UNIVERSITY OF FERRARA
- EX LABORE FRUCTUS -

DE: Department of Engineering Ferrara

Metodi di ottimizzazione

16 - Location Routing

Università degli Studi di Ferrara
Ingegneria Informatica e dell'Automazione
A.A. 2021/22

Andrea Bazerla - 151792
Alessandro Gilli - 122107
Luana Mantovan - 135151

Indice

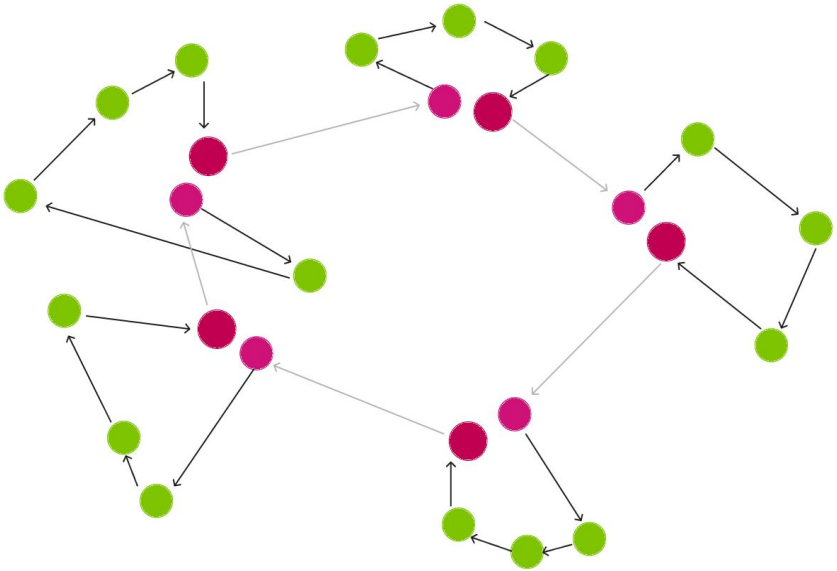
- PROGETTO
  - L’idea
  - I richiami teorici
- MODELLO MATEMATICO
  - Funzione obiettivo
  - Variabili decisionali
  - Vincoli
  - Raffinamento a coppie
  - Vincolo di CutSet
- ASPETTI TECNICI
  - Grafici
  - Debug, run parametrizzata e randomica su seed
- RISULTATI
  - Prima correzione
  - Versione definitiva
- POSSIBILI AGGIORNAMENTI

Progetto

Location Routing (16)

Si consideri un problema di location capacitato, in cui sono dati n clienti N={1..n} ciascuno con domanda q_i>0, e f luoghi possibili F={1..f} in cui costruire dei magazzini di capacità Q, occorre decidere dove costruire i magazzini noto gh il costo di costruzione del magazzino nella località h, quali clienti servire con ciascun magazzino sapendo che il costo di servizio è dato dal costo del ciclo hamiltoniano di costo minimo che visita il magazzino h e tutti i nodi associati ai clienti assegnati a quel magazzino. L'obiettivo è minimizzare la somma dei costi di costruzione e di servizio ai clienti.

Progetto - L’idea



- Creare un unico, grande ciclo, duplicando i magazzini e facendo in modo che ogni nodo abbia un solo arco di entrata e di uscita.
- La duplicazione dei magazzini consiste nel creare un magazzino di Start e uno di End per ogni posizione
- I costi degli archi tra magazzini non è rilevante.
- Il TSP si basa, oltre che sulla distanza, anche sulla domanda dei clienti che non deve superare la capacità di ogni magazzino

Progetto - Richiami teorici

S-TSP

- Si vuole minimizzare il costo totale, considerando la creazione di un ciclo hamiltoniano che tocca tutti i nodi presenti

- Il problema è NP-Hard

- La connessione si può imporre in due modi alternativi:
  - CutSet: da ogni sottoinsieme di nodi escono almeno due archi. In questo modo si obbliga ad avere un arco per ogni nodo
  - SubTour Breaking: per ogni sottoinsieme di n nodi, si impedisce la presenza di cicli formati da meno di n nodi

- Si può dimostrare che le due formulazioni sono equivalenti; abbiamo scelto il CutSet

Progetto - Richiami teorici

Facility Location versione capacitata

- Si vuole trovare la posizione migliore per allocare i magazzini, minimizzando il costo di costruzione dei magazzini e tale che il costo di trasporto dai magazzini ai clienti sia il più basso possibile

- Il problema è NP-Hard

Progetto - Richiami teorici

Bin Packing

- Dati dei contenitori di capacità definita, nel nostro caso i magazzini, e un insieme di oggetti con peso, qui i clienti con la domanda, si vuole minimizzare il numero di contenitori necessari a contenere gli oggetti

- Il problema è NP-Hard

- Possibilità di applicazione di tagli di tipo cover o di generazione di colonne

Modello matematico - le variabili

Q	Capacità di ogni possibile magazzino, uguale per ogni magazzino
P = 1…p	Punti nel piano, comprendono sia tutti i possibili magazzini, sia i clienti
S = 1…n	Magazzini di start
E = n+1…n*2	Magazzini di end
SE = 1…n*2	Insieme dei magazzini di start e di magazzini di end
C = n*2+1…p	Clienti totali

Modello matematico - Funzione obiettivo

$$\min\left(\sum_{i\in S} g_i x_i + \sum_{i\in P}\sum_{j\in P} d_{ij} y_{ij}\right)$$

Si vuole minimizzare il costo di costruzione dei magazzini e la distanza tra ogni nodo

In particolare:

- *g* è l'array dei costi di costruzione di ogni magazzino
- *d* è la matrice delle distanze tra due nodi
- *x* è l'array di variabili decisionali. Indica se il magazzino viene costruito o meno
- y rappresenta gli archi che connettono due nodi

Modello matematico - Variabili decisionali

$x_i \in \{0,1\} \quad \forall\, i \in SE$

La variabile è uguale a 1 se il magazzino deve essere costruito, uguale a 0 se non viene costruito

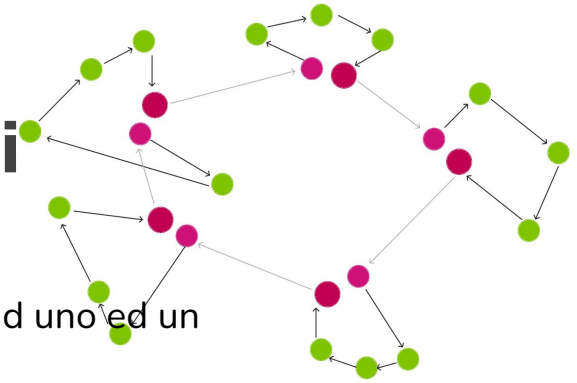
$y_{ij} \in \{0,1\} \quad \forall i,j \in P$

La variabile è uguale a 1 se nella soluzione esiste un arco tra i due nodi *i* e *j*, 0 altrimenti

$z_{ij} \in \{0,1\} \quad \forall i \in C, \forall j \in SE$

La variabile è uguale a 1 se il cliente *i* appartiene al magazzino *j*, 0 altrimenti

Modello matematico - Vincoli



$$\sum_{j \in S} z_{ij} = 1, \quad \forall i \in C$$

Ogni cliente appartiene ad uno ed un solo magazzino

$$\sum_{i \in C} q_i \cdot z_{ij} \leq Q, \quad \forall j \in S$$

Vincolo di capacità: per ogni magazzino, la somma delle domande dei suoi clienti deve essere minore o uguale la sua capacità

$$z_{ij} \leq x_j, \quad \forall i \in C, \forall j \in S$$

Se un magazzino è chiuso, allora nessun cliente può appartenere a quel magazzino

Quest'ultimo vincolo può collassare sul precedente creando un unico vincolo:

$$\sum_{i \in C} q_i \cdot z_{ij} \leq Q x_j, \quad \forall j \in S$$

Modello matematico - Vincoli

$$y_{ii} = 0, \forall i \in P$$

Non esistono archi che connettono un nodo a se stesso

$$\sum_{j \in P} y_{ij} = 1, \forall i \in P, i \neq j$$

Ogni nodo ha uno ed uno solo arco uscente; vincolo di stella uscente

$$\sum_{j \in P} y_{ji} = 1, \forall i \in P, i \neq j$$

Ogni nodo ha uno ed uno solo arco entrante; vincolo di stella entrante

Modello matematico - Vincoli

$$\sum_{j \in S} y_{ij} = 1, \forall i \in E, (i - n) \neq j$$

Dopo un magazzino di End ci deve essere un magazzino di Start. Tale magazzino non deve essere il suo corrispettivo

$$\sum_{j \in C} y_{ij} = x_i, \forall i \in S$$

Dopo ogni magazzino di Start aperto, ci deve essere un cliente

$$y_{ij} \leq z_{ij}, \forall i \in C, \forall j \in E$$

Ogni cliente che appartiene a un magazzino End aperto può avere un arco cliente-magazzino

Modello matematico - Vincoli

$$y_{ji} \leq z_{ij}, \forall i \in C, \forall j \in S$$

Ogni cliente che appartiene a un magazzino di Start aperto può avere un arco magazzino-cliente

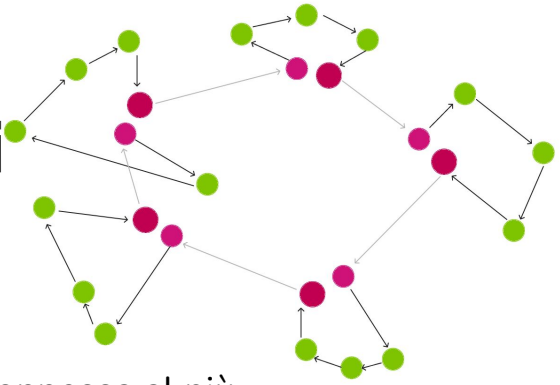
$$\sum_{i \in C} y_{ij} = x_j, \forall j \in E$$

Ogni magazzino di End, se costruito, viene raggiunto da uno ed un solo cliente

$$x_{i+n} = x_i, \forall i \in S$$

Se un magazzino di Start è aperto, allora sarà aperto anche il suo corrispettivo magazzino di End

Modello matematico - Vincoli



$$y_{ij} + y_{ji} \leq 1, \forall i, j \in P, i \neq j$$

Ogni coppia di nodi è connessa al più da un arco. In altre parole, non si può creare un ciclo formato da due nodi

Questo vincolo, apparentemente ridondante per l'esistenza del CutSet, è necessario affinché si scartino dei cicli formati da coppie di nodi.
Se venisse rimosso, il CutSet procederebbe per moltissime iterazioni, rendendo la risoluzione molto lenta.

Modello matematico - Vincoli

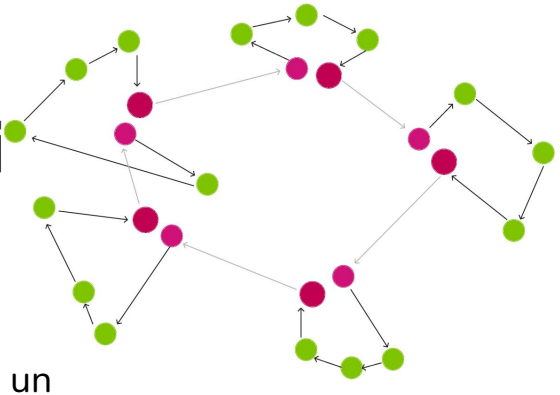
$$\sum_{j \in S} y_{ij} = 0, \forall i \in C$$

Per ogni cliente non deve esistere un arco diretto verso un magazzino di Start

$$z_{ij} = z_{i(j+n)}, \forall i \in C, j \in S$$

Se un cliente appartiene a un magazzino di Start, allora apparterrà anche al magazzino di End corrispondente

Modello matematico - Vincoli



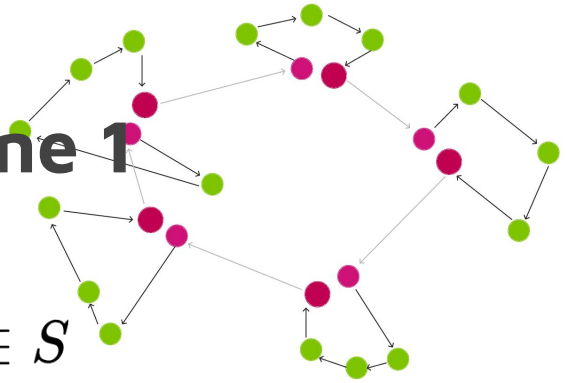
$$\sum_{i \in S} x_i \geq 1, |C| > 0$$

Se ci sono clienti, almeno un magazzino deve essere aperto

$$\sum_{i \in C} q_i \leq \sum_{j \in S} Q \cdot x_j$$

La somma delle domande di tutti i clienti deve essere al più uguale alla somma della capacità dei magazzini aperti

Modello matematico - Versione 1



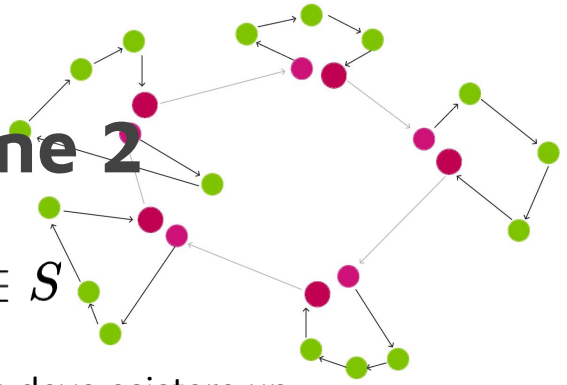
$$z_{ij} \ \wedge \ y_{ik} \rightarrow z_{kj}, \forall i,k \in C, j \in S$$

Se un cliente appartiene a un magazzino ed è connesso a un secondo cliente, allora anche il secondo cliente deve appartenere allo stesso magazzino.
Questo vincolo si crea suddividendolo in due parti:

$$z_{ij} \geq z_{kj} + y_{ik} - 1, \forall i \in C, \forall j \in S, \forall k \in C$$

$$z_{kj} \geq z_{ij} + y_{ik} - 1, \forall i \in C, \forall j \in S, \forall k \in C$$

Modello matematico - Versione 2



$$(z_{ik} + z_{jk}) \leq 1 \rightarrow y_{ij} = 0, \forall i,j \in C, \forall k \in S$$

Se una coppia di clienti appartiene a due magazzini diversi, allora non deve esistere un arco che li connette.
Creazione di un'ulteriore variabile di appoggio, *w*

$$w_{ijk} \in \{0,1\} \quad \forall i,j \in C, \forall k \in S$$

$$w_{ijk} \leq z_{ik}, \forall i,j \in C, \forall k \in S$$

$$w_{ijk} \leq x_k, \forall i,j \in C, \forall k \in S$$

$$w_{ijk} \leq z_{jk}, \forall i,j \in C, \forall k \in S$$

$$y_{ij} = \sum_{k \in S} w_{ijk}, \forall i,j \in C$$

Raffinamento per coppie

- Attivabili mediante parametro
- Controllo sulle coppie di clienti che appartengono a un magazzino (nel rilassamento continuo)
- Se una coppia di clienti dello stesso magazzino viola il vincolo di capacità, allora vanno separati in due magazzini diversi
- Vincoli generati in un ciclo, il quale termina quando per una iterazione non vengono più generati vincoli

$$z_{ik} + z_{jk} \leq 1, \forall i, j \in C, \forall k \in S$$

Modello matematico - Vincolo di CutSet

Per implementare il vincolo di CutSet si utilizza una visita del grafo per controllare che non sia già completamente connesso. Nel caso in cui ci siano dei nodi non ancora connessi, allora si aggiunge un nuovo vincolo. Si ripete la visita e l'aggiunta dei vincoli fino a quando il grafo non è completamente connesso

$$\sum_{i \in P} \sum_{j \in P} y_{ij} \geq 2, \; i \; not \, visited \; \vee \; j \, not \, visited$$

Vincolo di CutSet

Aspetti tecnici - debug, parametri e seed

- **stampa_matrici(true, true, true)** procedura per la stampa di vettori e matrici, come ad esempio:
  - **build_depot(i)** vettore dei magazzini costruiti (start e end)
  - **customer_depot(i,j)** per verificare l'appartenenza di un cliente i ad un magazzino j
  - **edge_exists(i,j)** per verificare nodo di inizio e fine degli archi (diagonale 0)
- **debug_magazzini_clienti**

---------- Debug magazzini clienti ----------
magazzino_start -> {clienti} -> magazzino_end | domanda_accumulativa
4 -> { 17 22 23 } -> 12 | 15

- **debug_archi**

---------- Debug archi ----------
nodo -> nodo -> nodo
---
23 -> 22 -> 17

Aspetti tecnici - debug, parametri e seed

locations_number := 8 ! Numero di locazioni dove poter costruire o meno i magazzini
customers_number := 12 ! Numero clienti da servire
depots_capacity := 15 ! Capacità dei magazzini
customer_max_demand := 10 ! Massima domanda dei singoli clienti
depot_max_cost := 100 ! Massimo costo di costruzione di un magazzino
point_max_range := 100 ! Range del grafo
clienteroutev1 := true ! Abilita/Disabilita vincolo Cliente-Route V1
clienteroutev2 := false ! Abilita/Disabilita vincolo Cliente-Route V2
raffinamento_coppie := false ! Abilita/Disabilita raffinamenti

Aspetti tecnici - debug, parametri e seed

- Per debuggare più facilmente abbiamo implementato dei parametri per direzionare l'output su file

debug_su_file := false ! Se a true, stampo su file output di debug
debug_file_name := "" + svgtimestamp + "_debug.txt" ! Nome del file di debug

- Per debuggare le configurazioni create dal generatore randomico, abbiamo implementato un seed che possiamo controllare per soffermarci e analizzarle caso per caso.

myseed := integer(round((100*random)+0.5)) ! Generatore di seed randomici 1-100
myseed := 59 ! Seed buono per il debugging? Lo salvo
setrandseed(myseed) ! Per poi settarlo

Aspetti tecnici - grafici

- Libreria SVG: **mmsvg**
- Parametri file immagini:
  - **svgcounter** := 0 ! Contatore
  - **svgtimestamp** := timestamp ! Timestamp
  - **saved** := true ! Se a true, salva le immagini nel filesystem
  - **filename** := "run" + svgtimestamp + "_plot" + svgcounter + ".svg"

- Procedure per il disegno: in ogni procedura creo un gruppo con **svgaddgroup,** a cui aggiungo degli elementi come **svgaddpoint, svgaddtext** oppure **svgaddarrow** e assegno uno stile con **svgsetstyle.**
  - **plot**: plotta su un grafico magazzini, clienti e routes con le seguenti procedure
  - **draw_locations**: plotta sul grafico le locazioni possibili dove costruire i magazzini
  - **draw_customers**: plotta sul grafico i clienti da servire
  - **draw_edges**: plotta sul grafico le routes

Aspetti tecnici - grafici

- **svgsetgraphviewbox**(-10, -10, point_max_range+20, point_max_range+20) ! imposta dei margini al grafico
- **svgsetgraphlabels**("x", "y") ! applica delle label agli assi cartesiani del piano
- **svgerase** ! Pulisce il grafo
- **svgsave**(filename) ! salva l'immagine nel filesystem
- **svgrefresh** ! Aggiorna il grafico
- **wait(1)** ! Mette in pausa l'esecuzione del codice per 1s
- **svgpause** ! Attende un comando dall'utente per continuare l'esecuzione

Risultati - è tutto corretto?

- Inizialmente i vincoli relativi alla versione 1 e versione 2 e i raffinamenti per coppie non erano stati ancora implementati
- Apparentemente il risultato sembrava già corretto MA...
  - Mismatch tra output grafico e output testuale
    - La matrice degli archi non corrispondeva con la matrice delle appartenenze cliente-magazzino
    - Talvolta si creavano cicli nei quali erano presenti più magazzini
- Tentativo di correzione dinamico sulla soluzione intera

Risultati - prima correzione

All’interno del ciclo di visita per creare i vincoli di CutSet, occorre vietare l’esistenza di alcuni archi per evitare inconsistenze nella soluzione. Infatti:

- Se da un cliente non esce un arco verso il magazzino di End, sicuramente dovrà uscire un arco verso i clienti che appartengono allo stesso magazzino

$$\sum_{j \in C} y_{ij} = 1, \forall i \in C, \forall k \in S, z_{jk} = 1, x_k = 1, z_{ik} = 1$$

- Altrimenti, se c’è un arco verso il magazzino di End, non uscirà nessun altro arco verso i clienti dello stesso magazzino

$$\sum_{j \in C} y_{ij} = 0, \forall k \in S, \forall i \in C, x_k = 1, z_{ik} = 1, z_{jk} = 1$$

Risultati - prima correzione

In ogni caso, non deve uscire nessun arco da un cliente verso un cliente di un altro magazzino

$$\sum_{j \in C} y_{ij} = 0, \forall k \in S, \forall i \in C, x_k = 1, z_{ik} = 1, z_{jk} = 0$$

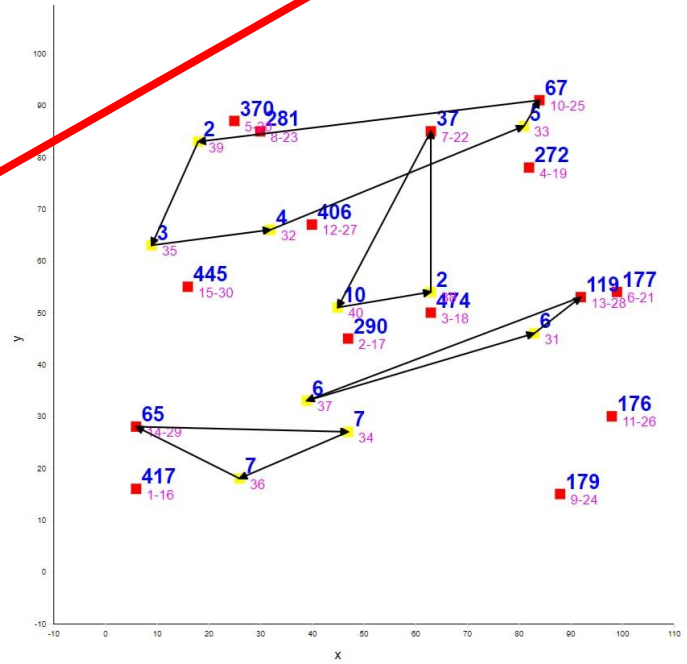
Inoltre, nessun arco di nessun altro magazzino di Start deve entrare nel cliente

$$\sum_{j \in S} y_{ji} = 0, \forall k \in S, \forall i \in C, x_k = 1, z_{ik} = 1, z_{ij} = 0$$

Risultati - prima correzione

PROBLEMI:

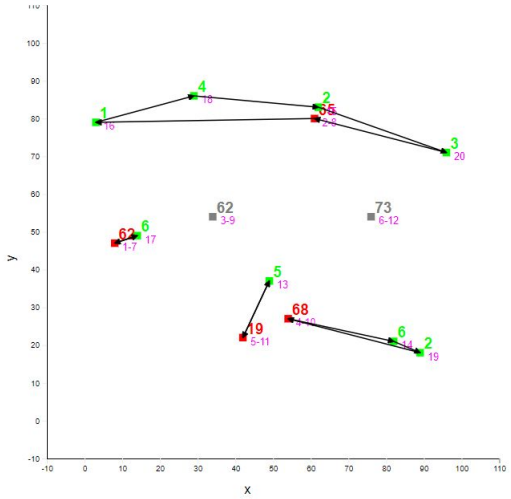
- I magazzini venivano scelti a priori e FISSATI con la prima esecuzione (prima risoluzione del modello)
- Di conseguenza, i cicli di CutSet non erano ottimi: veniva restituita una soluzione ma a colpo d'occhio si vedeva che non era la soluzione ottima



Risultati - versioni definitive

- Prima versione generalmente più veloce e meno dispendiosa in termini di memoria (non serve memorizzare una variabile decisionale d'appoggio)
- Versione con raffinamento per coppie (mediamente) più veloce computazionalmente (in termini del solo calcolo dell'operazione di *minimize*) rispetto quella senza
    - talvolta, la run con il raffinamento ha richiesto un maggior numero di iterazioni nel cutset, perdendo quindi il vantaggio di tempo ottenuto nella singola risoluzione
    - il vero vantaggio si riesce ad ottenere in esecuzioni di dimensioni sostenute, dove anche con un numero di iterazioni nel cutset maggiore, il risparmio di tempo nella *minimize* è tale da avere un tempo complessivo di esecuzione minore

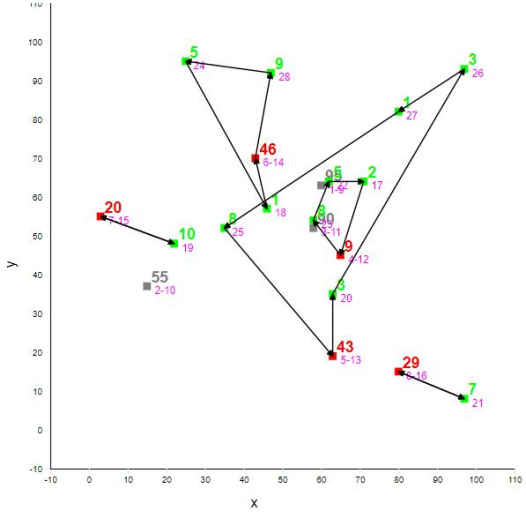
Risultati - Esempio 1



	Tempi	Nuovi vincoli CutSet	Nuovi vincoli raffinamento	Ultima minimize
Versione 1 - base	5,06 s	1	0	1,79 s
Versione 1 - raffinata	4,80 s	1	2	1,53 s
Versione 2 - base	68,20 s	9	0	2,46 s
Versione 2 - raffinata	31,44 s	3	0	2,33 s

seed 35 - 6 magazzini - 8 clienti - domanda massima clienti 8 - costo apertura massima 100 - capacità magazzino 10

Risultati - Esempio 2

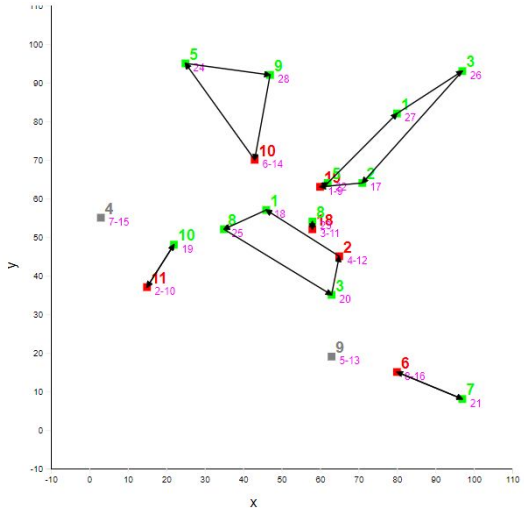


	Tempi	Nuovi vincoli CutSet	Nuovi vincoli raffinamento	Ultima minimize
Versione 1 - base	121,72 s	12	0	18,99 s
Versione 1 - raffinata	197,50 s	23	8	15,23 s

Anche se sono state effettuate molte più iterazioni in fase di CutSet, con il raffinamento il calcolo di *minimize* è molto più performante

seed 59 - 8 magazzini - 12 clienti - domanda massima clienti 10 - costo apertura massima 100 - capacità magazzino 15

Risultati - Esempio 3



	Tempi	Nuovi vincoli CutSet	Nuovi vincoli raffinamento	Ultima minimize
Versione 1 - base	25,47s	9	0	3,13 s
Versione 1 - raffinata	38,20s	20	3	3,02 s

A parità di tutte le altre condizioni, definendo un costo di apertura più basso, si preferisce aprire più magazzini ed effettuare route più brevi

seed 59 - 8 magazzini - 12 clienti - domanda massima clienti 10 - costo apertura massima 20 - capacità magazzino 15

Possibili aggiornamenti

- Con le adeguate modifiche, il vincolo $y_{ij} + y_{ji} \leq 1, \forall i, j \in P, i \neq j$ potrebbe essere espanso racchiudendo ogni tripletta di nodi anziché ogni coppia
- Raffinamento possibile anche su triplette

- Integrality gap migliorabile? (mediamente 36%)
  - Possibilità di creazione di tagli di tipo cover
- Esiste una formulazione migliore?
  - Column generation

- Il problema si poteva strutturare anche diversamente, sfruttando diverse route singole e mantenendo i magazzini non duplicati. Questo comporterebbe sicuramente un uso massiccio di variabili decisionali aggiuntive e di cicli for innestati, probabilmente facendo perdere di efficienza al problema così strutturato