Alma Mater Studiorum
Università di Bologna · Sede di Cesena

SCUOLA DI SCIENZE

Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

The home health care routing and scheduling problem with interdependent services

Relatore:
Chiar.mo Prof.
Vittorio Maniezzo

Presentata da:
Francesco Cassano

Sessione Luglio 2024
Anno Accademico 2022/2023

Ai miei genitori,
per avermi sempre incoraggiato a perseguire i miei sogni
e per il loro instancabile supporto.

Sommario

In questa tesi verrà discussa l'efficacia di un modello per la pianificazione giornaliera di interventi di assistenza domiciliare nel settore sanitario, condotti dal personale messo a disposizione da un'azienda sanitaria, presumibilmente di natura statale. L'attenzione è giustificata dall'importanza ormai consolidata delle cure domiciliari, fondamentali per garantire un'assistenza personalizzata e continua nel confort del proprio domicilio, migliorando la qualità della vita del paziente e alleggerendo il carico sui servizi ospedalieri. Tuttavia, la pianificazione e il coordinamento degli interventi di assistenza domiciliare presentano delle complessità considerevoli, evidenti in particolar modo quando si devono gestire interventi interdipendenti, pratiche che impongono vincoli temporali rigorosi. Inoltre, occorre considerare i servizi richiesti dai pazienti, programmandoli entro finestre temporali specifiche, dei membri dello staff, i caregiver, che possono erogarli e di quelli che non sono compatibili ad un specifica richiesta del paziente.

L'obiettivo della mia tesi è sviluppare un programma capace di fornire un itinerario preciso e affidabile, ottimizzando l'efficienza delle prestazioni di cura da parte dei caregivers. L'algoritmo deve essere in grado di risolvere in modo efficiente problemi di routing e scheduling, prestando attenzione ai vincoli di interdipendenza su un sottoinsieme di servizi richiesti.
L'algoritmo proposto per la risoluzione di questo problema è basato su un approccio metaeuristico. I metodi metaeuristici sono algoritmi di ottimizzazione che combinano tecniche di programmazione matematica con strategie di ricerca euristica. Grazie a queste tecniche, è possibile esplorare ampie zone

dello spazio delle soluzioni in tempi accettabili.
Nello specifico, l'algoritmo adottato è Adaptive Large Neighborhood Search o ALNS, un metaeuristico noto per la sua robustezza nella risoluzione di problemi complessi come il Pickup and Delivery Problem with Time Windows (PDPTW), che presenta vincoli simili all'HHCRSP.

Nel primo capitolo, verrà fornita un'introduzione dettagliata sul procedimento di routing e scheduling e sul problema che essa affronta, esplorando l'essenza e la complessità della risoluzioni di problemi NP-hard.
Nel secondo capitolo verrà fatta un'esposizione teorica del problema, degli algoritmi euristici nel complesso e degli algoritmi utilizzati.
Nel terzo capitolo verranno discusse nel dettaglio le procedure adottate e i metodi di realizzazione del sistema.
Nel quarto capitolo verranno valutati e discussi i risultati sperimetali del sistema.
Nel quinto capitolo verranno discusse le conclusioni tratte dai risultati.

Indice

Elenco delle figure

Capitolo 1

Introduzione al problema

L'assistenza domiciliare è un servizio previsto dai Livelli Essenziali di Assistenza (LEA) e mira a soddisfare i bisogni di salute delle persone fragili, come anziani, disabili e minori non autosufficienti, che richiedono cure a domicilio. Questo servizio, sia temporaneo che a lungo termine, mira a gestire la cronicità, prevenire la disabilità e migliorare la qualità della vita.
L'obiettivo dell'assistenza domiciliare è consentire al paziente di rimanere nel proprio ambiente familiare, ricevendo le cure necessarie per la sua condizione di salute. Questo approccio si basa sull'idea che la casa sia il luogo principale di cura, dove il paziente può essere assistito da personale medico e infermieristico mantenendo il legame con il proprio contesto sociale. La collaborazione tra professionisti multidisciplinari, come medici, operatori sociali, fisioterapisti e psicologi, garantisce una continuità assistenziale e una condivisione degli obiettivi e delle responsabilità. L'assistenza domiciliare comprende diversi livelli di intervento, adattati alle esigenze specifiche del paziente e alla situazione territoriale.
Rispetto all'ospedalizzazione, l'assistenza domiciliare offre un'alternativa più confortevole e adatta alle esigenze degli anziani, dei disabili e dei pazienti affetti da malattie cronico-degenerative. L'assistenza infermieristica domiciliare si concentra sul miglioramento delle condizioni di vita del paziente e sull'interazione con l'ambiente familiare, considerando le necessità non solo

sanitarie ma anche ambientali e relazionali dell'utente.

Nel contesto della nostra ricerca, esploreremo gli algoritmi per la pianificazione e la gestione degli itinerari giornalieri dell'assistenza domiciliare, ovvero il problema del Home Health Care Routing and Scheduling Problem (HHCRSP). Utilizzeremo come base la formulazione del problema proposta da Mankowska et al. nel 2012, adattata dai ricercatori dell'Università degli Studi di Udine.
Il problema si concentra sullo staff medico, noto come caregiver, composto da dipendenti con varie qualifiche, tra cui medici, terapisti, infermieri, assistenti sociali e corrieri alimentari.
Lo staff non subisce stravolgimenti basate sulla qualifica, poiché le informazioni associate, ovvero i servizi erogabili, ad essere determinanti. Si presume che ogni membro dello staff abbia la capacità di spostarsi autonomamente.
I pazienti richiedono servizi specifici che possono essere forniti solo dal personale autorizzato, preferibilmente durante determinate finestre temporali.
I servizi si dividono in prestazioni singole o interdipendenti. Le prestazioni interdipendenti possono essere ulteriormente suddivise in sincrone e sequenziali, introducendo quindi rigidi vincoli sulla sincronizzazione.

Il problema presenta somiglianze con il noto problema del commesso viaggiatore multiplo con finestre temporali (mTSPTW), ma si differenzia in diversi aspetti. Innanzitutto, nell'HHCRSP vi sono clienti che devono essere visitati più di una volta. In secondo luogo, i caregiver possiedono competenze e qualifiche diverse. In terzo luogo, è necessario considerare le interdipendenze temporali dei doppi servizi, richiedendo una sincronizzazione attenta dei piani di lavoro del personale.

1.1 Introduzione al HHCRSP

Il problema dell'Home Health Care (HHC) presenta una serie di sfide complesse, soprattutto nell'ottimizzazione a diversi livelli decisionali, come lo scheduling dei turni, l'assegnazione dello staff e la pianificazione dei percorsi. In molti casi, è necessario assegnare un insieme di infermieri eterogenei a clienti eterogenei distribuiti sul territorio operativo. Questo richiede la considerazione di vari requisiti, come la corrispondenza tra le competenze degli infermieri e i requisiti dei pazienti, il rispetto delle preferenze, le normative vigenti e le complicazioni reali dei servizi HHC, come la continuità delle cure o il bilanciamento del carico di lavoro. Inoltre, i servizi HHC sono spesso sensibili al fattore tempo; ad esempio, le iniezioni di insulina devono essere somministrate entro intervalli specifici, complicando ulteriormente le operazioni e richiedendo l'aggiunta di vincoli specifici. Di solito, un'infermiera viene assegnata a più clienti che possono richiedere diversi servizi in un dato periodo di tempo. Per evitare l'introduzione di nuovi vincoli, è preferibile pianificare i percorsi su base giornaliera.

Per risolvere i problemi legati all'Home Health Care (HHCRSP) è fondamentale rispettare una serie di vincoli che derivano dalle esigenze del dominio. Distinguiamo in una serie di vincoli **deboli** e vincoli **forti**. I vincoli deboli sono definiti così in quanto possono essere violati ma la cui violazione può influire più o meno significativamente sul risultato della funzione di costo, la quale in questo caso dovrebbe essere minimizzata. Mentre, i vincoli forti non possono assolutamente essere violati.
Le informazioni su cui HHCRSP richiede particolare attenzione includono:

- Il tempo trascorso dai caregiver nei viaggi.
- Il tempo di attesa dei caregiver prima di poter erogare le cure.
- I ritardi subiti dai pazienti.
- Il massimo ritardo registrato.
- La massima attesa registrata.

Le informazioni che hanno un particolare peso sull'algoritmo sono il tempo totale di viaggio, il ritardo subito dai pazienti e il ritardo più alto registrato che vengono usati come variabili nella funzione di costo.

Dal punto di vista della struttura, il problema è organizzato come un **grafo non orientato** in cui i nodi rappresentano i pazienti o le strutture a cui fanno capo i caregiver, chiamati depositi, e gli archi rappresentano le distanze tra di loro. Le distanze sono memorizzate in una matrice NxN e sono espresse in minuti, N indica il numero di pazienti più il numero di depositi presenti nell'istanza del problema.

I **pazienti** hanno la struttura più complessa dato che devono contenere gran parte delle informazioni utili, ovvero:

- La finestra temporale in cui il paziente può essere visitato.
- I servizi richiesti.
- Eventuali riferimenti alla sincronicità.
- I caregivers non graditi.

La finestra temporale è un vincolo sia forte che debole: la prestazione del servizio non può iniziare prima dell'apertura della finestra (vincolo forte) e dovrebbe iniziare prima della chiusura della finestra (vincolo debole), altrimenti nel primo caso la soluzione non sarebbe valida mentre nel secondo ne aumenterebbe il costo.
I servizi richiesti sono un vincolo forte che esclude alcuni dei caregiver che non possono offrire quel servizio.
Le informazioni di sincronocità sono un vincolo forte che serve a proteggere le tempistiche richieste da alcune terapie. Il vincolo impone la simultaneità o la sequenzialità fornendo la finestra temporale entro cui fornire il servizio successivo.
I caregivers non graditi sono route da escludere a priori nel routing se non graditi al paziente.

I **depositi** sono nodi particolari poiché costituiscono i punti di inizio e fine dei percorsi, ma come tutti gli altri nodi, sono collegati al resto del grafo. Di base la loro struttura non contiene informazioni fondamentali per la risoluzione dell'algoritmo.

I **caregivers** sono il fondamento delle route. La struttura dei caregiver incapsula diverse informazioni ma quelle fondamentali sono:

- ID.
- Abilità.

L'id viene utilizzato per distinguere le route, invece, le abilità del caregiver indicano i servizi che egli può offrire. Ad ogni route corrisponde un caregiver e viceversa, di conseguenza ogni route mette a disposizione solo i servizi offerti dal proprio caregiver. Anche questo è un vincolo forte

I **servizi** forniscono informazioni generiche sui servizi come ad esempio la durata del servizio se non fosse compresa nelle necessità del paziente.

Per trovare la soluzione migliore si cerca di minimizzare una **funzione di costo** che permetta di navigare nello spazio delle soluzioni e trovare un punto di minimo. Trattandosi di un algoritmo euristico, il minimo non è detto che sia assoluto, potrebbe essere locale.

1.2 Come si passa da HHCRSP a ALNS

Come Mankowska et al. fanno notare in [1], il problema di pianificazione considerato in questo documento è quello di instradare i membri del personale e programmare le operazioni di servizio singolo e doppio per l'HHC (Home Health Care) in base alle esigenze di servizio individuali di un dato gruppo di pazienti. Il problema differisce dal noto problema del commesso viaggiatore multiplo con finestre temporali (mTSPTW) nei seguenti aspetti: in primo luogo, nell'HHCRSP esistono clienti che devono essere visitati più di una volta. In secondo luogo, i caregiver possiedono competenze e qualifiche diverse. In terzo luogo, devono essere prese in considerazione le interdipendenze temporali dei doppi servizi. Quest'ultimo richiede un'attenta sincronizzazione dei piani di lavoro interdipendenti del personale.
Del resto, entrambi gli algoritmi hanno come obiettivo visitare tutti i nodi del grafo, fornendo a tutti i nodi i servizi richiesti utilizzando operatori multipli. Date queste caratteristiche, viene spontaneo notare la somiglianza tra HHCRSP e un altro noto problema: il Vehicle Routing Problem with Time Windows and Pairwise Synchronization (VRPTWPS) [2]. Anche in questo caso le differenze che dividono i due problemi sono minime e risiedono principalmente in due punti. In HHCRSP, il nodo di partenza del caregiver è lo stesso nodo di arrivo, e i caregiver non trasportano una quantità di beni tale da doversi rifornire prima di servire altri pazienti.
Nonostante ciò, avendo molti punti di contatto, è possibile pensare a una soluzione simile come l'algoritmo ALNS (Adaptive Large Neighborhood Search). ALNS si è dimostrato un valido algoritmo di ottimizzazione migliorando di molto le migliori soluzioni dei problemi di VRPTWPS su cui è stato applicato. La sua forza computazionale sta nell'incorporare diversi algoritmi euristici anziché usarne uno. Gli algoritmi sono divisi in due categorie: gli algoritmi di destroy e quelli di repair. ALNS sceglie la coppia con un meccanismo a ruota di roulette, dove gli algoritmi che sono riusciti a dare i migliori risultati hanno maggiori probabilit à di essere selezionati al ciclo di iterazioni successivo.

1.3 Definizione del problema

Per la definizione formale del problema si propone il modeling prodotto da Mankowska et al. in [1], una formulazione matematica.
HHCRSP è formato da un insieme di pazienti C , un insieme di tipologie di servizi S e l'insieme dei caregiver V. Una matrice sparsa identifica le abilità possedute dai caregivers a_{vs} con valori binari che identificano se è un tipo di abilità $s \in S$ è posseduto da un caregiver $v \in V$. Ogni route ha un solo caregiver v e parte e deve ritornare allo stesso deposito o ufficio centrale /. L'insieme che individua tutti i nodi è definito come $C^0 = C \cup \{0\}$. I requisiti di servizio di ciascun paziente $i \in C$ sono indicato dal parametro r_{is}, con $r_{is} = 1$ se il paziente i lo richiede un servizio di tipo s e 0 altrimenti.
Inoltre, per i pazienti con doppio servizio $i \in C^d$, una distanza temporale minima δ_i^{min} ed è specificata una distanza temporale massima δ_i^{max} tra il primo ed il secondo servizio. Se $\delta_i^{min} = \delta_i^{max} = 0$, entrambe le operazioni di servizio devono iniziare contemporaneamente. Altrimenti, per $\delta_i^{min} > 0$, la seconda operazione di servizio deve iniziare almeno δ_i^{min} unit à di tempo dopo l'inizio della prima operazione di servizio. Inoltre, il secondo servizio non deve iniziare oltre le unità di tempo δ_i^{max} successive al primo servizio $\delta_i^{max} \geq \delta_i^{min}$. È richiesto che la distanza temporale minima e massima tra le due operazioni in un doppio servizio vengono sempre rispettate.
Dati i vincoli esistenti sui tempi di attesa per i servizi successivi si possono definire due sottoinsiemi di C^d: C^{sim} l'insieme dei pazienti che richiedono un servizio doppio simultaneo dove $\delta_i^{min} = \delta_i^{max} = 0$ e C^{prec} l'insieme dei pazienti che richiedono un servizio doppio con precedenza se $0 < delta_i^{min} < \delta_i^{max}$. La finestra di disponibilità, definita come $[e_i, l_i]$. e_i è il tempo di apertura della finestra, un vincolo forte che impone al servizio richiesto da i di non iniziare prima e_i. Nel caso in cui il caregiver dovesse arrivare prima aspetterà. l_i è il tempo di chiusura della finestra, un vincolo debole la cui violazione produrrà un ritardo z_{is} calcolato come $z_{is} = t_{ivs} - l_i$.
Il routing del personale è modellato da variabili decisionali binarie x_{ijvs}, che assumono valore 1 se il caregiver $v \in V$ si sposta direttamente da $i \in C^0$ a

$j \in C^0$ per fornire l'operazione di servizio $s \in S$ al paziente j, e 0 altrimenti. La variabile di schedulazione t_{ivs} denota l'orario di inizio dell'operazione di servizio $s \in S$ presso il paziente $i \in C$ se eseguita dal membro del personale $v \in V$.
Per misurare le performance dell'algoritmo viene usata una funzione di costo Z definita su tre variabili:

- D la misura totale viaggiata da tutti i caregivers.

$$D = \sum_{v \in V} \sum_{i \in C^0} \sum_{i \in C^0} \sum_{s \in S} d_{ij} \cdot x_{ijvs} \tag{1}$$

- T la misura totale dei ritardi dei servizi che sono andato oltre la finestra temporale.

$$T = \sum_{v \in V} \sum_{s \in S} z_{is} \tag{2}$$

- T^{max} la misura del ritardo più alto misurato.

$$T^{max} \geq z_{is} \tag{3}$$

L'idea alla base di questa misurazione sta nel migliorare gradualmente la soluzione in modo da ridurre i ritardi e il tempo viaggiato ma allo stesso tempo evitare una situazione in cui pochi pazienti accumulano un notevole ritardo a favore del resto dei pazienti. La funzione di costo è quindi così definita:

$$minimizzare \rightarrow Z = \lambda_1 D + \lambda_2 T + \lambda_3 T^{max} \tag{4}$$

Pi ù precisamente, se un HCC considera rilevante solo uno degli obiettivi, gli altri vengono ignorati impostando i valori di λ a 0. È possibile esprimere una gerarchia tra gli obiettivi impostando valori significativamente diversi. Ad esempio, impostando λ_1 su un valore molto alto e λ_2 e λ_3 su valori molto pi ù bassi, si d à priorit à alla minimizzazione dei tempi di viaggio. In questa situazione, tra tutte le soluzioni con minimo sforzo di viaggio, verr à scelta quella con il minor ritardo nei servizi. Infine, si raggiunge un compromesso economico tra costo e qualit à del servizio se i valori di λ sono impostati

sul tasso di costo per unit à di distanza percorsa da un caregiver e sui tassi di penalit à per ogni unit à di tempo di ritardo. In questo caso, i tassi di costo particolari devono essere specificati dall'HCC. Di base i valori di λ della funzione di costo sono impostati a $\lambda_1 = \lambda_2 = \lambda_3 = 1/3$.

1.4 Definizione dei Vincoli

Se i vincoli deboli sono stati individuati e integrati nella funzione di costo, i vincoli forti devono essere definiti esplicitamente e chiaramente, poichè determinano la validità di una soluzione specifica.
Di seguito verranno elencati i vincoli forti, ciascuno con una breve illustrazione della loro funzione:
Il vincolo (5) garantisce che il percorso di ogni caregiver inizia e finisca nell'ufficio centrale.

$$\sum_{i \in C} \sum_{s \in S} x_{0ivs} = \sum_{i \in C^0} \sum_{s \in S} x_{i0vs} = 1 \quad \forall v \in V \tag{5}$$

Il vincolo (6) impone che ogni caregiver non ripassi in un nodo in cui è già stato.

$$\sum_{j \in C} \sum_{s \in S} x_{jivs} = \sum_{j \in C^0} \sum_{s \in S} x_{ijvs} \quad \forall i \in C, v \in V \tag{6}$$

Il vincolo (7) assicura che ogni richiesta venga eseguita soltanto da un caregiver.

$$\sum_{v \in V} \sum_{j \in C} a_{ij} \cdot x_{jivs} = r_{is} \quad \forall i \in C, s \in S \tag{7}$$

Il vincolo (8) determina gli orari di inizio delle operazioni di servizio rispetto alla durata del servizio e ai tempi di viaggio. Questi vincoli garantiscono che gli orari di inizio dei servizi lungo il percorso di un membro del personale siano strettamente crescenti. In questo modo, essi impediscono anche la formazione di cicli nei percorsi, poiché il ritorno a un paziente già visitato violerebbe l'orario di inizio della visita precedente.

$$t_{ivs_1} + p_{is_1} + d_{ij} \leq t_{jvs_2} + M(1 - x_{ijvs_2}), \quad \forall i \in C^0, j \in C, v \in V, s_1, s_2 \in S \tag{8}$$

I vincoli (9) e (10) determinano il rispetto degli orari di inizio dei servizi rispetto alle finestre temporali. L'orario di inizio e_i impone un vincolo rigido, mentre è possibile superare l'orario di fine, cosa che viene espressa da un ritardo.

$$t_{ivs} \geq e_i \quad \forall i \in C, v \in V, s \in S \tag{9}$$

$$t_{ivs} \leq l_i + z_{is} \quad \forall i \in C, v \in V, s \in S \tag{10}$$

L'interdipendenza temporale dei servizi doppi si riflette nelle formule (11) (12). La distanza minima tra l'inizio dei due servizi doppi è garantita da (11)

$$t_{iv_2s_2} - t_{iv_1s_1} \geq \delta_i^{min} - M \left(2 - \sum_{j \in C^0} x_{jiv_1s_1} - \sum_{j \in C^0} x_{jiv_2s_2} \right)$$

$$\forall i \in C, \forall v_1 \in V, \forall s_1 \in S : s_1 < s_2 \tag{11}$$

mentre quella massima da (12)

$$t_{iv_2s_2} - t_{iv_1s_1} \leq \delta_i^{max} + M \left(2 - \sum_{j \in C^0} x_{jiv_1s_1} - \sum_{j \in C^0} x_{jiv_2s_2} \right)$$

$$\forall i \in C, \forall v_1 \in V, \forall s_1 \in S : s_1 < s_2 \tag{12}$$

Con il vincolo (13) si specifica che $a_{vs} \cdot r_{js} = 1$, se e solo se il membro del personale $v \in V$ è qualificato a svolgere un servizio $s \in S$ richiesto da un paziente $j \in C$, e 0 altrimenti. Questa restrizione elimina le variabili binarie superflue per i pazienti i cui requisiti di servizio sono incompatibili con la qualifica del caregiver.

$$x_{ijvs} \in 0, a_{vs} \cdot r_{js}, \quad \forall i, j \in C^0, v \in V, s \in S \tag{13}$$

Il vincolo (14) la non negatività definisce il dominio per le variabili di pianificazione del ritardo.

$$t_{ivs}, z_{is} \geq 0 \quad \forall i \in C^0, v \in V, s \in S \tag{14}$$

I seguenti vincoli illustrano come determinare il tempo t all'arrivo al nodo j del cargiver v per il servizio s nelle varie situazioni che si possono verificare. Il vincolo (15) calcola il tempo di un normale servizio singolo.

$$t_{jvs} = \max\{e_j; b_{jv}\} \tag{15}$$

Il vincolo (16) calcola il tempo di un servizio doppio con servizio doppio e contemporaneo.

$$t_{jv_1s_1} = t_{jv_2s_2} = \max\{e_j, b_{jv_1}, b_{jv_2}\} \tag{16}$$

I vincoli (17) (18) definiscono il tempo per un servizio doppio e sequenziale: (17) per il primo e (18) per il secondo.

$$t_{jv_1s_1} = \max\{e_j; b_{jv_1}\} \tag{17}$$

$$t_{jv_2s_2} = \max\{t_{jv_1s_1} + \delta_j^{min}; b_{jv_2}) \tag{18}$$

Il vincolo (19) impone che, in caso di ritardo nel secondo servizio, sarà l'esecuzione del primo servizio a essere posticipata. Per minimizzare il ritardo e ridurre il rischio di compromettere altri percorsi, il nuovo tempo sarà calcolato sottraendo δ_j^{max}dal tempo di arrivo del secondo nodo.

$$t_{jv_1s_1} = t_{jv_2s_2} - \delta_j^{max} \tag{19}$$

Capitolo 2

Stato dell'arte

I problemi principale dell'assistenza sanitaria domiciliare sono migliorare la qualità del servizio e ridurre i costi. Questo perché c'è una crescente domanda di servizi HCC a costi convenienti. Molti fornitori di HCC operano a scopo di lucro e devono essere competitivi sul mercato, mentre, i servizi HCC rappresentano una parte significativa della spesa sanitaria pubblica e la riduzione dei costi in questo settore è di grande interesse pubblico. Per questo motivo, l'Home Health Care (HHC) è stato un ambito di ricerca molto attivo fin dagli anni '90.

I pianificatori HHC affrontano problemi di ottimizzazione complessi e impegnativi su diversi livelli decisionali, come la pianificazione dei turni, l'assegnazione del personale e le decisioni sul routing del personale. Nella maggior parte dei casi, un insieme di infermieri eterogenei deve essere assegnato a clienti eterogenei, distribuiti sul territorio operativo. Pertanto, è necessario considerare potenzialmente vari requisiti, come la corrispondenza tra le competenze degli infermieri e le richieste dei clienti, il rispetto delle preferenze, le varie normative e ulteriori complicazioni reali dei servizi HHC come la continuità delle cure o le misurazioni dell'equilibrio del carico di lavoro. Inoltre, i servizi HHC sono spesso sensibili al fattore tempo, ad esempio le iniezioni di insulina devono essere somministrate entro determinati intervalli di tempo piuttosto regolari, il che complica ulteriormente le operazioni.

In genere, un operatore viene assegnato a più clienti che possono richiedere più servizi in un determinato periodo di tempo. Per un determinato periodo di pianificazione che va da un singolo giorno a più mesi, è necessario prendere decisioni riguardanti l'ordine e gli orari in cui visitare i clienti. Ciò influenza le distanze di viaggio, i tempi di lavoro e la qualità del servizio. Per pianificare i percorsi di viaggio e raggiungere i clienti, l'operatore può utilizzare vari modi di trasporto, tra cui auto, biciclette, trasporti pubblici o spostarsi a piedi[5].

La maggior parte degli articoli di ricerca su HHC si concentra su problemi di ottimizzazione di un singolo periodo, assumendo come orizzonte di pianificazione un singolo giorno lavorativo. Le ricerche svolte spesso mostrano sostanziali differenze nella formulazione, dovute principalmente ai diversi contesti nazionali e normativi. Tuttavia, le strategie utilizzate per affrontare questi problemi si dividono sostanzialmente in due categorie: algoritmi metaeuristici e programmazione lineare [5]. Entrambi gli approcci sono utilizzati per risolvere problemi di ottimizzazione, ma presentano differenze significative in termini di metodologie, applicazioni e tipi di problemi per cui sono più adatti. Nel caso esaminato in questa tesi si esploreranno i vari aspetti di un approccio metaeuristico, tipologia a cui ALNS appartiene.

2.1 Complessità dei Problemi e Utilizzo degli Algoritmi Euristici

Di solito, gli algoritmi euristici e metaeuristici sono utilizzati per problemi complessi che non possono essere risolti facilmente. Per comprendere la complessità di questi problemi, possiamo classificarli in base a due categorie principali: **P** e **NP**.
La classe **P** include tutti quei problemi che possono essere risolti in tempo polinomiale su una macchina di Turing deterministica.
D'altra parte, la classe **NP** comprende problemi la cui soluzione può essere trovata in tempo polinomiale su una macchina di Turing non deterministica.

In pratica, questo significa che per un problema NP, se esiste una soluzione, può essere verificata in tempo polinomiale, ma non necessariamente trovata in tempo polinomiale.
Un sottoinsieme importante di NP è la classe **NP-completo**, che include problemi per i quali non è noto alcun algoritmo polinomiale esatto, ma se un problema NP-completo può essere risolto in tempo polinomiale, allora tutti i problemi NP possono essere risolti in questo modo. Infine, la classe **NP-hard** include problemi almeno così difficili come quelli NP-completi, ma non necessariamente appartenenti alla classe NP. In altre parole, i problemi NP-hard possono essere più difficili di quelli NP-completi.
L'uso di algoritmi euristici è giustificato quando si tratta di problemi NP-completi o NP-hard, per i quali non sono noti algoritmi polinomiali esatti. In queste situazioni, le euristiche sono sviluppate per trovare soluzioni accettabili in tempi ragionevoli per input di dimensioni realistiche[6].

2.2 Gli algoritmi euristici

Gli algoritmi euristici sono tecniche di risoluzione dei problemi progettate per trovare soluzioni soddisfacenti in tempi ragionevoli per problemi complessi, dove trovare una soluzione ottimale esatta potrebbe essere troppo dispendioso. Questi algoritmi sfruttano conoscenze specifiche del problema per guidare la ricerca verso soluzioni *promettenti*, senza garantire però che la soluzione trovata sia la migliore possibile.
Per cominciare, riassumiamo i principi principali algoritmi di ricerca tradizionali.
Il più semplice degli algoritmi di ricerca è la **ricerca esaustiva**, che prova tutte le possibili soluzioni da un insieme predeterminato e successivamente sceglie la migliore. La **ricerca locale** è una versione della ricerca esaustiva che si concentra solo su un'area limitata dello spazio di ricerca. La ricerca locale può essere organizzata in diversi modi. Le più diffuse tecniche di **hill-climbing** appartengono a questa classe. Tali algoritmi sostituiscono

coerentemente la soluzione corrente con la migliore tra quelle vicine, se questa è migliore di quella corrente. Ad esempio, l'euristica per il problema della replica intragruppo per il servizio di distribuzione multimediale basato sulla rete Peer-to-Peer si basa sulla strategia hill-climbing .
Gli algoritmi **divide et impera** cercano di suddividere un problema in problemi più piccoli, più facili da risolvere. Le soluzioni dei piccoli problemi devono essere combinabili con la soluzione del problema originale. Questa tecnica è efficace, ma il suo uso è limitato perché non esiste un gran numero di problemi che possano essere facilmente suddivisi e combinati in questo modo.
La tecnica **Branch-and-bound** è un'enumerazione critica dello spazio di ricerca. Enumera, ma cerca costantemente di escludere le parti dello spazio di ricerca che non possono contenere la soluzione migliore.
La **programmazione dinamica** è una ricerca esaustiva che evita di ricompilare i calcoli memorizzando le soluzioni dei sottoproblemi. Il punto chiave per l'utilizzo di questa tecnica è la formulazione del processo di soluzione come una ricorsione.
Un metodo popolare per costruire successivamente uno spazio di soluzioni è la tecnica **greedy**, che si basa sull'evidente principio di prendere la scelta migliore (locale) in ogni fase dell'algoritmo per trovare l'ottimo globale di una funzione obiettivo.
Per definire un algoritmo di ricerca locale, è necessario specificare diversi elementi:

- *Generazione della soluzione iniziale*: creare una soluzione iniziale inizializza il processo di ricerca. La soluzione iniziale può essere fatta in modo casuale o utilizzando un algortimo euristico.

- *Operatore di Movimento*: Definire una funzione che genera un insieme di soluzioni vicine (vicinato) a partire dalla soluzione corrente. Questo operatore descrive come passare da una soluzione all'altra.

- *Criterio di accettazione*: Definire una funzione di valutazione (o fun-

zione obiettivo) per misurare la qualità di una soluzione e che la rifiuti in caso non sia valida.

- *Criterio di Arresto*: Definire quando fermare l'algoritmo. Alcuni criteri di arresto comuni includono:
  - Un numero massimo di iterazioni.
  - Una soglia di miglioramento (se il miglioramento della soluzione è inferiore a un certo valore).
  - Un tempo massimo di esecuzione

Un insieme di vicinanza può essere generato a partire da una soluzione iniziale cambiando un sottoinsieme di componenti della soluzione stessa. Nel caso dei problemi di routing, questo può significare, ad esempio, eseguire uno scambio di archi o di vertici. La ricombinazione di questi componenti è detta "mossa". Attraverso una sequenza di mosse, è possibile esplorare l'insieme di vicinanza di una data soluzione.
Una volta definite le mosse per esplorare l'insieme di vicinanza, è necessario stabilire un criterio per valutare se le soluzioni trovate in questo insieme sono migliori rispetto alla soluzione di partenza. Se una soluzione nel vicinato risulta migliore rispetto alla soluzione corrente, viene accettata e la ricerca continua partendo da questa nuova soluzione.
Esistono due strategie di accettazione delle soluzioni:

- First-accept (FA): Accetta la prima soluzione migliore trovata durante l'analisi del vicinato.
- Best-accept (BA): Esamina interamente l'insieme di vicinanza e seleziona successivamente la soluzione migliore al suo interno.

Oltre agli algoritmi di ricerca locale, ci sono anche i metodi costruttivi, una famiglia di algoritmi euristici che costruiscono una soluzione ammissibile al problema trattato. Questi algoritmi aggiungono iterativamente elementi alla soluzione fino a completarla. I metodi costruttivi utilizzano un criterio di espansione per determinare, durante ogni iterazione, quale elemento

aggiungere alla soluzione. Questo criterio può basarsi su un ordinamento preliminare degli elementi o su una regola di ottimo locale.

2.3 Gli algoritmi metaeuristici

Le metaeuristiche sono strutture algoritmiche generali, spesso ispirate alla natura, progettate per risolvere problemi di ottimizzazione complessi. Questi algoritmi rappresentano un'area di ricerca in crescita da alcuni decenni e negli ultimi anni stanno emergendo come alternative di successo agli approcci più classici per risolvere problemi di ottimizzazione che includono nella loro formulazione matematica informazioni incerte, stocastiche e dinamiche [8]. Le metaeuristiche campionano un sottoinsieme di soluzioni che sarebbe troppo grande per essere completamente enumerato o esplorato in altro modo. A differenza degli algoritmi tradizionali, le metaeuristiche possono fare relativamente poche ipotesi sulla risoluzione del problema di ottimizzazione, risultando quindi utilizzabili per una varietà di problemi [7]. Come le euristiche, anche le metaeuristiche non garantiscono che la soluzione converga ad un ottimo globale. Tuttavia, rispetto alle euristiche tradizionali, le metaeuristiche riducono il rischio di rimanere bloccati in ottimi locali grazie all'implementazione di tecniche avanzate. Molte metaeuristiche incorporano una forma di ottimizzazione stocastica, in modo che la soluzione trovata dipenda dall'insieme di variabili casuali generate. Nell'ottimizzazione combinatoria, ricercando un ampio insieme di soluzioni fattibili, le metaeuristiche possono spesso trovare buone soluzioni con meno sforzo computazionale rispetto agli algoritmi di ottimizzazione esatti, ai metodi iterativi o alle semplici euristiche [8]. La maggior parte della letteratura sulle metaeuristiche è di natura sperimentale e descrive risultati empirici basati su esperimenti informatici con gli algoritmi. Tuttavia, sono disponibili anche alcuni risultati teorici formali, spesso riguardanti la convergenza e la possibilità di trovare l'ottimo globale [7]. L'approccio metaeuristico per risolvere problemi di ottimizzazione consiste nel costruire una o più soluzioni iniziali e, quindi, nell'utilizzare

le soluzioni individuate come punto di partenza per la ricerca guidata nello spazio delle soluzioni.
Alcuni esempi di algoritmi metaeuristici includono:

- **Simulated Annealing**: Ispirato al processo di ricottura dei metalli, questo algoritmo permette il peggioramento temporaneo delle soluzioni per sfuggire ai minimi locali, riducendo gradualmente la probabilità di accettare soluzioni peggiori nel tempo.

- **Tabu Search**: Questo algoritmo utilizza una lista di soluzioni vietate (tabu) per evitare di riesaminare soluzioni già esplorate, incorporando memoria a breve termine per guidare la ricerca e prevenire cicli.

- **Algoritmi Genetici**: Ispirati alla selezione naturale, questi algoritmi utilizzano operazioni di crossover e mutazione per generare nuove soluzioni a partire da una popolazione di soluzioni esistenti, favorendo quelle con fitness migliore.

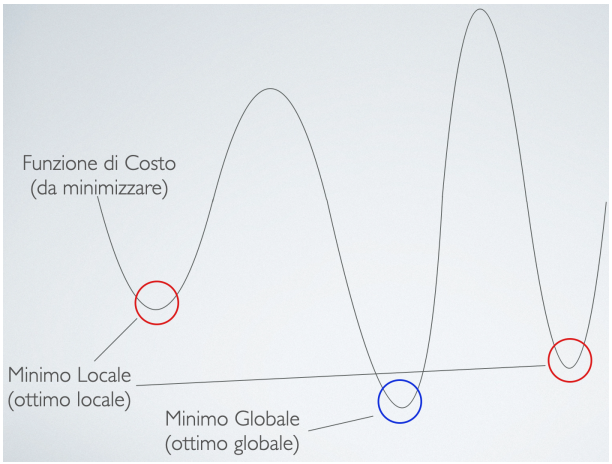

Figura 2.1: Ottimi locali e globali

2.4 Gli algoritmi utilizzati

Questa sezione esplora gli algoritmi euristici e metaeuristici, essenziali nel campo dell'ottimizzazione combinatoria per affrontare problemi complessi senza soluzioni esatte efficienti. Gli algoritmi euristici offrono soluzioni approssimative rapide, ideali per stabilire una base iniziale di lavoro. Le metaeuristiche, invece, utilizzano approcci più avanzati e ispirati alla natura per migliorare progressivamente le soluzioni, superando i limiti degli algoritmi euristici tradizionali.

2.4.1 Costruzione della soluzione iniziale

Un possibile instradamento iniziale viene costruito mediante la seguente euristica. Innanzitutto, i pazienti vengono ordinati in base alla fine crescente della loro finestra temporale. Ciò dà priorità ai servizi urgenti nel successivo processo di costruzione iterativo. Successivamente, l'algoritmo assegna ciascun paziente al percorso di uno o due caregiver a seconda delle esigenze di servizio del paziente considerato. Se il paziente considerato necessita di un unico servizio, l'algoritmo seleziona il membro del personale che può arrivare prima a questo paziente tra tutti i dipendenti adeguatamente qualificati. Se il paziente considerato necessita di un doppio servizio, vengono selezionati quei due caregiver adeguatamente qualificati che possono arrivare prima al paziente. Questa procedura viene ripetuta finché tutti i pazienti non vengono assegnati ai percorsi [1]. La procedura descritta è un esempio di algoritmo euristico greedy, poiché prende decisioni localmente ottimali ad ogni passo, senza considerare possibili conseguenze a lungo termine o alternative migliori che potrebbero emergere più avanti nel processo di assegnazione dei pazienti ai caregiver.

2.4.2 Programmazione dinamica

La programmazione dinamica è una tecnica di ottimizzazione algoritmica molto efficace per risolvere problemi complessi come il Problema del Commes-

so Viaggiatore (TSP). Il TSP prevede di trovare il percorso di costo minimo che visita un insieme di città una sola volta ciascuna, tornando infine alla città di partenza. La programmazione dinamica affronta questo problema suddividendolo in sottoproblemi più piccoli e memorizzando le soluzioni intermedie per evitare calcoli ripetuti.
Nel sistema prodotto l'algoritmo di programmazione dinamica si è occupato dello scheduling dei nodi in un solo percorso. Trattandosi sostanzialmente di un sottoinsieme del problema originale si può dire che il problema trattato dall'algoritmo sia stato un TSPTW (un problema del commesso viaggiatore con finestre temporali). La programmazione dinamica per il TSP si basa sul principio di ottimalità di Bellman, che afferma che una soluzione ottimale può essere costruita a partire da soluzioni ottimali dei suoi sottoproblemi. L'idea principale è di costruire la soluzione ottimale iterativamente, risolvendo prima i sottoproblemi più piccoli e utilizzando i loro risultati per risolvere problemi più grandi.
L'algoritmo si basa su due concetti: **stato** e **transizione**. Ogni stato può essere rappresentato come una coppia (S, i), dove S è l'insieme dei nodi visitati e i è il nodo corrente. La transizione implica calcolare il costo minimo per raggiungere un nodo j non ancora visitata partendo dal nodo i, aggiungendo j all'insieme dei nodi visitati.
La formula ricorsiva per calcolare il costo minimo $C(S, i)$ per visitare tutti i nodi nell'insieme S terminando nel nodo i è:

$$C(S, i) = \min_{j \in S} [C(S \setminus \{i\}, j) + d(j, i)] \tag{20}$$

$C(S, i)$ rappresenta il costo minimo per visitare tutti i nodi nell'insieme S terminando nel nodo i.
S è un insieme dei nodi visitati.
i è il nodo corrente.
j è un nodo nell'insieme S.
$d(j, i)$ è la distanza tra i nodi j e i.
La condizione di base è che se l'insieme S contiene solo due nodi, il costo è semplicemente la distanza tra queste due nodi.

Per un insieme di dimensione n, consideriamo $n-2$ sottoinsiemi ciascuno di dimensione $n-1$ in modo tale che tutti i sottoinsiemi non contengano n-esimo. Utilizzando la relazione ricorsiva di cui sopra, possiamo scrivere una soluzione dinamica basata sulla programmazione. Ci sono al massimo $O(n * 2n)$ sottoproblemi e ognuno di essi richiede tempo lineare per essere risolto. Il tempo di esecuzione totale è quindi$O(n^2 * 2^n)$. La complessità temporale è molto inferiore a $O(n!)$ ma pur sempre esponenziale.Questo approccio, pur essendo significativamente più efficiente rispetto alla soluzione esaustiva, diventa impraticabile per insiemi con un numero di vertici leggermente superiore. La complessità esponenziale sia in termini di tempo che di spazio rende difficile l'applicazione di questa tecnica a problemi di dimensioni grandi. Ne consegue che se la programmazione dinamica non può risolvere problemi di grandi dimensioni servono altri algoritmi per poter convergere ad una soluzione accettabile. Stiamo parlando di algoritmi euristici per il problema del commesso viaggiatore.

2.4.3 2-opt

L'algoritmo 2-opt è una tecnica euristica semplice ed efficace utilizzata per migliorare le soluzioni iniziali del Problema del Commesso Viaggiatore (TSP). Il TSP richiede di trovare il percorso più breve che visita una serie di città una sola volta ciascuna e ritorna alla città di partenza. L'algoritmo 2-opt migliora iterativamente una soluzione iniziale scambiando coppie di archi per ridurre la lunghezza totale del percorso. L'algoritmo di 2-opt deve partire da una soluzione valida. L'idea alla base del 2-opt è quella di rimuovere due archi dal percorso e sostituirli con altri due archi in modo che il nuovo percorso sia più corto, ripetendo questo processo fino a quando non sarà possibile trovare ulteriori miglioramenti.
I passi previsti da 2-opt sono essenzialmente quattro e prevedono ((2.2)):

- Identificare gli archi: Per ogni coppia di archi non consecutivi del percorso si considera cosa accadrebbe scambiando i loro nodi di arrivo.

- Calcolo del Costo: Calcoliamo il costo del nuovo percorso risultante dallo scambio.

- Scambio: Se il nuovo percorso è più corto, effettuiamo lo scambio; altrimenti, passiamo alla prossima coppia di archi.

- Iterazione: Ripetiamo questo processo per tutte le coppie di archi fino a quando non si trovano ulteriori scambi che riducono la lunghezza del percorso.

Il procedimento viene ripetuto finché avvengono scambi, quindi fino a quando l'algoritmo rileva che non ci sono più miglioramenti possibili [10].
Algoritmi come il 2-opt sono ampiamente utilizzati per la loro facilità di implementazione e la capacità di arrivare a soluzioni ottime o accettabili in tempi relativamente brevi, con una complessità di base di $O(n^2)$.
Nonostante la sua efficienza, per istanze molto grandi del TSP, il 2-opt può comunque richiedere tempi di esecuzione significativi. In questi casi, algoritmi più avanzati come l'algoritmo di Lin-Kernighan si rivelano molto più robusti. Il Lin-Kernighan è un algoritmo euristico molto più complesso, ma è progettato per gestire un alto numero di istanze con una maggiore efficienza. Questo algoritmo utilizza una serie di mosse più sofisticate rispetto al 2-opt, come scambi multipli di archi (k-opt), per esplorare un insieme più ampio di soluzioni possibili e trovare miglioramenti più significativi.

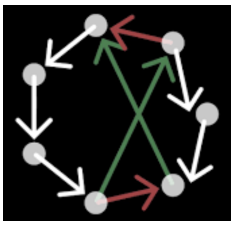

Figura 2.2: Funzionamento dell'algoritmo 2-opt

2.4.4 3-opt

Il 3-opt è un algoritmo euristico utilizzato per migliorare le soluzioni del Problema del Commesso Viaggiatore (TSP). è una generalizzazione del più semplice algoritmo 2-opt, ed è progettato per ottenere soluzioni di qualità superiore attraverso operazioni di scambio più complesse.
L'algoritmo 3-opt esegue miglioramenti locali partendo da soluzione esistente del TSP rimuovendo tre archi e ricollegando i tre segmenti risultanti in tutte le possibili configurazioni, tranne quella originale. Questo processo permette di esplorare un numero più ampio di soluzioni rispetto al 2-opt, che si limita a scambiare due archi. Le configurazioni generate vengono valutate per determinare se una nuova connessione riduce il costo totale del percorso. Se una configurazione migliore viene trovata, il percorso viene aggiornato e l'algoritmo continua il processo fino a quando non sono possibili ulteriori miglioramenti.
Il principale vantaggio del 3-opt rispetto al 2-opt è la sua capacità di ottenere soluzioni di qualità superiore, poiché esplora un insieme più ampio di possibilità di miglioramento. Tuttavia, questo aumento di qualità viene a costo di una maggiore complessità computazionale. La complessità del 3-opt è $O(n^3)$ (2.3), dove n è il numero di nodi, rendendolo più costoso in termini di tempo di esecuzione rispetto al 2-opt, che ha una complessità di $O(n^2)$. Pertanto, il 3-opt può essere meno pratico per istanze molto grandi del TSP senza ulteriori ottimizzazioni.

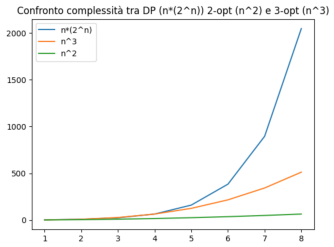

Figura 2.3: Grafico che confronta la complessità dei diversi algoritmi

2.4.5 Simulated Annealing

Il Simulated Annealing (SA) è un metodo metaeuristico ampiamente utilizzato per affrontare problemi di ottimizzazione globale in cui la funzione obiettivo non è data esplicitamente, ma può essere valutata solo tramite costose simulazioni al computer. È un algoritmo relativamente semplice da sviluppare [11].
L'algoritmo SA si basa su un'analogia con il processo di ricottura fisica dei metalli fusi. La ricottura, è un processo utilizzato per l'eliminazione di difetti reticolari dai cristalli metallici tramite riscaldamento seguito da lento raffreddamento. Nel caso dell'euristica un difetto reticolare corrisponde ad una combinazione errata di due oggetti all'interno della soluzione. In modo simile, l'algoritmo SA parte da una soluzione con alta temperatura e ne riduce gradualmente il parametro di temperatura ad ogni iterazione.
La base dell'algoritmo di ottimizzazione SA è descritta insieme a due proprietà teoriche fondamentali per SA: equilibrio statistico (ispirato alla fisica statistica elementare) e convergenza asintotica (basata sulla teoria delle catene di Markov) che ne dismostrano matematicamente l'affidabilità.
Date le sue proprietà matematiche, si può tranquillamente affermare che la peculiarità di SA risieda nella sua funzione di accettazione. Infatti, se la soluzione trovata nell'iterazione è migliore della soluzione corrente, essa può sostituire la soluzione corrente. In caso contrario, se la soluzione non è già stata esplorata, ha una probabilità pari a (21) per essere scelta come soluzione corrente. Questo implica che, più la temperatura è alta, maggiore è la probabilità che ciò accada [12].

$$P = \exp^{-\frac{\Delta_{\text{cost}}}{T_k}} \tag{21}$$

dove Δ_{cost} indica la differenza di costo tra la soluzione corrente e quella individuate all'istante k e T_k è la temperatura nello stesso istante. Questo approccio è progettato per evitare il rischio di rimanere intrappolati in minimi locali.

2.4.6 Large neighborhood search (LNS)

Negli LNS, una soluzione iniziale viene gradualmente migliorata distruggendo e riparando alternativamente la soluzione. L'euristica LNS appartiene alla classe di euristiche nota come algoritmi di ricerca di quartiere su larga scala (VLSN) [13].
Nella metaeuristica Large Neighborhood Search (LNS), il processo di miglioramento della soluzione inizia con la distruzione graduale e la successiva riparazione iterativa della soluzione corrente. Questa metodologia appartiene alla categoria degli algoritmi noti come ricerca di quartiere su larga scala (VLSN). L'intorno di una soluzione x è definito implicitamente dal metodo di distruzione e riparazione. Il metodo di distruzione rimuove una parte della soluzione attuale con un certo grado di stocasticità, mentre il metodo di riparazione ricostruisce la soluzione distrutta [13].
Ad esempio, considerando il problema del Vehicle Routing Problem with Capacities (CVRP), un metodo di distruzione potrebbe rimuovere una percentuale predeterminata di clienti dalla soluzione corrente, accorciando di conseguenza i percorsi in cui i clienti sono stati eliminati. Un metodo di distruzione semplice potrebbe selezionare casualmente i clienti da rimuovere. Il metodo di riparazione, d'altro canto, ricostruisce la soluzione reintegrando i clienti precedentemente rimossi. Questo processo può utilizzare un'euristica avida per inserire i clienti, ad esempio scegliendo di inserire prima i clienti con il costo di inserimento più basso [13].

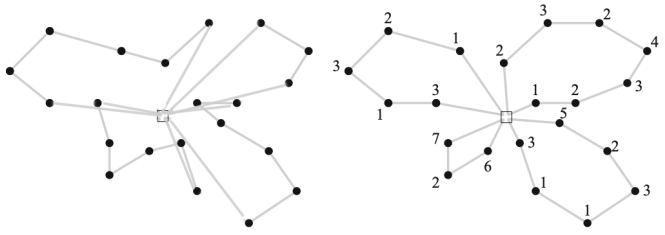

Figura 2.4: A sinistra la soluzione prima di applicare l'algoritmo, a destra una soluzione al passo i

2.4.7 Adaptive large neighborhood search (ALNS)

Una delle prime formulazioni di Adaptive Large Neighbothood search (ALNS) è stata composta da Ropke e Pisinger in [14] nel 2006. ALNS non è altro che un miglioramento della metaeuristica LNS, e costituisce un algoritmo di ricerca locale dove un insieme di algoritmi collaborano per modificare la soluzione corrente e costruirne una migliore. Come nel caso di LNS l'algoritmo si avvale di un procedimento *destroy / repair*, ovvero la soluzione corrente viene distrutta gradualmente e in seguito ricostruita in una nuova soluzione. La nuova soluzione viene accettata solo se risulta migliore della soluzione corrente; tuttavia, ereditando le dinamiche del simulated annealing, una soluzione che non è migliore, ma comunque non visitata, può comunque essere accettata con una certa probabilità, basata su un criterio di accettazione probabilistica (vedi formula (21)), che consente di evitare il blocco in minimi locali [14].
Per selezionare l'euristica da utilizzare, assegniamo pesi alle diverse euristiche e utilizziamo il principio di selezione della ruota della roulette [14]. Se abbiamo k euristiche con pesi w_i, $i \in \{1, 2, ..., k\}$, selezioniamo l'euristica j con probabilità

$$P = \frac{w_j}{\sum_{i=1}^{k} wi} \tag{22}$$

dove i pesi rappresentano quanto un operatore abbia contribuito al processo per la costruzione della soluzione. I pesi vengono aggiornati alla fine di ogni segmento che corrisponde ad un certo numero di iterazioni.
Possiamo scrivere l'algoritmo di ALNS nel seguente modo:

costruisci una soluzione ammissibile x;

poni $x* = x$

ripeti

scelta coppia destroy / repair, (d, r)

$x' = applica(d, r, x)$

se $C(x) < C(x*)$, poni $x^* = x$

se x' può essere accettata, $ponix = x'$

aggiorna i punteggi degli operatori usati

fino a che non si verificano le condizioni di termine

restituisci x^*

Capitolo 3

Metodologie applicate

LL'Home Health Care Routing and Scheduling Problem (HHCRSP) è un problema complesso che emerge da una problematica importante della assistenza sanitaria domiciliare. Il problema richiede la formulazione di percorsi e appuntamenti, talvolta sincronizzati, del personale sanitario presso un certo numero di pazienti, che giornalmente, hanno deli nodi. Il problema ha molte versioni e molte formulazioni a seconda dell'obbiettivo che viene imposto all'algoritmo di raggiungere.

Nella versione del problema studiata, è presente un unico ufficio centrale e tutte le rotte devono iniziare e concludersi in quel nodo. Per ogni percorso è presente una sola squadra dello staff sanitario (1, 2 persone) che mettono a disposizione le loro competenze. I pazienti che devono visitare definiscono quali sono le loro necessità in termini di prestazioni sanitarie, durata del trattamento, membri da evitare e finestra temporale gradita per l'appuntamento. Inoltre, sono presenti alcuni pazienti che richiedono quelli che vengono definiti servizi doppi o interdipendenti, che possono prevedere una simultaneità o una sequenzialità, e quindi devono fornire, oltre alle informazioni del secondo servizio richiesto, una finestra temporale entro cui è importante che il secondo servizio sia espletato. La possibilità che un unico caregiver fornisca entrambe le prestazioni per lo stesso paziente con doppia prestazione è ovviamente impossibile per prestazioni simultanee. è anche esplicitamente

vietato per quelli sequenziali dal lavoro di Mankowska et al. [4].
Si considera che lo scheduling degli appuntamenti venga fatto giorno per giorno, tutti i tempi vengano espressi in minuti, iniziando da 0 impostato di default come orario di inizio delle attività giornaliere assimilabile alle 6.00 del mattino, orario a cui si assume che tutti gli elementi dello staff siano nell'ufficio centrale. Le distanze sono disponibili come matrice di distanze e coprono il collegamento di tutti i pazienti tra di loro e tutti i pazienti con l'ufficio centrale e siano espressi in minuti di viaggio richiesti per viaggiare [4].

Il dataset di riferimento utilizzato per testare l'algoritmo è disponibile presso il repository github del gruppo dei ricercatori dell'Università degli Studi di Udine, come descritto nel loro lavoro [4].

Come già evidenziato in precedenza, il problema appartiene alla classe NP-Hard e richiede un elevato numero di tentativi, e quindi tempo (nell'ordine delle ore), per convergere a una soluzione, nonostante l'uso di un algoritmo metaeuristico. L'algoritmo scelto è ALNS, che, come dimostrato in precedenza, si presta bene agli obiettivi preposti.

Nel fare ciò è imperativo minimizzare i tempi, i ritardi e fornire tutti i servizi in tempi accettabili, pertanto saranno indispensabili i parametri scelti per la funzione di costo.

Un esempio di funzionamento del sistema può essere rappresentato come segue:

Patients

ID	Time window	Service (duration) first	second	Separation min	max
p_1	240 – 360	s_2 (30)	—	—	
p_2	120 – 180	s_3 (20)	—	—	
p_3	0 – 60	s_2 (45)	—	—	
p_4	120 – 210	s_2 (30)	s_3 (30)	0	0
p_5	270 – 420	s_1 (15)	s_3 (30)	30	45
p_6	360 – 420	s_1 (45)	s_3 (20)	60	90

Caregivers

ID	Services
c_1	s_1, s_2
c_2	s_3
c_3	s_2, s_3

Distances

	co	p_1	p_2	p_3	p_4	p_5	p_6
co	0	38	34	56	7	13	26
p_1	39	0	23	22	32	50	58
p_2	35	23	0	44	28	47	43
p_3	56	22	44	0	54	59	78
p_4	7	32	28	51	0	19	28
p_5	13	45	47	59	19	0	35
p_6	27	57	42	77	28	35	0

Figura 3.1: Dati d'esempio di una istanza

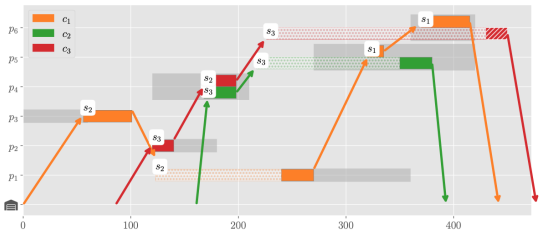

Figura 3.2: Possibile rappresentazione grafica di una soluzione in base al tempo e al servizio

3.1 Il sistema progettato progettato

Il sistema è completamente progettato in *C++* e utilizza il paradigma ad oggetti. È suddiviso in tre moduli principali:

1. Lettura e scrittura dei file per le istanze in input e le soluzioni in output
2. Produzione della prima soluzione e ottimizzazione
3. Validazione della soluzione

Questi moduli sono complementari e possono essere utilizzati per scrivere programmi diversi. Il sistema progettato utilizza tutti e tre i moduli per il programma principale, che si occupa della produzione e della validazione delle soluzioni. Inoltre, è presente un secondo programma dedicato esclusivamente alla validazione delle soluzioni fornite sulla base degli input.

3.2 Lettura e scrittura dei dati

3.2.1 La lettura degli input

Il primo modulo si occupa di leggere le istanze del dataset, rigorosamente in formato JSON, il cui percorso viene fornito in ingresso all'avvio del programma, come segue:

```
./HomeCare "<AbsoluteOrRelative_pathTo/InputFile.json>"
```

Figura 3.3: Comando di avvio del programma

L'istanza in questione è realizzata come un oggetto JSON i cui campi sono a loro volta oggetti che definiscono le varie entità che compongono l'istanza e che possiamo categorizzare come segue: I pazienti sono raccolti in un vettore di oggetti

"patients": [

....

], ...

In figura (3.5) è possibile vedere come si presentano i pazienti con servizio singolo all'interno del vettore.

```
{
  "id": "p1",
  "location": [45.62, 13.21],
  "time_window": [
    240,
    360
  ],
  "required_caregivers": [
    {
      "service": "s2",
      "duration": 30
    }
  ]
},
```

Figura 3.4: Struttura JSON di un paziente a servizio singolo

dove *id* banalmente rappresenta l'id del paziente, *location* la posizione nella mappa espressa in coordinate, *time_window* rappresenta la finestra temporale messa a disposizione dal paziente, *required_caregivers* contiene i servizi necessari distinti per l'id della tipologia e la loro durata.
I pazienti con servizio doppio e simultaneo si riconoscono da (3.5)

Figura 3.5: Struttura JSON di un paziente a servizio simultaneo

I campi sono praticamente identici a quelli del paziente a servizio singolo ma è possibile notare delle differenze ovvero due servizi nel campo *required_caregivers* e il campo riguardante l'interdipendenza *synchronization* che specifica la tipologia di servizio richiesto che in questo caso è *simultaneous*, simultaneo.

Per i pazienti con servizio doppio e sequenziale la struttura del JSON si presenta come in fig (3.6)

Figura 3.6: Struttura JSON di un paziente a servizio sequenziale

Anche in questo caso i campi sono identici ai pazienti precedenti, ma la tipologia sequenziale si differenzia da quella simultanea per i valori nel campo *type* di *synchronization*, inoltre, comprende un campo *distance* che contiene due valori con la distanza di tempo minima e quella massima che i servizi devono avere.
I servizi sono contenuti nel vettore di oggetti *services*

”services”: [

....

], ...

La struttura degli oggetti *service* è uguale per tutti non essendoci, per quest'oggetto, casi particolari. La struttura è espressa nell'immagine di seguito (3.7)

Figura 3.7: Struttura JSON di un servizio

Il campo *id* serve a identificare la tipologia di servizio, mentre il campo *default_duration* specifica la durata del servizio se non presente tra le informazioni del paziente.
Anche i *caregivers* sono contenuti in vettore di oggetti molto simile ai precedenti

"caregivers": [

....

], ...

La struttura del *caregiver* è molto semplice ed immediata e si organizza come in figura (3.8).

Figura 3.8: Struttura JSON di un caregiver

Dove l'id identifica il *caregiver* e il *percorso* a cui si riferisce, mentre *abilities* è un vettore che indica tutte le tipologie di servizio che il caregiver può offrire identificate per l'id del servizio.
La struttura degli uffici centrali si presenta come in figura (3.9).

Come è possibile notare nella figura (3.9) anche gli uffici centrali erano stati progettati per contenerne più di uno ma dati i vincoli del sistema, uno sufficiente. I dati contenuti nell'oggetto *central_offices* sono l'identificativo e

Figura 3.9: Struttura JSON degli uffici centrali

la posizione sulla mappa espressa nelle sue coordinate geografiche.
La matrice di distanze esprime il peso di tutti gli archi del grafo, rappresentati in minuti. Questa scelta è dovuta al fatto che si assume che il problema sia organizzato come un grafo non orientato in cui tutti i nodi sono collegati tra loro. La matrice di distanze si presenta nel file JSON come un vettore di N vettori composto ognuno da N numeri interi (3.10).

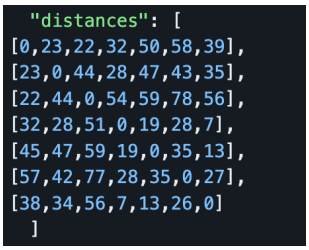

Figura 3.10: Struttura JSON della matrice delle distanze

In particolare, si assume che la matrice delle distanze sia nel seguente ordine (sia per righe che per colonne): {d, p1, p2, ..., pn} e che quindi la diagonale principale abbia solo zeri.

3.2.2 La scrittura degli output

I file di output sono prodotti secondo specifiche direttive, come indicato nel repository github precedentemente discusso.
Anche l'output è un oggetto JSON e la sua struttura è molto semplice.

```
{
    "cost_components" : {
        "EHHC_TotalExtraTime" : 2319,
        "EHHC_TotalTardiness" : 718,
        "EHHC_TotalWaitingTime" : 0,
        "EHHC_TravelTime" : 3252
    },
    "global_ordering" : [
        "p64",
        "p53",
        "p28",
        "p73",
        ...
    ],
    "routes" : [
        {
            "caregiver_id" : "c1",
            "locations" :
            [
                {
                    "arrival_time" : 19,
                    "departure_time" : 34,
                    "patient" : "p53",
                    "service" : "s1"
                },
                {
                    "arrival_time" : 65,
                    "departure_time" : 80,
                    "patient" : "p127",
                    "service" : "s1"
                },
                {
                    "arrival_time" : 91,
                    "departure_time" : 151,
                    "patient" : "p28",
                    "service" : "s1"
                },
                ...
```


Figura 3.11: Struttura file di output in un esempio semplificato

L'immagine (3.11) mostra una versione tagliata dell'output ma dalla è comunque possibile osservare la struttura degli oggetti che la compongono. Come prima cosa abbiamo i risultati della soluzione dopo l'ottimizzazione, ovvero:

- EHHC_HighestTardiness: Ritardo massimo.
- EHHC_MaxIdleTime: Anticipo massimo.
- EHHC_TotalExtraTime: Totale del tempo in speso in più.
- EHHC_TotalTardiness: Totale dei ritardi.
- EHHC_TotalWaitingTime: Totale del tempo di attesa dei caregiver.
- EHHC_TravelTime: Totale del tempo di viaggio dei caregive.r

La seconda struttura è un elenco di pazienti ordinati in base all'urgenza, ovvero in base alle finestre temporali che si chiudono prima, garantendo così

priorità ai pazienti con esigenze più immediate.
Infine, la terza struttura è un vettore di percorsi. Ogni percorso è associato a un caregiver specifico e include una lista dettagliata degli appuntamenti giornalieri. Questa lista comprende l'ID del paziente, la tipologia di servizio richiesto, il tempo di arrivo previsto e il tempo di partenza previsto, fornendo una panoramica completa delle attività pianificate per ciascun caregiver.

3.3 La produzione delle soluzioni

Questo modulo rappresenta certamente il nucleo del sistema. Il suo compito è sfruttare ALNS, l'algoritmo metaeuristico implementato, per risolvere il problema HHCRSP, garantendo tempi di risoluzione ragionevoli e ottenendo soluzioni di qualità accettabile.
In questo modulo viene prodotta anche la soluzione iniziale, che costituisce il punto di partenza della ricerca nello spazio delle soluzioni. La soluzione iniziale è cruciale poiché l'algoritmo metaeuristico ALNS richiede di partire da una soluzione di base che verrà successivamente migliorata in modo iterativo. Come descritto nella sezione dedicata ad ALNS, l'algoritmo utilizza una serie di operazioni di *distruzione/riparazione* (destroy/repair). Queste operazioni distruggono parzialmente una soluzione esistente per poi ripararla, cercando di creare una nuova soluzione nella speranza che sia migliore della precedente. Le coppie di operatori vengono estratte a sorte, scelte con un meccanismo noto come *roulette wheel* (routa di routelle). Ad ogni coppia viene associato un punteggio che mostra la loro efficacia nelle iterazioni precedenti. Ciò serve a guidare l'algoritmo nello scegliere le coppie nella successiva serie di iterazioni favorendo coppie che hanno dato risultati migliori. Il punteggio viene assegnato alla soluzione dal criterio di accettazione che, dopo aver verificato la validità della soluzione, utilizza il meccanismo dell'euristica di Simulated Annealing (SA). SA consente ad ALNS di diversificare le soluzioni, aumentando la probabilità di evitare di rimanere bloccati in ottimi locali.
Per gestire efficacemente i nodi su cui gravano i vincoli di sincronizzazione,

dopo aver prodotto la soluzione iniziale, ogni variazione apportata al percorso, si di distruzione che di riparazione, comporta una rivalutazione dello stesso e un rescheduling. Questa operazione assegna una priorità minore ai nodi indipendenti, che vengono utilizzati per bilanciare la soluzione e mantenere i vincoli esistenti sui nodi interdipendenti. In questa fase ricoprono un'importanza cruciale l'algoritmo di programmazione dinamica fino a 7 nodi, dopodiché data l'esplosione delle combinazioni, si utilizzano in sequenza gli algoritmi 2-opt e 3-opt.
Per controllare che la nuova soluzione non sia stata già prodotta si tiene traccia delle soluzioni visitate assegnando una chiave hash a ciascuna soluzione e memorizzando la chiave in una tabella hash.

3.3.1 Entità del problema

Oltre alle entità che definiscono il problema, sono state aggiunte altre entità che aiuteranno nella definizione e nella risoluzione del problema. Le entità sono:

- **Node**: rappresenta sia gli uffici centrali, punti di partenza e arrivo dei caregiver, sia i pazienti. Questa entità contiene tutte le informazioni necessarie per definire un appuntamento nella route assegnata e controllarne contestualmente i vincoli.

- **Route**: rappresenta un singolo percorso effettuato da un caregiver durante la giornata. Essa gestisce tutte le componenti del percorso, incluse le operazioni di aggiunta ed eliminazione dei nodi e la valutazione delle variabili di valutazione come il ritardo più alto la somma dei ritardi la somma della distanza percorsa.

- **Schedule**: rappresenta la soluzione, l'intero piano giornaliero di attività di un insieme di caregiver, organizzando e gestendo le diverse route assegnate a ciascuno di essi. Questa entità è fondamentale per

coordinare i percorsi e garantire che tutti i pazienti ricevano il servizio necessario nel tempo previsto. Inoltre, si occupa di tracciare le variabili di valutazione dell'intera soluzione.

- **ScheduleOptimiser**: rappresenta un'estensione della entità Schedule, arricchita con metodi specifici per il processo di ottimizzazione e gestione avanzata delle sincronizzazioni per il rescheduling dei caregiver e dei pazienti e variabili che mappano i pazienti nelle soluzioni.

3.3.2 Produzione della prima soluzione

Per la generazione della soluzione di partenza, ho optato per il metodo utilizzato da Mankowska, un euristica greedy molto semplice e veloce. Il procedimento consiste nell'ordinare tutti i pazienti, dando priorità ai pazienti più urgenti; questo si è tradotto nell'ordinare la lista in ordine crescente basato sulla chiusura delle finestre temporali. Successivamente, in un processo iterativo, si è analizzato uno per uno i pazienti così ordinati e, uno ad uno, vengono assegnati al caregiven o ai caregiven con il costo più basso, prestando ovviamente attenzione ai vincoli imposti dai servizi, dai caregiven e dai pazienti per mantenere la soluzione valida. Questa operazione viene eseguita dal metodo *append* presente nell'entità Schedule e sostanzialmente si occupa di inserire il nodo in coda alla route meno costosa considerando a costo infinito quelle route che non rispettano i vincoli.
La procedura viene ripetuta finché tutti i pazienti sono stati inseriti in una route. Alla fine del processo, avremo definito lo schdeuling che formerà la soluzione di partenza del sistema.

Algorithm 1: GeneraPrimaSoluzione

Input: Lista Pazienti P, id caregiver c

Output: Primo Schedule S

$S \leftarrow$ inizializza a vuoto;

$P \leftarrow order(P)$;

for *ogni Paziente* $p \in P$ **do**

  $c1 \leftarrow findCargiver(S, p_{s1})$;

  $time \leftarrow calculateArrivalTime(S, p_{s1}, c1)$;

  if p *ha servizio Doppio* **then**

    $c2 \leftarrow findCargiver(S, p_{s2})$;

    $syncTime \leftarrow calculateArrivalTime(S, p_{s2}, c2)$;

    if p_s1 *è simultaneo* **then**

      $time \leftarrow max(time, SyncTime)$;

      $syncTime \leftarrow time$;

    else

      $c1 \leftarrow findCargiver(S, p_{s1})$;

      $syncTime \leftarrow time + \delta$;

      `//Controlla che il tempo sia nella finestra di delay`

    $append(S, p_s2, syncTime)$;

  $append(S, p_s1, time)$;

return S;

3.3.3 Ottimizzazione di ALNS

Il passo successivo alla produzione della prima soluzione è l'ottimizzazione, quindi si può dare inizio all'algoritmo ALNS. All'inizio del ciclo di iterazioni, la prima operazione consiste nella scelta della coppia di algoritmi destory/repair da utilizzare nell'iterazione, mediante il meccanismo di **selezione a roulette**.
La selezione a roulette è un metodo di selezione stocastico in cui la probabilità di selezione di un individuo è proporzionale alla sua efficienza nell'identificare una nuova soluzione. Questo metodo, ispirato alla roulette del mondo reale, presenta importanti differenze: nella selezione a roulette utilizzata negli algoritmi, gli slot non restano sempre gli stessi e la probabilità di selezione è ponderata. Maggiore è l'efficienza di un individuo, maggiore è la probabilità di essere selezionato. Per implementare una versione ponderata della roulette, il primo componente è la proporzionalità tra l'efficienza di un individuo e la sua probabilità di selezione. Tuttavia, poiché la somma delle probabilità deve essere uguale a uno $\sum_{i=1}^{n} p\_i$ è necessario normalizzare i valori delle singole probabilità all'intervallo [0,1].

In sintesi, possiamo calcolare la probabilità di selezione di ciascun indivi-

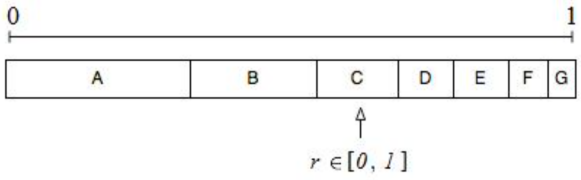

Figura 3.12: Esempio di distribuzione di probabilità in scelta a roulette

duo come segue: in una popolazione con n individui, per ciascuna coppia (d, r) con un corrispondente punteggio $f_{(d,r)}$, calcoliamo la corrispondente probabilità $p_{(d,r)}$ di selezione come:

$$p_{(d,r)} = \frac{f_{(d,r)}}{\sum_{i=1}^{n} f_i}$$

L'idea di base è tenere traccia di un punteggio per ciascuna euristica, che misura il rendimento recente dell'euristica. Un punteggio elevato corrisponde a un'euristica di successo. L'intera ricerca è divisa in una serie di segmenti. Un segmento è costituito da un numero definito di iterazioni dell'euristica ALNS; qui definiamo un segmento come 100 iterazioni. All'inizio di ciascun segmento, il punteggio di tutte le euristiche viene impostato su zero. Durante il segmento, il punteggio di un'euristica viene aumentato di σ_1, σ_2 o σ_3 nelle seguenti situazioni:

- σ_1 La soluzione prodotta nell'ultima iterazione dalle operazioni destroy/repair ha portato ad una nuova migliore soluzione globale.
- σ_2 La soluzione prodotta nell'ultima iterazione dalle operazioni destroy/repair ha portato ad una nuova soluzione non accettata ma migliore della soluzione corrente.
- σ_3 La soluzione prodotta nell'ultima iterazione dalle operazioni destroy/repair ha portato ad una nuova soluzione che nonostante sia peggiore della soluzione corrente e dell'ottimo globale.

Il caso diσ_1 è chiaro: se un'euristica è in grado di trovare una nuova soluzione migliore in assoluto, allora ha funzionato bene. Allo stesso modo, se un'euristica è riuscita a trovare una soluzione che non è stata visitata prima e viene accettata dai criteri di accettazione nella ricerca ALNS, allora l'euristica ha avuto successo poiché ha portato avanti la ricerca.
È sensato distinguere tra le due situazioni corrispondenti ai parametri σ_2 e σ_3 perché preferiamo euristiche che possano migliorare la soluzione (σ_2), ma ci interessano anche euristiche che possano diversificare la ricerca (σ_3). È importante notare che premiamo solo le soluzioni non visitate. Questo incoraggia le euristiche in grado di esplorare nuove parti dello spazio della soluzione.
Il peso di ogni coppia i viene aggiornato alla fine di ogni segmento con la formula

$$w_{i,j+1} = w_{ij}(1 - r) + r\frac{\pi_i}{\theta_i} \tag{23}$$

Il fattore di reazione r controlla la rapidità con cui l'algoritmo di aggiustamento del peso reagisce ai cambiamenti nell'efficacia dell'euristica. Se r è zero allora non utilizziamo affatto i punteggi e ci atteniamo ai pesi iniziali. Se r è impostato su uno, lasciamo che sia il punteggio ottenuto nell'ultimo segmento a decidere il peso. Per tenere in considerazione sia i vecchi tentativi che i nuovi r è stato impostato a 0.5.
Come evidenziato nel passaggio precedente, non sempre viene accettata la soluzione con il risultato migliore. Sono stati infatti definiti dei **criteri di accettazione** specifici per gestire questo aspetto basati su quelli di Simulated Annealing. Questo approccio è stato adottato per evitare che l'euristica rimanga intrappolata in un minimo locale. Pertanto, sembra sensato, a volte, accettare soluzioni peggiori di quella attuale. Per fare ciò, utilizziamo i criteri di accettazione definiti nel Simulated Annealing, che sono stati approfonditi nel capitolo precedente.
La temperatura iniziale non è fissa e viene calcolata come

$$T = -\frac{\omega_T}{\ln 0.5} f_c(S) \tag{24}$$

dove ω_T è una costante predefinita e $f_C(S)$ è il costo della soluzione iniziale. Ad ogni iterazione la temperatura viene ricalcolata con la formula $T = T \bullet c$ dove c è la velocità di raffreddamento e quindi $0 < c < 1$.
L'algoritmo ALNS può quindi essere riassunto nel seguente pseudo codice:

Gli operatori di destroy

Gli operatori di *destroy* sono algoritmi euristici che smantellano parzialmente la soluzione corrente. In pratica, questi algoritmi eliminano selettivamente alcuni nodi dalle liste esistenti, creando delle lacune nella soluzione. Questa fase è cruciale perché consente di esplorare nuove possibili configurazioni che potrebbero portare a soluzioni migliori. Gli operatori di *destroy* possono variare per complessità e strategia. Di seguito una panoramica degli euristici utilizzati.

Algorithm 2: ALNS

Input: Lista Pazienti P, Lista caregivers C

Output: Best Schedule S_{best}

$S \leftarrow generaPrimaSoluzione()$;

$S_{best} \leftarrow S$;

while *Attivazione del criterio di arresto* **do**

  $S' \leftarrow S$;

  $(d, r) \leftarrow rouletteWheel()$;

  `//torna la coppia di operatori scelta` $S' \leftarrow d(S)$;

  $S' \leftarrow r(S')$;

  if S' *è migliore di* S_{best} **then**

    $S \leftarrow S'$;

    $S_{best} \leftarrow S$;

  else

    if *criterio di accettazione* **then**

      $S \leftarrow S'$;

  $aggiornaPunteggiOperatori()$;

return S_{best};

Random Removal

L'operatore Random Removal rimuove un dato numero di nodi q, dalla soluzione scegliendo sia il percorso che il nodo stesso in maniera casuale.

Worst Removal

Il Worst Removal si basa su un operatore proposto in [15]. Questo operatore rimuove i clienti q con il guadagno di rimozione più elevato. Più precisamente, il guadagno è definito come la differenza tra il costo quando il cliente è nella soluzione e il costo quando viene rimosso $cost(i, s) = f(s) - f_{-i}(s)$ dove $f_{-i}(s)$ indica il costo della soluzione senza il nodo i. Il guadagno si normalizza dividendolo per il costo medio degli archi entranti del nodo corrispondente. Lo scopo di questa normalizzazione è quello di evitare di scegliere ripetutamente clienti che si trovano lontano da quelli rimanenti valutandone il peso all'interno del percorso.
Nel Worst Removal sono presenti due parametri q e p, che rappresentano rispettivamente il numero di nodi da eliminare e il parametro per il controllo della randomizzazione della rimozione. Possiamo riassumere l'algoritmo con le seguenti operazioni:

Algorithm 3: Worst Removal

Input: Soluzione corrente S, $q \in N$, $p \in R^+$

Output: soluzione smantellata S, lista nodi eliminati D

while $q > 0$ **do**

- $L \leftarrow$ Tutt i nodi $i \in S$, sorted by descending $cost(i, s)$ normalizzati;
- $y \leftarrow$ Numero casuale $[0, 1]$;
- $d \leftarrow L[y^p|L|]$;
- $S \leftarrow S - \{d\}$;
- $D \leftarrow D \cup \{d\}$ $q \leftarrow q - 1$;

Related Removal

Questa euristica di rimozione è stata proposta da Shaw e in questa progetto è leggermente modificato per adattarsi al HHCSRP. L'idea generale è quella di rimuovere i nodi che sono in qualche modo simili, poiché ci aspettiamo che sia ragionevolmente facile mescolare nodi simili e quindi creare soluzioni nuove, forse migliori. Se scegliamo di rimuovere nodi molto diversi tra loro, potremmo non guadagnare nulla quando reinseriamo i nodi poiché potremmo essere in grado di inserire i nodi solo nelle loro posizioni originali o in alcune posizioni subottimali.
Prima di iniziare, occorre definire la somiglianza di due nodi i e j utilizzando una misura di correlazione $R(i, j)$. Più $R(i, j)$ è basso, più le due nodi sono correlate.
La funzione di correlazione composta per HHCSRP si compone principalmente di tre componenti che vanno a testare le proprietà fondamentali per un nodo, individuate in:

1. La distanza tra i nodi, per valutare la similarità spaziale dei nodi
2. La distanza tra le finestre temporali, per valutare la similarità temporale dei nodi
3. Il rapporto tra la possibilità di servire il nodo j e il nodo seme i

ad ogni componente viene assegnata un peso, che non ha valore di per sé ma serve a dosare il peso della caratteristica associata all'interno dell'espressione. Nel caso implementato è stato dato a tutti i pesi lo stesso valore, 1.
Si può quindi definire la formula della somiglianza come segue:

$$\begin{aligned} R(i,j) = & \, distance_{i,j} \\ & + (|open\_window_i - open\_window_j| \\ & + |close\_window_i - close\_window_j|) \\ & + \left(1 - \left|\frac{|caregivers_i \cap caregivers_j|}{max\{caregivers_i, caregivers_j\}}\right|\right) \end{aligned} \tag{25}$$

Nell'implementazione, un nodo seme viene scelto in modo casuale e viene stilata una lista di tutti i nodi ordinati per somiglianza crescente. Anche in questo caso, come nel Worst Removal, viene aggiunto un ulteriore elemento di casualità: non viene eliminato il nodo più simile, ma il nodo individuato alla posizione $y^p|L|$, dove y è un numero casuale tra $[0, 1]$, p un parametro scelto a priori e $|L|$ fa riferimento alla lunghezza della lista prodotta da tutti i nodi ancora presenti in soluzione.
Dopo aver scelto il nodo da eliminare, esso viene rimosso dalla soluzione e e salvato in un insieme D. Dall'insieme D viene quindi estratto a sorte un nuovo nodo seme. Il procedimento si ripete finché l'insieme D non raggiunge cardinalità q [14].

Algorithm 4: Related Removal

Input: Soluzione corrente S, $q \in N, p \in R^+$
Output: soluzione smantellata S, lista nodi eliminati D
$seed \leftarrow randomnode \in S$;
$D \leftarrow \{seed\}$;
while $q > |D|$ **do**
  $L \leftarrow$ Tutt i nodi $j \in S$, sorted by ascending $shaw(seed, j)$;
  $y \leftarrow$ Numero casuale $[0, 1]$;
  $d \leftarrow L[y^p|L|]$;
  $remove(d, S)$;
  $D \leftarrow D \cup \{d\}$;
  $seed \leftarrow randomElementFrom(D)$

Cluster Removal

L'euristica di rimozione dei cluster, chiamata Cluster Removal, si basa su Related Removal, ma si concentra sui nodi appartenenti allo stesso percorso. Mentre in Related Removal i compiti vengono scelti per somiglianza indipendentemente dal percorso, in Cluster Removal i compiti selezionati sono quelli che si trovano attualmente sullo stesso percorso. Il percorso in questione

viene scelto casualmente e deve essere composto da almeno 7 nodi (escluso l'ufficio centrale) [16].
Tutti i nodi associati al percorso selezionato vengono uniti in un nuovo grafo completamente collegato e non orientato $G(V, E)$ dove i vertici V sono i nodi dei percorsi e gli archi E hanno peso pari alla similarità calcolata tra i vertici. Partendo da questo nuovo grafo, si costruisce il minimum spanning tree utilizzando l'algoritmo di Kruskal e vengono eliminati dal percorso i primi elementi con rango più alto. Se non si raggiunge il numero di eliminazioni q alla prima iterazione, si itera il procedimento con il successivo percorso.
Il raggruppamento e la rimozione dei compiti vengono ripetuti finché non sono stati rimossi almeno q compiti o non esistono più percorsi con almeno tre compiti.

Algorithm 5: Cluster Removal

Input: Soluzione corrente S, $q \in N$
Output: soluzione smantellata S, lista nodi eliminati D
$route \leftarrow randomroute \in S$;
$D \leftarrow \{\emptyset\}$;
while $q > |D|$ **do**
    $G \leftarrow$ grafo $G(V, E)$ $V \in route$, $E_{i,j}, i, j \in V, similarity(i, j)$;
    $T \leftarrow KruskalTree()$;
    $delete \leftarrow q - |D|$ **for** *first higher rank delete node* $\in T$ **do**
        $remove(node, S)$;
        $D \leftarrow D \cup \{node\}$;
    $route \leftarrow randomroute \in S$;

Gli operatori di Repair

Gli operatori di repair sono algoritmi progettati per ricostruire la soluzione dopo che è stata parzialmente distrutta dagli operatori di destroy. Il loro obiettivo è reinserire i nodi rimossi in modo ottimale, rispettando i vincoli

del problema e migliorando la qualità complessiva della soluzione. Utilizzando diverse strategie e tecniche, gli operatori di repair giocano un ruolo cruciale nel processo iterativo di miglioramento della soluzione all'interno dell'algoritmo ALNS.

3.3.4 Greedy Repair

L'euristica Greedy Repair di base è un'euristica di costruzione semplice. Esegue al massimo n iterazioni, e ad ogni iterazione cerca di inserire una richiesta in uno dei percorsi disponibili nella posizione che minimizza il costo complessivo dello scheduling, come descritto in [14].. Durante ciascuna iterazione, l'algoritmo cerca di inserire tutti i nodi non ancora assegnati in ogni percorso possibile e nella miglior posizione e mantiene lo scheduling meno costoso, da cui in seguito riparte per costruire le nuove soluzioni.

La caratteristica principale di questa euristica è che, in ogni iterazione, viene modificato solo il percorso in cui viene eseguito l'inserimento di un nuovo nodo. Questo evita la necessità di ricalcolare i costi di inserimento per tutti gli altri percorsi, il che porta a un notevole miglioramento delle prestazioni dell'euristica di inserimento. Tuttavia, un problema evidente è che i nodi più difficili da inserire, quelli con costi più elevati, tendono ad essere rimandati alle ultime iterazioni. Questo accade perché molti dei percorsi disponibili possono essere già saturi, limitando le opportunità di inserire questi nodi in posizioni vantaggiose. Per cercare di ovviare, almeno in parte questo problema viene implementato Regret Repair.

3.3.5 Regret Repair

L'euristica Regret Repair cerca di migliorare l'euristica Greedy Repair incorporando una sorta di look-ahead quando si seleziona il nodo da inserire. Sia $x_i, k \in \{1, ..., m\}$ una variabile che indica la rotta per la quale la richiesta i ha il k-esimo costo di inserzione più basso, ovvero $f_{i,x_{i,k}} \leqslant f_{i,x_{i,k'}}$ per $k \leqslant k'$ [14]. Usando questa notazione possiamo esprimere c_i come $c_i = f_{i,x_{i,2}} - f_{i,x_{i,1}}$.

In altre parole, il valore del rimpianto è la differenza nel costo di inserimento della richiesta nel suo percorso migliore e nel secondo percorso migliore. In ogni iterazione l'euristica Regret Repair sceglie di inserire la richiesta i che massimizza

$$\max_{i \in U} c_i \tag{26}$$

I pareggi vengono risolti selezionando l'inserimento con il costo più basso scegliendo l'inserimento di cui ci pentiremo di più se non verrà fatto adesso. L'euristica può essere estesa in modo naturale per definire una classe di euristiche Regret Repair: l'euristica del *regret-k* è l'euristica di costruzione che in ogni passo di costruzione sceglie di inserire la richiesta i che massimizza:

$$\max_{i \in U} \left\{ \sum_{j=1}^{k} (f_{i,x_{i,j}} - f_{i,x_{i,1}}) \right\} \tag{27}$$

Con questo presupposto possiamo definire l'euristica definita all'inizio del paragrafo una euristica Regret-2, mentre l'euristica Greedy, può essere vista come Regret-1.
Solitamente il limite delle euristiche Regret-k è che deve valere la relazione $k \leqslant m$ dove m indica il numero di possibili percorsi in cui è possibile inserire i. In HHCRSP non è detto che questa relazione venga rispettata, pertanto in quel caso il costo viene considerato infinito è il nodo viene scelto.

3.4 Scelta dei parametri

Adaptive Large Neighborhood Search (ALNS) è un metodo di ottimizzazione basato su metaeuristiche utilizzato per risolvere problemi complessi. La scelta dei parametri di ALNS è cruciale per ottenere buoni risultati e può essere un compito non banale. Di seguito, viene descritto un metodo strutturato per scegliere i parametri di ALNS, che include l'uso di esperimenti sistematici e tecniche di ottimizzazione dei parametri.
Per scegliere i parametri di ALNS è importante svolgere i seguenti passaggi:

- **Definire i parametri di ALNS**:

  - **Rimozione e riparazione** q: Numero di elementi su cui applicare l'algoritmo.
  - **Parametri delle euristiche** $p_{worst}, p_{related}$: Parametri che aggiungono randomicità nelle euristiche.
  - **Parametro di setting della temperatura iniziale** ω_T: Parametro utilizzato per il calcolo della temperatura iniziale T_i
  - **Temperatura finale** T_f: Parametro utilizzato nell'accettazione delle soluzioni (se si utilizza un approccio tipo Simulated Annealing).
  - **Fattore di Raffreddamento** c: Tasso di riduzione della temperatura.
  - **Numero di Iterazioni** I: Numero totale di iterazioni dell'algoritmo.
  - **Numero di Segmenti** N_S: Numero di volte in cui ripetere dell'algoritmo.
  - **Parametri di Adattamento** $\sigma_1, \sigma_2, \sigma_3$: Parametri che controllano come vengono aggiornati i pesi degli operatori.
  - **Parametro di aggiornamento persi** r: Parametro che dosa l'importanza dei pesi del segmento appena finito e di quelli passati nella fase di aggiornamento.

- **Definire il range dei parametri**: Vengono definiti intervalli ragionevoli per ciascun parametro sulla base di conoscenze preliminari (assunzioni basate su studio di funzioni) e letteratura esistente.
  Per quanto riguarda q, il **numero di elementi da eliminare e riparare ad ogni iterazione**, mi sono affidato alla documentazione esistente, facendo riferimento principalmente a [16], dove il parametro viene stabilito come un numero casuale nell'intervallo

$$[\min\{0.1|V|, 30\}, \min\{0.4|V|, 60\}]$$

dove $|V|$ indica il numero di nodi presenti nel problema, in questo caso il numero di servizi richiesti dai pazienti.

Per i **parametri di riferimento delle euristiche** p_{worst} e $p_{related}$ *Worst Removal* e *Related Removal* ho utilizzato uno studio di funzioni e preso una decisione basata su ragionamento. Dato che sia Worst Removal che Related Removal, utilizzano una funzione definita basata su una variabile casuele, supponendo che ogni numero della distribuzione abbia la stessa probabilità di essere estratto, si è fatta una speculazione basata sulla probabilità.

Dato che in entrambi i casi si vuole mantenere il risultato dell'euristica senza stravolgerlo troppo con la randomizzazione, si è scelto un parametro p con un range impostato a $5 \leqslant p \leqslant 7$. Con questo parametro si ha che, in istanze di diversa grandezza, in media l'elemento "premiato" dall'euristica ha una probabilità del 35% circa di essere estratto, mentre la probabilità che venga estratto un elemento del primo quarto è di circa il 60%.

Questo approccio bilancia l'efficacia dell'euristica con la necessità di esplorare nuove soluzioni, ottimizzando la rimozione e l'inserimento dei nodi per evitare minimi locali, mantenendo però l'integrità della soluzione corrente senza eccessiva casualità.

Per quanto riguarda gli altri parametri non definiamo esplicitamente il range e ci limitiamo a ricavarne il valore tramite prove nel passo successivo.

- **Esecuzione degli Esperimenti**:
  - **Simulazioni Multiple**: Eseguire l'ALNS con le diverse combinazioni di parametri definite. Ogni configurazione dovrebbe essere eseguita più volte per tenere conto della variabilità stocastica.
  - **Raccolta Dati**: Registrare i risultati di ciascuna esecuzione, inclusi il costo della soluzione, il tempo di esecuzione.
- **Analisi dei Risultati**:

– **Statistical Analysis**: Utilizzare tecniche di analisi statistica per identificare i parametri che hanno un impatto significativo sulle prestazioni. Analizzare i risultati per trovare combinazioni di parametri che producono le migliori soluzioni.

3.5 Validazione delle soluzioni

Si può accedere a questo modulo in due modi: viene richiamato direttamente dal modulo di generazione delle soluzioni ogni qualvolta una nuova soluzione viene generata o può essere eseguito il programma di validazione di soluzioni.
Nel secondo caso bisogna specificare sia il percorso del file JSON dell'istanza del problema, sia il percorso del file JSON dell'output.
Il processo di validazione delle soluzioni è cruciale per assicurare che

```
./Validate "<AbsoluteOrRelative_pathTo/InputFile.json>" "<AbsoluteOrRelative_pathTo/SolutionFile.json>"
```

Figura 3.13: Esecuzione programma di validazione

le soluzioni proposte siano non solo ottimali in termini di costi e tempi, ma anche aderenti a tutti i vincoli definiti dal problema. In particolare, la validazione delle soluzioni per il problema HHCRSP (Home Health Care Routing and Scheduling Problem) comporta diverse fasi chiave:

– **Compatibilità dei Servizi**: Verificare che ogni nodo riceva il servizio richiesto da un caregiver qualificato.

– **Rifiuti dei Pazienti**: Controllare che nessun paziente abbia rifiutato il caregiver assegnato.

– **Unicità delle Visite**: Assicurarsi che il caregiver non visiti lo stesso nodo più di una volta.

- **Sincronizzazione dei Servizi**: Verificare che tutti i servizi sincronizzati rispettino le tempistiche e le relazioni temporali necessarie.
- **Integrità dei Dati**: Assicurarsi che le distanze percorse e gli orari degli appuntamenti siano corretti e rispettino le finestre temporali dei pazienti.

Capitolo 4

Analisi sperimentale

Il sistema è stato interamente sviluppato in *C++* senza l'ausilio di framework di ricerca locale, per garantire una maggiore flessibilità e controllo sugli algoritmi implementati. Gli esperimenti sono stati eseguiti su un MacBook Pro con sistema operativo macOS Monterey, equipaggiato con un processore quad-core Intel Core i7 da 2,2 GHz, in modo da assicurare un ambiente di test stabile e riproducibile. Il repo GitHub del progetto è disponibile gratuitamente e testabile.
Le istanze utilizzate per la valutazione del sistema provengono dal database dei ricercatori dell'Università degli Studi di Udine. Queste istanze sono disponibili nel repository GitHub ufficiale (repo GitHub) e sono contenute nella cartella denominata "italian". In particolare, le istanze testate sono le numero 1, 3, 9, 12, 20 e 29, selezionate per la loro rappresentatività e variabilità in termini di complessità, anche se si è cercato di non eccedere nelle dimensioni per questioni di tempo. Queste istanze forniscono un banco di prova rigoroso per valutare le prestazioni del sistema sviluppato e confrontarlo con i risultati ottenuti dal gruppo di ricerca dell'università.

Come è possibile osservare di seguito, i risultati dell'esperimento sono inferiori rispetto a quelli ottenuti dal gruppo di ricerca dell'Università degli Studi di Udine. Tuttavia, è importante notare che l'algoritmo ALNS ha

Istanze di Valutazione

Istanza	P	S	C	Double service ratio [%]	Time windows avi [min]	Distances avg [min]	Radius [Km]
1	165	4	24	41,8	117,7	23,61	19
3	44	8	8	43,2	106,4	22,58	19
9	55	4	7	27,3	115,9	21,55	15
12	130	4	21	24,6	123,7	38,09	37
20	78	4	12	46,2	127,7	20,10	15
29	100	4	11	3,0	115,5	32,87	21

Figura 4.1: Istanze di prova

dimostrato di essere un metodo promettente per affrontare questo tipo di problemi. Nonostante le performance non ottimali, l'algoritmo ha mostrato capacità di adattamento e flessibilità nella risoluzione di istanze complesse, confermando il suo potenziale come strumento per l'ottimizzazione nel contesto del problema HHCRSP.

Confronto risultati

Istanza	Distanza Percorsa	Ritardo Totale	Ritardo Massimo	Costo	Distanza Percorsa	Ritardo Totale	Ritardo Massimo	Costo
1	3985	3244	413	2547,333	2536	9	4	849,667
3	1221	664	221	702,000	1095	1	1	365,667
9	1066	333	98	499,000	888	3	2	297,667
12	3252	2319	127	1899,333	2709	18	8	911,667
20	1604	1187	206	999,000	1279	7	7	431,000
29	2542	2091	183	1605,333	1791	32	11	611,333

Figura 4.2: Confronto risultati della prova

Conclusioni

Il lavoro svolto in questa tesi ha esplorato l'implementazione e l'applicazione dell'algoritmo ALNS (Adaptive Large Neighborhood Search) per il problema del routing e scheduling nell'assistenza domiciliare (HHCRSP). Nonostante gli sforzi profusi, i risultati ottenuti non sono ancora all'altezza di quelli raggiunti dal gruppo di ricerca dell'Università di Udine, con un algoritmo differente (AVNS Adaptive Variable Neighborhood Search), evidenziando così la necessità di ulteriori miglioramenti nel sistema attuale.
Una delle principali aree di miglioramento riguarda il sistema di training utilizzato per ottimizzare l'algoritmo. Un sistema di training più robusto e sofisticato è essenziale per facilitare la selezione dei parametri più adeguati, garantendo così prestazioni migliori. Per iniziare, è cruciale esaminare attentamente e ottimizzare alcuni parametri chiave:

- Numero di elementi distrutti ad ogni iterazione: La determinazione del numero ottimale di elementi da rimuovere e reinserire è fondamentale per mantenere un equilibrio tra l'esplorazione dello spazio delle soluzioni e la stabilità della soluzione corrente.

- Numero di iterazioni: Stabilire un numero appropriato di iterazioni è vitale per consentire all'algoritmo di convergere verso una soluzione ottimale senza incorrere in un eccessivo tempo di computazione.

- Numero di segmenti: La suddivisione dell'algoritmo in segmenti e la gestione dei punteggi delle euristiche all'interno di questi segmenti può

influire significativamente sull'efficacia dell'algoritmo nell'adattarsi e migliorare nel tempo.

Inoltre, ulteriori miglioramenti potrebbero includere l'integrazione di tecniche avanzate di machine learning per l'ottimizzazione automatica dei parametri e l'implementazione di nuove strategie euristiche per diversificare maggiormente la ricerca. Un'analisi più approfondita dei casi in cui l'algoritmo non riesce a fornire soluzioni ottimali potrebbe anche fornire preziose indicazioni su come affinare ulteriormente il sistema.
In conclusione è evidente che, sebbene i risultati attuali siano promettenti, c'è ancora molto lavoro da fare per raggiungere l'efficienza e l'efficacia desiderate. Migliorare il sistema di training e ottimizzare i parametri chiave rappresentano i primi passi fondamentali verso l'ottenimento di prestazioni superiori. Con poco lavoro in più e hardware più potente, è possibile colmare il divario con i risultati del gruppo di ricerca dell'Università di Udine e avanzare significativamente nella risoluzione del problema HHCRSP.

4.1 Sviluppi futuri

Come sviluppo futuro, si prevede di implementare un servizio multithreading per distribuire il carico di lavoro su diversi processori, migliorando così l'efficienza e l'affidabilità dell'algoritmo ALNS. Tuttavia, non tutte le parti dell'algoritmo sono parallelizzabili, quindi sarà necessario prestare attenzione nella progettazione per garantire che le sezioni del codice che possono beneficiare del multithreading siano identificate e ottimizzate correttamente. Questa implementazione richiederà un'analisi dettagliata delle dipendenze e della sincronizzazione tra i thread per evitare problemi di concorrenza e garantire l'integrità delle soluzioni generate.
Inoltre, per un utilizzo reale e pratico dell'algoritmo in contesti operativi, sarebbe essenziale avere una fonte di input formattata correttamente, come il formato JSON proposto, o integrarlo con un database (DB) nel caso fosse parte di un sistema software più ampio. Questa integrazione faciliterebbe la

gestione dei dati, migliorando l'interoperabilità e la scalabilità del sistema, rendendolo più adatto alle esigenze delle applicazioni reali.
Infine, si potrebbe pensare ad un modello capace di adattare le distanze (espresse in tempo) a seconda delle variazioni di traffico.

62

Bibliografia

[1] Mankowska, D.S., Meisel, F. & Bierwirth, C. The home health care routing and scheduling problem with interdependent services, Health Care Manag Sci 17, pp 15-30, 2014.

[2] David Bredstrom, Mikael Ronnqvist, Combined vehicle routing and scheduling with temporal precedence and synchronization constraints, *European Journal of Operational Research*, 191(1), pp. 19-31, 2008.

[3] Ropke, Stefan & Pisinger, David. An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows, Transportation Science, 2006.

[4] Ceschia, Sara & Di Gaspero, Luca & Rosati, Roberto & Schaerf, Andrea, Multi-Neighborhood Simulated Annealing for the Home Healthcare Routing and Scheduling Problem, 2024

[5] Fikar, Christian & Hirsch, Patrick, Home Health Care Routing and Scheduling: A Review, Computers & Operations Research 77, pp 86-95, 2017.

[6] Natallia Kokash, An introduction to heuristic algorithms, in *Research Methodology course*, Trento, June 2005.

[7] Blum Christian & Roli Andrea. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison, 2001.

[8] Leonora Bianchi, Marco Dorigo, Luca Maria Gambardella & Walter J. Gutjahr. A survey on metaheuristics for stochastic combinatorial optimization, Springer Science, 2008

[9] Blazinskas, Andrius; Misevicius, Alfonsas. Combining 2-OPT, 3-OPT and 4-OPT with K-SWAP-KICK perturbations for the traveling salesman problem, 17th International Conference on Information and Software Technologies, 2011.

[10] G. A. CROES (1958). A method for solving traveling salesman problems. Operations Res. 6 1958, pp., 791-812, 1958.

[11] Kirkpatrick, S., Gelatt Jr, C. D., & Vecchi, M. P. Optimization by simulated annealing. Science, 220(4598), 671-680, 1983.

[12] B Suman & P Kumar. A survey of simulated annealing as a tool for single and multiobjective optimization, Journal of the Operational Research Society 57, pp. 1143-1160, 2006.

[13] David Pisinger & Stefan Ropke. Large Neighborhood Search, International Series in Operations Research & Management Science 272, pp. 99-127, 2019.

[14] Ropke, Stefan & Pisinger, David. An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows, Transportation Science, 40, pp. 455-472, 2006.

[15] Vera C. Hemmelmayr, JeanFrancois Cordeau, Teodor Gabriel Crainic. An adaptive large neighborhood search heuristic for Two-Echelon Vehicle Routing Problems arising in city logistics, Computers & Operations Research, 39, pp. 3215-3228, 2012.

[16] Attila A Kovacs, Sophie N. Parragh, Karl F. Doerner, Richard F. Hartl, Adaptive large neighborhood search for service technician routing and scheduling problems, Springer Science, LLC, 2011.

[17] Mualla Gonca, Mustafa Avci. An adaptive large neighborhood search approach for multiple traveling repairman problem with profits, Computers and Operations Research, 111, pp. 367-385, 2019.