Alma Mater Studiorum · Università di Bologna

Dipartimento di Scienze

Corso di Laurea in

Informatica

RANDOMIZED CONFIGURATION FOR ALGORITHM SELECTOR SUNNY

Tesi di Laurea in
Linguaggi di programmazione

Relatore:
Chiar.mo Prof.
MAURIZIO GABBRIELLI

Presentata da:
LUCA BRUGALETTA

Correlatore:
Chiar.mo Dott.
TONG LIU

III Sessione
Anno Accademico 2017/2018

Indice

Introduzione

L'obiettivo di questa tesi è quello di discutere la selezione delle features per SUNNY e confrontare i risultati che si possono avere con l'utilizzo di una tecnica di intelligenza artificiale (Simulated Annealing) rispetto ad una selezione randomica per la scelta delle nostre features.

L'algoritmo Sunny può essere generalizzato nel problema della selezione dell'algoritmo,per cui , approfondiremo prima tale argomento e solo in seguito parleremo dei metodi che abbiamo implementato e delle tecniche utilizzate per effettuare la scelta delle features.

Preso un gruppo di algoritmi, il problema delle selezione degli algoritmi(AS) consiste nell'identificare quale di questi è il migliore per la risoluzione del problema corrente, la selezione dipende da un insieme di attributi (feature) che caratterizzano il problema da risolvere, di seguito vedremo l'impatto della scelta di queste feature sull'algoritmo Sunny, prendendo come riferimento la libreria AS (ASlib). Il problema della scelta dell'algoritmo inizialmente fu proposto da Rice nel 1976, negli ultimi decenni ha attirato attenzioni sempre crescenti e sono stati forniti diversi approcci per esso. AS viene utilizzato per selezionare il miglior algoritmo atto alla risoluzione di un problema, in modo più specifico, dato un insieme (portfolio) di algoritmi $\{A_1, ..., A_m\}$ il problema alla base è quello di predire il miglior algoritmo A_k per risolvere un problema non ancora analizzato, questa scelta dipende chiaramente da diversi fattori e da varie misurazioni, principalmente dalle performance (es. il tempo di esecuzione).Per determinare il valore di tali fattori si utilizza un vettore di attributi (feature), ma sfortunatamente non tutte le feature del vettore hanno la stessa importanza, infatti, molto spesso accade che alcune di questi attributi forniscono più informazioni rispetto agli altri; è quindi

importante fare una scelta, di quante e quali feature mantenere. Questo è il punto cruciale per le performance dell'algoritmo di selezione (AS), tale processo è noto in machine learning come Feature Selection (FS).

L'approccio di partenza di SUNNY è quello sequenziale, si pone come obiettivo quello di scegliere cinque feature e un valore k per essi. A livello pratico, il primo passo di questa tesi è stato quello di modificare questo primo approccio introducendone uno che sceglieva il numero di feature, la configurazione formata dalle feature e dal valore di k, tutto in modo casuale, abbiamo chiamato questa versione «randk»; una volta ottenuti i risultati da questo metodo li abbiamo poi utilizzati per fare un confronto con la versione che sfrutta Simulated annealing, tecnica implementata a partire dalla versione casuale.

Capitolo 1

Preliminari

L'algoritmo SUNNY è un approccio di Algorithm Selection originariamente creato per la programmazione a vincoli* (CP), CP è un paradigma dichiarativo dove le relazioni fra variabili vengono definite attraverso dei vincoli, tali vincoli non specificano azioni da eseguire ma proprietà che devono essere soddisfatte,tra gli obiettivi principali di CP vi è quello di modellare e risolvere problemi di soddisfazione di vincoli (CSP). Il trend è utilizzare il portfolio per aumentare l'efficienza delle soluzioni, perché' molto spesso un solver non riesce a garantire la prestazione migliore su tutti i problemi

Un risolutore CP che utilizza portfolio, può essere visto come un risolutore particolare, soprannominato risolutore portfolio, preso un insieme $m > 1$ di diversi risolutori s_1,...,$s_m$ne individua uno che sia globalmente migliore degli altri. Quando si presenta una nuova istanza i non ancora presa in esame, il risolutore portfolio prova a predire quali potrebbero essere i migliori risolutori s_1,..,s_k (con $1 \leq$ k $\leq$ m) per i, ed esegue ogni risolutore (s) sull'istanza i. La scelta su quale risolutore preferire è chiaramente uno dei punti chiave del successo di questo approccio e spesso viene affrontato con tecniche di *machine learning* (ML).

Sunny è un algoritmo portfolio lazy che individua le similitudini fra le istanze per individuare il miglior risolutore o i migliori da utilizzare, il nome deriva dall'acronimo di:

- *SUb-portfolio*: per un'istanza, selezioniamo il sotto-insieme portfolio più adatto per l'esecuzione.

*Constraint programming

- *Nearest Neighbor*: in poche parole, il vicino più simile, utilizziamo un algoritmo k-NN per determinare il sub-portfolio, infatti, con questo algoritmo, dalle istanze viste in precedenza, estraiamo le k istanze che sono più simili a quella che deve essere risolta.

- *laz*Y: perché non viene definito nessun tipo di modello di predizione in anticipo.

1.1 K-NN

K-nearest neighbors è un algoritmo di classificazione non parametrico, è uno tra gli algoritmi più semplici fra tutti gli algoritmi di machine learning. L'input comprende le k istanze più simili dell'insieme di addestramento e punta ad identificare a quale classe appartiene l'output. L'output è un elemento della classe, scelto tra le k istanze simili, dove kè un intero, tipicamente con un valore piccolo.

k-NN utilizza un metodo di apprendimento lazy, dove l'apprendimento consiste in una generalizzazione dei dati di addestramento che vengono poi computati solamente durante la fase di classificazione. La scelta di un buon valore per k dipende dai dati, valori più grandi per k riducono l'effetto del rumore per la classificazione, ma rendono la classificazione computazionalmente più costosa e possono ridurre le differenze tra le classi meno distinte, mentre valori di k più bassi sono più sensibili al rumore. La precisione dell'algoritmo k-NN, può essere infatti degradata dalla presenza di rumore o di features irrilevanti. Per la realizzazione di un algoritmo di classificazione k-NN più preciso, è quindi utile assegnare un peso alle istanze simili, in modo che quelle più vicine possano offrire un contributo maggiore di quelle distanti, un metodo comune di farlo, è attraverso la distanza Euclidea. In figura 1.1 possiamo osservare un esempio di classificazione k-NN

SUNNY è un algoritmo portfolio lazy, individua le similarità tra le istanze per trovare il miglior risolutore s. Per una istanza i, SUNNY utilizza un algoritmo k-Nearest Neighbors (k-NN), ma a differenza di k-NN che sceglie un algoritmo per un'istanza, Sunny sceglie un sottoinsieme N(i, k) delle k istanze simili ad i. In seguito, crea una sequenza di esecuzione considerando il più piccolo sub-portfolio in grado di risolvere il maggior numero di istanze nello spazio del sotto insieme N(i, k). In questa sequenza SUNNY partiziona

il tempo di esecuzione per ogni risolutore del sub-portfolio in proporzione al numero di istanze che risolve in N(i, k).

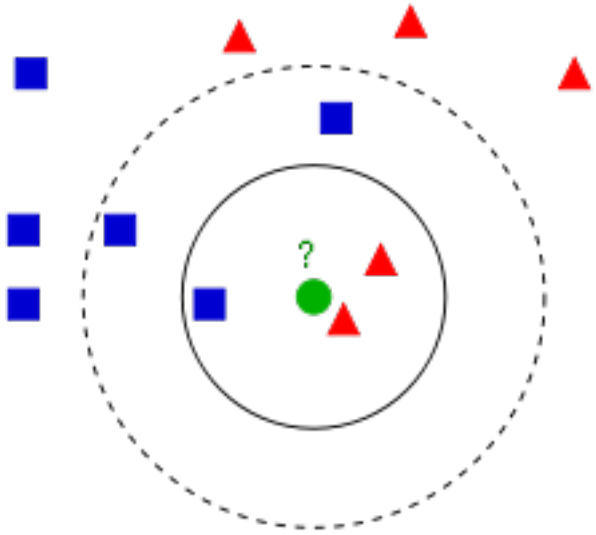

Figura 1.1: Esempio di classificazione k-NN: La sequenza di prova (cerchio verde) potrebbe essere classificata nella prima classe, con i quadrati blu, o nella seconda dei triangoli rossi. Se k=3 (cerchio con la linea continua) viene assegnato alla seconda classe perchè ci sono 2 triangoli e solo 1 quadrato. Se k=5 (cerchio tratteggiato) allora viene assegnato alla prima classe perchè ci sono 3 quadrati contro i 2 triangoli

1.2 Algorithm Selection

AS è un argomento che attira sempre più interesse tra ricercatori e professionisti, dopo anni di applicazioni si sono venuti a creare molti dati sull'argomento, ma non si è mai creato uno standard, questo ovviamente rende difficile la condivisione e il confronto dei diversi approcci. Inoltre, crea diversi ostacoli per chi è interessato a a lavorare o ad effettuare ricerche in questo campo. Per risolvere questo problema, è stato introdotto un formato standard per la rappresentazione di scenari AS e un sistema informativo che contiene un numero sempre crescente di dati ASlib. Per comprendere i vantaggi di ASlib, cominciamo parlando dello studio di un modello di selezione di algoritmi (AS) con un'interfaccia comune.

Rice è stato il primo a formalizzare l'idea di scegliere fra diversi algoritmi per ogni istanza, introducendo così la selezione di algoritmi. Il problema della selezione di algoritmi per un'istanza i mira a scegliere il miglior algoritmo s(i) da eseguire su tale istanza i.

Questa scelta viene spesso fatta utilizzando le feature di istanza. Le configurazioni sono poi mappate ad un certo algoritmo utilizzando tecniche di intelligenza artificiale. Ci sono molti modi per affrontare la selezione di algoritmi per istanza, queste strategie possono includere un singolo modello che apprende o la combinazione di più di essi, di modo che, presa l'istanza di un nuovo problema, questi metodi vengano poi utilizzati per decidere quale o quali di questi algoritmi selezionare.

La via più naturale sarebbe scegliere un solo algoritmo per risolvere una determinata istanza, ma lo svantaggio maggiore di questo approccio è che non vi è modo di tornare indietro nel caso in cui l'algoritmo selezionato non offre buone prestazioni.

in alternativa, possiamo cercare una sequenza con un ordine ben definito e un quanto di tempo nel quale possiamo eseguire tutti o un sottoinsieme degli algoritmi del nostro portfolio. In alcuni di questi approcci, il calcolo della sequenza viene trattata come un problema di ottimizzazione con l'obiettivo di massimizzare(minimizzare) uno o più parametri voluti, ne è un caso l'approccio che abbiamo utilizzato, simulated annealing, una tecnica di ottimizzazione, con l'obiettivo della ricerca di un punto di minimo (massimo) globale[†], particolaremnte in situazioni nella quale abbiamo la presenza di più minimi (massimi) locali[‡] .

1.2.1 ASlib

Per valutare SUNNY su diversi scenari si utilizza la libreria Algorithm Selection (ASlib). ASlib è stato proposto come formato dati per gli scenari degli algoritmi di selezione, questo formato e tutti i dati che ne risultano ci permettono un confronto equo e conveniente per la selezione degli algoritmi.

[†]Sia $y{=}f(x)$ una funzione con dominio $Dom(f)$. Diciamo che x_0è un punto di minimo (massimo) globale per la funzione, e che $f(x_0)$ è il minimo (massimo) globale della funzione se per ogni $x \epsilon\ Dom(f)$ risulta che $f(x){\geq}f(x_0)$.

[‡]Ci troviamo in presenza di un minimo (massimo) locale, se data $y = f(x)$ con dominio$Dom(f)$. x_0 è un punto di minimo(massimo) locale se preso un intorno di dimensione δ, con al centro x_0,per ogni x appartenente all'intorno δ, si ha che $f(x) \geq f(x_0)$.

Le specifiche del formato assumono un approccio generico alla selezione degli algoritmi e sono definite come segue:

1. Viene calcolato un vettore di features $f(i) \in \mathcal{F}$
2. Una tecnica di machine learning s seleziona un algoritmo $a \in \mathcal{A}$ in base ai risultati del vettore di features del punto 1.
3. L'algoritmo a viene applicato all'istanza i.
4. Viene calcolato un indice m che da una misurazione delle performance, tenendo conto del costo delle features e le prestazioni dell'algoritmo scelto.
5. Alcuni AS non scelgono un solo algoritmo, ma producono una sequenza di diversi algoritmi. Applicano a ad i utilizzando un certo budget $r \in \mathcal{R}$, valutano le prestazioni, criteri di arresto e ripetono finché necessario, tenendo conto dei progressi fatti durante l'esecuzione di a

Lo scopo è quello di garantire la disponibilità ed un unico formato per tutte le informazioni necessarie, rendendo così più facile la comparazione di diversi approcci AS con l'utilizzo degli stessi scenari, tramite diversi parametri di confronto e di strumenti. Chi usa ASlib, inoltre, non deve preoccuparsi di eseguire realmente l'algoritmo sulle istanze, in quanto questo è gia stato fatto, essendo una sorta di simulazione dove i runtime sono già stati pre-computati, in questo modo si riducono drasticamente i costi di computazione per la riproduzione dei dati.

1.3 Feature Selection

Come abbiamo già detto il problema della scelta dell'algoritmo (AS) mira a scegliere il miglior algoritmo per risolvere un problema, avendone diversi a disposizione. Per fare questo utilizziamo un vettore di features, tali features hanno utilità diverse e alcune risultano più attinenti e informative di altre, scegliere quali di queste features risultano utili è un problema importante per le performance di AS, questo processo è chiamato scelta degli attributi (FS).

In questa sezione parleremo dell'impatto sulle prestazioni delle tecniche di FS, con l'utilizzo di ASlib come standard per confrontare i risultati. Lo scopo è mostrare come SUNNY può trarre vantaggio dall'utilizzo di diversi

insiemi di features e in particolare dagli studi effettuati si riesce a dimostrare che con l'utilizzo di tecniche di FS, anche un piccolo numero di features può essere sufficiente per ottenere risultati simili all'approccio iniziale che sfrutta invece tutte le features disponibili.

Le features sono molto importanti per caratterizzare una determinata istanza di un problema e costruire modelli di previsione accurati, la scelta degli attributi (FS) ha lo scopo di rimuovere quelle features ridondanti, irrilevanti, e potenzialmente dannose. Scegliere un insieme di features quanto più piccolo, ha il vantaggio di semplificare il modello di predizione, riducendo il tempo di training e il costo di estrazione delle features, riducendo i casi di overfitting[§] e migliorando l'accuratezza delle previsioni.

Le tecniche di FS non sono altro che la combinazione di due ingredienti:

- Una tecnica di ricerca per trovare sottoinsiemi di attributi validi
- Una metrica di punteggio per valutare ogni sottoinsieme

Dato che valutare tutti i possibili sottoinsieme è computazionalmente impossibile, vengono utilizzate tecniche euristiche per la ricerca dei sottoinsiemi. In particolare, parliamo di tecniche greedy[¶], sfruttando la selezione in avanti e l'eliminazione all'indietro. Nella selezione in avanti, le features vengono integrate in un sottoinsieme sempre più grande. Nell'eliminazione all'indietro, invece, le combinazioni vengono progressivamente rimosse partendo da tutte le features disponibili.

1.3.1 Metodo di valutazione

: considerando un portfolio di solver come un solver unico (che sostituisce il solver solo), il feature cost ha comunque un impatto negativo(o significativo) sul questo portfolio di solver perché per fare il portfolio abbiamo bisogno di sapere i valori del features, ma per sapere questi features, in pratica abbiamo bisogno di computarlo e il tempo di computazione è denominato come feature cost. E questo costo deve essere considerato come tempo consumato da un portfolio di solver.

[§]Caso in cui la capacità di prevedere o adattarsi aumentano sui dati di allenamento e dimiuiscono su quelli non ancora visionati

[¶]Tecnica «golosa», perché ad ogni istante sceglie sempre il candidato più promettente

Lo scopo di Algorithm Selection, come già detto, è sicuramente quello di ottenere prestazioni migliori rispetto a quanto possa fare un singolo algoritmo. Considerando però un portfolio di risolutori come un unico risolutore, il costo per il calcolo delle features, diventa significativo, parliamo di questo costo come overhead, considerato come il tempo consumato dal portfolio di risolutori e deve essere preso in considerazione durante la valutazione di sistema di selezione dell'algoritmo.

Per essere in grado di valutare il guadagno in termini di prestazioni di un sistema che fa uso di AS, si utilizzano due linee guida per fare il confronto:

- Le migliori prestazioni ottenute da un singolo algoritmo su tutte le istanze del training, chiamato SBS[‖], ci permette di capire quali sono i risultati che si possono utilizzare senza fare uso di AS
- Le prestazioni del miglior risolutore virtuale, VBS[**], anche chiamate prestazioni oracolo. VBS effettua sempre la scelta giusta senza produrre overhead sulle istanze, corrisponde ad un portfolio senza overhead che esegue tutti gli algoritmi in parallelo e interrompe la sua esecuzione non appena il primo algoritmo termina.

Le performance delle linee guida su ogni sistema di AS variano in base allo scenario. Si normalizzano quindi le performance $m_s = \sum_{i \in \mathcal{I}} m(s(i), i)$ del risolutore s su uno scenario in base ad SBS e VBS, dove le differenze tra i due viene evidenziata come segue:

$$\hat{m}_s = \frac{m_{SBS} - m_S}{m_{SBS} - m_{VBS}}$$

$\hat{m}$ è stato definito come 0, che corrisponde alle prestazioni di SBS, 1 corrisponde invece alle prestazioni ottimali e quindi a VBS, le prestazioni di un sistema AS saranno sempre tra 0 e 1, e se il valore scende sotto lo 0, allora significa che scegliere SBS è una strategia migliore.

Si possono utilizzare diverse metriche per la valutazione degli approcci AS su un certo scenario. FSI[††] è la proporzione tra il numero di istanze che l'approccio risolve e il numero di istanze dello scenario, viene comunemente utilizzata per confrontare i diversi approcci di AS data la sua semplicità e

[‖]Single best solver
[**]Virtual best solver
[††]Fraction of Solved Istances

le informazioni che dà, tuttavia, non tiene conto del tempo impiegato per la risoluzione del problema, per ovviare a questo problema si utilizza spesso PAR[‡‡]. Fissato uno scenario e preso un intero positivo k, PAR_k(S) rappresenta il tempo medio che impiega un approccio S per risolvere tutto i problemi dello scenario, la penalizzazione di k x τ è data dalle istanze non risolte da S entro il timeout τ.

1.4 Sunny nel dettaglio

L'algoritmo SUNNY prende in input vari parametri: l'istanza del problema che dobbiamo risolvere, il portfolio dei risolutori, una copia speciale del risolutore[§§], un parametro k (≥ 1) che rappresenta la dimensione del vicino da prendere in considerazione, un timeout τ che sarà il massimo tempo a disposizione per l'esecuzione, un portfolio $\mathcal{A}$ di algoritmi. Il nostro algoritmo individua le similitudini fra le varie istanze e produce un sequenza di esecuzione $\sigma = [(\mathrm{A}_1, \mathrm{t}_1), \ldots, (\mathrm{A}_\mathrm{h}, \mathrm{t}_\mathrm{h})]$, l'algoritmo $\mathrm{A_i} \in \mathcal{A}$ viene eseguito per $\mathrm{t_i}$secondi e la $\sum_{i=1}^{h} t_i = \tau$.

Per ogni problema x in input , SUNNY utilizza l'algoritmo k-NN per estrarre tra un insieme di istanze simili, il sottoinsieme N(x, k) di k istanze più adatte per il problema x. In generale, il vettore delle feature di x è un insieme di $F(\mathrm{x}) \in \mathcal{R}_\mathrm{d}$ di attributi numerici che caratterizzano x (es. vincoli di x o statistiche sulle variabili).

Partendo dal sottoinsieme N(x, k), SUNNY si basa su tre euristiche utili per creare la sequenza σ:

- $\mathrm{H_{sel}}$: seleziona il più piccolo sotto insieme portfolio S $\subseteq A$ che risolve il maggior numero di sotto istanze in N(x, k), utilizzando il tempo di esecuzione come parametro di confronto.

- $\mathrm{H_{all}}$: assegna ad ogni $\mathrm{A}_i \in \mathcal{A}$ il tempo di esecuzione $\mathrm{t}_i \in [0, \tau]$ for i = 1, . . . , h in base al numero di istanze che è riuscito e risolvere, e utilizza la copia del risolutore, per risolve tutte le istanze che non erano state risolte.

- $\mathrm{H_{sch}}$: riordina gli algoritmi in base al loro tempo di esecuzione per N(x,k) e ritorna la sequenza σ

[‡‡]Penalized Average Runtime: metrica che utilizza il tempo totale di esecuzione e penalizza le esecuzioni che non riescono a raggiungere la soluzione cercata

[§§]Risolutore speciale del portfolio mirato alla gestione di casi particolari

Capitolo 2

Sunny-as & OASC

Per sperimentare le due tecniche volte a creare i risultati utili per il confronto, tra l'approccio random e simulated annealing, siamo partiti dall'utilizzo di Sunny-as, in particolare la versione utilizzata per la OASC challenge*. Sunny-as è stato utile per svolgere la fase di training e di test sugli scenari AS in accordo con gli standard di ASlib, permettendoci di poter sperimentare sui metodi per la scelta degli attributi.

Sunny-as prevede tre fasi diverse: una di training, una pre-risolutiva e una finale di testing. Nella fase di training sunny-as estrae alcune informazioni base dallo scenario (es. portfolio, il risolutore speciale, il timeout) e costruisce tutto quello che poi viene utilizzato da SUNNY per produrre la sequenza di esecuzione, come già spiegato. in generale, questa fase di training è quella dove si fanno tutte le associazioni tra le istanze, il vettore delle features e il tempo di esecuzione di ogni risolutore sull'istanza i. Questa fase può essere vista più come una fase di inizializzazione più che come una di addestramento, dal momento che non vi è nessun modello di apprendimento.

Nella seconda fase, la pre-risolutiva, è possibile filtrare il portfolio iniziale o effettuare scelte tra le features iniziali. Infine, la fase di testing è quella in cui si esegue e si valutano le predizioni di SUNNY.

Sunny-as è completamente scritto in python e la versione per OASC, che chiameremo sunny-OASC, prevede due diverse modalità di esecuzione, autok ed fkvar:

- La versione *autok*, è una versione migliorata di sunny-as che effettua un training anche sulla dimensione delle k istanze simili, differisce da T-

*Open Algorithm Selection Challenge

SUNNY[†] perché per la scelta dei risolutori utilizza l'algoritmo originale di sunny-as.

- La versione *fkvar*, effettua la fase di training sia sulla dimensione delle istanze simili che sui sottoinsiemi di features. SUNNY viene utilizzato come valutatore e viene adottata una tecnica di selezione in avanti di tipo greedy, con lo scopo di scegliere i sottoinsiemi di features per calcolare le istanze simili. Questa procedura è definita nel seguente modo: si considera l'insieme delle features «non selezionate» e se ne sceglie una alla volta, da aggiungere ad un insieme di features «selezionate», inizialmente vuoto. In questo modo si va a creare un altro insieme che è quello delle features «da testare», dopo aver calcolato il valore di K, Sunny calcola il miglior punteggio PAR_{10} che si può raggiungere con l'insieme delle features «da testare». In base al risultato, vengono aggiunte altre features, finché le prestazioni non decrescono o la configurazione non ha il numero di features cercate.

Dal momento che la fase di training è costosa a livello di computazione e di tempo, sunny-OASC non viene utilizzato su tutte le istanze disponibili, ma solamente su di alcune. Le istanze su cui viene effettuato il training vengono scelte in questo ordine:

1. Sunny-oasc associa ad ogni istanza il risolutore più veloce per essa.
2. Per ogni risolutore, le istanze vengono ordinate in ordine decrescente in base al tempo di esecuzione impiegato.
3. Per ogni risolutore, si sceglie un'istanza alla volta fino a raggiungere il numero di istanze cercate.

Per la competizione OASC si è ritenuto che la dimensione del sottoinsieme di features per ottenere buone prestazione fosse equivalente a cinque. Per questo motivo, per l'approccio *fkvar* si sono considerate sufficienti 1500 istanze e un valore di k compreso fra 3 e 30, tutto questo allo scopo di ridurre il tempo di esecuzione. Durante l'esecuzione di fkvar, viene eseguito anche *autok* con il valore di k che va da 3 a 80, *autok* viene utilizzato come risolutore speciale, così, se SUNNY produce un risultato migliore con l'utilizzo dell'intero insieme delle features, allora, si tiene il risultato di *autok*

[†]Trained SUNNY: Versione modificata di Sunny, aggiungendo una fase di training

Sunny-OASC richiede python v2 ed è composto da cinque cartelle: «data» e «results» che contengono rispettivamente i dati e i risultati della competizione OASC, «src» contiene lo script originale di sunny-as, «oasc» contiene gli script per lavorare sulla cartella «src» nella fase di training e di testing, «main» contiene gli script che chiamano automaticamente «oasc» per le diverse modalità di esecuzione.

Per eseguire lo script, va prima svolta la fase di training poi quella di test, bisogna quindi recarsi nella cartella «main» ed eseguire i seguenti comandi:

- Training:

```
main:$~ sh make_oasc_tasks.sh > tasks.txt
main:$~ sh oasc_train.sh run_fkvar tasks.txt
```

- Testing:

```
main:$~ sh make_oasc_tasks.sh > tasks.txt
main:$~ sh oasc_test.sh fkvar tasks.txt
```

Nel caso in cui volessimo cambiare metodo di approccio, basta sostituire «fkvar» con l'approccio desiderato, quelli da noi implementati, sono «randk» e «simann».

Inoltre, è possibile eseguire SUNNY su di un singolo scenario, in modo da ridurre i tempi di training e di testing, questa modalità è poi quella che abbiamo utilizzato per fare le varie prove durante la fase di progettazione e implementazione dei nuovi metodi, per usare questa modalità, bisogna prima recarsi nella cartella «oasc» e poi eseguire i seguenti comandi:

```
oasc:$~ python run_fkvar.py Caren # training,
oasc:$~ python result.py Caren fkvar # testing
```

Anche qui, ovviamente, si può cambiare approccio semplicemente sostituendo «fkvar», in aggiunta si può scegliere lo scenario su cui eseguire SUNNY andando a sostituire «Caren» con uno degli scenari disponibili. Dopo aver eseguito la fase di testing, ci tornerà dei risultati che saranno salvati in un file json con il nome dello scenario utilizzato, questo file potremo trovarlo dentro la cartella con il nome dell'approccio utilizzato, «fkvar» nel nostro caso. Di seguito riporto la lista degli scenari a disposizione, aventi come obiettivo la minimizzazione e su cui abbiamo provato i vari approcci:

```
Bado Caren Magnus Mira Monty Quill Sora Svea
```

Infine, per la valutazione dei risultati abbiamo utilizzato OASC_starterkit un wrapper[‡] che utilizza lo stesso metodo di valutazione utilizzato per la competizione oasc, questo script ritorna un valore sulla bontà in termini di efficienza dell'approccio utilizzato, con un punteggio compreso tra [0,1], per utilizzarlo bisogna prima integrare OASC_starterkit e dopo esserci posizioni in questa cartella dare il seguente comando:

```
OASC_starterkit:$~ python stats.py
                        ../oasc-master/results/fkvar/Caren.json
```

Caren.json contiene i risultati provenienti dalla fase di testing.

2.1 Approccio «randk»

Abbiamo deciso di implementare questo approccio come punto di partenza e baseline per simulated annealing, ma anche per poi poterne confrontare i risultati, l'approccio «fkvar» è stato utilizzato come punto di partenza, abbiamo creato i file run_randk.py e randk.py, copie in seguito modificate dei rispettivi file run_fkvar.py e kit.py, il limite sul numero di features fissato a 5 è stato rimosso e abbiamo invece optato per la scelta di questo valore in modo casuale e sopratutto ad ogni iterazione, in un range che poteva andare da [3,15].

Le features non vengono più scelte con la tecnica di selezione in avanti, ma vengono calcolate in modo casuale insieme ad una k scelta anch'essa casualmente, una volta ottenuto tutto l'occorrente, ovvero l'insieme delle features «da valutare» e la k, vengono poi passate alla funzione run_evaluator che calcolava il PAR_{10} e in base al risultato si decideva, se tenere questo insieme di features o se ricominciare questo ciclo mantenendo il miglior risultato ottenuto fino a quel momento. In questo metodo abbiamo impostato un totale di 1000 confronti e ritornare in output il miglior insieme di features, la miglior k e il PAR_{10} che ne risultavano.

Di seguito riporto due stringhe di codice utilizzate per effettuare il calcolo dei valori casuali:

[‡]metodo di approccio di FS, valuta i sottoinsiemi di features cercando di inviduarne le interazioni

```
elements = random.randint(3,15)
random.sample(features,elements)
```

La variabile «elements» è il numero di features che deve avere la configurazione finale, e viene calcolato ad ogni iterazione. «features» è invece il vettore delle features.

random.randint() ritorna un valore casuale intero compreso nel range indicato, random.sample() calcola invece una configurazione formata da un numero di features pari al valore di «elements», senza la possibilità di avere ripetizione.

2.2 Approccio «simann»

Per questo approccio, il punto di partenza è stata la versione «randk», con l'aggiunta di un modulo per l'utilizzo di simulated annealing, prima di introdurre però questa tecnica sembra doveroso approfondire cos'è la tecnica simulated annealing e del perché abbiamo scelto proprio questa tecnica.

2.2.1 Simulated annealing

Una tra le motivazioni principali per la quale abbiamo deciso di implementare tale tecnica è data dalle sue prestazioni, infatti, avremmo potuto usare altri algoritmi di ricerca locale, come ad esempio hill-climbing[§], ma soffre del problema di arrestarsi su dei massimi(minimi) locali,come possiamo osservare dalla figura 2.1, questo algoritmo, partendo da current state, andrebbe a muoversi in direzione crescente, fino ad arrivare a local maximum ed arrestarsi in quel punto perché non sarebbe più possibile ottenere ulteriori miglioramenti. Questo problema può essere risolto con la versione casuale che abbiamo implementato, ma risulta essere poco efficiente.

Quindi, combinando entrambe le tecniche riusciamo ad ottenere una visita completa ed efficiente e parliamo in questo caso di Simulated annealing (SA). SA fonda le sue basi nella statistica meccanica. Sviluppato originariamente da Kirkpatrick [10] per risolvere problemi di ottimizzazione combinatoria e discreta. SA è nato come metodo di simulazione della tempra (annealing)

[§]Algoritmo di ricerca, continua a muoversi verso valori crescenti e si arresta non appena trova un punto di massimo, locale o globale che sia

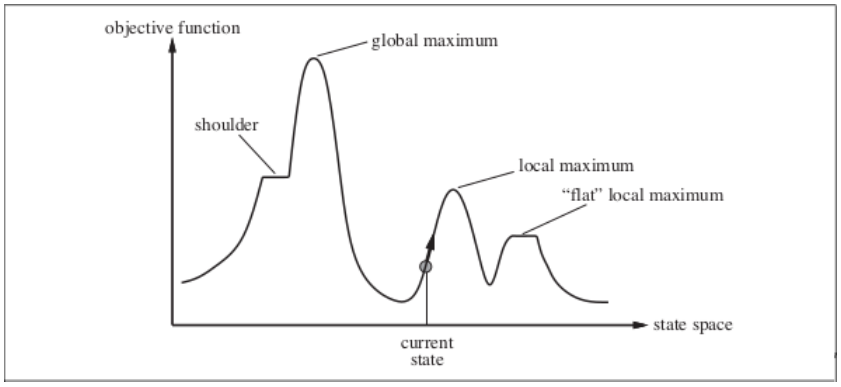

Figura 2.1: Hill climbing, con l'obiettivo di cercare un punto di massimo globale.

dei solidi. L'annealing è il processo con il quale un solido, portato allo stato fluido, tramite riscaldamento ad alte temperature, viene riportato di nuovo allo stato solido o cristallino, a temperature basse, controllando e riducendo gradualmente la temperatura.

Per comprendere SA, possiamo introdurre un altro problema, la discesa del gradiente, ed immaginare come obiettivo, quello di far scendere una pallina da ping pong nel punto più basso di una superficie rocciosa. Se lasciamo scivolare la pallina, si fermerò in un punto che è un minimo locale, scuotendo al superficie, possiamo farla uscire dal minimo locale e farla spostare verso un altro minimo, l'idea alla base è quella di scuotere la superficie in modo abbastanza forte da farla rimbalzare via dal punto di minimo locale, ma non così forte da farla uscire dal punto di minimo globale. SA si basa proprio su questo, cominciare a scuotere tanto (partendo da una temperatura alta) e ridurre gradualmente l'intensità delle scosse (abbassando gradualmente la temperatura).

SA si comporta come hill climbing, ma invece di scegliere la miglior mossa disponibile, ne sceglie una casuale. Se la nuova mossa, fornisce un miglioramento della soluzione, viene sempre accettata. In caso contrario, l'algoritmo decide se accettare la nuova mossa con un probabilità minore di 1. Tale probabilità decresce esponenzialmente in base a quanto è «cattiva» la nuova mossa. Altro valore che diminuisce la probabilità è la temperatura, più si abbassa, minori saranno le probabilità di accettare una mossa che peggiora il risultato.

L'algoritmo 1 riporta lo pseudo-codice di un esempio di Simulated Annealing:

Function `SA`$(steps, Tmax, Tmin)$**:**
  "crea una prima sequenza in modo casuale";
  $step \leftarrow 0$;
  while $(step < steps)\&\&(timer < limit)$ **do**
    `self`.$move()$ "genera nuova sequenza";
    $E \leftarrow$ `self`.$energy()$ "valuta la mossa";
    $\triangle E \leftarrow E - E'$ "nuovaE - vecchiaE";
    if $(\triangle E > 0.0)\&\&(exp(\frac{-\triangle E}{T}) < random.random())$ **then**
      "ripristina i valori precedenti";
    else
      "accetta il nuovo stato e confrontalo con il migliore"
    end
    $step \leftarrow step + 1$;
  end
return sequenza

Algorithm 1: Simulated Annealing

SA parte dalla creazione di una prima mossa casuale, che non sarà altro che la nostra configurazione di 5 feature e il valore di k, inizialmente scelti in modo casuale e con l'obiettivo di fornire un punto di partenza per le scelte di ottimizzazione, in seguito si sceglie la temperatura, che richiede due valori, un valore massimo e uno minimo, la temperatura massima è il punti di partenza che va lentamente a scendere e la temperatura minima è il punto di arresto.

Altro paramentro importante è «steps», con cui si definisce il numero di operazioni che dovranno essere effettuate prima delle terminazione, «step» è invece un contatore con valore iniziale «0» e viene incrementato di 1 per ogni operazione. Ad ogni step si calcola una nuova mossa, o modifica alla configurazione di partenza, questa nuova configurazione viene poi valutata e in base al risultato si decide poi se tenerla o se escluderla. Anche se il risultato è negativo SA prevede l'accettazione di risultati negativi con una certa probabilità:

La formula per il calcolo della probabilità per un valore che peggiora il risultato è data da questa formula:

$$exp(\frac{-\triangle E}{T}) < random.random()$$

- dove $\triangle E$ equivale alla differenza tra il nuovo valore di energia e il precedente.
- exp (x) produce l' esponenziale di x: e^x.
- random.random() produce un valore in virgola mobile compreso tra 0 ed 1.

2.2.2 Implementazione

Come già accennato, per l'implementazione siamo partiti dall'approccio randk, abbiamo creato due file run_simann.py e simann.py, e importato dentro la cartella «oasc» il modulo simanneal contenente la classe Annealer.

In tale versione, abbiamo impostato il numero di features a 5, motivo per il quale, nella creazione della configurazione di partenza, abbiamo mantenuto lo stesso procedimento, ma fissando il numero di elementi a 5.

```
random.sample(features,5)
```

Per sfruttare simulated annealing, dopo aver inizializzato la classe, abbiamo definito la nostra mossa, che andava poi valutata e in base al risultato si decideva se tenerla o se scartarla come spiegato nella sottosezione 2.2.1.

Function `Move():`
  $change \leftarrow random.\texttt{randint}(1, 10)$;
  if *(change)* > 7 **then**
    $new_range \leftarrow$list(kRange) ;
    $new\_range.\texttt{remove}(self.state[\text{"}k\text{"}])$;
  else
    $pos \leftarrow random.\texttt{choice}(new\_range)$;
    $z \leftarrow \texttt{self}.state[\text{'}feat\text{'}].split(\text{','})$;
    $featexcept \leftarrow \texttt{list}(features)$;
    $featexcept \leftarrow \texttt{list}(\texttt{set}(features) - \texttt{set}(z))$;
    $newfeat \leftarrow random.\texttt{choice}(featexcept)$;
    $z[pos] \leftarrow newfeat$ `self`$.state[\text{'}feat\text{'}] \leftarrow \text{','}.join(z)$
  end

Algorithm 2: Calcolo mossa

La variabile «change» viene utilizzata con lo scopo di rendere il cambio di una delle cinque feature più probabile rispetto al cambio del valore della k, essattamente il settanta per cento contro il tretnta, dato che le k possono essere scelte solamente fra 28 valori possibili e ce ne servirà solamente una alla volta, mentre di features ne serviranno cinque e in base allo scenario in esame potranno andare da un minimo di 46 nel caso di Quill ad un massimo di 483 nel caso di Sora, abbiamo optato per dare maggiore probabilità alla sostituzione delle features.

Durante l'esecuzione se si entra nel primo caso, si sostituisce la k con un nuovo valore, scelto tramite random.choice() tra i valori di «new_range». La variabile «new_range» è utilizzata per decidere il nuovo valore di k senza avere ripetizioni, processo ottenuto dalla rimozione del precedente valore di k dal range dei possibili valori, che come nel metodo fkvar che in randk è compreso fra 3 e 30.

Nel secondo caso, se change vale meno di 8, andiamo invece a sostituire una delle feature, la feature da sostituire viene scelta con random.choice(), Il nuovo valore, come per la k, evita la ripetizione, per raggiungere questo proposito utilizziamo la variabile featexcept, che contiene tutte le features «non selezionate».

Una volta decisa la nostra mossa dobbiamo valutarla, per fare questo utilizziamo energy():

Function `energy()`:
    $e \leftarrow$
    $run\_evaluator(src\_path, sub\_scenario\_path, \texttt{self}.state, context)$;
return e

Algorithm 3: Calcolo Energia

La funzione energy() valuta semplicemente la nostra configurazione e la nostra k, il risultato che otteniamo è poi il nostro PAR_{10}, che in base ai risultati precedenti decideremo se accettare o scartare, la probabilità di accettazione dipende dalla temperatura che abbiamo raggiunto, infatti, una temperatura più alta ci garantisce una probabilità di accettazione maggiore. L'esecuzione dell'algoritmo e di questo ciclo continua finché non raggiungiamo il punto di raffreddamento.

2.2.3 Scelta dei parametri

Per ottenere buoni risultati da simulated annealing, si devono impostare dei buoni parametri, in particolare relativi alla temperatura massima, alla temperatura minima e al numero di passi a disposizione.

La scelta di tali parametri gioca un ruolo chiave in simulated annealing, perchè queste scelte possono fornire un grandissimo impatto all'efficienza del metodo, purtroppo non è possibile scegliere dei parametri perfetti che possano andare bene per tutti i casi e non esiste nemmeno una via generale per scegliere i più adatti ad uno specifico problema, ma possiamo semplicemente avvicinarci con delle approssimazioni.

Come definito da Kirkpatrick et al. [10], il valore della temperatura T, deve essere settato con un valore abbastanza grande da rendere la probabilità iniziale di accettazione quasi pari ad 1. Però c'è da tener conto che una temperatura iniziale troppo alta può causare peggiori prestazioni e tempi di computazione più lunghi.

In particolare abbiamo utilizzato simulated annealing con due diversi tipi parametri, una volta con quelli di default che comprendono una temperatura massima di 25000.00, una temperatura minima di 2.5 e 4000 steps, in realtà i 4000 steps sono una limitazione per garantire tempi di esecuzione ragionevoli, dato che per fare una media abbiamo eseguito ogni singolo scenario per 10 volte, con un tempo massimo di 4 ore, oltre il quale, l'esecuzione veniva interrotta.

La seconda volta, abbiamo utilizzato dei parametri ottimizzati, nel nostro caso per settare questi parametri, abbiamo utilizzato una funzione già presente nel modulo di Simulated Annealing, tale funzione auto() è stata utile per calcolare dei parametri di partenza, non ottimi, perchè come già detto risulta difficile farlo, ma che ci hanno permesso di ottenere buoni risultati nella maggioranza dei casi , abbiao calcolato i valori automatici per ogni scenario e ne abbiamo fatto una media, questo ci ha permesso di avere tempi di esecuzione decisamente ridotti rispetto ai parametri di default, senza mai andare oltre il tempo limite. I parametri risultanti sono stati impostati a 2800 steps, 50000.00 come temperatura massima e 10.00 come temperatura minima.

Capitolo 3

Risultati

In questo capitolo presentiamo i risultati sperimentali ottenuti dai nostri metodi per la selezione delle features. I risultati vengono forniti come una media del punteggio su 10 diverse esecuzioni. Abbiamo preso come risultato finale il gap tra SBS e VBS, utilizzando lo stesso approccio della competizione OASC, abbiamo quindi ritenuto un punteggio vicino ad 1 come VBS e con punteggio di 0 come SBS.

Abbiamo visto il comportamento dei 3 metodi, «randk», «simann» e «simann-mod» su otto diversi scenari, analizzandone il tempo di esecuzione e il punteggio finale. Come ultimo passo abbiamo ricavato un punteggio medio tra tutti gli scenari per individuare un approccio vincitore che nel complesso si comportasse meglio degli altri.

3.1 Informazioni sull'esperimento

Gli esperimenti sono stati effettuati su due laptop ASUS modello n56j, entrambi con processore Intel core i-7 4770HQ, SSD da 240gb, uno con 8gb di RAM, l'altro con 16. In ogni caso l'esperimento è stato svolto su di una macchina virtuale con Ubuntu 18.04.1 LTS a 64 bit e 4 GB di RAM dedicata e ha richiesto circa 70 ore di lavoro per ogni dispositivo. Ci riferiremo al primo laptop con 8gb di RAM con L_A e a quello con 16gb di RAM con L_B.

3.2 Scenari

Nel nostro caso, abbiamo utilizzato otto scenari, ve ne sarebbero altri ma hanno come obiettivo la massimizzazione e in questa tesi ci siamo occupati solo di problemi di minimizzazione. Nella tabella 3.1 possiamo approfondire gli scenari sul quale abbiamo lavorato, fornendo informazioni e caratteristiche su di essi, le informazioni sugli scenari presi in esame possono essere trovate nella tabella 3.1, con numero di algoritmi, di feature e di istanze, nel dettaglio facciamo una distinzione tra le istanze della fase di training e le istanze della fase di testing:

Scenario	Algoritmi	Features	Istanze Training	Istanze Testing
Bado	8	86	786	393
Caren	8	95	66	34
Magnus	19	37	400	201
Mira	5	143	145	73
Monty	18	37	420	210
Quill	24	46	550	275
Sora	10	483	1333	667
Svea	31	115	1076	538

Tabella 3.1: Scenari presi in esame.

Come si può vedere dalla tabella, visto il ridotto numero di istanze, Caren è stato lo scenario più semplice e di conseguenza il più veloce come tempi di esecuzione per la fase di trainng, mentre il più complesso e su cui sono state impiegate la maggioranza delle ore di esecuzione è stato Sora. L'approccio utilizzato ha ovviamente influenzato i tempi di esecuzione e anche la macchina sulla quale sono stati effettuati i test seppur con minor peso, e possiamo concludere che il metodo «randk» è stao il più veloce, ma producendo risultati instabili, a seguire «simann-mod». è stato leggermente più lento di randk, mentre simann, non avendo dei paramatri ottimizzati ha impiegato il doppio e in alcuni casi anche il triplo del tempo richiesto da simann-mod. Proprio per questo motivo, e per produrre dei risultati in tempo ragionevole, considerando che ogni scenario andava eseguito 10 volte per ogni approccio, abbiamo deciso

di introdurre un timeout, dopo il quale l'esecuzione si arrestava e forniva il risultato migliore ottenuto fino a quel momento.

Tuttavia, gli unici scenari in cui abbiamo superato questo timeout sono stati Sora, e in alcuni casi anche Svea, inoltre, l'unico approccio che ha superato il timeout delle 4 ore è stato «simann»; i casi dove Svea ha superato il timeout, si sono registrati tutti e quattro su L_Ae sono state le quattro esecuzioni effettuate su questo laptop, hanno riportato rispettivamente un numero di step compiuti pari a 3840,3800,3880,3240. Mentre, nel caso di Sora, entrambe le macchine hanno superato il timeout in tutte le 5 esecuzioni che hanno effettuato, quindi hanno impiegato lo stesso tempo, ma hanno avuto un numero di step compiuti diverso, nel caso di L_A i passi compiuti sono stati circa di 1500 a differenza di $L_B$che ha avuto una media più vicina a 1800.

Nella tabella 3.3 possiamo vedere in percentuale su quante delle 10 esecuzioni si è superato il limite temporale, nella seconda colonna invece viene mostrata una media di quanti step sono stati compiuti nel caso in cui si eccedeva il tempo limite. Possiamo evidenziare come nel caso di Sora, non si sia arrivati nemmeno a metà del numero di step prefissati, questo spiega il motivo per cui «simann» su «Sora»abbia prodotto un risultato mediamente più basso rispetto agli altri due approcci.

Scenario	Sunny simann	avg. step
Svea	40%	3690
Sora	100%	1685

Tabella 3.2: La percentuale si riferisce al numero di esecuzioni nella quale si è superato il limite di tempo delle 4h

3.3 Prestazioni FS

Per fornire un resoconto su come sono andati gli approcci per la selezione delle feature, abbiamo utilizzato un valore tra 0 ed 1, di cui abbiamo già parlato, ed è chiamato «Gap».

Nella tabella 3.3 vi sono i risultati per ogni scenario con ogni approccio utilizzato, tali risultati sono la media dei valori di gap ottenuti durante le 10 esecuzioni, in grassetto è evidenziato il risultato medio migliore. I tre approcci che hanno preso parte all'esperimento sono:

- Sunny randk: approccio con comportamento casuale, calcola una configurazione di n feature ad ogni passo ed un valore per k, valuta il tutto e tiene il risultato migliore tra 1000 confronti.
- Sunny simann: approccio che utilizza la tecnica di ottimizzazione di simulated annealing, abbiamo impostato $Tmax$ a 25000.00, $Tmin$ a 2.5 e *steps* a 4000.
- Sunny simann-mod: approccio basato su simann ma modificato nei parametri, in questo metodo abbiamo calcolato dei parametri più specifici, facendo delle esecuzioni di prova per valutare quali fossere dei parametri accettabili, alla fine abbiamo impostat $Tmax$ a 50000.00, $Tmin$ a 10.00 e *steps* a 2800.

Scenario	Sunny randk	Sunny simann	Sunny simann-mod.
Bado	0.7898	0.7800	**0.7900**
Caren	0.7760	0.8380	**0.8476**
Magnus	0.5268	0.5890	**0.6067**
Mira	**0.0318**	-0.1519	-0.1604
Monty	0.6803	**0.8831**	0.8751
Quill	0.7835	0.8325	0.**8331**
Sora	0.2955	0.2687	**0.3556**
Svea	0.5484	0.5933	**0.6213**
Media	0.5540	0.5791	**0.5962**

Tabella 3.3: Gap su ogni singolo scenario e Gap finale

In particolare «simann-mod», differisce da «simann» per i valori della temperatura massima, della temperatura minima e del numero di step da effettuare, è stato quello che si è distinto rispetto agli altri due approcci in quanto a punteggio medio.

Lo scenario dove si è comportato meglio è stato Monty, anche se «simann» ha paradossalmente ottenuto un punteggio migliore, c'è pero da tener conto che «simann-mod» ha ottenuto quel risultato con tempi medi di ($\approx$) 14min, a differenza di «simann» che ne ha impiegati invece ($\approx$) 26min. Lo scenario

dove «simann-mod» ha invece ottenuto un punteggio peggiore è stato «Mira», questo scenario è anche stato l'unico, dove l'approccio «randk» ha avuto una resa migliore rispetto ad entrambi gli approcci di simulated annealing.

Il punteggio più alto è stato registrato durante l'ottava esecuzione di «Monty» con «simann» e ha riportato un punteggio di 0.9865, mentre il più basso è stato ottenuto sempre da «simann» sullo scenario «Mira» con un punteggio di -0.8496.

Calcolando il gap medio su tutti gli scenari il vincitore risulta «simann-mod» con un risultato di 0.5962, a seguire «simann» con un punteggio di 0.5791 ed infine «randk» con 0.5540. Possiamo quindi concludere che l'approccio che ha prodotto i risultati migliori è stato «simann-mod» ed in generale gli approcci di simulated annealing si sono comportati meglio rispetto ad un approccio totalmente casuale.

Abbiamo fatto un ultimo confronto tra il metodo migliore trovato durante la nostra sperimentazione, «simann-mod», ed il metodo di partenza «fkvar», che come ricordiamo è stato presentato alla OASC challenge del 2017:

- Sunny fkvar: approccio che presenta un metodo di visita sequenziale, parte da una lista di feature inizialmente vuota ed effettua i confronti su tutto l'insieme di feature, scegliendo quella che produce il miglior risultato, in questo modo crea la configurazione cercata, formata da 5 feature e dal miglior valore di k per essa.

Nel nostro confronto abbiamo notato che l'approccio più efficiente è stato comunque quello di partenza «fkvar», e il caso che richiama maggiore attenzione si ha sullo scenario Mira dove c'è una grande differenza di punteggio, pari a 0.5924. Nel complesso, parlando di Gap finale «fkvar» ne esce vincitore con un punteggio di 0.6401, contro i 0.5962 di «simann-mod», i dettagi del confronto possono comunque essere consultati nella tabella 3.4, dove sono evidenziati gli scenari in cui «fkvar» ha avuto una resa migliore rispetto all'approccio di simulated annealing e anche quelli dove è stato invece superato.

Nel complesso possiamo concludere che l'approccio di simulated annealing se definito con dei buoni parametri produce ottimi risultati sopratutto con ottimi tempi di esecuzione, in particolare si possono provare diverse temperature per ottenere risultati diversi, da quello che abbiamo notato durante la sperimentazione, una temperatura troppo bassa, come nel caso di «simann» non ci permette di esplorare per bene lo spazio delle soluzioni, causando così

Scenario	Sunny fkvar	Sunny simann-mod.
Bado	**0.8470**	0.7900
Caren	**0.9450**	0.8476
Magnus	0.5807	**0.6067**
Mira	**0.4320**	-0.1604
Monty	**0.9105**	0.8751
Quill	0.5691	0.**8331**
Sora	0.1786	**0.3556**
Svea	**0.6578**	0.6213
Media	**0.6401**	0.5962

Tabella 3.4: Risultati fkvar

risultati più instabili, in particolare, in alcuni scenario abbiamo notato differenze di punteggio di 0.4 tra un'esecuzione e l'altra. Se si scelgono invece dei parametri più adeguati, nel nostro caso alzando la temperatura massima, si riesce ad avere una visione più completa ottenendo così dei risultati più stabili, tenendo sempre in cosiderazione, che la nuova mossa dell'algoritmo di simulated annealing viene operata in modo casuale ed in casi particolarmente sfortunati, si possono ottenere ugualmente risultati affetti da rumore.

Conclusione

Lo scopo di questa tesi è stato quello di valutare, metodi di selezione per le feature diversi da quelli che erano già stati proposti, in particolare andando ad evidenziare i benefici che si possono raggiungere introducendo una tecnica di ottimizzazione come simulated annealing e dimostrando che con l'uso di tali tecniche si possono ottenere ottimi risultati in tempi ragionevoli, ovviamente, questa tesi non si propone come un risultato definitivo, ma lascia aperti varie possibilità tra cui quella di prendere questo metodo come paragone per altre tecniche di FS o anche la possibilità di effettuare modifiche sui parametri delle tecniche di simulated annealing, in paricolare si possono variare le temperature e gli step da effettuare per vedere quali risultati si possono produrre, o ancora si può modificare la funzione move(), da cui si può decidere se cambiare più feature contemporaneamente, rispetto alla nostra move() che modifica il valore di un'unica feature, o cambiare la probabilità di modifica delle feature rispetto al valore di k, al momento tale valore di scelta è fissato al 30% per la sostituzione della k e al 70% per sostituire una feature.

Ringraziamenti

Bibliografia

[1] Roberto Amadini, Fabio Biselli, Maurizio Gabbrielli, Tong Liu, Jacopo Mauro. SUNNY for Algorithm Selection: A Preliminary Study. CILC, Jul 2015, Genova, Italy. <hal-01227595>

[2] Roberto Amadini, Maurizio Gabbrielli, Jacopo Mauro. SUNNY: a Lazy Portfolio Approach for Constraint Solving. TPLP

[3] Bischl, B. and Kerschke, P. and Kotthoff, L. and Lindauer, M. and Malitsky, Y. and Frechétte, A. and Hoos, H. and Hutter, F. and Leyton-Brown, K. and Tierney, K. and Vanschoren, J. ASlib: A Benchmark Library for Algorithm Selection In: Artificial Intelligence Journal (AIJ) (2016), (pp. 41–58)

[4] Roberto Amadini, Fabio Biselli, Maurizio Gabbrielli, Tong Liu, Jacopo Mauro. Feature Selection for SUNNY: a Study on the Algorithm Selection Library. ICTAI, Nov 2015, Vietri sul Mare, Italy. <hal-01227600>

[5] Tong Liu, Roberto Amadini, Jacopo Mauro. Sunny with algorithm configuration. Proceedings of Machine Learning Research 79:12-14, 2017

[6] Marius Lindauer, Rolf-David Bergdoll, Frank Hutter (2016). An empirical study of per-instance algorithm scheduling. Proceedings of the LION10 (pp. 253–259)

[7] Stuart J. Russell, Peter Norvig. Artificial Intelligence A Modern Approach, (pp. 120-125)

[8] K-nearest neighbors algorithm, Wikipedia, The Free Encyclopedia.

[9] Park, M.-W. and Y.-D. Kim. (1998). "A Systematic Procedure for Setting Parameters in Simulated AnnealingAlgorithms."Computers & Operations Research24(3), 207–217

[10] Kirkpatrick, S., Gelatt, C. D.Jr. and Vecchi, M. P., Optimization by simulated annealing. Science, 1983, 220, 671-680