The courtesy bus problem

Progetto del corso di Algoritmi di Ottimizzazione

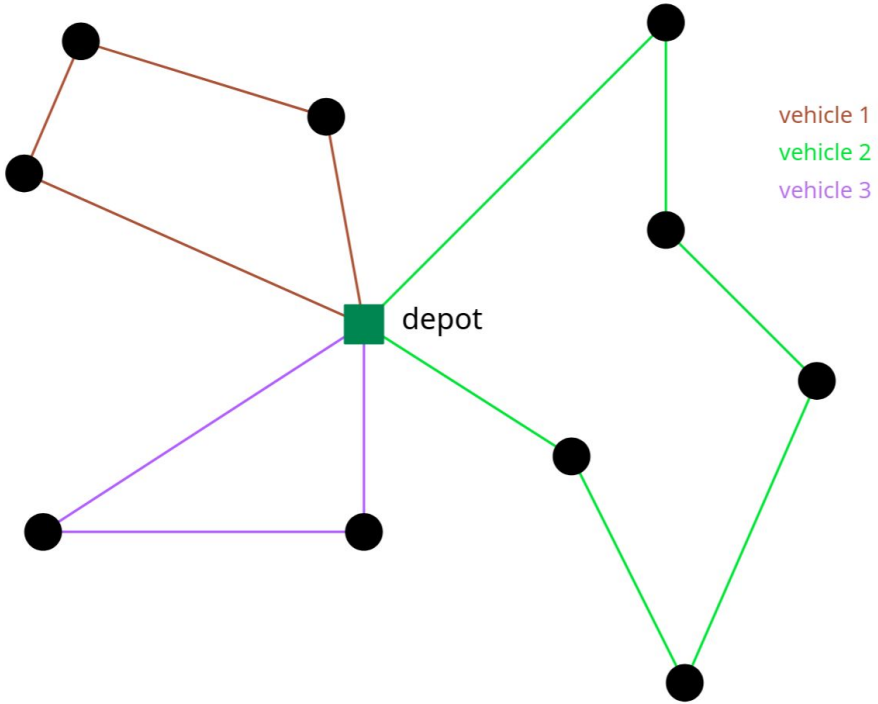
A.A. 2021/2022

Giulia Calanca

Il problema in letteratura

Il **Vehicle Routing Problem** e la sua variante con **Time Windows** sono classici problemi di ottimizzazione combinatoria e programmazione intera.

- ★ Introdotto da Dantzig e Ramser (1959)
- ★ Generalizzazione del problema del commesso viaggiatore (TSP)
- ★ **Obiettivo**: trovare l'insieme di percorsi con costo minimo, che un gruppo di veicoli deve percorrere affinché tutti i clienti vengano serviti.
- ★ Problema NP-hard



Varianti del problema

Esistono numerose varianti del VRP

- **VRPTW**: *Vehicle Routing problem con Time Windows*
- **CVRP**: *Capacitated Vehicle Routing Problem*
- **VRPMT**: *Vehicle Routing Problem with Multiple Trips*
- **MDVRP**: *Multi-Depot Vehicle Routing Problem*

Il problema in questione è una variante del **CVRPTW** (*Capacitated Vehicle Routing Problem with Time Windows*) nella quale i veicoli hanno una capacità limitata e sono presenti vincoli di tempo per la consegna.

Modellazione del problema (1)

Modello compatto a tre indici

$$\min \sum_{i,j \in V} c_{ij} \sum_{k \in K} x_{ijk}$$

sub. to

$$\sum_{i \in V} \sum_{k \in K} x_{ijk} = 1 \ \forall j \in V \setminus 0 \quad (1)$$

$$\sum_{j \in V} \sum_{k \in K} x_{ijk} = 1 \ \forall j \in V \setminus 0 \quad (2)$$

$$\sum_{i \in V} \sum_{k \in K} x_{ihk} - \sum_{j \in V} \sum_{k \in K} x_{hjk} = 0 \ \forall k \in K, h \in V \quad (3)$$

$$\sum_{i \in V} q_i \sum_{j \in V} x_{ijk} \leq Q_k \ \forall k \in K \quad (4)$$

$$\sum_{ijk} x_{ijk} = |S| - 1 \ \forall S \subseteq P(V), 0 \notin S \quad (5)$$

$$x_{ijk} \in 0, 1 \quad (6)$$

Extra vincoli temporali

$$a_i \leq s_{ik} \leq b_i \quad (7)$$

$$s_{ik} + t_{ij} - M(1 - x_{ijk}) \leq s_{jk} \quad (8)$$

La modellazione del CVRPTW prevede gli stessi vincoli del CVRP con l'aggiunta dei vincoli sulle time windows

Inoltre sono rimossi i vincoli che regolano i subtour

I vincoli temporali di fatto eliminano la formazione di subtour

Modellazione del problema (2)

Modello esteso (column generation)

- Sia *K* l'insieme di tutte le colonne (i.e. i cammini accettabili)
- $a_{ik} = 1$ il percorso definito dalla colonna *k* serve il cliente *i* e c_k il costo di quel percorso
- Il problema diventa trovare i cammini che, attraversando tutti i nodi, minimizzano il costo totale della route (**set partitioning**)
- Le variabili sono in numero esponenziale!
- Nel sottoproblema si richiede di risolvere il problema noto come ESPPTCC (*elementary shortest path with time windows and capacity constraints*)

$$min \sum_{k \in K} c_k x_k$$

$$\text{sub. to}$$

$$\sum_{k \in K} a_{ik} x_k = 1 \ \forall i$$

$$x_k \in 0, 1$$

Metodi di risoluzione

Metodi esatti

Gli approcci esatti al problema si possono suddividere in tre filoni principali:

1. Basati sul rilassamento Lagrangiano
2. Column Generation
3. Dynamic programming

Il vehicle routing problem è **NP-hard** in tutte le sue varianti, perciò la dimensione dei problemi risolvibili in modo esatto, ovvero trovando l'ottimo, è molto limitata.

Metodi euristici e metaeuristici

- Algoritmi costruttivi
- **Local Search**: soluzione ammissibile in un minimo locale
- Concetto di **neighbourhood**: l'insieme delle soluzioni raggiungibili tramite l'utilizzo di **mosse**
- Mosse comuni: *2-opt/ 3-opt/ OR-opt, insert, shift, swap, k-swap*, etc.
- **Local Search Multistart**
- Metaeuristiche: *tabu search*, *algoritmi genetici*, *simulated annealing*, ALNS…

The Courtesy Bus Problem

- In un pub si hanno alcuni *courtesy bus*, veicoli che effettuano un servizio taxi per i clienti
- Essi devono riportare a casa i clienti dopo la serata trascorsa al pub a partire dall'orario da loro richiesto.
- Ogni cliente fornisce un lower bound dell'orario desiderato di arrivo a casa
- La soddisfazione del cliente dipenderà da quanto l'effettiva ora d'arrivo differirà da quella richiesta.

Obiettivi e Vincoli

Gli obiettivi sono essenzialmente due:

1. Trovare l'insieme di route che minimizzino il costo di percorrenza dei vari courtesy bus
2. Massimizzare la soddisfazione dei clienti portandoli a casa il prima possibile considerando l'orario richiesto

I vincoli da rispettare sono:

1. Non eccedere la capacità dei bus
2. Portare a casa tutti i clienti
3. Rispettare l'orario di arrivo desiderato dai clienti
4. Far partire e ritornare ogni bus al pub

Formalizzazione del problema

Sia:

- K l'insieme dei bus di capacità Q
- C l'insieme dei clienti del pub
- a_i l'orario di arrivo desiderato, richiesto dal cliente i
- $[a_i +\infty]$ la time windows in cui portare a casa il cliente associato al nodo i
- $G = (V,A)$ un grafo orientato con $V = \{0\} \cup C$ e A insieme degli archi
- t_{ij} e c_{ij} il tempo ed il costo di attraversamento dell'arco $(i,j) \in A$

Variabili

Oltre alle variabili t_{ij} e c_{ij} definiamo una variabile a tre indici x_{ijk} per definire quali bus percorrono quali archi.

$$x_{i,j,k} = \begin{cases} 1 & \text{if bus } k \text{ travels from } i \text{ to } j \text{ directly} \\ 0 & \text{otherwise} \end{cases}$$

Per implementare le time windows c'è bisogno di un modo per determinare quando un cliente viene riportato a casa. Introduciamo quindi due variabili temporali:

- z_i rappresenta l'istante in cui il cliente *i* arriva a casa
- y_{ik} che rappresenta l'istante nel quale il bus *k* arriva a casa del cliente i

Aggiungiamo anche la variabile w_{ik} che determina se il bus *k* porta a casa il cliente *i*

$$w_{i,k} = \begin{cases} 1 & \text{if bus } k \text{ takes customer } i \text{ home} \\ 0 & \text{otherwise} \end{cases}$$

Funzione obiettivo

Considerando di dover minimizzare sia i costi delle route, sia la soddisfazione dei clienti, la f.o. diventa:

$$\min \alpha \sum_{k \in K} \sum_{(i,j) \in A} c_{i,j} x_{i,j,k} + \beta \sum_{i \in C} z_i - a_i$$

Con α e β parametri per stabilire a quale delle componenti dare più importanza.

Vincoli (1)

1. Non eccedere la capacità dei bus

$$\sum_{(i,j)\in A(-,-,k)} x_{i,j,k} \leq Q;\ k \in K$$

2. Portare i clienti a casa una volta sola e con un solo bus

$$\sum_{k\in K} \sum_{i\in A(-,j,k)} x_{i,j,k} = 1;\ j \in C$$

3. Bilanciamento del flusso

$$\sum_{i\in A(-,h,k)} x_{i,h,k} - \sum_{j\in A(h,-,k)} x_{h,j,k} = 0;\ h \in C, k \in K$$

4. Vincoli bus trip

$$\sum_{j\in A(0,-,k)} x_{0,j,k} <= 1;\ k \in K$$

$$\sum_{i\in V} x_{i,0,k} <= 1;\ k \in K$$

5. Vincoli di tempo

$$z_i \geq a_i, \text{ for } i \in C$$

6. Consecutività di arrivo di un bus

$$y_{j,k} \geq y_{i,k} + t_{i,j}x_{i,j,k} - M(1 - x_{i,j,k})$$

$$y_{j,k} \leq y_{i,k} + t_{i,j}x_{i,j,k} + M(1 - x_{i,j,k})$$

Vincoli (2)

7. Valorizzazione z_i

$$z_i = \sum_{k \in K} y'_{i,k};\ i \in I$$

8. Valorizzazione w_{ik}

$$w_{i,k} = \sum_{j \in A(i,-,k)} x_{i,j,k};\ i \in C,\ k \in K$$

9. *MW*

$$Mw_{i,k} = M \cdot w_{i,k};\ i \in C,\ k \in K$$

10. Valorizzazione y'_{ik}

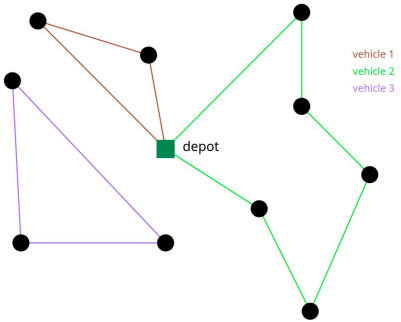
$$y'_{i,k} = min(Mw_{i,k},\ y_{i,k})$$

Note

- *M* , detta *big-M*, è una costante dal valore tendente a +∞
- *A* è lista di tutte le possibili combinazioni di nodi e bus. È formata da 3 elementi *(*$nodo_i$*,* $nodo_j$ *, bus)*. La notazione *A(−, j, k)* indica gli archi con nodo *i* fisso, nodo *j* non fisso e bus *k* non fissi.

Eliminazione subtour

- Un subtour *S* è una partizione chiusa del grafo con più di 2 nodi
- I vincoli di eliminazione dei subtour sono in numero esponenziale nel numero dei nodi



- Approccio **lazy**: aggiungere un vincolo solamente se nella soluzione è presente un subtour
- Gurobi **callback**

```
def callback(model, where):
    def detect_subtours(passages):
        # detect subtours here ...
        return bad_trips

    if where == gurobipy.GRB.Callback.MIPSOL:
        # Get the X of the current solution
        current_X = model.cbGetSolution(X)

        # Build the passages
        passages = []
        for (a, val) in current_X.items():
            if val > EPSILON:
                passages.append(a)

        bad_trips = detect_subtours(passages)

        if bad_trips:
            # add subtour elimination constraints
            for bad_trip in bad_trips:
                model.cbLazy(quicksum(X[(i, j, k)] for (i, j, k) in bad_trip)
                             <= len(bad_trip) - 1)

# solve
m.optimize(callback)
```

Euristiche: euristica costruttiva

Per generare una soluzione accettabile si sono testati tre differenti **algoritmi *greedy***

1. Il primo algoritmo riempie un bus alla volta aggiungendo ad ogni passo il cliente più vicino in termini di costo e tempo
2. Il secondo algoritmo scorre invece la lista dei bus e aggiunge per il bus selezionato un cliente alla volta scegliendo quello più vicino (in termini di costo e tempo)
3. L'ultimo algoritmo abbina in modo casuale ogni cliente ad un bus che abbia sufficiente capacità.

Gli algoritmi generano soluzioni simili a livello di costo computazionale.

L'algoritmo 3 genera una soluzione che risulta più adatta (con più possibilità di miglioramento) da dare in input alla local search.

Euristiche: local search

La local search consiste nell'esplorazione di una **neighbourhood** generata essenzialmente a partire una **mossa**:

➢ *MoveAndOptTime(node,bus,pos)*, una versione della **insert** che sposta il nodo *node* nella lista di clienti di *bus* nella posizione *pos*. *bus* può essere lo stesso di partenza o un altro.

All'interno della mossa è presente una **sotto-mossa** che ottimizza il tempo di partenza del bus *bus*. Viene calcolato per ogni nodo il tempo minimo di partenza che rispetti le time windows e tra questi viene selezionato il massimo.

Metaeuristiche: local search multistart

- Come prima metaeuristica si è implementata la possibilità di utilizzare la local search con un multistart.
- Il risolutore greedy costruisce ogni volta una soluzione diversa, che viene ottimizzata tramite local search finchè non si supera il tempo limite stabilito

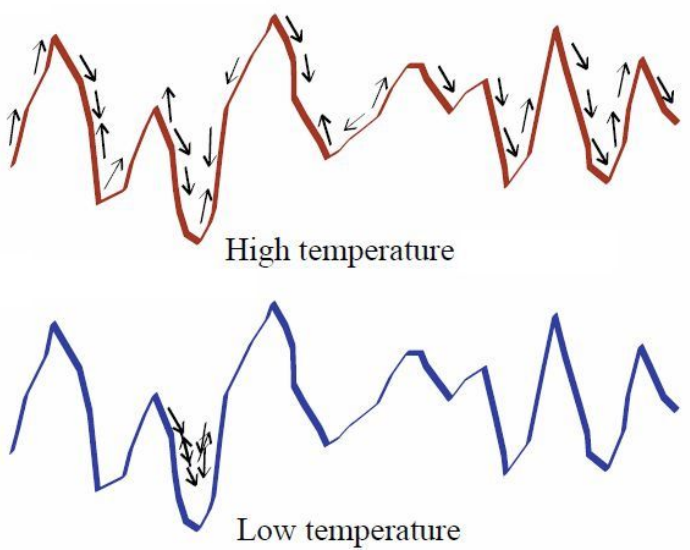
Metaeuristiche: simulated annealing (1)

- Il nome deriva da un processo utilizzato in ambito metallurgico, chiamato appunto "**annealing**" (letteralmente ricottura), usato per determinare lo stato di minima energia di un metallo
- Il materiale viene riscaldato e portato in uno stato nel quale gli atomi sono più liberi di muoversi, per poi essere raffreddato e tornare nuovamente allo stato solido
- L'idea è quella di accettare soluzioni all'interno della neighbourhood che migliorano la soluzione corrente e con una certa probabilità, anche **soluzioni peggiori**
- La probabilità è determinata da un valore decrescente di "temperatura" che permette di rendere più selettiva la scelta delle soluzioni man mano che si va avanti nella ricerca dell'ottimo.

Metaeuristiche: simulated annealing (2)

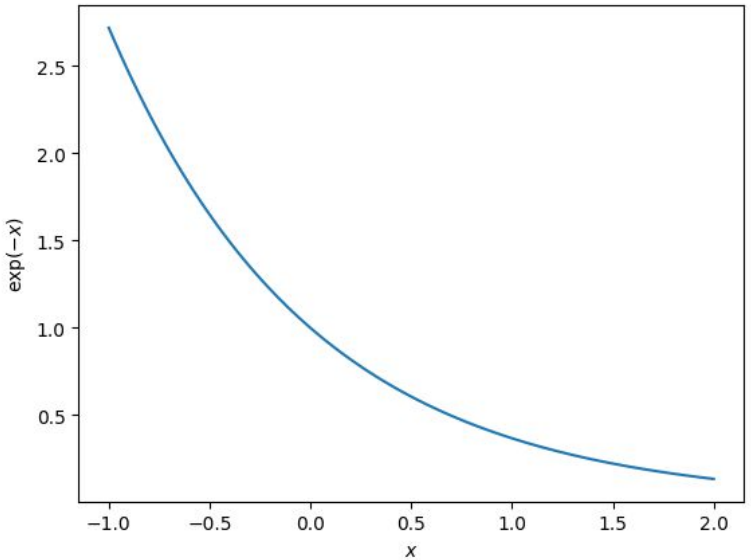
I parametri da specificare sono:

- **cooling rate**, ovvero il tasso di raffreddamento, che serve a regolare la velocità con la quale scende la temperatura. Ha un valore massimo di 1 (la temperatura rimane costante) e minimo di 0 (la temperatura cala subito al minimo)
- **initial e minimum temperature**, ovvero la temperatura di partenza e quella minima alla quale arrestare l'ottimizzazione
- **iterations per temperature**, il numero di iterazioni nelle quali è permesso applicare una mossa nella neighbourhood



Metaeuristiche: simulated annealing (3)

La probabilità di accettare soluzioni peggiori segue l'andamento di una funzione con esponente negativo

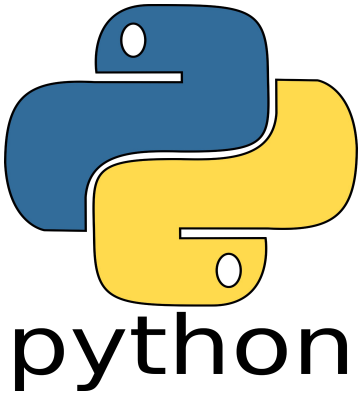



```
solution = costructive_solver.solve()

while temperature > min_temperature & (time < end_time):
  for i < n_of_iterations:
    new_solution = solution

    MoveAndOptTime(new_solution, random_src_node,
                   random_dst_bus, random_dst_pos).apply()

    if new_solution is feasible do
    delta = old_cost - new_cost
      if random() <= exp(delta/temperature)
        solution = new_solution

      if new_cost < best_cost:
        best_solution = solution
    i++
  temperature = temperature * cooling_rate

return best_solution
```

Struttura del progetto

Il progetto è scritto in python ed è strutturato nei seguenti moduli:

- model.py
- solution.py
- validator.py
- gurobisolver.py
- heuristics.py
  - GreedySolver, MoveOptTime, LocalSearch, SimulatedAnnealing



Altri moduli:

- main.py
- datasets
- results

Istanze del problema e risultati (1)

- Le istanze del problema sono state generate casualmente, aumentando man mano il numero di clienti e bus (fino ad arrivare a problemi non più risolvibili all'ottimo) e variando anche la sparsità dei nodi.
- Per ogni istanza viene descritto il numero di nodi (*# nodes*), ovvero clienti, da servire, il numero di bus (*# buses*), i secondi lasciati all'esecuzione del risolutore (*seconds to solve*) ed il valore della soluzione (solution value)
- Si riescono ad ottenere soluzioni ottime con Gurobi fino ad numero di clienti contenuto [10 nodi, 3 bus] e altrettanto buone con *local search multistart* e *simulated annealing*
- Passando a [20 nodi, 5 bus] notiamo invece che *Gurobi* non riesce più ad arrivare all'ottimo e anche *local search multi-start* non scende quanto *simulated annealing*.

Istanze del problema e risultati (2)

Tabella dei risultati (soluzioni indicate con * rappresentano soluzioni ottime)

solver	# nodes	# buses	seconds to solve	solution value
gurobi	3	2	10	47.2431*
ls	3	2	10	47.2431
ls-ms	3	2	10	47.2431
sa	3	2	10	47.2431
gurobi	4	2	10	56.2986*
ls	4	2	10	63.2672
ls-ms	4	2	10	56.2986
sa	4	2	10	56.2986
gurobi	7	3	10	116.6939*
ls	7	3	10	118.4272
ls-ms	7	3	10	116.6939
sa	7	3	10	116.6939
gurobi	10	3	10	202.4896*
ls	10	3	10	261.5168
ls-ms	10	3	10	202.4896
sa	10	3	10	202.4896
gurobi	11	10	100	213.2659*
ls	11	10	100	223.1647
ls-ms	11	10	100	213.2659
sa	11	10	100	213.2659
gurobi	20	5	100	3238.1498
ls	20	5	100	3379.7927
ls-ms	20	5	100	2998.8899
sa	20	5	100	2998.8899
gurobi	42	8	200	772.44067
ls	42	8	200	715.7607
ls-ms	42	8	200	660.1733
sa	42	8	200	588.4076
gurobi	52	11	200	4610.3419
ls	52	11	200	3131.1859
ls-ms	52	11	200	2984.3199
sa	52	11	200	2889.8812
gurobi	201	13	200	-
ls	201	13	200	59069.6611
ls-ms	201	13	200	57914.4437
sa	201	13	200	4914.3869

Conclusioni

- Come previsto il risolutore ottimo implementato con Gurobi risolve problemi di grandezza ridotta (10-12 clienti)
- A parità di tempo lasciato alla risoluzione e a parità di neighbourhood l'algoritmo di simulated annealing performa meglio del local search con multistart, riuscendo ad uscire da ottimi locali più rapidamente
- Spesso le stesse soluzioni sono esplorate più volte, si potrebbe utilizzare un meccanismo tabu-list da integrare nel simulated annealing per evitare che ciò accada