IGERT Data Science : Graph Algorithms

Nirman Kumar

University of California, Santa-Barbara

September 15, 2015

Why study Graphs and Graph algorithms

Graphs are useful in modeling complex systems with interacting components

Why study Graphs and Graph algorithms

Graphs are useful in modeling complex systems with interacting components

We look at, understand, and interact with graphs locally: Graph algorithms compute more global properties

Graphs - Mathematics and Computer Science

Graphs have a deep Mathematical theory

Graphs - Mathematics and Computer Science

Graphs have a deep Mathematical theory

“Intrinsic” to computer science - represent hard problems, program states and execution, search structures, etc.

Network Science :: Graphs

Network Science :: Graphs

$=$

Network Science

Network Science :: Graphs

$$=$$

Statistical Mechanics :: Mechanics

Examples of “real networks”

- **The internet**
- **The WWW**
- **The Power Grid**
- **Collaboration network**
- **Citations**

An indispensable tool : Visualization

Courtesy : The OPTE project

Goals of Network Science

Understand and infer "macro" properties of networks

- **Network vulnerability**
- **How things spread : information, epidemics etc.**
- **Identifying "important" nodes in networks**
- **...**

Page rank : Google's secret formula

Page rank : Google's secret formula

Depends on computing over the WWW network - EFFICIENTLY

Graph theory and Algorithms

Page rank : Google's secret formula

Depends on computing over the WWW network - EFFICIENTLY

Graph algorithms : indispensable to compute statistics on large networks

Fundamental graph problems

- **Traversals**
- **Connectivity**
- **Shortest paths**
- **Identifying subgraphs with specific structure**
- **Characterizing graphs**
- **Optimization problems**
- **...**

The Shortest Path problem

Given graph and a vertex s find shortest paths from s to all other vertices.

The Shortest Path problem

Given graph and a vertex s find shortest paths from s to all other vertices.

- **Map routing, robot navigation, urban traffic planning**
- **Network routing protocols (OSPF, BGP, RIP)**
- **...**

Traversals : Depth First Search (DFS)

Visit all vertices of the graph

Traversals : Depth First Search (DFS)

Visit all vertices of the graph

Procedure DFS(G, v)
begin
  label v as discovered
  forall the *edges from* v *to* w *in* $G.adjacentEdges(v)$ **do**
    if w *is not labeled as discovered* **then**
      recursively call DFS(G, w)
    end
  end
end

Non recursive DFS

```
Procedure DFS-iterative(G, s)
begin
    S ← empty stack
    S.push(s)
    while S is not empty do
        v ← S.pop()
        if v is not labeled as discovered then
            label v as discovered
            forall the edges from v to w in G.adjacentEdges(v) do
                S.push(w)
            end
        end
    end
end
```

Breadth First Search

Can compute shortest paths in unweighted graphs!

Breadth First Search

```
Procedure BFS(G, s)
begin
    s.dist ← 0
    for currDist ← 0 to |V| do
        for all vertices v do
            if v not labeled as discovered and v.dist == currDist then
                Label v as discovered
                forall the edges from v to w in G.adjacentEdges(v) do
                    Handle w
                end
            end
        end
    end
end
```

Breadth First Search

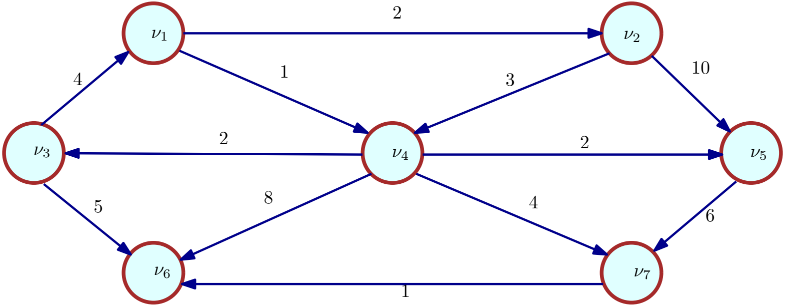
Runtime : $O(|V|^2)$

Breadth First Search

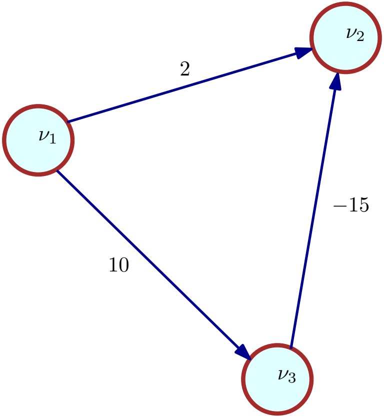
Runtime : $O(|V|^2)$

Can achieve $O(|V| + |E|)$ **using a smarter implementation**

Edge weighted graphs



Negative edge weights



Main structural properties of Shortest Paths

- **Prefixes of shortest paths are shortest paths**

- **Shortest paths from s - make up a tree**

Advancing the frontier

- **At any time we know the distance of some vertices**

Advancing the frontier

- **At any time we know the distance of some vertices**
- **How do we discover further?**

Advancing the frontier

- **At any time we know the distance of some vertices**
- **How do we discover further?**
- **Relaxation of an edge** $(v, w) : d(w) = \min(d(w), d(v) + c_{vw})$

Djikstra algorithm : Generalize BFS

- **Store d_v, v.isKnown, v.path**

Djikstra algorithm : Generalize BFS

- **Store d_v, v.isKnown, v.path**

- **Pick vertex with minimum d_v (that is not known)**

Djikstra algorithm : Generalize BFS

- **Store d_v, v.isKnown, v.path**
- **Pick vertex with minimum d_v (that is not known)**
- **Mark it known**

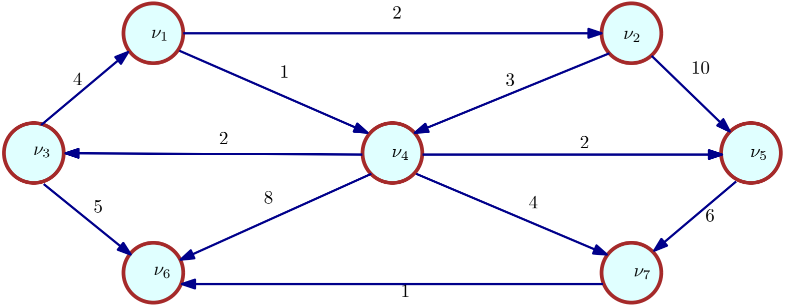
Djikstra algorithm : Generalize BFS

- **Store d_v, v.isKnown, v.path**
- **Pick vertex with minimum d_v (that is not known)**
- **Mark it known**
- **Relax all edges outgoing from it**

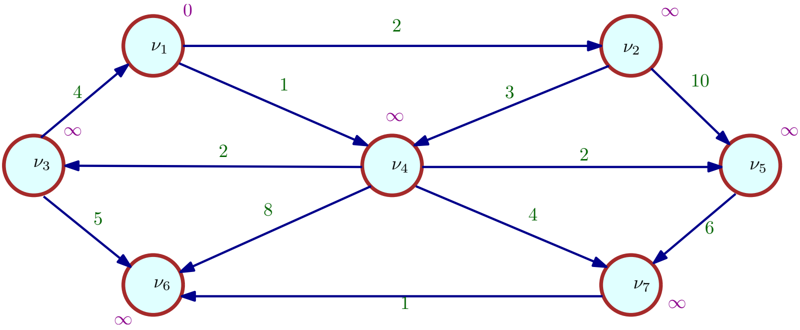
Djikstra algorithm : Generalize BFS

- **Store d_v, v.isKnown, v.path**
- **Pick vertex with minimum d_v (that is not known)**
- **Mark it known**
- **Relax all edges outgoing from it**
- **Repeat until all vertices are known**

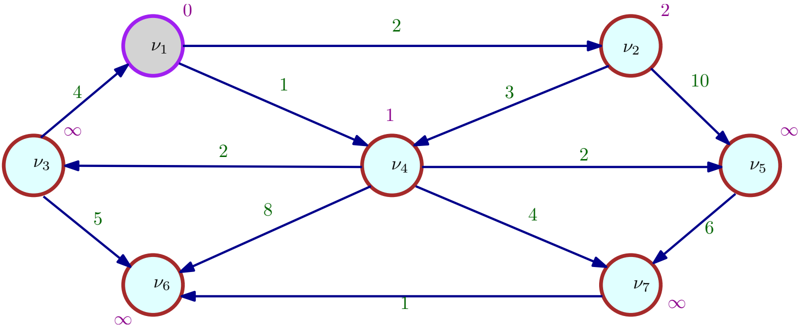
Example of Djikstra in action



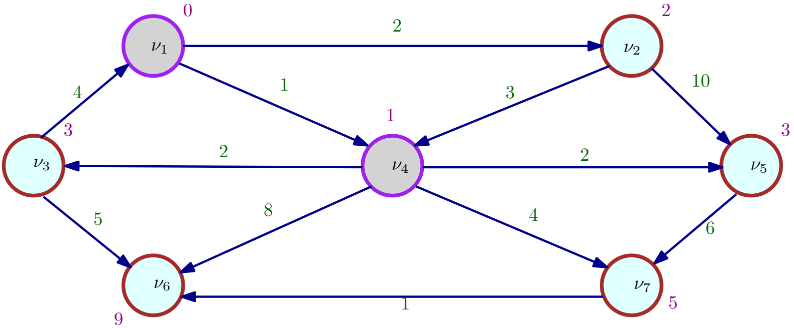
Example of Djikstra in action



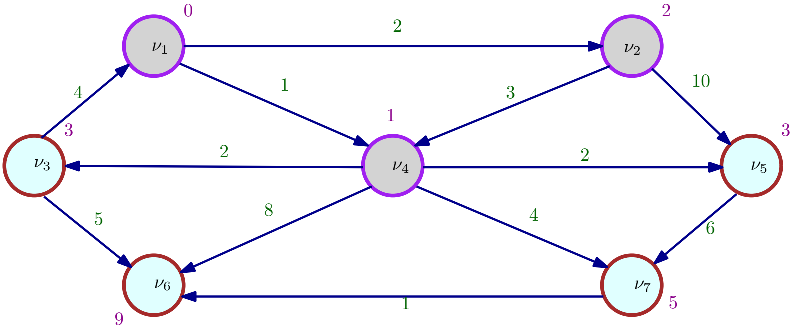
Example of Djikstra in action



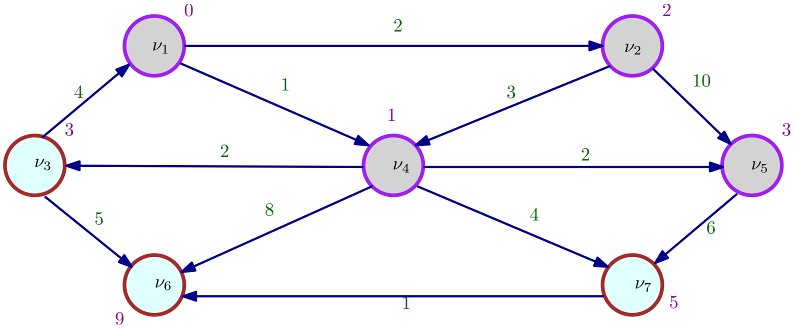
Example of Djikstra in action



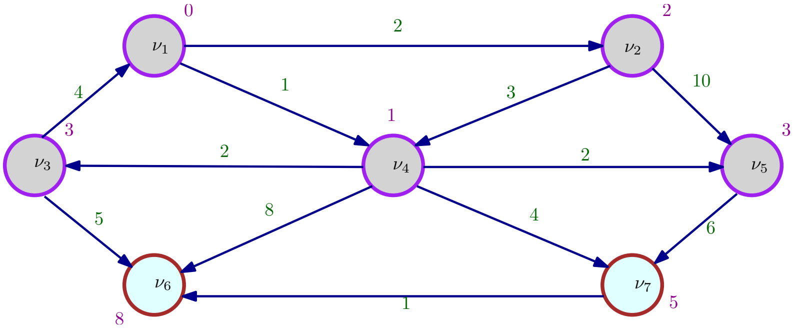
Example of Djikstra in action



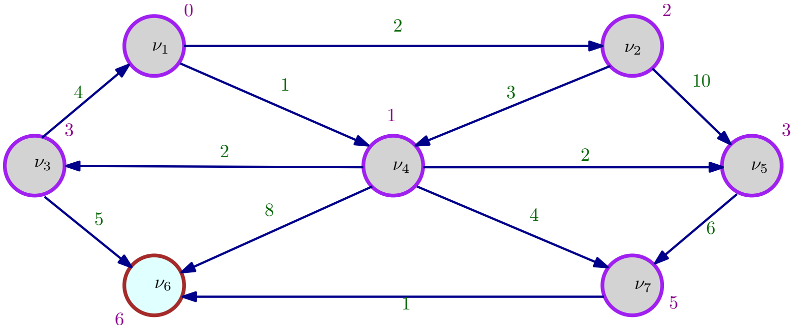
Example of Djikstra in action



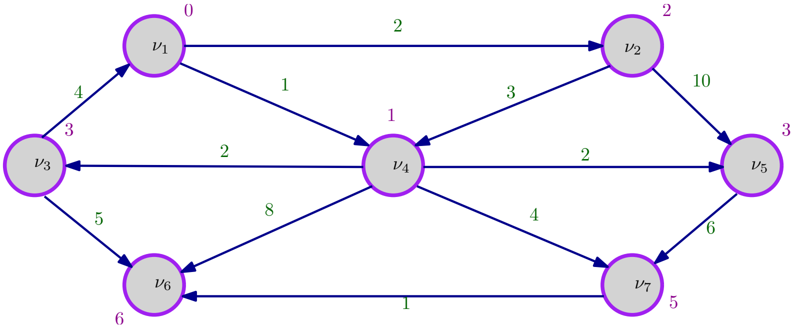
Example of Djikstra in action



Example of Djikstra in action



Example of Djikstra in action



How efficient is Djikstra

Naive implementation : $O(|E| + |V^2|) = O(|V|^2)$

How efficient is Djikstra

Naive implementation : $O(|E| + |V^2|) = O(|V|^2)$

Using a heap : $O(|E| \log |V|)$

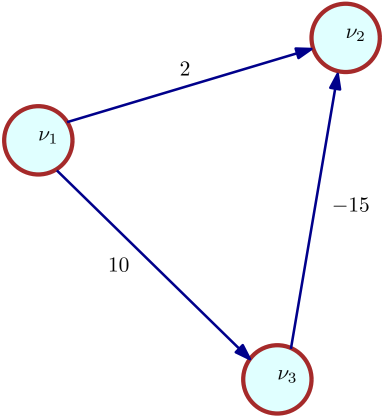
How efficient is Djikstra

Naive implementation : $O(|E| + |V^2|) = O(|V|^2)$

Using a heap : $O(|E| \log |V|)$

Using Fibonacci heap : $O(|E| + |V| \log |V|)$

Negative edge weights!



The Bellman Ford algorithm

Procedure Bellman-Ford(G, s) **begin**
    s.dist $\leftarrow 0$;
    for $i \leftarrow 1$ **to** $|V|$ **do**
        Relax each edge
    end
end

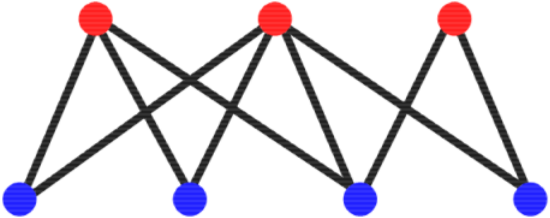
Runtime : $O(|E||V|)$

All-Pairs Shortest Path

- **Can run $|V|$ Djikstra's -** $O(|E||V|\log|V|)$
- **Floyd Warshall -** $O(|V|^3)$

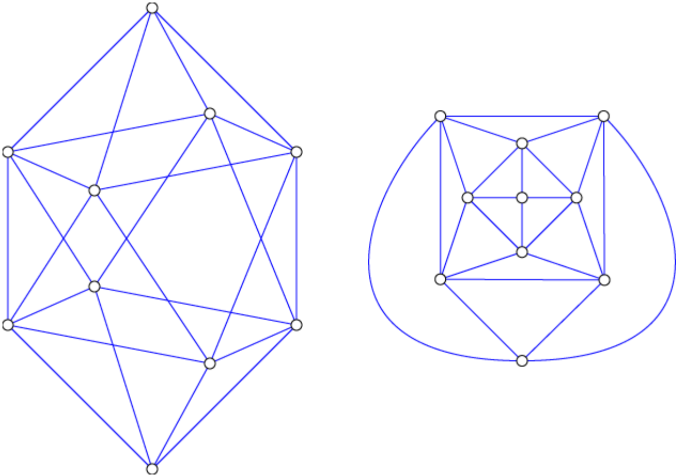
Graphs with special structure

Bipartite : $V = V_1 \cup V_2$, **edges only between** V_1, V_2



Graphs with special structure

Planar: Can be drawn in the plane without crossings - sparse



Back to networks

How can we model real networks?

Back to networks

How can we model real networks?

Doing mathematical analysis on the abstracted models helps us understand networks

Models should fit

Models should be guided by observed properties

Models should fit

Models should be guided by observed properties

Test their computed properties against observed ones

Some common network models

- **Random graph models - why?**
- **Random (n, m) graphs**
- **Erdös-Rényi model**
- **Albert-Barbási model - preferential attachment**
- **Watts Strogatz model - small world phenomena**

Summary statistics and Performance metrics

- **Diameter and mean path length**
- **Clustering, average clustering, global clustering**
- **Degree distribution**
- **Centrality and its distribution**

The definitions

Clustering coefficient: $C_i = 2L_i/k_i(k_i - 1)$

Global clustering coefficient : $\frac{3\times\textbf{Number of triangles}}{\textbf{Number of connected triples}}$

Betweenness Centrality

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

Why so many network models are needed?

- **Erdös-Rényi model does not capture gloabl clustering**

- **Watts-Strogatz model captures small world properties**

- **Networks *grow* - how can we accomodate that?**

Graphs algorithms - the message

Network Science is important

Graphs algorithms - the message

Network Science is important

Understanding graphs and computing with them is the key to understanding networks better

Graphs algorithms - the message

Network Science is important

Understanding graphs and computing with them is the key to understanding networks better

Need approximate but very fast algorithms for basic tasks