Algoritmo VNS per il Bike sharing Rebalancing Problem

Relazione di progetto

Algoritmi di ottimizzazione, A.A. 2020/21
Corso di Laurea Magistrale in Informatica
Università degli Studi di Modena e Reggio Emilia

Nome e cognome: Nicola Gambini
Matricola: 137755

---

1. Descrizione del problema

1.1. INTRODUZIONE

Un sistema di bike sharing è un sistema nel quale un ente, pubblico o privato, mette a disposizione dei propri utenti un insieme di biciclette che possono essere usate temporaneamente dagli stessi.
Tali biciclette sono organizzate all'interno di stazioni, dislocate all'interno di un territorio (es. una città), dalle quali gli utenti possono prelevare biciclette e/o depositarle una volta terminato l'utilizzo.

Ognuna di queste stazioni definisce un proprio *livello di occupazione ottimale*, ovvero una quantità di biciclette ideale che garantisca un'adeguata capacità del servizio (in termini di biciclette prelevabili dalla stazione) e che, al contempo, garantisca che una parte degli slot sia lasciata libera, in modo tale da rendere possibili eventuali depositi da parte degli utenti.
Al fine di offrire un servizio efficiente, l'ente che gestisce il sistema di bike sharing deve prevedere un *sistema di ribilanciamento* delle biciclette in grado di riportare, periodicamente, le stazioni al loro stato ottimale. Questa operazione di ribilanciamento[1] viene effettuata mediante l'utilizzo di una flotta di veicoli omogenei che, partendo da un singolo deposito, visitano ogni stazione ritirandovi (o rilasciandovi) biciclette con lo scopo di ripristinarne il livello di occupazione ottimale.

Il problema sopra descritto prende il nome di *Bike sharing Rebalancing Problem* (BRP) e può essere classificato come un *One-commodity Pickup and Delivery Vehicle Routing Problem* (1-PDVRP), ovvero una variante del VRP che prevede le seguenti caratteristiche aggiuntive:

- il ricollocamento di una singola tipologia di beni ("One-commodity") - ovvero le biciclette;
- la possibilità, da parte dei veicoli, di ritirare ("Pickup") e/o depositare ("Delivery") tali prodotti all'interno delle loro tratte (pur rispettando i limiti di capacità previsti).

L'obiettivo del problema è quello di determinare le tratte dei veicoli in modo tale che:

a) tutte le stazioni vengano visitate una volta, e che le loro *demand* di biciclette vengano soddisfatte (nel caso in cui si trovino già nella condizione ottimale, devono comunque essere visitate);
b) la distanza percorsa dai veicoli sia la minore possibile.

---

[1] nella presente relazione verrà considerato il ribilanciamento nella sua versione *statica*, secondo la quale esso viene effettuato quando il servizio non è attivo (ovvero quando gli utenti non possono usufruire del servizio; ad esempio, durante la notte)

1.2. FORMULAZIONE MATEMATICA DEL PROBLEMA

Il problema in questione può essere modellato tramite l'utilizzo di un grafo $G = (V, A)$ diretto e completo, nel quale i nodi $i \in V$ rappresentano le stazioni ($i = 0$ è il deposito), mentre gli archi $(i, j) \in A$ rappresentano i collegamenti di percorrenza tra esse.
Ad ogni stazione $i \in V \setminus 0$ è associata una demand di bike $q_i \in \mathbb{Z}$, mentre ad ogni collegamento è associato un costo di percorrenza $c_{ij} \geq 0$.
I veicoli hanno una capacità massima pari a Q biciclette.

1.3. MODELLO DI OTTIMIZZAZIONE

Si riporta di seguito il modello di ottimizzazione per il BRP proposto in [2]:

$$\text{min} \quad z = \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \tag{1}$$

$$\text{s.t.} \quad \sum_{i \in V} x_{ij} = 1 \quad \forall j \in V \setminus \{0\} \tag{2}$$

$$\sum_{i \in V} x_{ji} = 1 \quad \forall j \in V \setminus \{0\} \tag{3}$$

$$\sum_{j \in V} x_{0j} \leq m \tag{4}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq S - \max \left\{ 1, \left\lceil \frac{|\sum_{i \in S} q_i|}{Q} \right\rceil \right\} \qquad S \subseteq V \setminus \{0\}, S \neq \varnothing \tag{5}$$

$$x_{ij} \in \{0,1\}, \quad i, j \in V \tag{6}$$

Le variabili binarie x_{ij} sono da intendere pari a 1 nel caso in cui l'arco che collega i nodi $i, j \in V$ sia percorso da un veicolo, 0 altrimenti.
La funzione obiettivo (1) definisce la minimizzazione dei costi di percorrenza, mentre i vincoli (2) e (3) impongono che le stazioni vengano visitate una e una sola volta.
Il vincolo (4) garantisce che vengano impiegati al più m veicoli.
I vincoli (5), infine, garantisce che per ogni subset di nodi $S \subseteq V \setminus \{0\}$ il numero di archi con inizio e fine in S non superi la cardinalità di S meno il numero minimo di veicoli necessari per servire S (*generalized subtour elimination constraints*). Si noti che, come anche sottolineato in [2], i vincoli (5) hanno una cardinalità esponenziale.

2. Letteratura di riferimento

Trattandosi di un problema dalla rilevante importanza pratica, nel corso degli ultimi anni il BRP è diventato oggetto di interesse da parte di numerose ricerche.
Ai fini della presente relazione si è deciso di considerare, all'interno della letteratura esistente, i lavori di seguito riportati:

- "The bike sharing rebalancing problem: Mathematical formulations and benchmark instances" di Dell'Amico et al. (2013) [1], nel quale viene proposta una risoluzione del problema tramite l'utilizzo di un metodo Branch-and-cut (ottimizzazione esatta);
- "A Destroy and Repair Algorithm for the Bike sharing Rebalancing Problem" di Dell'Amico et al. (2016) [2], nel quale viene proposto un algoritmo metaeuristico di Destroy and Repair;
- "Tabu Search Algorithm for the Bike Sharing Rebalancing Problem" di Pan et al. (2020) [3], nel quale viene proposto un algoritmo metaeuristico di Tabu Search.

All'interno della presente relazione verrà proposto un algoritmo metaeuristico di Variable Neighborhood Search (VNS), i cui risultati verranno comparati con quelli ottenuti dai tre paper sopracitati.

3. Descrizione delle istanze benchmark

Al fine di valutare l'efficacia dell'algoritmo di VNS proposto si è scelto di utilizzare le istanze benchmark proposte in [1], reperibili all'indirizzo [5]. Si tratta di un dataset di 65 istanze contenenti i dati di 22 sistemi di bike sharing implementati in diverse città (proposti in più varianti, sulla base della capacità massima dei veicoli impostata).
Ogni file di istanza contiene al proprio interno le seguenti informazioni:

- la quantità di stazioni $|V|$;
- le demand q_i di ogni stazione $i \in V$;
- la capacità massima dei veicoli Q;
- la matrice delle distanze $c_{ij} \forall i, j \in V$.

All'interno della relazione si farà riferimento alle istanze sopracitate in base alla classificazione proposta in [1], secondo la quale ogni istanza viene definita:

- *small*, se $|V| \leq 50$;
- *medium*, se $50 < |V| < 100$;
- *large*, se $|V| \geq 100$.

L'utilizzo delle istanze designate è risultato particolarmente conveniente, dal momento che le stesse istanze vengono utilizzate sia in [1], che in [2] che in [3]: questo fatto ha reso il possibile il confronto incrociato dei risultati con le soluzioni ottenute dai metodi già presenti in letteratura.

Si riporta in **Figura 1** la rappresentazione grafica di un'istanza di esempio (6ReggioEmilia10.txt), nella quale è possibile osservare la struttura del grafo delle percorrenze (generato tramite l'utilizzo di un layout force-based, che ne dispone i nodi secondo le distanze triangolari definite dai costi degli archi) e l'insieme delle stazioni, colorate in base alla loro demand di biciclette.

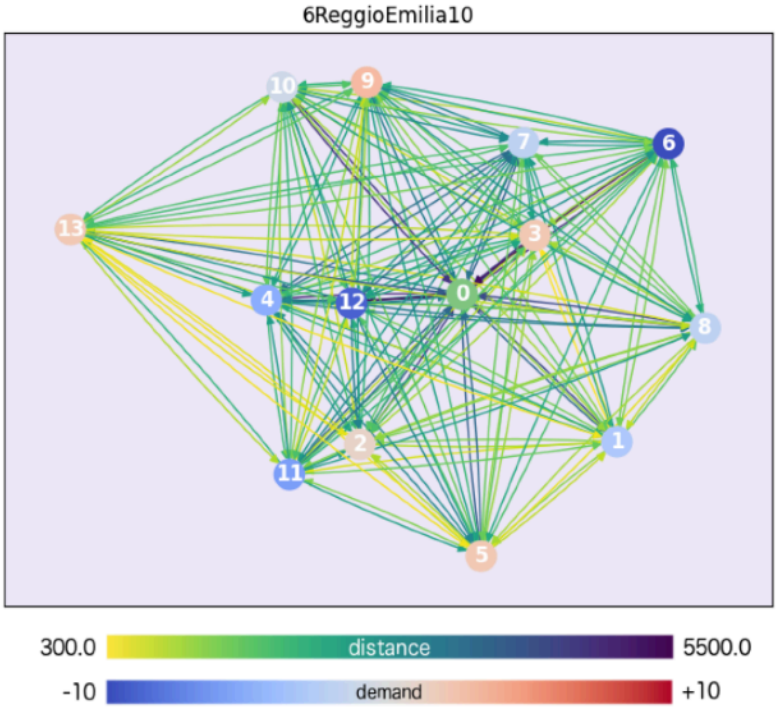

Figura 1: rappresentazione grafica dell'istanza 6ReggioEmilia10.txt

4. Dettagli implementativi

L'implementazione proposta è stata realizzata utilizzando il linguaggio di programmazione Python (versione dell'interprete 3.7.2), ed è stata strutturata secondo lo schema di ricerca tramite VNS classico - riportato in **Algoritmo 1**.
Si noti come, per poter essere implementato, l'algoritmo di VNS necessiti di:

- un metodo per la generazione della soluzione iniziale (riga 1);
- una lista di $k_{\max}$ metodi di generazione di neighborhood sulla quale iterare (righe 2-12);
- una o più condizioni di terminazione, sia per l'algoritmo di Local Search (riga 6) che per l'algoritmo VNS (riga 13).

Algorithm VNS
1. find an initial solution $\mathbf{x}$; $\mathbf{x}^* = \mathbf{x}$;
2. **repeat**
3. $k := 1$:
4. **while** $(k < k_{\max})$ **do**
5. randomly select a solution $\mathbf{x}' \in N_k(\mathbf{x})$;(shake)
6. $\mathbf{x} :=$ Local_Search($\mathbf{x}'$,k);
7. **if** $(f(\mathbf{x}) < f(\mathbf{x}^*))$ **then**
8. $\mathbf{x}^* := \mathbf{x}$; $k := 1$;
9. **else**
10. $k := k + 1$;
11. **end if**
12. **end while**
13. **until** (stopping condition)
14. **return**($\mathbf{x}^*$)

Algoritmo 1: algoritmo generale di VNS

Per quanto riguarda il terzo punto dell'elenco sopra riportato, sono state implementate le seguenti condizioni di terminazione per l'algoritmo di VNS (riga 13):

a) l'esecuzione termina qualora venga raggiunto un limite massimo di iterazioni (da intendersi come iterazioni del ciclo while riportato alla riga 4 di Algoritmo 1; max_it=5000);
b) l'esecuzione termina qualora la medesima soluzione risulti essere la migliore per un determinato numero di volte consecutive (max_consecutive_solutions=20);
c) l'esecuzione termina qualora venga raggiunto un limite di tempo (in secondi), pari a:
   - max_time=10 per le istanze small;
   - max_time=600 per le istanze medium;
   - max_time=1800 per le istanze large.

Per l'algoritmo di Local Search (riga 6), invece, è stato previsto che l'algoritmo termini qualora il delta tra il valore della soluzione corrente e quello della migliore soluzione trovata risulti essere inferiore ad una certa soglia di tolleranza (tolerance=1e-50).

Seguono le descrizioni relative al metodo costruttivo ed ai neighborhood utilizzati per implementare l'algoritmo di VNS sopra indicato.

4.1. METODO COSTRUTTIVO

Al fine di generare la soluzione iniziale - propedeutica all'esecuzione dell'algoritmo di VNS - si è deciso di implementare l'algoritmo *Savings&Losses* proposto in [2]. Tale algoritmo estende il noto algoritmo di Savings proposto da Clarke-Wright in [4], introducendo il concetto di "loss of flexibility"; segue una descrizione dell'algoritmo utilizzato.

Data un'istanza descritta dal grafo $G = (V, A)$ si genera una prima soluzione composta da $|V|$ tratte, ognuna delle quali è composta da una singola stazione (soluzione sempre ammissibile). A partire da questa soluzione viene quindi lanciato un processo iterativo di *merging* delle tratte così strutturato:

1. si valutano tutti i possibili merge tra coppie di tratte $P \oplus R$ (generati concatenando le tratte risultanti dalla rimozione dell'ultimo nodo di P ed il primo di R). I merge che superano il controllo di ammissibilità passano al punto successivo (si rimanda a [2] per ulteriori dettagli sul controllo di ammissibilità delle soluzioni);
2. per ogni merge viene calcolata la relativa funzione di *saving*, definita come $S_{P\oplus R} = c_{0,R_1} + c_{P_{|P|-1},0} - c_{P_{|P|-1},R_1}$. Questa funzione descrive il risparmio di costo che si otterrebbe concatenando le tratte P e R;
3. per ogni merge viene calcolata la relativa funzione di *loss*, definita come $L_{P\oplus R} = -(\Delta_P + \Delta_R - 2\Delta_{P\oplus R})$, dove Δ_P rappresenta l'*amount of feasibility* della tratta P (ovvero una misura di quanto tale tratta sia predisposta ad essere concatenata, in maniera ammissibile, ad altre tratte; si rimanda a [2] per ulteriori dettagli su questa definizione). Intuitivamente, più $L_{P\oplus R}$ assume un valore negativo, più si riduce la possibilità che il merge risultante si presti ad essere ulteriormente concatenato nelle iterazioni successive;
4. per ogni merge si calcola la funzione di *score* definita come $E_{P\oplus R} = \alpha S_{P\oplus R} + (1-\alpha)L_{P\oplus R}$. Questa funzione risulta essere un *tradeoff* tra la funzione di saving e quella di loss, delle quali è possibile controllare l'incidenza sul risultato finale tramite il tuning del parametro α (che verrà fissato ad $\alpha = 0.7335$ durante la fase di test, come indicato in [2]).

Una volta ottenuti gli score dei merge candidati, si seleziona quello avente lo score più elevato e lo si applica alla soluzione esistente;

5. nel caso in cui non risulti possibile ottenere nuovi merge ammissibili, l'algoritmo termina; in caso contrario, si riprende l'esecuzione dal punto 1.

Al termine di questo processo, ciò che si ottiene è una soluzione ammissibile da potere utilizzare per inizializzare l'algoritmo di ricerca tramite neighborhood.

4.2. DEFINIZIONE DEI NEIGHBORHOOD

Si riportano di seguito i metodi implementati per la generazione dei neighborhood[2], proposti originariamente in [2].
Si evidenzia che, all'interno dell'implementazione proposta nella presente consegna, sono stati effettuati gli opportuni controlli di ammissibilità (sempre secondo quanto indicato in [2]) sulle soluzioni generate da ciascun metodo.
Come unica nota, si ritiene opportuno riportare che tali controlli sono stati realizzati mediante l'utilizzo specifiche proprietà matematiche basate sul concetto di *load windows* (che definiscono, per ogni stazione, l'intervallo di possibili valori di carico che un veicolo deve rispettare affinché la tratta della quale la stazione fa parte sia ammissibile). Come affermato in [2], il fatto di utilizzare controlli di questo tipo implica, per la maggior parte dei metodi, un abbassamento dei costi computazionali (e quindi un risparmio in termini di tempo, rispetto a quanto ne richiederebbero una serie di controlli estensivi).
Si rimanda a [2] per ulteriori dettagli riguardanti tali controlli di ammissibilità.

1. *Move*: si seleziona una stazione, la si rimuove dalla tratta in cui si trova e la si inserisce in un'altra posizione, che può essere all'interno della stessa tratta o all'interno di un'altra tratta. Si noti che, nel caso in cui la stazione di interesse venga rimossa da una tratta mono-stazione, la tratta di partenza viene rimossa dalla soluzione (**Figura 2**);

Figura 2: rappresentazione grafica di un'operazione di move tra due tratte

2. *Or-opt(κ)*: si seleziona una sequenza di stazioni all'interno di una tratta avente una lunghezza pari o inferiore a κ (fissato a $\kappa = 35$ durante la fase di test, come indicato in [2]) e la si inserisce in un'altra posizione, che può essere all'interno della stessa tratta o all'interno di un'altra. Si noti che, nel caso in cui la tratta di partenza sia lunga esattamente $\kappa + 2$, tale tratta viene rimossa dalla soluzione (**Figura 3**);

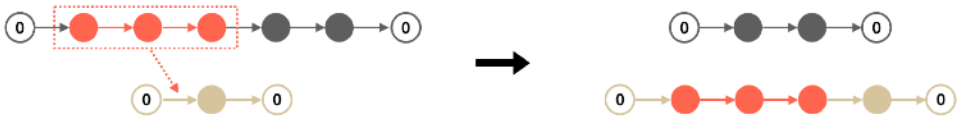

Figura 3: rappresentazione grafica di un'operazione di Or-opt tra due tratte ($\kappa \geq 3$)

[2] il processo di generazione dei neighbor, in quanto particolarmente oneroso per via dell'alto numero di soluzioni generate, è stato parallelizzato su N=4 processi indipendenti con il fine di migliorare le prestazioni

3. *Swap(1,1)*: si selezionano due stazioni, appartenenti ad una o due tratte, e le si scambia (**Figura 4**);

Figura 4: rappresentazione grafica di un'operazione di swap(1,1) tra due tratte

4. *Swap(2,2)*: si selezionano due sequenze di stazioni distinte aventi lunghezza pari a 2, appartenenti a una o due tratte, e le si scambia (**Figura 5**);

Figura 5: rappresentazione grafica di un'operazione di swap(2,2) tra due tratte

5. *Swap(1,1,1)*: si selezionano tre stazioni, appartenenti ad una, due o tre tratte, e le si scambia nei due modi possibili (v_1 con v_2, v_2 con v_3 e v_3 con v_1 oppure v_1 con v_3, v_3 con v_2 e v_2 con v_1; **Figura 6**);

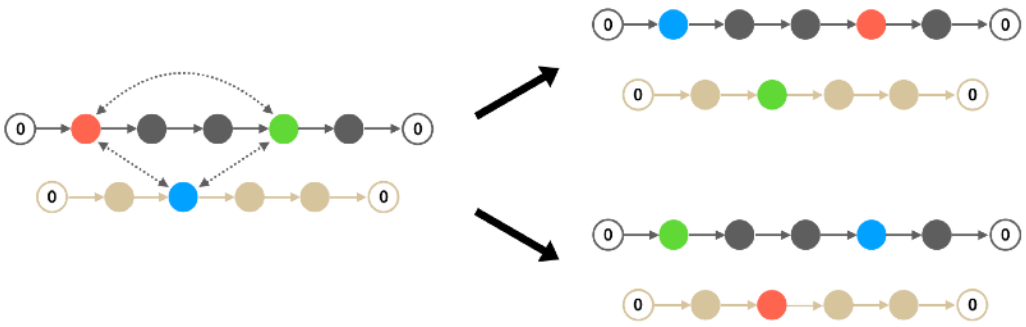

Figura 6: rappresentazione grafica di un'operazione di swap(1,1,1) tra due tratte

6. *Cross*: si selezionano due tratte distinte, una stazione per ciascuna di esse e si scambiano le porzioni di tratta successive alle stazioni selezionate (**Figura 7**). Si noti che nel caso in cui la stazione designata per la prima tratta sia la penultima, e quella per la seconda tratta sia la prima stazione, si verifica un caso particolare per il quale le due tratte vengono fuse (provocando la rimozione della seconda tratta);

Figura 7: rappresentazione grafica di un'operazione di cross tra due tratte

7. *Cross(3)*: si selezionano tre tratte distinte, una stazione per ciascuna di esse e le si scambia nei due modi possibili come descritto per il neighborhood *cross* (**Figura 8**).

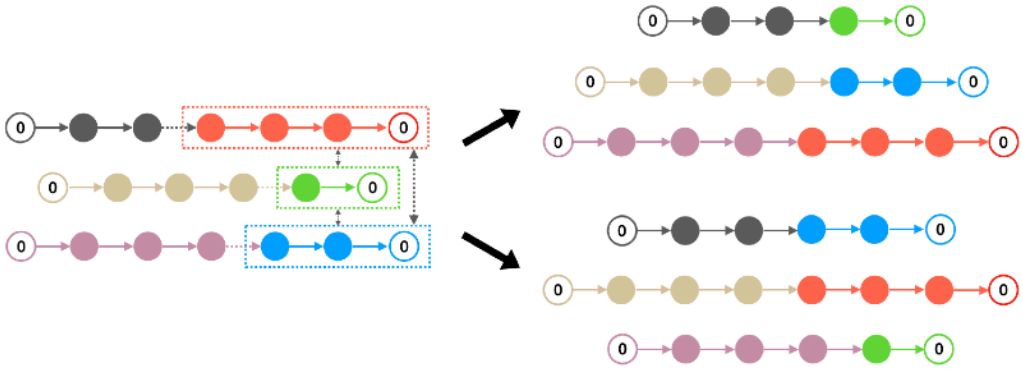

Figura 8: rappresentazione grafica di un'operazione di cross(3) tra tre tratte

All'interno di [2] viene anche riportato che, a fronte di test computazionali preliminari, si è stabilito che l'ordinamento dei neighborhood 1-3-5-4-6-2-7 fosse quello migliore da utilizzare; nella presente implementazione è stato mantenuto il medesimo ordinamento.

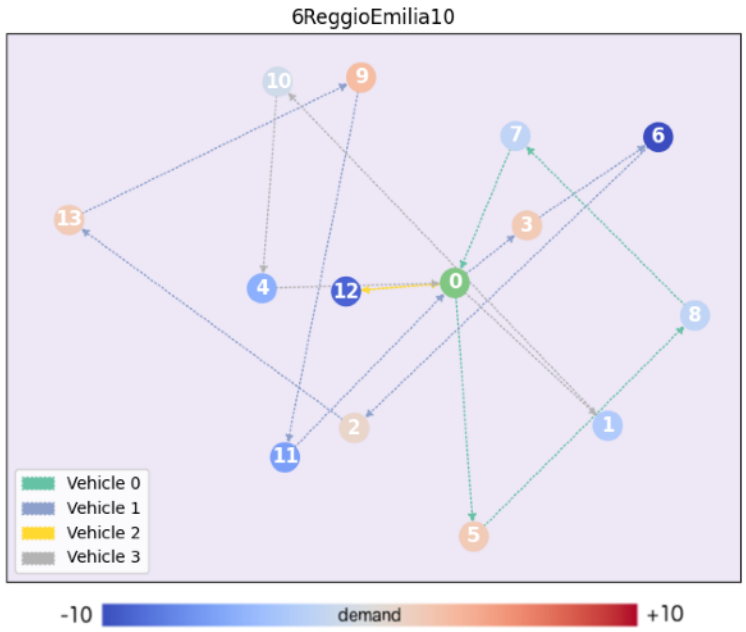

Figura 9: rappresentazione grafica di soluzione ottenuta sull'istanza 6ReggioEmilia10.txt ($z_{\min} = 40700$)

5. Confronto dei risultati

In questa ultima sezione verranno presentati i risultati ottenuti dall'esecuzione dell'algoritmo implementato sulle istanze benchmark di riferimento.
Allo scopo di ottenere un'indicazione sulla robustezza dell'algoritmo proposto si è deciso di eseguire, su ognuna delle istanze, un certo numero di *run* (fissato a 10). Dalle informazioni memorizzate a seguito di ogni run si è potuto estrarre, per ogni istanza:

- la migliore soluzione trovata (ed il valore $z_{\min}$ ad essa associato);

- lo scostamento percentuale medio $avg_{\%gap}$ tra i valori delle soluzioni trovate ed il valore della soluzione migliore;
- il tempo di esecuzione medio t_{avg} delle run.

All'interno delle tabelle **Tabella 1** e **Tabella 2** è possibile visionare il riepilogo dei risultati ottenuti dall'algoritmo di VNS proposto, comparati con quelli ottenuti dall'utilizzo dei metodi proposti in [1], [2] e [3] (ai quali ci si riferisce, all'interno delle suddette tabelle, con le rispettive sigle "B&C", "DR" e "TS").

Il focus di tali confronti risiede nella valutazione del divario percentuale tra le migliori soluzioni ottenute tramite VNS e le migliori soluzioni (o upper/lower bound) ottenute dai metodi sopra riportati.

5.1 COMMENTO DEI RISULTATI

I test eseguiti sull'algoritmo VNS hanno evidenziato un'inferiore efficacia dello stesso, in termini di qualità delle soluzioni ottenute, rispetto agli algoritmi già presenti in letteratura.
Dai risultati ottenuti, infatti, emerge un gap medio del 14.86% con le soluzioni ottenute tramite l'utilizzo dell'algoritmo di Branch-and-cut proposto [1], del 14.16% con quelle ottenute tramite l'utilizzo della metaeuristica di Destroy&Repair proposta [2] e del 12.81% con la metaeuristica di Tabu Search proposta in [3].

Le soluzioni ottenute dalle run VNS presentano uno scostamento medio del 1.86% dalla migliore soluzione trovata.

		VNS					B&C		DR		TS	
original_idx	city	stations_count	vehicles_capacity	local_optimum_value	avg_percentage_gap	avg_execution_time	local_optimum_value	vns_gap	local_optimum_value	vns_gap	local_optimum_value	vns_gap
1	Bari	13	30	15400	0,0000%	9,5441	14600	-5,19%	14600	-5,19%	14600	-5,19%
2	Bari	13	20	17900	0,0000%	15,5009	15700	-12,29%	15700	-12,29%	15700	-12,29%
3	Bari	13	10	21900	0,0000%	10,7534	20600	-5,94%	20600	-5,94%	20600	-5,94%
4	ReggioEmilia	14	30	20400	0,0000%	7,5812	16900	-17,16%	16900	-17,16%	16900	-17,16%
5	ReggioEmilia	14	20	23800	4,9160%	15,2706	23200	-2,52%	23200	-2,52%	23200	-2,52%
6	ReggioEmilia	14	10	34300	16,0058%	15,2058	32500	-5,25%	32500	-5,25%	32500	-5,25%
7	Bergamo	15	30	12800	3,3594%	12,2855	12600	-1,56%	12600	-1,56%	12600	-1,56%
8	Bergamo	15	20	13000	2,8462%	15,2189	12700	-2,31%	12700	-2,31%	12700	-2,31%
9	Bergamo	15	12	14700	2,0408%	15,1366	13500	-8,16%	13500	-8,16%	13500	-8,16%
10	Parma	15	30	29600	4,3581%	13,9763	29000	-2,03%	29000	-2,03%	29000	-2,03%
11	Parma	15	20	30500	2,2951%	14,2994	29000	-4,92%	29000	-4,92%	29000	-4,92%
12	Parma	15	10	32900	3,2827%	15,5671	32500	-1,22%	32500	-1,22%	32500	-1,22%
13	Treviso	18	30	29858	0,0389%	15,4828	29259	-2,01%	29259	-2,01%	29259	-2,01%
14	Treviso	18	20	32897	6,0236%	15,1903	29259	-11,06%	29259	-11,06%	29259	-11,06%
15	Treviso	18	10	32064	6,4172%	15,1617	31443	-1,94%	31443	-1,94%	31443	-1,94%
16	LaSpezia	20	30	21705	0,6823%	15,6408	20746	-4,42%	20746	-4,42%	20746	-4,42%
17	LaSpezia	20	20	21705	1,2195%	15,5943	20746	-4,42%	20746	-4,42%	20746	-4,42%
18	LaSpezia	20	10	24971	1,6091%	15,3735	22811	-8,65%	22811	-8,65%	22811	-8,65%
19	BuenosAires	21	30	92673	0,0000%	7,1586	76999	-16,91%	76999	-16,91%	76999	-16,91%
20	BuenosAires	21	20	109383	0,0000%	15,0643	91619	-16,24%	91619	-16,24%	91619	-16,24%
21	Ottawa	21	30	18091	0,0000%	15,6832	16202	-10,44%	16202	-10,44%	16202	-10,44%
22	Ottawa	21	20	18091	7,0941%	16,0385	16202	-10,44%	16202	-10,44%	16202	-10,44%
23	Ottawa	21	10	23005	0,1552%	15,1336	17576	-23,60%	17576	-23,60%	17576	-23,60%
24	SanAntonio	23	30	24736	2,5590%	17,2101	22982	-7,09%	22982	-7,09%	22982	-7,09%
25	SanAntonio	23	20	30477	0,0607%	15,3498	24007	-21,23%	24007	-21,23%	24007	-21,23%
26	SanAntonio	23	10	46467	2,8457%	15,7778	40149	-13,60%	40149	-13,60%	40149	-13,60%
27	Brescia	27	30	31700	6,3091%	15,8634	30300	-4,42%	30300	-4,42%	30300	-4,42%
28	Brescia	27	20	36700	5,3951%	16,7602	31100	-15,26%	31100	-15,26%	31100	-15,26%
29	Brescia	27	11	44800	1,1830%	13,3101	35200	-21,43%	35200	-21,43%	35200	-21,43%
30	Roma	28	30	69700	0,2439%	13,8279	61900	-11,19%	61900	-11,19%	61900	-11,19%
31	Roma	28	20	72500	0,0000%	10,5244	66600	-8,14%	66600	-8,14%	66600	-8,14%
32	Roma	28	18	73400	0,0000%	13,6957	68300	-6,95%	68300	-6,95%	68300	-6,95%
33	Madison	28	30	40510	0,0000%	24,8250	29246	-27,81%	29246	-27,81%	29246	-27,81%
34	Madison	28	20	37828	2,0384%	15,1754	29839	-21,12%	29839	-21,12%	29839	-21,12%
35	Madison	28	10	47063	1,0915%	19,5142	33848	-28,08%	33848	-28,08%	33848	-28,08%
36	Guadalajara	41	30	60853	0,8938%	54,4042	57476	-5,55%	57476	-5,55%	57476	-5,55%
37	Guadalajara	41	20	64186	0,1980%	30,5760	59493	-7,31%	59493	-7,31%	59493	-7,31%
38	Guadalajara	41	11	74577	0,0000%	15,1853	64981	-12,87%	64981	-12,87%	64981	-12,87%
39	Dublin	45	30	42224	1,2401%	18,1110	33548	-20,55%	33548	-20,55%	33548	-20,55%
40	Dublin	45	20	47230	5,0472%	32,9281	39786	-15,76%	39786	-15,76%	39786	-15,76%
41	Dublin	45	11	65446	2,8825%	15,7083	54392	-16,89%	54392	-16,89%	54392	-16,89%
Avg.					2,3008%			-10,83%		-10,83%		-10,83%

Tabella 1: risultati ottenuti sulle istanze *small* ($0 < |V| \leq 50$)

		VNS					B&C					DR		TS	
original_idx	city	stations_count	vehicles_capacity	local_optimum_value	avg_percentage_gap	avg_execution_time	local_optimum_LB	local_optimum_UB	LB_UB_gap	vns_gap_LB	vns_gap_UB	local_optimum	vns_gap	local_optimum	vns_gap
42	Denver	51	30	55677	4,0297%	650,5250	51583	51583	0,0000%	-7,35%	-7,35%	51583	-7,35%	51583	-7,35%
43	Denver	51	20	59183	1,5207%	527,1145	53465	53465	0,0000%	-9,66%	-9,66%	53465	-9,66%	53465	-9,66%
44	Denver	51	10	81739	2,3831%	619,6249	67459	67459	0,0000%	-17,47%	-17,47%	67459	-17,47%	67459	-17,47%
45	RioDeJaneiro	55	30	137936	1,9397%	675,5394	122547	122547	0,0000%	-11,16%	-11,16%	122547	-11,16%	122547	-11,16%
46	RioDeJaneiro	55	20	194486	1,3460%	600,3665	155446	156140	0,4445%	-20,07%	-19,72%	155517	-20,04%	160991	-17,22%
47	RioDeJaneiro	55	10	300045	0,3689%	546,8360	253690	259049	2,0687%	-15,45%	-13,66%	257147	-14,30%	268665	-10,46%
48	Boston	59	30	88474	0,3013%	1049,2000	65669	65669	0,0000%	-25,78%	-25,78%	65669	-25,78%	65669	-25,78%
49	Boston	59	20	99720	0,0000%	600,3457	71879	71879	0,0000%	-27,92%	-27,92%	71916	-27,88%	71879	-27,92%
50	Boston	59	16	123369	0,0000%	601,2994	74790	75065	0,3663%	-39,38%	-39,15%	75085	-39,14%	82373	-33,23%
51	Torino	75	30	64702	0,0000%	600,4661	47634	47634	0,0000%	-26,38%	-26,38%	47634	-26,38%	50958	-21,24%
52	Torino	75	20	66651	4,4197%	757,0000	50204	50204	0,0000%	-24,68%	-24,68%	50438	-24,33%	53849	-19,21%
53	Torino	75	10	81013	1,1498%	601,4915	58814	64797	9,2335%	-27,40%	-20,02%	61717	-23,82%	64100	-20,88%
54	Toronto	80	30	59293	0,0000%	600,2761	40794	41549	1,8171%	-31,20%	-29,93%	41390	-30,19%	47495	-19,90%
55	Toronto	80	20	66785	0,0000%	600,5300	42621	47898	11,0172%	-36,18%	-28,28%	46631	-30,18%	50351	-24,61%
56	Toronto	80	12	75208	0,0000%	600,4437	54238	60763	10,7384%	-27,88%	-19,21%	58539	-22,16%	61658	-18,02%
57	Miami	82	30	158642	0,9791%	674,5033	152229	156104	2,4823%	-4,04%	-1,60%	154038	-2,90%	155994	-1,67%
58	Miami	82	20	221235	1,0465%	605,5014	209379	229237	8,6627%	-5,36%	3,62%	214250	-3,16%	219710	-0,69%
59	Miami	82	10	447407	0,9613%	775,6190	390536	415762	6,0674%	-12,71%	-7,07%	397921	-11,06%	429702	-3,96%
60	CiudadDeMexico	90	30	107801	2,9073%	603,5081	67894	88227	23,0462%	-37,02%	-18,16%	72279	-32,95%	77473	-28,13%
61	CiudadDeMexico	90	20	124661	1,7406%	602,6695	88952	116418	23,5926%	-28,64%	-6,61%	94319	-24,34%	95375	-23,49%
62	CiudadDeMexico	90	17	137392	0,6715%	601,6102	99714	109573	8,9977%	-27,42%	-20,25%	103658	-24,55%	109332	-20,42%
Avg.					1,2269%				5,1683%	-22,06%	-17,64%		-20,42%		-17,26%

		VNS					B&C					DR		TS	
original_idx	city	stations_count	vehicles_capacity	local_optimum_value	avg_percentage_gap	avg_execution_time	local_optimum_LB	local_optimum_UB	LB_UB_gap	vns_gap_LB	vns_gap_UB	local_optimum	vns_gap	local_optimum	vns_gap
42	Denver	116	30	168861	1,3156%	4045,9156	136148	137843	1,2297%	-19,37%	-18,37%	138467	-18,00%	156620	-7,25%
43	Denver	116	20	204185	0,0000%	1800,9386	157736	186449	15,3999%	-22,75%	-8,69%	166150	-18,63%	175905	-13,85%
44	Denver	116	10	296617	0,5153%	1803,1578	246133	298886	17,6499%	-17,02%	0,76%	262936	-11,36%	279382	-5,81%
Avg.					0,6103%				11,4265%	-19,71%	-8,76%		-15,99%		-8,97%

Tabella 2: risultati ottenuti sulle istanze *medium* ($50 < |V| < 100$) e *large* ($|V| \geq 100$)

Bibliografia

[1] Dell'Amico, Mauro & Hadjiconstantinou, Eleni & Iori, Manuel & Novellani, Stefano. (2013). The bike sharing rebalancing problem: Mathematical formulations and benchmark instances. Omega. 45. 10.1016/j.omega.2013.12.001.

[2] Dell'Amico, Mauro & Iori, Manuel & Novellani, Stefano & Stützle, Thomas. (2016). A Destroy and Repair Algorithm for the Bike sharing Rebalancing Problem. Computers & Operations Research. 71. 10.1016/j.cor.2016.01.011.

[3] Pan, Lijun & Liu, Ximei & Xia, Yangkun & Xing, Lining. (2020). Tabu Search Algorithm for the Bike Sharing Rebalancing Problem. IEEE Access. PP. 1-1. 10.1109/ACCESS.2020.3011844.

[4] Clarke, Geoff & Wright, J.V.. (1964). Scheduling of Vehicles From a Central Depot to a Number of Delivery Points. Operations Research. 12. 568-581. 10.1287/opre.12.4.568.

[5] http://www.or.unimore.it/site/home/online-resources/bike-sharing-rebalancing-problems/documento1090044445.html