**Corso di Data Mining**

**Tesina di fine corso**

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

**di problemi WTRSP**

Autore: G. Perrone

Revisione: 4 – 10/07/17

Sommario

[1 Introduzione e scopo del progetto 3](#_Toc487490166)

[2 Descrizione del problema 4](#_Toc487490167)

[2.1 Requisiti 5](#_Toc487490168)

[2.1.1 Requisiti sulle risorse operative 5](#_Toc487490169)

[2.1.2 Requisiti sui tasks 5](#_Toc487490170)

[2.1.3 Requisiti sulle metriche (funzione obiettivo) 6](#_Toc487490171)

[2.1.4 Requisiti non funzionali (performance) 6](#_Toc487490172)

[3 Caratterizzazione dei dati 7](#_Toc487490173)

[3.1 Caratteristiche dei task per la classificazione 8](#_Toc487490174)

[3.1.1 Caratteristiche spaziali 8](#_Toc487490175)

[3.1.2 Caratteristiche temporali 9](#_Toc487490176)

[3.1.3 Caratteristiche legate alle risorse 10](#_Toc487490177)

[4 Generazione del dataset 11](#_Toc487490178)

[4.1 Parametri di configurazione 11](#_Toc487490179)

[4.1.1 Parametri generici e complessità dei dataset 11](#_Toc487490180)

[4.1.2 Parametri di distribuzione spaziale e temporale per le risorse 12](#_Toc487490181)

[4.1.3 Parametri di distribuzione temporale per i task 15](#_Toc487490182)

[5 Risoluzione del problema 16](#_Toc487490183)

[5.1 Performance del risolutore 16](#_Toc487490184)

[6 Applicazione sviluppata 17](#_Toc487490185)

[6.1 Rappresentazione delle entità 17](#_Toc487490186)

[6.2 Generazione dei dataset e relativa risoluzione 18](#_Toc487490187)

[6.2.1 Requisiti 18](#_Toc487490188)

[6.2.2 Flow chart di esecuzione 20](#_Toc487490189)

[6.2.3 Implementazione in Java 21](#_Toc487490190)

[6.3 Generazione e validazione del classificatore 23](#_Toc487490191)

[6.3.1 Requisiti 23](#_Toc487490192)

[6.3.2 Flow chart di esecuzione 24](#_Toc487490193)

[6.3.3 Implementazione in Java 25](#_Toc487490194)

[6.4 Valutazione del classificatore su istanze più complesse (valutazione “in Avanti”) 28](#_Toc487490195)

[6.4.1 Requisiti 28](#_Toc487490196)

[6.4.2 Flow-chart di esecuzione 29](#_Toc487490197)

[6.4.3 Implementazione in Java 30](#_Toc487490198)

[7 Classificatore utilizzato 31](#_Toc487490199)

[7.1 Selezione degli attributi 31](#_Toc487490200)

[7.2 Il classificatore 32](#_Toc487490201)

[8 Risultati sperimentali 33](#_Toc487490202)

[8.1 Abstract 33](#_Toc487490203)

[8.2 Dataset utilizzati 33](#_Toc487490204)

[8.3 Validazione del classificatore 35](#_Toc487490205)

[8.4 Utilizzo di un classificatore per la riduzione del tempo di risoluzione di scenari di uguale complessità 37](#_Toc487490206)

[8.4.1 Valutazione dei risultati 37](#_Toc487490207)

[8.4.2 Dati completi 40](#_Toc487490208)

[8.5 Validazione “in avanti” del classificatore 43](#_Toc487490209)

[8.5.1 Validazione del classificatore 43](#_Toc487490210)

[8.5.2 Valutazione dei risultati 44](#_Toc487490211)

[8.5.3 Dati completi 45](#_Toc487490212)

[9 Indice delle figure 46](#_Toc487490213)

[10 Riferimenti 47](#_Toc487490214)

# 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 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.

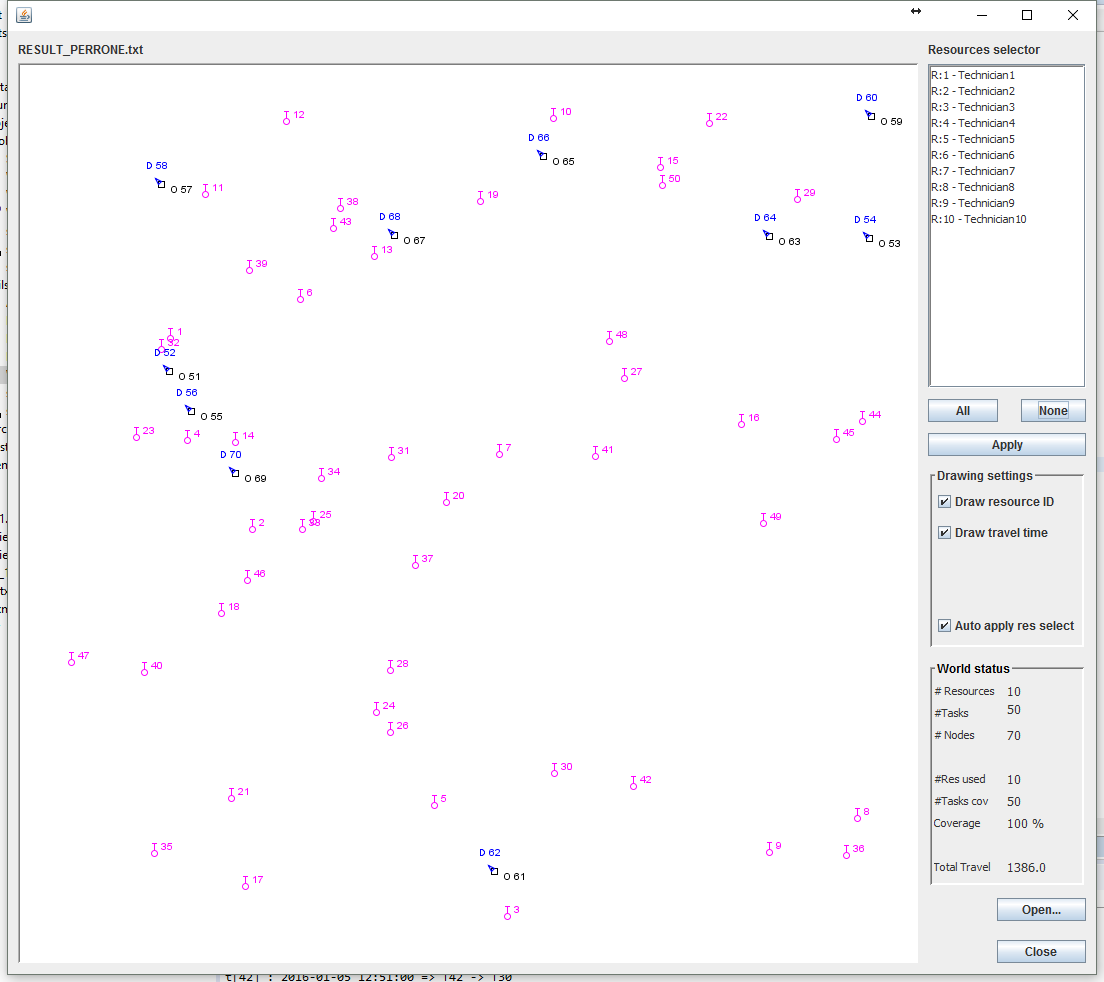


Figura : rappresentazione grafica di un dataset

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 e complessità dei dataset

Questo insieme di parametri sono generici e si riferiscono al “mondo” generato (e.g il numero dei task e 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 |

Tabella : parametri di configurazione generici per la creazione del dataset

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 |

Tabella : parametri di configurazione per la distribuzione spaziale e temporale delle risorse

#### 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).

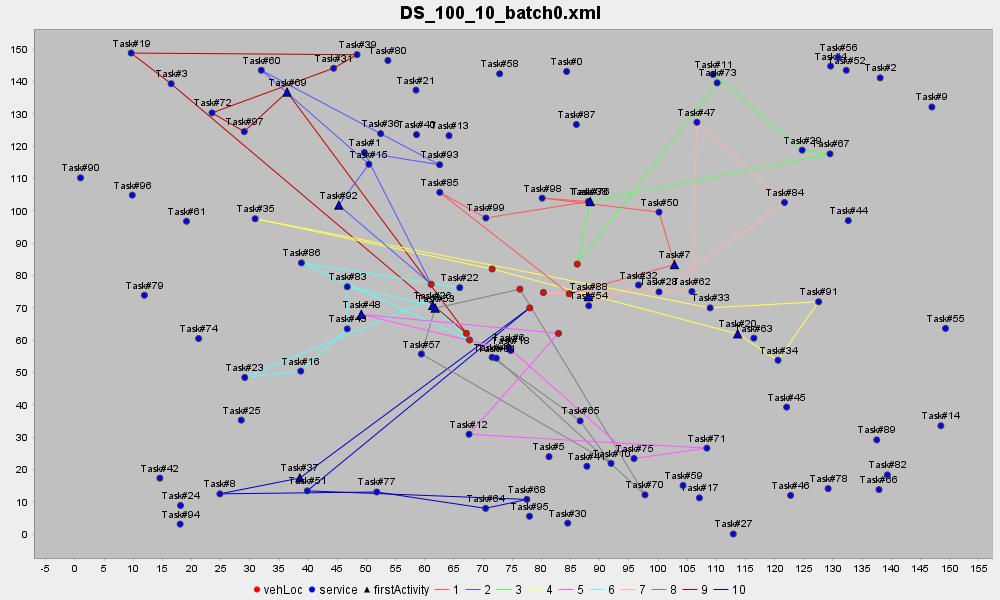


Figura : esempio di dataset (risolto) con posizioni di partenza dei task generate intorno al centro del piano

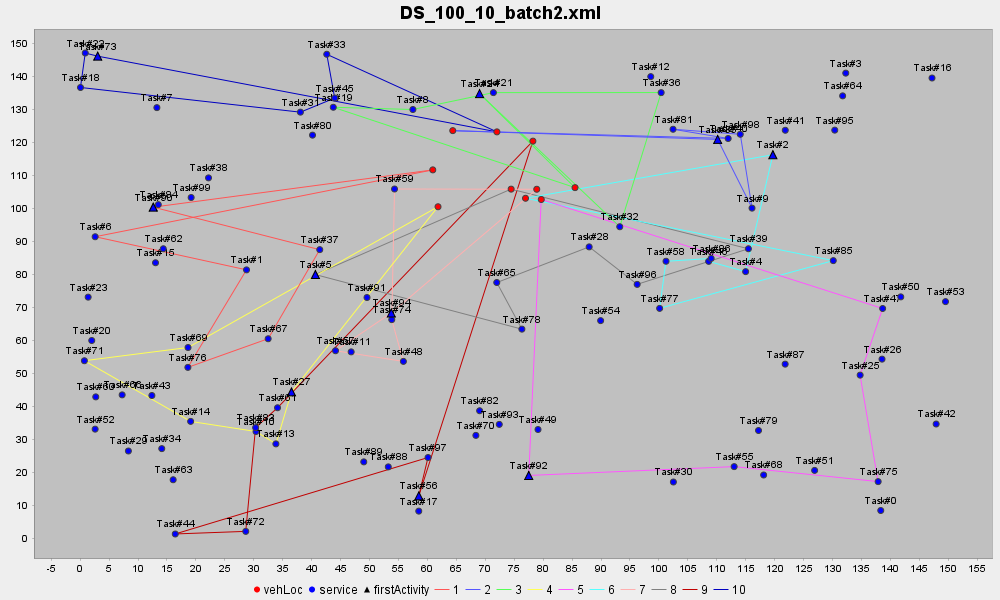


Figura : esempio di dataset (risolto) con posizioni di partenza dei task generate intorno alla parte "nord" del piano

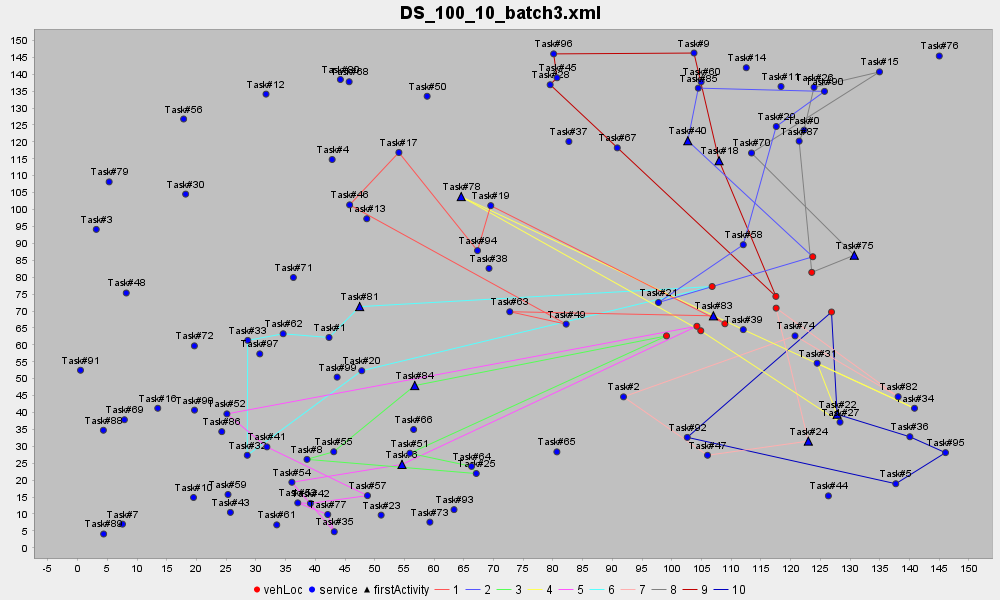


Figura : esempio di dataset (risolto) con posizioni di partenza dei task generate intorno alla parte "est" del piano

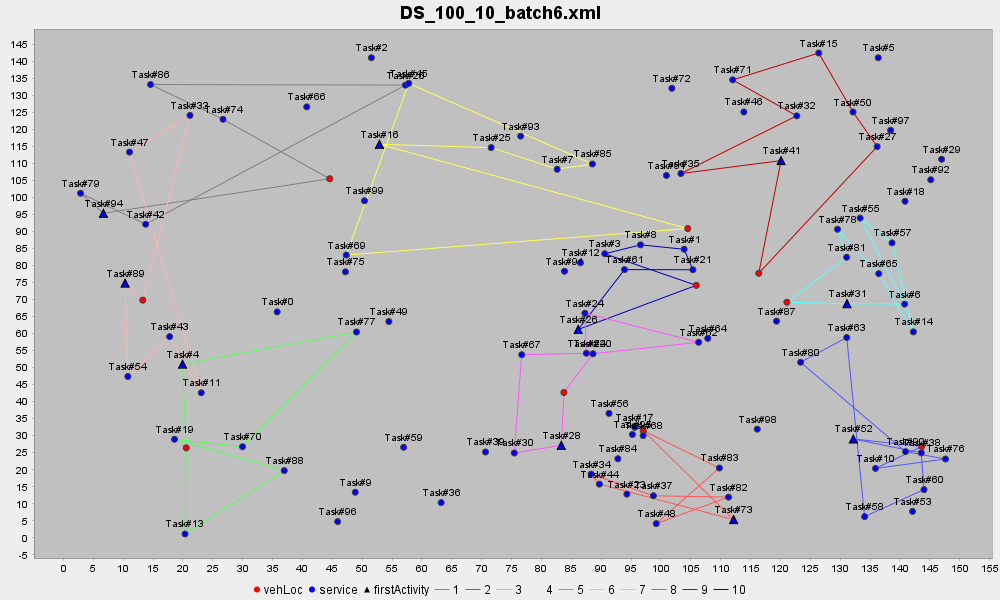


Figura : esempio di dataset (risolto) con posizioni di partenza dei task generate intorno alla parte "est" del piano

### 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) ) |

Tabella : parametri di configurazione per la distribuzione temporale dei task

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. Maggiori dettagli sono disponibili direttamente sul github del framework[[5]](#endnote-2)

## 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 valutazione 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 valutazione su istanze di complessità “>x” (e.g. 2000 task e 200 risorse), con produzione dei relativi files di analisi e statistica.

Per la generazione del classificatore e la sua valutazione verranno utilizzate le API dell’applicativo Weka.

## Rappresentazione delle entità

Per rappresentare le entità utilizzate dal progetto (e.g. risorse, task, dataset, etc.) sono state create classi specifiche che a loro volta utilizzano istanze di altre classi create per modellare entità specifiche di questo progetto (e.g. la classe TimeInterval che modella un intervallo temporale).

Ciascuna classe espone anche dei metodi specifici che vengono utilizzati per semplificare le attività principali con gli oggetti della classe stessa. La figura seguente riporta il diagramma UML delle classi implementate per rappresentare le entità del progetto, insieme ai loro *data members* e ai *metodi*.

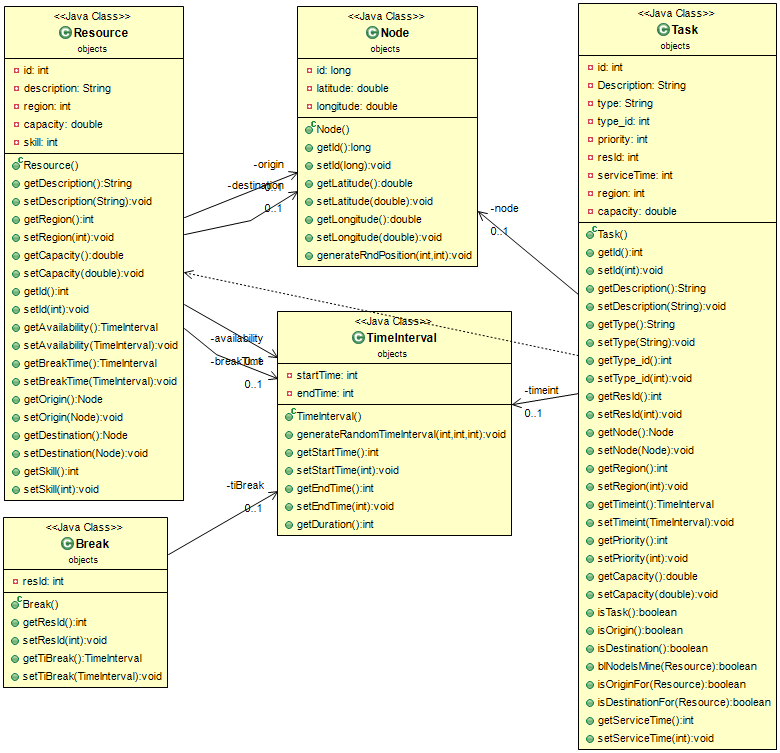


Figura : classi utilizzate per la rappresentazione delle entità principali

## 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 valutazione 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

La seguente figura riporta il diagramma UML, completo di *data members* e *metodi*, della classe principale BatchLauncher.java che implementa la generazione e risoluzione dei dataset in batch.

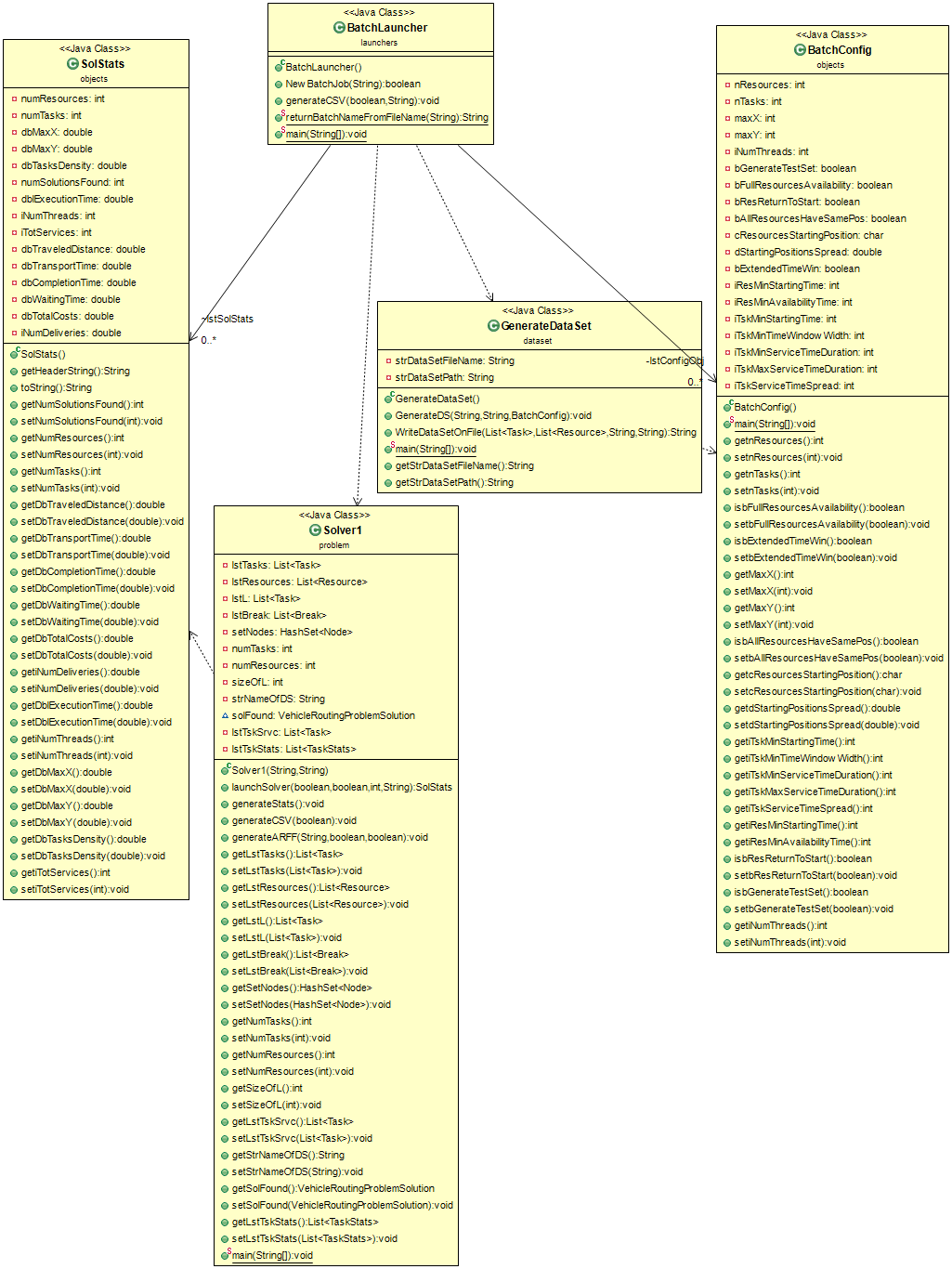


Figura : diagramma UML della classe BatchLauncher e sue dipendenze

Come si nota, la classe BatchLauncher utilizza istanze delle classi GenerateDataSet e Solver1, che rispettivamente si occupano di:

1. generare e memorizzare su disco un dataset secondo i parametri contenuti nell’oggetto istanza della classe BatchConfig, che viene passato al costruttore della classe per impostare i parametri;
2. caricare in memoria e risolvere il dataset il cui nome di file e path completo è passato come parametro al costruttore. La classe Solver1 espone anche metodi per:
   1. generazione degli attributi a partire dai dati del dataset e della risoluzione;
   2. generazione di file .arff e .csv per lo scambio di informazioni con il classificatore e la successiva analisi dei risultati di risoluzione.

I risultati dell’esecuzione vengono memorizzati in istanze della classe SolStats, che espone getters e setters per l’impostazione ed il recupero di una serie di informazioni relative alla soluzione trovata. In particolare, vengono memorizzate i seguenti valori:

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
| int | numResources | number of resources in this execution |
| int | numTasks | number of tasks in this execution |
| double | dbMaxX | maximum value for X |
| double | dbMaxY | maximum value for Y |
| double | dbTasksDensity | density of the tasks |
| int | numSolutionsFound | Number of solutions found by the solver |
| double | dblExecutionTime | Running time of the solver |
| int | iNumThreads | number of threads used |
| int | iTotServiced | total number of tasks serviced |
| double | dbTraveledDistance | traveled distance for the best solution |
| double | dbTransportTime | transport time for the best solution |
| double | dbCompletionTime | completion time for the best solution |
| double | dbWaitingTime | waiting time for the best solution |
| double | dbTotalCosts | total costs for the best solution |

Tabella : risultati memorizzati per ciascuna soluzione trovata nella classe SolStats

## 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 “**potato**” (dall’inglese “*pruned*”). Chiaramente, l’eventuale guadagno prestazionale dovrà essere valutato insieme agli altri parametri caratteristici della soluzione generata a partire dal dataset potato (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 potato a cui si accennava in precedenza);
3. eseguire il solver prima sul dataset completo e poi su quello potato 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 valutazione 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.

### Flowcharts.pngFlow chart di esecuzione

Figura : flusso di esecuzione del modulo di generazione e validazione del classificatore

Il flusso di esecuzione del modulo è, ad alto livello, costituito da due loop nidificati:

1. il primo loop (quello più esterno) consente di eseguire le attività successive in modalità batch, semplificando notevolmente il lavoro quando si devono effettuare valutazioni su più dataset;
2. il secondo loop implementa la *cross-validation*, e consente quindi, per ciascun insieme di n dataset (e.g. tutti i dataset presenti nella cartella 100-1000-d01), di utilizzare n-1 dataset per la generazione delle istanze per l’addestramento del classificatore e la rimanente n-esima istanza per la valutazione. Si è quindi implementata una *n-fold validation*.

All’interno del secondo loop viene inoltre creato il dataset potato utilizzando il classificatore sul dataset di test (di cui ovviamente non viene tenuta in considerazione l’attributo che rappresenta la classe, ossia la schedulabilità in base alla risoluzione già effettuata).

Viene quindi lanciato il risolutore prima sul dataset completo e poi su quello potato allo scopo di valutare le differenze di performance in termini di efficienza (ossia il tempo richiesto dal risolutore per trovare una soluzione ottima) ed efficacia (ossia quanti task sono stati definiti come schedulabili rispetto al dataset completo e quanto sono variati i valori di costo della soluzione trovata sul dataset potato).

I risultati del confronto sono memorizzati prima in memoria e poi su disco in modo da semplificare le attività di valutazione dei risultati.

### Implementazione in Java

La seguente figura riporta il diagramma UML, completo di *data members* e *metodi*, della classe principale BatchClassifier.java che implementa la generazione e risoluzione dei dataset in batch.

Sono state riportate anche le principali classi delle API Weka che vengono utilizzate direttamente dalle classi dell’applicazione; in particolare si distinguono le seguenti classi:

* Instances: implementa un set di istanze;
* Attribute: implementa un attributo;
* ArffLoader: implementa un lettore di file .arff che viene utilizzato per popolare un oggetto della classe Instances;
* ArffReader: richiamata dalla ArffLoader, implementa il lettore vero e proprio di files .arff;
* AttributeSelectedClassifier: implementa un classificatore con selezione degli attributi.

Per semplificare il diagramma, per le classi delle API Weka non sono mostrati né i data members né i metodi. Non sono inoltre mostrate le classi utilizzate per la valutazione del classificatore che sono incluse nel package weka.classifiers.evaluation.Evaluation.

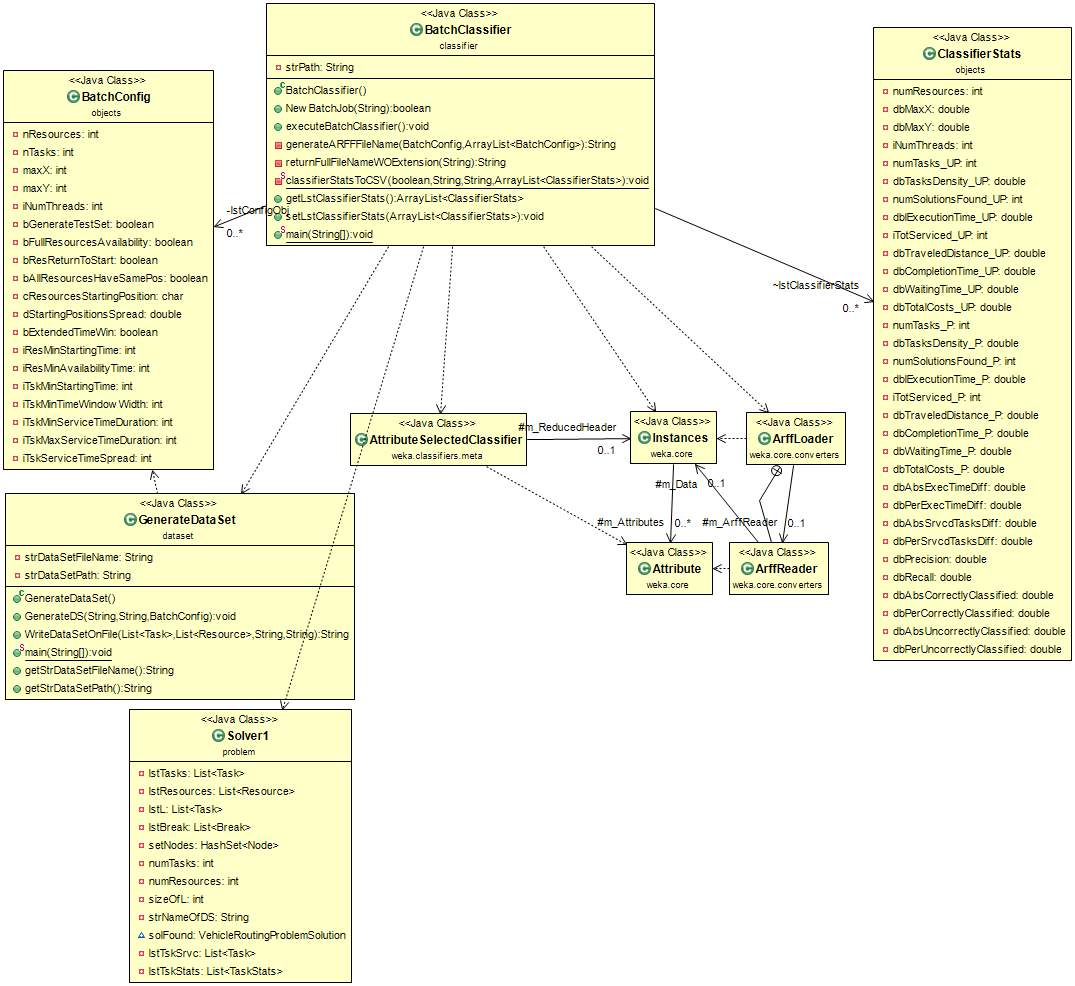


Figura : diagramma UML della classe BatchClassifier e relative dipendenze

Il flusso di esecuzione è gestito dalla classe BatchClassifier, che utilizza gli stessi file di configurazione xml usato per la generazione dei dataset. Questi file sono, come già detto, essenzialmente un dump di istanze della classe BatchConfig, dove ciascuna istanza descrive le caratteristiche del dataset.

Nel caso specifico, la classe BatchClassifier andrà a leggere tutti i file di configurazione presenti nella cartella resources/batch e, per ciascun file, eseguirà le attività descritte nel paragrafo precedente.

I risultati dell’esecuzione sono memorizzati in istanze della classe ClassifierStats che sono poi memorizzati su file csv utilizzando i metodi esposti dalla classe stessa – tali metodi utilizzano metodi della classe Java Reflection per assicurare la massima flessibilità in caso di modifiche alla composizione della classe stessa.

I parametri memorizzati sono quindi i data members della classe ClassifierStats, che sono riportati nella tabella seguente:

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| int | numResources | number of resources in this execution |
| double | dbMaxX | maximum value for X |
| double | dbMaxY | maximum value for Y |
| int | iNumThreads | number of threads used |
| int | numTasks\_UP | number of tasks in this execution |
| double | dbTasksDensity\_UP | density of the tasks |
| int | numSolutionsFound\_UP | Number of solutions found by the solver |
| double | dblExecutionTime\_UP | Running time of the solver |
| int | iTotServiced\_UP | total number of tasks serviced |
| double | dbTraveledDistance\_UP | traveled distance for the best solution |
| double | dbCompletionTime\_UP | completion time for the best solution |
| double | dbWaitingTime\_UP | waiting time for the best solution |
| double | dbTotalCosts\_UP | total costs for the best solution |
| int | numTasks\_P | number of the pruned tasks |
| double | dbTasksDensity\_P | tasks density for the pruned set of tasks |
| int | numSolutionsFound\_P | Number of solutions found by the solver |
| double | dblExecutionTime\_P | Running time of the solver |
| int | iTotServiced\_P | total number of tasks serviced |
| double | dbTraveledDistance\_P | traveled distance for the best solution |
| double | dbCompletionTime\_P | completion time for the best solution |
| double | dbWaitingTime\_P | waiting time for the best solution |
| double | dbTotalCosts\_P | total costs for the best solution |
| double | dbAbsExecTimeDiff | difference between the execution times (absolute) |
| double | dbPerExecTimeDiff | difference between the execution times (%) |
| double | dbAbsSrvcdTasksDiff | difference between the execution times (absolute) |
| double | dbPerSrvcdTasksDiff | difference between the execution times (%) |
| double | dbPrecision | classifier precision |
| double | dbRecall | classifier recall |
| double | dbAbsCorrectlyClassified | number of correctly classified instances (absolute) |
| Double | dbPerCorrectlyClassified | number of correctly classified instances (%) |
| Double | dbAbsUncorrectlyClassified | number of uncorrectly classified instances (absolute) |
| Double | dbPerUncorrectlyClassified | number of uncorrectly classified instances (%) |

Tabella : parametri memorizzati nella classe ClassifierStats

## Valutazione del classificatore su istanze più complesse (valutazione “in Avanti”)

In questo caso si vogliono valutare efficacia ed efficienza di un classificatore costruito su un dataset di istanze di complessità “x” (ove per complessità si intende la tupla numero di risorse-numero di task-densità) su dataset di complessità maggiore di x.

Lo scopo ultimo è quello di valutare se, utilizzando un classificatore addestrato su dataset gestibili dal risolutore, sia possibile affrontare problemi di complessità maggiore (anche non risolvibili in tempi utili dal risolutore) e se sì, con quali risultati. Si noti come la differenza fondamentale rispetto a quanto visto finora sia nella differenza trai dataset utilizzati per il training set (di complessità “x”) ed i dataset utilizzati per il test set (che avranno complessità >x).

Si valuterà quindi come si comporta il classificatore nel gestire dataset più complessi rispetto a quelli che sono stati utilizzati per il suo addestramento. Per questo motivo, per semplicità questo modulo è stato rinominato come modulo di valutazione “in avanti” del classificatore.

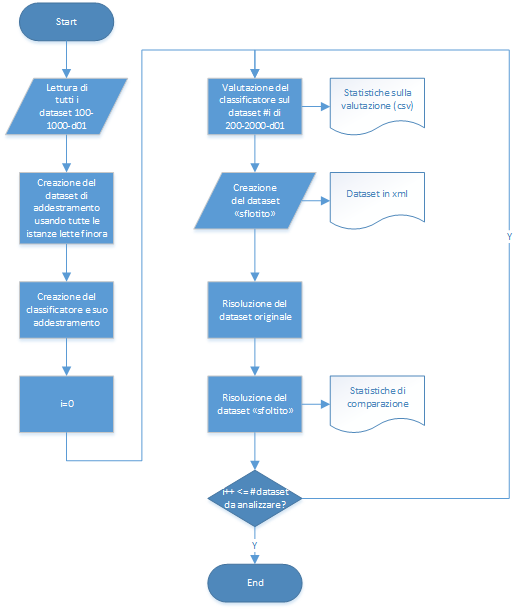
### Requisiti

Come già detto, si desidera addestrare un classificatore utilizzando un determinato tipo di dataset e poi valutarne le performance su dataset più complessi. Data la natura dell’attività, non è necessario un approccio “generalistico” (in cui è quindi possibile parametrizzare la configurazione dell’esecuzione) ma si è scelto di fissare direttamente nel codice alcuni dei parametri di esecuzione (come ad esempio la scelta dei dataset per l’addestramento e di quelli per la valutazione).

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

1. deve essere possibile creare un classificatore utilizzando tutti i dataset di un determinato livello di complessità;
2. utilizzando il classificatore appena creato, deve essere possibile effettuare una valutazione delle performance su un insieme di dataset di complessità maggiore e deve essere possibile memorizzare i risultati di questa valutazione;
3. deve essere inoltre possibile creare dei dataset potati di complessità maggiore usando il classificatore creato ai punti precedenti allo scopo di valutare le performance del risolutore sui questi stessi dataset.

### Flow-chart di esecuzione

La seguente figura illustra il flusso di esecuzione del modulo di valutazione “in avanti” del classificatore.

Come si può vedere, il modulo procede con un’esecuzione lineare nella prima fase di addestramento del classificatore e procede poi con un loop per la fase di valutazione sui dataset più complessi. All’interno del loop di valutazione vengono generati due tipi diversi di statistiche:

Figura : flow chart di esecuzione del modulo di validazione del classificatore

1. un file .csv contenente i risultati della valutazione del classificatore sulle singole istanze;
2. un file .csv contenente i risultati del confronto sulle performance del risolutore sul dataset potato e non.

### Implementazione in Java

La seguente figura riporta il diagramma UML, completo di *data members* e *metodi*, della classe principale ForwardClassifier.java che implementa la valutazione “in avanti” del classificatore. Anche in questo caso si faranno uso della API Weka per la lettura di files .arff, l’implementazione di un set di istanze, ecc. ecc. (per maggiori informazioni si rimanda al capitolo 6.3.3).

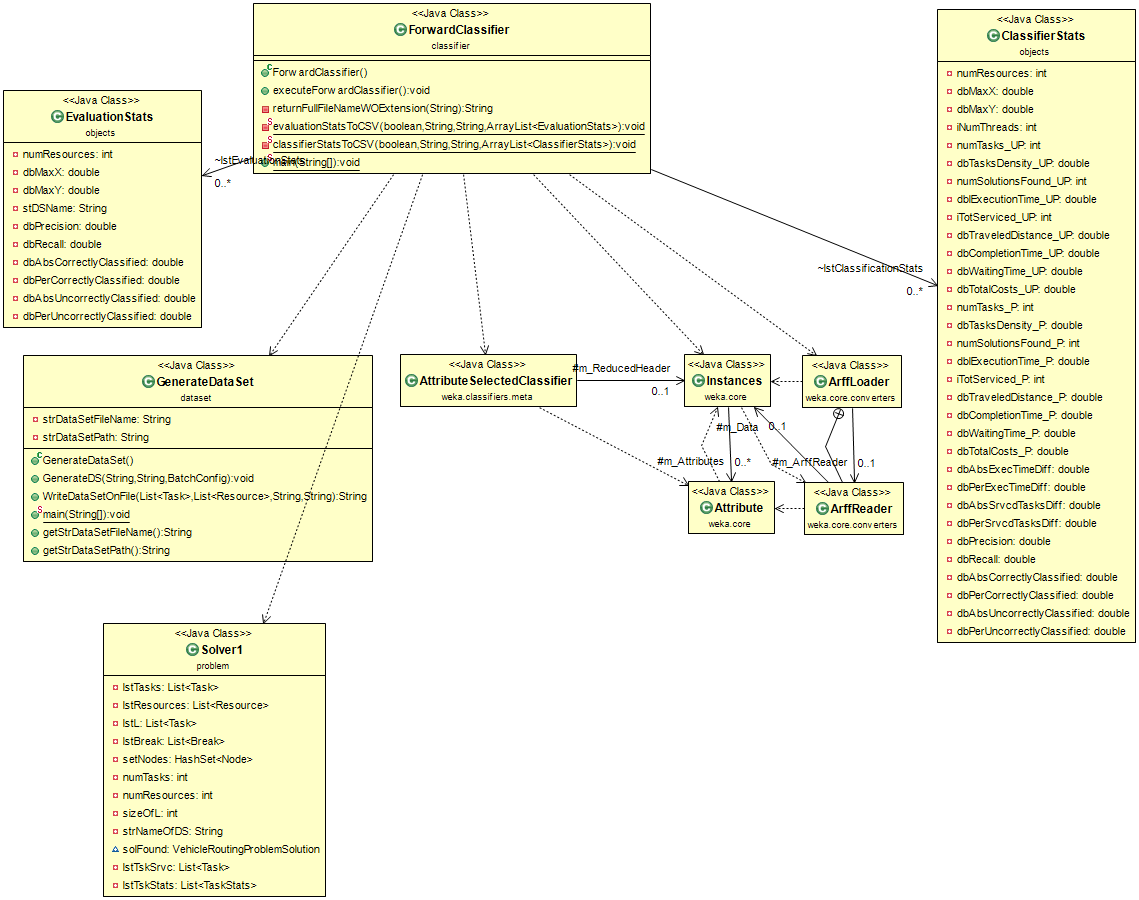


Figura : diagramma UML del modulo di validazione "in avanti"

La classe che gestisce il flusso di attività è la classe ForwardClassifier.java, tramite il metodo executeForwardClassifier.

Come già accennato, la scelta delle directory dove sono memorizzati i files dei dataset da utilizzare per l’addestramento e la valutazione è stata implementata per semplicità tramite una costante String all’interno del codice. Per comodità, l’assegnazione viene effettuata subito dopo la *signature* del metodo ed è quindi facilmente possibile modificare il codice per poter gestire questi valori come parametri di esecuzione a *run-time*:

**public** **void** executeForwardClassifier() {

String strPathForTraining = "output/Batch100\_1000\_d01";

String strPathToBeEvaluated = "output/Batch200\_2000\_d01";

Nel caso specifico, come si vede, per la valutazione sono stati scelti dataset 100-1000-d01 per l’addestramento e 200-2000-d01 per la valutazione.

I risultati della valutazione del classificatore sono memorizzati tramite istanze della classe EvaluationStats, che espone anche un metodo per la scrittura su disco dei contenuti dell’istanza. I parametri memorizzati sono riportati nella seguente tabella:

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
| int | numResources | number of resources in this execution |
| double | dbMaxX | maximum value for X |
| double | dbMaxY | maximum value for Y |
| String | stDSName | Filename of the dataset being validated |
| double | dbPrecision | classifier precision |
| double | dbRecall | classifier recall |
| double | dbAbsCorrectlyClassified | number of correctly classified instances (absolute) |
| Double | dbPerCorrectlyClassified | number of correctly classified instances (%) |
| Double | dbAbsUncorrectlyClassified | number of uncorrectly classified instances (absolute) |
| Double | dbPerUncorrectlyClassified | number of uncorrectly classified instances (%) |

Tabella : data members della classe EvaluationStats usati per la memorizzazione dei risultati di validazione

# Classificatore utilizzato

Il classificatore utilizzato è, come già accennato, un classificatore con selezione degli attributi, implementato dalla classe meta.AttributeSelectedClassifier delle API Weka.

## Selezione degli attributi

L’utilizzo di un classificatore con rimozione degli attributi permette di ottenere dei miglioramenti di performance nell’utilizzo del classificatore, dal momento che questo viene addestrato prima e applicato poi utilizzando solo un sotto-insieme degli attributi disponibili. La fase di selezione degli attributi selezionerà quindi, dall’insieme degli attributi disponibili nel dataset, solo quelli che hanno una maggiore correlazione con l’attributo che è stato impostato come classe, eliminando quelli che sono o ridondanti (ossia replicano in parte o completamente informazioni già disponibili tramite altri attributi) o irrilevanti (ossia che non hanno nessuna correlazione con il valore assunto dalla classe). A differenza dei metodi basati sulla riduzione della dimensionalità, in questo caso lo spazio degli attributi non viene trasformato ma semplicemente ridotto in numerosità utilizzando un sotto-insieme degli attributi disponibili inizialmente.

Chiaramente, per poter effettuare una selezione di attributi sono necessarie due azioni:

1. un metodo di valutazione dell’attributo, ossia un algoritmo che introduca una metrica di correlazione tra l’attributo in esame e la classe;
2. un metodo di ricerca degli attributi, ossia un algoritmo che determini come ci si debba muovere all’interno dell’insieme degli attributi per poter determinare il sotto-insieme migliore.

Il metodo di ricerca e la metrica di valutazione degli attributi dipendono chiaramente dagli algoritmi che sono scelti per la fase di ricerca e valutazione.

Non essendo state modificate le impostazioni di default del classificatore, nel nostro caso saranno utilizzati i seguenti algoritmi:

* per la valutazione viene utilizzato il valutatore CfsSubsetEval, che valuta insiemi di attributi misurando allo stesso tempo la correlazione dell’insieme con la classe e la ridondanza degli attributi che fanno parte dell’insieme. Vengono preferiti insiemi ad alta correlazione con la classe e bassa ridondanza interna;
* per la ricerca viene utilizzato l’algoritmo BestFirst, che implementa una combinazione di ricerca e selezione in avanti e all’indietro.

## Il classificatore

Il classificatore utilizzato per default dalla classe AttributeSelectedClassifier è la classe J48, che implementa un generatore di alberi decisionali basati sull’algoritmo C4.5. In pratica, viene costruito un albero decisionale utilizzando l’information gain come misura di rilevanza degli attributi nei confronti della classe.

A differenza dell’algoritmo ID3, può lavorare anche con attributi numerici (come nel nostro caso), evitando quindi la necessità di dover ricorrere ad un algoritmo di discretizzazione a priori dei valori degli attributi. C4.5 prevede inoltre una fase di potatura dell’albero decisionale dopo che questo è stato creato, che ha come obiettivo quello di ridurre il numero totale di parametri (nodi dell’albero) che rappresentano l’albero stesso, senza però compromettere la rappresentatività dell’albero nei confronti del lavoro di classificazione. La potatura permette inoltre di eliminare o comunque contenere il fenomeno dell’*overtraining* o *overfitting*, in cui un classificatore “ricalca” troppo fedelmente il dataset su cui è stato addestrato, causando quindi una perdita di generalità nell’efficacia del classificatore nell’essere utilizzato su altri dataset.

Tra le varie opzioni disponibili, J48 permette di disattivare o meno la potatura, che però per default è attiva ed è pertanto rimasta attivata durante le attività di addestramento e valutazione di questo progetto. Tra le opzioni disponibili non è comunque possibile scegliere una parte del dataset di training come *validation set* per la potatura, dal momento che l’algoritmo implementato in Weka utilizza tutto il dataset e valuta l’errore di ciascun sotto-albero per decidere se il sotto-albero deve essere potato o meno[[6]](#endnote-3).

# Risultati sperimentali

## Abstract

I risultati sperimentali mostrano che l’utilizzo di un classificatore introduce dei **significativi vantaggi** nella risoluzione dei problemi VRPTW in termini di tempo di esecuzione del risolutore, al costo di marginali (e talvolta assenti) riduzioni nel numero di task schedulati al termine dell’esecuzione. Il vantaggio appare aumentare all’aumentare della complessità dello scenario, mentre per scenari piccoli i risultati sono meno incoraggianti, sicuramente a causa del numero ridotto di istanze utilizzate per l’addestramento del classificatore e del ridotto tempo di esecuzione del risolutore anche sul dataset completo.

Dai risultati riscontrati emerge inoltre che, tramite l’introduzione di un classificatore per effettuare una potatura preventiva del dataset, è possibile approcciare problemi di complessità (in termini di numero di task e risorse) con vantaggi che possono arrivare fino ad un risparmio del 25% del tempo di risoluzione. Anche se questo miglioramento non può in assoluto garantire la possibilità di risolvere problemi altrimenti non risolvibili nei tempi tipicamente utili per un normale utilizzo di un risolutore di problemi VRP, si tratta comunque di un miglioramento sostanziale i cui benefici sono da non sottovalutare.

Conseguentemente, riteniamo che l’introduzione di un classificatore con selezione di attributi possa essere vantaggiosamente utilizzata nell’ambito della risoluzione di problemi VRP.

## Dataset utilizzati

Per l’esecuzione dei test e la generazione dei dati sono stati utilizzati 60 dataset generati tramite il modulo di generazione dataset descritto nel capitolo 6.2.

Per la definizione delle categorie di complessità, sono stati definite 5 coppie #risorse-#task e per ciascuna coppia sono state presi in considerazione tre diversi valori della densità dei task, come evidenziato nella seguente tabella:

|  |  |  |  |
| --- | --- | --- | --- |
| #Risorse | #Task | Densità[[7]](#footnote-4) | Gruppo dataset |
| 10 | 100 | 0,1 | 10-100-d01 |
| 10 | 100 | 0,05 | 10-100-d005 |
| 10 | 100 | 0,01 | 10-100-d001 |
| 20 | 200 | 0,1 | 20-200-d01 |
| 20 | 200 | 0,05 | 20-200-d005 |
| 20 | 200 | 0,01 | 20-200-d001 |
| 100 | 1000 | 0,1 | 100-1000-d01 |
| 100 | 1000 | 0,05 | 100-1000-d005 |
| 100 | 1000 | 0,01 | 100-1000-d001 |
| 200 | 2000 | 0,1 | 200-2000-d01 |
| 200 | 2000 | 0,05 | 200-2000-d005 |
| 200 | 2000 | 0,01 | 200-2000-d001 |

Tabella :gruppi di dataset utilizzati per la sperimentazione

Per ciascun gruppo sono stati poi generati 5 dataset utilizzando sempre gli stessi parametri di configurazione per il generatore di dataset, ossia:

| Parametro | Valore | Descrizione / Note |
| --- | --- | --- |
| bResReturnToStart | true | Tutte le risorse rientrano al punto di partenza a fine turno |
| bAllResourcesHaveSamePos | false | Le risorse partono da punti diversi |
| cResourcesStartingPosition | C | I punti di partenza delle risorse sono definiti in un intorno del centro del piano |
| dStartingPositionsSpread | 10.0 | Raggio di spread dei punti di partenza delle risorse dal centro del piano |
| iResMinStartingTime | 480 | Valore minimo di partenza della finestra di disponibilità delle risorse (8:00) |
| iResMinAvailabilityTime | 480 | Valore minimo dell’ampiezza della finestra di disponibilità delle risorse (8 h) |
| iTskMinStartingTime | 480 | Valore minimo di partenza della finestra di disponibilità di un task (8:00) |
| iTskMinTimeWindowWidth | 240 | Valore minimo dell’ampiezza della finestra di disponibilità di un task (4 h) |
| iTskMinServiceTimeDuration | 30 | Durata minima di un task (30 min) |
| iTskMaxServiceTimeDuration | 90 | Durata massima di un task (90 min) |
| iTskServiceTimeSpread | 30 | Spread del momento di partenza della finestra di disponibilità del task rispetto a iTskMinStartingTime |

Tabella : parametri di configurazione utilizzati per la generazione dei dataset

Per maggiori informazioni sui parametri di configurazione del generatore di dataset si rimanda al capitolo 4.

I 5 dataset così costruiti sono stati quindi utilizzati a rotazione per l’addestramento e la valutazione del classificatore (*5-fold validation*).

## Validazione del classificatore

Come prima cosa, si rende necessario analizzare il comportamento del classificatore, ed in particolare la sua accuratezza nel predire se un dato task risulterà schedulabile o meno. La seguente tabella riassume i risultati sperimentali per quanto riguarda la precisione del classificatore.

|  |  |  |  |
| --- | --- | --- | --- |
| **Dataset** | Precision (MIN) | Precision (MAX) | Precision (AVG) |
| ***10-100*** | 63,72% | 82,81% | 71,4% |
| ***20-200*** | 60,78% | 81,17% | 73,5% |
| ***100-1000*** | 70,16% | 89,06% | 80,1% |
| ***200-2000*** | 78,14% | 94,36% | 86,4% |

Tabella : precisione del classificatore al variare della complessità

Si può facilmente notare come i valori medi di precisione siano sufficientemente alti per poter presumere che il classificatore, a valle della selezione degli attributi, possa effettivamente classificare le varie istanze con buoni risultati.

Gli stessi risultati sono graficati utilizzando un box plot[[8]](#footnote-5) nel seguente grafico:

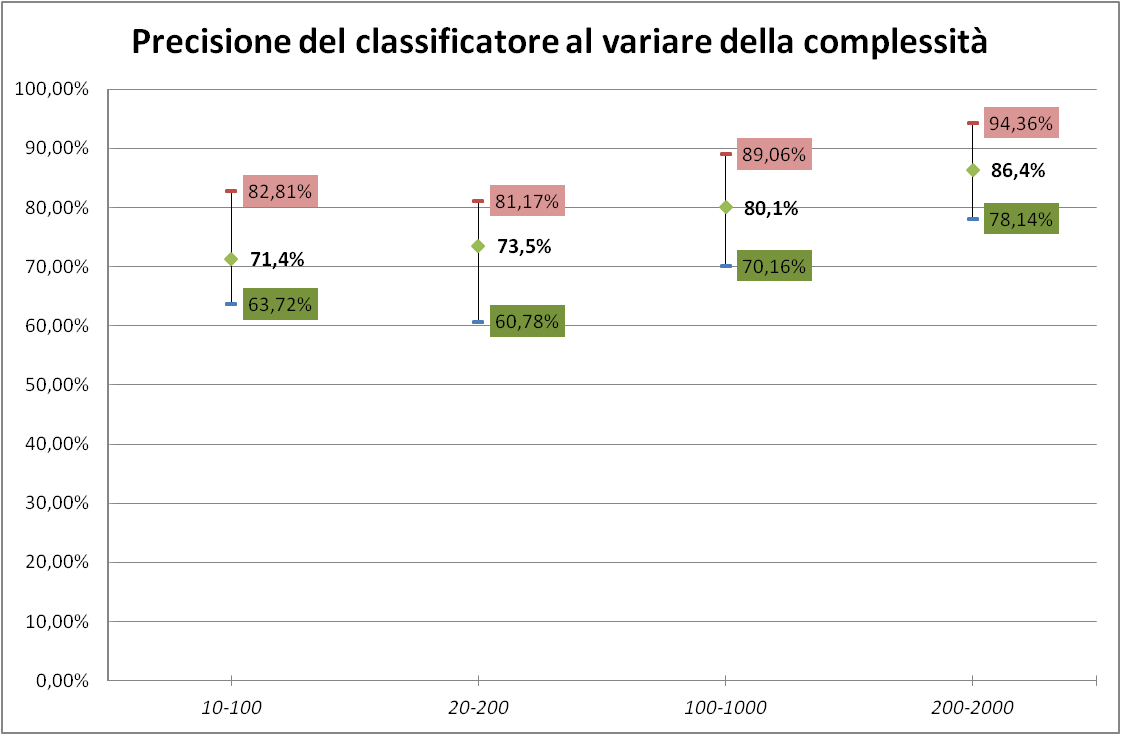


Figura : precisione del classificatore al variare della complessità (box plot)

Come già detto, il valor medio della precisione del classificatore utilizzato varia da un minimo del 71,5% per dataset non complessi fino ad arrivare al **86,4% per dataset 200-2000**. Questo progressivo miglioramento è senz’altro dovuto al fatto che all’aumentare della complessità aumenta il numero delle istanze utilizzate per l’addestramento e conseguentemente migliorano le performance del classificatore.

Dal momento che stiamo utilizzando numerosi campioni per ciascun gruppo di complessità, è interessante analizzare il comportamento della deviazione standard dei parametri presi in considerazione per valutare quanto siano “spalmati” i singoli valori. Il seguente grafico riporta quindi i valori della deviazione standard per Precision, Recall e % di istanze correttamente e non correttamente classificate.

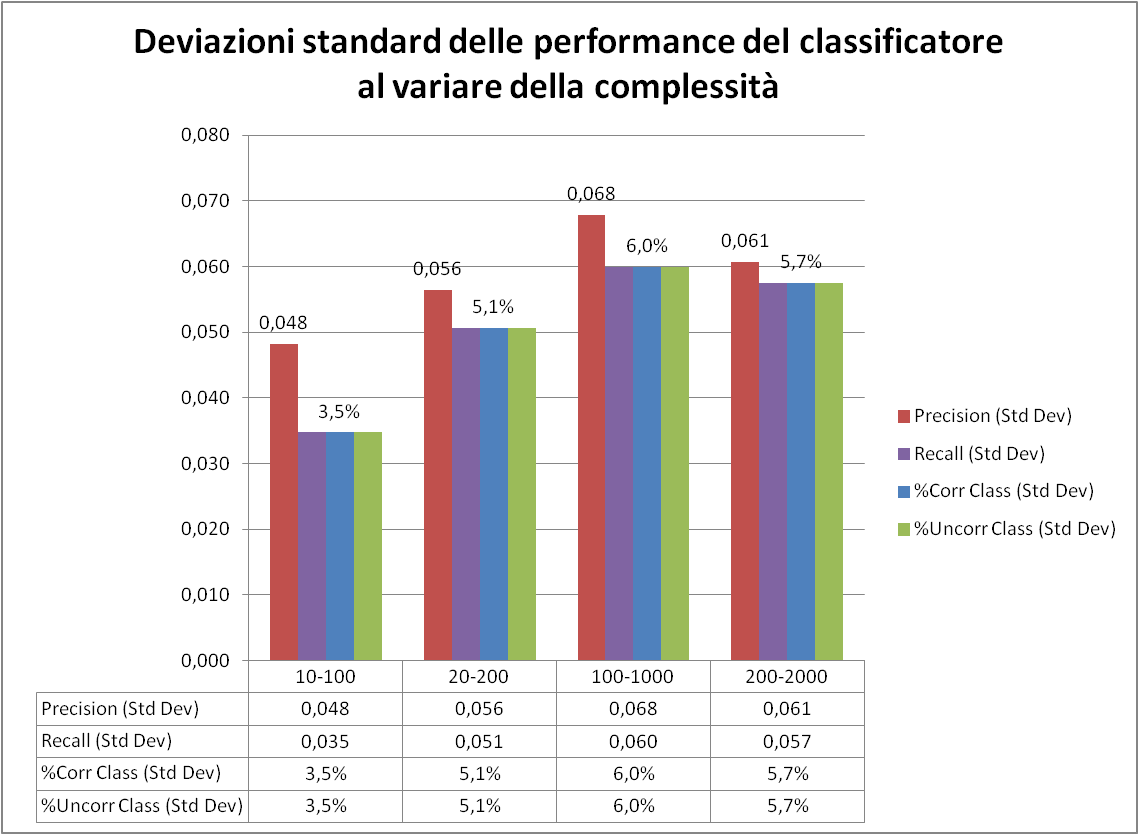


Figura : deviazioni standard delle performance del classificatore al variare della complessità

Come si nota, i valori sono bassi il che indica un minimo scostamento trai singoli campioni.

## Utilizzo di un classificatore per la riduzione del tempo di risoluzione di scenari di uguale complessità

In questa prima sezione analizzeremo i risultati sperimentali legati all’utilizzo di un classificatore addestrato su scenari di complessità “x” e applicato poi per la potatura di dataset di uguale complessità. In altre parole, utilizzeremo training set di complessità uguale ai test set.

Come già descritto nel capitolo 6.3, in questa fase, si è implementato un processo di 5-fold validation, per cui per ciascun gruppo di dataset da analizzare si sono utilizzati 5 dataset, di cui 4 impiegati a rotazione per l’addestramento ed il rimanente per la valutazione. Per ciascun gruppo (e.g. 10-100-d01) sono quindi disponibili 5 risultati di valutazione; tralasciando a questo punto la densità dei task, che ha sì un impatto sulle performance del risolutore ma non è necessariamente un parametro controllabile in un ambiente di esecuzione reale, si sono ottenuti risultati per i 4 macro-gruppi rappresentati dalle 4 coppie #risorse-#task che sono state prese in considerazione.

### Valutazione dei risultati

Dal momento che lo scopo del progetto consiste nel valutare gli eventuali miglioramenti di efficienza ed efficacia derivanti dall’impiego di un classificatore nella risoluzione di problemi VRP, cominciamo ad analizzare come variano le performance di risoluzione al variare della complessità del problema.

Il seguente grafico illustra in modo sintetico le differenze percentuali medie di tempo di esecuzione e numero di task schedulati al variare della complessità.

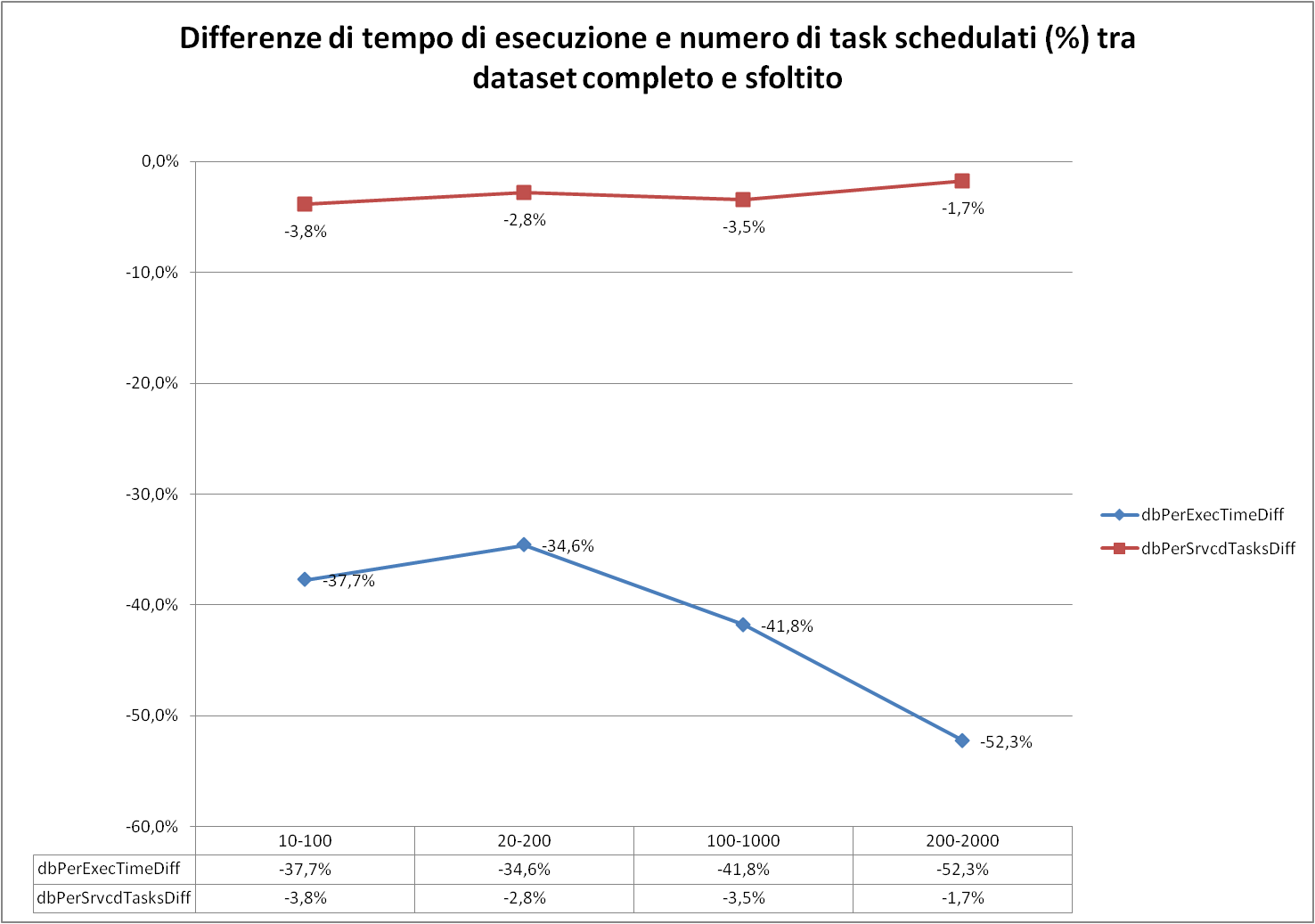


Figura : differenze di tempo di esecuzione e numero task schedulati al variare della complessità

Unitamente al valor medio, è interessante valutare anche l’ampiezza dello scarto tra valor minimo e valor massimo per i differenti gruppi di dataset per i valori di guadagno percentuale del tempo di esecuzione e della variazione nel numero di task schedulati.

Analizziamo quindi tramite *box plot* le caratteristiche di distribuzione della differenza del tempo di esecuzione.

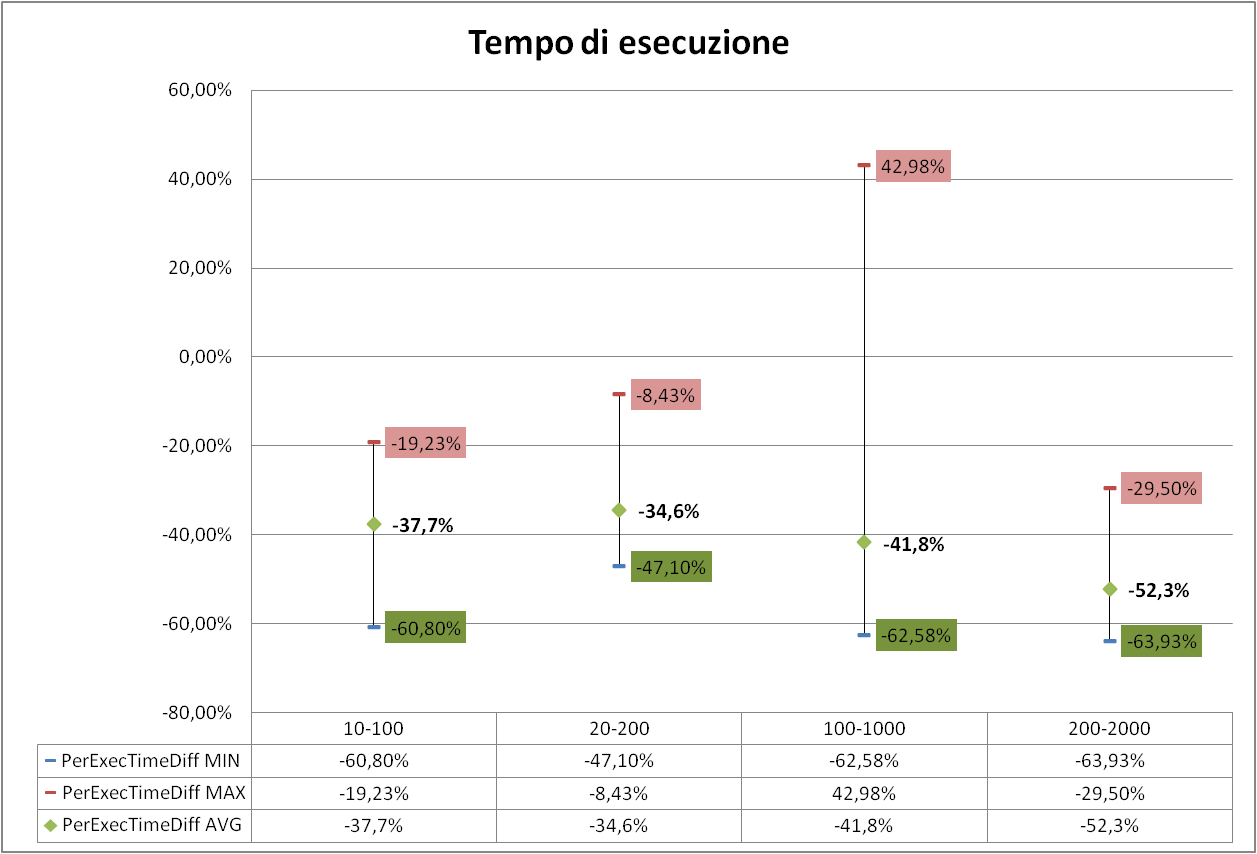


Figura : *box plot* della variazione del tempo di esecuzione al variare della complessità

Come già notato, le performance tendono a migliorare all’aumentare della complessità del dataset analizzato; si nota però che in corrispondenza del gruppo 100-1000 si ha un *outlier[[9]](#footnote-6)* corrispondente ad un significativo peggioramento del tempo di esecuzione del risolutore utilizzando il dataset potato piuttosto che quello completo.

Questo fenomeno si riscontra solo con uno dei dataset del grupppo 100-1000-d0,01 per il quale, nonostante buoni valori di *precision* e *recall* del classificatore, le performance di esecuzione del risolutore calano notevolmente. Il problema andrebbe analizzato in maggiore dettaglio analizzando le caratteristiche specifiche del dataset in esame e confrontandole con quelle del dataset potato per evidenziare eventuali caratteristiche nella distribuzione degli attributi che potrebbero spiegare questo comportamento alla luce anche dell’algoritmo di risoluzione specifico usato dal risolutore.

Il seguente *box plot* riporta invece le caratteristiche di distribuzione della differenza del numero di task schedulati.

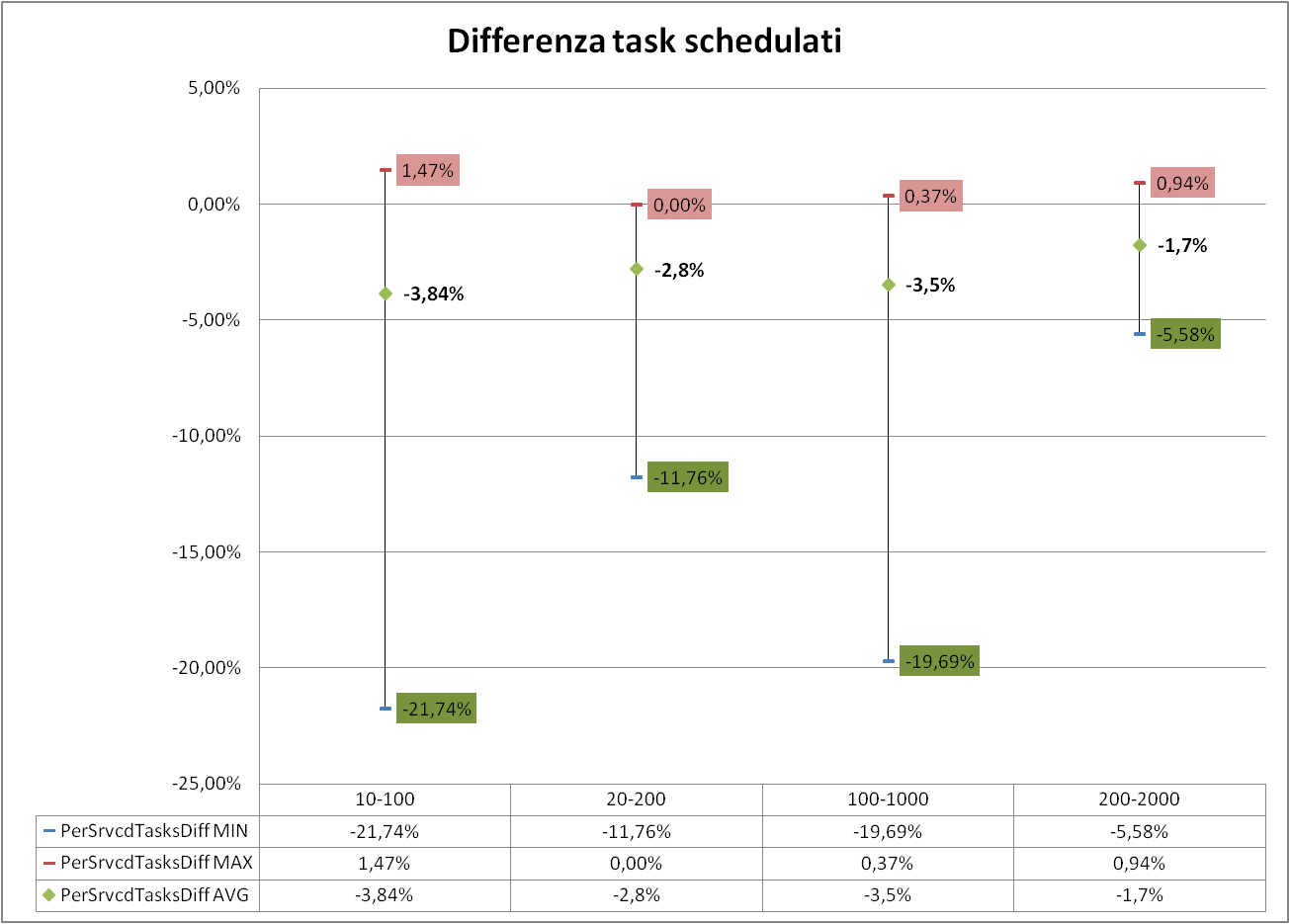


Figura : *box plot* della variazione dei task schedulati al variare della complessità

Si nota la presenza di *outlier* negativi corrispondenti a variazioni anche significative (in peggio) del numero di task schedulati; i valori medi si attestano però su valori significativamente migliori (tipicamente con un rapporto di 1:5 o anche di più).

E’ interessante rilevare anche la presenza di *outlier* positivi, che si registrano cioè in corrispondenza di dataset che hanno avuto un maggior numero di task schedulati nel dataset potato piuttosto che in quello completo. Nuovamente, un’analisi esaustiva del fenomeno richiederebbe un’attenta valutazione delle caratteristiche dei dataset, ma è lecito supporre che in questi casi il classificatore segnali come schedulabili (e quindi come istanze da mantenere nel dataset potato) dei task che risultano avere delle caratteristiche “migliori” (e.g. tempo minore di esecuzione) rispetto a quelli che sarebbero stati scelti originariamente.

### Dati completi

Le seguenti tabelle e grafici riportano in modo sintetico i risultati sperimentali registrati ed in particolare le differenze di tempo di esecuzione e di task schedulati per ciascun gruppo.

#### Gruppo 10-100



Tabella : riassunto dei dati di valutazione per il gruppo 10-100

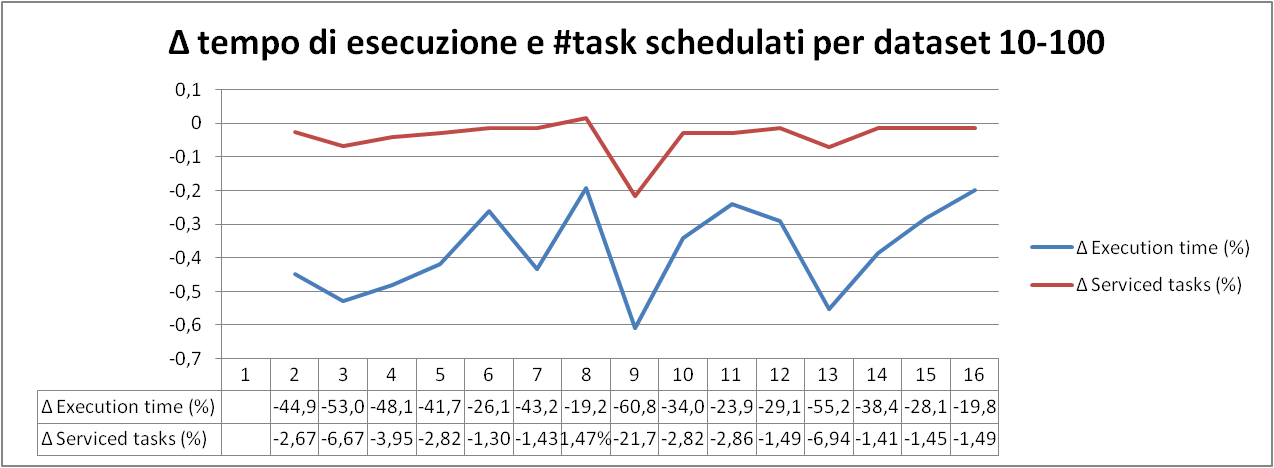


Figura : andamento della differenza del tempo di esecuzione e #task schedulati per il gruppo 10-100

#### Gruppo 20-200



Tabella : riassunto dei dati di valutazione per il gruppo 20-200

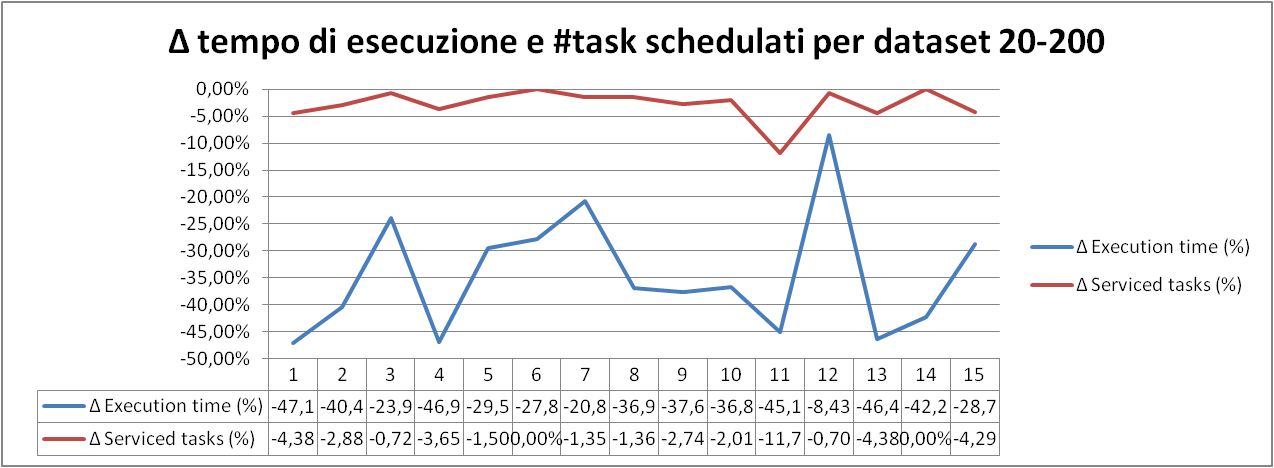


Figura : andamento della differenza del tempo di esecuzione e #task schedulati per il gruppo 20-200

#### Gruppo 100-1000



Tabella : riassunto dei dati di valutazione per il gruppo 100-1000

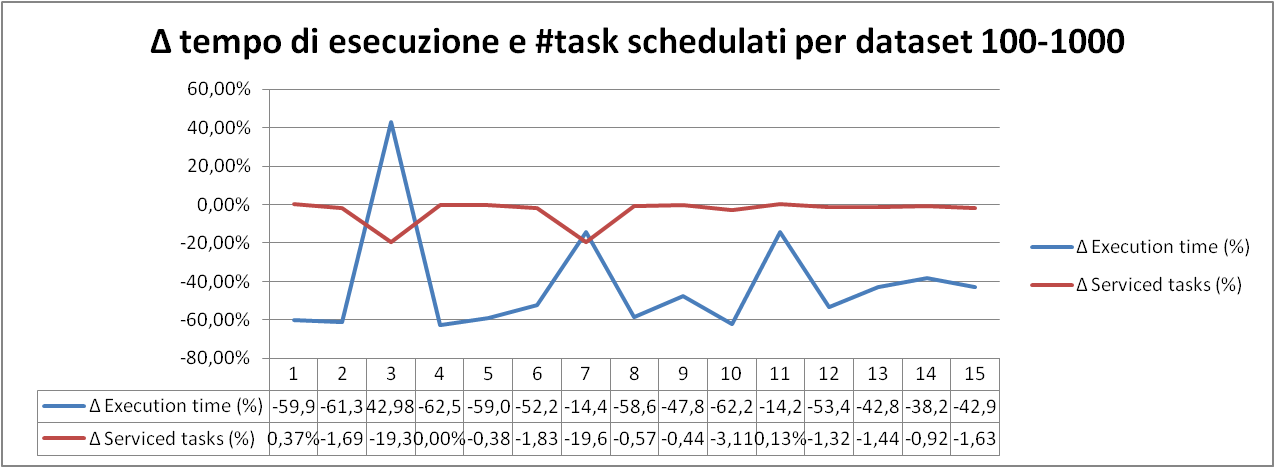


Figura : andamento della differenza del tempo di esecuzione e #task schedulati per il gruppo 100-1000

#### Gruppo 200-2000



Tabella : riassunto dei dati di valutazione per il gruppo 200-2000

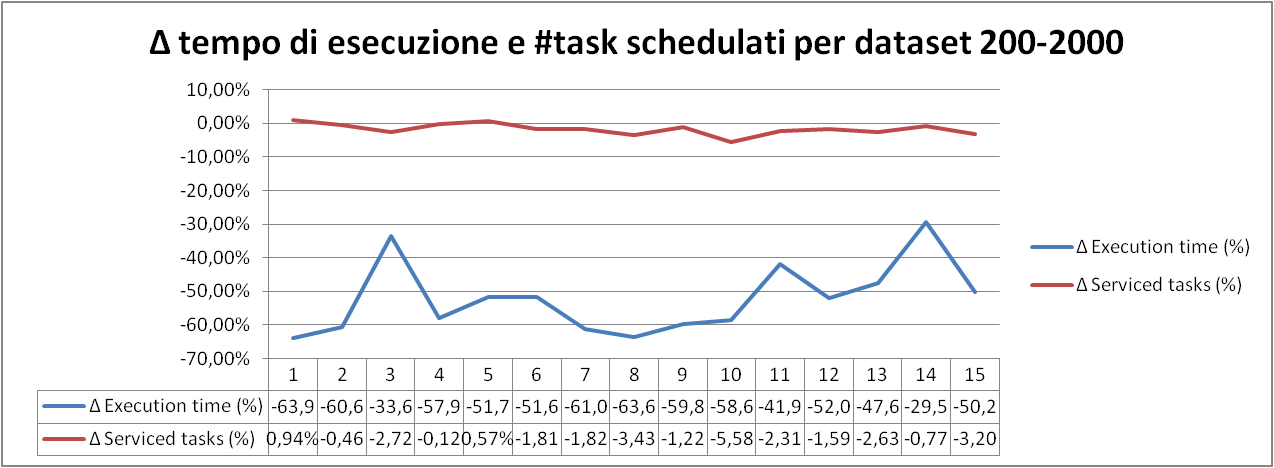


Figura : andamento della differenza del tempo di esecuzione e #task schedulati per il gruppo 200-2000

## Validazione “in avanti” del classificatore

In questa seconda fase analizzeremo i risultati sperimentali legati all’utilizzo di un classificatore addestrato su scenari di complessità “x” e applicato per la potatura di ***dataset di complessità maggiore***. Utilizzeremo quindi training set di complessità *minore* rispetto ai dataset da risolvere.

Come già descritto nel capitolo 6.4, in questa fase è stato scelto un gruppo di dataset (in questo caso il 100-1000), i cui set di istanze sono stati utilizzati tutti per l’addestramento del classificatore. Una volta addestrato il classificatore, sono stati costruiti dei dataset potati a partire dai dataset del gruppo 200-2000 in modo del tutto analogo a quanto fatto nella fase precedente.

Per semplicità, sia per il training set che per il test set sono stati usati solo dataset con densità d=0,1, limitando quindi il numero di dataset da lavorare ed il tempo complessivo di elaborazione.

### Validazione del classificatore

Come primo passo, vogliamo verificare i risultati della valutazione del classificatore sui test set del gruppo 200-2000. Anche in questo caso, come già fatto in precedenza, useremo le funzioni del modulo eval delle API di Weka per ottenere informazioni su precision, recall, % e numero assoluto di istanze correttamente e incorrettamente classificate.

Di seguito si riportano quindi i risultati sperimentali di questa prima fase di valutazione:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Dataset name** | **Precision** | **Recall** | **#Correctly Classified** | **%Correctly Classified** | **#Incorrectly Classified** | **%Incorrectly Classified** |
| DS\_2000\_200\_batch0\_stats.arff | 0,775 | 0,785 | 1570 | 78,50% | 430 | 21,50% |
| DS\_2000\_200\_batch1\_stats.arff | 0,771 | 0,785 | 1569 | 78,45% | 431 | 21,55% |
| DS\_2000\_200\_batch2\_stats.arff | 0,784 | 0,791 | 1581 | 79,05% | 419 | 20,95% |
| DS\_2000\_200\_batch3\_stats.arff | 0,756 | 0,765 | 1530 | 76,50% | 470 | 23,50% |
| DS\_2000\_200\_batch4\_stats.arff | 0,765 | 0,778 | 1555 | 77,75% | 445 | 22,25% |

Tabella : risultati sperimentali della validazione "in avanti" del classificatore

Si tratta di valori buoni che consentono di procedere con il passo successivo (i.e. creazione di dataset potati e valutazione del classificatore) ma inferiori rispetto a quanto riscontrato nella valutazione del classificatore su test set della stessa complessità del training set (v. capitolo 8.3 per maggiori dettagli).

Interessante notare come i risultati risultino inferiori anche rispetto ai dati riportati nel capitolo 8.3 per il gruppo 100-1000; potrebbe essere interessante valutare il rischio che il classificatore in queste condizioni sia esposto al fenomeno dell’*overfitting*, perdendo quindi in generalità di rappresentazione.

### Valutazione dei risultati

Dati i risultati comunque in assoluto buoni riportati durante la fase di valutazione preliminare (v. paragrafo precedente), possiamo quindi procedere alla generazione di dataset potati e loro risoluzione. Come già descritto nel capitolo 6.4.2, i passi che sono stati eseguiti sono:

1. generazione del dataset potato e scrittura su disco;
2. esecuzione del risolutore sul dataset completo e memorizzazione dei risultati;
3. esecuzione del risolutore sul dataset potato e memorizzazione dei risultati.

Dal momento che, come già detto, per questa fase sono stati presi in considerazione solamente dataset con densità d=0,1, avremo a disposizione 5 istanze di confronto di esecuzione corrispondenti ad altrettanti dataset del gruppo 200-2000 che sono stati analizzati. Il seguente grafico riassume in modo sintetico le differenze di tempo di esecuzione e numero di task (in percentuale in entrambi i casi) tra dataset potato e completo.

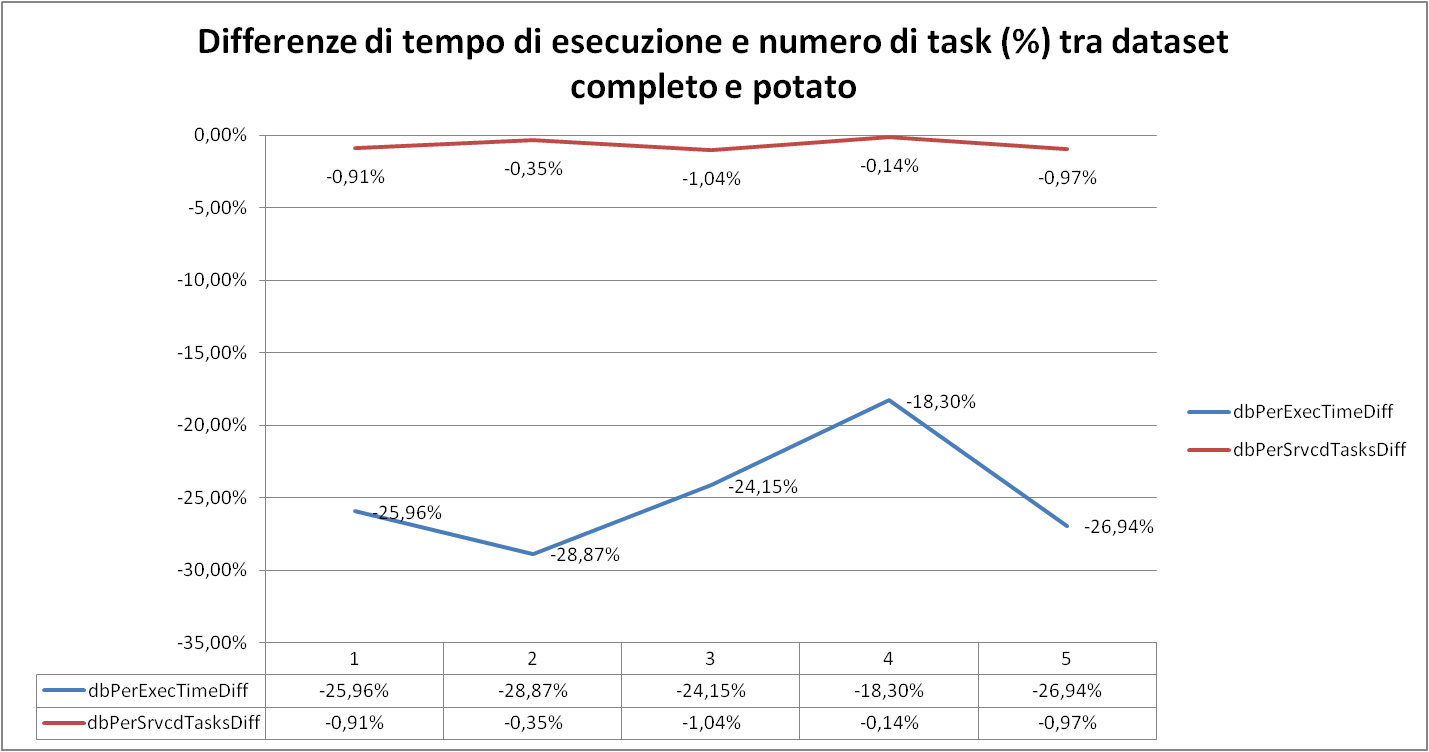


Figura : differenze di tempo di esecuzione e task schedulati per la validazione in avanti

I risultati sono incoraggianti e possono essere riassunti dalla seguente tabella:

|  |  |  |
| --- | --- | --- |
|  | **dbPerExecTimeDiff** | **dbPerSrvcdTasksDiff** |
| Min | -28,87% | -1,04% |
| Max | -18,30% | -0,14% |
| Media | -24,84% | -0,68% |
| StdDev | 4,03% | 0,41% |

Tabella : tabella riassuntiva dei risultati della validazione "in avanti"

Si rilevano quindi dei discreti miglioramenti di performance di esecuzione che permettono di ridurre di circa ¼ il tempo di risoluzione del problema. Si nota però che questi valori sono sostanzialmente inferiori rispetto a quanto visto in precedenza nel paragrafo 8.4.1 dove per il gruppo 200-2000 si riusciva ad ottenere un miglioramento medio del tempo di esecuzione superiore al 50%.

Le variazioni del numero di task schedulati sono essenzialmente contenute e possono ritenersi accettabili.

### Dati completi

La seguente tabella riporta l’insieme dei dati sperimentali rilevati per la valutazione “in avanti” per il gruppo 200-2000.



Tabella : risultati sperimentali per la validazione "in avanti"

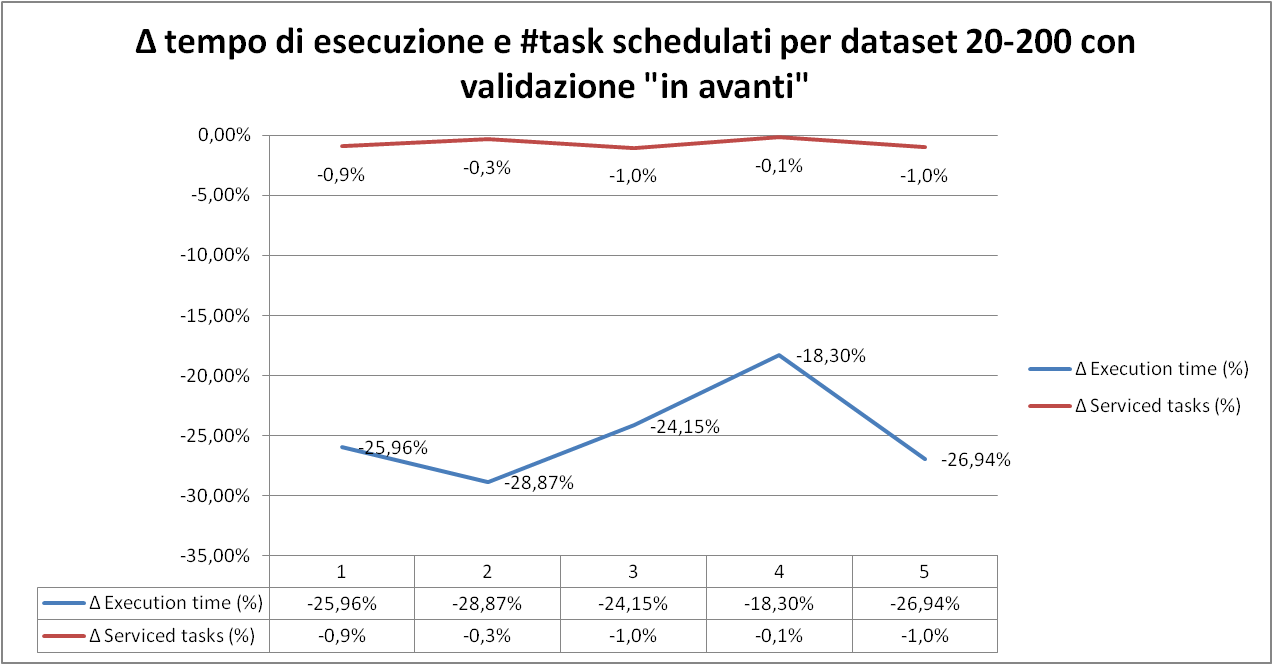


Figura : risultati sperimentali per la validazione "in avanti" su gruppo 200-2000

# Indice delle figure

[Figura 1: rappresentazione grafica di un dataset 8](#_Toc487487730)

[Figura 2: esempio di dataset (risolto) con posizioni di partenza dei task generate intorno al centro del piano 13](#_Toc487487733)

[Figura 3: esempio di dataset (risolto) con posizioni di partenza dei task generate intorno alla parte "nord" del piano 13](#_Toc487487734)

[Figura 4: esempio di dataset (risolto) con posizioni di partenza dei task generate intorno alla parte "est" del piano 14](#_Toc487487735)

[Figura 5: esempio di dataset (risolto) con posizioni di partenza dei task generate intorno alla parte "est" del piano 14](#_Toc487487736)

[Figura 6: tempi di risoluzione per tipi diversi di dataset 16](#_Toc487487738)

[Figura 7: classi utilizzate per la rappresentazione delle entità principali 18](#_Toc487487739)

[Figura 8: flow chart di esecuzione del modulo di generazione e risoluzione dataset 20](#_Toc487487740)

[Figura 9: diagramma UML della classe BatchLauncher e sue dipendenze 21](#_Toc487487741)

[Figura 10: flusso di esecuzione del modulo di generazione e valutazione del classificatore 24](file:///C:\Users\gperr\workspace\Data_Mining\docs\DM_Tesina%20Perrone_rv3_170708.docx#_Toc487487743)

[Figura 11: diagramma UML della classe BatchClassifier e relative dipendenze 26](#_Toc487487744)

[Figura 12: flow chart di esecuzione del modulo di valutazione del classificatore 29](file:///C:\Users\gperr\workspace\Data_Mining\docs\DM_Tesina%20Perrone_rv3_170708.docx#_Toc487487746)

[Figura 13: diagramma UML del modulo di valutazione "in avanti" 30](#_Toc487487747)

[Figura 14: precisione del classificatore al variare della complessità (box plot) 35](#_Toc487487752)

[Figura 15: deviazioni standard delle performance del classificatore al variare della complessità 36](#_Toc487487753)

[Figura 16: differenze di tempo di esecuzione e numero task schedulati al variare della complessità 37](#_Toc487487754)

[Figura 17: *box plot* della variazione del tempo di esecuzione al variare della complessità 38](#_Toc487487755)

[Figura 18: *box plot* della variazione dei task schedulati al variare della complessità 39](#_Toc487487756)

[Figura 19: andamento della differenza del tempo di esecuzione e #task schedulati per il gruppo 10-100 40](#_Toc487487758)

[Figura 20: andamento della differenza del tempo di esecuzione e #task schedulati per il gruppo 20-200 41](#_Toc487487760)

[Figura 21: andamento della differenza del tempo di esecuzione e #task schedulati per il gruppo 100-1000 41](#_Toc487487762)

[Figura 22: andamento della differenza del tempo di esecuzione e #task schedulati per il gruppo 200-2000 42](#_Toc487487764)

[Figura 23: differenze di tempo di esecuzione e task schedulati per la valutazione in avanti 44](#_Toc487487766)

[Figura 24: risultati sperimentali per la valutazione "in avanti" su gruppo 200-2000 45](#_Toc487487769)

[Tabella 1:rappresentazione dei task 7](#_Toc487487728)

[Tabella 2: rappresentazione delle risorse operative 7](#_Toc487487729)

[Tabella 3: parametri di configurazione generici per la creazione del dataset 11](#_Toc487487731)

[Tabella 4: parametri di configurazione per la distribuzione spaziale e temporale delle risorse 12](#_Toc487487732)

[Tabella 5: parametri di configurazione per la distribuzione temporale dei task 15](#_Toc487487737)

[Tabella 6: risultati memorizzati per ciascuna soluzione trovata nella classe SolStats 22](#_Toc487487742)

[Tabella 7: parametri memorizzati nella classe ClassifierStats 27](#_Toc487487745)

[Tabella 8: data members della classe EvaluationStats usati per la memorizzazione dei risultati di valutazione 31](#_Toc487487748)

[Tabella 9:gruppi di dataset utilizzati per la sperimentazione 33](#_Toc487487749)

[Tabella 10: parametri di configurazione utilizzati per la generazione dei dataset 34](#_Toc487487750)

[Tabella 11: precisione del classificatore al variare della complessità 35](#_Toc487487751)

[Tabella 12: riassunto dei dati di valutazione per il gruppo 10-100 40](#_Toc487487757)

[Tabella 13: riassunto dei dati di valutazione per il gruppo 20-200 40](#_Toc487487759)

[Tabella 14: riassunto dei dati di valutazione per il gruppo 100-1000 41](#_Toc487487761)

[Tabella 15: riassunto dei dati di valutazione per il gruppo 200-2000 42](#_Toc487487763)

[Tabella 16: risultati sperimentali della valutazione "in avanti" del classificatore 43](#_Toc487487765)

[Tabella 17: tabella riassuntiva dei risultati della valutazione "in avanti" 44](#_Toc487487767)

[Tabella 18: risultati sperimentali per la valutazione "in avanti" 45](#_Toc487487768)

# 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)
5. https://github.com/graphhopper/jsprit/blob/master/docs/Meta-Heuristic.md [↑](#endnote-ref-2)
6. “C4.5: Programs for Machine Learning by J. Ross Quinlan. Morgan Kaufmann Publishers, Inc., 1993” – Machine Learning, September 1994, vol. 16, issue 3, pp 235-240 [↑](#endnote-ref-3)
7. I punti decimali della densità non sono stati riportati nei nomi dei files dei dataset per evitare problemi con la memorizzazione dei file stessi [↑](#footnote-ref-4)
8. Si noti che, a causa di limitazioni del tool di visualizzazione, è possibile graficare nel box plot solamente i valori minimo, massimo e medio per ciascun attributo [↑](#footnote-ref-5)
9. Dato il numero ridotto di campioni, identificheremo con outlier i campioni che sono significativamente lontani dal valor medio [↑](#footnote-ref-6)