Transport Efficiency Manager

# Constraint Satisfaction Problem per la formulazione di un programma di corse efficiente

Progetto di Fondamenti di Intelligenza Artificiale

| | |
|---|---|
| Data | 18/03/2021 |
| Destinatario | Prof. Fabio Palomba<br>Prof. Fabio Narducci |
| Presentato da | Team F$^3$<br>Federica Pica 0512103077<br>Federica Attianese 0512105719<br>Francesca Moschella 0512105977 |

# Indice

1. Contesto ..... 3
2. Idea iniziale ..... 4
3.Problema ..... 4
**3.1 Panoramica ..... 4**
**3.2 Formulazione del problema ..... 4**
4. Ricerca locale ..... 5
**4.1 Motivazioni ..... 5**
**4.2 Nuova formulazione del problema ..... 5**
5. Una prima soluzione: algoritmo genetico ..... 6
**5.1 Breve panoramica sugli algoritmi genetici ..... 6**
***5.1.1 Principali definizioni ..... 6***
***5.1.2 Principio di funzionamento ..... 6***
**5.2 Setup dell'algoritmo genetico (scelta dei parametri) ..... 7**
**5.3 Perché cambiare rotta ..... 8**
6. Soluzione definitiva: algoritmo Constraint Satisfaction Problem ..... 9
**6.1 Breve panoramica sui problemi CSP ..... 9**
***6.1.1 Struttura formale del problema ..... 9***
***6.1.2 Algoritmi di risoluzione ..... 9***
**6.2 Setup dell'algoritmo CSP ..... 10**
***6.2.1 Definizione delle variabili ..... 10***
***6.2.2 Ricerca con backtracking ..... 13***
***6.2.3 Propagazione dei vincoli ..... 13***
***6.2.4 Implementazione ..... 14***
7.Esempio dimostrativo ..... 18
**7.1 Risorse e Dati corsa dell'azienda ..... 18**
**7.2 Assegnamenti per la variabile Orario ..... 21**
**7.3 Assegnamenti per la variabile Conducente ..... 23**
**7.4 Assegnamenti per la variabile Mezzo ..... 29**
8. Conclusioni ..... 36
9. Fonti e riferimenti ..... 36

# 1. Contesto

Transport Efficiency Manager è un progetto che nasce dalla necessità di supportare le aziende di trasporti nella memorizzazione e gestione delle risorse disponibili, quali conducenti, mezzi e linee, oltre che nella generazione del programma delle corse. Questa seconda funzionalità viene offerta in due modalità: manuale, ovvero a cura dell'azienda stessa, e automatica, tramite un modulo di intelligenza artificiale.

Nello specifico, tramite la web-app, l'addetto incaricato, dopo aver registrato la propria azienda, dovrà inserire tutte le risorse disponibili. Dopo l'inserimento sarà possibile, in ogni momento, visualizzare, modificare o eliminare ognuna di esse.

Inoltre, potrà inserire dati specifici relativi a singole corse: questi dati vengono raccolti dall'autista al termine di ogni corsa e consegnati all'addetto che provvede all'inserimento degli stessi nel sistema e, anche in questo caso, avrà la possibilità in qualsiasi momento di visualizzare, modificare o eliminare i dati.

La generazione del programma, come anticipato, è disponibile in due modalità:

- **Generazione manuale**: In questo caso, l'addetto crea in totale autonomia il programma delle corse, associando ad ogni corsa l'orario e i 3 tipi di risorse, ovvero: una linea, uno o più conducenti ed uno o più mezzi. La limitazione sta nel fatto che l'addetto deve tener conto da sé dei vincoli temporali e spaziali delle risorse, oltre al fatto che non dispone di informazioni empiriche da cui capire se ogni corsa inserita convenga o meno all'azienda. Risulta quindi difficile evitare sprechi e fornire un servizio totalmente soddisfacente ai passeggeri.
- **Generazione automatica**: In questo caso la generazione avviene tramite un modulo di intelligenza artificiale; inoltre entrano in gioco i dati delle singole corse, molto utili a tener traccia, sulla base delle corse provate, del numero stimato di passeggeri e dell'eventuale presenza di traffico. In pratica, l'addetto potrà disporre di un programma completo ed efficiente, semplicemente cliccando un bottone.

# 2. Idea iniziale

Durante la prima fase di progettazione, l'idea di soluzione sviluppata in principio e derivata dalla non totale conoscenza della materia e quindi da una visione incompleta rispetto a quella acquisita al termine del corso di Fondamenti di Intelligenza Artificiale, era una previsione dei passeggeri e del traffico tramite un algoritmo di apprendimento. Strutturando così il modulo, però, il supporto alle aziende sarebbe stato minimo: le stime e le previsioni sarebbero state d'aiuto per migliorare l'efficienza del programma proposto, ma la creazione vera e propria sarebbe stata comunque un compito quasi totalmente riversato nelle mani dell'addetto. Il team ha deciso, quindi, di formulare il tutto come un problema di ricerca.

# 3.Problema

### 3.1 Panoramica

Gli agenti intelligenti devono massimizzare la loro misura di prestazione. Un agente basato su obiettivi conosce la descrizione dello stato corrente, oltre a informazioni riguardanti il suo obiettivo (goal), ovvero una situazione che desidera raggiungere.
Un agente risolutore di problemi è un particolare tipo di agente basato su obiettivi, che considera gli stati del mondo come entità atomiche, senza struttura interna visibile agli algoritmi. Questo tipo di agente lavora su un ambiente osservabile (per conoscere sempre lo stato corrente), noto (per sapere quali stati sono raggiungibili da ciascuna azione, deterministico (per ogni azione esiste uno ed un solo risultato), discreto (per avere un numero finito di azioni in qualunque stato).
Sotto queste ipotesi, si riduce a una sequenza fissata di azioni la soluzione di qualsiasi problema, ovvero un cammino che porta da uno stato iniziale a uno finale. Per fare ciò, però, il problema deve essere ben formalizzato.

### 3.2 Formulazione del problema

Una prima formulazione del problema in atto, tenendo conto che ogni stato rappresenta un programma delle corse più o meno compilato (a partire da quello vuoto fino ad arrivare, corsa dopo corsa, a quello completo) è la seguente:

- Stato iniziale: il programma delle corse è vuoto;
- Azioni possibili: aggiungere una corsa, rimuovere una corsa, utilizzare un bus più capiente, utilizzare un bus meno capiente, spostare l'orario di mezz'ora;
- Modello di transizione: restituisce il programma modificato, ovvero con una corsa in più o in meno o con una modifica relativa all'orario o al bus utilizzato;

- Test obiettivo: il programma delle corse è ottimizzato al meglio delle possibilità, ovvero permette di coprire gli spostamenti necessari della giornata utilizzando, per ogni corsa, l'autobus meno capiente capace di ospitare il numero di passeggeri previsti.

Gli algoritmi che risolvono un problema così strutturato, esplorano sistematicamente lo spazio degli stati fino a raggiungere uno stato obiettivo: considerando, quindi, interi cammini dallo stato iniziale a quello obiettivo. La soluzione restituita è il cammino compiuto.

# 4. Ricerca locale

## 4.1 Motivazioni

Per quanto riguarda il problema in questione, il cammino che porta dallo stato iniziale a quello obiettivo è irrilevante: possiamo, quindi, considerare come soluzione lo stato obiettivo, ovvero il programma delle corse efficiente e già compilato.

Il team ha deciso, per questo motivo, di considerare la classe degli algoritmi di **ricerca locale**, ovvero quegli algoritmi che ignorano il cammino che porta alla soluzione. Per farlo, lo spazio degli stati rappresenta l'insieme delle configurazioni complete, ovvero quelle che portano alla risoluzione del problema. Partendo da ciò, questi algoritmi considerano, e mantengono in memoria, solo lo stato corrente e cercano, ad ogni iterazione, di migliorarlo finché è possibile.

La principale differenza tra gli algoritmi di ricerca locale e quelli di ricerca semplice, precedentemente citati, sta nel fatto che, ovviamente, il test obiettivo non è più presente: subentra una **funzione obiettivo** a rendere efficace la ricerca nello spazio degli stati, calcolando matematicamente la bontà della soluzione considerata e indicando, ad ogni iterazione, se e quanto quella stessa soluzione è stata migliorata.

## 4.2 Nuova formulazione del problema

Data la definizione di ricerca locale, è evidente la necessità di una modifica della formulazione del problema. Mentre le azioni possibili e il modello di transizione restano gli stessi, definiamo gli altri punti:

- **Stato iniziale**: tutte le possibili combinazioni di corse che rendono il programma pieno;
- **Funzioni obiettivo**:

  f(n): massimizza il numero di passeggeri, mantenendo basso il numero di posti vuoti.

  g(n): minimizza il traffico.

Occorre far presente che, per poter definire la funzione obiettivo, il team ha notato che un programma efficiente, può essere definito tale se riesce ad accontentare più passeggeri possibili senza spreco di risorse, quindi non solo utilizzando pullman con una capienza tale da non lasciare

nessuno a piedi, ma anche evitando troppi posti lasciati vuoti durante le corse: questo lo si può ottenere tramite la funzione f(n). Inoltre, il programma è reso ancora più efficiente, se evita il traffico: questo lo si può ottenere tramite la funzione g(n).
Pertanto, il problema risulta essere un problema multi-obiettivo.

# 5. Una prima soluzione: algoritmo genetico

Tra i vari algoritmi di ricerca locale proposti a lezione, il team ha posto l'attenzione sugli algoritmi genetici. Oltre ad essere algoritmi molto flessibili e adatti a problemi multi-obiettivo, hanno quel tocco in più dato dal fascino dell'ispirazione alla natura e al processo di evoluzione. Inoltre, riescono a trovare buone soluzioni in modo rapido. Da notare; è stato utilizzato il termine "buone" e non "ottime", in quanto non sempre l'ottimo è raggiungibile da questo tipo di algoritmi poiché, simulando la legge del più forte e non partendo sempre dalla totalità degli individui, tendono a portare avanti l'individuo che sembra il migliore in quel momento, senza considerare soluzioni che potrebbero essere migliori.

## 5.1 Breve panoramica sugli algoritmi genetici

### 5.1.1 Principali definizioni

- **Spazio di ricerca (search space):** rappresenta l'insieme delle possibili soluzioni o valori che un algoritmo genetico può assumere;
- **Popolazione**: insieme ben definito di soluzioni, che vengono chiamate individui. Ogni soluzione è formata da N geni.
- **Individuo**: è identificato, solitamente, come una stringa finita che definisce una soluzione al problema. (anche detto cromosoma)
- **Gene**: Singolo valore della stringa di un individuo. A seconda dei diversi valori effettivamente assunti dai diversi geni, si ottiene un diverso individuo e, quindi, una soluzione differente.
- **Generazione** (o iterazione): si intende l'introduzione nella popolazione di nuovi individui, i figli.

### 5.1.2 Principio di funzionamento

Gli algoritmi genetici partono da una popolazione iniziale di soluzioni e la fanno evolvere iterativamente. Ad ogni iterazione, le soluzioni sono valutate da una funzione di fitness e sulla base di questa valutazione, vengono selezionate alcune di esse, privilegiando le soluzioni (genitori) con fitness maggiore. Le soluzioni selezionate vengono tra loro ricombinate per generare nuove soluzioni (figli) che tendono a trasmettere le buone caratteristiche delle soluzioni genitori nelle successive generazioni.
Lo schema generale di un algoritmo genetico è il seguente:

1. Codifica delle soluzioni dello specifico problema.
2. Creazione di un insieme iniziale di soluzioni (popolazione iniziale).
3. Calcolo della funzione di fitness;
4. Iterativamente, fino alla soddisfazione di un criterio di arresto:

   a) Selezione: alcuni individui vengono copiati nella successiva generazione; ispirandosi al principio di sopravvivenza del più forte, solitamente, la scelta viene effettuata tramite il valore della funzione di fitness.

   b) Crossover: gli individui si accoppiano per crearne altri da aggiungere alla nuova generazione; ciò avviene ispirandosi all'incrocio del corredo genetico.

   c) Mutazione: vengono modificate alcune parti del corredo genetico; ciò avviene perché i primi due operatori tendono a portare avanti delle caratteristiche che sembrano ottimali ma non lo sono, quindi tramite questo operatore si cerca di esplorare un po' di più lo spazio degli stati.

   d) Rinnovo della popolazione e calcolo della funzione di fitness sui nuovi individui.
5. Viene restituita la migliore soluzione generata.

Partendo da questo schema generale, la scelta dei parametri per ogni operazione modella il funzionamento dell'algoritmo genetico rispetto ad uno specifico problema.

## 5.2 Setup dell'algoritmo genetico (scelta dei parametri)

Innanzitutto, bisogna notare che pensare agli individui come combinazioni di corse (e quindi di programmi già totalmente riempiti), non è adatto in questa situazione: bisogna aumentare la granularità per ottenere un risultato più corretto e specifico.

Inoltre, non viene considerato il conducente, in quanto non è rilevante ai fini dell'efficienza; il conducente verrà poi scelto random tra i disponibili ed aggiunto alla corsa definita dall'algoritmo.

L'algoritmo genetico pensato dal team comincia prelevando tutti i dati delle singole corse inseriti per l'azienda, raggruppandoli, poi, per linea e successivamente per fascia oraria (range di 60 minuti). Il raggruppamento per linea è una scelta che viene fatta per comodità, ovvero, per ridurre la lunghezza delle stringhe degli individui: rappresentare anche qualche cifra in più che sta ad indicare la linea sarebbe stato inutile, in quanto non avrebbe influito sul miglioramento della corsa stessa.

**Inizializzazione**: tutti quegli individui che soddisfano il requisito capienza>persone attese. Questa scelta è stata fatta per scartare a priori le soluzioni inconsistenti.

**Size della popolazione**: è variabile e dipende dal criterio della selezione.

**Rappresentazione degli individui:** Ogni individuo è codificato in una stringa di 6 cifre.

In particolare:

- le prime 4 cifre rappresentano l'orario nel quale si è tenuta la corsa (es: 0800 per una corsa effettuata alle 08:00);
- la quinta cifra, inserita casualmente, può assumere valore 0 (la corsa mantiene il suo orario di partenza) o 1 (la corsa viene posticipata di 30 minuti per evitare il traffico);
- la sesta cifra rappresenta la tipologia di pullman utilizzata, in base alla sua capienza (le possibili tipologie sono: Minibus da 15 posti, rappresentato dalla cifra 2; Autobus da 35 posti, rappresentato dalla cifra 3; Pullman**S** da 40 posti, rappresentato dalla cifra 4; Pullman**M** da 55 posti, rappresentato dalla cifra 5; Pullman**L** da 85 posti, rappresentato dalla cifra 6.)

**Crossover:** metodo single point: le cifre dell'orario restano invariate, c'è accoppiamento solo per le ultime due cifre.

**Mutazione:** l'orario resta invariato, la cifra del traffico viene mutata con il metodo bit-flip per un numero random di individui, la cifra del pullman viene mutata per un numero random di individui, inserendo una tipologia random di pullman tra quelle disponibili.

**Criterio di terminazione:** Si itera per al più tre volte senza miglioramenti; questa scelta è stata fatta perché le combinazioni possibili traffico/pullman per ogni corsa sono in totale 10, quindi 3 iterazioni senza miglioramento risultano accettabili per il team.

**Funzione di preferenza:** p(n): definisce un ordinamento totale dato dalla preferenza di ottimizzazione della funzione f(n); questo perché preferiamo mantenere la corsa allo stesso orario, anche a costo di trovare traffico, piuttosto che lasciare passeggeri a piedi o troppi posti vuoti per evitare il traffico.

### 5.3 Perché cambiare rotta

L'algoritmo genetico sopracitato risolve, effettivamente, il problema della generazione di un programma delle corse a partire dai dati inseriti. La scelta del team di spostare l'attenzione verso un altro tipo di algoritmo deriva dal fatto che la soluzione ottenuta fosse a scopo puramente didattico e poco realistico. Non è possibile, infatti, controllare:

- la disponibilità delle risorse;
- dove si trova un pullman sulla base della destinazione della corsa precedente;
- dove si trova un conducente sulla base della destinazione della corsa precedente;
- se un pullman è ancora in viaggio;
- se un conducente è ancora in viaggio.

Questo perché l'algoritmo crea e valuta combinazioni come se le risorse fossero illimitate e disponibili in ogni dove e ad ogni ora. Volendo rendere la soluzione più realistica sarebbe stato

necessario inserire altre funzioni di fitness per valutare la bontà delle soluzioni, portando alla decisione di impostarlo come un problema di soddisfacimento di vincoli.

# 6. Soluzione definitiva: algoritmo Constraint Satisfaction Problem

Nei problemi di ricerca standard finora considerati, ogni stato viene sempre considerato come una rappresentazione atomica e la soluzione al problema rappresenta sempre una ricerca nello spazio degli stati, valutando la bontà di ognuno di essi.

Nel contesto di problemi di soddisfacimento dei vincoli, si vanno a rilassare il concetto di rappresentazione atomica e di conseguenza, necessariamente, anche di test obiettivo, in quanto non sarà più possibile calcolare la bontà di uno stato semplicemente valutandolo dall'esterno.

Si parla, dunque, di una fattorizzazione degli stati: ognuno di essi è, in questo contesto, rappresentato da n variabili, ognuna avente un dominio di valori che può assumere. Per quanto riguarda, invece, il test obiettivo, il problema è considerato risolto nel momento in cui ad ogni variabile viene assegnato un valore legale, ovvero che soddisfa tutti i vincoli su di essa.

Uno dei vantaggi di questo tipo di contesto è che risulta essere una rappresentazione naturale di un gran numero di problemi reali. In questo modo, i risolutori di problemi CSP risultano spesso utilizzabili nell'ambito di più problemi.

## 6.1 Breve panoramica sui problemi CSP

### 6.1.1 Struttura formale del problema

Un problema di soddisfacimento dei vincoli è costituito da tre componenti:

X: insieme di variabili, $\{X_1, \dots, X_n\}$

D: insieme di domini, $\{D_1, \dots, D_n\}$, uno per ogni variabile

C: insieme di vincoli che specificano combinazioni di valori ammesse.

Ogni dominio $D_i$ è costituito da un insieme di valori ammessi, $\{v_1, \dots, v_k\}$, per la variabile $X_i$. Ogni vincolo $C_i$ è costituito da una coppia <ambito, rel>, dove *ambito* è una tupla di variabili che partecipano nel vincolo e *rel* è una relazione che definisce i valori che tali variabili possono assumere.

### 6.1.2 Algoritmi di risoluzione

Gli algoritmi di ricerca CSP sono formulati cercando di sfruttare la nuova idea di stato, andando quindi ad analizzare la struttura interna di ognuno di essi. In questo modo è possibile sfruttare euristiche di uso generale anziché trovarne di specifiche del problema.

Il principio alla base è quello di eliminare contemporaneamente delle porzioni dello spazio di ricerca, trovando (e non prendendo poi in considerazione) le combinazioni variabile/valore che

violano i vincoli stabiliti. In questo modo, si riduce drasticamente lo spazio di ricerca e si velocizza la ricerca della soluzione al problema. Questa idea rende gli algoritmi CSP molto efficienti.
La risoluzione di un problema CSP consiste nella definizione di uno spazio degli stati, ognuno di essi definito dall'assegnamento di valori ad alcune o tutte le variabili.
La buona definizione dei vincoli si rende necessaria per permettere all'algoritmo di trovare una soluzione **completa** (ogni variabile viene valorizzata) e **consistente** (non viene violato nessun vincolo).

## 6.2 Setup dell'algoritmo CSP

### 6.2.1 Definizione delle variabili

Per la risoluzione del problema in questione, viene definita una variabile per ogni informazione relativa ad una singola corsa da "comporre". Vengono, quindi, prese in considerazione le seguenti variabili:

- Linea
- Traffico
- Passeggeri attesi
- Andata/Ritorno
- Orario
- Conducente
- Mezzo

È opportuno notare che le variabili Linea, Traffico, Passeggeri Attesi e Andata/Ritorno hanno tutte un dominio contenente un unico valore, in quanto derivano (o come copia, o come risultato di qualche calcolo) dai dati delle corse già provate dall'azienda. Sarebbe inutile includerle nell'algoritmo per questioni di complessità temporale. Inoltre, considerarle come variabili andrebbe a violare la commutatività del problema, caratteristica fondamentale per un problema CSP.
La linea scelta per la corsa sarà equivalente alla linea provata e di cui si dispone di dati specifici su cui basarsi per trovare un'alternativa più efficiente; il traffico avrà valore "sì" se è stato riscontrato almeno nella metà delle corse provate; il valore di passeggeri attesi sarà la media dei passeggeri che si sono presentati nelle corse provate (il numero di passeggeri di cui viene calcolata la media è dato dalla somma dei passeggeri che sono effettivamente saliti sulle corse provate e quelli che, per mancanza di posti, non sono riusciti ad effettuare la corsa); il valore di andata/ritorno è lo stesso delle corse provate. A monte di ciò, quindi, le variabili del problema risultano essere 3:

- X {Orario, Conducente, Mezzo}

Per ognuna delle variabili viene definito un dominio, ovvero l'insieme di valori che possono essere assunti:

- $D_{ORARIO}$ = {Orario già provato, Orario già provato posticipato di 30 minuti}
- $D_{CONDUCENTE}$ = {Tutti i conducenti dell'azienda}
- $D_{MEZZO}$ = {Tutti i mezzi dell'azienda}

Definiamo ora i vincoli:

- $C_1$= {<orario>, (traffico no, orario già provato), (traffico sì, orario posticipato)}

  Con questo vincolo si cerca di risolvere il problema del traffico. Sono elencate le due possibili combinazioni: se non c'è stato traffico si mantiene l'orario provato, se c'è stato traffico, si posticipa la corsa di 30 minuti. Si sottolinea che posticipare la corsa di 30 minuti non garantisce di riuscire ad evitare il traffico al 100%; il team ha, però, ritenuto che posticipare di un tempo minore di 30 quasi sicuramente non avrebbe aiutato ad evitarlo, posticipare, invece, di un tempo maggiore di 30 minuti avrebbe cambiato di troppo i connotati della corsa, facendo diminuire il numero di persone attese (questo perché, se il pullman viene preso in un determinato orario per andare a lavoro o ad un appuntamento importante, posticipare di troppo porterebbe i passeggeri a scegliere un'altra compagnia che ha una corsa in quell'orario).

- $C_2$={<conducente>, (mai utilizzato), (utilizzato && destinazione precedente=partenza corrente)}

  Con questo vincolo si tengono in considerazione le limitazioni spaziali di un conducente. In pratica, se il conducente non ha ancora effettuato alcuna corsa, può essere assegnato, qualunque sia la sua posizione di partenza; se, invece, il conducente ha già effettuato una corsa, lo si può assegnare solo ad un'altra la cui posizione di partenza sia uguale al posto in cui si è fermato al termine della corsa precedente. Questo permette anche di ottimizzare l'"utilizzo" dei conducenti, evitando periodi di inattività dovuti agli spostamenti necessari da un luogo ad un altro per poter compiere il proprio lavoro.

- $C_4$={<mezzo>, (mai utilizzato && capienza minima >= passeggeri attesi), (utilizzato && capienza minima >= passeggeri attesi && destinazione precedente = partenza corrente)}

  Con questo vincolo, oltre a tenere in considerazione le limitazioni spaziali di un mezzo (nel modo descritto precedentemente per il conducente), si cerca di accontentare più passeggeri possibili senza, però, sprecare posti: in pratica, tenendo conto dei passeggeri attesi, viene scelto il pullman meno capiente (se possibile) tra quelli la cui capienza è maggiore o uguale al numero di passeggeri attesi.

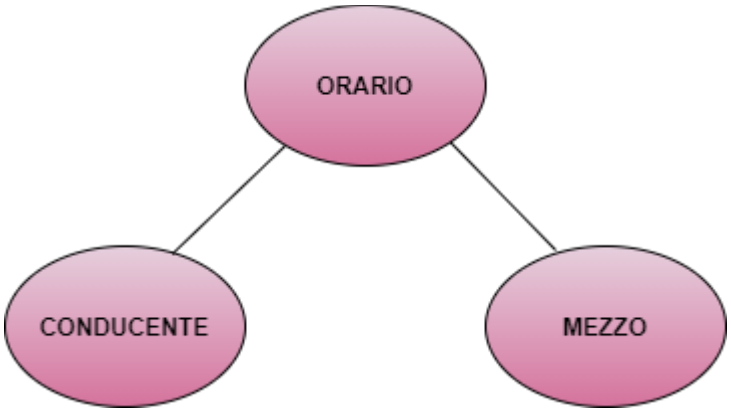
- $C_3$={<orario, conducente>, (orario, conducente mai utilizzato), (orario, conducente utilizzato && conducente soddisfa vincolo $C_2$ && orario corrente >= orario della precedente corsa del conducente + durata)}

Con questo vincolo si tengono in considerazione le limitazioni temporali del conducente. In pratica, se il conducente non ha ancora effettuato una corsa, è disponibile a qualsiasi orario; nel caso contrario, invece, l'orario di partenza della corsa corrente deve necessariamente essere maggiore o uguale alla partenza della precedente corsa sommata alla sua durata (informazione reperibile dalla linea), oltre a tenere in considerazione il vincolo $C_2$.

- $C_5$={<orario, mezzo>, (orario, mezzo mai utilizzato && mezzo soddisfa vincolo $C_4$), (mezzo utilizzato && mezzo soddisfa vincolo $C_4$ && orario corrente >= orario della precedente corsa del mezzo + durata)}

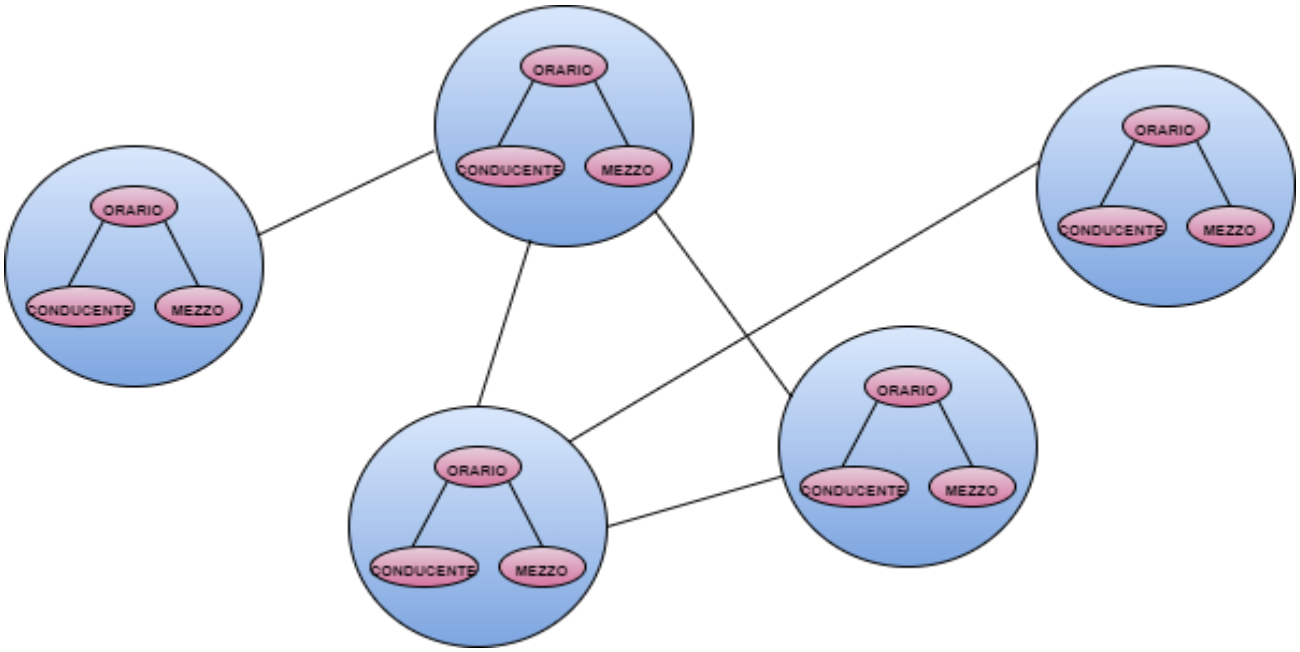
Con questo vincolo si tengono in considerazione le limitazioni temporali del mezzo (nel modo descritto precedentemente per il conducente), oltre a tenere conto del soddisfacimento del vincolo $C_4$.

Segue una rappresentazione del problema come **grafo di vincoli**.




Ogni nodo del grafo rappresenta una variabile del problema; ogni arco connette una coppia di variabili che partecipano ad un vincolo.

È opportuno notare che, senza tenere conto del precedente utilizzo di una determinata risorsa, la soluzione al problema si riduce al trovare un assegnamento alle variabili di una singola corsa. Volendo estendere il contesto, per poter soddisfare i vincoli che tengono conto dell'"utilizzato" e del "non utilizzato" e per rendere il progetto più completo e vicino al mondo reale, bisogna ampliare la visione delle cose: la soluzione rappresenta, a questo punto, la creazione dell'intero programma giornaliero delle corse, includendo iterativamente l'assegnamento di tutte le singole corse in esso contenute. Fatta questa considerazione, il grafo dei vincoli risulta espanso e con una diversa granularità: le variabili diventano le singole corse, ognuna delle quali è valorizzata dalle 3 variabili sopracitate.




Si tenga presente, inoltre, che questo grafo è rappresentativo solo in maniera generale in quanto non è possibile definire, senza disporre dei dati delle corse provate per la singola azienda, quante corse sarà necessario valorizzare, né quali tra di loro avranno un arco che le collega.

### 6.2.2 Ricerca con backtracking

La ricerca con backtracking è, sostanzialmente, una ricerca in profondità per un problema di soddisfacimento di vincoli in cui ogni assegnamento è fatto su singole variabili in maniera iterativa; ciò significa che l'algoritmo assegna valori ad una variabile per volta tornando, però, indietro quando, seguendo quella strada, non ci sono più valori legali da assegnare.

Questo tipo di ricerca rappresenta l'algoritmo di ricerca non informata di base per la risoluzione di problemi di soddisfacimento dei vincoli.

La decisione di assegnare singole variabili volta per volta incide significativamente sull'efficienza: oltre a ridurre l'albero di ricerca, questo tipo di ricerca trova una soluzione ottima (a differenza del sub-ottimo della ricerca in profondità) ma potrebbe richiedere più tempo. In conclusione, è la soluzione migliore per il problema che si sta considerando in quanto il fine ultimo (non solo del modulo di intelligenza artificiale ma di tutto il sistema) è supportare la creazione di un programma che sia il più efficiente possibile.

### 6.2.3 Propagazione dei vincoli

Per soddisfare i vincoli riguardanti la "dipendenza" delle variabili conducente e mezzo con la variabile orario, il team ha deciso di mettere in atto la **propagazione dei vincoli**: essa è un'inferenza che utilizza i vincoli per ridurre il numero di valori legali per una variabile, che a sua

volta può ridurre i valori legali per un'altra variabile e così via. È stato anche deciso di intrecciare la propagazione con la ricerca perché in essa non è contenuto il controllo per l'assegnazione del valore alla variabile orario, che viene settato precedentemente alla sua chiamata; si serve però di esso per propagare i vincoli. Inoltre, un valore assegnato non influenza tutti gli assegnamenti successivi, ma solo alcuni e con l'inferenza alternata alla ricerca è possibile, ogni volta che viene scelto un valore per una variabile, avere un'opportunità totalmente nuova di inferire riduzioni di dominio sulle variabili adiacenti.

La propagazione dei vincoli garantisce la **consistenza d'arco**. Una variabile è **arco-consistente** se ogni valore del suo dominio soddisfa i suoi vincoli binari. In altre parole, la variabile $X_i$ è arco-consistente rispetto ad un'altra variabile $X_j$ se per ogni valore nel dominio corrente $D_i$ c'è un valore nel dominio $D_j$ che soddisfa il vincolo binario sull'arco $(X_i, X_j)$. Il metodo di propagazione implementato rende la variabile Conducente di una corsa arco-consistente rispetto alla variabile Conducente delle corse precedenti, tenendo conto anche della variabile Orario; allo stesso modo, rende la variabile Mezzo di una corsa arco-consistente rispetto alla variabile Mezzo delle corse precedenti, tenendo conto anche della variabile Orario. In poche parole, riduce la lista dei conducenti e dei mezzi da considerare controllando, sulla base dell'orario scelto, se una determinata risorsa risulta ancora in viaggio.

In particolare, la strategia di propagazione dei vincoli scelta prende il nome di **forward checking** (anche detta verifica in avanti). Ogni volta che una variabile X è assegnata, il processo di verifica in avanti stabilisce la consistenza d'arco per essa: per ogni variabile non assegnata Y collegata a X da un vincolo, "cancella" dal dominio di Y ogni valore non consistente con quello scelto per X.

### 6.2.4 Implementazione

In questa sezione viene descritta l'implementazione del modulo.

Il codice in questione può essere reperito al link contenuto nella sezione "Fonti e riferimenti".

L'algoritmo di intelligenza artificiale è contenuto nella classe *ProgrammaAutomaticoMaker* e viene lanciato tramite il metodo *doOperation()*; questo metodo è la conseguenza dell'implementazione dell'interfaccia Strategy, in quanto la funzionalità di creazione dei programmi è gestita attraverso il design pattern Strategy. Questo pattern permette di definire più strategie che portano allo stesso risultato e di scegliere una di esse a run-time. Ovviamente, in Transport Efficiency Manager, le strategie sono due: creazione manuale e creazione automatica, gestita tramite l'intervento dell'intelligenza artificiale.

Innanzitutto, partendo da una serie di dati corsa inseriti dell'azienda si cerca di ottenere una base iniziale di dati da modellare attraverso l'intelligenza artificiale per ottenere un programma ottimale. I primi dati grezzi ottenuti non sono altro che il frutto di una query effettuata sul database che raggruppa i dati delle corse già provate per linea, orario, andata/ritorno. Per ogni

"gruppo" viene creato un oggetto di tipo *DatiGenerazione* e tutti questi oggetti coesistono nella lista *listaDatiGenerazione*. Ognuno di essi ha come attributi principali Linea, Traffico, Numero di passeggeri attesi, Andata/Ritorno, Orario, Conducente, Mezzo. I primi, come detto in precedenza, vengono settati sulla base di calcoli fatti sui dati corsa appartenenti al gruppo in questione. Per dare il valore alle tre variabili, si utilizza la strategia descritta di seguito.

Per rendere il lavoro più chiaro e lineare, il team ha deciso di dividere i casi delle 3 variabili: tenendo conto del grafo più completo che rappresenta l'intero programma giornaliero, ad ognuna di esse, in momenti indipendenti l'uno dall'altro, viene assegnato un valore, tenendo conto del valore dato alle precedenti corse.

In primis, viene considerata la variabile Orario. In questo caso non si ricorre all'algoritmo di backtracking, in quanto l'orario di ogni corsa è indipendente da quello delle altre. Iterativamente, per ogni oggetto *DatiGenerazione*, viene assegnato un orario alla corsa tramite un metodo che permette di controllare se l'assegnamento dato è legale o meno.

- Il metodo *checkOrario()* prende in input, oltre all'oggetto che rappresenta la corsa in questione, uno dei due possibili orari appartenenti al dominio. È un metodo *boolean* e restituisce *true* solo nei due casi descritti nel vincolo $C_1$.

In secundis, viene considerata la variabile Conducente. Prima di esplicitare l'algoritmo di backtracking, è bene capire il funzionamento dei metodi di controllo sui valori legali e delle strutture dati utilizzate per metterli in atto:

- L'oggetto *illegalValuesConducenti* è di tipo *ArrayList<ArrayList<Object>>*: in sostanza, è una lista di liste contenente i conducenti che sono in viaggio in un certo range di tempo. Questa lista si rende necessaria per sapere se un determinato valore del dominio dei conducenti è legale o meno al momento della partenza della corsa che si sta componendo. Ogni lista contenuta nella lista principale contiene 3 campi: il conducente in viaggio, l'orario in cui è partito (dato dall'orario della corsa a cui è stato precedentemente assegnato) e l'orario in cui sarà nuovamente disponibile (dato dall'orario della corsa a cui è stato precedentemente assegnato a cui viene sommata la durata della linea della stessa corsa); gli ultimi due campi prendono, rispettivamente, i nomi *inizioRange* e *fineRange*.
- Il metodo *checkConducente()* prende in input l'oggetto di tipo *DatiGenerazione* che rappresenta la corsa in questione, il conducente assegnato in una iterazione e la lista *illegalValuesConducenti* contenente i valori non legali del dominio dei conducenti. Controlla, innanzitutto, se il conducente è disponibile o meno, scorrendo la lista dei valori illegali. Se non trova un'occorrenza di quel conducente nella lista degli illegali, va avanti; se, invece, la trova, controlla se l'orario di partenza della corsa che si sta componendo è posteriore al range temporale del viaggio precedentemente effettuato.

Dopodiché, controlla se il suddetto conducente ha già effettuato una corsa scorrendo all'indietro la lista *listaDatiGenerazione* partendo dalla posizione precedente a quella della corsa che si sta componendo. Nel caso in cui viene trovata una occorrenza lo scorrimento termina (in quanto bisogna fare il confronto solo con l'ultima corsa effettuata da quel conducente) e si controlla che il luogo di destinazione della precedente corsa coincida con il luogo di partenza della corsa che si sta popolando. Si noti, inoltre, che la partenza e la destinazione di ogni linea dipendono dal valore andata/ritorno dell'oggetto relativo alla stessa corsa a cui appartengono: se è un viaggio di andata, queste restano invariate, se è un viaggio di ritorno vengono invertite.

È un metodo *boolean* e restituisce *true* solo se: la corsa in questione è la prima della lista (quindi il conducente è sicuramente disponibile); se il conducente in questione è un valore legale da assegnare, ovvero se non è contenuto nella lista degli illegali o se, pur essendo nella lista degli illegali, il range di quell'occorrenza è terminato e quindi il conducente è nuovamente disponibile; se, dopo i controlli precedenti, l'assegnamento rispetta il vincolo $C_2$.

- Il metodo *forwardConducente()* prende in input l'oggetto *DatiGenerazione* che rappresenta la corsa che si sta componendo, il conducente assegnato (se ha superato il check), la lista *illegalValuesConducenti* contenente i valori non legali del dominio dei conducenti sulla base dei range orari e la lista totale dei conducenti dell'azienda.

  Questo metodo è utilizzato per mettere in atto la propagazione dei vincoli: innanzitutto aggiunge alla lista *illegalValuesConducenti* una lista contenente il valore del conducente appena assegnato, oltre all'inizio e la fine del range orario in cui il conducente rimarrà in viaggio. Dopodiché, scorre la lista degli oggetti rappresentanti le corse a partire dalla posizione successiva a quella che si sta componendo, fino all'ultima corsa il cui orario è compreso nel range del viaggio del conducente, ovvero quelle il cui dominio avrà ripercussioni, dovendo fare a meno di un conducente; non va avanti fino alla fine in quanto le corse sono ordinate per orario crescente e quindi, appena si trova una corsa che non sarà pregiudicata, allora non lo saranno neanche quelle successive.

  Per ognuna delle corse sopracitate, il metodo prende in considerazione l'orario di partenza e lo confronta con tutti i range contenuti nella lista degli illegali: se il numero di occorrenze per cui l'orario è compreso in qualche range è pari al numero di conducenti, significa che per quella corsa il dominio è vuoto. In questo caso, il metodo restituisce un fallimento e comunica all'algoritmo di backtracking di assegnare immediatamente un nuovo valore alla variabile Conducente, evitandogli ulteriori esplorazioni su quella strada che porterà di sicuro a tornare indietro.

È un metodo *boolean* e restituisce *true* solo se nessuna, tra le corse successive, vedrà svuotato il proprio dominio a causa dell'assegnamento appena effettuato.

Veniamo ora al punto cruciale, l'algoritmo di backtracking, che combina tutti gli elementi appena descritti al fine di ottenere un assegnamento legale per la variabile Conducente di ogni corsa.

- Il metodo *ricercaBacktrackingConducente()* prende in input la lista totale dei conducenti dell'azienda e la lista dei valori illegali nei rispettivi range.

  Innanzitutto, scorre la lista degli oggetti *DatiGenerazione* relativi alle corse da comporre e controlla, per ognuno di essi, se la variabile Conducente ha un valore o è *null*. Se a tutte le corse è stato assegnato un Conducente, allora significa che è stata trovata una soluzione, altrimenti si "posiziona" sul primo Conducente non assegnato per trovargli un valore:
  tramite un ciclo for sulla lista totale dei conducenti, ci si focalizza su uno di essi; per prima cosa, se il conducente "supera" il metodo *checkConducente()*, questo viene settato per la corsa che si sta componendo. Dopodiché si attua la propagazione dei vincoli tramite il metodo *forwardConducente()*. Se il conducente "supera" anche questo step, viene richiamato ricorsivamente il metodo *ricercaBacktrackingConducente()*, per continuare gli assegnamenti tenendo in considerazione anche quello appena inserito: se questa chiamata ha esito *true*, l'algoritmo è andato a buon fine e ha trovato una soluzione; in caso contrario, ovvero se la chiamata ricorsiva ha esito *false*, si resetta l'assegnamento dato e si elimina dalla lista *illegalValuesConducenti* l'ultima riga inserita, ovvero quella contenente il range in cui il conducente non sarebbe stato disponibile se fosse stato assegnato alla corsa. A questo punto la ricerca può ripartire focalizzandosi sul conducente successivo a quello che ha portato al fallimento.

  È un metodo *boolean* e restituisce *true* se riesce a trovare un assegnamento completo e consistente, *false* altrimenti.

Per ultima, viene considerata la variabile Mezzo. Il trattamento dato è lo stesso della variabile Conducente, tramite i metodi *checkMezzo*, *forwardMezzo*, *ricercaBacktrackingMezzo*. L'unica aggiunta si ha nel metodo di check: il primo controllo effettuato è sulla capienza, che deve essere maggiore o uguale al numero di passeggeri attesi.

Una breve nota aggiuntiva riguarda la lista dei mezzi dell'azienda: quest'ultima viene precedentemente ordinata per valore crescente di capienza; questa operazione si rende necessaria per avere la certezza di selezionare sempre il pullman (con valore legale) di capienza minore tra quelli di capienza maggiore al numero di passeggeri attesi: ciò permette di non sprecare posti vuoti che potrebbero far comodo in un'altra corsa in cui è atteso un più alto numero di passeggeri.

Se, sulla base dei dati corsa inseriti, entrambe le ricerche backtracking restituiscono esito positivo (l'assegnamento dell'Orario è sempre garantito, non può avere esito *false*), il metodo *doOperation()*

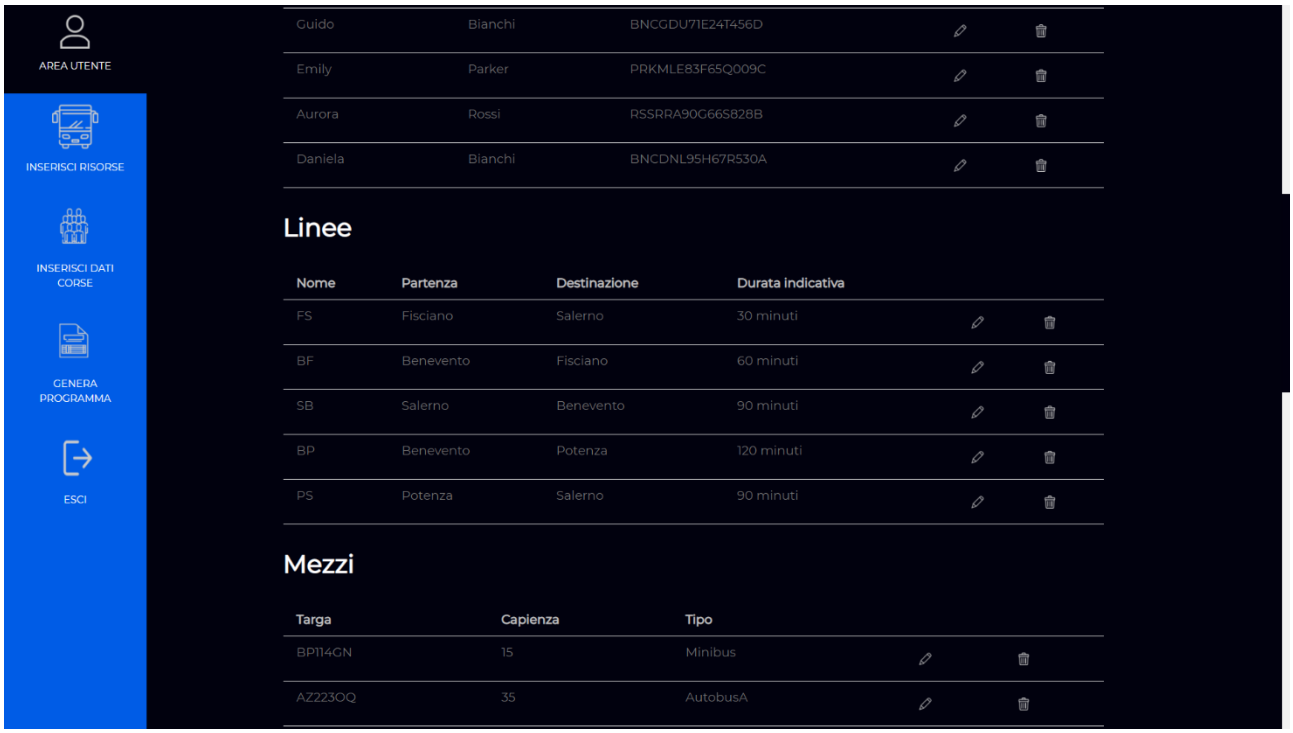
che gestisce il tutto, provvede iterativamente a convertire gli oggetti *DatiGenerazione* in vere e proprie corse e ad aggiungerle al programma di riferimento in cui sono state specificate dall'azienda, prima di richiamare la generazione automatica, le date di inizio e fine validità dello stesso.

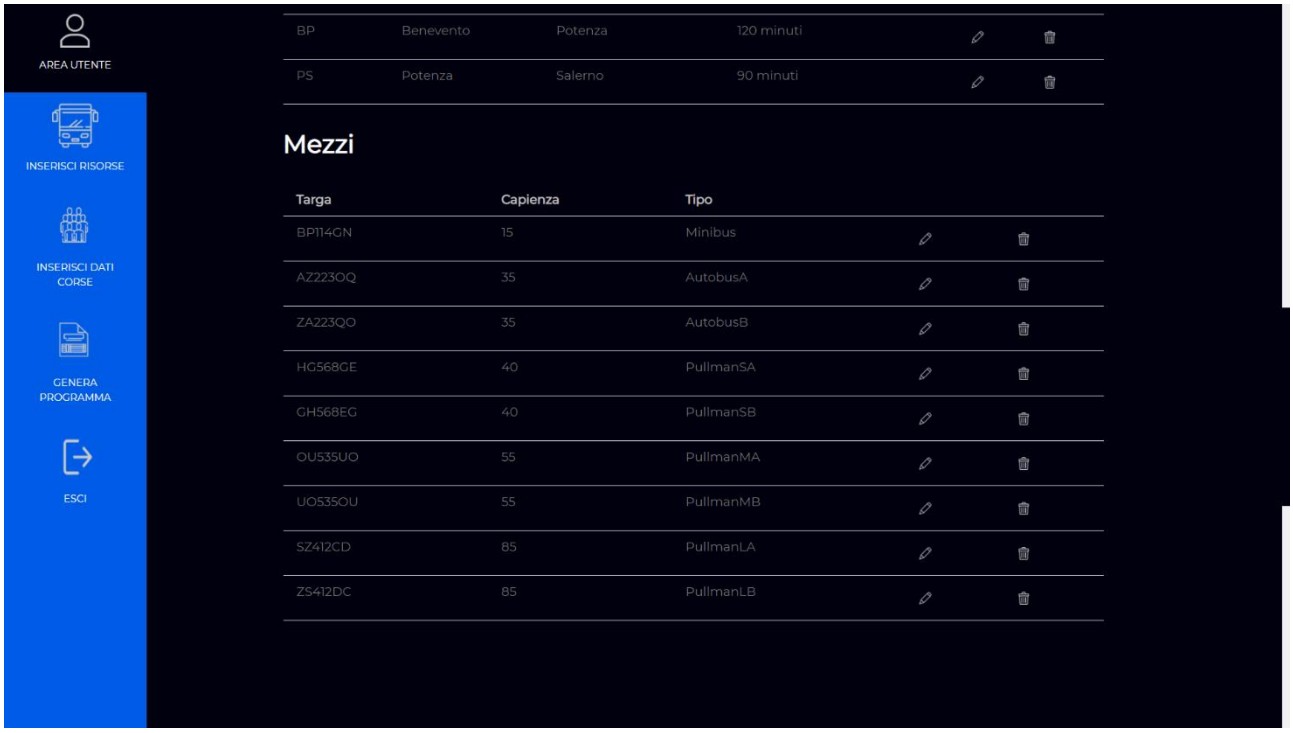
# 7.Esempio dimostrativo

Dopo aver descritto il funzionamento di ciò che è stato progettato, in questa sezione viene fornito un esempio per dimostrarne la corretta implementazione. Sulla base di dati iniziali, il team ha trovato una soluzione "immedesimandosi" nell'agente, dopodiché i risultati ottenuti mentalmente per ogni fase sono stati confrontati con quelli effettivamente prodotti dall'algoritmo tramite delle stampe a video.

## 7.1 Risorse e Dati corsa dell'azienda

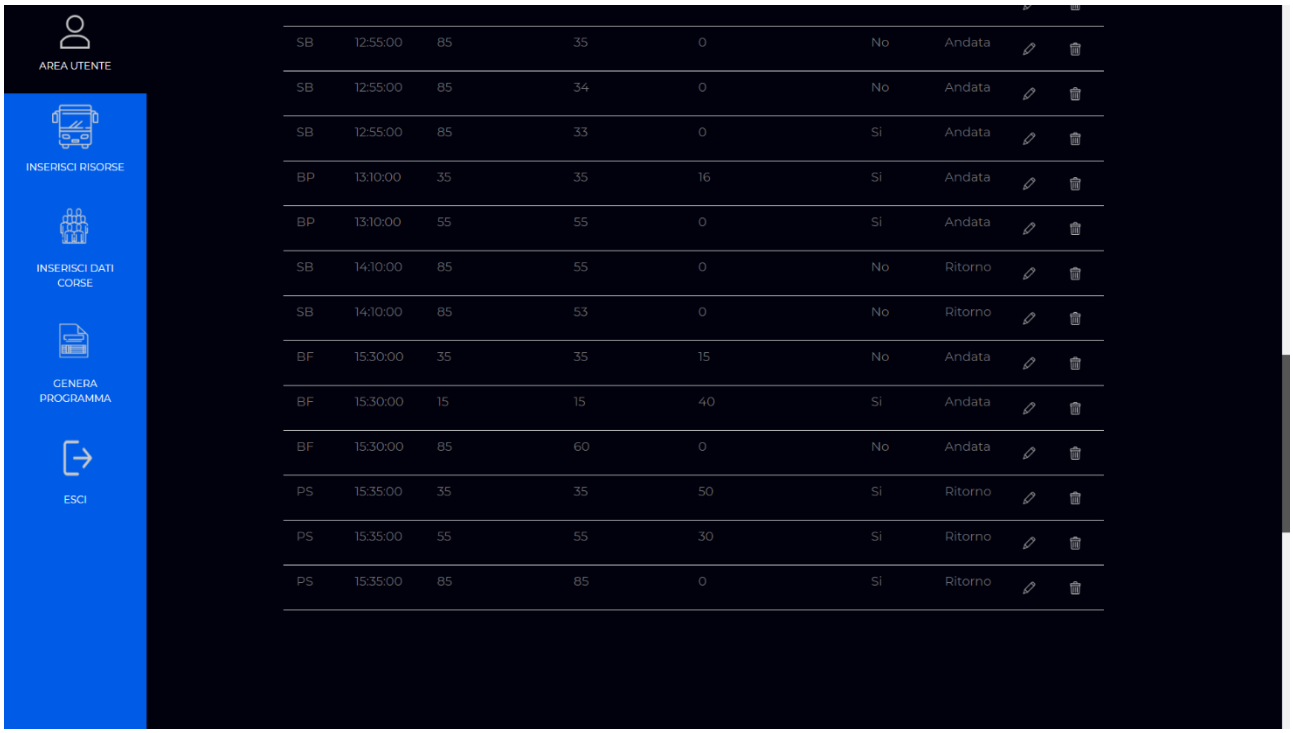
Di seguito sono riportate le risorse inserite dall'azienda IntArtificiale s.r.l..




Di seguito sono riportati i dati delle singole corse di prova effettuate dall'azienda IntArtificiale s.r.l. nei giorni precedenti alla generazione del programma.


AREA UTENTE
INSERISCI RISORSE
INSERISCI DATI CORSE
GENERA PROGRAMMA
ESCI
Transport Efficiency Manager
Lista dati corse
Linea Orario Posti disponibili Posti occupati Passeggeri non saliti Traffico
FS 08:00:00 40 40 15 Si Andata
FS 08:00:00 40 40 20 No Andata
FS 08:00:00 40 39 0 Si Andata
BF 08:50:00 15 15 3 No Andata
BF 08:50:00 15 14 0 No Andata
BF 08:50:00 55 5 0 No Andata
SB 09:30:00 85 50 0 No Andata
SB 09:30:00 85 52 0 No Andata
SB 09:30:00 85 51 0 Si Andata
BP 09:00:00 35 35 1 No Andata
BP 09:00:00 40 38 0 No Andata
BP 09:00:00 35 22 0 No Andata
PS 10:00:00 55 30 0 No Andata
PS 10:00:00 55 25 0 No Andata
PS 10:00:00 55 33 0 No Andata
FS 10:00:00 35 35 40 No Ritorno
FS 10:00:00 35 35 30 Si Ritorno
BF 10:45:00 85 25 0 No Ritorno
BF 10:45:00 85 45 0 No Ritorno
SB 11:00:00 35 18 0 No Ritorno
SB 11:00:00 35 18 0 No Ritorno
BP 11:10:00 55 55 0 Si Ritorno
BP 11:10:00 85 45 0 Si Ritorno
FS 12:10:00 15 15 5 No Andata
FS 12:10:00 15 7 0 No Andata




I dati grezzi, in ordine di orario, ottenuti raggruppando per Linea, andata/ritorno, orario, sono i seguenti:

```
[ FS Si 51 08:00 null null true ]
[ BF No 12 08:50 null null true ]
[ BP No 32 09:00 null null true ]
[ SB No 51 09:30 null null true ]
[ PS No 29 10:00 null null true ]
[ FS Si 70 10:00 null null false ]
[ BF No 35 10:45 null null false ]
[ SB No 18 11:00 null null false ]
[ BP Si 50 11:10 null null false ]
[ FS No 14 12:10 null null true ]
[ SB No 34 12:55 null null true ]
[ BP Si 53 13:10 null null true ]
[ SB No 54 14:10 null null false ]
[ BF No 55 15:30 null null true ]
[ PS Si 85 15:35 null null false ]
```

Nella prima "colonna" è riportata la linea; nella seconda la presenza di traffico; nella terza il numero di passeggeri attesi; nella quarta l'orario che è stato provato; la quinta e la sesta rappresentano, rispettivamente, conducente e mezzo; nella settima il valore è *true* se si tratta di un viaggio di andata, *false* se, invece, si tratta di un viaggio di ritorno.

Si ricordi che è previsto traffico se almeno nel 50% delle corse di un determinato gruppo è stato effettivamente riscontrato; i passeggeri attesi, invece, sono frutto di una media dei passeggeri che si sono presentati durante le corse di un determinato gruppo.

È possibile, a questo punto, dare il via alla generazione.

### 7.2 Assegnamenti per la variabile Orario

Come primo step, si determina il valore dell'orario di ogni corsa del futuro programma, considerando la previsione del traffico precedentemente calcolata. Per comodità, il valore dell'orario è già settato ed è, inizialmente, uguale a quello del gruppo di corse provate; dopo aver eseguito il check, questo rimarrà uguale o sarà posticipato di 30 minuti.

- Nella prima iterazione, l'orario provato è 08:00 ed è previsto traffico, quindi il valore finale sarà 08:30;
- Nella seconda iterazione, l'orario provato è 08:50 e non è previsto traffico, quindi il valore finale sarà 08:50;
- Nella quarta iterazione, l'orario provato è 09:00 e non è previsto traffico, quindi il valore finale sarà 09:00;
- Si procede in questo modo fino all'ultima corsa.

```
0 iterazione:
Dominio: [08:00, 08:30]
C'è traffico, quindi 08:00 non è valido
C'è traffico, quindi 08:30 è valido
1 iterazione:
Dominio: [08:50, 09:20]
Non c'è traffico, quindi 08:50 è valido
Non c'è traffico, quindi 09:20 non è valido
2 iterazione:
Dominio: [09:00, 09:30]
Non c'è traffico, quindi 09:00 è valido
Non c'è traffico, quindi 09:30 non è valido
3 iterazione:
Dominio: [09:30, 10:00]
Non c'è traffico, quindi 09:30 è valido
Non c'è traffico, quindi 10:00 non è valido
4 iterazione:
Dominio: [10:00, 10:30]
C'è traffico, quindi 10:00 non è valido
C'è traffico, quindi 10:30 è valido
5 iterazione:
Dominio: [10:00, 10:30]
Non c'è traffico, quindi 10:00 è valido
Non c'è traffico, quindi 10:30 non è valido
6 iterazione:
Dominio: [10:45, 11:15]
Non c'è traffico, quindi 10:45 è valido
Non c'è traffico, quindi 11:15 non è valido
7 iterazione:
Dominio: [11:00, 11:30]
Non c'è traffico, quindi 11:00 è valido
Non c'è traffico, quindi 11:30 non è valido
8 iterazione:
Dominio: [11:10, 11:40]
C'è traffico, quindi 11:10 non è valido
C'è traffico, quindi 11:40 è valido
```

```
9 iterazione:
Dominio: [12:10, 12:40]
Non c'è traffico, quindi 12:10 è valido
Non c'è traffico, quindi 12:40 non è valido
10 iterazione:
Dominio: [12:55, 13:25]
Non c'è traffico, quindi 12:55 è valido
Non c'è traffico, quindi 13:25 non è valido
11 iterazione:
Dominio: [13:10, 13:40]
C'è traffico, quindi 13:10 non è valido
C'è traffico, quindi 13:40 è valido
12 iterazione:
Dominio: [14:10, 14:40]
Non c'è traffico, quindi 14:10 è valido
Non c'è traffico, quindi 14:40 non è valido
13 iterazione:
Dominio: [15:30, 16:00]
Non c'è traffico, quindi 15:30 è valido
Non c'è traffico, quindi 16:00 non è valido
14 iterazione:
Dominio: [15:35, 16:05]
C'è traffico, quindi 15:35 non è valido
C'è traffico, quindi 16:05 è valido
```

A questo punto, i dati vengono nuovamente riordinati per orario in quanto, nell'eventualità in cui fosse posticipata qualche corsa, l'ordine corretto potrebbe essere alterato, portando l'algoritmo a dare un risultato errato, non riuscendo a calcolare correttamente i dati.

### 7.3 Assegnamenti per la variabile Conducente

Come secondo step, si assegna un valore alla variabile Conducente di ogni futura corsa. In questo passo, bisogna tenere in considerazione la lista dei conducenti dell'azienda (riportata all'inizio di questa sezione) e la lista dei conducenti illegali, la quale viene popolata nel corso delle iterazioni. Si noti che i conducenti vengono considerati in ordine di inserimento, senza assegnare alcun tipo di priorità. Inoltre, l'algoritmo di backtracking considera il codice fiscale dei conducenti per evitare casi di omonimia.

- All'inizio della prima iterazione, la lista dei valori illegali è vuota.

  Poiché si scorre la lista delle corse in ordine di orario, la prima corsa in cui verrà trovata la variabile Conducente senza valore è la prima: linea FS (Fisciano, Salerno), viaggio di andata e orario 08:30.

  Si considera il primo conducente: Federica Pica. Questo conducente viene assegnato alla corsa senza effettuare tutti i controlli del check, in quanto è la prima corsa da popolare e non c'è bisogno di controllare se il conducente è in viaggio o se si trovi in un luogo diverso.

  A questo punto, il conducente "passa" nella funzione forward.

Vengono inseriti, nella lista degli illegali, i seguenti dati: Federica Pica, 08:30, 09:00, ovvero il nome del conducente assegnato e il range di tempo in cui non sarà disponibile (l'orario di fine range è dato dall'orario di partenza a cui viene sommata la durata della linea in questione, cioè 30 minuti).

Questo assegnamento non rende vuoto nessun dominio dei conducenti per le corse successive il cui orario di partenza è compreso nel range, quindi tutto è andato a buon fine e si può procedere richiamando la funzione di backtracking sulla lista delle corse aggiornata.

```
iterazione
la prima corsa trovata senza valore alla variabile Conducente è la numero 0:[ FS Si 51 08:30 null null true ]
FedericaPica
CHECK
OK: è la prima corsa, non c'è bisogno di ulteriori controlli
il valore scelto viene assegnato alla variabile: [ FS Si 51 08:30 PCIFDR95A60H501H null true ]
FORWARD
il conducente Pica non sarà disponibile dalle 08:30 alle 09:00
OK: nessun dominio delle corse successive è stato svuotato
Lista degli illegali aggiornata:
[Federica Pica, 08:30, 09:00]
iterazione
```

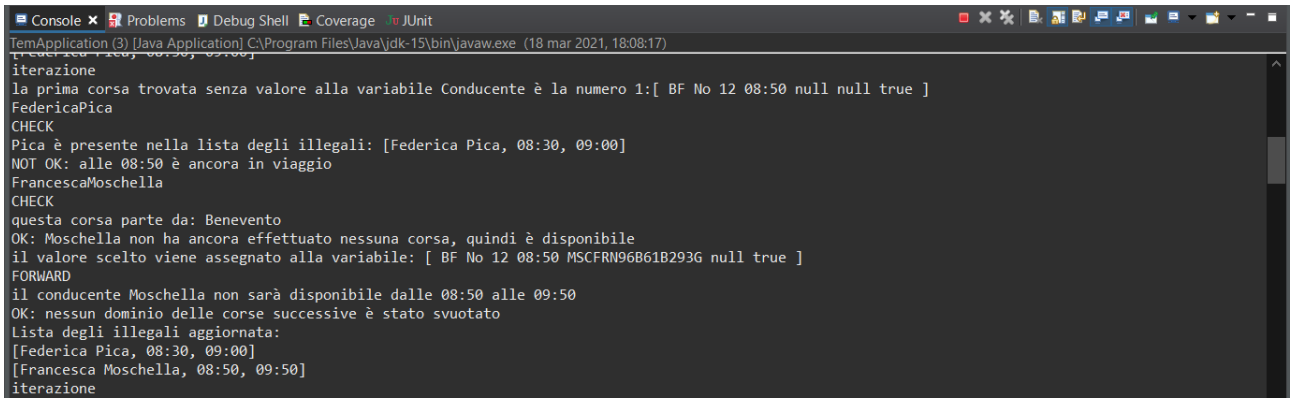
- Alla seconda iterazione, la prima corsa in cui verrà trovata la variabile Conducente senza valore è la seconda: linea BF (Benevento, Fisciano), viaggio di andata e orario 08:50.

  Si considera il primo conducente: Federica Pica. Questo conducente non è disponibile, in quanto ancora in viaggio.

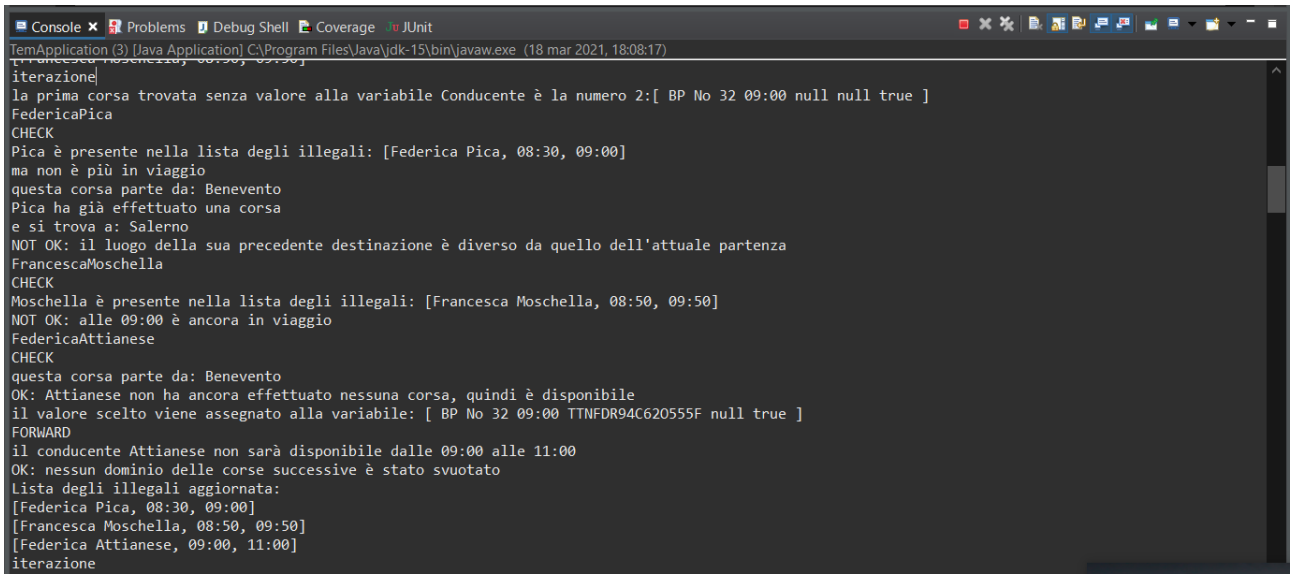
  Si considera il secondo conducente: Francesca Moschella. Questo conducente non ha alcun range in cui risulta illegale e non ha ancora effettuato alcuna corsa, dunque è disponibile per qualunque luogo di partenza; il valore viene assegnato alla variabile.

  Vengono inseriti, nella lista degli illegali, i seguenti dati: Francesca Moschella, 08:50, 09:50, ovvero il nome del conducente assegnato e il range di tempo in cui non sarà disponibile (l'orario di fine range è dato dall'orario di partenza a cui viene sommata la durata della linea in questione, cioè 60 minuti).

  Questo assegnamento non rende vuoto nessun dominio dei conducenti per le corse successive il cui orario di partenza è compreso nel range, quindi tutto è andato a buon fine e si può procedere richiamando la funzione di backtracking sulla lista delle corse aggiornata.




- Alla terza iterazione, la prima corsa in cui verrà trovata la variabile Conducente senza valore è la terza: linea BP (Benevento, Potenza), viaggio di andata e orario 09:00.
  Si considera il primo conducente: Federica Pica. Questo conducente ha terminato il viaggio precedente, quindi sarebbe disponibile, ma nella precedente corsa si è fermato a Salerno, dunque non può partire da Benevento.
  Si considera il secondo conducente: Francesca Moschella. Questo conducente non è disponibile in quanto è ancora in viaggio.
  Si considera il terzo conducente: Federica Attianese. Questo conducente non ha alcun range in cui risulta illegale e non ha ancora effettuato alcuna corsa, dunque è disponibile per qualunque luogo di partenza; il valore viene assegnato alla variabile.
  Vengono inseriti, nella lista degli illegali, i seguenti dati: Federica Attianese, 09:00, 11:00, ovvero il nome del conducente assegnato e il range di tempo in cui non sarà disponibile (l'orario di fine range è dato dall'orario di partenza a cui viene sommata la durata della linea in questione, cioè 120 minuti).
  Questo assegnamento non rende vuoto nessun dominio dei conducenti per le corse successive il cui orario di partenza è compreso nel range, quindi tutto è andato a buon fine e si può procedere richiamando la funzione di backtracking sulla lista delle corse aggiornata.

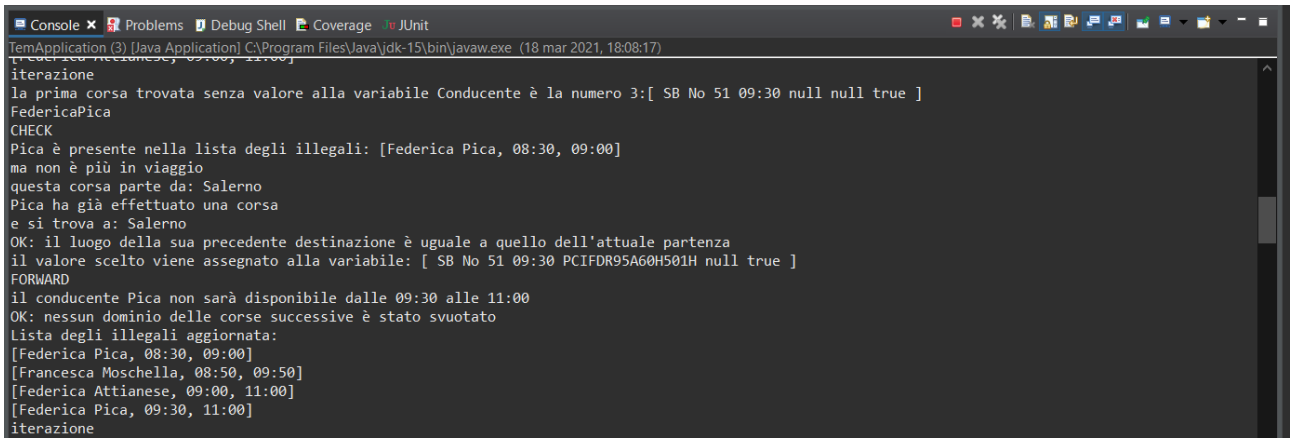



- Alla quarta iterazione, la prima corsa in cui verrà trovata la variabile Conducente senza valore è la quarta: linea SB (Salerno, Benevento), viaggio di andata e orario 09:30.

  Si considera il primo conducente: Federica Pica. Questo conducente ha terminato il viaggio precedente a Salerno, quindi risulta disponibile sia come orario che come luogo.

  Vengono inseriti, nella lista degli illegali, i seguenti dati: Federica Pica, 09:30, 11:00, ovvero il nome del conducente assegnato e il range di tempo in cui non sarà disponibile (l'orario di fine range è dato dall'orario di partenza a cui viene sommata la durata della linea in questione, cioè 90 minuti).

  Questo assegnamento non rende vuoto nessun dominio dei conducenti per le corse successive il cui orario di partenza è compreso nel range, quindi tutto è andato a buon fine e si può procedere richiamando la funzione di backtracking sulla lista delle corse aggiornata.




- Alla quinta iterazione, la prima corsa in cui verrà trovata la variabile Conducente senza valore è la quarta: linea PS (Potenza, Salerno), viaggio di andata e orario 10:00.

  Si considera il primo conducente: Federica Pica. Questo conducente è ancora in viaggio.

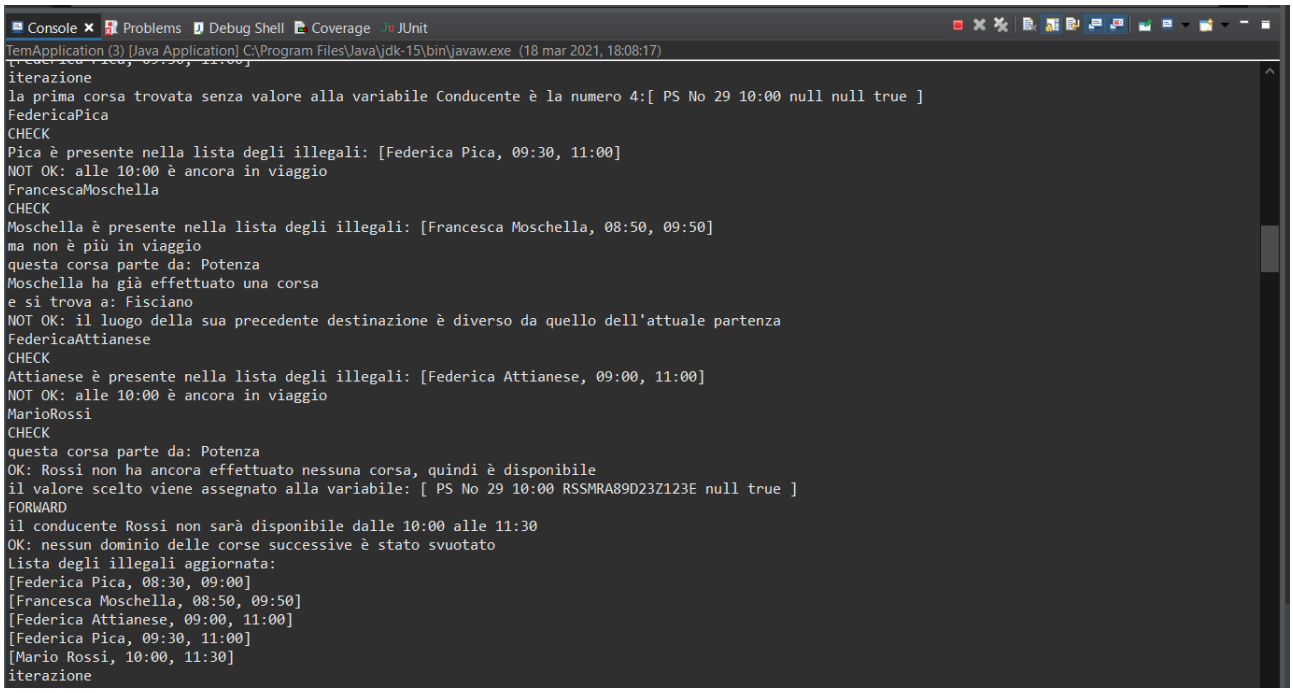
Si considera il secondo conducente: Francesca Moschella. Questo conducente ha terminato il viaggio precedente, quindi sarebbe disponibile, ma nella precedente corsa si è fermato a Fisciano, dunque non può partire da Potenza.

Si considera il terzo conducente: Federica Attianese. Questo conducente è ancora in viaggio.

Si considera il quarto conducente: Mario Rossi. Questo conducente non ha alcun range in cui risulta illegale e non ha ancora effettuato alcuna corsa, dunque è disponibile per qualunque luogo di partenza; il valore viene assegnato alla variabile.

Vengono inseriti, nella lista degli illegali, i seguenti dati: Mario Rossi, 10:00, 11:30, ovvero il nome del conducente assegnato e il range di tempo in cui non sarà disponibile (l'orario di fine range è dato dall'orario di partenza a cui viene sommata la durata della linea in questione, cioè 90 minuti).

Questo assegnamento non rende vuoto nessun dominio dei conducenti per le corse successive il cui orario di partenza è compreso nel range, quindi tutto è andato a buon fine e si può procedere richiamando la funzione di backtracking sulla lista delle corse aggiornata.




- Alla sesta iterazione, la prima corsa in cui verrà trovata la variabile Conducente senza valore è la sesta: linea FS (Fisciano, Salerno), viaggio di ritorno e orario 10:30. Si noti che, essendo un viaggio di ritorno, bisogna considerare invertiti i valori di partenza e destinazione.

  Si considera il primo conducente: Federica Pica. Questo conducente è ancora in viaggio.

Si considera il secondo conducente: Francesca Moschella. Questo conducente ha terminato il viaggio precedente, quindi sarebbe disponibile, ma nella precedente corsa si è fermato a Fisciano, dunque non può partire da Salerno.

Si considera il terzo conducente: Federica Attianese. Questo conducente è ancora in viaggio.

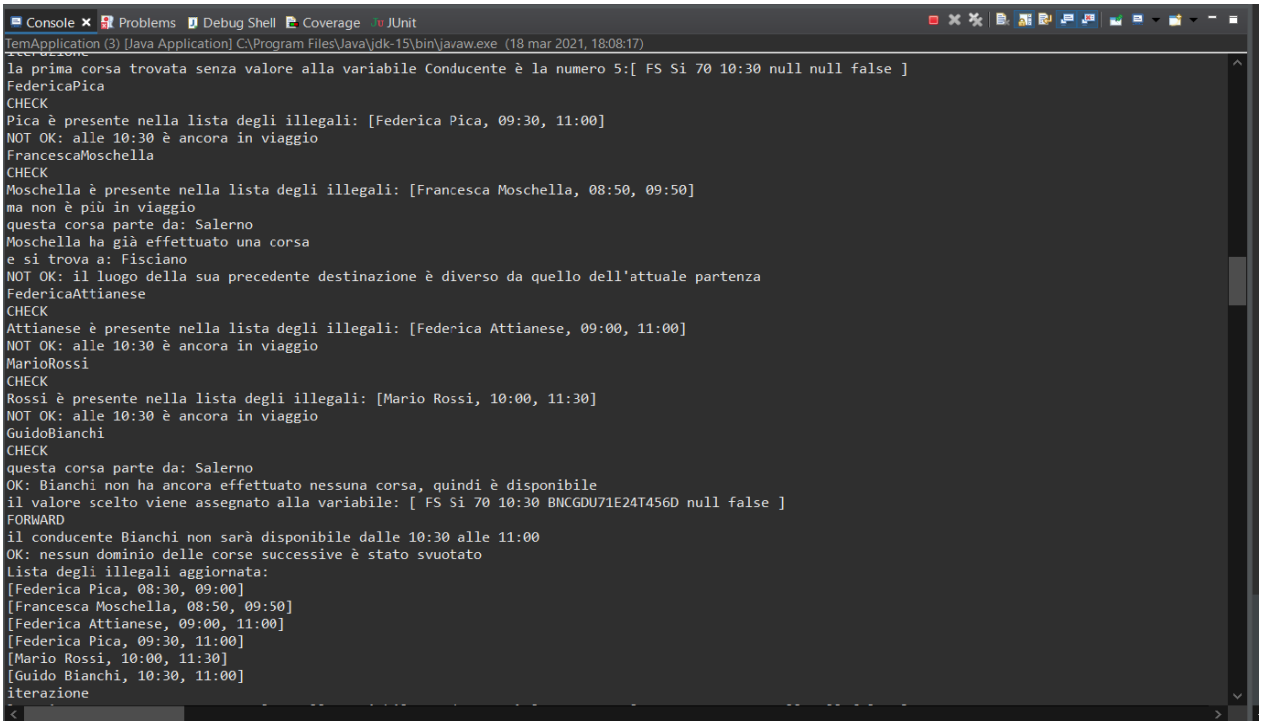
Si considera il quarto conducente: Mario Rossi. Questo conducente è ancora in viaggio.

Si considera il quinto conducente: Guido Bianchi. Questo conducente non ha alcun range in cui risulta illegale e non ha ancora effettuato alcuna corsa, dunque è disponibile per qualunque luogo di partenza; il valore viene assegnato alla variabile.

Vengono inseriti, nella lista degli illegali, i seguenti dati: Guido Bianchi, 10:30, 11:00, ovvero il nome del conducente assegnato e il range di tempo in cui non sarà disponibile (l'orario di fine range è dato dall'orario di partenza a cui viene sommata la durata della linea in questione, cioè 30 minuti).

Questo assegnamento non rende vuoto nessun dominio dei conducenti per le corse successive il cui orario di partenza è compreso nel range, quindi tutto è andato a buon fine e si può procedere richiamando la funzione di backtracking sulla lista delle corse aggiornata.

Si noti, inoltre, che, quando si considera un conducente che ha terminato il viaggio precedente ma non si trova nel luogo esatto, si passa direttamente al prossimo conducente senza continuare a scorrere la lista delle corse. Ciò è possibile in quanto è da considerarsi valida solo l'ultima corsa effettuata dal suddetto conducente, non è rilevante controllare quelle ancora precedenti.



- L'assegnamento dei valori alla variabile Conducente continua allo stesso modo fino al fallimento o al completamento delle assegnazioni.

### 7.4 Assegnamenti per la variabile Mezzo

Come terzo step, si assegna un valore alla variabile Mezzo di ogni futura corsa. In questo passo, bisogna tenere in considerazione la lista dei mezzi dell'azienda (riportata all'inizio di questa sezione) e la lista dei mezzi illegali, la quale viene popolata nel corso delle iterazioni. Si noti che i mezzi vengono considerati in ordine di capienza crescente, in modo da considerare sempre prima il più piccolo tra i mezzi con capienza maggiore del numero dei passeggeri attesi; in questo modo si evitano sprechi di posti vuoti. Per comodità, i mezzi inseriti per questo esempio sono già ordinati per capienza crescente. Inoltre, l'algoritmo di backtracking considera l'id dei mezzi per evitare casi di "omonimia".

Gli id non sono visibili nello screen inserito precedentemente, quindi sono riportati di seguito: Minibus id2, AutobusA id3, AutobusB id4, PullmanSA id5, PullmanSB id6, PullmanMA id7, PullmanMB id8, PullmanLA id9, PullmanLB id10.

- All'inizio della prima iterazione, la lista dei valori illegali è vuota.

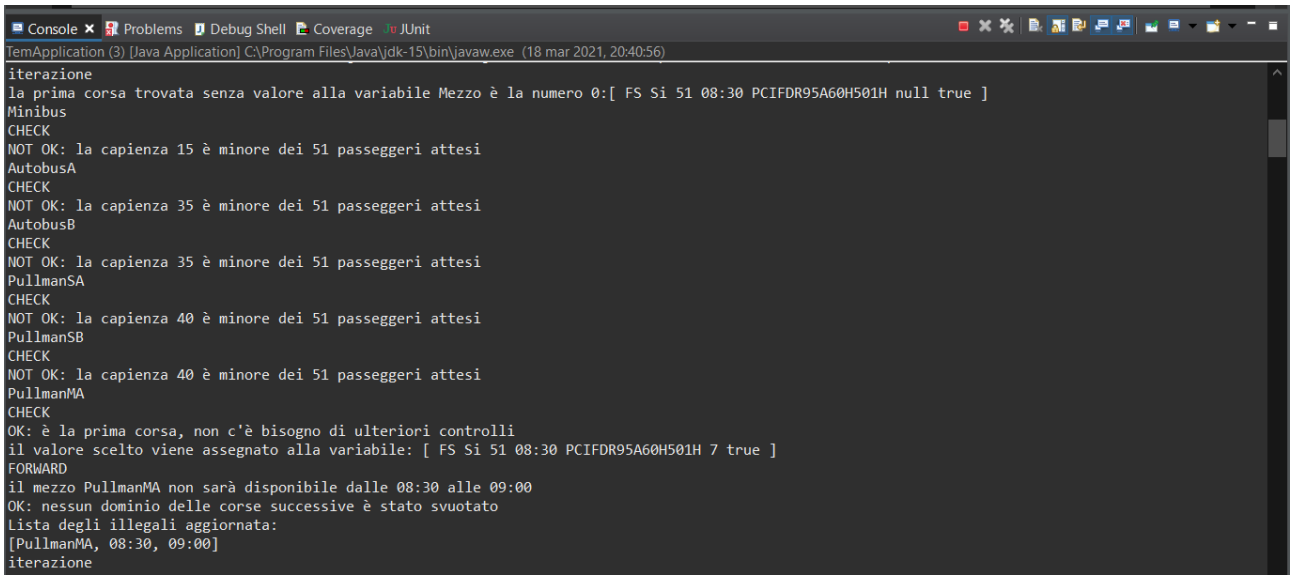
  Poiché si scorre la lista delle corse in ordine di orario, la prima corsa in cui verrà trovata la variabile Mezzo senza valore è la prima: linea FS (Fisciano, Salerno), viaggio di andata, orario 08:30, passeggeri attesi 51.

  Si scorre la lista dei mezzi fino a trovare il primo la cui capienza sia maggiore o uguale al numero dei passeggeri attesi: PullmanMA, con capienza 55. Questo mezzo viene assegnato alla corsa senza effettuare tutti i controlli del check, in quanto è la prima corsa da popolare e non c'è bisogno di controllare se il mezzo è in viaggio o se si trovi in un luogo diverso.

  A questo punto, il mezzo "passa" nella funzione forward.

  Vengono inseriti, nella lista degli illegali, i seguenti dati: PullmanMA, 08:30, 09:00, ovvero il nome del mezzo assegnato e il range di tempo in cui non sarà disponibile (l'orario di fine range è dato dall'orario di partenza a cui viene sommata la durata della linea in questione, cioè 30 minuti).

  Questo assegnamento non rende vuoto nessun dominio dei mezzi per le corse successive il cui orario di partenza è compreso nel range, quindi tutto è andato a buon fine e si può procedere richiamando la funzione di backtracking sulla lista delle corse aggiornata.

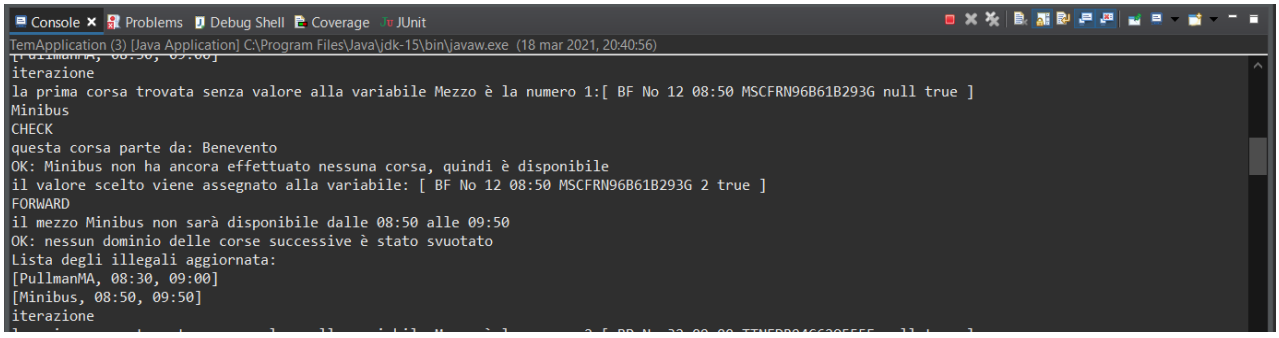



- Alla seconda iterazione, la prima corsa in cui verrà trovata la variabile Mezzo senza valore è la seconda: linea BF (Benevento, Fisciano), viaggio di andata, orario 08:50 e numero di passeggeri attesi 12.

  Si scorre la lista dei mezzi fino a trovare il primo la cui capienza sia maggiore o uguale al numero dei passeggeri attesi: Minibus, con capienza 15. Questo mezzo non ha alcun range in cui risulta illegale e non ha ancora effettuato alcuna corsa, dunque è disponibile per qualunque luogo di partenza; il valore viene assegnato alla variabile.

  Vengono inseriti, nella lista degli illegali, i seguenti dati: Minibus, 08:50, 09:50, ovvero il nome del mezzo assegnato e il range di tempo in cui non sarà disponibile (l'orario di fine range è dato dall'orario di partenza a cui viene sommata la durata della linea in questione, cioè 60 minuti).

  Questo assegnamento non rende vuoto nessun dominio dei mezzi per le corse successive il cui orario di partenza è compreso nel range, quindi tutto è andato a buon fine e si può procedere richiamando la funzione di backtracking sulla lista delle corse aggiornata.



Come si può notare, l'algoritmo procede allo stesso modo della fase di assegnazione dei conducenti. Il team ha voluto, però, porre l'attenzione alle ultime iterazioni, in cui si attua effettivamente il backtracking.

- Alla tredicesima iterazione, la prima corsa in cui verrà trovata la variabile Mezzo senza valore è la tredicesima: linea SB (Salerno, Benevento), viaggio di ritorno, orario 14:10 e numero di passeggeri attesi 54.

  La lista degli illegali, a questo punto, è così composta:

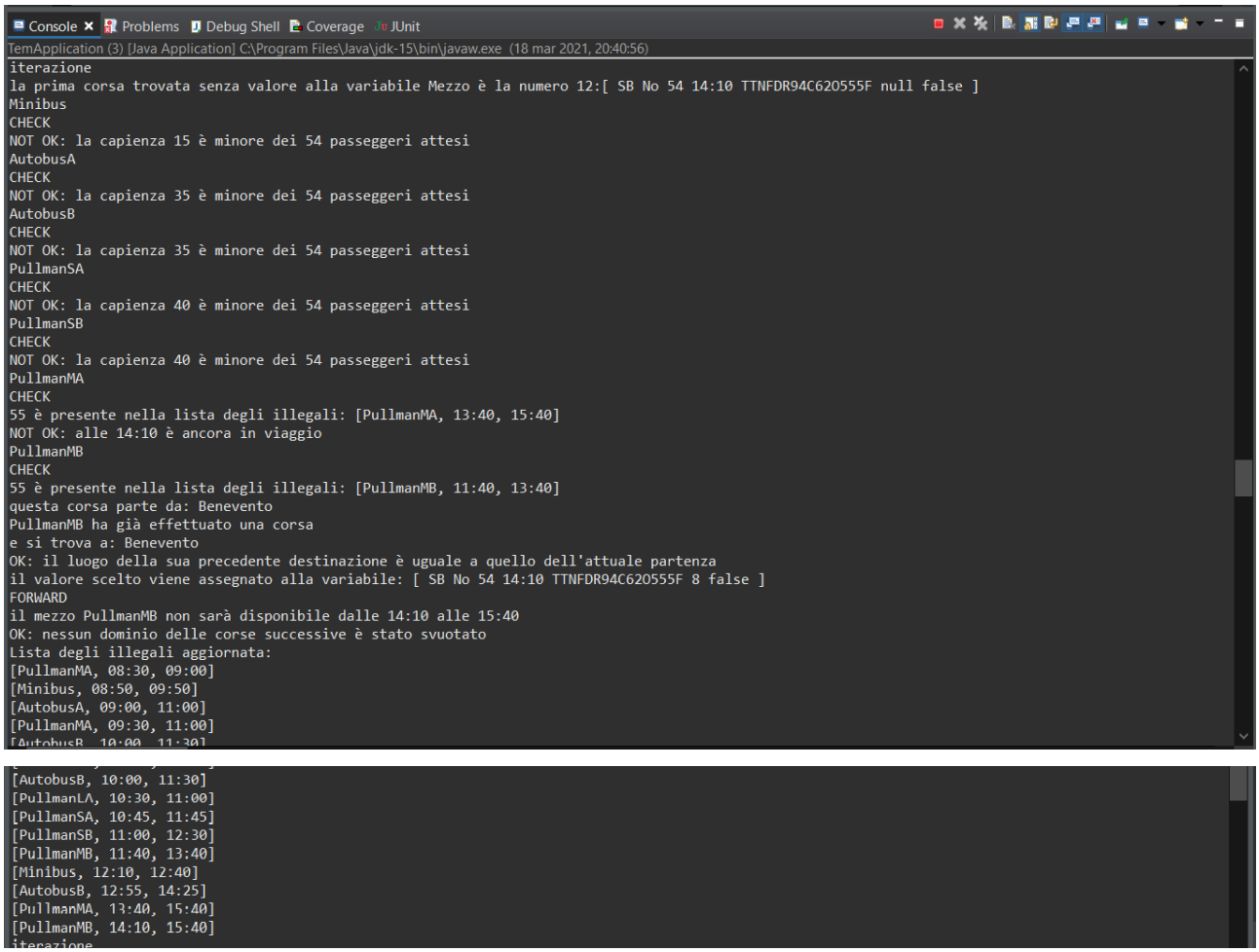
  PullmanMA, 08:30, 09:00; Minibus, 08:50, 09:50; AutobusA, 09:00, 11:00; PullmanMA, 09:30, 11:00; AutobusB, 10:00, 11:30; PullmanLA, 10:30, 11; PullmanSA, 10:45, 11:45; PullmanSB, 11:00, 12:30; PullmanMB, 11:40, 13:40; Minibus, 12:20, 12:40; AutobusB, 12:55, 14:25; PullmanMA, 13:40, 15:40.

  Si scorre la lista dei mezzi fino a trovare il primo la cui capienza sia maggiore o uguale al numero dei passeggeri attesi: PullmanMA, con capienza 55. Questo mezzo è, però, ancora in viaggio.

  Si considera il PullmanMB, anch'esso con capienza 55. Questo mezzo ha terminato il viaggio precedente a Benevento, quindi è disponibile, e il valore viene assegnato alla variabile.

  Vengono inseriti, nella lista degli illegali, i seguenti dati: PullmanMA, 14:10, 15:40, ovvero il nome del mezzo assegnato e il range di tempo in cui non sarà disponibile (l'orario di fine range è dato dall'orario di partenza a cui viene sommata la durata della linea in questione, cioè 90 minuti).

  Questo assegnamento non rende vuoto nessun dominio dei mezzi per le corse successive il cui orario di partenza è compreso nel range, quindi tutto è andato a buon fine e si può procedere richiamando la funzione di backtracking sulla lista delle corse aggiornata.




- Alla quattordicesima iterazione, la prima corsa in cui verrà trovata la variabile Mezzo senza valore è la quattordicesima: linea BF (Benevento, Fisciano), viaggio di andata, orario 15:30 e numero di passeggeri attesi 55.

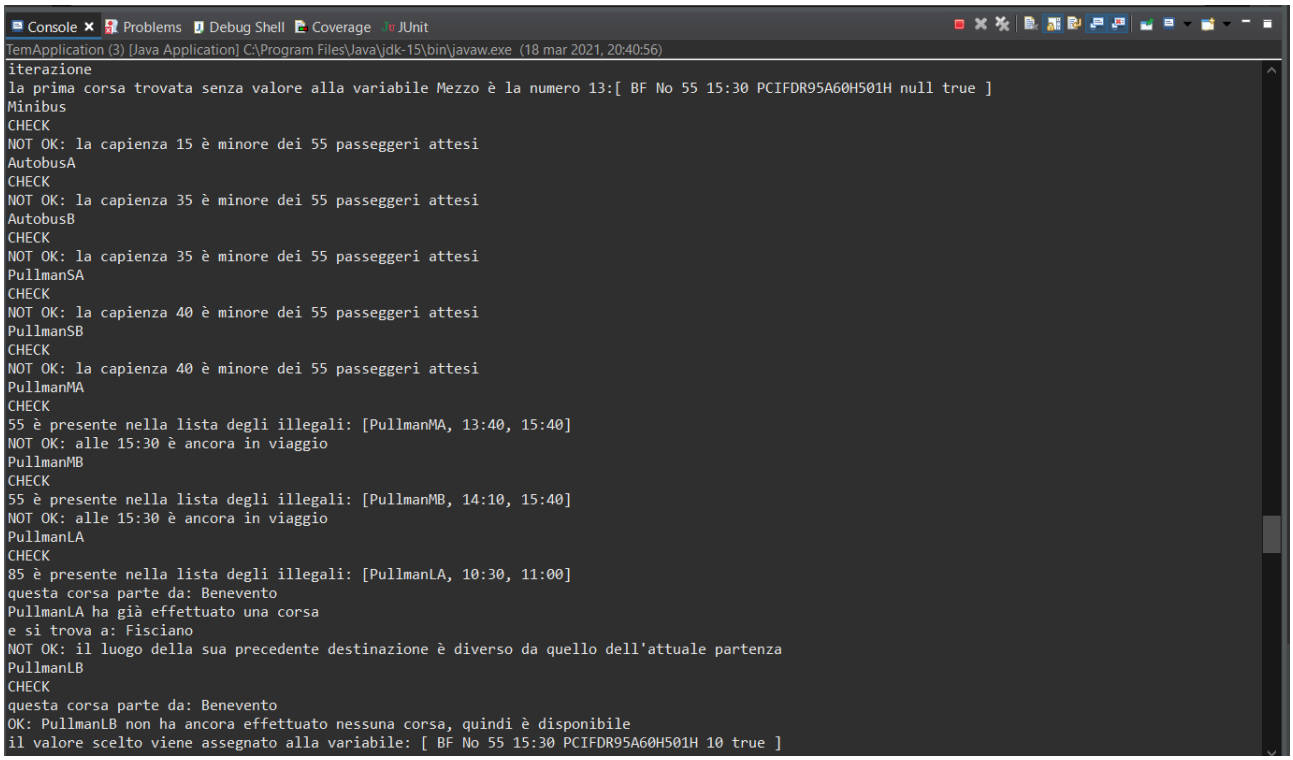
  Si scorre la lista dei mezzi fino a trovare il primo la cui capienza sia maggiore o uguale al numero dei passeggeri attesi: PullmanMA, con capienza 55. Questo mezzo è, però, ancora in viaggio, così come il PullmanMB.

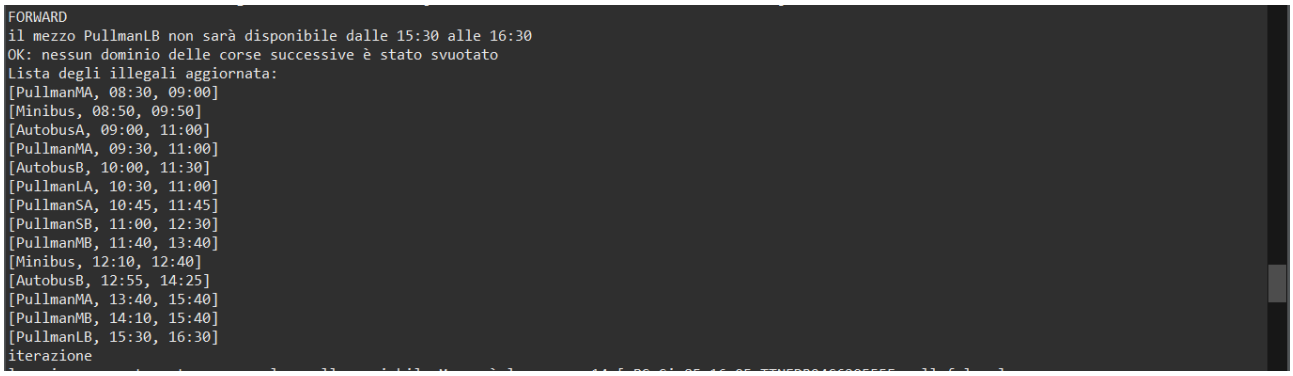
  Si considera il PullmanLA, con capienza 85. Questo mezzo ha terminato il viaggio precedente a Fisciano, quindi non è gli è possibile partire da Benevento.

  Si considera il PullmanLB, anch'esso con capienza 85. Questo mezzo non ha ancora effettuato alcuna corsa, dunque è disponibile, e il valore viene assegnato alla variabile.

  Vengono inseriti, nella lista degli illegali, i seguenti dati: PullmanLB, 15:30, 16:30, ovvero il nome del mezzo assegnato e il range di tempo in cui non sarà disponibile (l'orario di fine range è dato dall'orario di partenza a cui viene sommata la durata della linea in questione, cioè 60 minuti).

  Questo assegnamento non rende vuoto nessun dominio dei mezzi per le corse successive il cui orario di partenza è compreso nel range, quindi tutto è andato a buon fine e si può procedere richiamando la funzione di backtracking sulla lista delle corse aggiornata.



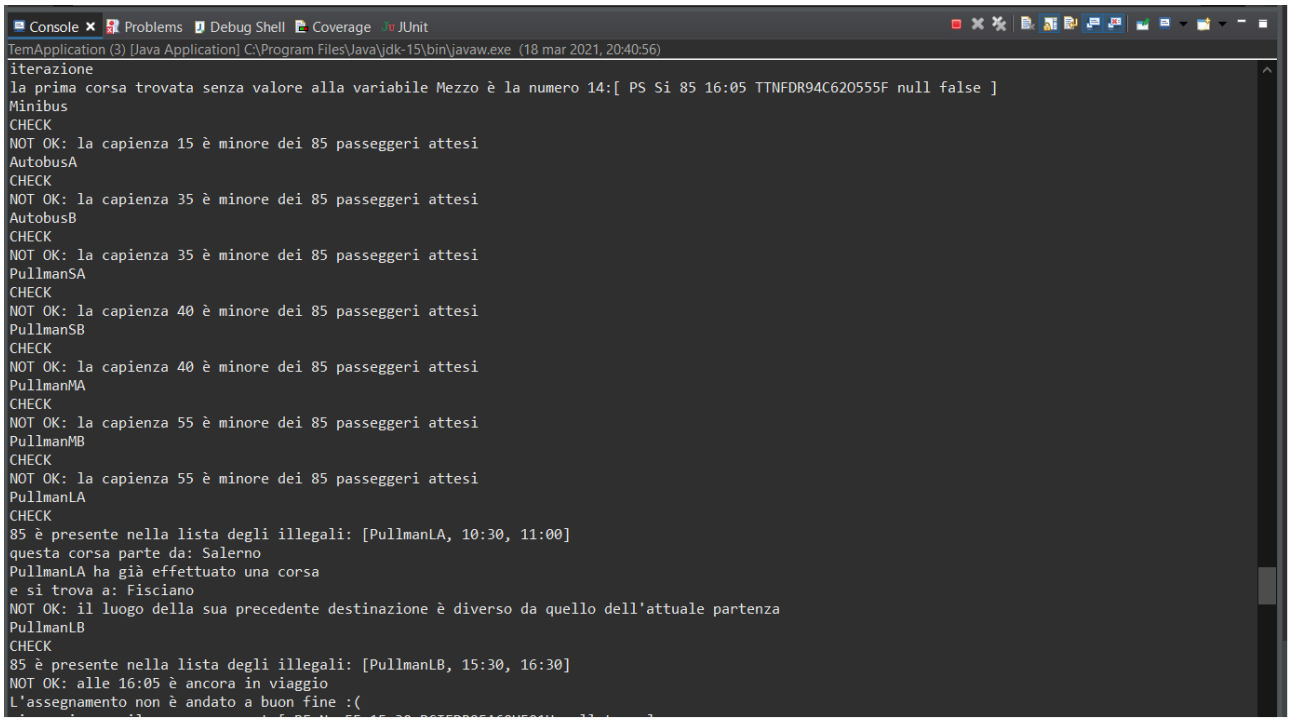




- Alla quindicesima iterazione, la prima corsa in cui verrà trovata la variabile Mezzo senza valore è la quindicesima: linea PS (Potenza, Salerno), viaggio di ritorno, orario 16:05 e numero di passeggeri attesi 85.

  Si scorre la lista dei mezzi fino a trovare il primo la cui capienza sia maggiore o uguale al numero dei passeggeri attesi: PullmanLA, con capienza 85, la cui destinazione del viaggio precedente è diversa dal luogo dell'attuale partenza.

  Si considera il PullmanLB, anch'esso con capienza 85, che però non ha ancora terminato il precedente viaggio.

  Ne consegue che nessun Pullman supera il check, quindi non ci sono valori legali da assegnare alla variabile in questione.

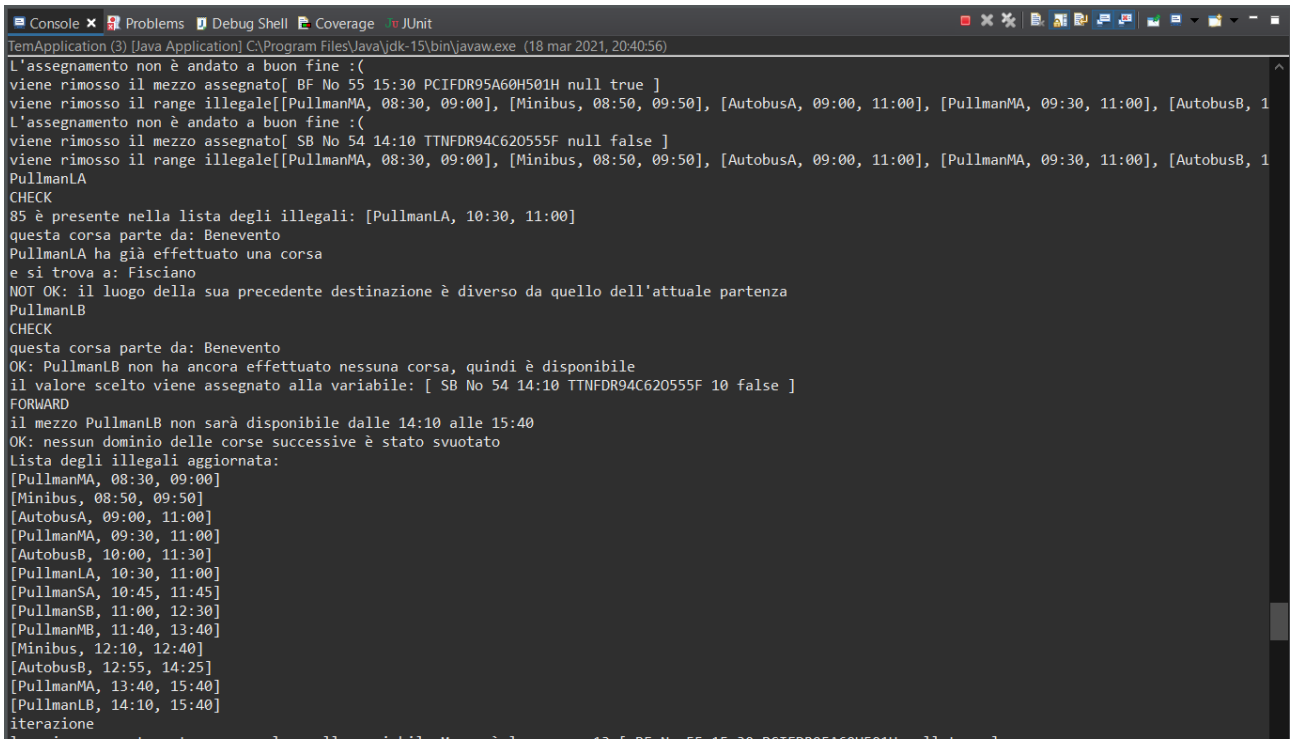



- A questo punto, l'agente torna alla corsa precedente, la numero 14. Il mezzo che era stato assegnato è il PullmanLB: questo assegnamento viene rimosso, così come il suo range di invalidità dalla lista degli illegali. Essendo il PullmanLB l'ultimo da considerare nella lista per quella iterazione, l'agente dovrà tornare ancora una volta indietro, nello specifico alla corsa numero 13. Il mezzo assegnato era il PullmanMB che, ovviamente, non ha portato ad un risultato, quindi si passa a considerare il successivo della lista: il PullmanLA. Questo mezzo ha terminato la precedente corsa a Fisciano, quindi non può partire da Benevento e viene scartato.

  Si considera il PullmanLB. Questo mezzo non ha ancora effettuato alcuna corsa, dunque è disponibile, e il valore viene assegnato alla variabile.

  Vengono inseriti, nella lista degli illegali, i seguenti dati: PullmanLB, 14:10, 15:40, ovvero il nome del mezzo assegnato e il range di tempo in cui non sarà disponibile (l'orario di fine range è dato dall'orario di partenza a cui viene sommata la durata della linea in questione, cioè 90 minuti).

  Questo assegnamento non rende vuoto nessun dominio dei mezzi per le corse successive il cui orario di partenza è compreso nel range, quindi tutto è andato a buon fine e si può procedere richiamando la funzione di backtracking sulla lista delle corse aggiornata.




- Ora l'agente ha la possibilità di procedere alle corse successive, come fatto fino ad ora, riuscendo, effettivamente a trovare una soluzione.

Poiché tutti gli assegnamenti di tutte le variabili sono andati a buon fine, i dati grezzi vengono tramutati in corse e il programma delle corse viene definitivamente aggiunto tra quelli dell'azienda IntArtificiale s.r.l..

# 8. Conclusioni

Al termine di questa analisi e di un gran numero di prove effettuate su diversi dati inseriti, si è giunti alla conclusione che la soluzione trovata sia tra le migliori in quanto il numero di passeggeri rimasti a piedi si azzera (a meno di consistenti cambiamenti rispetto ai dati inseriti), e il numero dei posti rimasti vuoti diminuisce di molto. Inoltre, spesso, e come si può notare anche nei dati utilizzati come esempio, l'azienda riesce a coprire tutte le corse della giornata utilizzando un numero minore di conducenti; questo potrebbe essere un vantaggio per i conducenti, potendo godere di meno ore di lavoro da dividersi tra di loro, o per l'azienda stessa, potendo licenziare i conducenti inutilizzati e diminuendo le proprie spese o adibendoli ad altre mansioni.

# 9. Fonti e riferimenti

Link al progetto generale: https://github.com/FedericaPica/TransportEfficiencyManager

Link al package contenente il modulo di intelligenza artificiale: https://github.com/FedericaPica/TransportEfficiencyManager/tree/main/tem/src/main/java/com/java/tem/aimodule

Link alla documentazione: https://github.com/FedericaPica/TransportEfficiencyManager/tree/main/Documentazione%20FIA