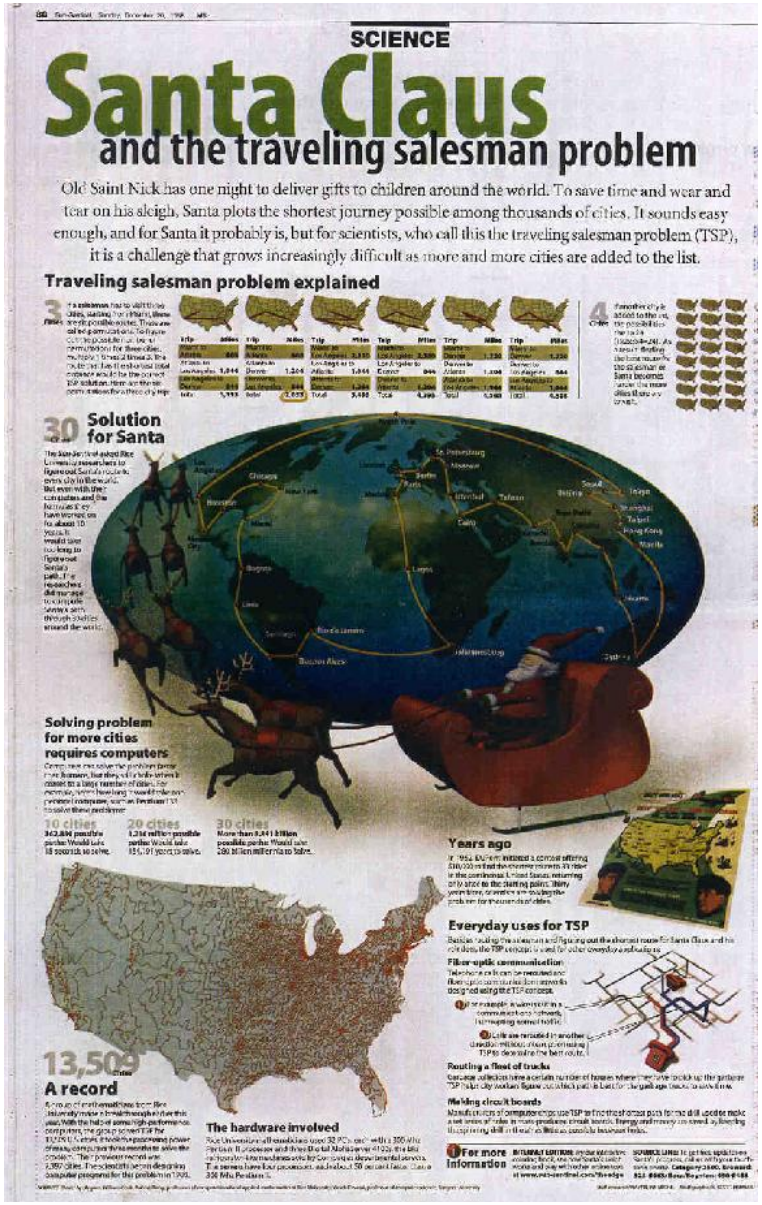
Arthur Charpentier
arthur.charpentier@univ-rennes1.fr
https://freakonometrics.github.io/
Université Rennes 1, 2017

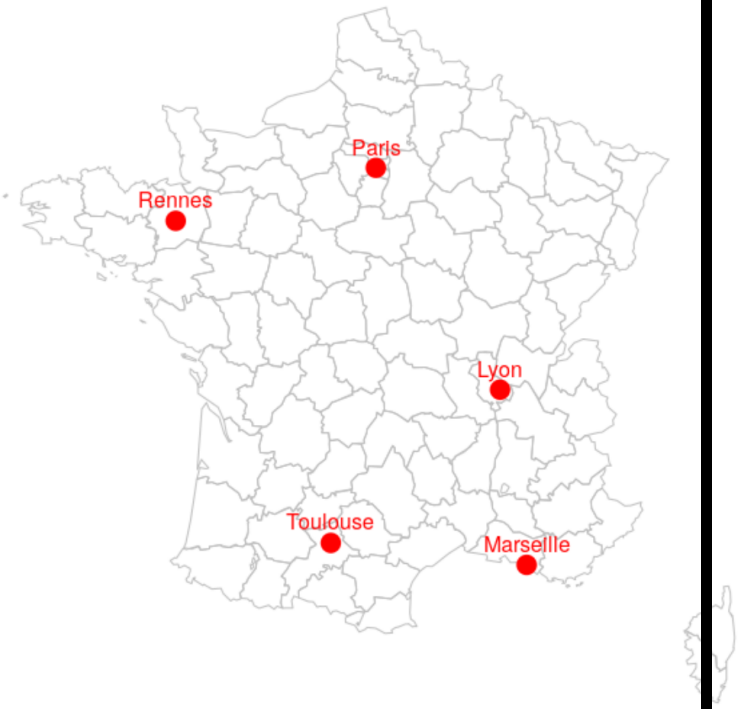
Graphs, Networks & Flows # 1



Travelling Salesman Problem

Consider 5 cities,

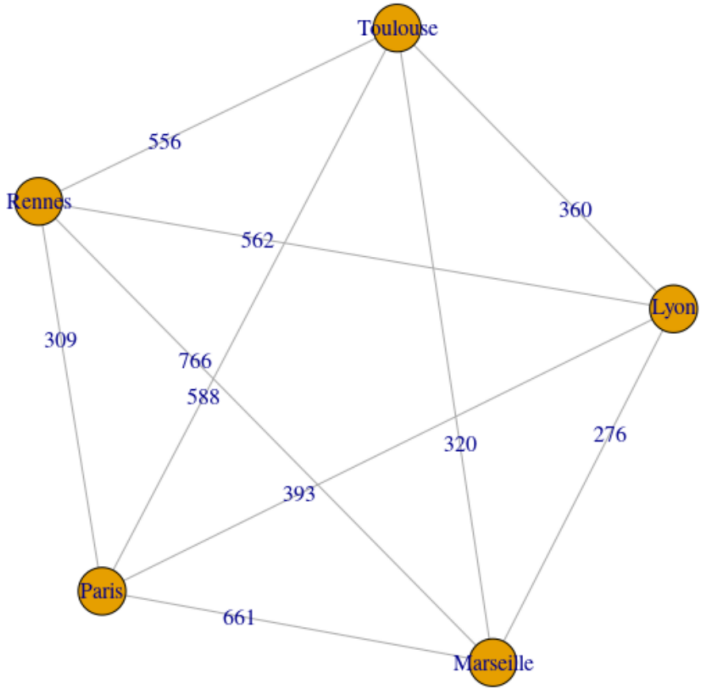
```
> v=c("Rennes","Paris","Lyon","Marseille","
    Toulouse")
> x=c(-1.6742900,2.348800,4.846710,
    5.38107,1.443670)
> y=c(48.1119800,48.853410,45.748460,
    43.29695,43.6042600)
> library(maps)
> france<-map(database="france",col="grey")
> points(x,y,pch=19,cex=2,col="red")
> text(x,y,v,pos=3,col="red")
```



It can also be seen as a complete weighted graph.

Travelling Salesman Problem

```
> df=data.frame(name=v,lat=y,lon=x,index
    =1:5)
> D=round(GeoDistanceInMetresMatrix(df)/
    1000)
> library(igraph)
> i=c(rep(1,4),rep(2,3),rep(3,2),4)
> j=c(2:5,3:5,4:5,5)
> df=data.frame(a = i, b=j, w=diag(D[i,j]))
> g=graph.data.frame(df, directed=FALSE)
> V(g)$label=v
> plot(g, edge.label=E(g)$w)
```



What is the path with minimal cost (length) that goes through all cities (and returns back home) ?

Travelling Salesman Problem

Can be formulated as an integer linear programming problem. Set x_{ij} be the dummay variable with value 1 if the path goes from city i to city j, u_i be some dummy variable, and c_{ij} denote the distance from city i to city j. Consider

$$\min \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij}$$

$$0 \leq x_{ij} \leq 1 \qquad i, j = 1, \ldots, n;$$

$$u_i \in \boldsymbol{Z} \qquad i = 1, \ldots, n;$$

$$\sum_{i=1, i \neq j}^{n} x_{ij} = 1 \qquad j = 1, \ldots, n;$$

$$\sum_{j=1, j \neq i}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n;$$

$$u_i - u_j + n x_{ij} \leq n - 1 \qquad 2 \leq i \neq j \leq n.$$

Travelling Salesman Problem

Let $\boldsymbol{D} = [D_{i,j}]$ denote the distance matrix, with $D_{i,i} = \infty$, $c_{i,j}$ otherwise

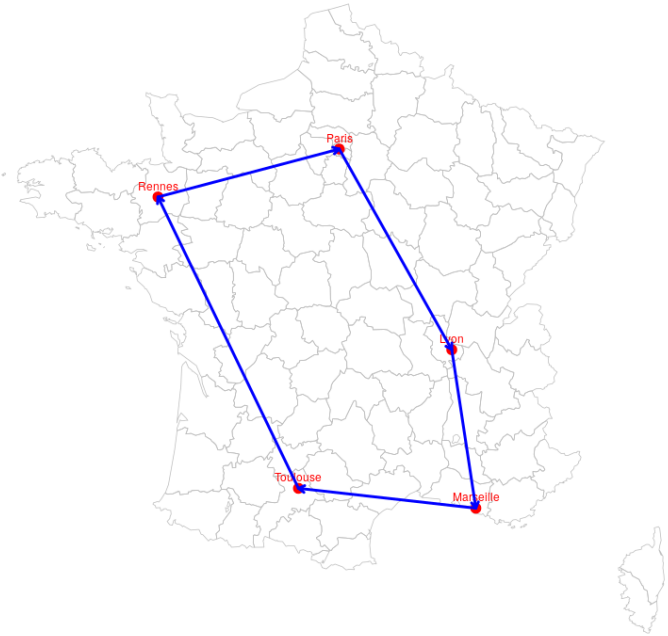
	Rennes	Paris	Lyon	Marseille	Toulouse
Rennes	∞	309	562	766	556
Paris	309	∞	393	661	588
Lyon	562	393	∞	276	360
Marseille	766	661	276	∞	320
Toulouse	556	588	360	320	∞

5 cities, $5! = 24$ possibilities (with assymmetric costs, otherwise 12).

Travelling Salesman Problem

One can use brute force search

```
> library(combinat)
> M=matrix(unlist(permn(1:5)),nrow=5)
> sM=M[,M[1,]==1]
> sM=rbind(sM,1)
> traj=function(vi){
+   s=0
+   for(i in 1:(length(vi)-1)) s=s+D[vi[i],
    vi[i+1]]
+   return(s) }
> d=unlist(lapply(1:ncol(sM),function(x)
    traj(sM[,x])))
> sM=rbind(sM,d)
> sM[,which.min(d)]
                                        d
  1    2    3    4    5    1 1854
```

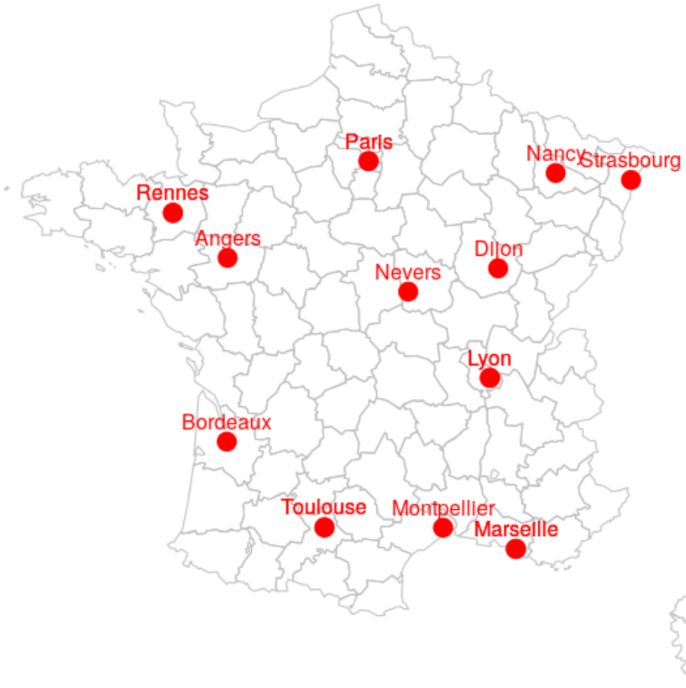


Travelling Salesman Problem

12 cities, $12! = 39916800$ possibilities (with assymmetric costs, otherwise 2 millions).

```
> v2=c(v,"Strasbourg","Angers","Bordeaux","
    Dijon","Nancy","Montpellier", "Nevers")
> x2=c(x
    ,7.75,-0.55,-0.566667,5.016667,6.2,3.883333

> y2=c(y
    ,48.583333,47.466667,44.833333,47.316667,48
     )
```



Little Algorithm

Branch and bound, Little algorithm, from Little, Murty, Sweeney & Karel (1963)
An algorithm for the traveling salesman problem

Step 1 find a lower bound for the total cost

Derive reduced matrix $\widetilde{\boldsymbol{D}}$ by removing to each raw the smallest element

	Rennes	Paris	Lyon	Marseille	Toulouse	
Rennes	∞	0	253	457	247	(-309)
Paris	0	∞	84	352	279	(-309)
Lyon	286	117	∞	0	84	(-276)
Marseille	490	385	0	∞	44	(-276)
Toulouse	236	268	40	0	∞	(-320)

Little Algorithm

and by removing to each column the smallest element

	Rennes	Paris	Lyon	Marseille	Toulouse	
Rennes	∞	0	253	457	203	(-309)
Paris	0	∞	84	352	235	(-309)
Lyon	286	117	∞	0	40	(-276)
Marseille	490	385	0	∞	0	(-276)
Toulouse	236	268	40	0	∞	(-320)
					(-44)	

Again $\boldsymbol{D}$ can be non-symmetric (and it will be afterwards).

Starting value for the cost: 309+309+276+276+320+44=1534

Little Algorithm

Step 2: compute regrets (min on rows and columns)

	Rennes	Paris	Lyon	Marseille	Toulouse	
Rennes	∞	0	253	457	203	(203)
Paris	0	∞	84	352	235	
Lyon	286	117	∞	0	40	
Marseille	490	385	0	∞	0	
Toulouse	236	268	40	0	∞	
		(117)				(320)

Little Algorithm

	Rennes	Paris	Lyon	Marseille	Toulouse	
Rennes	∞	0 (320)	253	457	203	
Paris	0	∞	84	352	235	(84)
Lyon	286	117	∞	0	40	
Marseille	490	385	0	∞	0	
Toulouse	236	268	40	0	∞	
	(236)					(320)

Little Algorithm

	Rennes	Paris	Lyon	Marseille	Toulouse	
Rennes	∞	0 (320)	253	457	203	
Paris	0 (320)	∞	84	352	235	
Lyon	286	117	∞	0	40	(40)
Marseille	490	385	0	∞	0	
Toulouse	236	268	40	0	∞	
				(0)		(40)

Little Algorithm

	Rennes	Paris	Lyon	Marseille	Toulouse
Rennes	∞	0 (320)	253	457	203
Paris	0 (320)	∞	84	352	235
Lyon	286	117	∞	0 (40)	40
Marseille	490	385	0 (40)	∞	0 (40)
Toulouse	236	268	40	0 (40)	∞

Select the maximal regret path (arbitrarily if not unique) : Rennes $\rightarrow$ Paris (RP).

Consider now two alternatives:

- either we keep (RP)
- either we remove (RP)

Little Algorithm

If we exclude (RP), the cost is known : 1534+320=1854.

If we include, consider the simplified matrix

	Rennes	Lyon	Marseille	Toulouse
Paris	∞	84	352	235
Lyon	286	∞	0	40
Marseille	490	0	∞	0
Toulouse	236	40	0	∞

Here (PR) is ∞ otherwise, it will be a round trip.

Little Algorithm

Then derive the reduced matrix

	Rennes	Lyon	Marseille	Toulouse	
Paris	∞	0	268	151	(-84)
Lyon	50	∞	0	40	(0)
Marseille	254	0	∞	0	(0)
Toulouse	0	40	0	∞	(0)
	(-236)				

Total (reduction) cost is 84+236=320.

Little Algorithm

And again, compute regrets

	Rennes	Lyon	Marseille	Toulouse
Paris	∞	0 (151)	268	151
Lyon	50	∞	0 (40)	40
Marseille	254	0 (0)	∞	0 (40)
Toulouse	0 (50)	40	0 (0)	∞

The maximal regret path is Paris $\rightarrow$ Lyon (PL). Consider now two alternatives:

- either we keep (PL)
- either we remove (PL) : cost will be 1534+320+151=1725

If we include, consider the simplified matrix (with possibly ∞ for (LP))

Little Algorithm

	Rennes	Marseille	Toulouse
Lyon	50	0	40
Marseille	254	∞	0
Toulouse	0	0	∞

Note that this matrix cannot be simplified, here.

	Rennes	Marseille	Toulouse	
Lyon	50	0	40	(0)
Marseille	254	∞	0	(0)
Toulouse	0	0	∞	(0)
	(0)	(0)	(0)	

So the total cost if we keep (RL) is 1534+320=1584

Little Algorithm

Then compute regrets

	Rennes	Marseille	Toulouse
Lyon	50	0 (40)	40
Marseille	254	∞	0 (294)
Toulouse	0 (50)	0 (0)	∞

The maximal regret path is Marseille $\rightarrow$ Toulouse (MT)

Consider now two alternatives:

- either we keep (MT)
- either we remove (MT) : cost will be 1854+294

If we include, consider the simplified matrix (with ∞ for (TM))

	Rennes	Marseille
Lyon	50	0
Toulouse	0	∞

Here again no simplification. The regret matrix is

	Rennes	Marseille
Lyon	50	0 (50)
Toulouse	0 (50)	∞

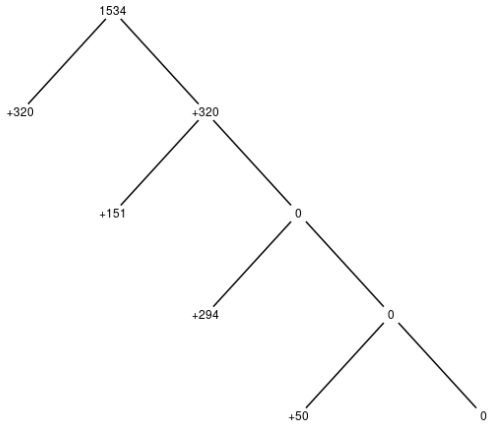
The maximal regret path is Lyon $\rightarrow$ Marseille (LM)

Consider now two alternatives:

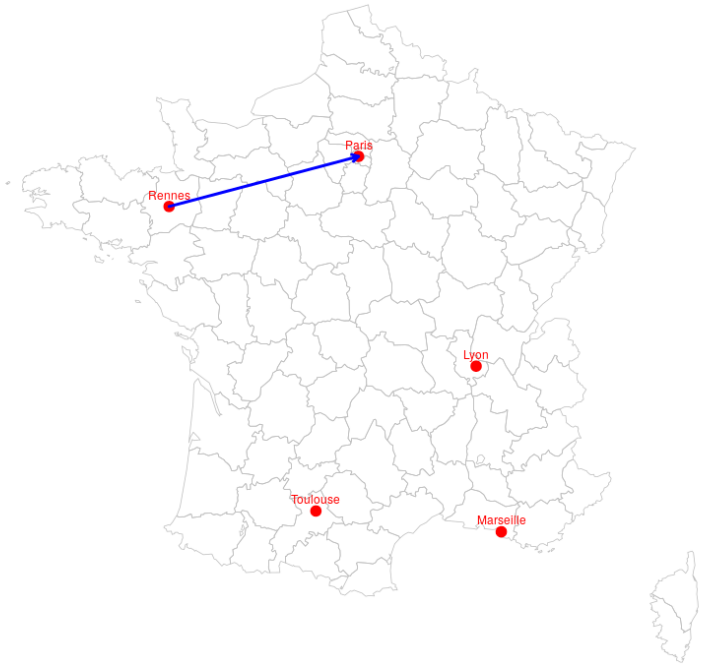
- either we keep (MT): cost will be 1854+0
- either we remove (MT) : cost will be 1854+50

Little Algorithm

Here we have constructed a search tree.



click to visualize the construction



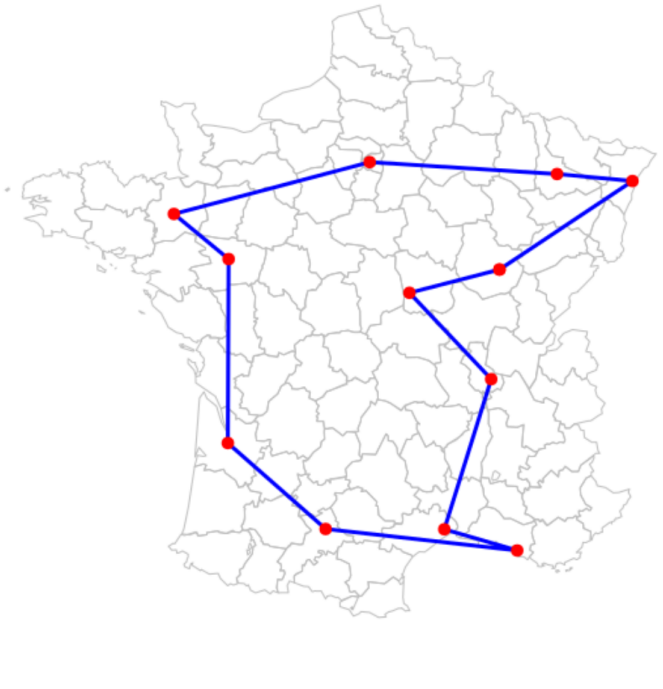
click to visualize the construction

Remark This is an exact algorithm

Travelling Salesman

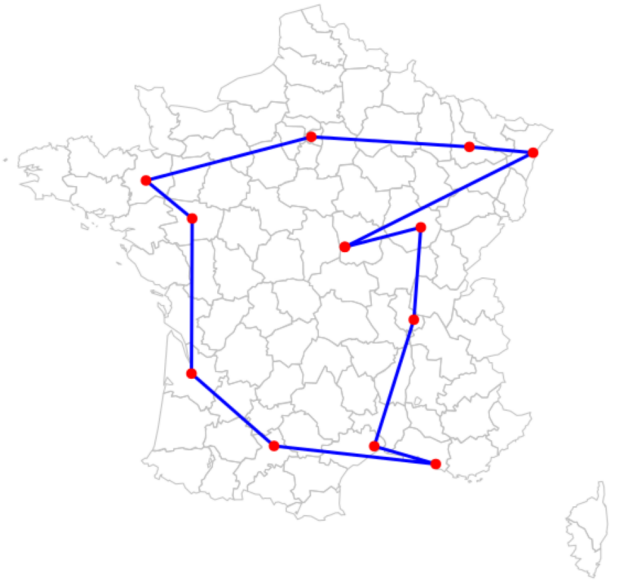
Exact algorithms are usually extremely slow. For Little Algorithm, the maximal number of nodes, with n cities, is of order N_n with $N_n = (n-1)[1+N_{n-1}]$ (with is of order $n!$).

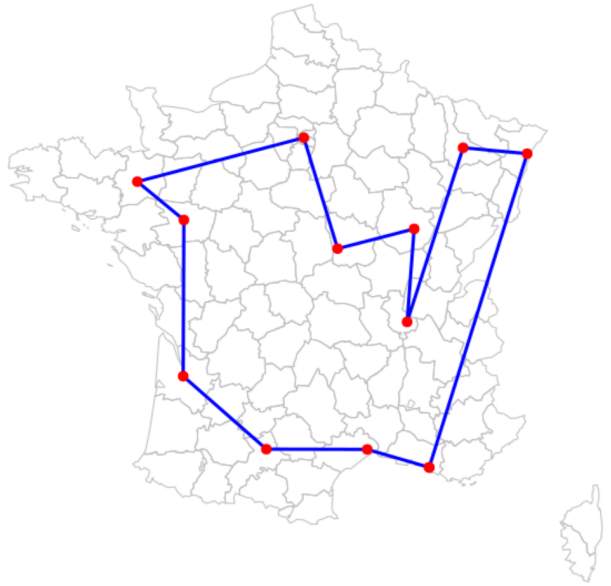
```
> library(TSP)
> df2=data.frame(name=v2,lat=y2,lon=x2,index
    =1:12)
> D2=round(GeoDistanceInMetresMatrix(df2)/
    1000)
> listeFR=TSP(D2,v2)
> tour=solve_TSP(listeFR, method = "nn")
> COORD=df2[as.numeric(tour),]
> COORD=rbind(COORD,COORD[1,])
> france<-map(database="france",col="grey")
> lines(COORD$lon,COORD$lat,lwd=3,col="blue"
    )
```



Travelling Salesman

```
> tour=solve_TSP(listeFR, method = "nn")
> COORD=df2[as.numeric(tour),]
> COORD=rbind(COORD,COORD[1,])
> france<-map(database="france",col="grey")
> lines(COORD$lon,COORD$lat,lwd=3,col="blue")
>
> tour=solve_TSP(listeFR, method = "nn")
> COORD=df2[as.numeric(tour),]
> COORD=rbind(COORD,COORD[1,])
> france<-map(database="france",col="grey")
> lines(COORD$lon,COORD$lat,lwd=3,col="blue")
```

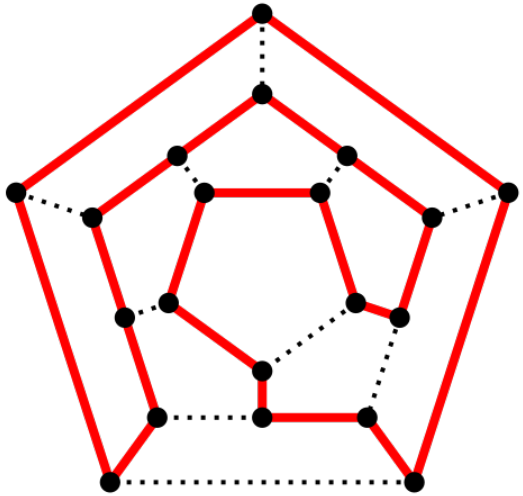




It is not stable... local optimization?

Travelling Salesman

Historically, introduced by William Rowan Hamilton in the context of dodecahedrons.
A Hamiltonian path is a path in agraph that visits each vertice exactly once
Done with $n = 20$ cities (regular aglebraic structure)



Dantzig, Fulkerson & Johnson (1954) Solution of a large-scale traveling-salesman problem with $n = 49$ cities (US State capitals)

In 2000, it was perfomed on $n = 15,000$ cities, nowadays close to 2 million.

See also Applegate *et al.* (2006) The traveling salesman problem: a computational study.

Stochastic Algorithm

One can use a Greedy Algorithm

Heuristically, makes locally optimal choices at each stage with the hope of finding a global optimum.

nearest neighbour, "*at each stage visit an unvisited city nearest to the current city*"

(1) start on an arbitrary vertex as current vertice.

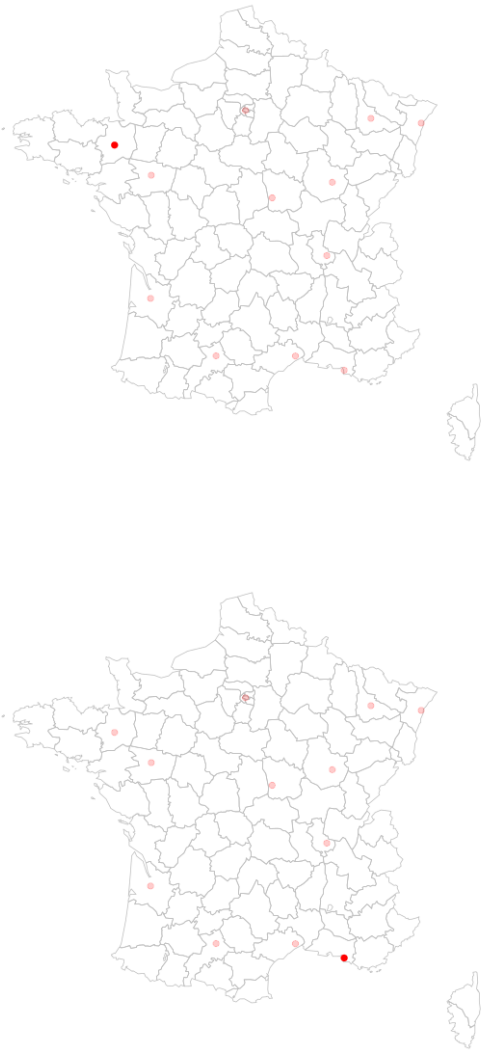
(2) find out the shortest edge connecting current vertice and an unvisited vertice v.

(3) set current vertice to v.

(4) mark v as visited.

(5) if all the vertices in domain are visited, then terminate.

(6) Go to step 2.



click to visualize the construction

Stochastic Algorithm

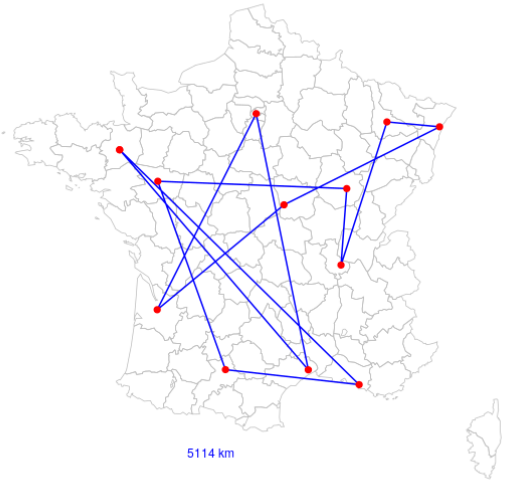
E.g. descent algorithm, where we start from an initial guess, then consider at each step one vertice s, and in the neighborhood of s, denoted $V(s)$ search

$$f(s^\star) = \min_{s' \in V(s)} \{f(s')\}$$

If $f(s^\star) < f(s)$ then $s = s^\star$.

See Lin (1965) with 2 opt or 3 opt, see also Lin & Kernighan (1973) An Effective Heuristic Algorithm for the Traveling-Salesman Problem

The pairwise exchange or 2-opt technique involves iteratively removing two edges and replacing these with two different edges that reconnect the fragments created by edge removal into a new and shorter tour.

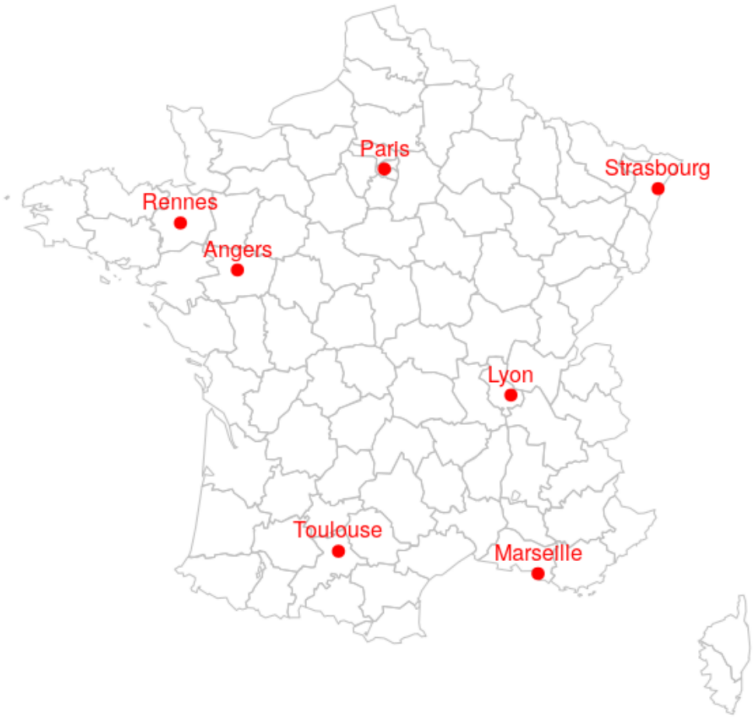


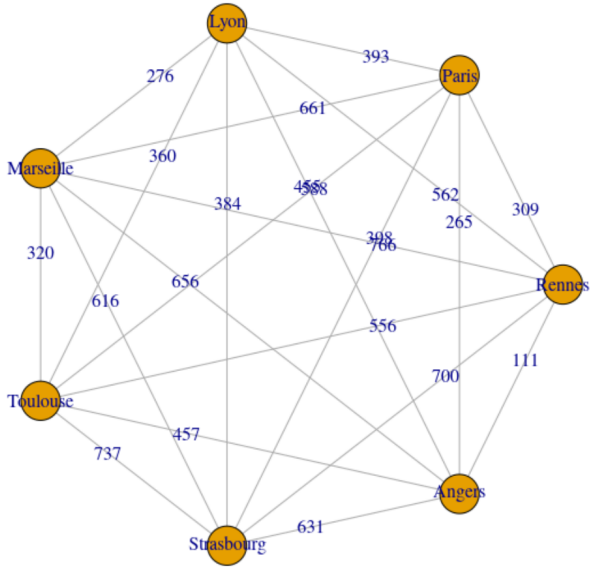
click to visualize the construction

Spanning Algorithm

Kruskal's algorithm is a minimum-spanning-tree algorithm, see Kruskal (1956)

On the shortest spanning subtree of a graph and the traveling salesman problem



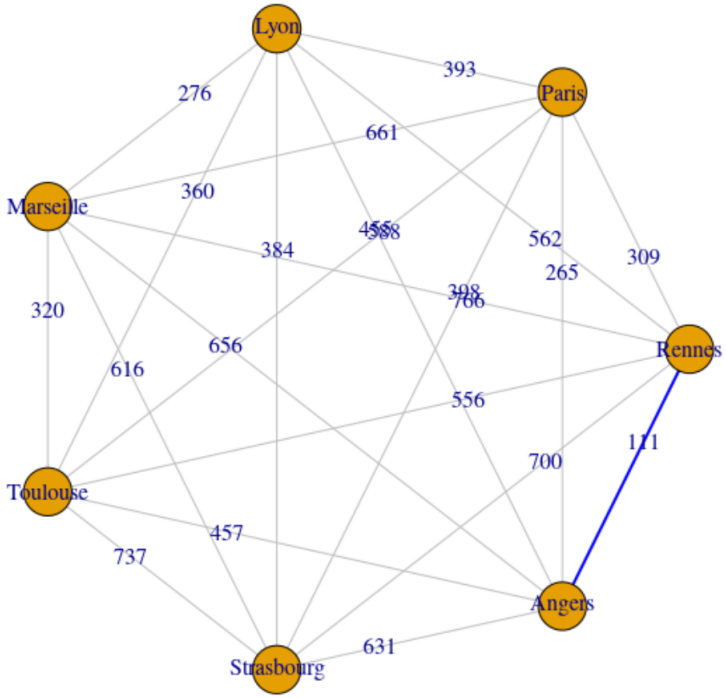


Spanning Algorithm

Select edge $e = (u, v) \in E$ which minimizes ω_ϵ,

$$e = \underset{\epsilon \in E}{\operatorname{argmin}}\{\omega_\epsilon\}$$

Then remove e from the set of electible edges
$\mathcal{E} = E \backslash \{e\}$
and mark that edge $\mathcal{F} = \{e\}$



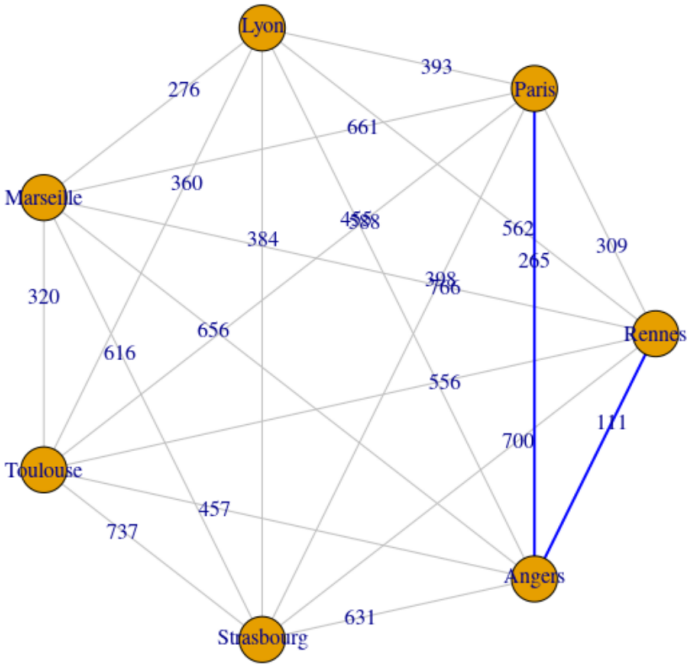
Spanning Algorithm

Select edge $e = (u, v) \in \mathcal{E}$ which minimizes ω_e,

$$e = \underset{\epsilon \in \mathcal{E}}{\text{argmin}}\{\omega_\epsilon\}$$

Then remove e from the set of electible edges
$\mathcal{E} = \mathcal{E} \backslash \{e\}$
and mark that edge $\mathcal{F} = \mathcal{F} \cup \{e\}$

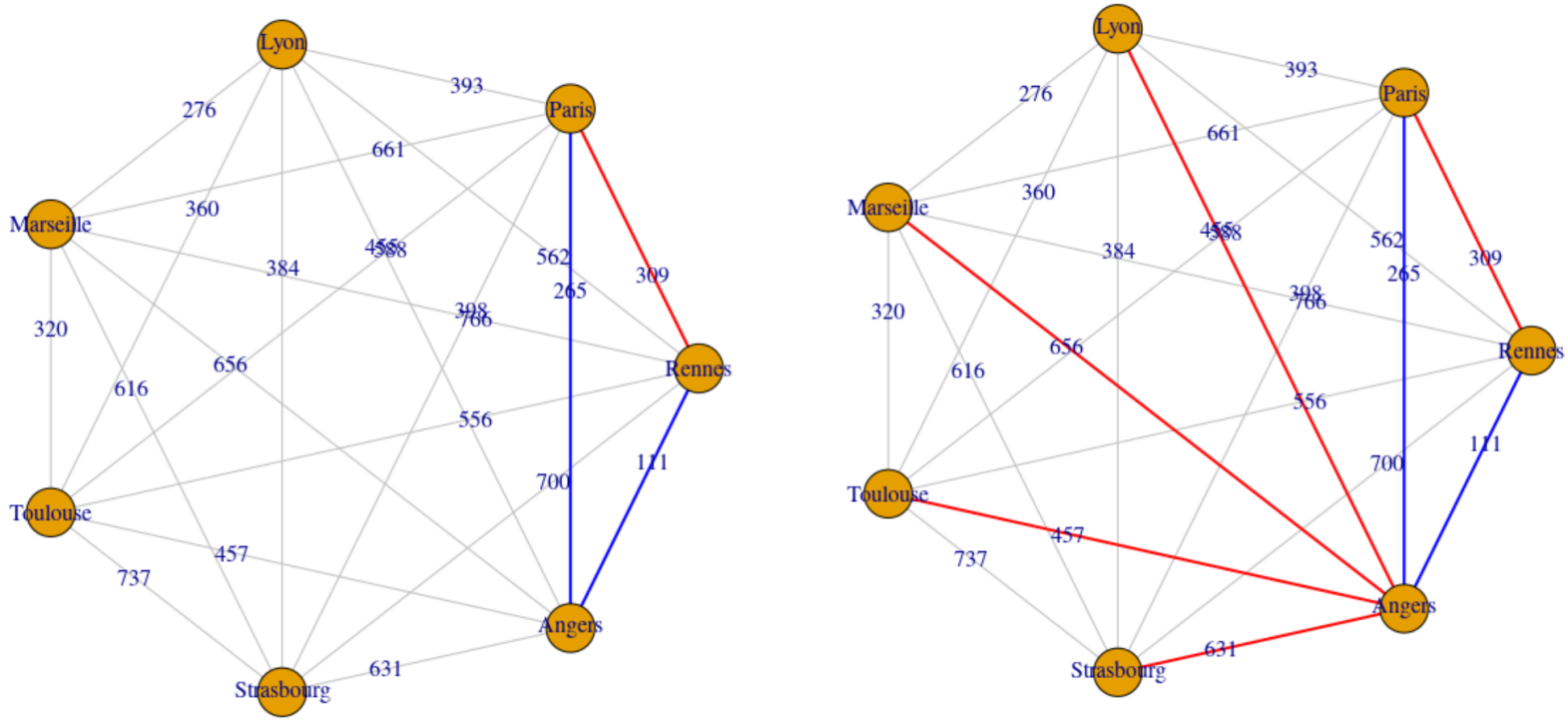
If the minima is not unique, choose randomly



Spanning Algorithm

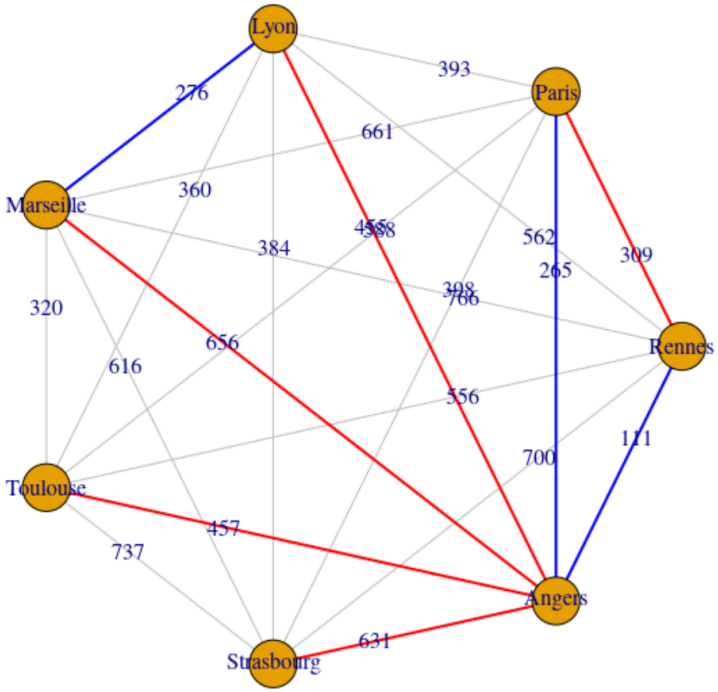
$\mathcal{E} = \mathcal{E} \backslash \{E_1 \cup E_2\}$ where we remove

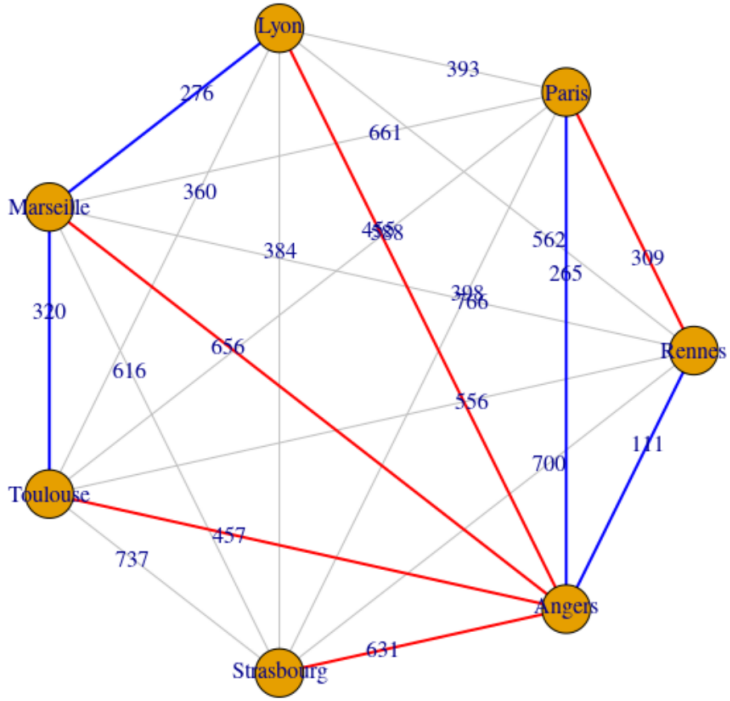
- all edges E_1 such that $\mathcal{E} \cup E_1$ contains a cycle (here Rennes-Paris)
- all edges from a node that appears twice in $\mathcal{E}$ (here all edges connected to Angers)



Spanning Algorithm

As long as there are no cycle, set $e = \underset{\epsilon \in \mathcal{E}}{\text{argmin}}\{\omega_\epsilon\}$ and then remove e from the set of electible edges $\mathcal{E} = \mathcal{E} \backslash \{e\}$, and mark those $\mathcal{F} = \mathcal{F} \cup \{e\}$

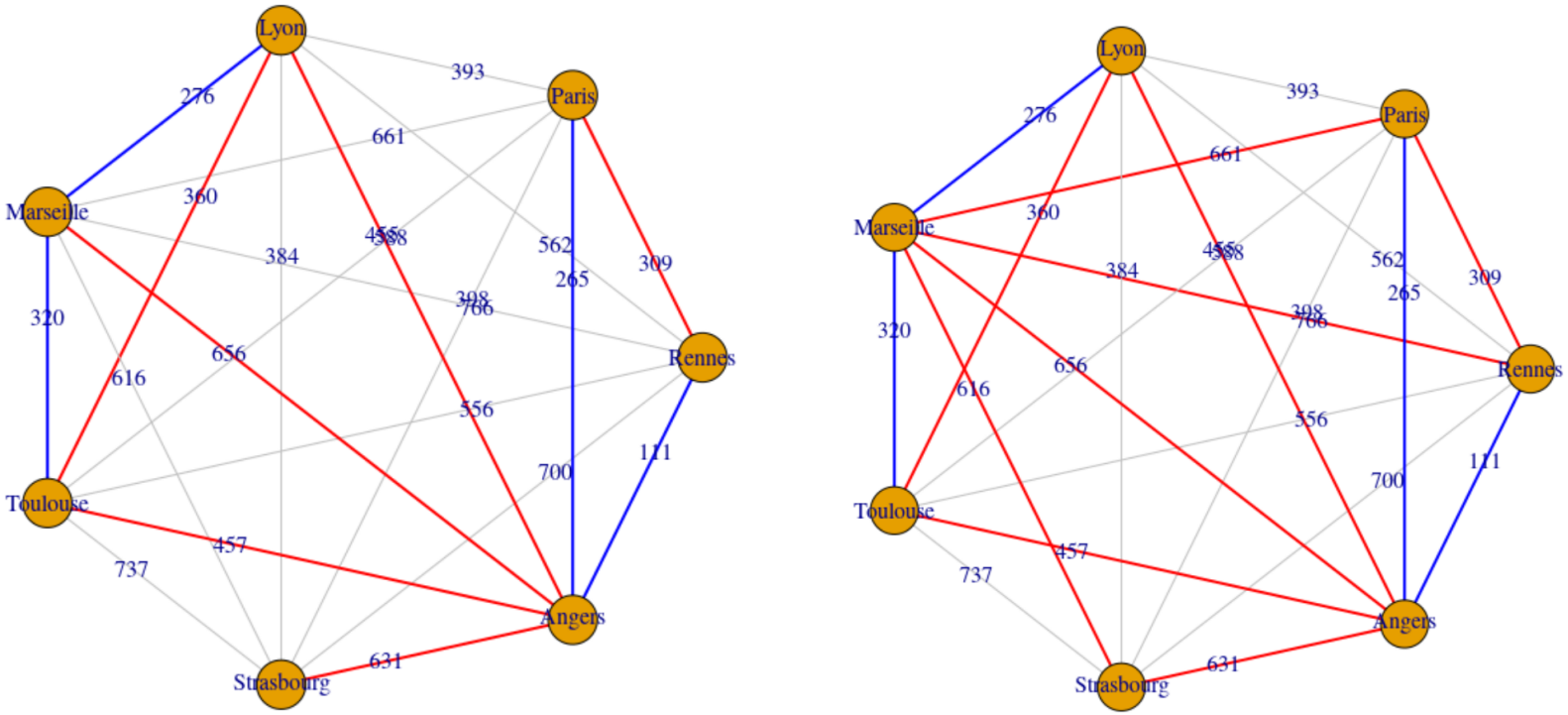




Spanning Algorithm

Here again remove

- all edges E_1 such that $\mathcal{E} \cup E_1$ contains a cycle (here Rennes-Paris)
- all edges from a node that appears twice in $\mathcal{E}$ (here all edges connected to Angers)

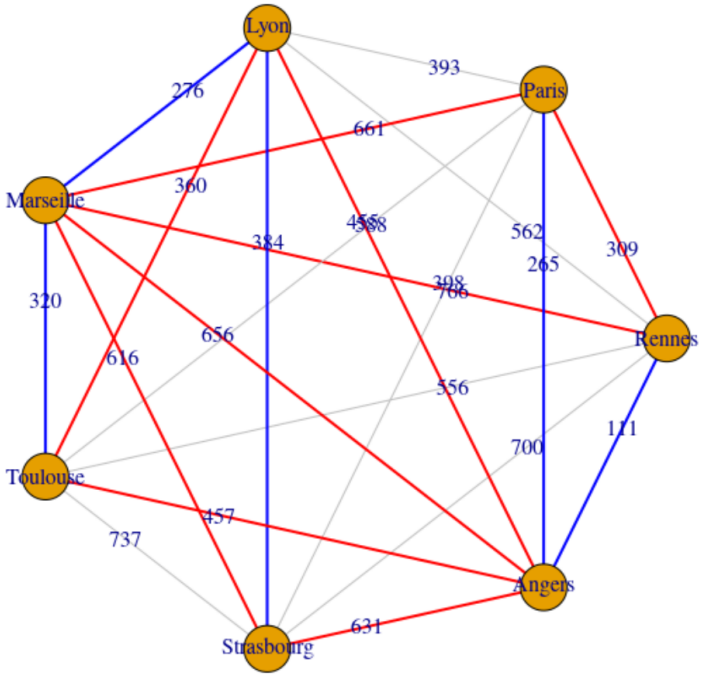


Spanning Algorithm

Select edge $e = (u, v) \in \mathcal{E}$ which minimizes ω_e,

$$e = \underset{\epsilon \in \mathcal{E}}{\text{argmin}}\{\omega_\epsilon\}$$

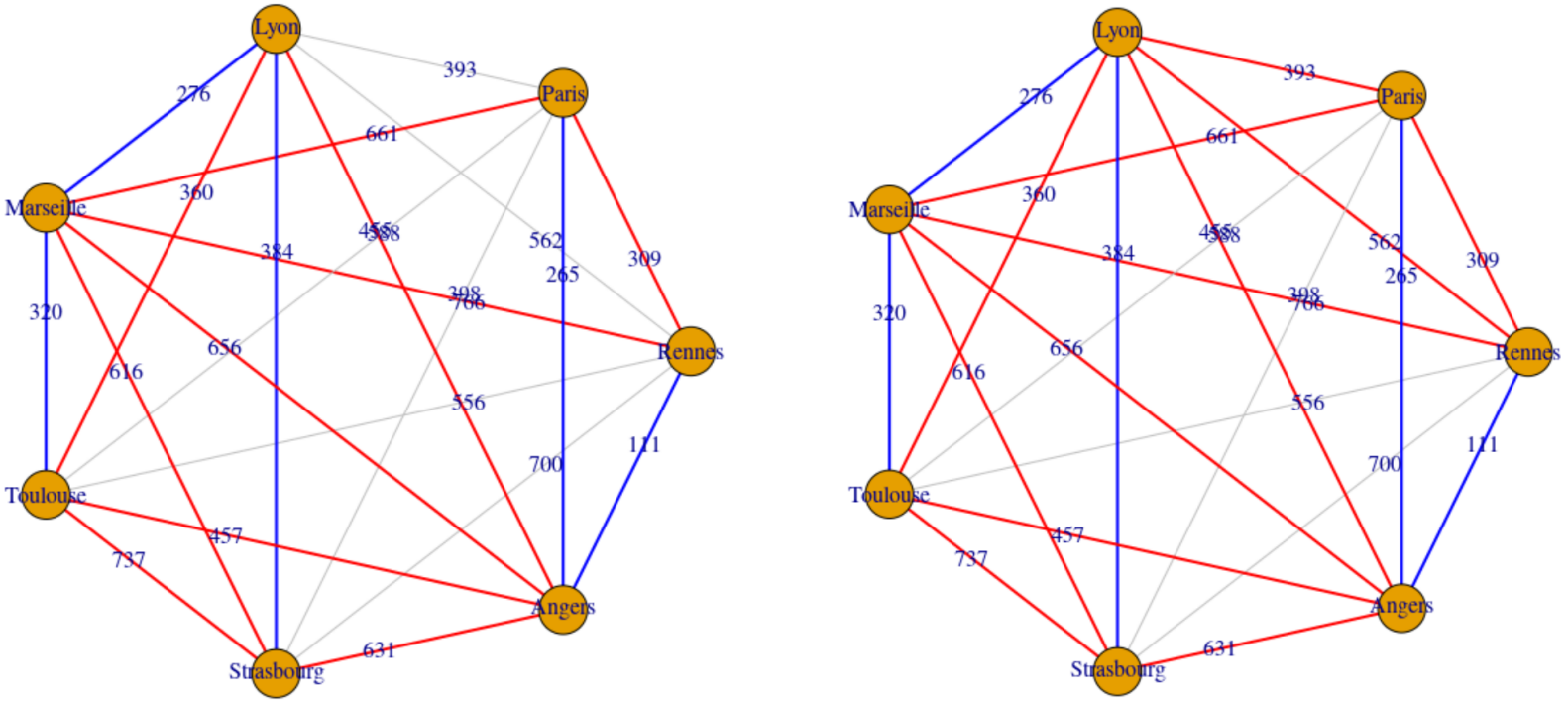
Then remove e from the set of electible edges
$\mathcal{E} = \mathcal{E} \backslash \{e\}$
and mark it $\mathcal{F} = \mathcal{F} \cup \{e\}$



Spanning Algorithm

Remove

- all edges E_1 such that $\mathcal{E} \cup E_1$ contains a cycle (here Rennes-Paris)
- all edges from a node that appears twice in $\mathcal{E}$ (here all edges connected to Angers)



Spanning Algorithm

Select edge $e = (u, v) \in \mathcal{E}$ which minimizes ω_ϵ,

$$e = \underset{\epsilon \in \mathcal{E}}{\text{argmin}}\{\omega_\epsilon\}$$

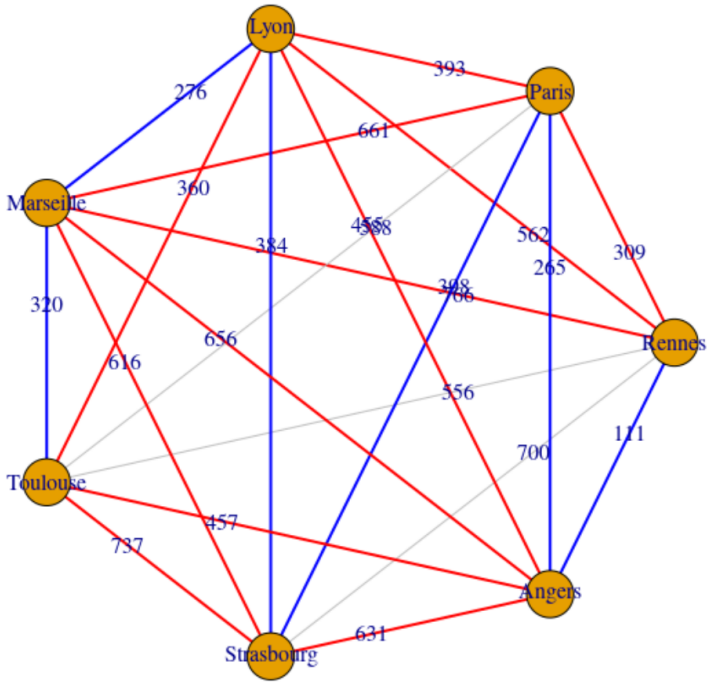
Then remove e from the set of electible edges

$\mathcal{E} = \mathcal{E} \backslash \{e\}$

and mark it $\mathcal{F} = \mathcal{F} \cup \{e\}$

Here e=Strasbourg-Paris

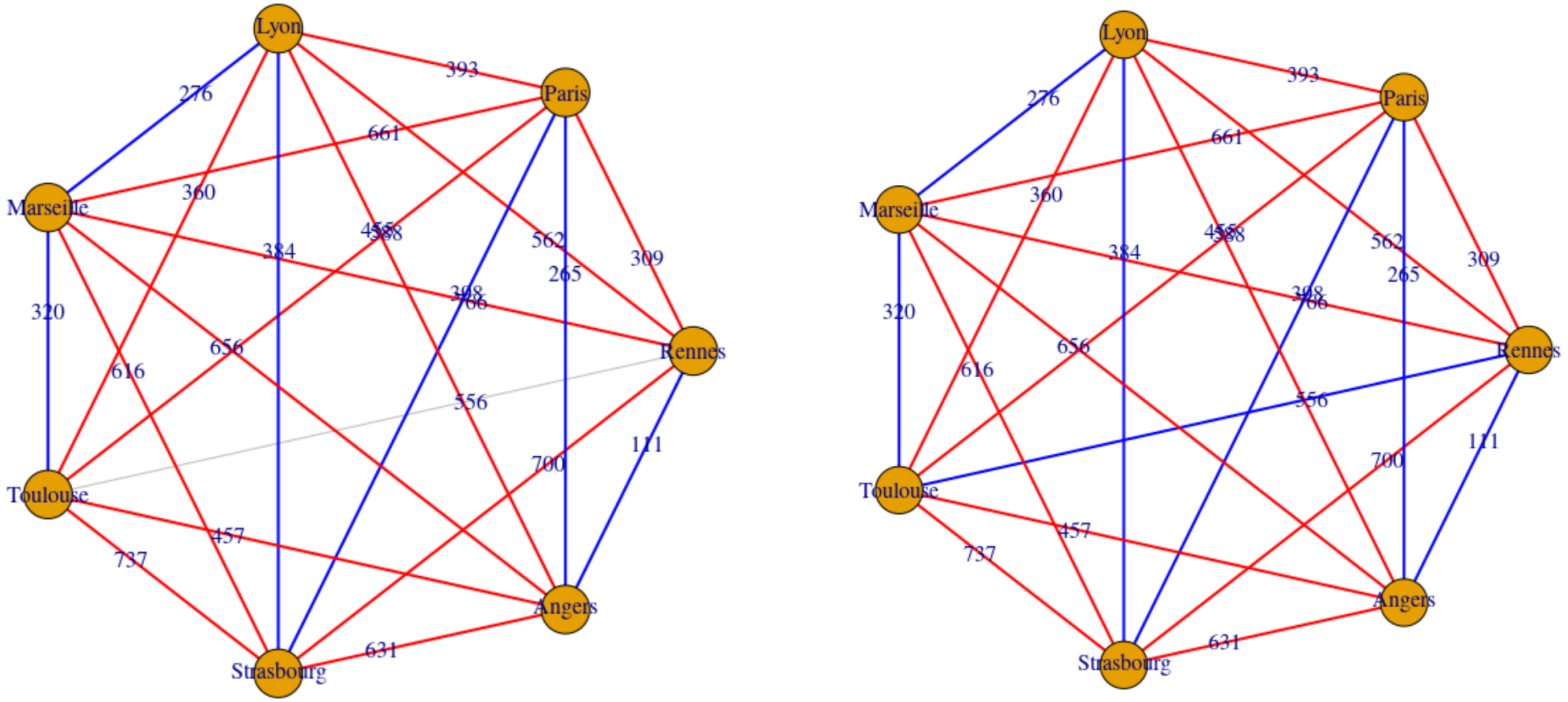
If the minima is not unique, choose randomly



Spanning Algorithm

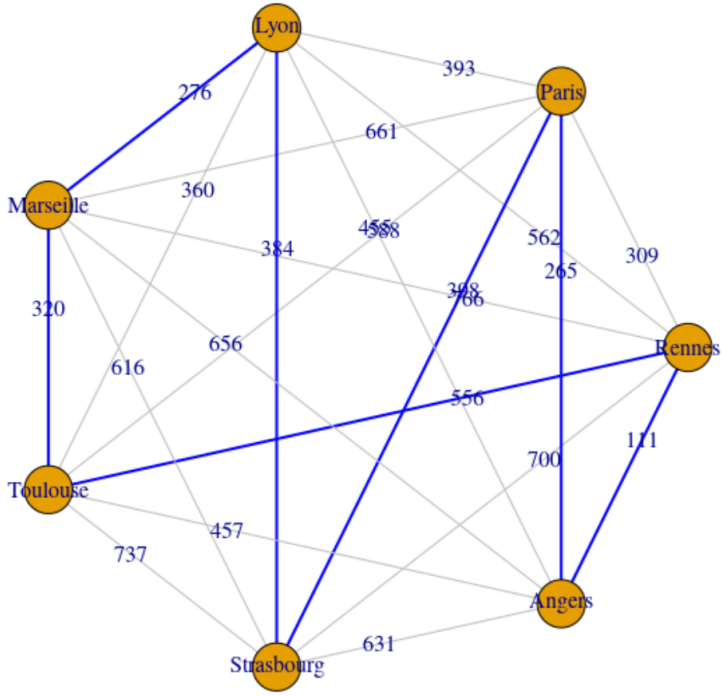
Last but not least, remove

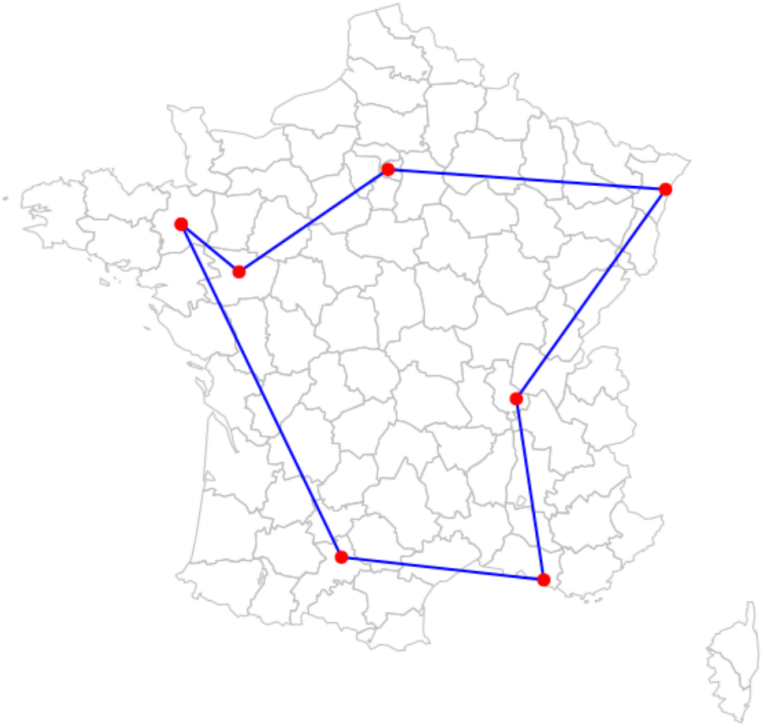
- all edges E_1 such that $\mathcal{E} \cup E_1$ contains a cycle (here Rennes-Paris)
- all edges from a node that appears twice in $\mathcal{E}$ (here all edges connected to Angers)



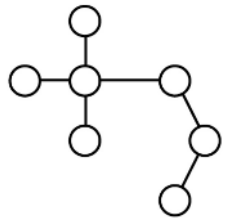
Spanning Algorithm

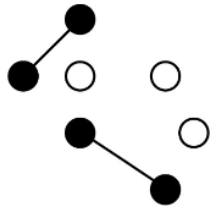
The collection of edges that were marked in $\mathcal{F}$ is our best cycle.





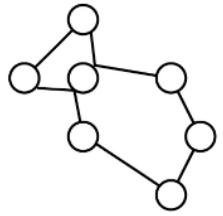
Christofides Algorithm





Use a spanning algorithm (as in Kruskal algorithm) $G = (V, E)$ but allow for vertices to have more than 2 incident edges.

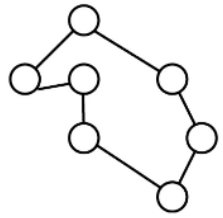
Let V_1 denote the set of vertices with odd degree, and let $G_1 = (V_1, E_1)$ denote the induced subgraph of $G = (V, E)$.



Find a perfect coupling G_1' of G_1 (discussed in the next course).

Consider graph G' with vertices V and edges $E \cup E_1'$.

If there are still vertices v with degree exceeding 2, consider shortcuts i.e. (u, v) and (v, w) becomes (u, w).

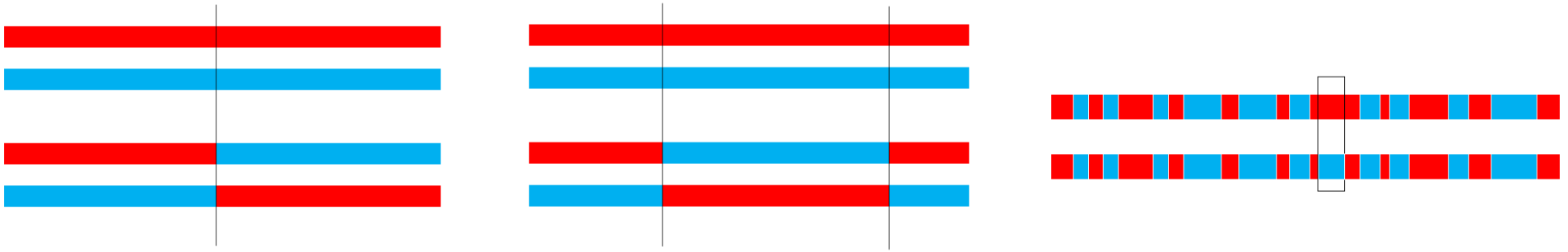


Genetic Algorithm

See Potvin (1996) Genetic algorithms for the traveling salesman problem

Mutation can be used to diversify solutions by alterating (randomly) an individual (possibly bit string)

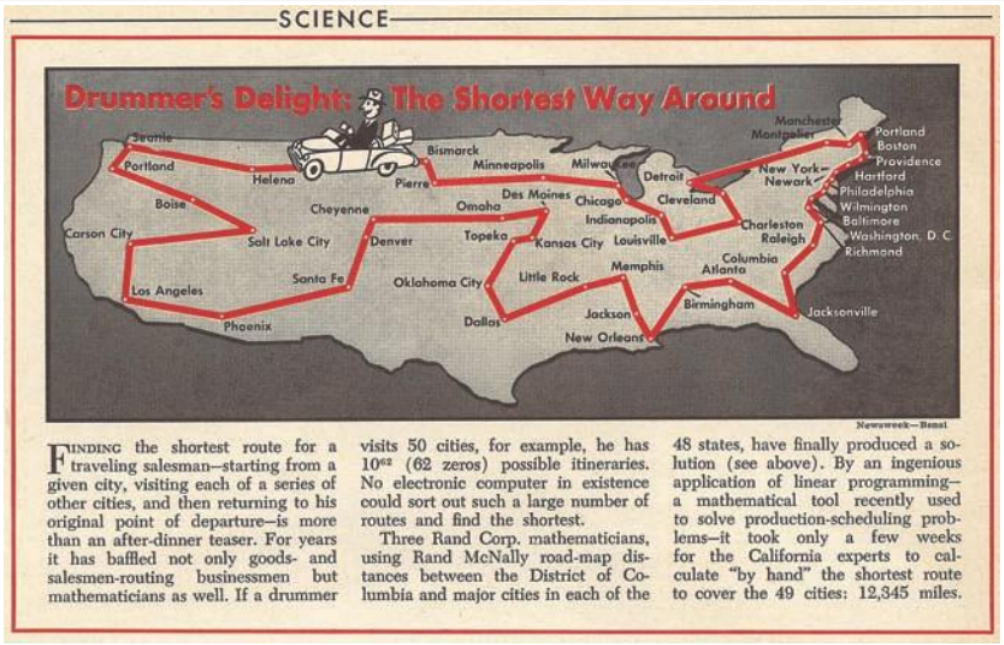
Cross-Over is obtained by mixing two individuals (single point or two points)



Travelling Salesman

Can be performed on a (much) larger scale
e.g. 48 State capitals in mainland U.S.

see Cook (2011) In Pursuit of the Traveling Salesman
for a survey.



SCIENCE

Drummer's Delight: The Shortest Way Around

Newsweek—Benzi

Finding the shortest route for a traveling salesman—starting from a given city, visiting each of a series of other cities, and then returning to his original point of departure—is more than an after-dinner teaser. For years it has baffled not only goods- and salesmen-routing businessmen but mathematicians as well. If a drummer visits 50 cities, for example, he has 10^{62} (62 zeros) possible itineraries. No electronic computer in existence could sort out such a large number of routes and find the shortest.

Three Rand Corp. mathematicians, using Rand McNally road-map distances between the District of Columbia and major cities in each of the 48 states, have finally produced a solution (see above). By an ingenious application of linear programming—a mathematical tool recently used to solve production-scheduling problems—it took only a few weeks for the California experts to calculate "by hand" the shortest route to cover the 49 cities: 12,345 miles.