Home Work Movement Solver & Generator

Federico Motta

Università di Modena e di Reggio Nell'Emilia

191685@studenti.unimore.it

August 14, 2015

Indice

Cosa è il Car Pooling ?

Per car pooling si intende la condivisione di un collettivo di utenti dei mezzi di trasporto.
Ovvero nel caso in esame i dipendenti di un'azienda invece di recarsi al lavoro autonomamente, ognuno con la propria automobile, condividono la propria auto con i colleghi, passando a prenderne 4 prima di arrivare all'azienda.
Il tutto è finalizzato al risparmio globale (sia esso considerato in km o litri di carburante), alla riduzione dell'inquinamento dell'ambiente ed alla riduzione del traffico e del sovraffollamento dei parcheggi.

Strategie adottate

- Car Pooling (appena descritto)
- Minibus (Un mezzo con 10 posti più l'autista è reso disponibile dall'azienda ai dipendenti)

Formalizzazione

Input

Un grafo costituito da luoghi.
Un luogo può essere l'azienda, il deposito, un incrocio o una casa.
Ogni luogo ha un ID, e due cordinate (x; y).
Gli archi del grafo sono orientati e rappresentano strade a senso unico (per una strada a doppio senso di marcia occorre inserire due archi).
In input viene anche passato il numero di bus presenti nel deposito.

Formalizzazione

Input

Un grafo costituito da luoghi.
Un luogo può essere l'azienda, il deposito, un incrocio o una casa.
Ogni luogo ha un ID, e due cordinate (x; y).
Gli archi del grafo sono orientati e rappresentano strade a senso unico (per una strada a doppio senso di marcia occorre inserire due archi).
In input viene anche passato il numero di bus presenti nel deposito.

Output

In output viene restituito un elenco di routes ed il costo totale della soluzione proposta.
Una route è una sequenza di luoghi attraversati, ognuno dei quali ha un flag booleano che indica se in quel luogo (casa) è stato prelevato il dipendente che vi abita.
Ogni route parte da una casa o dal deposito e termina all'azienda.

Esempio di file di input

Example (input.dat)

```
Ax: [   0.0  ]   Dx: [  61.81 ]   Bus: [  50    ]
Ay: [  -4.0  ]   Dy: [ -31.41 ]

Ix: [   ... array  ascisse  degli incroci ...   ]
Iy: [   ... array  ordinate degli incroci ...   ]

Cx: [   ... array  ascisse  delle    case ...   ]
Cy: [   ... array  ordinate delle    case ...   ]

St: [   ... array  di coppie di    interi ...   ]
```

nota: l'array St è costituito da coppie di ID dei luoghi connessi da strade (da, a).

Esempio di file di output

Example (output.txt)

```
Cost of best solution = 43'090,003
Route 0 (Car):
 800 952 915 953 739 966 384 248 235 677 450 0
  T   T   T   T   F   T   F   F   F   F   F  F
Route 1 (Car):
 984 943 803 675 879 350 808 360 140 845 771 552 0
  T   T   T   F   F   F   T   F   F   T   F   F  F
...   ...   ...   ...   ...   ...   ...   ...

Route N (Car):
 805 896 390 336 825 970 580 818 229 876 367 362 450 0
  T   T   F   F   T   T   F   T   F   F   F   F   F  F
```

Pre-processing

Dijkstra

```
function allMinDist
    for all { firm, depot, houses } do
        distance=Dijkstra(currentItem)
    end-for
end-function
```

Calcola tutti i cammini minimi tra case, deposito ed azienda e li salva in una matrice.

Heuristic (part 1)

```
function Heuristic
    for all { houses } do
        calculateThickCoeff(currentItem)
    end-for
```

Calcola i coefficienti di thickness di tutte le case e li salva in un array.

Heuristic (part 1)

```
function Heuristic
    for all { houses } do
        calculateThickCoeff(currentItem)
    end-for
```

Calcola i coefficienti di thickness di tutte le case e li salva in un array.

```
    for all (thickCoeff < thickThreshold) do
        createRoute(clusterOfCurrentItem)
    end-for
```

Crea le route a partire dai clusters di case.

Heuristic (part 2)

```
for all { houses not in any route } do
    for all { houses not in any route } do
        calculateSaving(item1,item2)
    end-for
end-for
```

Calcola i savings tra tutte le coppie di case rimaste, se l'item1 e l'item2 fossero in una stessa route rispetto al caso in cui fossero in routes diverse.

Heuristic (part 2)

```
for all { houses not in any route } do
    for all { houses not in any route } do
        calculateSaving(item1,item2)
    end-for
end-for
```

Calcola i savings tra tutte le coppie di case rimaste, se l'item1 e l'item2 fossero in una stessa route rispetto al caso in cui fossero in routes diverse.

```
for all { houses not in any route } do
    L=getLength(Distance(currentItem,firm))
end-for
R=reverseRandomizedQuickSort(L)
```

Calcola le lunghezze dei cammini minimi calcolati in precedenza, e le ordina in modo decrescente.

Heuristic (part 3)

```
for all { in R*farThreshold } do
    createEmptyRoute
    while (route emplyee < K employee)
        extractRandomPositiveSaving
        addToRoute(houseExtracted)
    end-while
end-for
```

Per una certa percentuale di dipendenti lontani rimasti, creo delle route ottime con colleghi aventi saving positivo scelti casualmente finché ognuna ha 5 - K posti liberi.

Heuristic (part 4)

```
    for all { houses not in any route } do
        createEmptyRoute
        while(route employee < W emplyee)
            getRandomSet(remaining houses)
            getGreaterPositiveSaving(randSet)
            addToRoute(collegue extracted)
        end-while
    end-for
    return solution
end-function
```

Per ogni casa rimanente crea una route vuota, e finché non ha 5 - W posti liberi o non ci sono più case senza route, si estrae la casa con saving maggiore e non nullo da un insieme ottenuto scegliendo casualmente le case tra quelle rimaste.

Mid-processing & Meta-heuristic (part 1)

function RemoveDuplicatedHeuristicSolution
end-function

Mid-processing & Meta-heuristic (part 1)

```
function RemoveDuplicatedHeuristicSolution
end-function
```

```
function Meta-heuristic
    for each set of 2 or 3 routes
        if (pickedEmployees < 10) than
            evaluateBusRoute(pickedEmplyees)
            if (busRouteCost<carRouteCost) than
                keepBusRoute
                removeCarRoutes
            end-if
        end-if
    end-for
```

Ogni gruppo di 2 o 3 auto con meno di 10 dipendenti prelevati, viene sostituito da un bus (se ciò conviene).

Meta-heuristic (part 2)

```
for all { routes } do
    for all { houses } do
        calculateSaving(item1, item2)
    end-for
end-for
```

Per ogni coppia route-casa si calcola il saving che si avrebbe se la casa venisse spostata dalla route in cui è (item2), a quella item1.
Le routes calcolate sono sempre ottime per le case incluse.

Meta-heuristic (part 3)

```
while (positiveSavingNumber > 0)
    if (signal stop) than
        break
    end-if
    applyGreaterSaving
    refreshInfluencedSavings
    if (percentProgress greater by 1 and currentSaving < 10 -4) than
        break
    end-if
end-while
```

Finché rimangono savings positivi, se si riceve il segnale stop si esce dalla ricerca locale altrimenti si estrae ed applica il saving maggiore. Quindi si aggiornano i savings inerenti le routes/case modificate. Se la percentuale di progresso è aumentata di 1 ed il saving corrente è minore di 10^{-4} si esce dalla ricerca locale, altrimenti si reitera.

Meta-heuristic (part 4)

```
    if (checkSolutionAdmissibility) than
        return solution
    else
        makeSolutionAdmissible
        return solution
    end-if
end-function
```

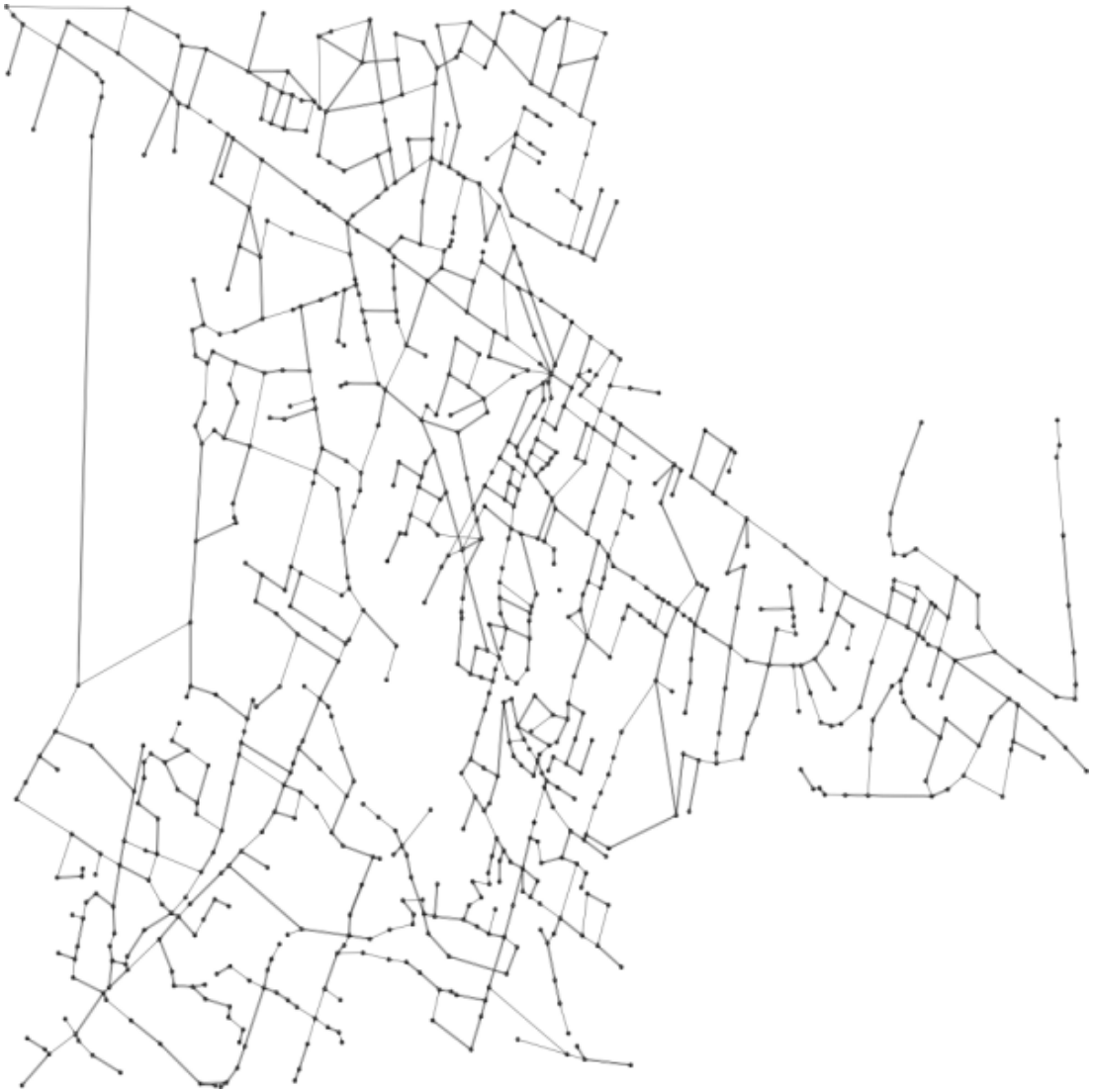
Si verifica che la soluzione sia ammissibile, altrimenti la si rende tale e la si ritorna.

Post-processing

```
function getBestSolution
    for all { solutions } do
        if (currentItem cost < others) than
            return currentItem
        end-if
    end-for
end-function
```

Tra tutte le soluzioni trovate si estrae quella con costo minore e la si ritorna.

Reggio Nell'Emilia

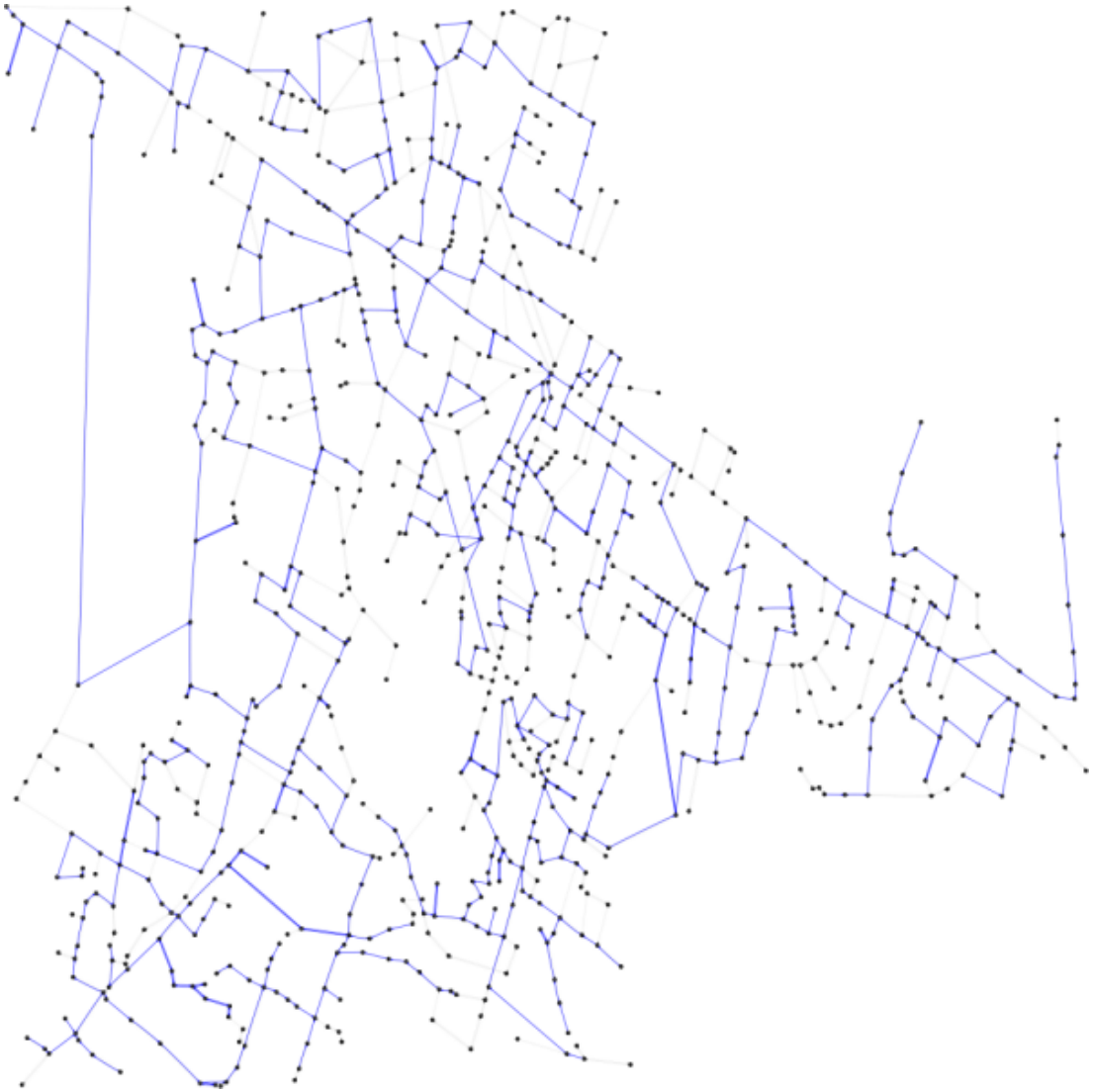


- 822 places
- 30% house (250)
- 1641 streets
- 37 buses

▸ Open svg (graph+labels)

▸ Open svg (just graph)

Reggio Nell'Emilia (solution)



- Cost 23'922 *u*
- 42 auto routes
- 4,8 average seats
- 1,5 h

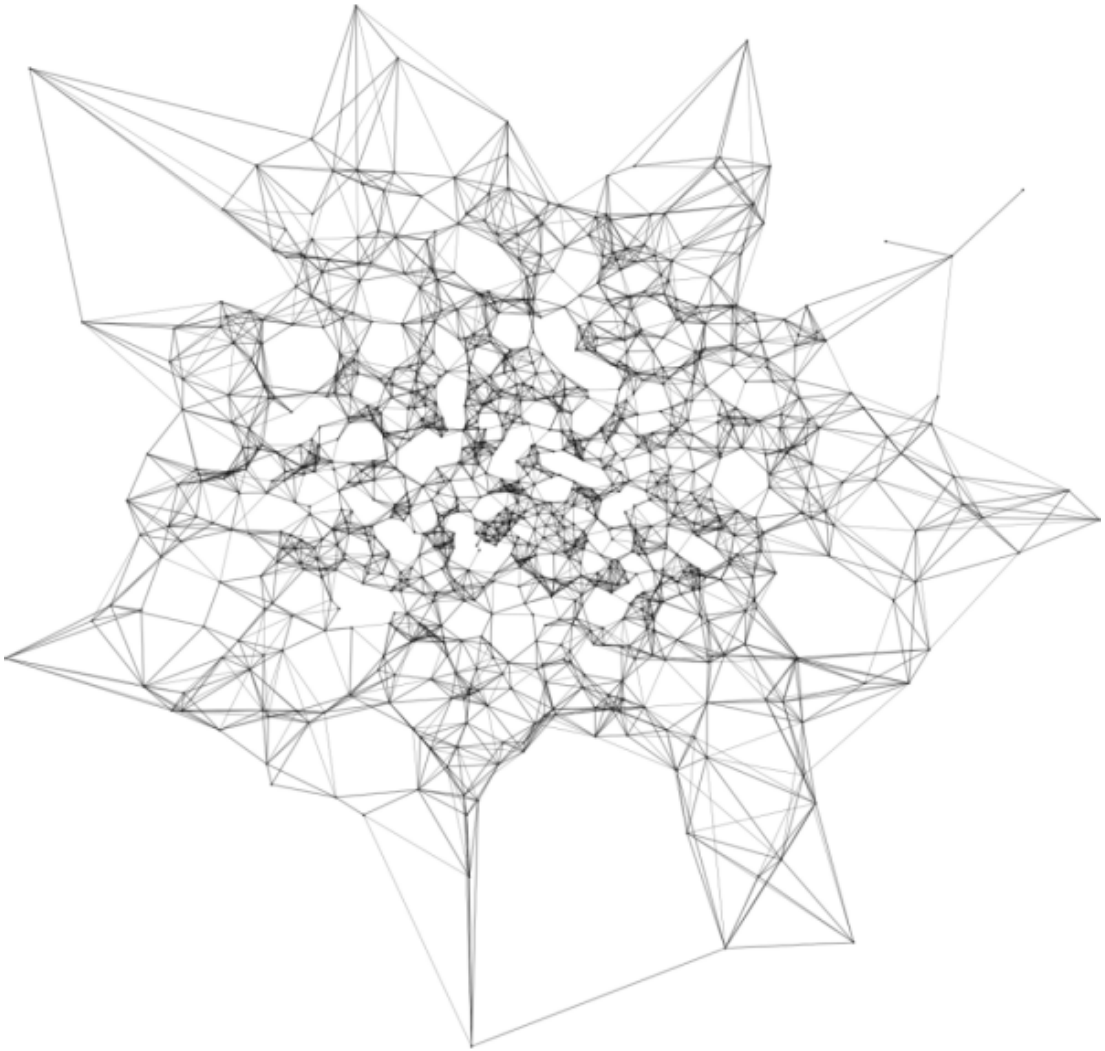
▸ Open svg (graph+labels)

▸ Open svg (just graph)

▸ Open svg (solution)

(64 threads)
Real cost 16,27 km
(1 km$\simeq$1470 u)

Sample 1

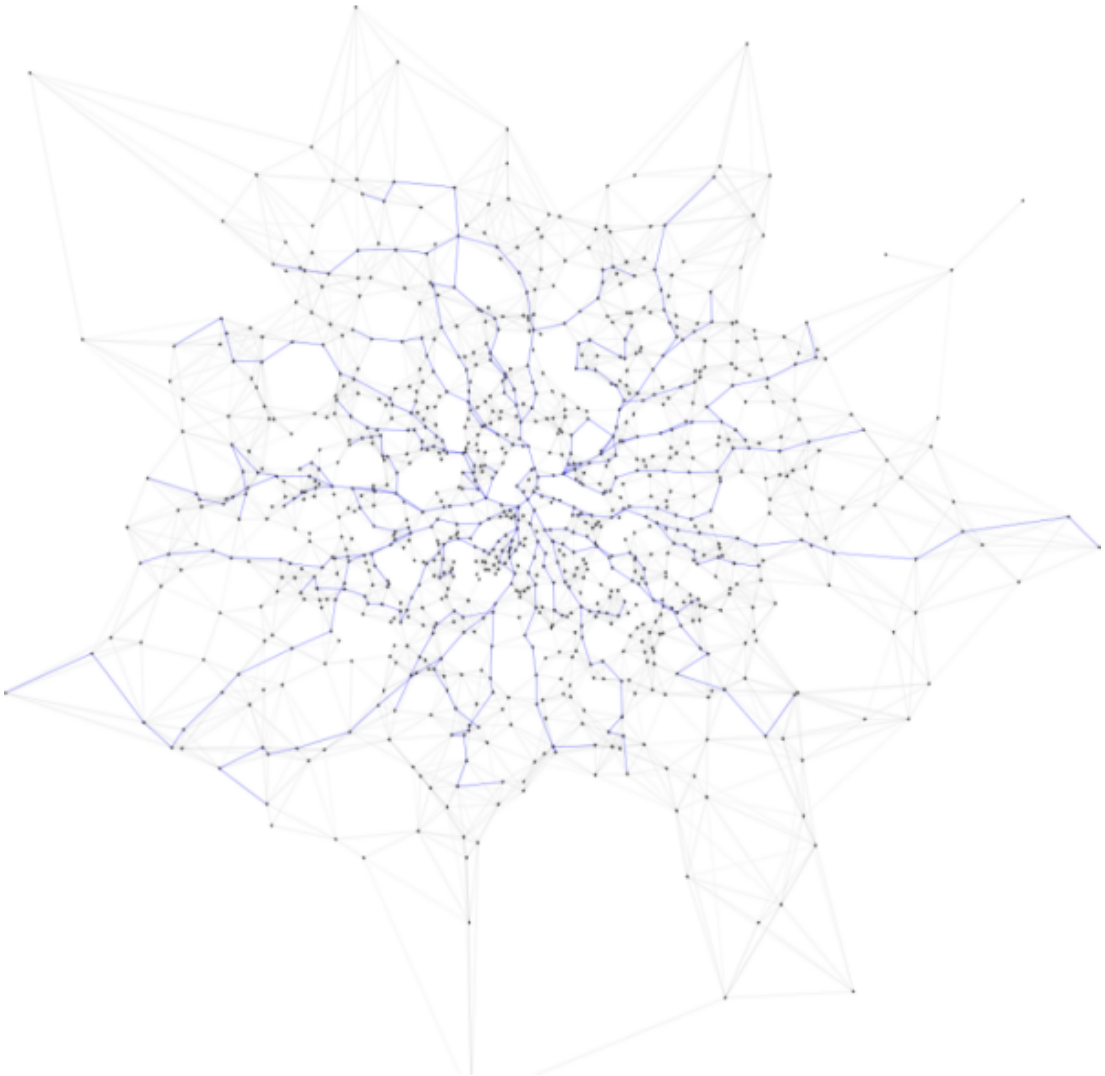


- 1'000 places
- 20% house (200)
- 10'000 streets
- 200 buses

▸ Open svg (graph+labels)

▸ Open svg (just graph)

Sample 1 (solution)



- Cost 43'090 u
- 42 auto routes
- 4,8 average seats
- 1,5 h

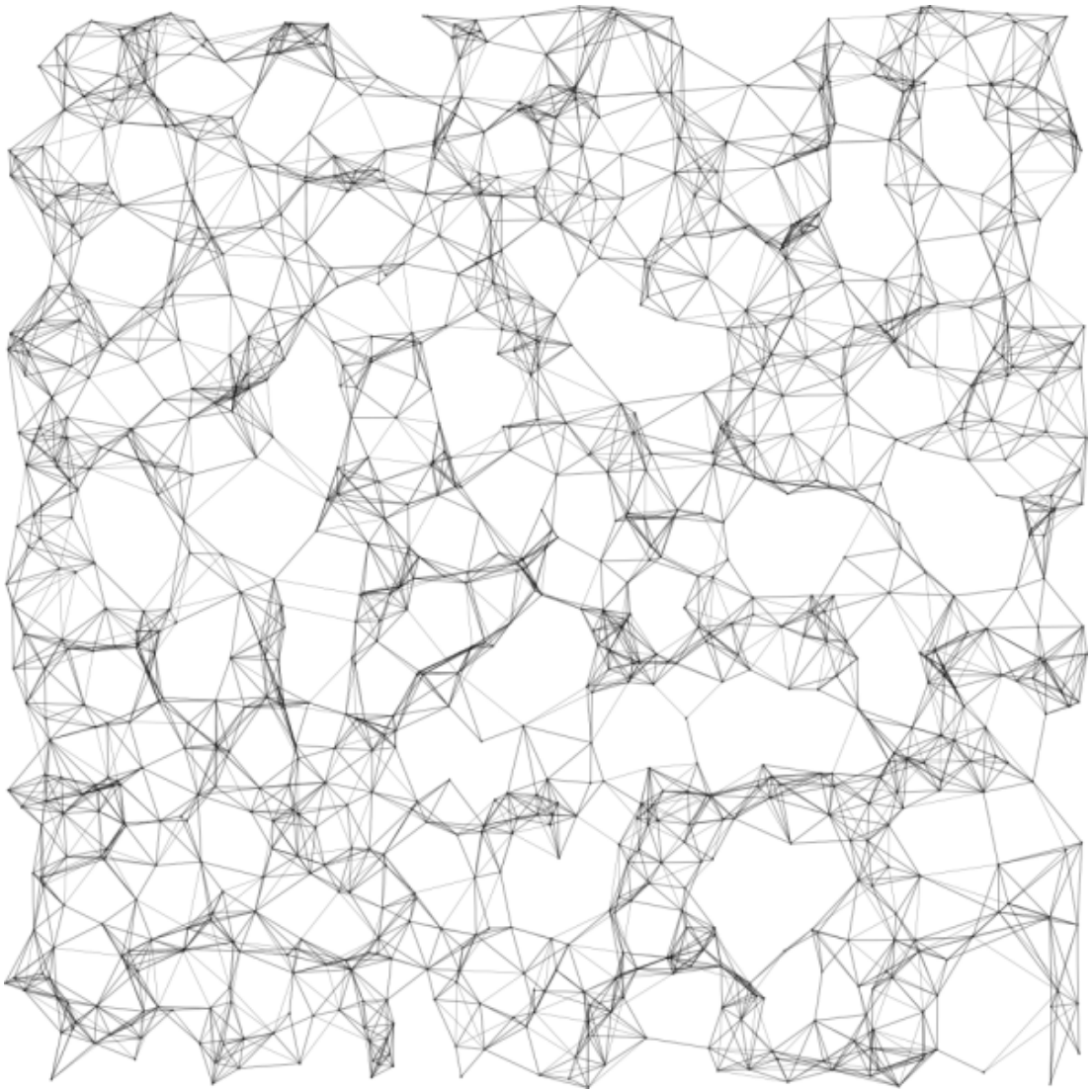
▸ Open svg (graph+labels)

▸ Open svg (just graph)

▸ Open svg (solution)

(32 threads)
Intel G630
2 core - 2.7GHz
3MB cache

Sample 2

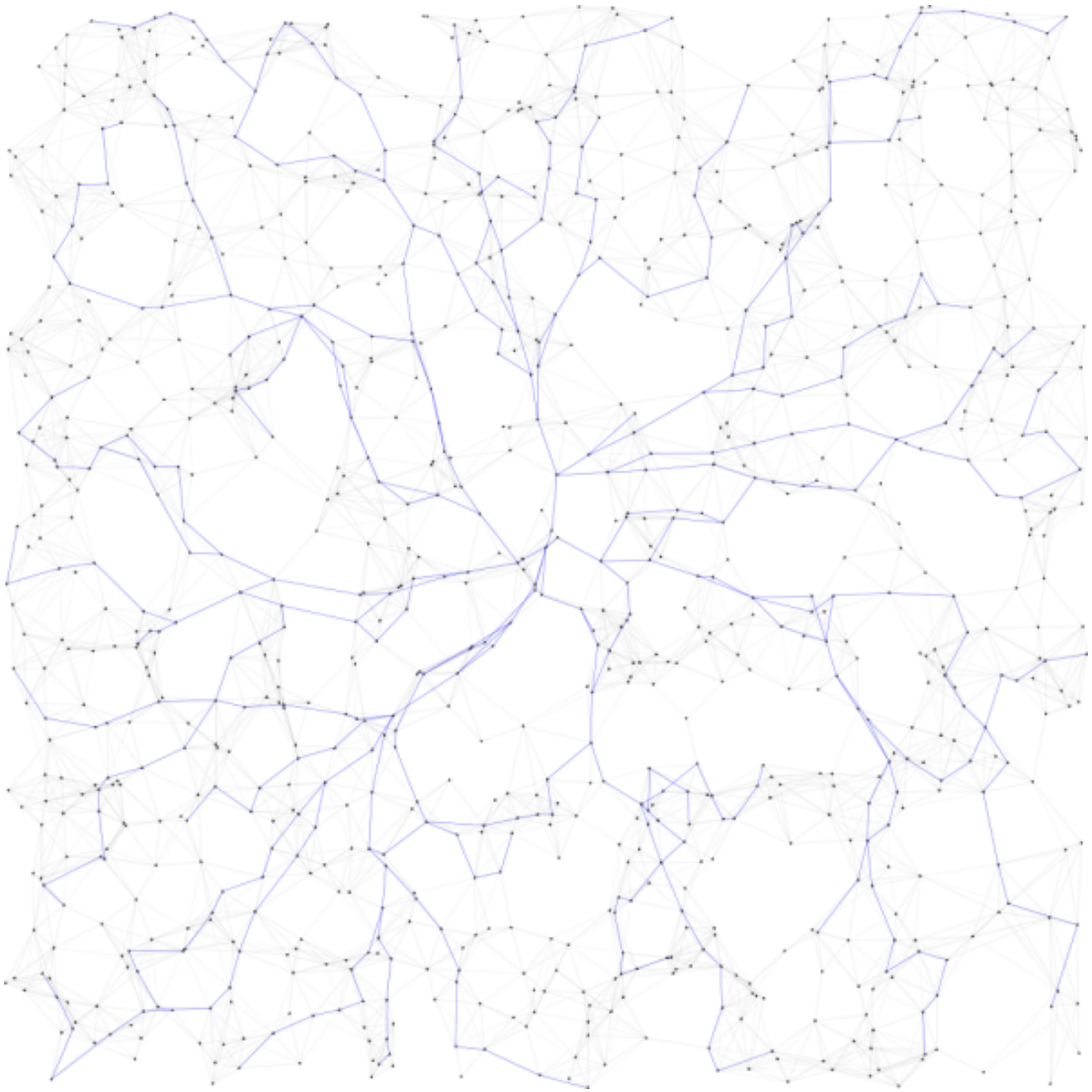


- 1'000 places
- 20% house (200)
- 10'000 streets
- 200 buses

▸ Open svg (graph+labels)

▸ Open svg (just graph)

Sample 2 (solution)



- Cost 23'922 *u*
- 42 auto routes
- 4,8 average seats
- 1,5 h

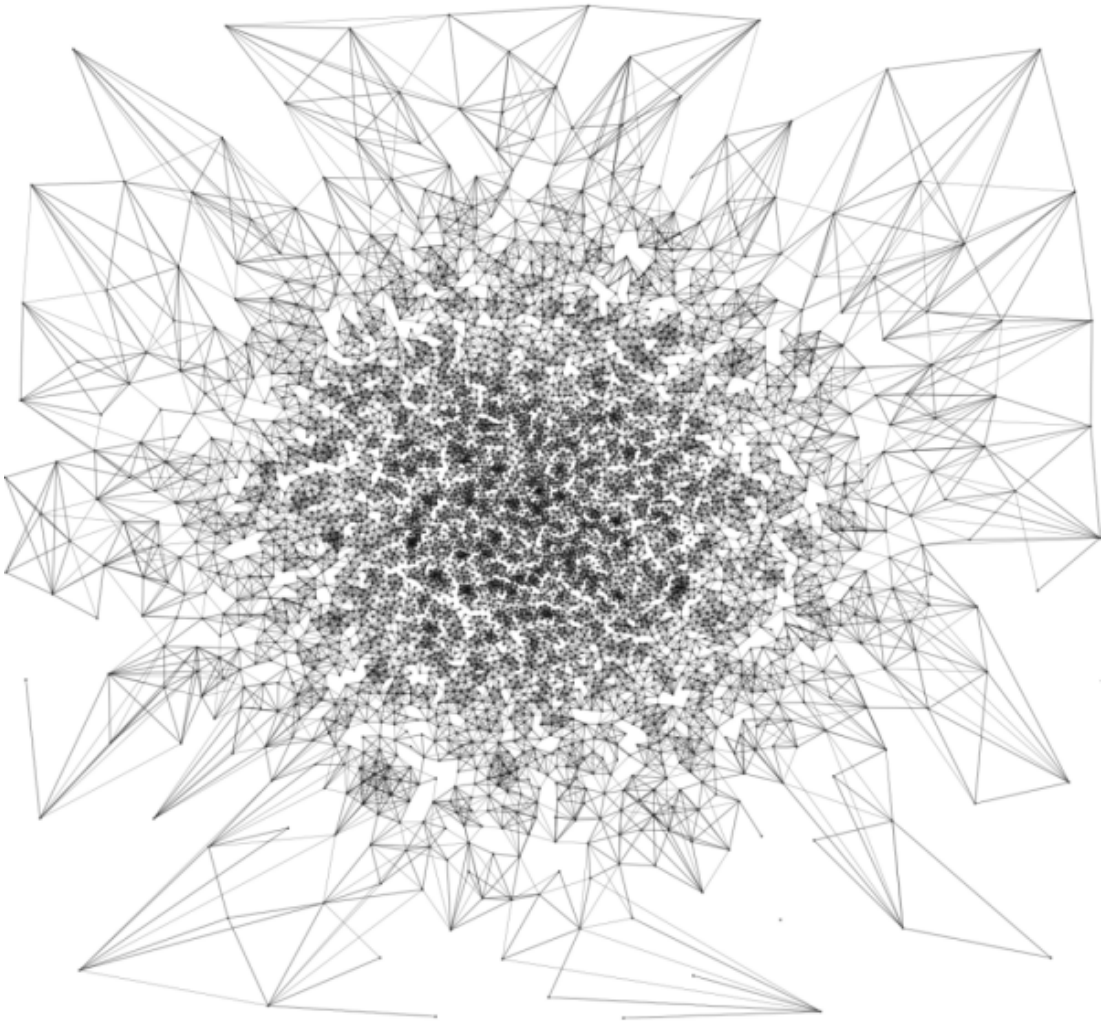
▸ Open svg (graph+labels)

▸ Open svg (just graph)

▸ Open svg (solution)

(32 threads)
Intel G630
2 core - 2.7GHz
3MB cache

Sample 3

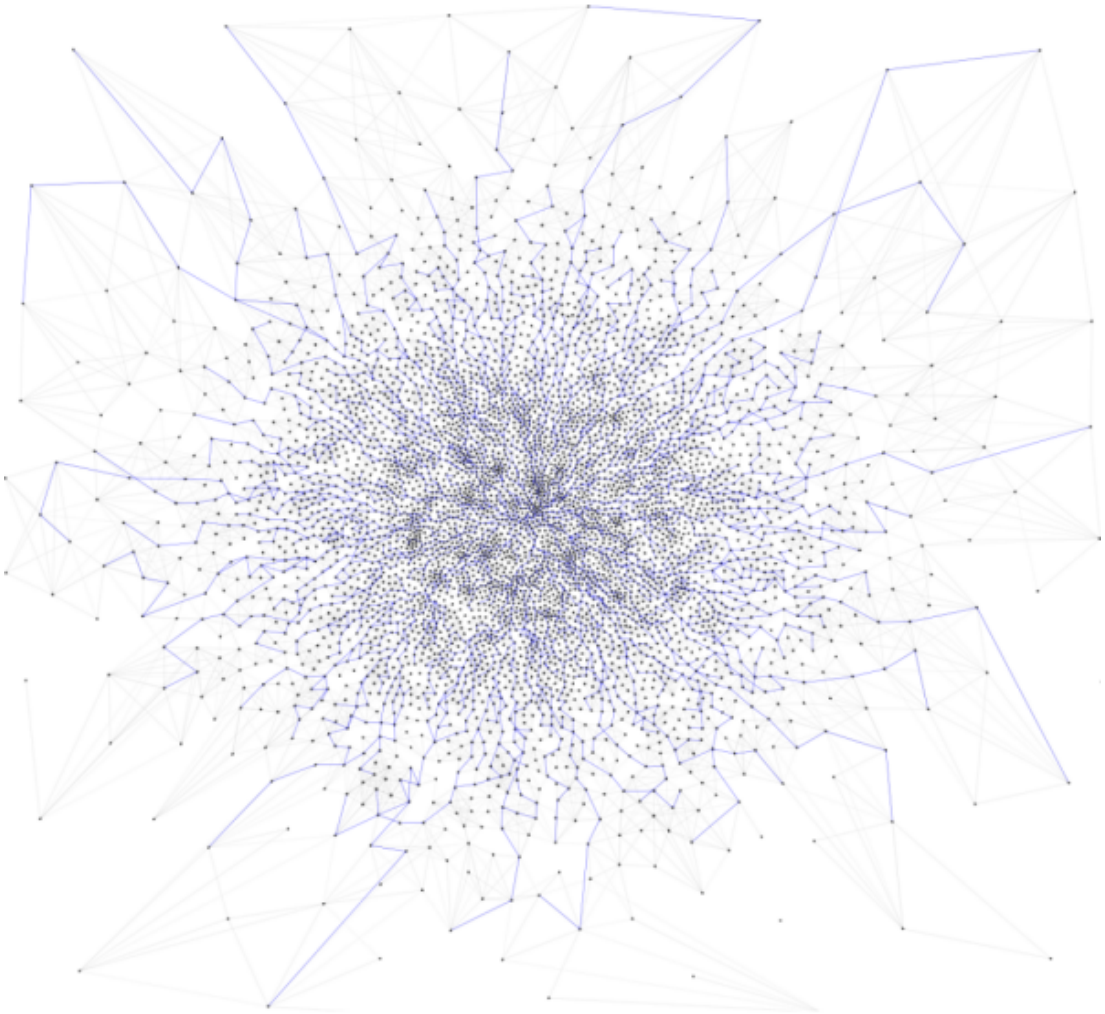


- 5'000 places
- 40% house (2'000)
- 50'000 streets
- 500 buses

▸ Open svg (graph+labels)

▸ Open svg (just graph)

Sample 3 (solution)



- Cost 1'718'107 *u*
- 408 auto routes
- 4,9 average seats
- 8 h

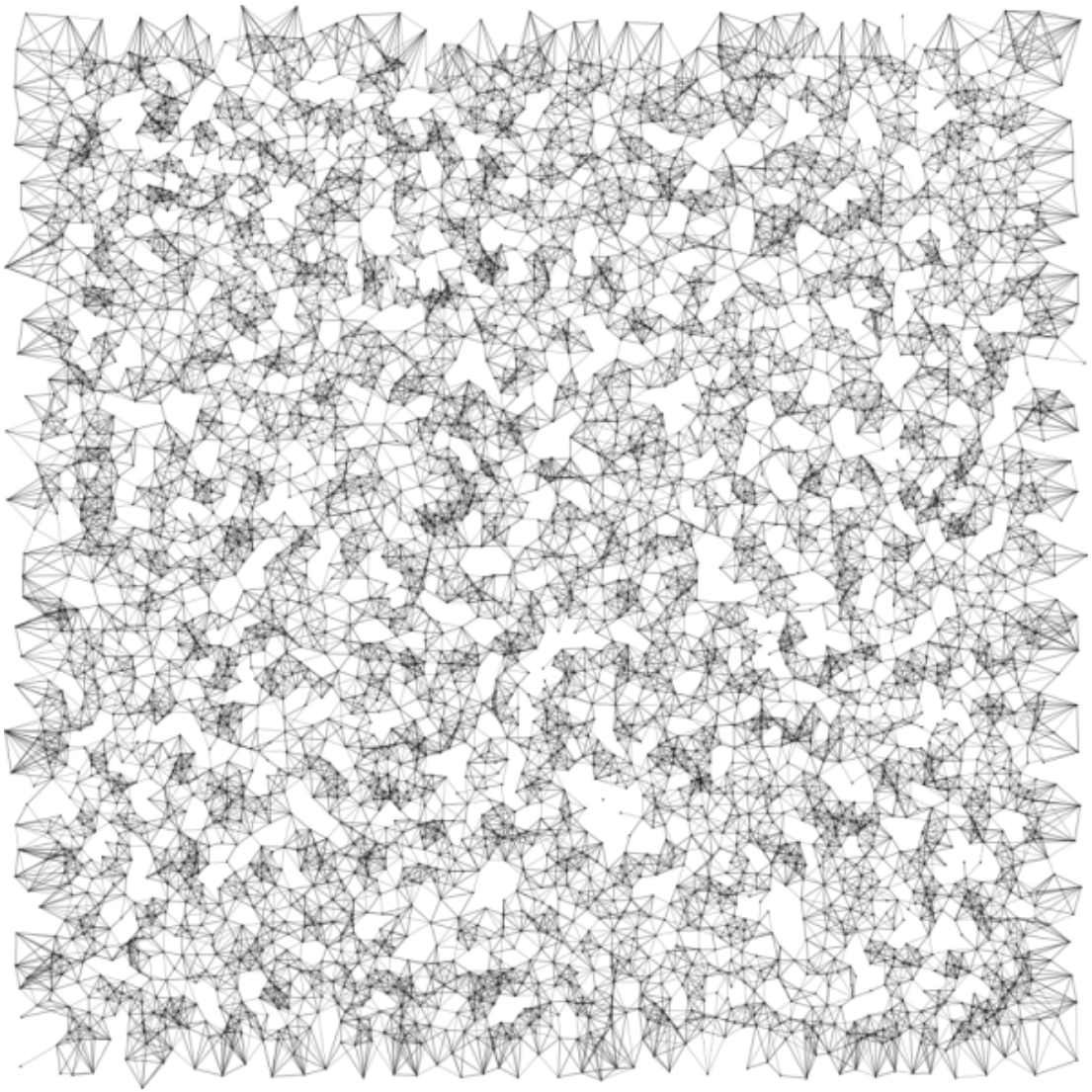
▸ Open svg (graph+labels)

▸ Open svg (just graph)

▸ Open svg (solution)

(16 threads)
Intel i3-330M
4 core - 2.13GHz
3MB cache

Sample 4

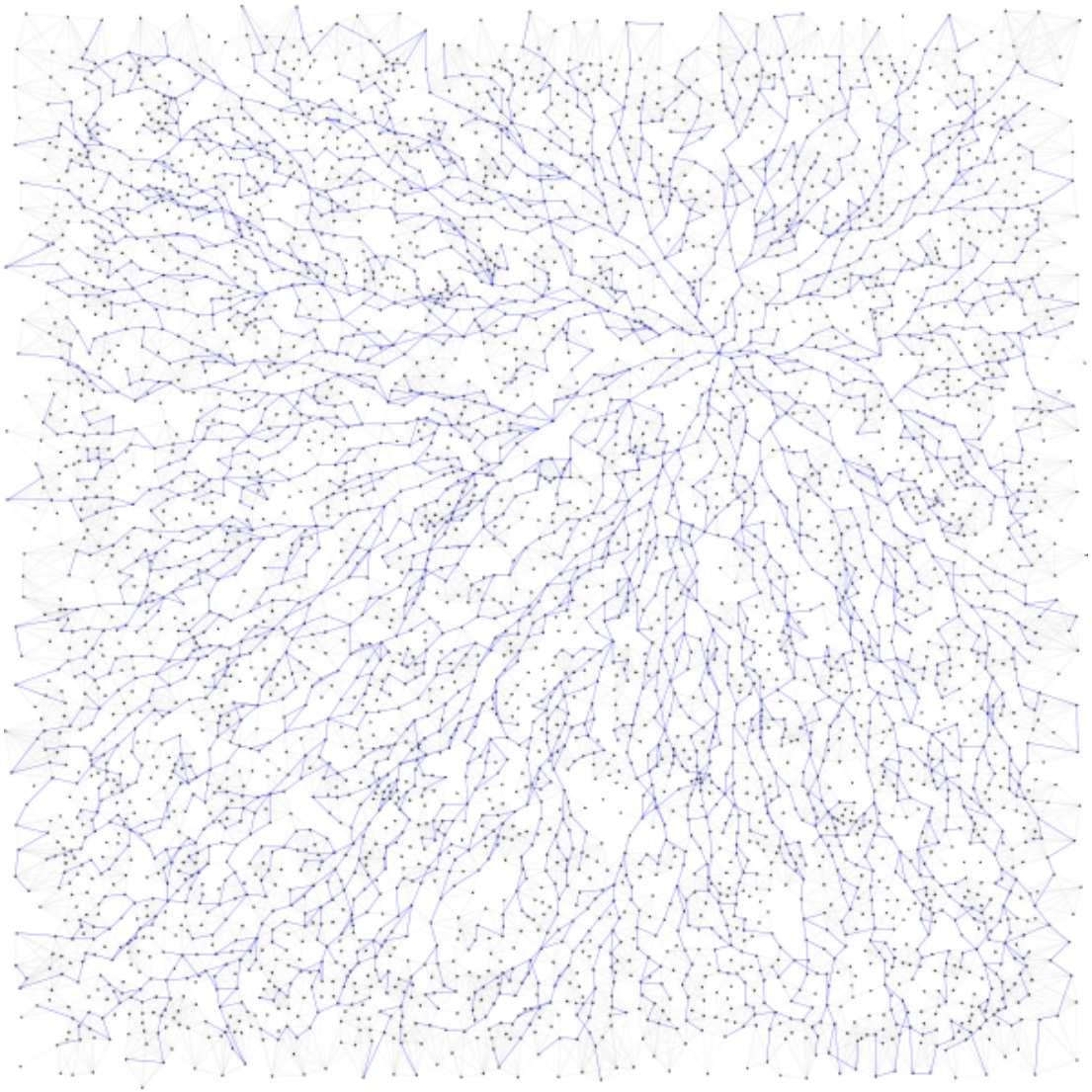


- 5'000 places
- 40% house (2'000)
- 50'000 streets
- 500 buses

▸ Open svg (graph+labels)

▸ Open svg (just graph)

Sample 4 (solution)



- Cost 1'130'839 *u*
- 402auto routes
- 4,98 average seats
- 2 d 19 h 15 min

▸ Open svg (graph+labels)

▸ Open svg (just graph)

▸ Open svg (solution)

(16 threads)
Intel i3-330M
4 core - 2.13GHz
3MB cache

Sample 5 & 6

Sample 5

- 10'000 places
- uniform spread
- 25% house (2'500)
- 50'000 streets
- 5% one-way streets (2'500)

? (8 threads)
? Intel i7-5960X
? 8 core - 3,5GHz
? 20MB cache
? 1'250 $

Sample 6

- 10'000 places
- gaussian spread
- 25% house (2'500)
- 50'000 streets
- 5% one-way streets (2'500)

? (8 threads)
? Intel i7-5960X
? 8 core - 3,5GHz
? 20MB cache
? 1'250 $

Osservazioni

- Somiglianza distribuzione gaussiana $\sim$ città a ragnatela

Osservazioni

- ⋆ Somiglianza distribuzione gaussiana $\sim$ città a ragnatela
- ⋆ Somiglianza distribuzione uniforme $\sim$ reti scale - free

Osservazioni

- ⋆ Somiglianza distribuzione gaussiana $\sim$ città a ragnatela
- ⋆ Somiglianza distribuzione uniforme $\sim$ reti scale - free
- ⋆ Probabile dipendenza del tempo di calcolo dalla topologia del grafo

Osservazioni

- ⋆ Somiglianza distribuzione gaussiana $\sim$ città a ragnatela
- ⋆ Somiglianza distribuzione uniforme $\sim$ reti scale - free
- ⋆ Probabile dipendenza del tempo di calcolo dalla topologia del grafo
- × Antieconomicità dei minibus (auto : bus ~~1 : 3~~)

Osservazioni

- ⋆ Somiglianza distribuzione gaussiana $\sim$ città a ragnatela
- ⋆ Somiglianza distribuzione uniforme $\sim$ reti scale - free
- ⋆ Probabile dipendenza del tempo di calcolo dalla topologia del grafo
- × Antieconomicità dei minibus (auto : bus ~~1 : 3~~)
- ✓ Prestazioni ottime per $<$ 1'000 luoghi

Osservazioni

- ⋆ Somiglianza distribuzione gaussiana $\sim$ città a ragnatela
- ⋆ Somiglianza distribuzione uniforme $\sim$ reti scale - free
- ⋆ Probabile dipendenza del tempo di calcolo dalla topologia del grafo
- × Antieconomicità dei minibus (auto : bus ~~1 : 3~~)
- ✓ Prestazioni ottime per $<$ 1'000 luoghi
- ✓ Prestazioni discrete per $\sim$ 2'000 case (distribuzione gaussiana)

Osservazioni

- ⋆ Somiglianza distribuzione gaussiana $\sim$ città a ragnatela
- ⋆ Somiglianza distribuzione uniforme $\sim$ reti scale - free
- ⋆ Probabile dipendenza del tempo di calcolo dalla topologia del grafo
- × Antieconomicità dei minibus (auto : bus ~~1 : 3~~)
- ✓ Prestazioni ottime per $<$ 1'000 luoghi
- ✓ Prestazioni discrete per $\sim$ 2'000 case (distribuzione gaussiana)
- ▽ Ottimizzazioni future possibili:

Osservazioni

- ⋆ Somiglianza distribuzione gaussiana $\sim$ città a ragnatela
- ⋆ Somiglianza distribuzione uniforme $\sim$ reti scale - free
- ⋆ Probabile dipendenza del tempo di calcolo dalla topologia del grafo
- × Antieconomicità dei minibus (auto : bus ~~1 : 3~~)
- ✓ Prestazioni ottime per $<$ 1'000 luoghi
- ✓ Prestazioni discrete per $\sim$ 2'000 case (distribuzione gaussiana)
- ▽ Ottimizzazioni future possibili:
  - ✓ Calcolo dei cammini (in parallelo)

Osservazioni

- ⋆ Somiglianza distribuzione gaussiana $\sim$ città a ragnatela
- ⋆ Somiglianza distribuzione uniforme $\sim$ reti scale - free
- ⋆ Probabile dipendenza del tempo di calcolo dalla topologia del grafo
- × Antieconomicità dei minibus (auto : bus ~~1 : 3~~)
- ✓ Prestazioni ottime per $<$ 1'000 luoghi
- ✓ Prestazioni discrete per $\sim$ 2'000 case (distribuzione gaussiana)
- ▽ Ottimizzazioni future possibili:
  - ✓ Calcolo dei cammini (in parallelo)
  - ✓ Calcolo dei savings (in parallelo)

Osservazioni

- ⋆ Somiglianza distribuzione gaussiana $\sim$ città a ragnatela
- ⋆ Somiglianza distribuzione uniforme $\sim$ reti scale - free
- ⋆ Probabile dipendenza del tempo di calcolo dalla topologia del grafo
- × Antieconomicità dei minibus (auto : bus ~~1 : 3~~)
- ✓ Prestazioni ottime per $<$ 1'000 luoghi
- ✓ Prestazioni discrete per $\sim$ 2'000 case (distribuzione gaussiana)
- ▽ Ottimizzazioni future possibili:
  - ✓ Calcolo dei cammini (in parallelo)
  - ✓ Calcolo dei savings (in parallelo)
  - ✓ Calcolo delle routes perfette (in parallelo)

Osservazioni

- ⋆ Somiglianza distribuzione gaussiana $\sim$ città a ragnatela
- ⋆ Somiglianza distribuzione uniforme $\sim$ reti scale - free
- ⋆ Probabile dipendenza del tempo di calcolo dalla topologia del grafo
- × Antieconomicità dei minibus (auto : bus ~~1 : 3~~)
- ✓ Prestazioni ottime per $<$ 1'000 luoghi
- ✓ Prestazioni discrete per $\sim$ 2'000 case (distribuzione gaussiana)
- ▽ Ottimizzazioni future possibili:
  - ✓ Calcolo dei cammini (in parallelo)
  - ✓ Calcolo dei savings (in parallelo)
  - ✓ Calcolo delle routes perfette (in parallelo)
  - ✓ Salvataggio su file delle routes perfette già calcolate ...

Osservazioni

- ⋆ Somiglianza distribuzione gaussiana $\sim$ città a ragnatela
- ⋆ Somiglianza distribuzione uniforme $\sim$ reti scale - free
- ⋆ Probabile dipendenza del tempo di calcolo dalla topologia del grafo
- × Antieconomicità dei minibus (auto : bus ~~1 : 3~~)
- ✓ Prestazioni ottime per $<$ 1'000 luoghi
- ✓ Prestazioni discrete per $\sim$ 2'000 case (distribuzione gaussiana)
- ▽ Ottimizzazioni future possibili:
  - ✓ Calcolo dei cammini (in parallelo)
  - ✓ Calcolo dei savings (in parallelo)
  - ✓ Calcolo delle routes perfette (in parallelo)
  - ✓ Salvataggio su file delle routes perfette già calcolate ...
  - ✓ ... e ricerca binaria (o tramite hashing) delle stesse

Osservazioni

- ⋆ Somiglianza distribuzione gaussiana $\sim$ città a ragnatela
- ⋆ Somiglianza distribuzione uniforme $\sim$ reti scale - free
- ⋆ Probabile dipendenza del tempo di calcolo dalla topologia del grafo
- × Antieconomicità dei minibus (auto : bus ~~1 : 3~~)
- ✓ Prestazioni ottime per $<$ 1'000 luoghi
- ✓ Prestazioni discrete per $\sim$ 2'000 case (distribuzione gaussiana)
- ▽ Ottimizzazioni future possibili:
  - ✓ Calcolo dei cammini (in parallelo)
  - ✓ Calcolo dei savings (in parallelo)
  - ✓ Calcolo delle routes perfette (in parallelo)
  - ✓ Salvataggio su file delle routes perfette già calcolate ...
  - ✓ ... e ricerca binaria (o tramite hashing) delle stesse
  - ✓ Tabu Search per uscire dai minimi locali

Domande ?