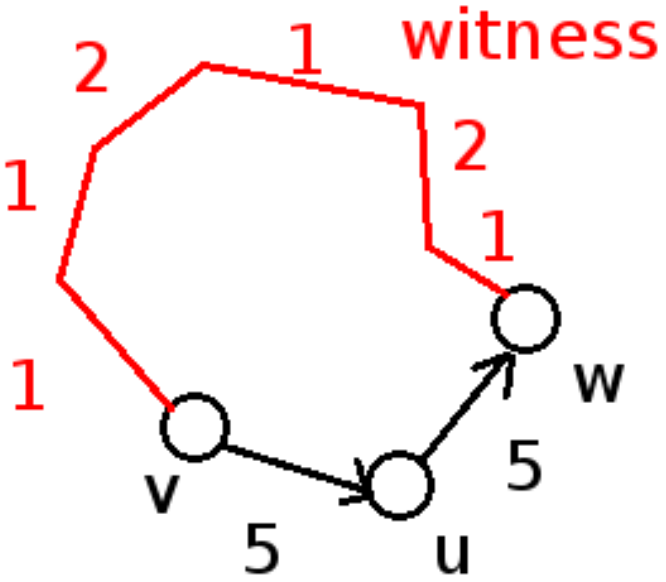




算法细节&优化

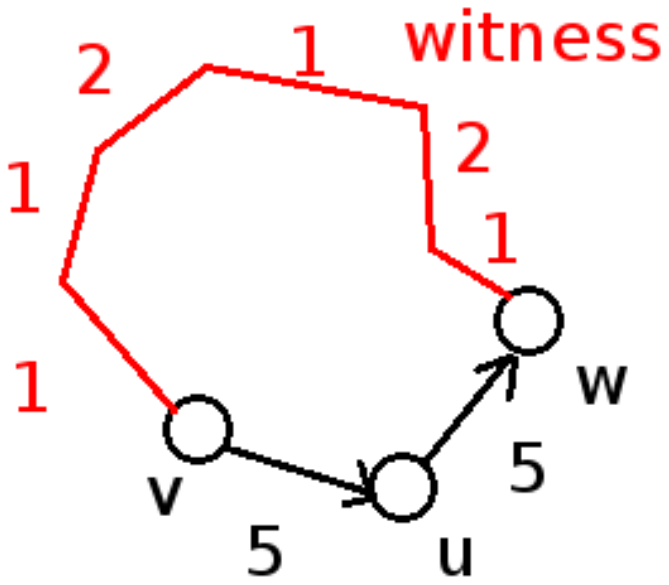
- Witness path search (确定v->u->w是不是v到w唯一的最短路)
  - 如何找witness path ?
  - 具体方法：
    - 在V/{u}构成的子图中运行最短路算法
    - 枚举v，w ，如果没有找到不长于v->u->w的路径，则增加边(v,w)
    - 删除点u
  - 这一过程称为预处理中的局部搜索





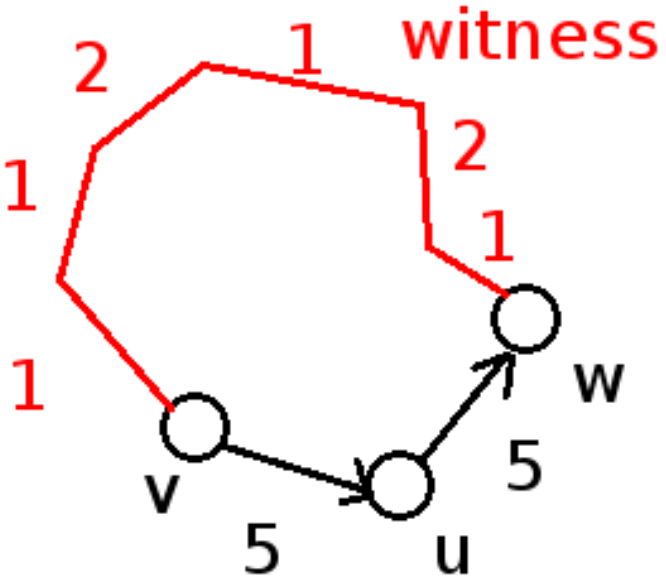
算法细节&优化

- Witness path search (确定v->u->w是不是v到w唯一的最短路)
  - 预处理需要找很多次witness path，Dijkstra太慢怎么办？



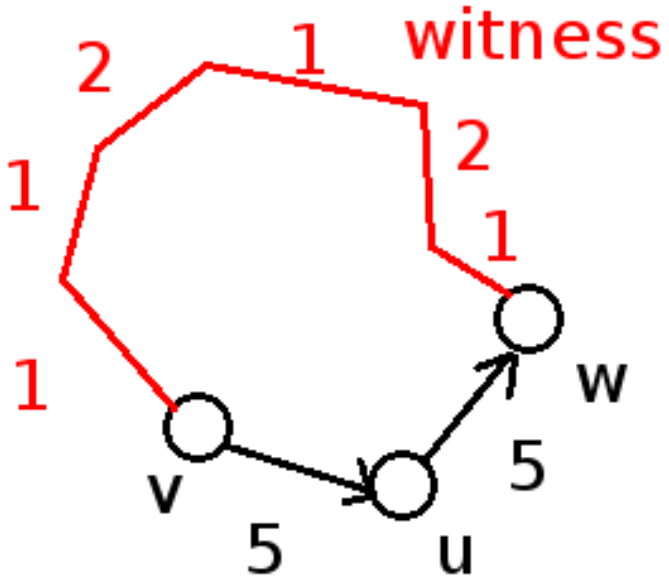
算法细节&优化

- Witness path search (确定v->u->w是不是v到w唯一的最短路)
  - 预处理需要找很多次witness path，Dijkstra太慢怎么办？
    - 可以用别的加速算法
    - 可以加常数优化
      - 比如从终点开始倒着开始搜一层，给所有点存下到w1,w2,w3,...,wk的距离，之后从起点开始搜的时候就可以少搜一层
    - 但更好的选择是……



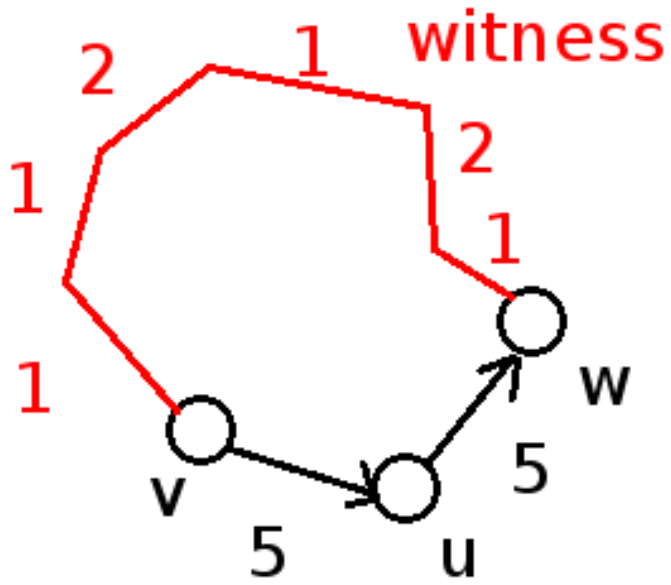
算法细节&优化

- Witness path search (确定v->u->w是不是v到w唯一的最短路)
  - 预处理需要找很多次witness path，Dijkstra太慢怎么办？
    - 实际上多加一条shortcut对查询结果的精确性是否会有影响？



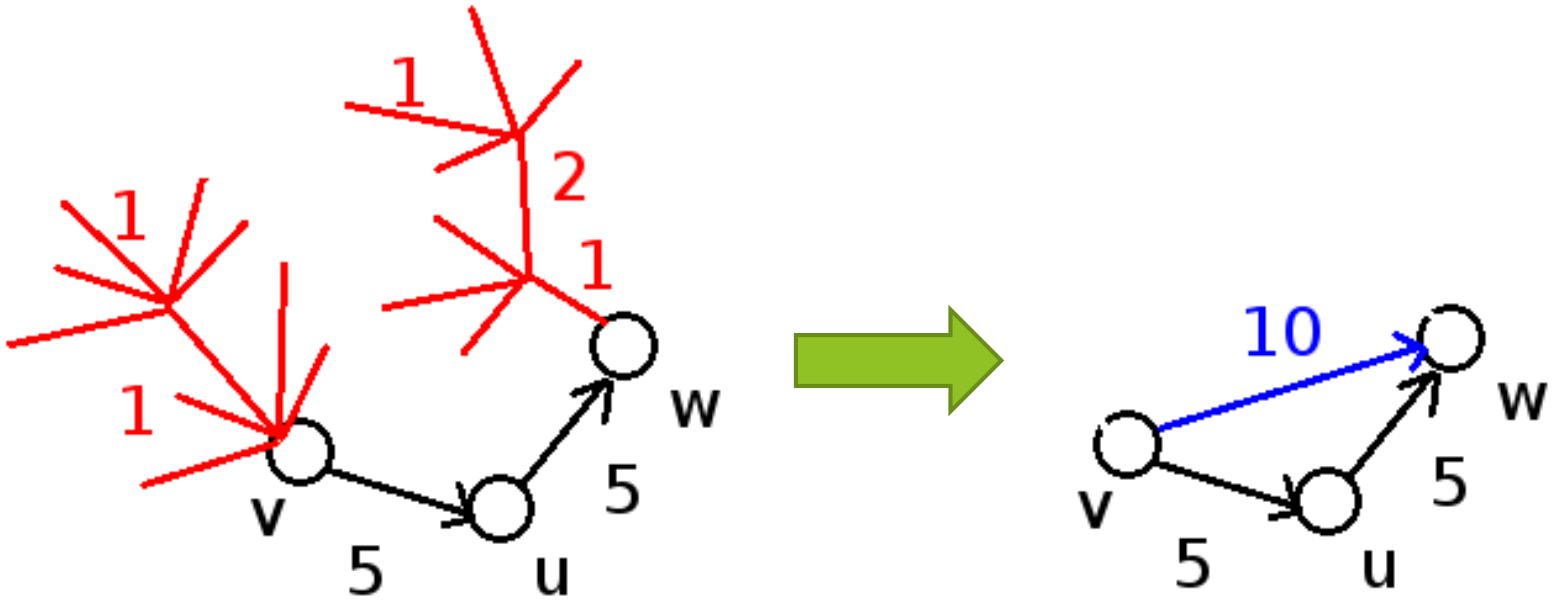
算法细节&优化

- Witness path search (确定v->u->w是不是v到w唯一的最短路)
  - 预处理需要找很多次witness path，Dijkstra太慢怎么办？
    - 实际上多加一条shortcut对查询结果的精确性是否会有影响？
      - 不会，最多图中的边变多，速度变慢，空间开销变大



算法细节&优化

- Witness path search (确定v->u->w是不是v到w唯一的最短路)
  - 预处理需要找很多次witness path，Dijkstra太慢怎么办？
    - 所以如果费力搜索还搜不到witness path，干脆放弃算了
    - 加一条shortcut损失也不大



算法细节&优化

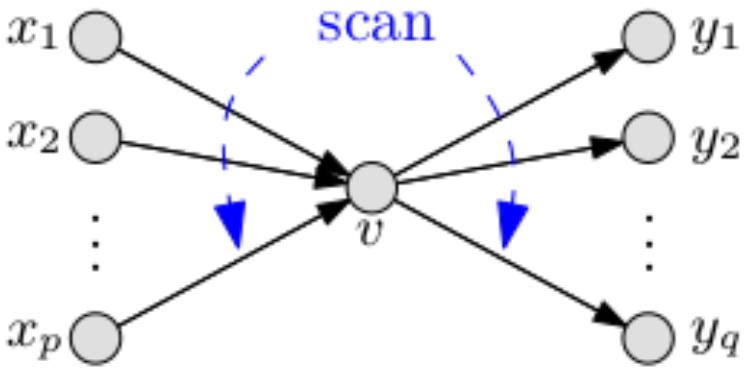
- Witness path search (确定v->u->w是不是v到w唯一的最短路)
  - 预处理需要找很多次witness path，Dijkstra太慢怎么办？
    - 限制搜索范围的几种方法
      - 限制搜索点数
      - 限制搜索深度

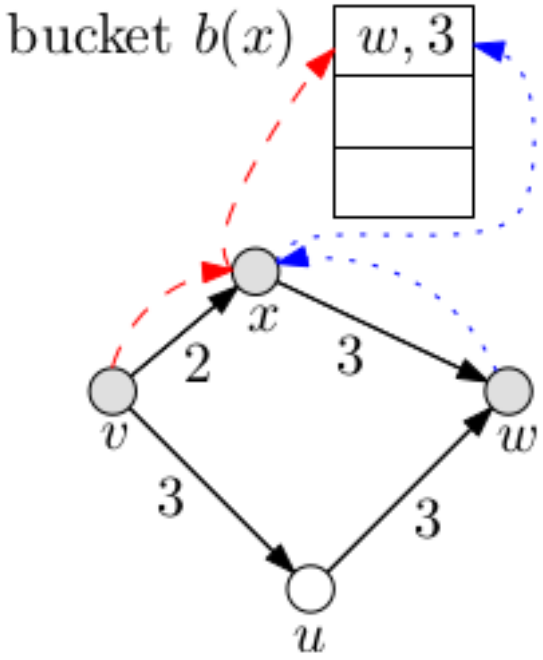
Limit the number of settled nodes. A local search, implemented as a modified Dijkstra algorithm, can be stopped after a certain number of nodes is settled.

Limit the number of hops / edges from the start node. Only find shortest-paths with a limited number of edges. We will call this *hop limit*.

算法细节&优化

- Witness path search (确定v->u->w是不是v到w唯一的最短路)
  - 预处理需要找很多次witness path，Dijkstra太慢怎么办？
    - 限制搜索范围的几种方法
      - CH针对搜索深度为1和2的情况特别优化过了





算法细节&优化

- Witness path search (确定v->u->w是不是v到w唯一的最短路)
  - 预处理需要找很多次witness path，Dijkstra太慢怎么办？
    - 实践证明，搜索深度到6已经足够好了（限制搜索点个数为1000）

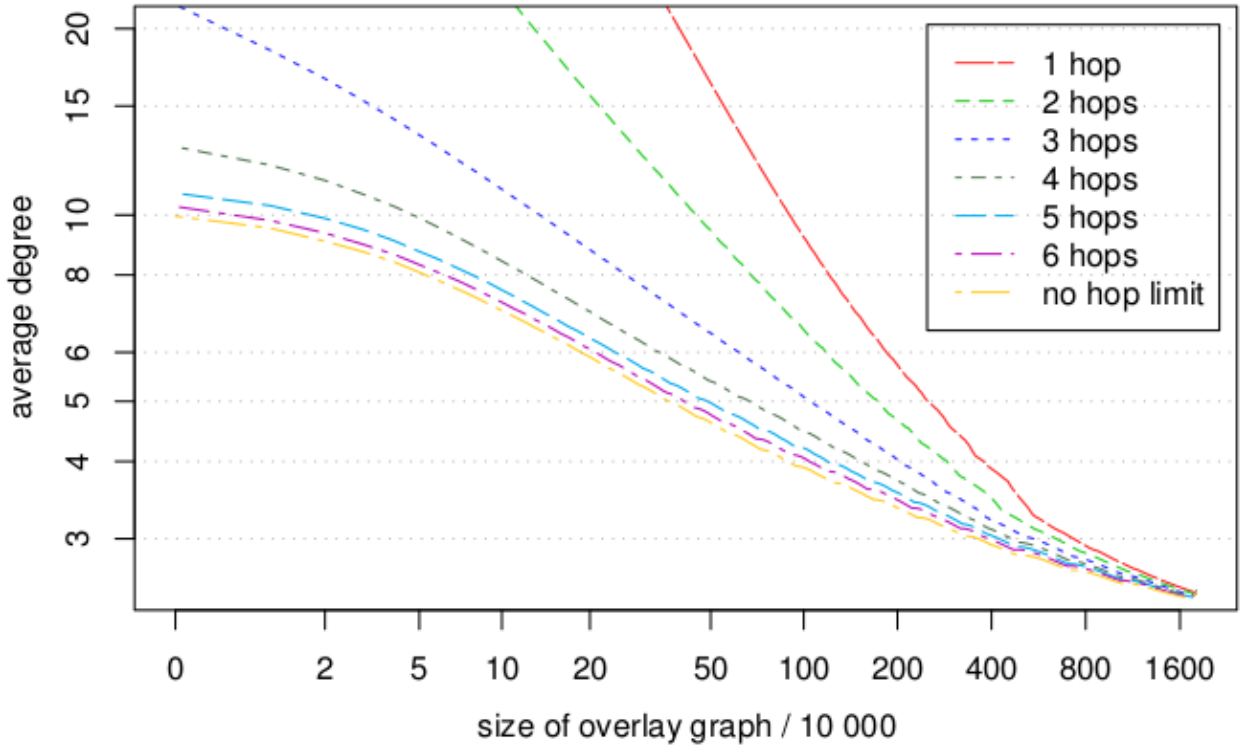


Figure 26: Average degree development for different hop limits.

算法细节&优化

- Witness path search (确定v->u->w是不是v到w唯一的最短路)
  - 进一步的优化
    - 在contraction的前期用比较小的搜索深度，之后提高
    - 如何衡量contraction进度？
      - 图的平均点度数

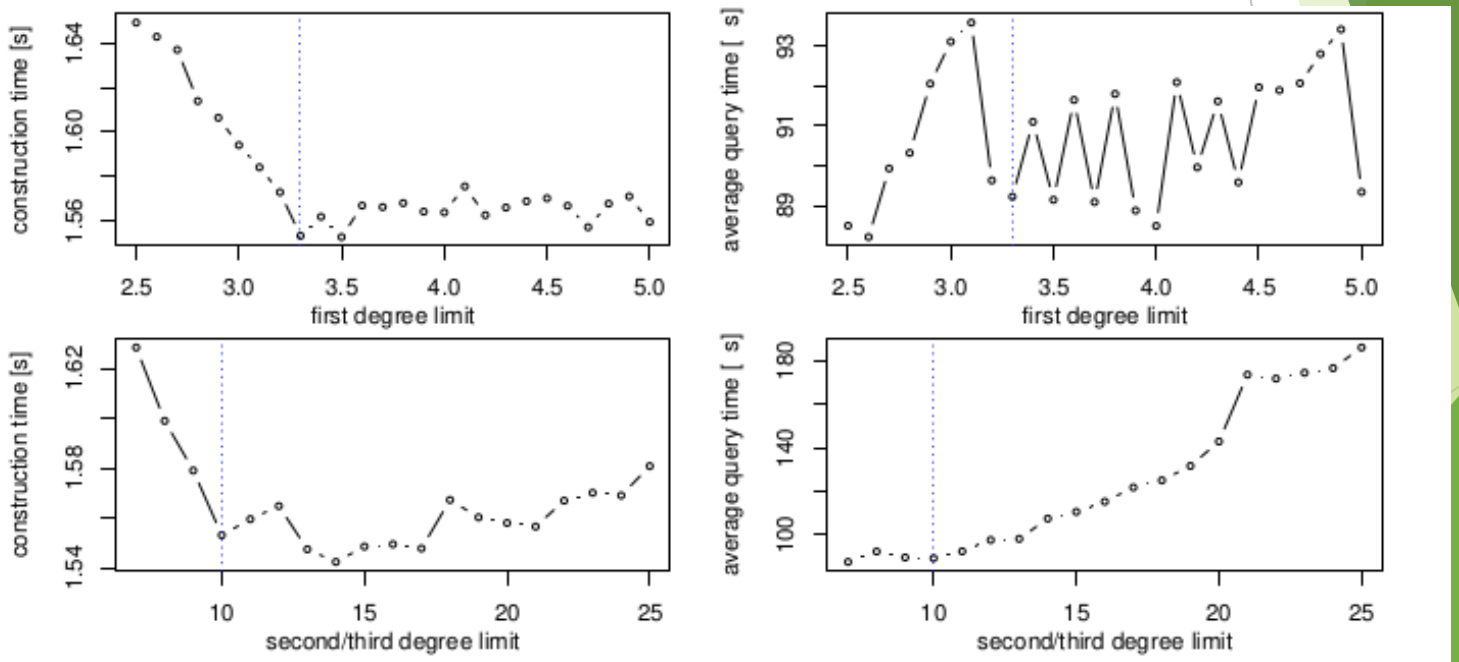


Figure 27: Sensitivity analysis with road network of Belgium. Parameters used: edge difference multiplier 190, search space multiplier 1, contracted neighbors multiplier 120, original edges term multiplier 70, hop@degree limits: 1@3.3, 2@10, 3@10, 5

算法细节&优化

- ---------- 华丽的分割线 ----------

算法细节&优化

- 回顾CH的预处理
  - 确定一个**缩点的顺序（顺序是什么？）**
  - 每次按这个顺序选一个点u缩
    - 如果v->u->w**可能是v到w的唯一的最短路（怎么确定？）**，在原图中增加shortcut捷径(v,w)，权值为(v,u)和(u,w)的权值和
  - 删除点u

Algorithm 1: SimplifiedConstructionProcedure($G = (V, E), <$)

```
1 foreach u ∈ V ordered by < ascending do
2     foreach (v, u) ∈ E with v > u do
3         foreach (u, w) ∈ E with w > u do
4             if ⟨v, u, w⟩ "may be" the only shortest path from v to w then
5                 E:= E ∪ {(v, w)} (use weight w(v, w):= w(v, u) + w(u, w))
```

算法细节&优化

- Node order (缩点顺序)
- Witness path search (确定v->u->w是不是v到w惟一的最短路)

算法细节&优化

- Node order (缩点顺序)
  - 启发式规则
  - 几个term（项）线性加权组合
  - 数值最小的先contract（缩）

eventually meet at the most important node on a shortest path. We use a *simple* but *extensible* heuristic to obtain the node order: a priority queue whose priority function for each node is a linear combination of several terms, e.g. one term weights nodes depending on the sparsity of the remaining graph after the contraction. Another term regards the

  - 手调权值

算法细节&优化

- Node order (缩点顺序)
  - 两个问题
    - 论文中用到了哪些term？
    - 每一个term（项）的数值在预处理过程中是固定的吗？

算法细节&优化

- Node order (缩点顺序)
  - 论文中用到了哪些term决定node order?
  - Edge difference（边数差）
  - Cost of contraction（缩点代价）
  - Uniformity（均匀性）
  - Cost of queries（查询代价）
  - Global measures（全局特征）

算法细节&优化

- Node order (缩点顺序)
  - Edge difference（边数差）
    - 缩完点加了几条边（可以是负数）
  - New edges（新边）
    - 缩完点新加了几条边

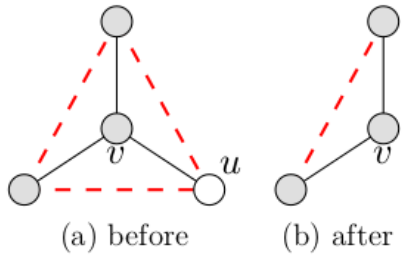


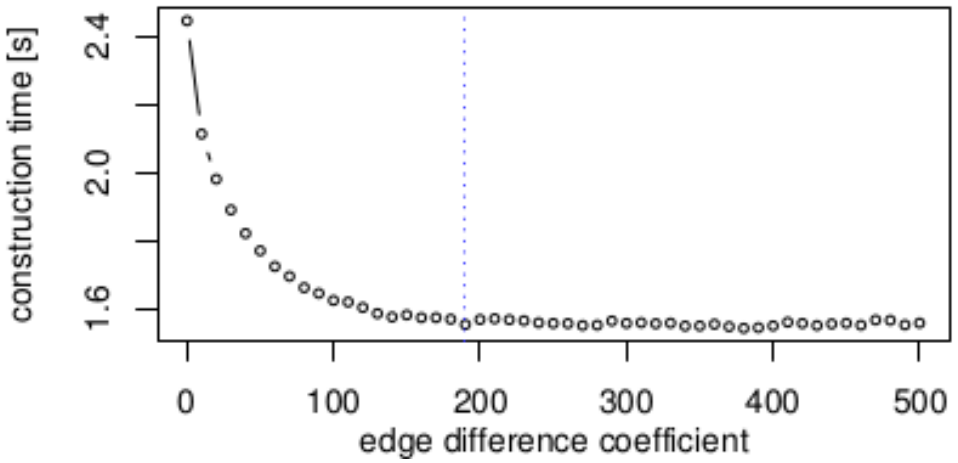
Figure 4: After the contraction of node u, the edge difference and the number of new edges of a neighbor v changes from $3-3=0$ to $1-2=-1$. Note that an undirected edge is counted twice.

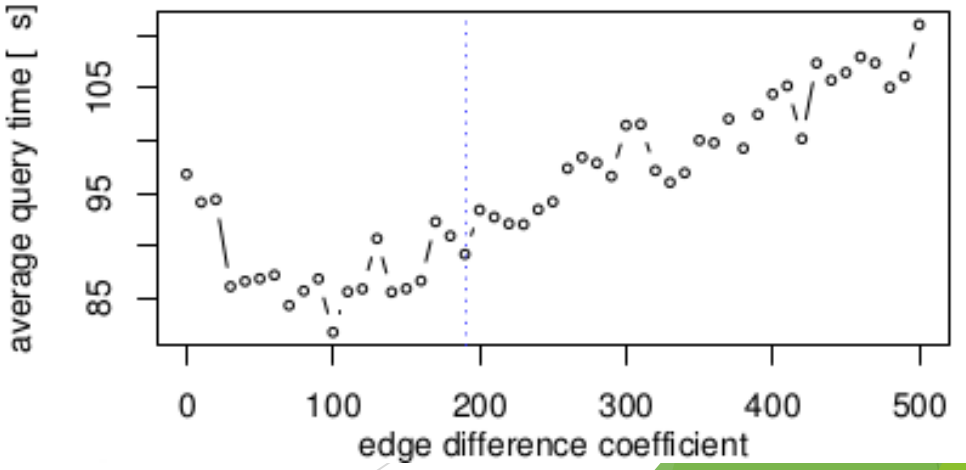
Edge difference. The edge difference is calculated between the two graphs G' and G''. This is the most important parameter to get a good contraction, otherwise the remaining graph will most likely converge to the complete graph.

New edges. An edge is a new edge if and only if it needs to be added to G''. Note that not all shortcut edges are new edges, because sometimes just the edge weight function changes. This parameter did not prove to be useful and we omitted it in the presented experiments for the sake of simplicity.

算法细节&优化

- Node order (缩点顺序)
  - Edge difference（边数差）
    - 缩完点加了几条边（可以是负数）
  - 最重要的term（项）
  - 目的：
    - 使缩完点后图更加稀疏，防止变成完全图
    - 从而间接提高预处理速度，降低空间开销





算法细节&优化

- Node order (缩点顺序)
  - Edge difference（边数差）
    - 缩完点加了几条边（可以是负数）
  - 实际实现可以使用缩点前后的空间差，顺便把空间复杂度也考虑进去
  - 只用这一项就已经比之前HNR的实现要快了

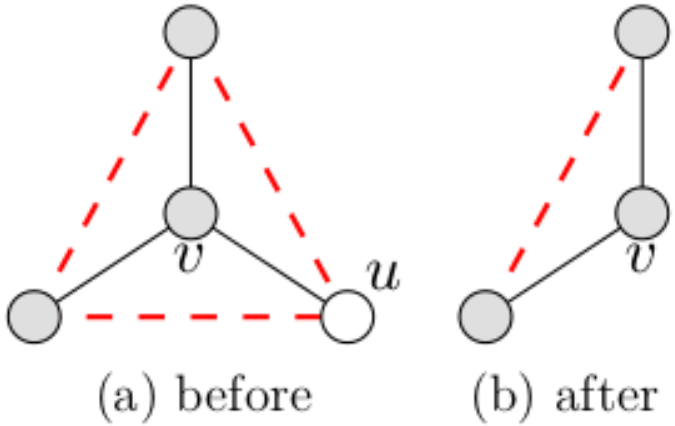


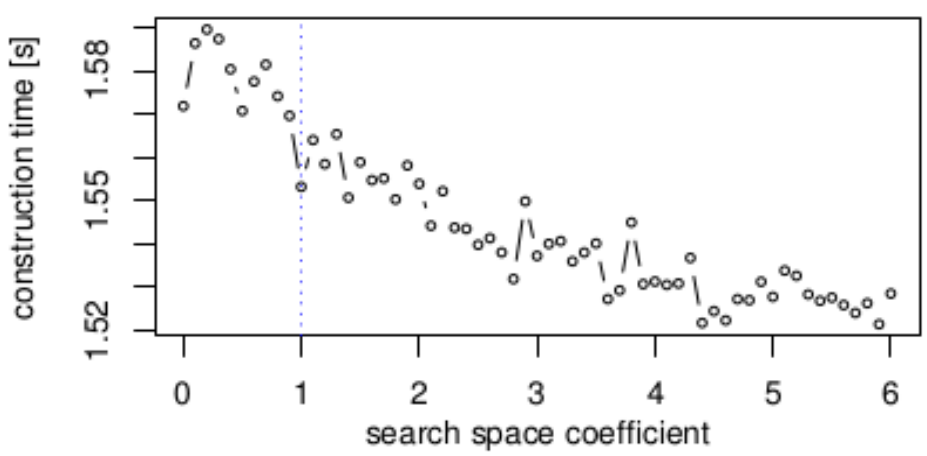
Figure 4: After the contraction of node u, the edge difference and the number of new edges of a neighbor v changes from $3 - 3 = 0$ to $1 - 2 = -1$. Note that an undirected edge is counted twice.

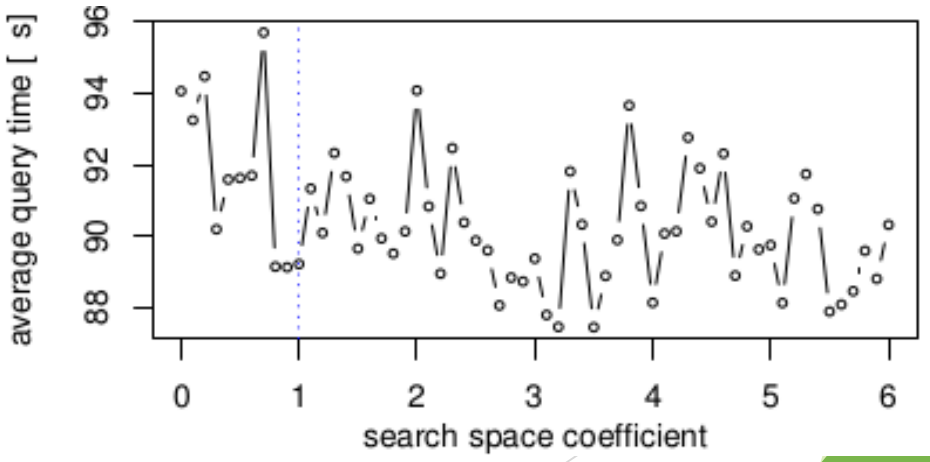
算法细节&优化

- Node order (缩点顺序)
  - 论文中用到了哪些term决定node order?
  - Edge difference（边数差）
  - Cost of contraction（缩点代价）
  - Uniformity（均匀性）
  - Cost of queries（查询代价）
  - Global measures（全局特征）

算法细节&优化

- Node order (缩点顺序)
  - Cost of contraction（缩点代价）
    - 加shortcut时做的局部搜索的搜索空间大小，具体就是局部搜索结束时固定的点数，和局部搜索的策略有关
  - 目的：
    - 一般来说，搜索空间越小代表这个点越“偏僻”
    - 先缩搜索空间小的点可以提高预处理速度



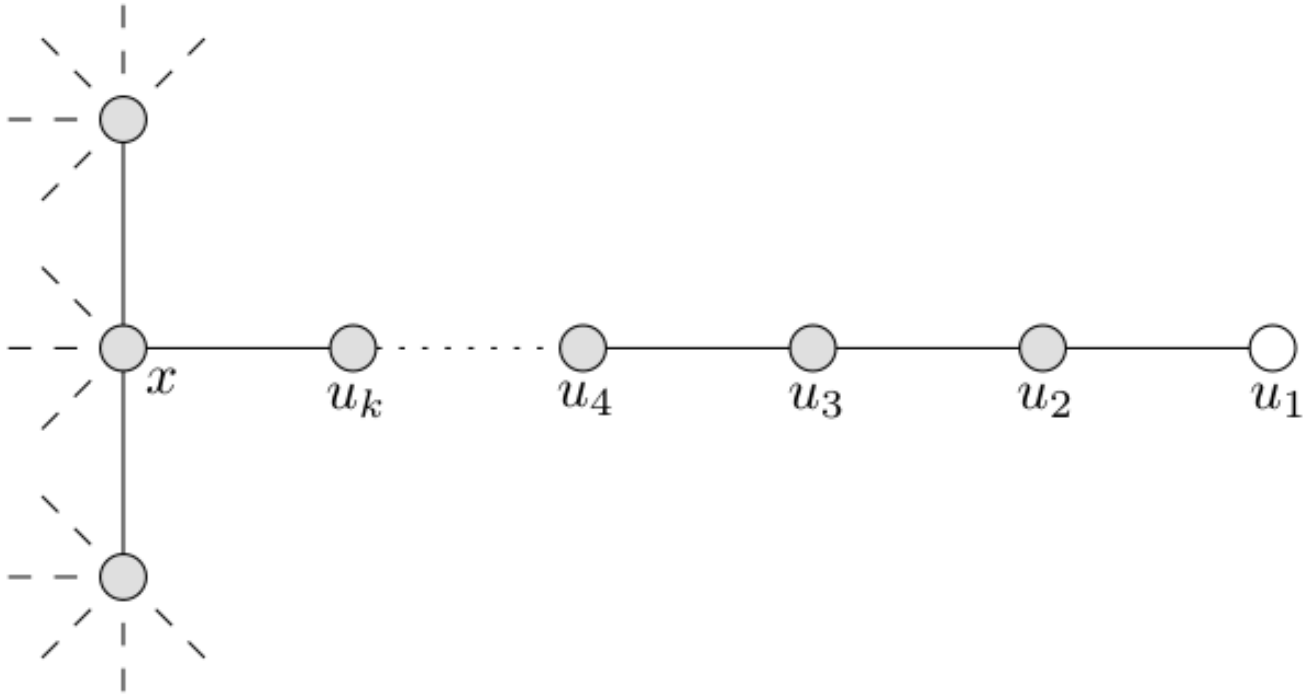


算法细节&优化

- Node order (缩点顺序)
  - 论文中用到了哪些term决定node order?
  - Edge difference（边数差）
  - Cost of contraction（缩点代价）
  - Uniformity（均匀性）
  - Cost of queries（查询代价）
  - Global measures（全局特征）

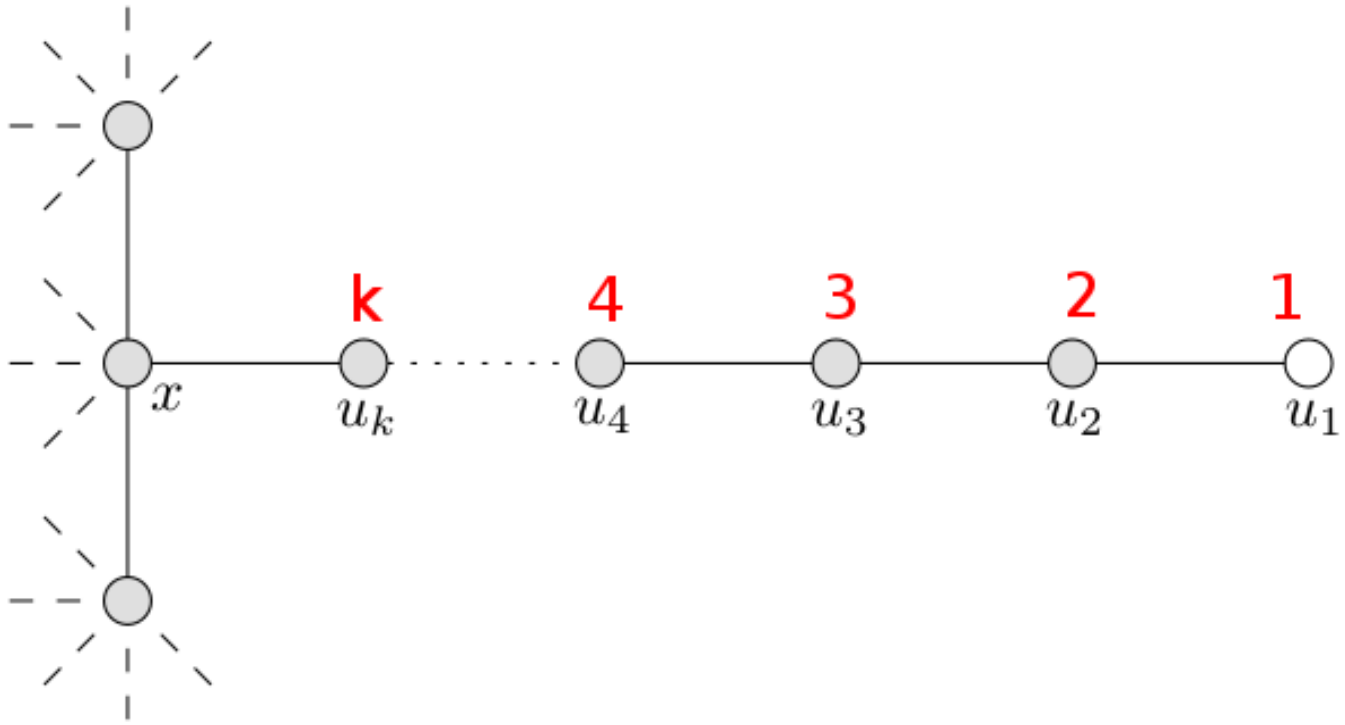
算法细节&优化

- Node order (缩点顺序)
  - Uniformity（均匀性）
    - 我们想要每次被contract(缩)的点更均匀，这样得到的hierarchy（层次）更利于加速搜索
  - 一种情况



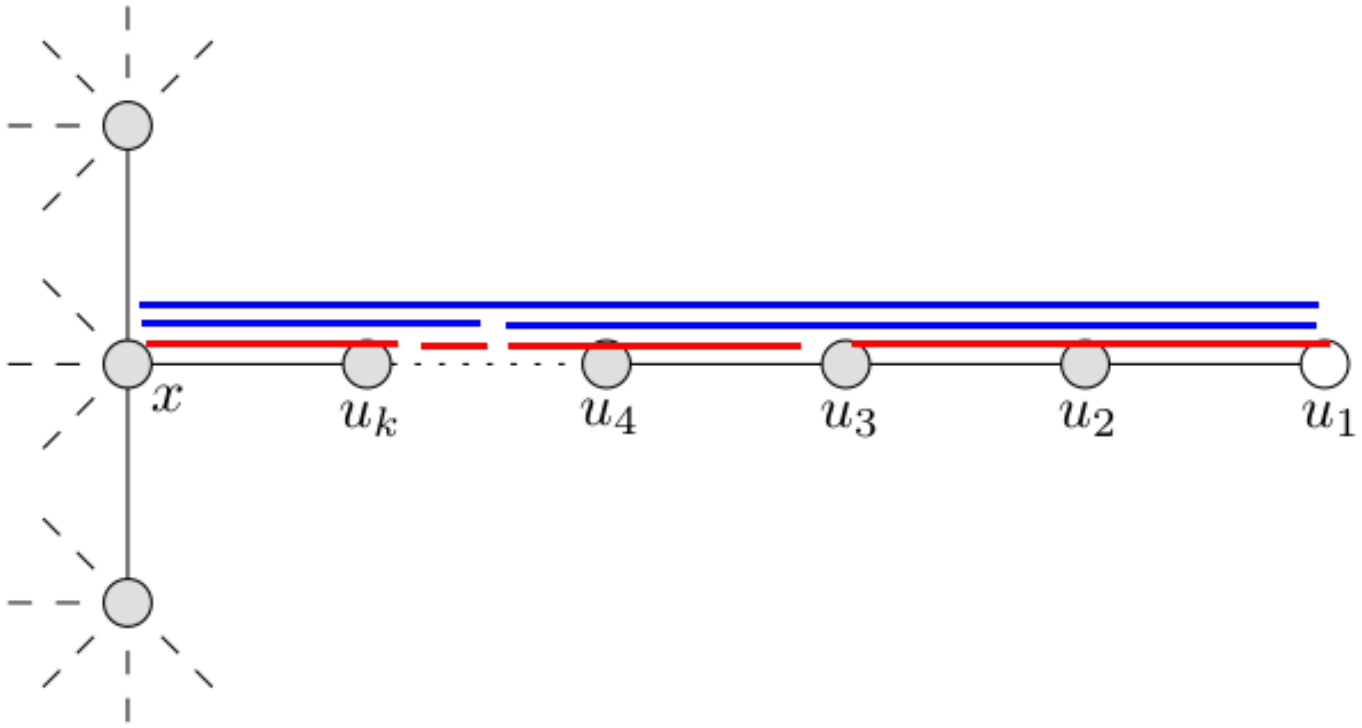
算法细节&优化

- Node order (缩点顺序)
  - Uniformity（均匀性）
    - 不好的node order



算法细节&优化

- Node order (缩点顺序)
  - Uniformity（均匀性）
    - 好的node order



算法细节&优化

- Node order (缩点顺序)
  - Uniformity（均匀性）
  - 具体的term（项）
    - term1: 曾经被contract（缩）过的邻居数
    - term2: 新加shortcut在原图中对应的边数
    - term3: Voronoi图（细胞图）对应的区域中被contract过的点数

算法细节&优化

- Node order (缩点顺序)
  - Uniformity（均匀性）
    - term1: 曾经被contract（缩）过的邻居数
    - term2: 新加shortcut在原图中对应的边数

Contracted neighbors. Count for each node the number of previous neighbors that have been contracted.

Sum of original edges of the new shortcuts. Count the number of original edges of the newly added shortcuts during the contraction of a node, see Figure 8. For convenience, we will refer to this term as the *original edges term*.

算法细节&优化

- Node order (缩点顺序)
  - Uniformity（均匀性）
    - term1: 曾经被contract（缩）过的邻居数
    - term2: 新加shortcut在原图中对应的边数
  - 目的（除均匀性外）：
    - 考虑edge difference（term2一定程度上考虑了）
    - 更少的空间消耗（？？？）
  - 缺点：
    - term2在稠密图中会造成比较大的空间开销，慎用

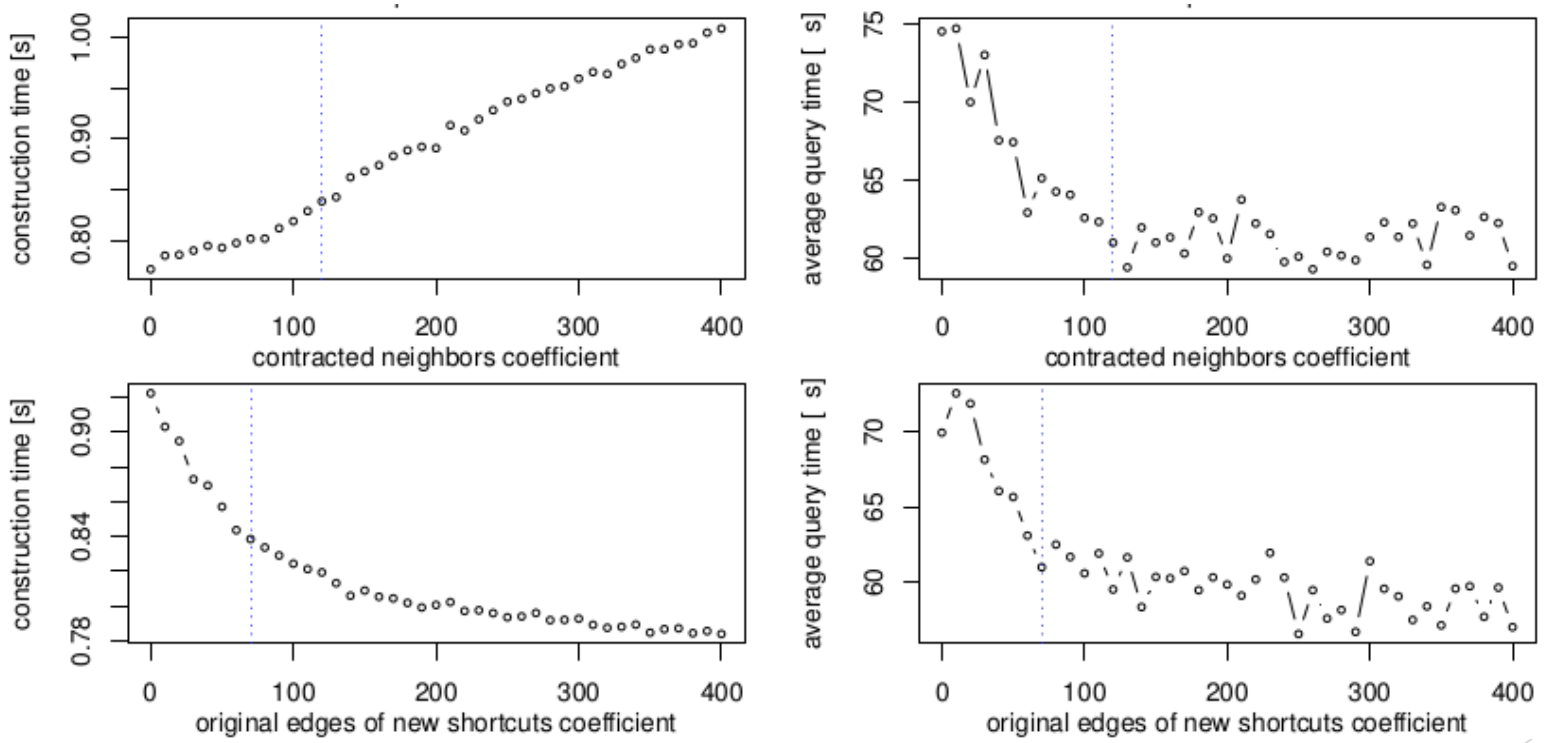
算法细节&优化

- Node order (缩点顺序)
  - Uniformity（均匀性）
    - term1: 曾经被contract（缩）过的邻居数
    - term2: 新加shortcut在原图中对应的边数
  - 目的（除均匀性外）：
    - 考虑edge difference（term2一定程度上考虑了）
    - 更少的空间消耗（？？？）
  - 缺点：
    - term2在稠密图中会造成比较大的空间开销，慎用

only a few original edges will hopefully lead to a hierarchy with this property. And fast path unpacking routines that store complete representations of shortcuts may also benefit from this new priority term. Their memory consumption is lowered and that is especially important on mobile devices.

算法细节&优化

- Node order (缩点顺序)
  - Uniformity（均匀性）
    - term1: 曾经被contract（缩）过的邻居数
    - term2: 新加shortcut在原图中对应的边数



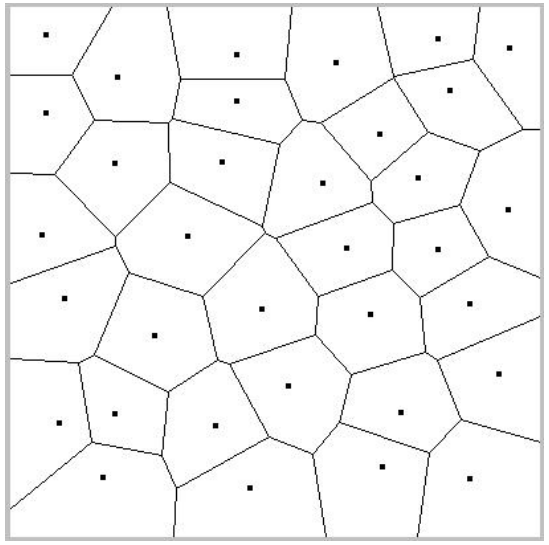
算法细节&优化

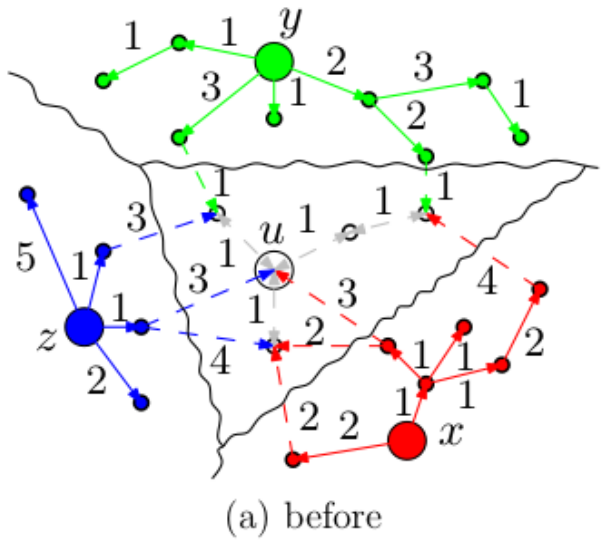
- Node order (缩点顺序)
  - Uniformity（均匀性）
    - term3: Voronoi图（细胞图）对应的区域中被contract过的点数
    - 原因：
      - 只算邻居数太特(min)定(ke)了，我们需要更通(ke)用(xue)的做法

> Counting contracted neighbors is simple to implement but it seems a little bit ad-hoc to ensure uniform contraction. The next approach uses something more specific to this task.

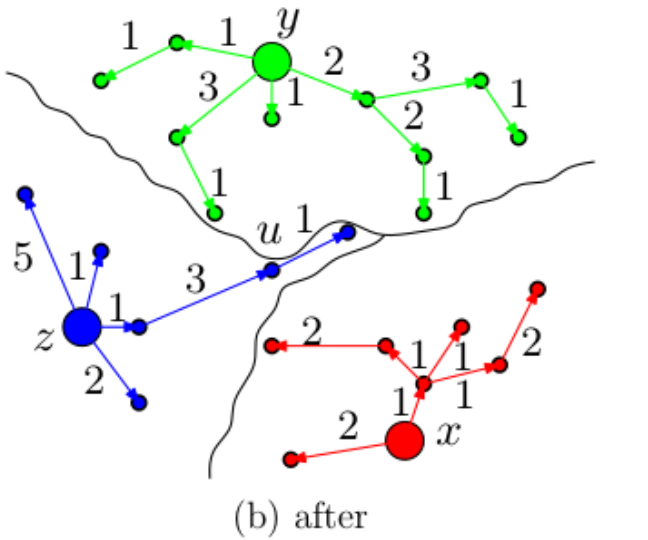
算法细节&优化

- Node order (缩点顺序)
  - Uniformity（均匀性）
    - Voronoi图和作者口中的Voronoi图





(a) before



(b) after

算法细节&优化

- Node order (缩点顺序)
  - Uniformity（均匀性）
    - 原图中v能到的且离v最近（比其他还没被contract的点近）的点构成v的Voronoi区域
    - 计算v的Voronoi区域中被contract过的点数，取根号

Definition 1. *Let u be currently the highest contracted node, so all $v \in G$ with $v \leq u$ are contracted and all $v \in G$ with $v > u$ are not contracted. The Voronoi region of $v \in G$, $v > u$ is defined as follows:*

$$R(v) := \{x \in G \mid x \leq u \text{ and } d(v,x) < \infty \text{ and } \forall y \in G, y > u \Rightarrow d(v,x) \leq d(y,x)\} \cup \{v\}$$

Voronoi region. Count the number of already contracted nodes in each Voronoi region $R(u)$ of a remaining node u. Because the number of nodes in the Voronoi regions of the remaining nodes grows quite fast, we also extract the square root before we add this property to the linear combination of the priority. If we think of the Voronoi region as a disc and of the number of nodes as its area, extracting the square root calculates something like the diameter of the disc.

算法细节&优化

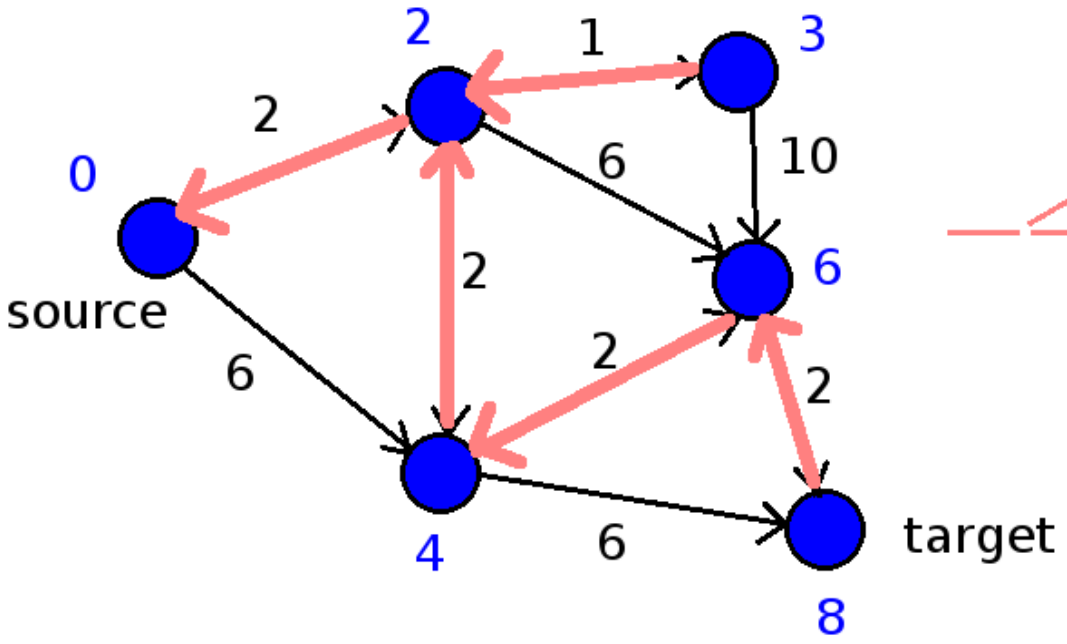
- Node order (缩点顺序)
  - Uniformity（均匀性）
    - term3: Voronoi图（细胞图）对应的区域中被contract过的点数
    - 缺点：
      - 相比计算被contract过的邻居数，需要的预处理时间更长

算法细节&优化

- Node order (缩点顺序)
  - 论文中用到了哪些term决定node order?
  - Edge difference（边数差）
  - Cost of contraction（缩点代价）
  - Uniformity（均匀性）
  - Cost of queries（查询代价）
  - Global measures（全局特征）

算法细节&优化

- Node order (缩点顺序)
  - Cost of queries（查询代价）
    - 减小查询时的搜索空间-> 减小查询时的最短路树的深度
  - 目的：
    - 提高查询速度



算法细节&优化

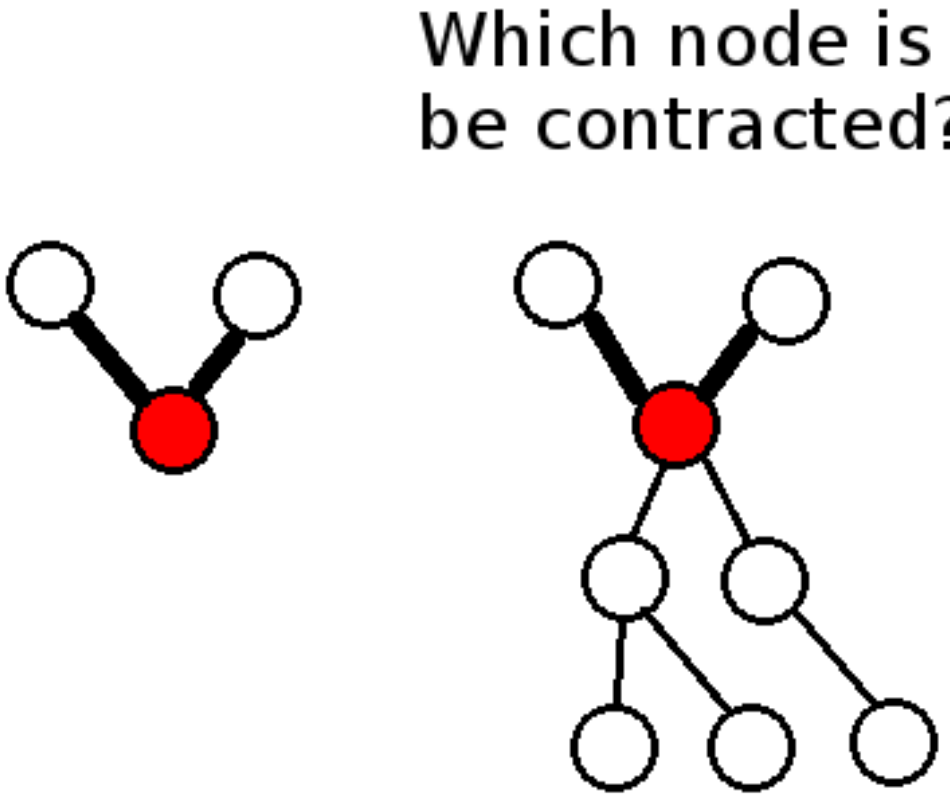
- Node order (缩点顺序)
  - Cost of queries（查询代价）
    - 减小查询时的搜索空间-> 减小查询时的最短路树的深度
  - 具体实现：
    - 最短路树深度的一种近似

Search space depth. We want to reduce the depth of the shortest paths trees our modified Dijkstra algorithm grows during a query. The later a node is contracted, the higher is its order, and it is more likely that it increases the depth of an shortest paths tree. So if there is a node u that is still not contracted, has a contracted neighbor v and there is a node $s \in V$ with a large value for $\text{depth}(s, v)$, node u should be contracted later since there is a chance that $\text{depth}(s, u) = \text{depth}(s, v) + 1$. If node u is contracted later, it is more unlikely that there will be another node u' that now has u as contracted neighbor with $\text{depth}(s, u') = \text{depth}(s, u) + 1$. We use an upper bound on the search space depth as priority term.

$$A[v] := \max\{ \overbrace{A[v]}^{\text{current upper bound}}, \underbrace{A[u] + 1}_{\text{upper bound including } u} \}.$$

算法细节&优化

- Node order (缩点顺序)
  - Cost of queries（查询代价）
    - 减小查询时的搜索空间-> 减小查询时的最短路树的深度



算法细节&优化

- Node order (缩点顺序)
  - 论文中用到了哪些term决定node order?
  - Edge difference（边数差）
  - Cost of contraction（缩点代价）
  - Uniformity（均匀性）
  - Cost of queries（查询代价）
  - Global measures（全局特征）

算法细节&优化

- Node order (缩点顺序)
  - Global measures（全局特征）
    - 利用一些全局特征表示点的重要程度
      - Betweenness

The betweenness centrality of a node v is given by the expression:

$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where σ_{st} is the total number of shortest paths from node s to node t and $\sigma_{st}(v)$ is the number of those paths that pass through v.

      - Reach

vertices. Let P be a shortest s–t path that contains vertex u. The reach $r(u, P)$ of u with respect to P is defined as $\min\{\text{dist}(s, u), \text{dist}(u, t)\}$. The (global) reach of u in the graph G is the maximum reach of u over *all* shortest paths that contain u. Like other centrality

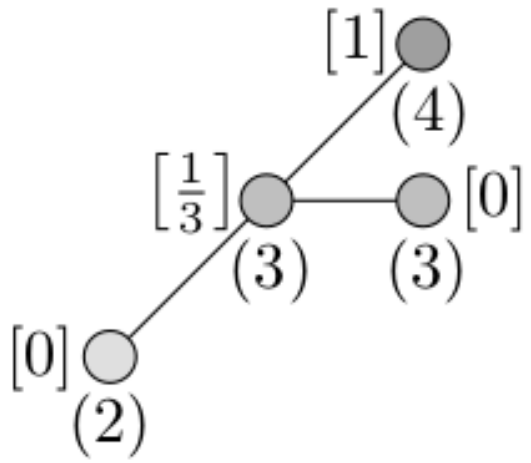
算法细节&优化

- Node order (缩点顺序)
  - Global measures（全局特征）
    - 利用一些全局特征表示点的重要程度
    - 问题：
      - 对于不同的图betweenness和reach的值域不固定，需要归一化
      - 对于规模比较大的图，求解比较慢

算法细节&优化

- Node order (缩点顺序)
  - Global measures（全局特征）
    - 利用一些全局特征表示点的重要程度
    - 归一化的一种方法：

$$\frac{|\{v \in N(u) \mid C(v) < C(u)\}|}{|N(u)|} \text{ or } 0 \text{ if and only if } N(u) = \emptyset.$$



算法细节&优化

- Node order (缩点顺序)
  - Global measures（全局特征）
    - 利用一些全局特征表示点的重要程度
    - 如果求解困难则可以考虑近似

算法细节&优化

- Node order (缩点顺序)
  - Global measures（全局特征）
  - 目标：
    - 更合理的hierarchy（层次），降低查询时间

算法细节&优化

- Node order (缩点顺序)
  - Edge difference（边数差）
  - Cost of contraction（缩点代价）
  - Uniformity（均匀性）
  - Cost of queries（查询代价）
  - Global measures（全局特征）

算法细节&优化

- Node order (缩点顺序)
  - 两个问题
    - 论文中用到了哪些term？
    - 每一个term（项）的数值在预处理过程中是固定的吗？

算法细节&优化

- Node order (缩点顺序)
  - 每一个term的数值在预处理过程中是固定的吗？
  - 答：不是。
  - 哪些term会随着contraction变化？
  - Edge difference（边数差）？
  - Cost of contraction（缩点代价）？
  - Uniformity（均匀性）？
  - Cost of queries（查询代价）？
  - Global measures（全局特征）？

算法细节&优化

- Node order (缩点顺序)
  - 每一个term的数值在预处理过程中是固定的吗？
  - 答：不是。
  - 哪些term会随着contraction变化？
  - **Edge difference（边数差）？**
  - **Cost of contraction（缩点代价）？**
  - **Uniformity（均匀性）？**
  - **Cost of queries（查询代价）？**
  - Global measures（全局特征）？

算法细节&优化

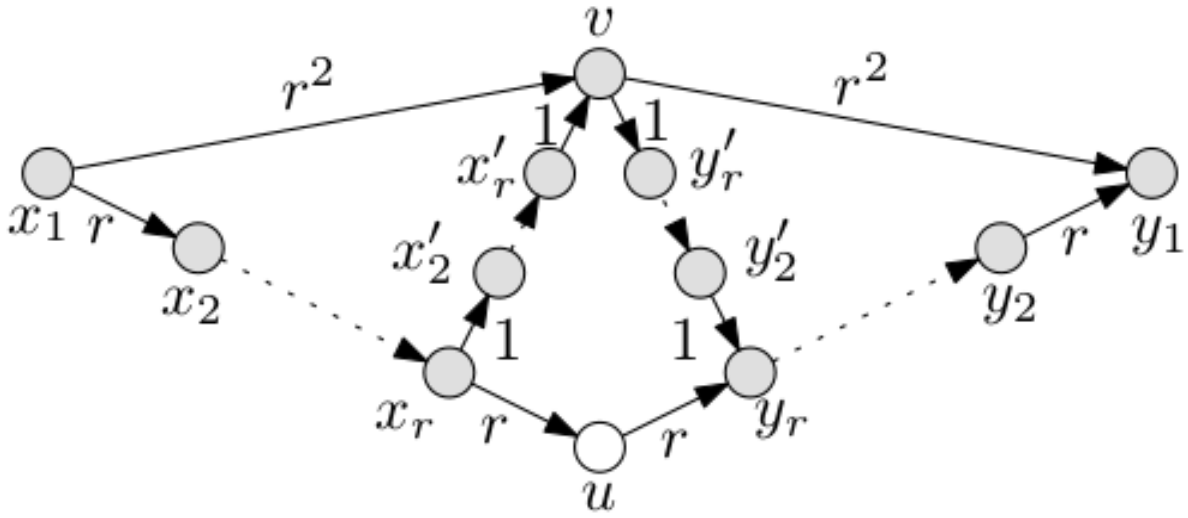
- Node order (缩点顺序)
  - 每一个term的数值在预处理过程中是固定的吗？
  - 怎么办？
  - 答：
    - 周期性全局update
    - Contract一个点后update邻居点
    - Lazy update

算法细节&优化

- Node order (缩点顺序)
  - 每一个term的数值在预处理过程中是固定的吗？
  - 只有update邻居点够不够？

算法细节&优化

- Node order (缩点顺序)
  - 每一个term的数值在预处理过程中是固定的吗？
  - 只有update邻居点够不够？
    - 不够
    - contract一个点可能消除一个点的witness path，从而间接影响是否添加shortcut，导致edge difference变化
    - 而这种影响可能任意远

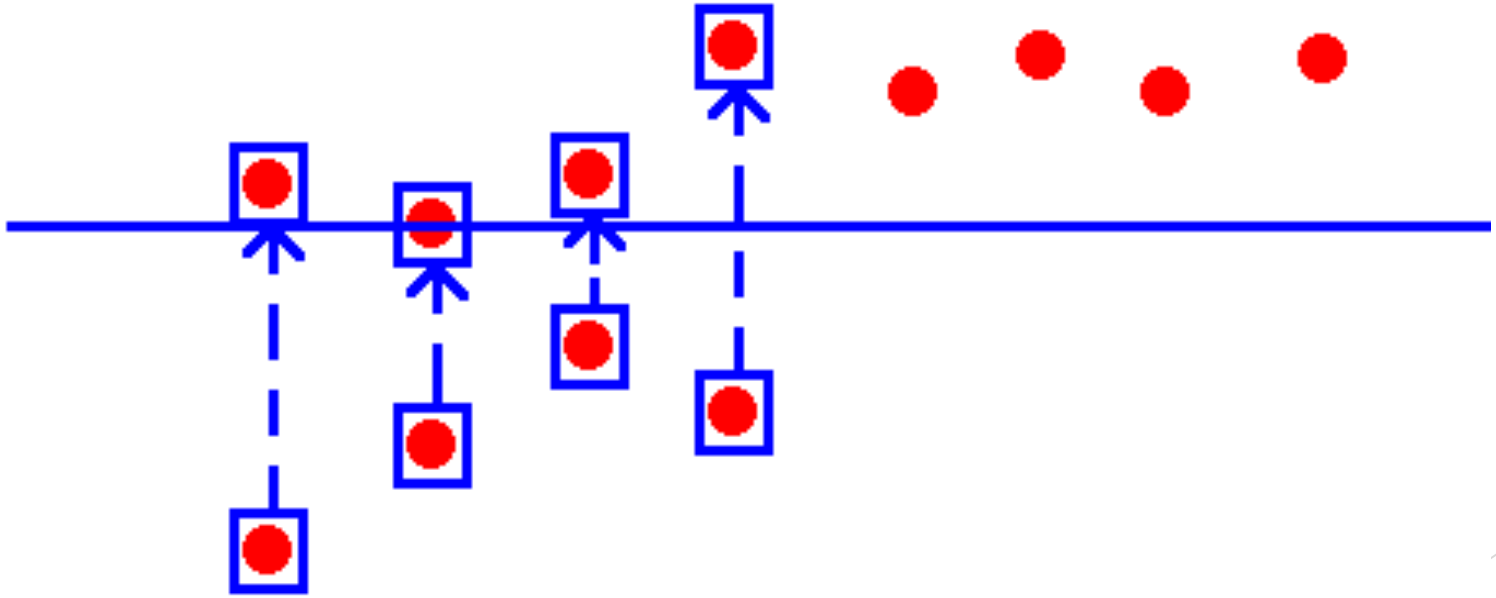


算法细节&优化

- Node order (缩点顺序)
  - 每一个term的数值在预处理过程中是固定的吗？
    - Lazy update
      - 1. 使用一个优先队列维护所有当前未被删除的点
      - 2. 更新队列头
      - 3. 如果其还是队列头那么该点就是当前的最小点
      - 4. 否则更换队列头，跳到2
    - 对于权值随着contraction单调上升的情况是正确的

算法细节&优化

- Node order (缩点顺序)
  - 每一个term的数值在预处理过程中是固定的吗？
    - Lazy update



算法细节&优化

- Node order (缩点顺序)
  - 每一个term的数值在预处理过程中是固定的吗？
    - Lazy update
    - 对于权值随着contraction不一定单调上升的情况？

算法细节&优化

- Node order (缩点顺序)
  - 每一个term的数值在预处理过程中是固定的吗？
    - 据我所知现在我们使用的算法是当作固定处理的
    - 优点：预处理快（不用lazy update，有稳定的速度加成）
    - 缺点：没有理论支撑，实际上有可能预处理和查询都变慢了

算法细节&优化

- ---------- 华丽的分割线 ----------

算法细节&优化

- 其他优化
  - 看过abstract的同学有没有注意到这样一句话？

speedup techniques and a *negative* space overhead, i.e., the data structure for distance computation needs *less* space than the input graph. CHs can serve as foundation for many

  - 大意就是：我们的空间开销比原图还小！

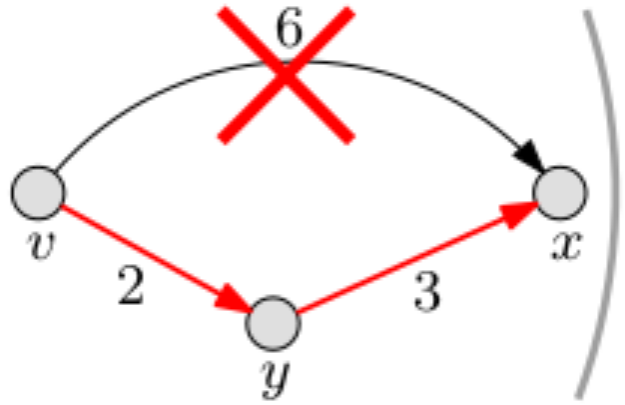
算法细节&优化

- 其他优化
  - 对于这个问题我想了很久，最终的结论是，negative space overhead是因为一个优化
  - 而这个优化在原来70页的论文里只有半页

speedup techniques and a *negative* space overhead, i.e., the data structure for distance computation needs *less* space than the input graph. CHs can serve as foundation for many

算法细节&优化

- 其他优化
  - On-the-fly Edge Reduction(即时缩边)
    - 你没有看错！号称只有缩点的CH算法也是有缩边的！
    - 总结起来就是一句话：
      - 如果<v,x>的长度在witness path的搜索过程中发现还不如v到x的路径的长度短，就删掉它



算法细节&优化

- 其他优化
  - Stall-on-demand
    - 之前我们一直在讲预处理时的事情

算法细节&优化

- 回顾CH的预处理
  - 确定一个**缩点的顺序（顺序是什么？）**
  - 每次按这个顺序选一个点u缩
    - 如果v->u->w**可能是v到w的唯一的最短路（怎么确定？）**，在原图中增加shortcut捷径(v,w)，权值为(v,u)和(u,w)的权值和
  - 删除点u

Algorithm 1: SimplifiedConstructionProcedure($G = (V, E), <$)

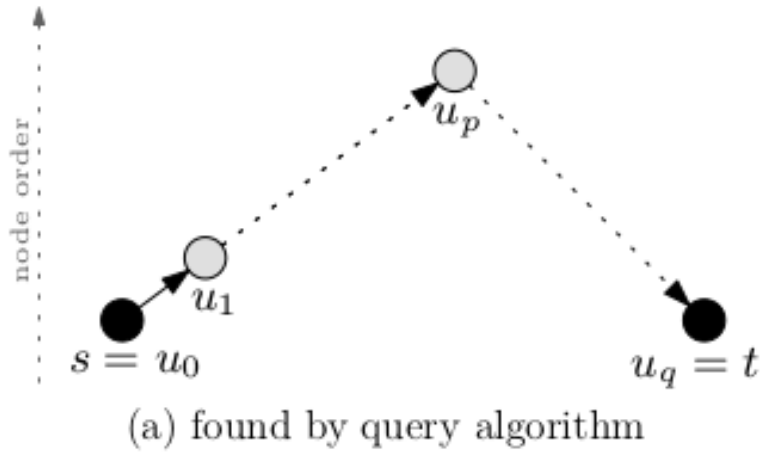
```
1 foreach u ∈ V ordered by < ascending do
2     foreach (v, u) ∈ E with v > u do
3         foreach (u, w) ∈ E with w > u do
4             if ⟨v, u, w⟩ "may be" the only shortest path from v to w then
5                 E:= E ∪ {(v, w)} (use weight w(v, w):= w(v, u) + w(u, w))
```

算法细节&优化

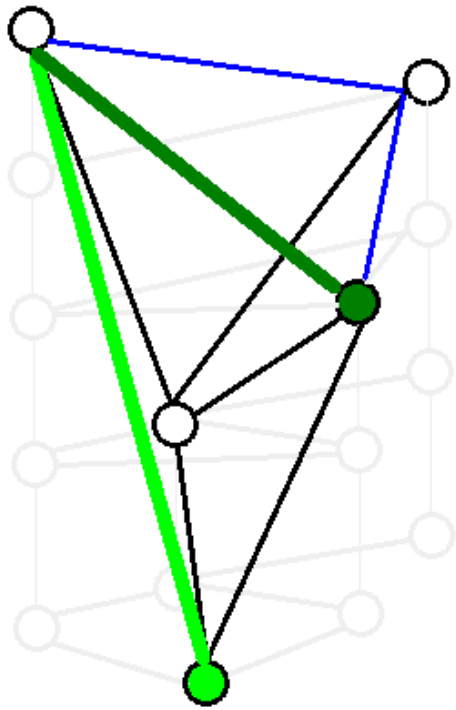
- 其他优化
  - Stall-on-demand
    - 现在我们来讲讲查询时的搜索优化

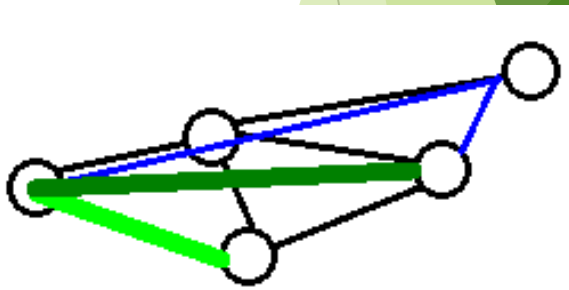
简介

- 回顾CH的最短路查询
  - 搜索时做双向Dijkstra搜索
  - 但有一个限制
    - 起点开始只往上层搜索，终点开始只往上层回溯



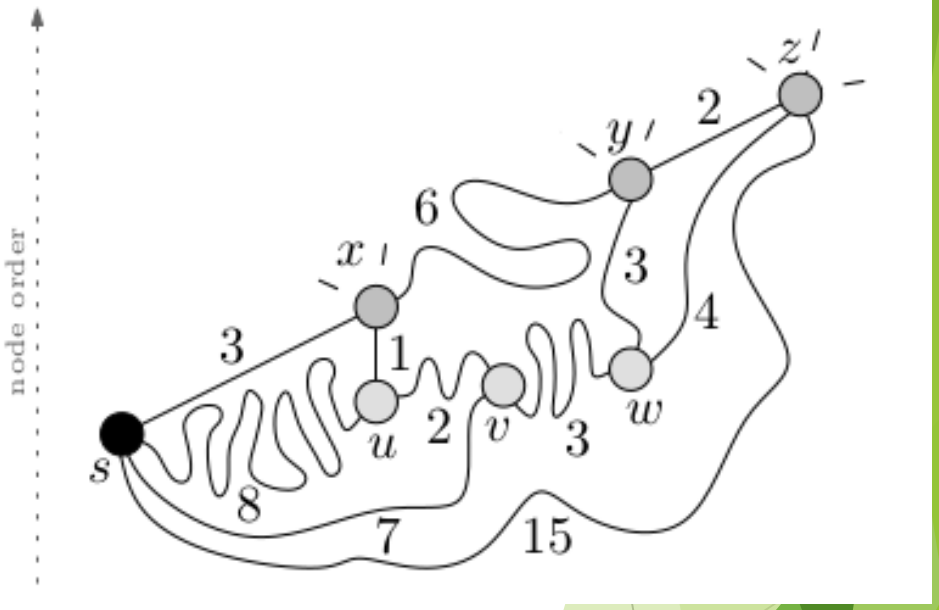
(a) found by query algorithm





算法细节&优化

- 其他优化
  - Stall-on-demand
    - s->x->u的路径不会被CH发现
    - s->u的距离会是8
    - 用s->x->u可以知道s->u肯定不是最短路
    - 所以可以stall(阻止)点u和其后继节点
    - 等到第二次被更新时解除stall（策略不唯一）



算法细节&优化

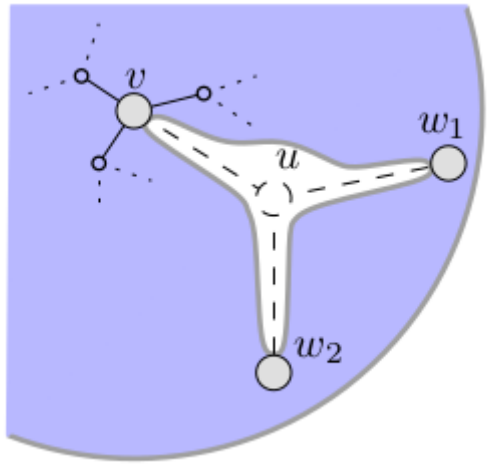
- 思考题
  - 确定一个缩点的顺序
  - 每次按这个顺序选一个点u缩
    - 如果v->u->w可能是v到w的唯一的最短路（思考witness path是怎么找的），在原图中增加shortcut捷径(v,w)，权值为(v,u)和(u,w)的权值和
  - 删除点u（删除后会不会导致原来的最短路被删掉？）

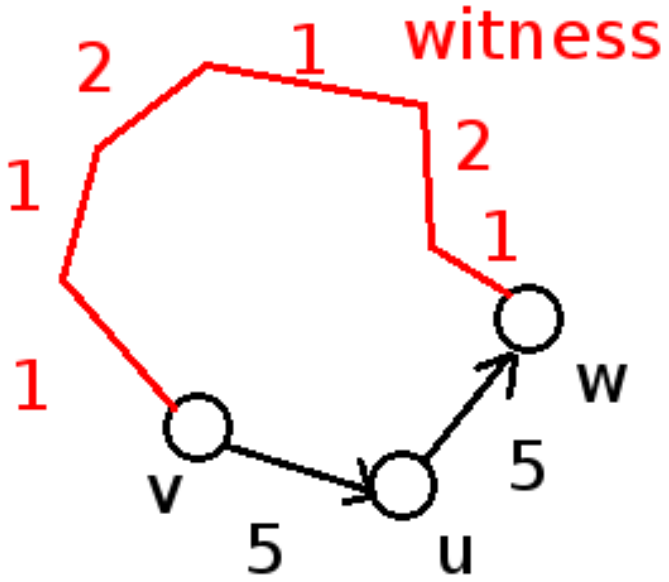
Algorithm 1: SimplifiedConstructionProcedure($G = (V, E), <$)

```
1 foreach u ∈ V ordered by < ascending do
2     foreach (v, u) ∈ E with v > u do
3         foreach (u, w) ∈ E with w > u do
4             if ⟨v, u, w⟩ "may be" the only shortest path from v to w then
5                 E := E ∪ {(v, w)} (use weight w(v, w) := w(v, u) + w(u, w))
```

算法细节&优化

- Witness path search (确定v->u->w是不是v到w唯一的最短路)
  - 如何找witness path ?
  - 具体方法：
    - 在V/{u}构成的子图中运行最短路算法
    - 枚举v，w ，如果没有找到不长于v->u->w的路径，则增加边(v,w)
    - 删除点u
  - 这一过程称为预处理中的局部搜索



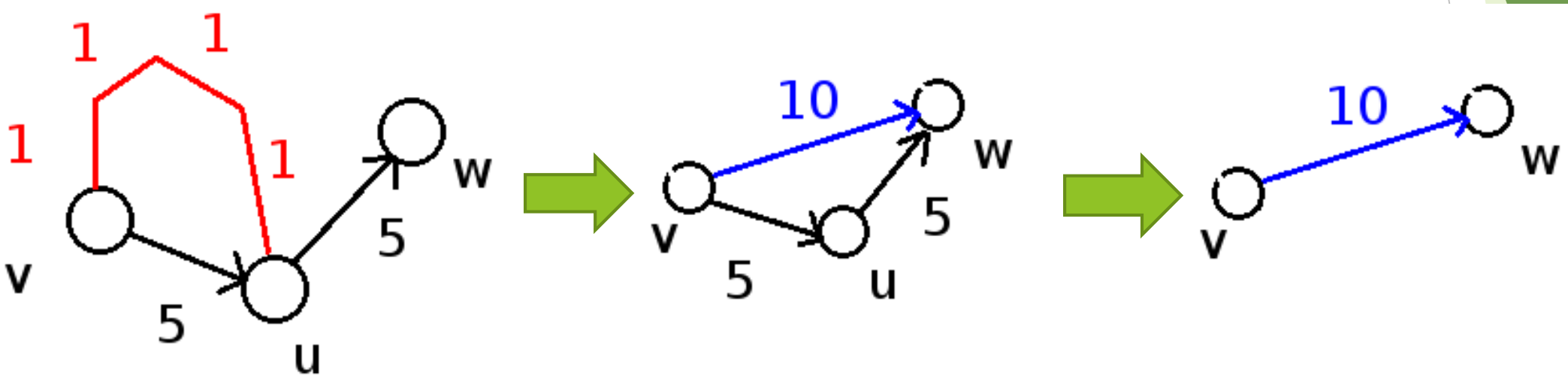


算法细节&优化

- This page is intentionally left blank

算法细节&优化

- 思考题
  - 是否有考虑过这种情况？

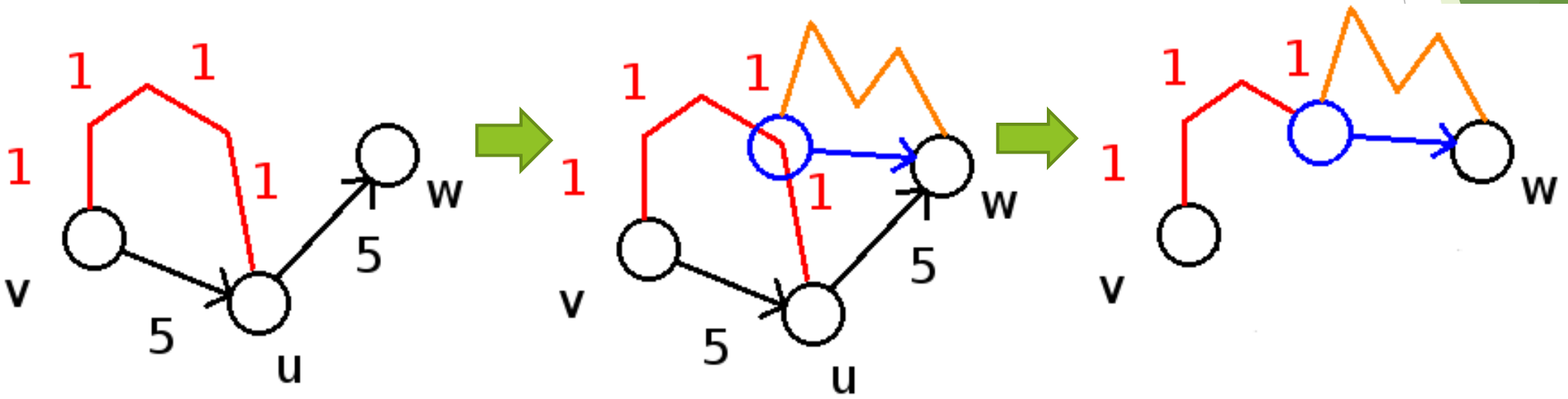


算法细节&优化

- This page is intentionally left blank

算法细节&优化

- 思考题
  - 仍然正确



算法改进

算法改进

- 改进方向
  - 一些可能的小改进
    - 添加不同的term，比如路径特征
    - 记录witness search path加速预处理（实践比较困难）
    - 结合地图的几何特性
  - 更快的查询速度
    - CHASE（结合CH和Arc flag）
  - 更好地刻画现实网络
    - Time-dependent Contraction Hierarchies (TCH)
  - 更好地应对动态网络（更快的预处理）
    - Customizable Contraction Hierarchies (CCH)

算法改进

- 改进方向
  - 更快的查询速度
    - 当前最快的算法（如TNR，PHAST，HL）
      - 基本思路都是打最短路表
    - 另一类方法比如Arc flag，A*
      - 基本思路是Goal directed
  - 结合CH和这两种思路可能可以有更快的查询速度

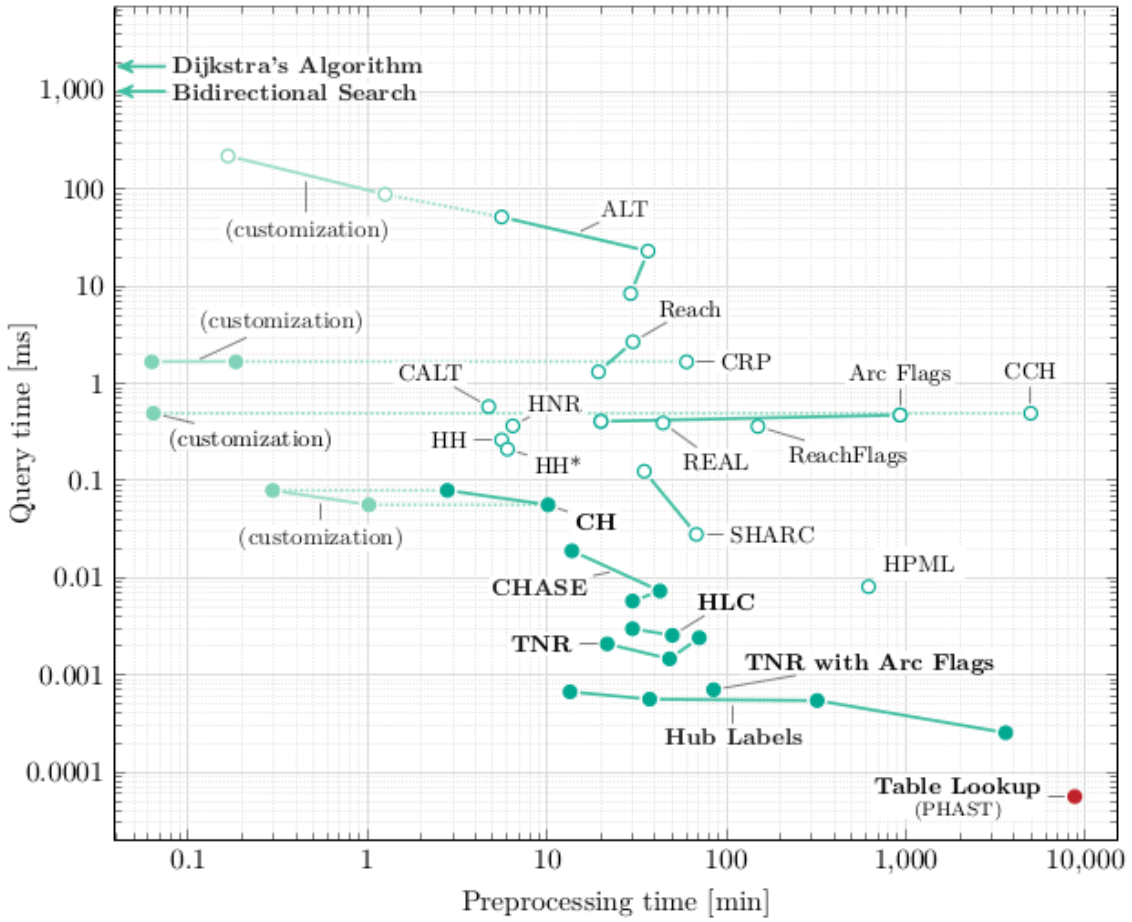
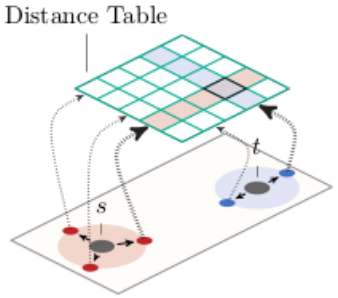


Figure 7. Preprocessing and average query time performance for algorithms with available experimental data on the road network of Western Europe, using travel times as edge weights. Connecting lines indicate different trade-offs for the same algorithm. The figure is inspired by [238].

算法改进



- 改进方向
  - TNR
    - 选取一些点T，两两最短路打表
    - 再把T和所有点之间的最短路打表
    - 用某种方法判断s到t是否经过点T中的点
      - 如果经过，枚举取最小后直接输出
      - 如果不经过，调用其他算法（比如CH）
  - PHAST
    - 用CH算法+硬件(cache，SSE，多核，GPU)加速最短路树的构建

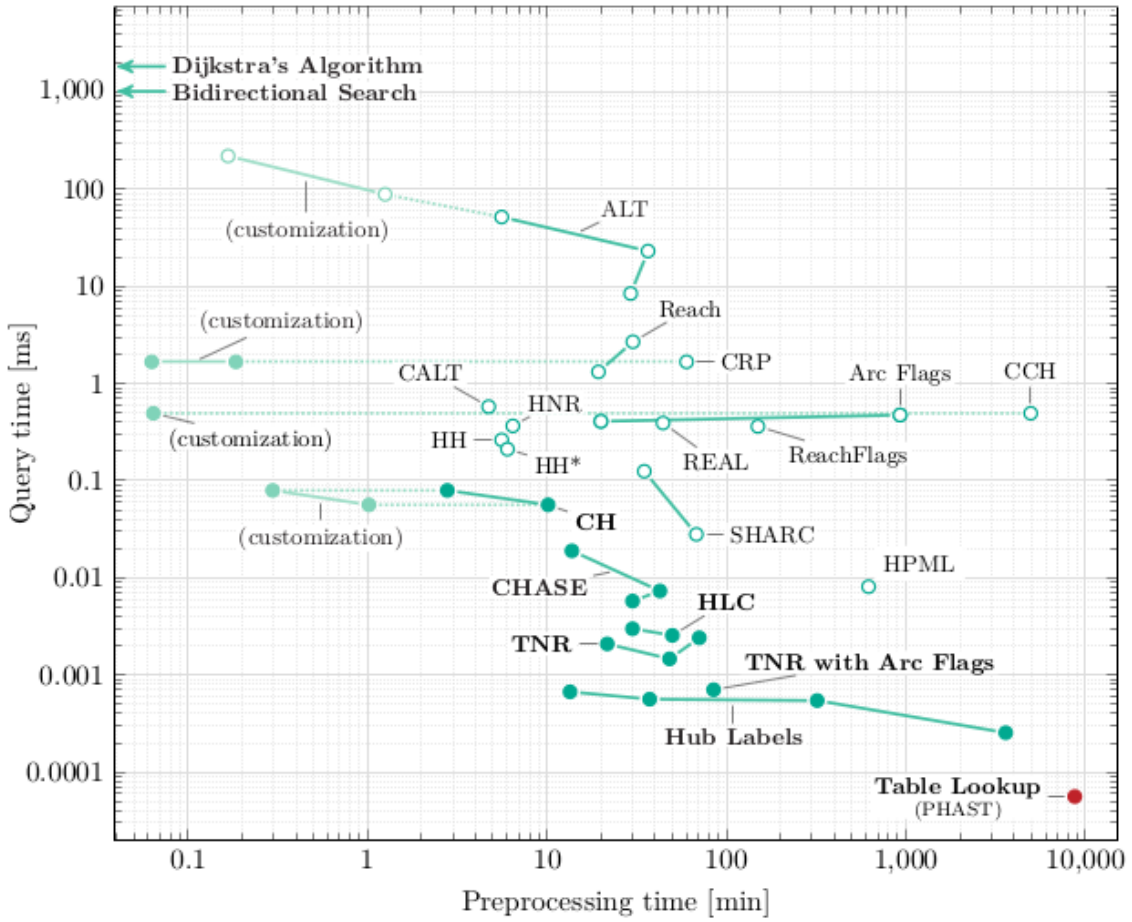


Figure 7. Preprocessing and average query time performance for algorithms with available experimental data on the road network of Western Europe, using travel times as edge weights. Connecting lines indicate different trade-offs for the same algorithm. The figure is inspired by [238].

算法改进

- 改进方向
  - Hub Labeling(HL)
    - 每个点u有一个label集L(u)
    - 对任意点s，t，L满足L(s)∩L(t)至少有一个点在s到t的最短路上
    - 预处理所有u到L(u)集合的最短路，打表
    - 查询时枚举L(s)∩L(t)中的所有点v，计算s->v->t最小值
  - Arc flag
    - 划分图为几个部分
    - 对于每个部分，预处理到达这个目的的逆最短路树，给树上的边打上标记(arc flag)
    - 查询s到t的最短路时，只考虑有到达t所在区域的标记的那些边

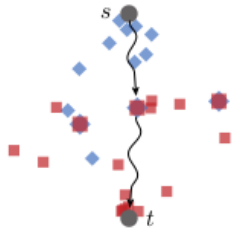
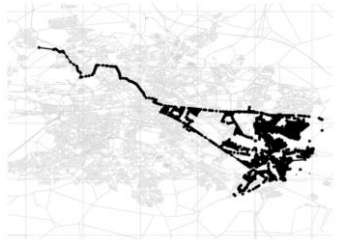


Figure 5. Illustrating hub labels of vertices s (diamonds) and t (squares).



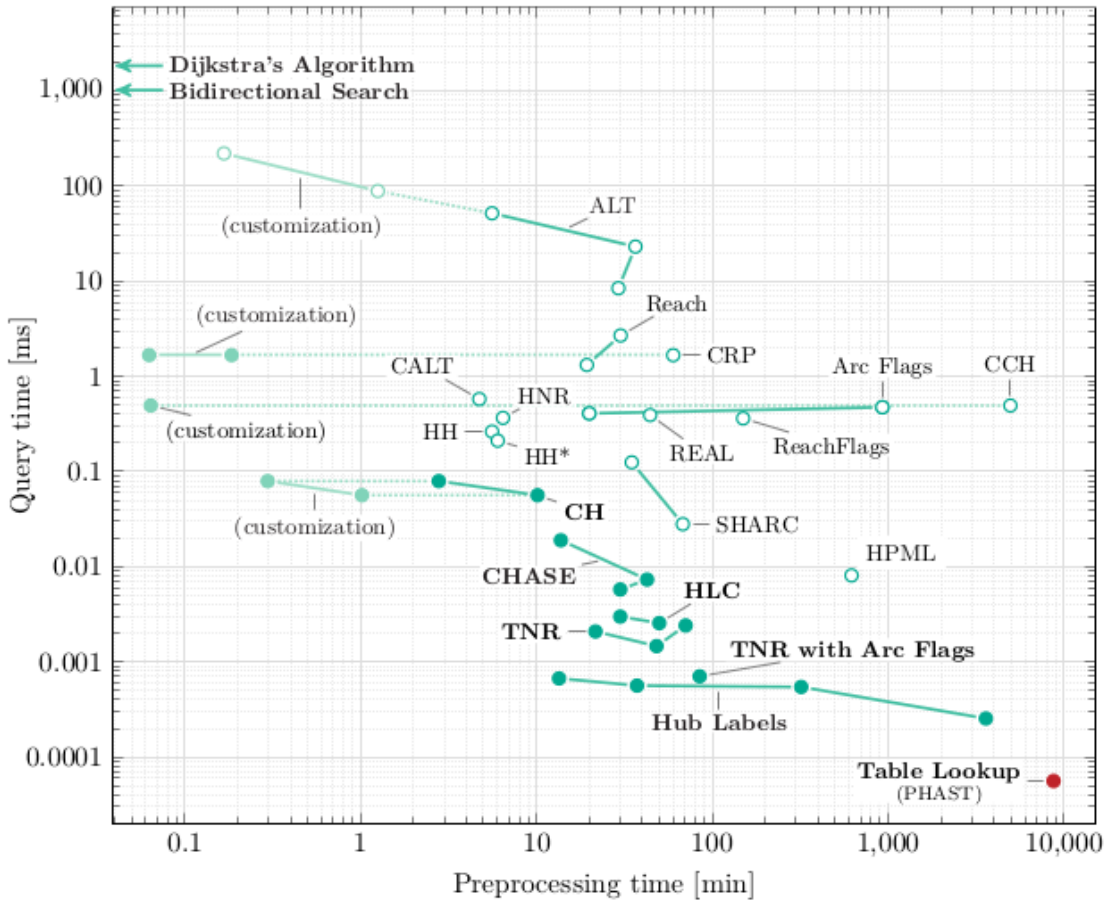
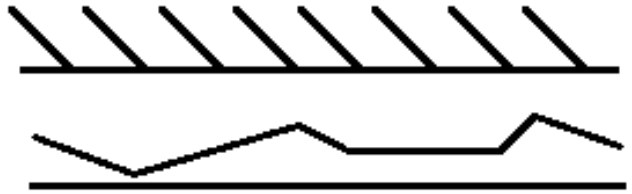


Figure 7. Preprocessing and average query time performance for algorithms with available experimental data on the road network of Western Europe, using travel times as edge weights. Connecting lines indicate different trade-offs for the same algorithm. The figure is inspired by [238].

算法改进

- 改进方向
  - 更好地刻画现实网络
    - TCH
      - 将道路通过时间函数刻画为分段线性函数
      - FIFO假设（先进先出）
      - 综合所有时间，得到通过一条shortcut的上下界函数
      - 添加所有可能的shortcut，记录有用的witness path



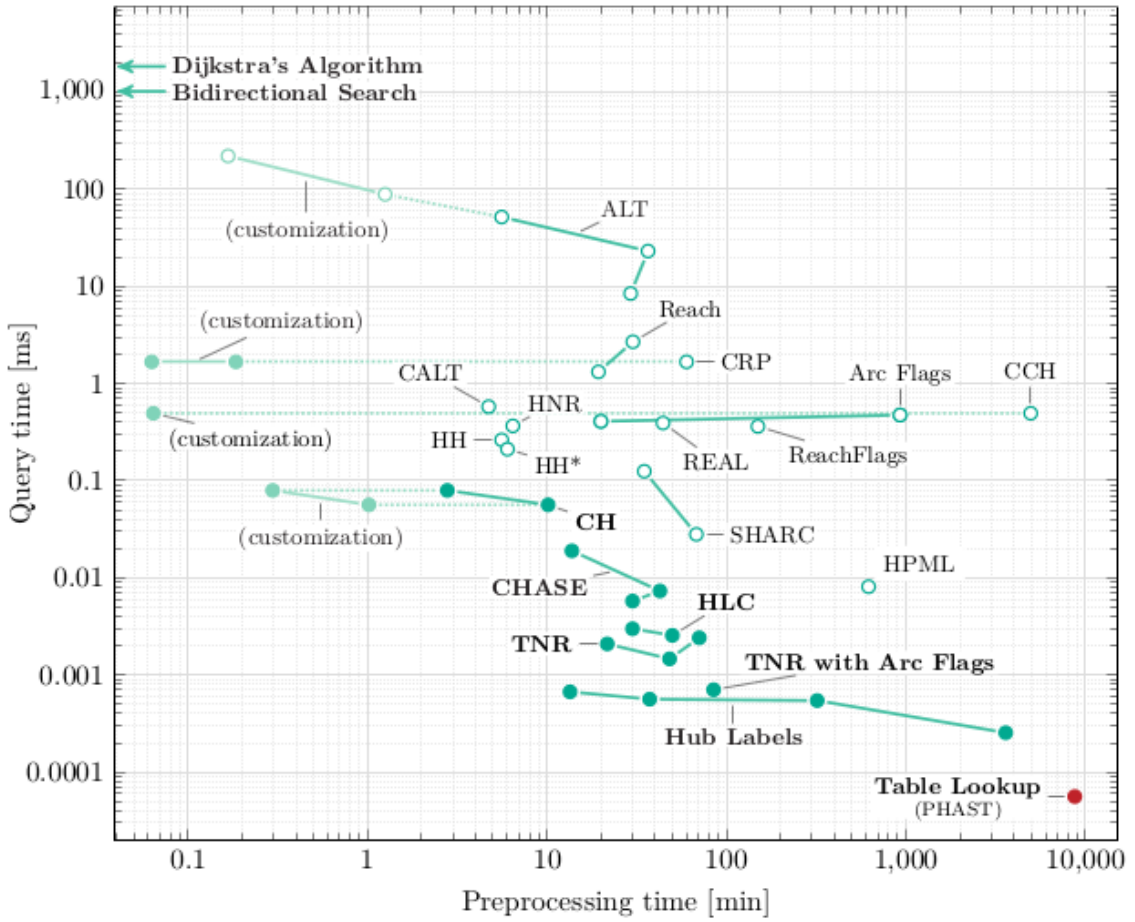


Figure 7. Preprocessing and average query time performance for algorithms with available experimental data on the road network of Western Europe, using travel times as edge weights. Connecting lines indicate different trade-offs for the same algorithm. The figure is inspired by [238].

算法改进

- 改进方向
  - 更好地应对动态网络
    - CCH
      - 选取只和图的形状有关而和权值无关的node order
      - 预处理分为preprocessing和customization，分别负责构图（慢）和更新权值（快）
        - 构图加入所有可能的shortcut
        - 更新权值利用三角形线性更新，无需witness path search

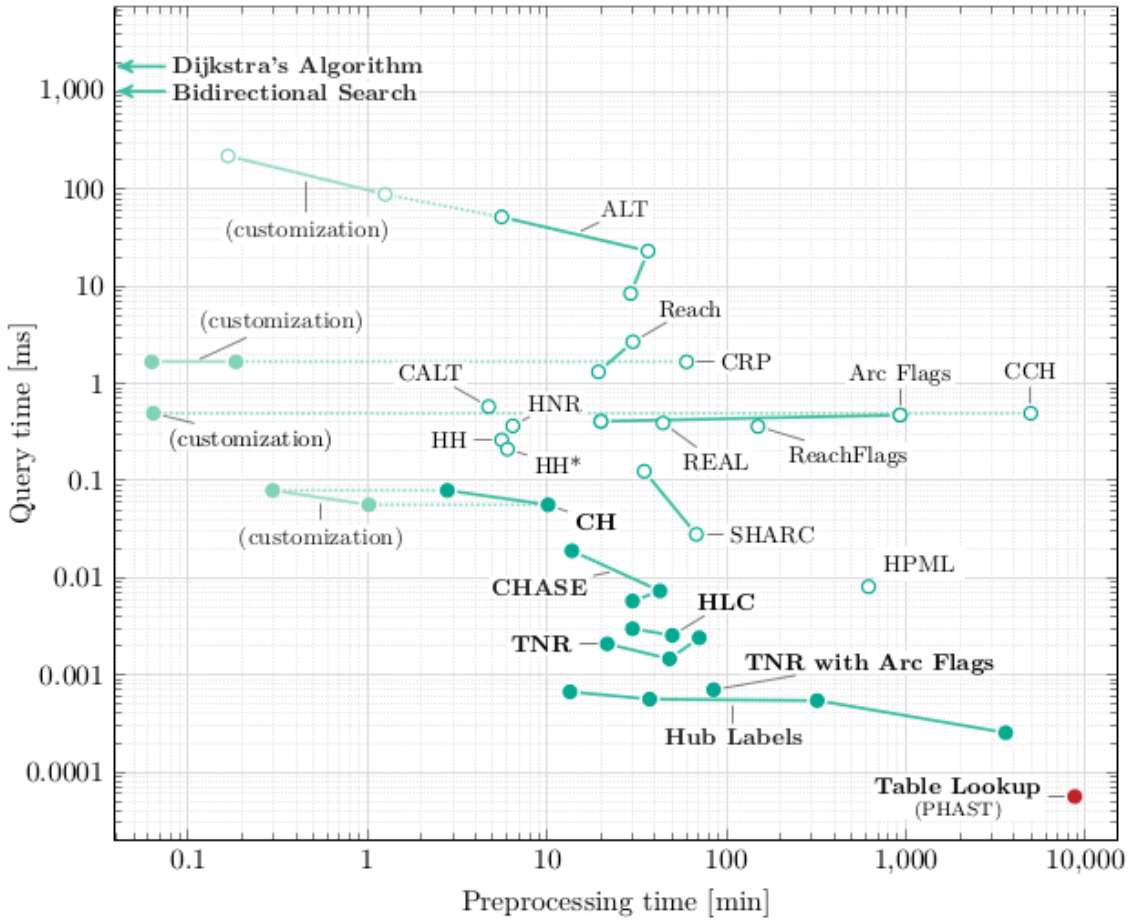


Figure 7. Preprocessing and average query time performance for algorithms with available experimental data on the road network of Western Europe, using travel times as edge weights. Connecting lines indicate different trade-offs for the same algorithm. The figure is inspired by [238].

算法改进

- Customizable Contraction Hierarchies (CCH)
  - 选取只和图的形状有关而和权值无关的node order
  - 预处理分为preprocessing和customization，分别负责构图（慢）和更新权值（快）
    - 构图加入所有可能的shortcut
    - 更新权值利用三角形线性更新，无需witness path search

算法改进

- Customizable Contraction Hierarchies (CCH)
  - 选取只和图的形状有关而和权值无关的node order
  - 什么顺序比较好？
    - Nested dissection
    - 选一些点把原图分开，这样shortcut比较少
      - 来自历史的思想——separator

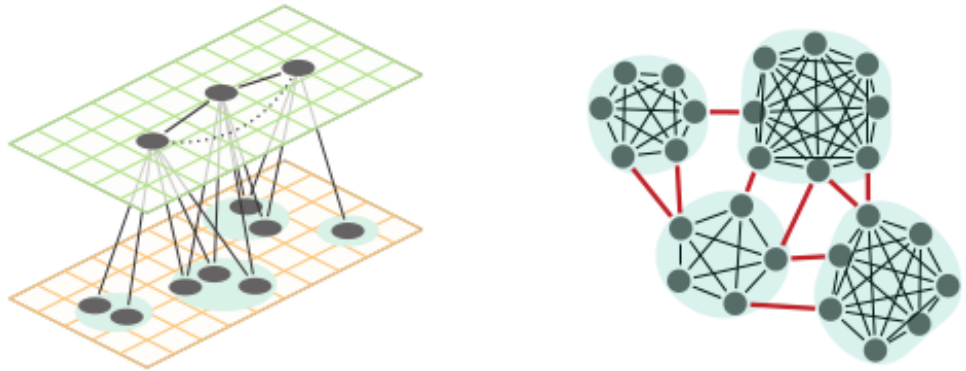
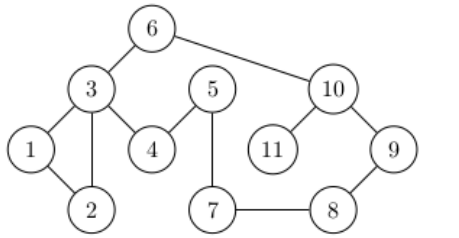
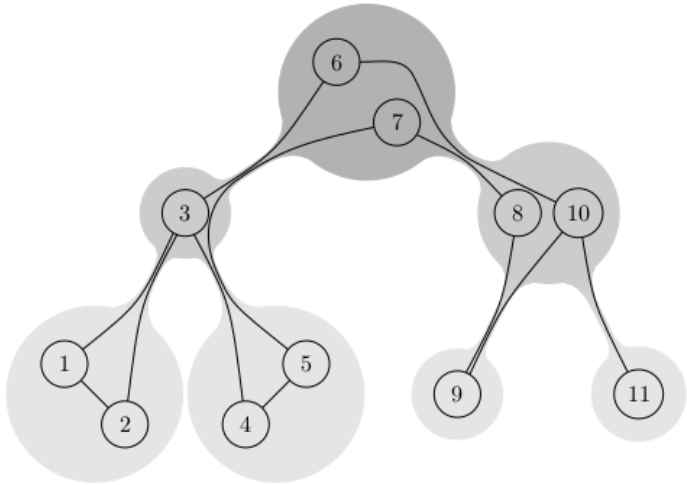


Figure 3. Left: Multilevel overlay graph with two levels. The dots depict separator vertices in the lower and upper level. Right: Overlay graph constructed from arc separators. Each cell contains a full clique between its boundary vertices, and cut arcs are thicker.



(a) A graph $G = (V, E)$ with vertices numbered 1 through 11.



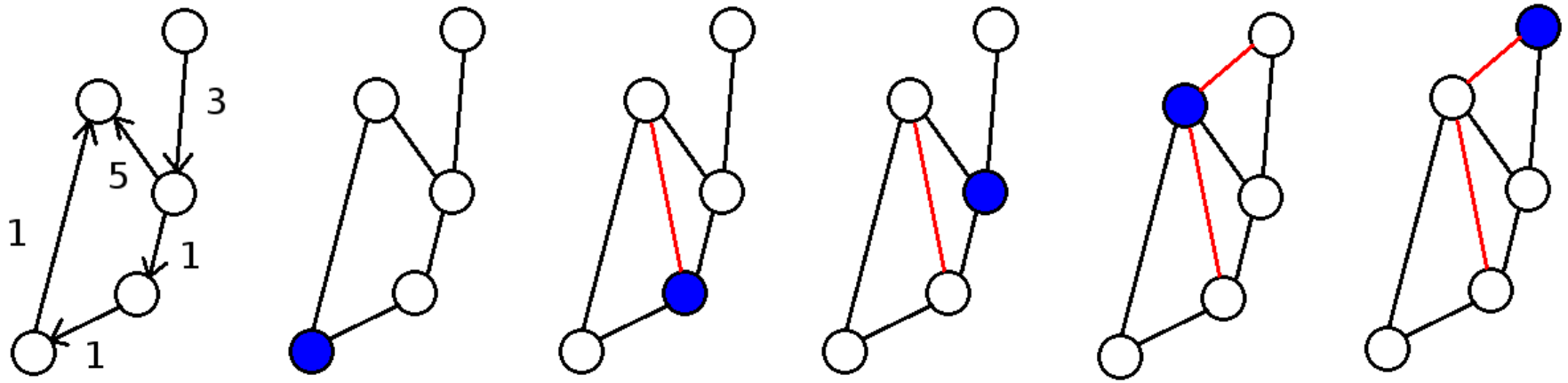
(b) A $(1/2, \sqrt{n})$-separator decomposition $\mathcal{T} = (\mathcal{X}, \mathcal{E})$ of the graph shown in Figure 3.6a. The nodes $X \in \mathcal{X}$ are drawn as grey circles containing the vertices $u \in X$. The shades of grey become increasingly lighter with ascending levels of the nodes $X \in \mathcal{X}$. The root of $\mathcal{T}$ is the node drawn at the top.

算法改进

- Customizable Contraction Hierarchies (CCH)
  - 选取只和图的形状有关而和权值无关的node order
  - 预处理分为preprocessing和customization，分别负责构图（慢）和更新权值（快）
    - 构图加入所有可能的shortcut
    - 更新权值利用三角形线性更新，无需witness path search

算法改进

- Customizable Contraction Hierarchies (CCH)
  - 预处理分为preprocessing和customization，分别负责构图（慢）和更新权值（快）
    - 构图加入所有可能的shortcut
  - 把图看作无向无权图，之后再定方向
  - 通过customization更新权值

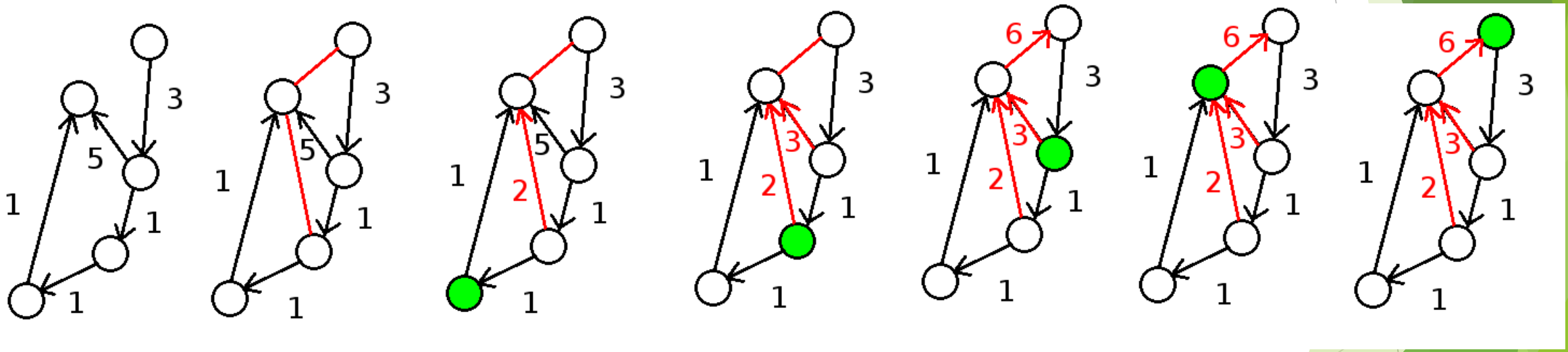


算法改进

- Customizable Contraction Hierarchies (CCH)
  - 选取只和图的形状有关而和权值无关的node order
  - 预处理分为preprocessing和customization，分别负责构图（慢）和更新权值（快）
    - 构图加入所有可能的shortcut
    - 更新权值利用三角形线性更新，无需witness path search

算法改进

- Customizable Contraction Hierarchies (CCH)
  - 预处理分为preprocessing和customization，分别负责构图（慢）和更新权值（快）
    - 更新权值利用三角形线性更新，无需witness path search
    - 这里只先展示下三角的部分（保证正确性）



算法改进

- Customizable Contraction Hierarchies (CCH)
  - Nested dissection问题有多难？
  - 用什么算法可以求解？
  - 怎样可以做到快速的完美witness path搜索？
  - 为什么正确？

  - 等等问题待下次详细展开

参考文献

参考文献

- **CH算法**
  - [1] Geisberger, R., Sanders, P., Schultes, D., and Delling, D. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In 7th Workshop on Experimental Algorithms. LNCS Series, vol. 5038. Springer, 319–333, 2008.
- **综述**
  - [2] Hannah Bast, Daniel Delling, Andrew V. Goldberg, Matthias M¨uller Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. Route planning in transportation networks. Technical Report abs/1504.05140, ArXiv e-prints, 2015.
- HH算法
  - [3] Sanders, P. and Schultes, D. Engineering highway hierarchies. In 14th European Symposium on Algorithms (ESA). LNCS Series, vol. 4168. Springer, 804–816, 2006.

参考文献

- HNR算法
  - [4] Schultes, D. and Sanders, P. Dynamic highway-node routing. In 6th Workshop on Experimental Algorithms (WEA). LNCS Series, vol. 4525. Springer, 66–79, 2007.
- TCH算法
  - [5] Batz, G., Delling, D., Sanders, P., and Vetter, C. Time-dependent contraction hierarchies. In 10th Workshop on Algorithm Engineering and Experiments (ALENEX). 97–105. http://algo2.iti. uni-karlsruhe.de/1222.php, 2009.
- Dijkstra和A*
  - [6] http://www.cs.princeton.edu/courses/archive/spr06/cos423/Handouts/EPP%20shortest%20path%20algorithms.pdf

参考文献

- Arc flag
  - [7] U. Lauther. An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background. In Geoinformation und Mobilität – von der Forschung zur praktischen Anwendung, volume 22, pages 219–230. IfGI prints, Institut für Geoinformatik, Münster, 2004.
- CCH算法
  - [8] Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable contraction hierarchies. In Proceedings of the 13th International Symposium on Experimental Algorithms (SEA’ 14), volume 8504 of Lecture Notes in Computer Science, pages 271–282. Springer, 2014.
  - [9] Reinhard Bauer, Tobias Columbus, Ignaz Rutter, and Dorothea Wagner. Search-space size in contraction hierarchies. In Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP’ 13), volume 7965 of Lecture Notes in Computer Science, pages 93–104. Springer, 2013.

参考文献

- [10] Peter Sanders and Christian Schulz. Think locally, act globally: Highly balanced graph partitioning. In Proceedings of the 12th International Symposium on Experimental Algorithms (SEA’ 13), volume 7933 of Lecture Notes in Computer Science, pages 164–175. Springer, 2013.
- 其他
  - [11] Schultes, D. Route planning in road networks. Ph.D. thesis, Universit ¨at Karlsruhe (TH), 2008.

Q & A