**Corso di Data Mining**

**Tesina di fine corso**

**Applicazione di metodologie di data mining per la soluzione**

**di problemi WTRSP**

Autore: G. Perrone

Revisione: 3 – 03/07/17

Sommario

[1 Introduzione e scopo del progetto 2](#_Toc487192367)

[2 Descrizione del problema 3](#_Toc487192368)

[2.1 Requisiti 4](#_Toc487192369)

[2.1.1 Requisiti sulle risorse operative 4](#_Toc487192370)

[2.1.2 Requisiti sui tasks 4](#_Toc487192371)

[2.1.3 Requisiti sulle metriche (funzione obiettivo) 4](#_Toc487192372)

[2.1.4 Requisiti non funzionali (performance) 5](#_Toc487192373)

[3 Caratterizzazione dei dati 6](#_Toc487192374)

[3.1 Caratteristiche dei task per la classificazione 7](#_Toc487192375)

[3.1.1 Caratteristiche spaziali 8](#_Toc487192376)

[3.1.2 Caratteristiche temporali 9](#_Toc487192377)

[3.1.3 Caratteristiche legate alle risorse 9](#_Toc487192378)

[4 Generazione del dataset 10](#_Toc487192379)

[4.1 Parametri di configurazione 10](#_Toc487192380)

[4.1.1 Parametri generici 10](#_Toc487192381)

[4.1.2 Parametri per le risorse 10](#_Toc487192382)

[5 Risoluzione del problema 13](#_Toc487192383)

[5.1 Performance del risolutore 13](#_Toc487192384)

[6 Indice delle figure 14](#_Toc487192385)

[7 Riferimenti 14](#_Toc487192386)

# Introduzione e scopo del progetto

Scopo del progetto è l'applicazione di metodologie di data mining per il miglioramento delle performances di risoluzione di problemi WTRSP e VRPTW (Vehicle Routing Problem Time Window). La tipologia del problema è meglio descritta nella sezione 2 - Descrizione del problema; si tratta comunque di una variante del problema di *vehicle routing* generico, introdotto da G. Dantzig nel 1959 e che rappresenta uno dei problemi di ricerca operativa maggiormente soggetto a studi e ricerche.

Il motivo dell'importanza del VRP va ricercato nel fatto che esso ha numerose applicazioni in ambiti industriali ed in particolare per l'ottimizzazione della logistica e delle squadre di intervento. Si tratta inoltre in un problema NP-Hard, la cui risoluzione mediamente metodi tradizionali di ricerca di ottimo assoluta può essere di fatto infattibile a causa del tempo richiesto per la risoluzione del problema stesso all'aumentare del numero di variabili e vincoli da gestire.

Si ricorre quindi tipicamente a risolutori di tipo euristico, che permettono di trovare una soluzione buona "abbastanza" (ma non ottima) in tempi che però sono compatibili con i normali scenari operativi.

A questo proposito, facendo seguito al lavoro di analisi già svolto per il corso di Metodi e Tecnologie di Simulazione, si è interessati ad analizzare la possibilità di utilizzare metodologie di data mining (classificazione e clustering) per poter effettuare un pre-processing dei dati in ingresso (definiti come *istanze* del problema) che permetta di:

* predire in anticipo se un task ha buone probabilità di essere schedulato, ossia di essere incluso nella soluzione generata dall'algoritmo risolutore;
* effettuare in anticipo un raggruppamento dei task per migliorare l'efficienza dell'algoritmo risolutore (i.e. creare dei gruppi di task e "forzare" il risolutore a trattarli insieme, utilizzando magari un indice di priorità).

Si dovranno quindi eseguire i seguenti passi:

1. per l'oggetto task, identificare delle caratteristiche che possano poi essere utilizzate per la classificazione (ad es. vicinanza geografica, vicinanza temporale, ecc.);
2. applicare un risolutore (probabilmente euristico) per la soluzione di istanze note del problema;
3. utilizzare le soluzioni così ottenute per l'addestramento dell'algoritmo di classificazione;
4. verificare se ed in che modo le informazioni ottenute come output dell'algoritmo di classificazione così addestrato siano in grado di migliorare l'efficienza di risoluzione del risolutore stesso.

Il progetto potrà poi essere esteso ad altri tipi di istanze, note in letteratura, per verificare se l'utilizzo del classificatore possa migliorare le performances risolutive note.

# Descrizione del problema

Il problema da risolvere rientra nella categoria più ampia dei *Task Scheduling Problems* che a loro volta derivano dal **Traveling Salesman Problem (TSP)** ed, in particolare, è riassumibile come segue:

Dati:

* un insieme di risorse, ciascuna caratterizzata da un orario di disponibilità, un luogo di partenza e di origine e di capacità (skills) per svolgere un compito
* un insieme di compiti (tasks), ciascuno caratterizzato da una finestra oraria di eseguibilità, una posizione fisica di esecuzione e di una serie di skills richiesti per eseguire il compito

determinare, per ciascuna risorsa, le assegnazioni task - risorsa che garantiscano la copertura del maggior numero di task (tutti nel caso del modello con vincoli hard) minimizzando al tempo stesso il tempo di viaggio delle risorse.

Si tratta quindi più propriamente di un problema di *Workforce and Technician Routing and Scheduling Problem (WTRSP)*  che rappresenta a sua volta una variante del *Vehicle Routing Problem with Time Windows* (*VRPTW)* con l'aggiunta di limitazione sugli skill delle risorse. In termini di programmazione lineare è possibile dire che dovremo definire ed implementare:

* una funzione di costo rappresentata da un insieme di termini che penalizzeranno l'aumento dei tempi di percorrenza, la non copertura di uno o più task
* una serie di espressioni che imporranno i vincoli descritti sopra, come ad esempio:
  + un task non può essere eseguito da una stessa risorsa prima che questa abbia terminato il task precedentemente assegnato e che abbia avuto la possibilità di recarsi dal luogo di esecuzione del task precedente al luogo di esecuzione del task
  + tutte le risorse devono partire dal proprio luogo di origine e tornare al proprio luogo di destinazione
  + le risorse non possono lavorare al di fuori dell'orario di disponibilità
  + i task non possono essere eseguiti al di fuori delle finestre di disponibilità
  + un task non può essere eseguito da più di una risorsa

## Requisiti

La tabella seguente (in Inglese), riporta l'insieme dei requisiti (e quindi dei vincoli) previsti nel problema ed il relativo stato di implementazione nell’applicazione.

### Requisiti sulle risorse operative

|  |  |  |
| --- | --- | --- |
| ID req. | Impl. ? | Description |
| RO1 | S | An operative resource can execute one task at a time. |
| RO2 | S | A task can be executed only by one operative resource. |
| RO3 | S | When an operative resource assigned to a task starts executing it, the travel time needed to reach that task must be taken into account. |
| RO4 | N | An operative resource ρ can decide to take a rest (a break) just once, within the declared break interval. It means that the break must be taken within the time interval duration. During the break, the interrupted task cannot be reassigned to another operative resource. The break is continuous. |
| RO5 | S | An operative resource should start and end its working session (delimited by the availability time interval) in the declared Origin and Destination locations, so also the starting and ending travelling time should be taken into account. |
| RO6 | S | The scheduler is stateless, in the sense that once an operative resource starts to execute an assigned task, the task will not be scheduled again and the operative resource will be scheduled when it terminates the execution of the assigned task. |
| RO7 | S | The current position of the operative resource is discrete and not continuous. |
| RO8  [NOT MANDATORY] | N | In order to avoid solutions highly unbalanced, in which an operative resource works more than others, specific constraints are imposed with the aim of balancing the workload among the operative resources. For example, this could be formalized imposing that each operative resource cannot work more than a specific upper bound on his/her working time. |

### Requisiti sui tasks

|  |  |  |
| --- | --- | --- |
| ID | Impl ? | Description |
| RT1 | S | A task can be only executed within its Time Interval (or Time Window). |
| RT2 | N | A task can be assigned to any available operative resource that has the required skills. |

### Requisiti sulle metriche (funzione obiettivo)

|  |  |  |
| --- | --- | --- |
| ID | Impl ? | Description |
| RM1 | S | **Coverage maximization** : the scheduler should maximize the number of tasks assigned (with some priority). Not necessarily all tasks should be assigned.[[1]](#footnote-1) |
| RM2 | S | **Total travel times minimization** : the scheduler should minimize the sum of all travel times. |
| RM3 | S | Coverage maximization has priority higher than total travel time minimization. |

### Requisiti non funzionali (performance)

|  |  |  |
| --- | --- | --- |
| ID | Impl ? | Description |
| RP1 | S[[2]](#footnote-2) | Given the task scheduling problem (TSP) defined in the previous section, the scheduler should be able to schedule up to 1.000 tasks to a maximum of 100 operative resources. |
| RP2 | S | The scheduling horizon should cover at least 24h. |

# Caratterizzazione dei dati

Il problema in esame consiste, come già detto, nel determinare, dato un insieme di partenza di task da schedulare, la successione migliore (se esiste) che permetta di eseguire tutti i task, rispettando i vincoli sulle finestre temporali, minimizzando al tempo stesso la distanza percorsa.

L'insieme degli elementi che costituiscono il problema è quindi composto da:

* i task da eseguire
* le risorse a disposizione per eseguire i task

Dato quanto riportato nell'introduzione e nella descrizione del problema, un task potrà essere rappresentato (in modo univoco) tramite le seguenti proprietà:

* ID del task
* Posizione del sito dove il task deve essere eseguito (può essere espressa con un set di coordinate (x,y))
* Finestra temporale di disponibilità per l'esecuzione del task (assumendo di lavorare nell'arco delle 24 ore, può essere espressa tramite una coppia (Ts, Te))
* Un livello di skill richiesto per l’esecuzione del task

Conseguentemente, ciascun task può essere rappresentato da una tupla nel modo seguente:

|  |  |  |  |
| --- | --- | --- | --- |
| *ID\_t* | *(Px, Py)* | *(Ts, Te)* | *Sk* |

Tabella :rappresentazione dei task

Per quanto riguarda invece le risorse operative, esse potranno essere descritte utilizzando le seguenti proprietà:

* ID della risorsa
* Posizione di partenza della risorsa (ossia la posizione da dove la risorsa inizierà il proprio percorso)
* Posizione finale della risorsa (ossia la posizione dove la risorsa dovrà terminare il proprio percorso)
* Finestra temporale di disponibilità della risorsa
* Finestra temporale per il break (i.e. periodo di pausa e quindi di indisponibilità) della risorsa
* Skill posseduti dalla risorsa (devono essere ≥ lo skill di un task perché una risorsa possa eseguire un task)

Utilizzando la notazione già vista per i task, possiamo quindi rappresentare ciascuna risorsa con una tupla costituita nel modo seguente:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *ID\_r* | *(Sx, Sy)* | *(Ex, Ey)* | *(Rs,Re)* | *(Bs, Be)* | *Sk* |

Tabella : rappresentazione delle risorse operative

Nel proseguire di questo capitolo, analizzeremo il problema nella sua formulazione più generica, senza quindi introdurre assunzioni o limitazioni relativamente (ad esempio) al numero delle risorse operative, alla eventuale coincidenza tra luogo di partenza e di destinazione o alla presenza o meno del periodo di pausa.

## Caratteristiche dei task per la classificazione

Come si è già detto, scopo del progetto è l'applicazione di metodologie di data mining (di classificazione o predittive) al problema in oggetto per poter:

1. predire la schedulabilità o meno di un dato task;
2. verificare la possibilità di introdurre un raggruppamento forzato dei task da schedulare che possa portare ad un miglioramento delle caratteristiche della soluzione trovata.

In entrambi i casi è necessario definire le caratteristiche per questi su cui poi applicheremo gli algoritmi di data mining.

Come si è già visto, un oggetto task è essenzialmente caratterizzato in modo spaziale (dalla sua posizione) e temporale (dalla sua finestra di disponibilità). Vogliamo quindi definire delle caratteristiche per entrambe le dimensioni che possano essere significative ai fini del nostro problema.

### Caratteristiche spaziali

Per identificare le caratteristiche spaziali che potrebbero essere di interesse per l'analisi del problema, iniziamo analizzando una rappresentazione grafica di un possibile dataset, costituito da 50 task e 10 risorse operative. Nella figura, i punti indicati con l'etichetta "Tx" rappresentano i task da eseguire, mentre i punti rappresentati con "Ox" e "Dx" rappresentano rispettivamente i punti di partenza e destinazione delle risorse operative.

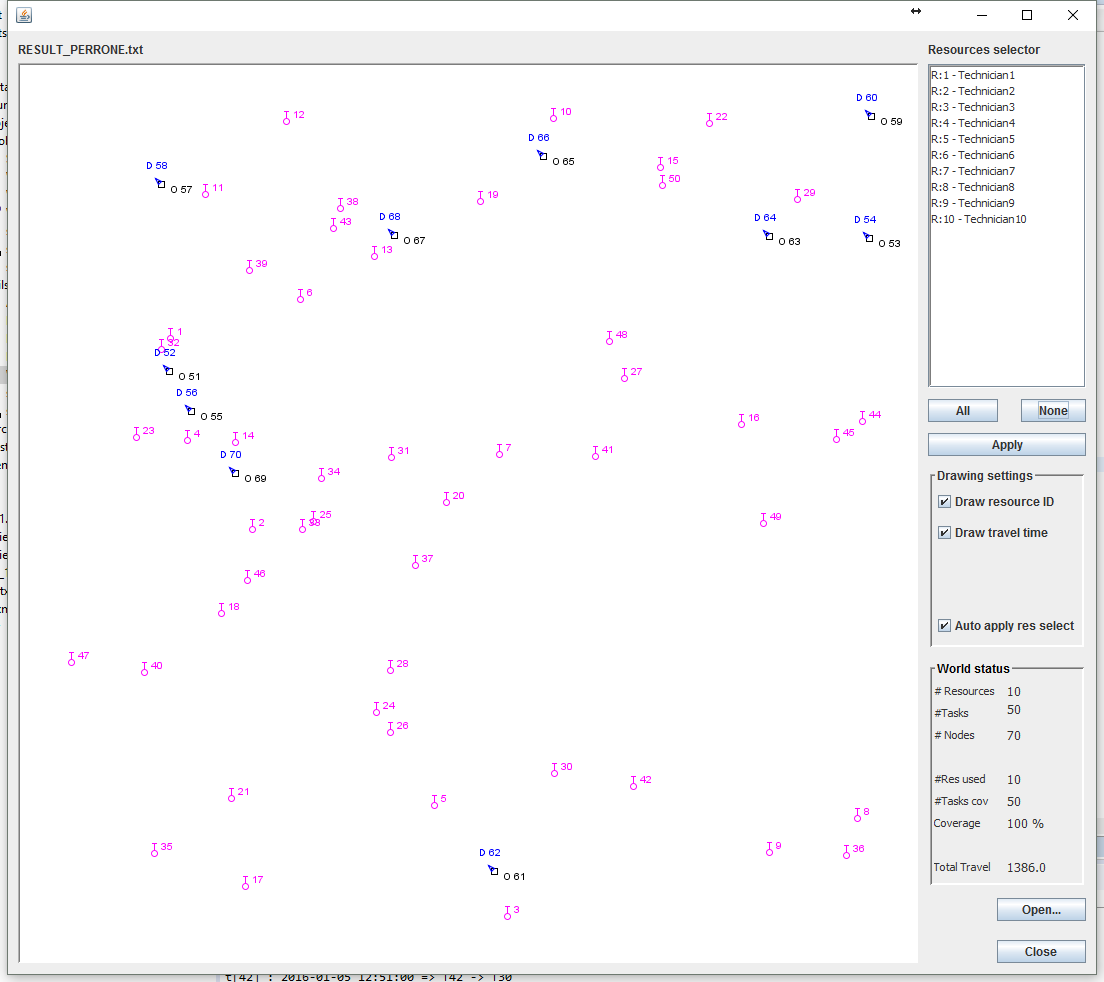


Figura : rappresentazione grafica di un dataset

Si supponga di utilizzare un operatore che, date le coordinate di due punti, ritorni come risultato una valutazione della distanza tra questi due punti (ad esempio utilizzando l'espressione della distanza euclidea).

Si supponga inoltre di indicare con *n* il numero dei task e con *m* il numero delle risorse operative, possiamo identificare per ciascun task i-esimo le seguenti caratteristiche spaziali:

1. *ClosestDistance*: distanza del task i dal task più vicino
2. *FarestDistance*: distanza del task i dal task più lontano
3. *avgDist*: media aritmetica delle distanze del task i a tutti gli altri tasks
4. *stvDist*: deviazione standard delle distanze del task i a tutti gli altri tasks
5. *mdnDist*: moda della distanza

Le caratteristiche in esame tendono ad analizzare la posizione del task rispetto agli altri task che compongono il dataset, ed in particolare tendono ad analizzare la densità, ossia indicano quanto è probabile che un altro task sia nelle vicinanze del task in esame.

### Caratteristiche temporali

Analogamente a quanto visto per le caratteristiche spaziali, è opportuno introdurre delle caratteristiche che permettano di tenere conto dei vincoli sui tempi di esecuzione del task stesso.

Si supporranno vere le seguenti affermazioni per le variabili temporali:

* tutte le finestre sono riferite ad un singolo giorno;
* sono espresse come un interno in formato mmm dove mmm rappresenta i minuti trascorsi dall’ora 00. I secondi non sono tenuti in considerazione
* la durata massima per la pausa per ciascuna risorsa operativa può essere al massimo il 11% della sua finestra di disponibilità (e.g. su una finestra di 9 ore l’11% rappresenta circa 60 minuti di pausa);
* non ci sono limiti all'ampiezza della finestra di disponibilità per i task (i.e. un task potrebbe essere disponibile anche per 24 ore).

Si supponga, come già fatto, di indicare con *n* il numero dei task e con *m* il numero delle risorse operative, possiamo identificare per ciascun task le seguenti caratteristiche temporali:

1. *timWind*: ampiezza della finestra temporale di disponibilità del task (in minuti);
2. *svrTime*: durata del task, i.e. tempo di esecuzione richiesto (in minuti);
3. *blTWB[k]:* (con k ∈ [0;23]) bin di tipo boolean che indicano se l’orario di partenza della finestra temporale cade nell’intervallo orario hh -> hh+1 con hh = k (e.g. il bin blTWB[0] verrà settato a true per tutti i task il cui inizio della finestra temporale è compreso tra le ore 00:00 e 00:59);
4. *perResUnav*: percentuale delle risorse che hanno la finestra di indisponibilità sovrapposta (anche parzialmente) con l’orario di inizio disponibilità del task;

Le prime tre caratteristiche forniscono un'indicazione sulla durata della finestra di eseguibilità, sul suo posizionamento (nell’arco dell’orizzonte temporale di 24 ore) e dell’estensione della durata di esecuzione del task.

La quarta caratteristica infine permette di identificare se il momento di inizio di un task cade all'interno della pausa di indisponibilità per una data risorsa (tenendo in considerazione anche l'ampiezza della finestra di eseguibilità del task potrà essere eventualmente possibile scartare a priori l'utilizzo di una data risorsa operativa).

### Caratteristiche legate alle risorse

Per ultimo, si tengono in considerazione alcune caratteristiche legate alle risorse. In particolare, considerato che ciascuna risorsa è caratterizzata da uno skill (i.e. un insieme di competenze) e ciascun task da uno skill richiesto, vengono identificate le seguenti caratteristiche per ciascun task:

1. *perResWSkills*: percentuale (sul totale delle risorse) delle risorse che possiedono skills sufficienti per eseguire il task in questione[[3]](#footnote-3)

# Generazione del dataset

Il dataset viene generato dalla classe GenerateDataSet utilizzando dei parametri di configurazione che permettono di variare il comportamento del generatore di dataset. In particolare, è possibile introdurre delle polarizzazioni in alcuni parametri (come ad es. la posizione di partenza delle risorse) allo scopo di verificare l’efficacia degli algoritmi di predizione al variare delle condizioni di partenza.

I parametri di configurazione sono gestiti tramite file xml in modo tale da assicurare leggibilità e facilità di modifica. In caso di esecuzione di un batch di esecuzioni, il file di configurazione sarà rappresentato da una successione di entries xml ciascuna corrispondente ai parametri di configurazione da assegnare al job in questione.

## Parametri di configurazione

### Parametri generici

Questo insieme di parametri sono generici e si riferiscono al “mondo” generato (ad es., la dimensione massima delle coordinate)

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
| Int | nResources; | number of resources to be used for dataset generation |
| Int | nTasks; | number of tasks to be used for dataset generation |
| Int | maxX; | maximum value for the X coordinates |
| Int | maxY; | maximum value for the Y coordinates |

In particolare, si noti come la combinazione del numero di task da generare ed i valori massimi di X e Y (che rappresentano quindi l’estensione del piano in cui verranno posizionati task e risorse) determini la densità dei task che, come si vedrà, risulta essere un parametro importante per il tempo di risoluzione degli scenari.

Nel prosieguo di questo documento, una specifica istanza sarà identificata dal numero dei task, dal numero delle risorse e dalla densità delle risorse stesse, come nel seguente esempio:

10-100-d0,08

Numero di risorse

Densità dei task:

(#task/(maxX \* maxY))

Numero di task

### Parametri di distribuzione spaziale e temporale per le risorse

Questo insieme di parametri permette di modificare il comportamento del generatore relativamente alla distribuzione spaziale delle risorse.

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
| Boolean | bFullResourcesAvailability; | forces all resources to have maximum availability window |
| Boolean | bResReturnToStart; | forces origins and destinations for resources to be the same |
| Boolean | bAllResourcesHaveSamePos; | forces all resources to have the same starting position |
| Char | cResourcesStartingPosition; | specifies how the starting positions will be distributed (R = random, C = Center, N = North, S = South, E = East, W = West, G = tasks center of gravity) |
| Double | dStartingPositionsSpread; | specifies the spreading factor for the starting positions (must be < maxX and maxY) |
| Boolean | bExtendedTimeWin; | forces extension of the time window duration for tasks |
| Int | iResMinStartingTime | minimum starting time for the resources time window |
| Int | iResMinAvailabilityTime | min duration of availability |

#### Distribuzione geografica delle risorse

Come si è già visto nel paragrafo precedente, è possibile specificare al generatore di dataset come distribuire le posizioni di partenza e di arrivo delle risorse.

* In particolare, è possibile specificare (in modo indipendente):
* se posizione di partenza e arrivo devono coincidere;
* se tutte le posizioni di partenza devono coincidere;
* se le posizioni di partenza devono essere generate in modo randomico indipendentemente l’una dalle altre;
* se le posizioni di partenza devono essere generate a partire da un centro di riferimento e disposte entro un raggio definito da un parametro di spread.

A sua volta, è possibile specificare se la posizione del centro (se viene utilizzato) deve:

* coincidere con il centro del piano;
* coincidere con il baricentro dei tasks;
* essere posizionato nella parte “Nord”, “Sud”, “Est”, “Ovest” del piano (v. figura)

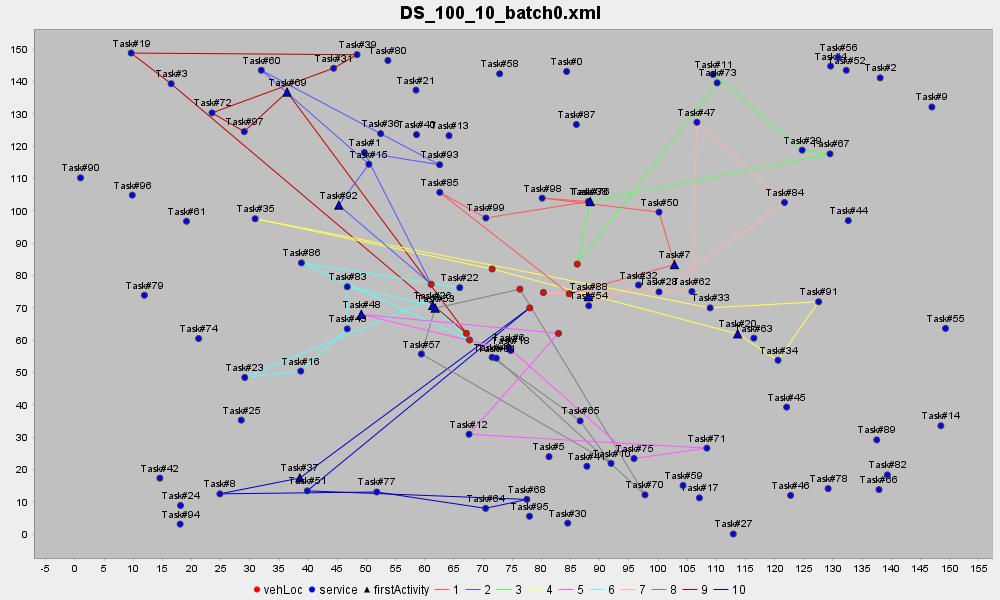


Figura : esempio di dataset (risolto) con posizioni di partenza dei task generate intorno al centro del piano

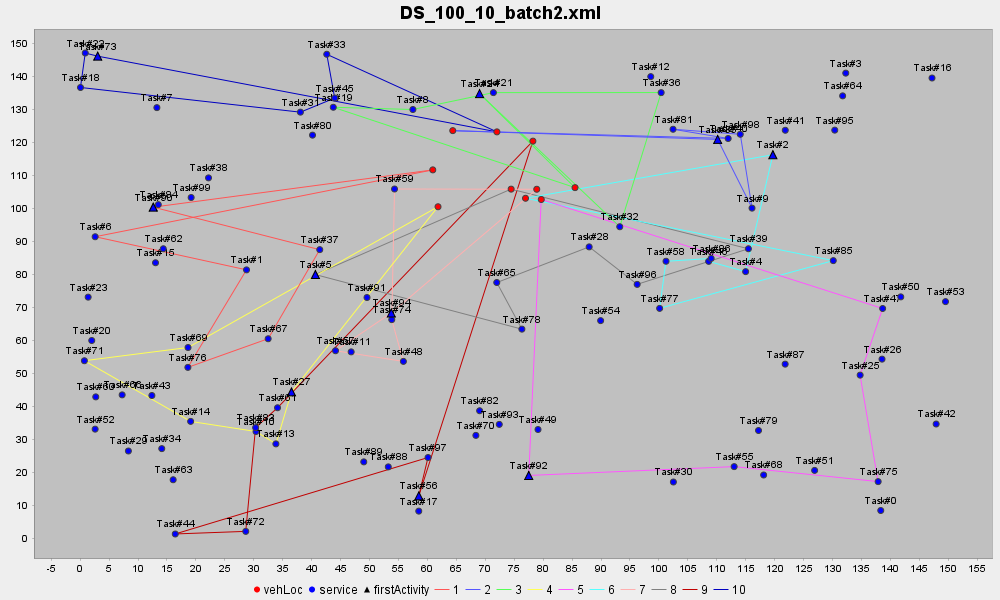


Figura : esempio di dataset (risolto) con posizioni di partenza dei task generate intorno alla parte "nord" del piano

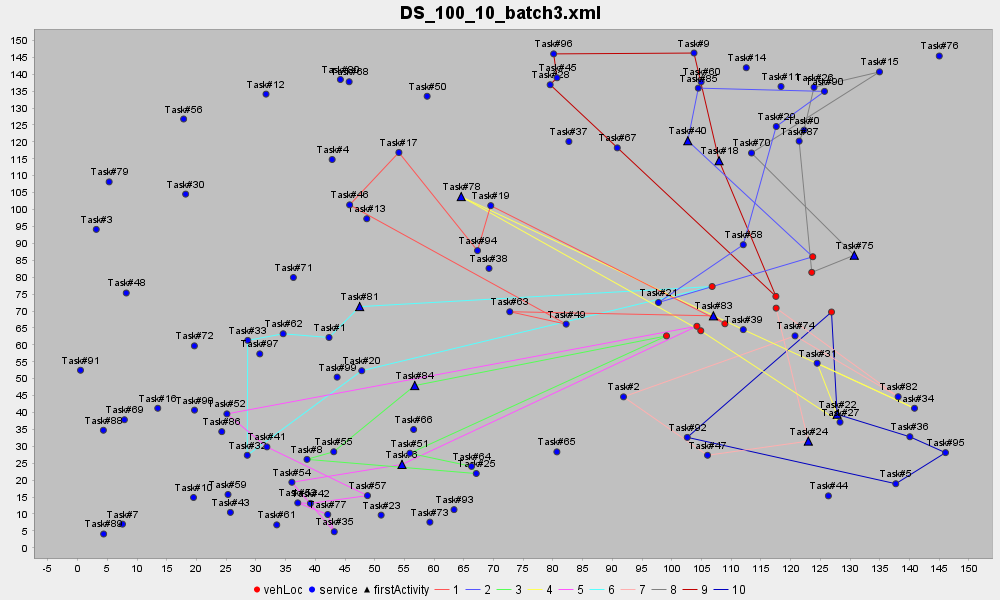


Figura : esempio di dataset (risolto) con posizioni di partenza dei task generate intorno alla parte "est" del piano

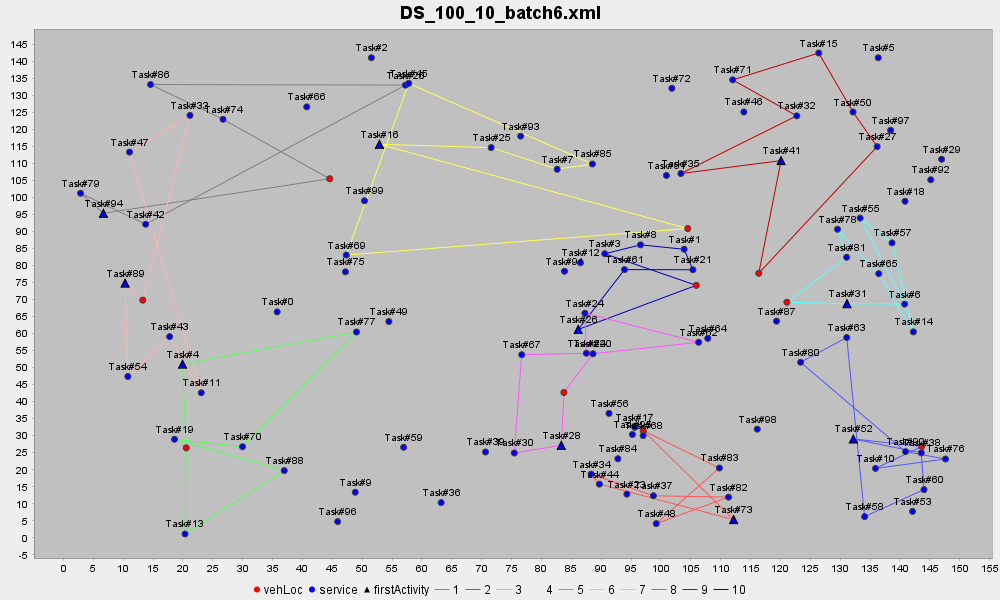


Figura : esempio di dataset (risolto) con posizioni di partenza dei task generate intorno alla parte "est" del piano

### Parametri di distribuzione temporale per i task

Questo insieme di parametri permette di modificare il comportamento del generatore relativamente alla distribuzione temporale dei task.

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
| Int | iTskMinStartingTime; | minimum starting time for the tasks time window |
| Int | iTskMinTimeWindowWidth; | minimum width of the tasks time window |
| Int | iTskMinServiceTimeDuration; | minimum service time duration |
| Int | iTskMaxServiceTimeDuration; | maximum service time duration |
| Int | iTskServiceTimeSpread; | spread for the task starting time (i.e. task start = min start + (0 -> spread) ) |

Ciascun valore (e.g. momento di inizio della finestra temporale, durata del tempo di esecuzione, ecc.) viene generato in modo randomico utilizzando come estremi inferiori e superiori i relativi valori specificati dai parametri di configurazione.

La seguente figura rappresenta la relazione trai vari parametri relativamente alla finestra di disponibilità e alla durata di esecuzione di un task.

0

1440

Finestra di disponibilità

iTskMinStartingTime + RND

(iTskMinStartingTime + RND) + iTskMinTimeWindowWidth + RND

Durata del task

iTskMinServiceTimeDuration < RND < iTskMaxServiceTimeDuration

RND < iTskServiceTimeSpread

Nella figura, RND rappresenta un valore generato con un generatore PRNG.

# Risoluzione del problema

Per la risoluzione del problema è stato utilizzato il *framework* Jsprit, che offre allo sviluppatore una serie di classi ed interfacce per la gestione e la risoluzione dei problemi derivanti dal TSP.

Il *framework* Jsprit usa di default un algoritmo di ottimizzazione euristico basato su un meccanismo di “ruin and recreate” descritto da Schrimpf ed altri[[4]](#endnote-1) che permette di ottenere risultati particolarmente interessanti in scenari complessi caratterizzati dalla difficoltà nell’identificare soluzioni ammissibili al problema. L’algoritmo, quando confrontato con altri algoritmi più “classici” di ricerca delle soluzioni, ha dimostrato performances nettamente migliori in particolare quando applicato a problemi TSP, VRP con time windows ed a problemi di ottimizzazione di rete.

## Performance del risolutore

A titolo esemplificativo, il seguente grafico mostra l’andamento delle performance del risolutore al variare della complessità dello scenario.

Le esecuzioni sono state effettuate su un PC con SO Windows 10, processo Intel i7 (4-core, 8 core logici) @ 2,8GHz e 16Gb di Ram. Dal momento che è possibile impostare, tramite un’opzione di configurazione, il numero di thread che il solver deve utilizzare, per tutte le esecuzioni del progetto sono stati utilizzati 7 thread.

Figura : tempi di risoluzione per tipi diversi di dataset

Come si nota, i tempi di risoluzione variano ovviamente al variare della complessità dello scenario da risolvere, ma anche al variare della densità dei task che caratterizza lo scenario stesso. Questo fenomeno è senz’altro dovuto al fatto che in scenari con densità maggiore il risolutore ha la possibilità di scegliere tra più soluzioni possibili e di conseguenza l’efficacia del “pruner” nell’algoritmo di risoluzione risulta nel complesso diminuita.

# Applicazione sviluppata

Per svolgere tutte le attività legate al progetto è stata sviluppata una applicazione Java che implementa tre funzioni principali:

1. generazione e risoluzione in batch di dataset, con produzione dei relativi files (dataset, input per il classificatore e statistiche di risoluzione)
2. generazione e validazione in batch del classificatore, con produzione dei relativi files (dataset e statistiche sui modelli generati)
3. generazione di un modello tramite training su istanze di complessità “x” (e.g. 1000 task e 100 risorse) e sua validazione su istanze di complessità “>x” (e.g. 2000 task e 200 risorse), con produzione dei relativi files di analisi e statistica

## Generazione dei dataset e relativa risoluzione

### Requisiti

Il primo passo per svolgere le attività previste dal progetto ha riguardato la generazione dei dataset, secondo i parametri specificati nel file di configurazione, e la relativa risoluzione utilizzando il solver Jsprit. Allo scopo di poter analizzare in seguito i risultati e di poter poi proseguire con le attività di data mining vero e proprio (training, cross-validation e validazione del modello), tutte le informazioni relative ai dataset, agli attributi generati e agli esiti della risoluzione del suddetto devono essere salvate su disco in modo da poter essere poi recuperate dagli altri moduli dell’applicazione.

I requisiti di prodotto per questo modulo possono quindi essere riassunti come segue:

1. generazione di dataset (memorizzati su disco in formato xml) utilizzando i parametri di configurazione e conseguentemente i gradi di libertà specificati nel capitolo 4
2. esecuzione del solver sul dataset generato e memorizzazione su disco delle informazioni statistiche sui risultati ottenuti dalla soluzione ottima trovata dal risolutore (come minimo: tempo di esecuzione, numero di thread usati, %di task schedulati, costo totale della soluzione, tempo di viaggio e di attesa per la soluzione)
3. calcolo degli attributi per ciascun dataset come da specifiche definite nel capitolo 3 e scrittura su file (in formato .arff per la successiva compatibilità con Weka)
4. possibilità di eseguire i task da 1 a 3 in modalità batch utilizzando parametri di configurazione per ciascun job memorizzati in un file xml (per assicurare una gestione del file utilizzando anche strumenti esterni)

### Flow chart di esecuzione

La figura seguente riassume il flow chart di esecuzione dell’applicazione di generazione e risoluzione dei dataset seguendo le specifiche descritte nel paragrafo 6.1.1.



Figura : flow chart di esecuzione del modulo di generazione e risoluzione dataset

In pratica, il modulo utilizza un file xml in cui sono memorizzate le varie istanze dei job – è quindi possibile variare in modo indipendente le caratteristiche di ciascuna istanza in modo da generare dataset tra di loro indipendenti. Una volta caricato in memoria l’insieme dei job da processare, per ciascun job viene eseguito un ciclo del loop principale che prevede:

1. generazione del dataset
2. risoluzione del dataset
3. generazione degli attributi
4. memorizzazione su file dei risultati

Per assicurare la modularità e la riusabilità del codice, gli step 1 e 2 sono eseguiti da classi distinte (v. paragrafo successivo), in modo tale da ridurre al minimo le modifiche in caso di cambiamenti (ad esempio dell’algoritmo di risoluzione).

### Implementazione in Java

Unz funz

## Generazione e validazione del classificatore

### Requisiti

Una volta generati i dataset ed ottenuti i valori di riferimento del risolutore per i dataset completi, è possibile entrare nelle attività più specificatamente di data mining del progetto. In particolare, in questa fase si è interessati a valutare due aspetti:

1. l’efficacia predittiva di un classificatore costruito sulla base degli attributi scelti (utilizzando come confronto dataset risolti dal risolutore)
2. l’eventuale guadagno prestazionale nell’utilizzare come input al solver un dataset da cui siano state eliminate tutte le istanze che il classificatore abbia classificato come non schedulabili (questo dataset prenderà il nome di dataset “sfoltito”). Chiaramente, l’eventuale guadagno prestazionale dovrà essere valutato insieme agli altri parametri caratteristici della soluzione generata a partire dal dataset sfoltito (e.g. il numero dei task che sono risultati schedulabili, il costo complessivo della soluzione trovata, ecc.)

Conseguentemente, i requisiti per questo modulo possono essere riassunti come segue:

1. per ciascun insieme di dataset appartenenti alla stessa categoria di complessità e contenente n dataset, costruire un insieme di istanze da utilizzare per l’addestramento del classificatore utilizzando n-1 dataset e valutare poi il classificatore così costruito sul dataset n-esimo
2. utilizzare il classificatore così costruito per costruire, a partire dal dataset n-esimo, un nuovo dataset costituito dalle sole istanze definite come schedulabili dal classificatore (il dataset sfoltito a cui si accennava in precedenza)
3. eseguire il solver prima sul dataset completo e poi su quello sfoltito e memorizzare i risultati ottenuti dal solver in entrambi i casi insieme alle caratteristiche del classificatore (e.g. precisione, recall, #falsi positivi, #falsi negativi, errore pesato, ecc.)
4. effettuare una cross-validation utilizzando lo stesso procedimento di cui al punto 1 ripetuto n volte, ruotando i dataset utilizzati per l’addestramento e la validazione dei risultati
5. i passi fin qui descritti devono poter essere eseguiti in batch, utilizzando tutti i dataset memorizzati in una determinata cartella e memorizzando tutti i risultati in un singolo file

### Flow chart di esecuzione

### Implementazione in Java

# Classificatore utilizzato

Il classificatore utilizzato

# Risultati sperimentali

# Indice delle figure

[Figure 1: tempi di risoluzione per tipi diversi di dataset 13](#_Toc486842206)

# Riferimenti

1. Il modello con vincoli hard richiede la copertura di tutti i task pena la non fattibilità del problema [↑](#footnote-ref-1)
2. La capacità dello scheduler di gestire fino a 1.000 tasks e 100 risorse operative dipende fortemente dalle performance e dalle caratteristiche dell’hardware utilizzato per eseguire l’applicazione [↑](#footnote-ref-2)
3. I vincoli sui task, sebbene siano previsti sia dal codice che dal risolutore, non sono stati attivati e conseguentemente questo attributo risulta non utilizzato [↑](#footnote-ref-3)
4. “Record Breaking Optimization Results Using the Ruin and Recreate Principle” – Journal of Computational Physics, vol. 159, Issue 2, 10/04/2009, pp. 139 – 171 (<https://doi.org/10.1006/jcph.1999.6413>) [↑](#endnote-ref-1)