Università degli studi di Torino

Dipartimento di Informatica

Intelligenza Artificiale e Laboratorio

Travel Agency

Giacomo Bonazzi - Marco Caldera - Giacomo Costarelli

27 novembre 2019

Indice

1 Introduzione

Questo studio è frutto di un lavoro collaborativo all'interno dell'insegnamento di Intelligenza Artificiale ed in questa relazione consideriamo e spieghiamo tutte le scelte che abbiamo effettuato per completare il progetto.

Il progetto ha richiesto di sviluppare un sistema esperto che possa essere utilizzato da un'agenzia di viaggi per suggerire ai clienti dei pacchetti vacanze a seconda di alcune informazioni da essi fornite. In particolare nella relazione spieghiamo come abbiamo sviluppato il dominio, che tipo di scelte abbiamo fatto, come abbiamo modellato l'incertezza e come siamo poi giunti così alla soluzione finale.

Terminato lo sviluppo abbiamo poi creato un'applicazione per smartphone al fine di visualizzare graficamente i viaggi creati dal nostro applicativo, rendendoci meglio conto dell'effettiva bontà del sistema creato.

2 Progettazione

Il progetto è basato sull'utilizzo di CLIPS, uno strumento creato dalla NASA per supportare lo sviluppo di sistemi esperti. Questo sistema ha subito svariate migliorie nel corso degli anni ed è stato costruito per facilitare, mediante l'utilizzo di regole, lo sviluppo di software che siano in grado di modellare la conoscenza umana ed in particolare quella degli esperti del nostro dominio di interesse.

In questa sezione spieghiamo i diversi costrutti utilizzati e forniamo una giustificazione delle scelte che abbia effettuato ripercorrendo i problemi che abbiamo riscontrato e le ulteriori migliorie che potrebbero essere implementate.

2.1 Template

Una volta analizzati i requisiti del sistema il primo passo è stato quello di andare a definire alcuni template in grado di fornirci la possibilità di creare fatti strutturati in modo da rappresentare i diversi oggetti del dominio.

In base a quanto stabilito nei documenti di progetto abbiamo così creato i seguenti template.

2.1.1 Location

```
;Rappresenta una cittadina presente sul territorio italiano
(deftemplate place
    (slot name (type SYMBOL))
    (slot region (type SYMBOL))
    (slot ID (type INTEGER))
    (multislot coordinates (type FLOAT) (cardinality 2 2))
)

;Tipo di turismo associato ad una cittadina
(deftemplate tourism-type
    (slot place-ID (type INTEGER))
    (slot type (type SYMBOL))
    (slot score (type INTEGER))
)

;Alberghi presenti sul territorio italiano
(deftemplate resort
    (slot name (type SYMBOL))
    (slot star (type INTEGER))
    (slot rooms-number (type INTEGER))
    (slot place-ID (type INTEGER))
)
```

Codice 1: Location template

Nel Codice 1 abbiamo i template che modellano gli aspetti legati ai luoghi presenti all'interno del sistema. Nello specifico abbiamo:

Place Questo template serve per modellare le città presenti sul territorio italiano. Al suo interno è composto dal nome della città, la regione di appartenza, un ID univoco che lo rappresenti e un multislot chiamato *coordinates* che contiene longitudine e latitudine della città.

Utilità del template:

- Permette di associare i tipi di turismo direttamente alle città
- Permette, nel corso della computazione, di effettuare alcune scelte basate sulla regione a cui appartiene un luogo (e.g., l'utente gradisce visitare una specifica regione e quindi è bene andare a penalizzare i luoghi che non appartengono ad essa)
- Permette, attraverso le coordinate, di calcolare la distanza tra due città. Tale informazione risulta infatti essere determinante nella scelta dei viaggi in modo da prevenire spostamenti troppo lunghi.

Tourism type Il secondo template è legato al tipo di turismo presente nelle città. Ogni città può avere tanti tipi di turismo e per ogni turismo, come da requisiti, abbiamo assegnato un punteggio (*score*) da 0 a 5 al fine di rappresentare la correlazione tra una città e un determinato tipo di turismo. La principale utilità di questo template è quella di fornire delle informazioni sulle città in modo che tali informazioni possano essere usate per creare un viaggio il più fedele possibile alle preferenze dell'utente.

Resort Infine l'ultimo template legato ai luoghi presenti nel sistema è quello che ci permette di specificare quali siano gli hotel effettivamente presenti sul territorio e con essi alcune informazioni utili come le stanze disponibili, le stelle, il luogo in cui si trova l'hotel, etc. Come per il template sul tipo di turismo la principale utilità di questi fatti è quella di aiutare a trovare una sistemazione per i viaggi dell'utente in modo che essa sia il più possibile consona con le aspettative dei viaggiatori.

2.1.2 Trip

Durante la computazione abbiamo la necessità di creare dei viaggi da mostrare all'utente. Per fare ciò abbiamo strutturato un template chiamato *trip* in grado di rappresentare le caratteristiche di ogni viaggio con il grado di confidenza con cui il sistema ritiene che esso sia un viaggio con caratteristiche simili a quelle fornite dall'utente.

```
(deftemplate trip
   (multislot resort-sequence (type SYMBOL))
   (multislot place-sequence (type SYMBOL))
   (multislot certainties (type FLOAT))
   (multislot days-distribution (type INTEGER))
   (multislot price-per-night (type INTEGER))
   (slot penalty (default FALSE))
)
```

Codice 2: Trip template

Ogni viaggio proposto all'utente è rappresentato dalle seguenti informazioni:

- La lista degli hotel in cui pernottare
- La lista delle città che si visitano
- Il grado di sicurezza con cui il sistema ritiene che il viaggio rispetti le preferenze dell'utente

- La distribuzione dei giorni nei vari luoghi
- Il costo totale del pernottamento in ogni tappa
- Un'indicazione del fatto che sia stata aggiunta o meno una penalità dovuta alla distanza tra le città del viaggio

2.1.3 Template di supporto

All'interno del programma sono inseriti due template di supporto alla computazione.

```
(deftemplate rule
    (slot certainty (type FLOAT) (range -1.0 +1.0) (default 1.0))
    (multislot if)
    (multislot then)
)

(deftemplate travel-benchmark
    (slot name)
    (slot value)
    (slot certainty (type FLOAT) (range -1.0 +1.0) (default 1.0))
)
```

Codice 3: Template di Supporto

Il template *rule* serve a poter modellare le regole dell'esperto del dominio. Ogni regola, se soddisfatta, permette di immettere nel sistema una serie di informazioni, insieme al loro grado di sicurezza, sulle possibili preferenze dell'utente. Tali informazioni vengono modellate attraverso il template *travel banchmark*.

2.2 Workflow

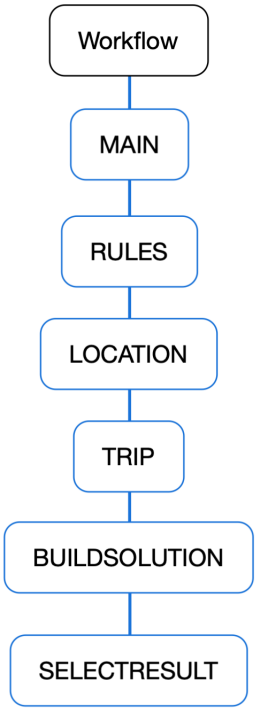


Figura 1: Stack dei moduli del progetto

Il flusso d'esecuzione del programma è suddiviso in diversi moduli che permettono una corretta gestione del sistema ed una suddivisione ottimale delle regole. In totale abbiamo definito 6 moduli che andiamo ora a presentare brevemente per poi dettagliarli successivamente nel paragrafo dedicato alle regole:

- **MAIN**

  Modulo per la gestione del flusso d'esecuzione e per la definizione di alcune regole utili in diversi moduli del programma.

- **RULES**

  Modulo che contiene le principali regole dell'esperto del dominio. Queste regole permettono di creare i fatti di tipo *travel banchmark* che permettono di guidare l'esecuzione del programma in favore di certi viaggi piuttosto che altri.

- **LOCATION**

Questo modulo ha lo scopo di analizzare le informazioni presenti nel sistema e generare un insieme di luoghi interessanti per la costruzione del viaggio.

- **TRIP**

  Il modulo TRIP è quello più importante dell'applicazione in quanto si occupa di creare i diversi possibili viaggi. In questo modulo vengono effettivamente composti i diversi possibili viaggi seguendo sempre i requisiti ed i vincoli del sistema.

- **BUILDSOLUTION**

  BUILDSOLUTION determina il grado di certezza di ogni viaggio andando a penalizzare i viaggi che contengono spostamenti tra luoghi posti a più di 100km di distanza.

- **SELECTRESULT**

  Questo è l'ultimo modulo che viene eseguito e va semplicemente ad assicurarsi che per ogni gruppo di città usato nei viaggi ne venga selezionato solamente uno, quello con il grado di certezza maggiore.

2.3 L'uso dei CF

Il nostro sistema esperto deve essere in grado di ragionare in un ambiente incerto. I viaggi proposti all'utente sono infatti frutto di una modellazione incerta degli oggetti del dominio e pertanto il programma deve poter prendere decisioni a seconda dei gradi di certezza ed incertezza che ha sui diversi fatti del mondo. La modellazione di questi gradi di incertezza (*certainty factors* (CF)) si basa sulla teoria e sugli esempi visti a lezione e sfrutta delle tecniche che permettono di creare e modificare il valore di una *certainty* in maniera coerente rispetto ai requisti progettuali.

Il nostro programma utilizza principalmente i CF in due modi: per valutare le caratteristiche dei viaggi basandosi sulle preferenze dell'utente e per assegnare, al soddisfacimento di certe condizioni, alcune penalità ad essi.

Preferenze dell'utente Come detto in precedenza, in base alle preferenze dell'utente vengono creati all'interno del sistema dei fatti di tipo *travel-benchmark*. Ad ognuno di essi è associato un fattore di incertezza ed il calcolo di questo fattore si basa sulla conoscenza dell'esperto del dominio e funge da guida nei calcoli che permettono di valutare i diversi viaggi.

Penalità La seconda modalità secondo cui abbiamo utilizzato i CF ci permette invece di introdurre delle penalità. In base a certe caratteristiche dei viaggi, dovute sia alle preferenze dell'utente (e.g., regione preferita) che a vincoli di dominio (e.g., lo spostamento tra due tappe di un viaggio non deve superare i 100km) può essere necessario andare a penalizzare un certo viaggio piuttosto che un altro. Alcune regole contengono quindi delle particolari condizioni per cui certe caratteristiche dei viaggi vengono penalizzate e di conseguenza alcuni di essi potrebbero risultare sfavoriti rispetto ad altri.

3 Sviluppo

In questa sezione vediamo, attraverso la presentazione delle regole sviluppate, il modo in cui abbiamo ragionato per creare l'insieme di viaggi da mostrare all'utente. Al fine di migliorare la chiarezza dell'elaborato abbiamo suddiviso concettualmente il progetto in 3 parti:

- Una prima fase fortemente caratterizzata dall'interazione tra requisti dell'utente e regole dell'esperto di dominio.
- Un seconda fase che rappresenta il nucleo centrale dell'applicazione e porta alla creazione dei diversi viaggi.
- Un'ultima fase in cui si scelgono i migliori viaggi trovati.

3.1 Prima fase

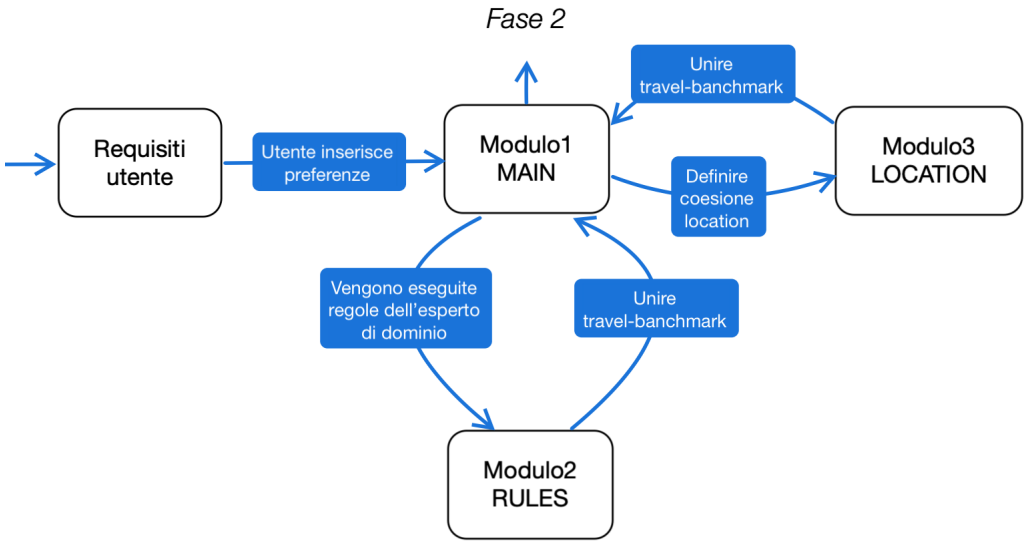


Figura 2: Fase 1

In questa sezione parliamo della fase iniziale dell'esecuzione. Lo scopo di questa parte, sintetizzato dalla Figura 2, vuole essere quello di analizzare i requisti dell'utente creando all'interno del sistema un insieme di fatti (i.e., *travel-benchmark*) derivanti dall'interazione delle preferenze dell'utente con le regole dell'esperto del dominio (definite nel modulo RULES). Una volta creati questi fatti abbiamo usato il modulo LOCATION per assegnare ai diversi luoghi presenti nel sistema il grado di certezza secondo cui sarebbe opportuno sceglierli.

3.1.1 Requisiti dell'utente

Prima di proseguire con l'analisi del flusso d'esecuzione del programma può essere utile andare a capire come abbiamo gestito i requisti dell'utente.

Un utente alla ricerca di nuovi viaggi deve immettere nel nostro sistema un insieme di fatti (chiamati *travel-benchmark*) così che l'esecuzione possa essere guidata verso i viaggi più appropriati.

In una prima fase di sviluppo abbiamo creato un modulo QUESTIONS in grado, attraverso il terminale, di porre delle domande all'utente che opzionalmente poteva fornire delle risposte andando così a definire la conoscenza secondo cui elaborare gli itinerari. Questo modulo era presente in cima allo stack d'esecuzione ma non è più presente nella versione finale, presentata in

questo elaborato, in quanto in questa versione l'utente inserisce le informazioni attraverso un'applicazione mobile che abbiamo creato (presentata nella Sezione 4). Tale modulo ora eliminato era basato sull'omonimo modulo presente nel nel sistema esperto *wine.clp* messo a disposizione su moodle e sulla documentazione di clips.

La discussione che stiamo per affrontare presuppone dunque l'esistenza all'interno del sistema dei requisti dell'utente. Terminata questa parte mostreremo nella sezione dedicata all'applicazione mobile la modalità secondo cui l'utente può inserire liberamente le proprie preferenze.

3.1.2 Avvio della computazione

Il modulo MAIN contiene la prima regola che deve essere lanciata durante l'esecuzione ed un insieme di funzioni utili in più moduli dell'applicazione.

```
(defrule MAIN::start
    ;salience massima
    (declare (salience 10000))
    =>
    ;permette di avere fatti duplicati
    (set-fact-duplication TRUE)
    ;ordine dello stack focus
    (focus RULES LOCATION TRIP BUILDSOLUTION SELECTRESULT)
)
```

Codice 4: Regola di start

La regola nel Codice 4 è molto semplice e garantisce che essa sia la prima regola eseguita dal programma. Permette di definire lo stack secondo cui eseguire i moduli e di abilitare la possibilità di avere dei fatti duplicati. Normalmente su CLIPS non si possono creare due fatti identici ma, come vedremo nel paragrafo dedicato alle regole dell'esperto del dominio, il nostro sistema, al fine di modellare correttamente la conoscenza, ne richiede l'uso.

3.1.3 Regole dell'esperto del dominio

Dopo che la regola *start* viene eseguita lo stack assegna il focus al modulo RULES. Questo modulo contiene la conoscenza dell'esperto del dominio. Questa conoscenza viene implementata seguendo le indicazioni fornite a lezione sul sistema esperto MYCIN e ispirandosi al programma *wine.clp*.

```
(rule
    (if personal-trait is avventura)
```

```
    (then tourism-type is naturalistico with certainty 0.4 and
        tourism-type is balneare with certainty -0.2 and
        tourism-type is sportivo with certainty -0.2 and
        tourism-type is termale with certainty -0.2 and
        tourism-type is lacustre with certainty 0.5)
)
```

Codice 5: Esempio di regola dell'esperto di dominio

Il Codice 5 rappresenta un esempio di regola dell'esperto del dominio e va interpretata in questo modo: se un'utente ha risposto con *avventura* alla domanda relativa ai tratti personali (*personal-trait*) allora possiamo assumere con un certo grado di certezza (*certainty*) che preferisca certi tipi di turismo (*tourism-type*) piuttosto che altri.

Al fine di utilizzare la conoscenza formalizzata nel modo appena descritto abbiamo quindi usato delle semplici regole di clips che ci hanno permesso di immettere nel sistema i *travel-benchmark* relativi ai corpi delle regole il cui antecedente è soddisfatto.

3.1.4 Raffinamento travel-benchmark

Visto che il numero di regole definite dell'esperto del dominio è potenzialmente elevato, abbiamo definito, nel modulo MAIN, una regola in grado di ottenere il focus in maniera automatica ogni qual volta esistano due *travel-benchmark* creati sullo stesso *name* (e.g., tipo di turismo) e *value* (e.g., balneare). In questo modo possiamo andare ad unire i due fatti definendone uno solo con una nuova *certainty* basata sui valori dei due fatti che si stanno considerando.

```
;Combina insieme i cf dei travel-benchmark che hanno stesso name e value
(defrule MAIN::combine-certainties
    (declare (salience 100) (auto-focus TRUE))
    ?rem1 <- (travel-benchmark (name ?rel) (value ?val) (certainty ?c1))
    ?rem2 <- (travel-benchmark (name ?rel) (value ?val) (certainty ?c2))
    (test (neq ?rem1 ?rem2))
    =>
    (retract ?rem1)
    (modify ?rem2 (certainty (new-certainty ?c1 ?c2)))
)

(deffunction MAIN::new-certainty (?c1 ?c2)
    (if (and (>= ?c1 0) (>= ?c2 0))
    then (return (- (+ ?c1 ?c2) (* ?c1 ?c2)));(c1+c2)-(c1*c2)
    else (if (and (< ?c1 0) (< ?c2 0))
        then (return (+ (+ ?c1 ?c2) (* ?c1 ?c2)));(c1+c2)+(c1*c2)
```

```
17          else (return (/ (+ ?c1 ?c2) (- 1 (min (abs ?c1) (abs ?c2)))));(c1
    +c2)/(1-min(|c1|,|c2|))
18      )
19  )
20 )
```

Codice 6: Descrizione dell'unione dei *travel-benchmark*

Il Codice 6 implementa il concetto appena descritto. La funzione *new-certainty* serve per assicurare che le certainty siano unite in maniera corretta indipendentemente che siano:

- Entrambe positive
- Entrambe negative
- Una positiva ed una negativa

3.1.5 Definire la conformità delle location

Terminato lo studio delle regole di dominio inizia l'esecuzione del modulo LOCATION. Questo modulo ha l'unica ed importantissima funzione di stabilire, per ogni luogo presente all'interno dei sistemi dell'agenzia viaggi, con quanta certezza tali luoghi si rivelano essere una buona destinazione per il tipo di viaggio che l'utente vuole fare.

```
1  (defrule LOCATION::select-location
2      (place (name ?place-name) (region ?region) (ID ?id))
3      (tourism-type (place-ID ?id) (type ?tourism-type) (score ?score))
4      (resort (name ?resort-name) (star ?resort-star) (place-ID ?id) (
       rooms-number ?rooms-number))
5
6      (travel-benchmark (name min-resort-star) (value ?min-resort-star&~
       unknown))
7      (travel-benchmark (name tourism-type) (value ?tourism-type) (certainty
        ?c1))
8      (travel-benchmark (name people-number) (value ?people-number))
9
10     ;rispettare il numero minimo di stelle
11     (test (>= ?resort-star ?min-resort-star))
12     ;escludere i resort che non hanno abbastanza stanze
13     (test (>= (* ?rooms-number 2) ?people-number))
14     =>
15     (assert
16         (travel-benchmark (name location) (value ?resort-name)
17                                     (certainty (min ?c1 (/ ?score 5))))
18     )
```

```
19 )
```

Codice 7: Generazione accuratezza location

Il Codice 7 serve ad implementare il seguente concetto: per ogni tipo di turismo di una località se, grazie alle regole dell'esperto di dominio, abbiamo definito che l'utente ha mostrato un certo grado di interesse per quel turismo allora per ogni resort presente in quel luogo possiamo andare a definire un grado di certezza per cui esso potrebbe essere un luogo adatto da visitare. Questo grado di certezza viene definito in base al valore minimo tra due elementi:

- La certezza secondo cui quel tipo di turismo è importante per le preferenze dell'utente
- La certezza secondo cui una città è importante un certo tipo di turismo (e.g., un resort che si trova in una città marittima ha uno score associato al turismo balneare molto alto).

In questo modo creiamo dei nuovi fatti che associano ad ogni possibile resort una nuova *certainty* basata sulle caratteristiche del luogo e sulle preferenze dell'utente in modo che il sistema sia in grado di costruire i viaggi nella maniera più appropriata possibile.

Un'ultima importante considerazione sul Codice 7 è che, grazie alle righe 10-13, ci assicuriamo che vengano presi in considerazione solamente i resort con un numero di stelle superiore o uguale al minimo richiesto e che i resort che non hanno un numero di stanze sufficiente a contenere tutte le persone che partecipano al viaggio vengano scartati.

```
1  (defrule LOCATION::penalize-location
2      (place (name ?place-name) (region ?region) (ID ?id))
3      (resort (name ?resort-name) (star ?resort-star) (place-ID ?id) (
       rooms-number ?rooms-number))
4
5      (travel-benchmark (name favourite-region) (value ?fav-region&~unknown)
       )
6      (travel-benchmark (name min-resort-star) (value ?min-resort-star&~
       unknown))
7      (travel-benchmark (name people-number) (value ?people-number))
8      (test (neq ?fav-region ?region))
9      (test (>= ?resort-star ?min-resort-star))
10     (test (>= (* ?rooms-number 2) ?people-number))
11     =>
12     (assert
13         (travel-benchmark
```

```
            (name location)
            (value ?resort-name)
            (certainty -0.3)
        )
    )
)
```

Codice 8: Penalizzare location

La seconda ed ultima regola del modulo LOCATION (mostrata nel Codice 8 si occupa, qualora sia stata selezionata una regione preferita, di penalizzare tutti i resort che non si trovino in tale regione. Per fare ciò viene creato un fatto legato alla location con una *certainty* negativa.

Come è facilmente intuibile queste operazioni possono portare alla creazione di più *travel-benchmark* per ogni resort e pertanto, in modo analogo a quanto mostrato nel Paragrafo 3.1.4, viene richiama la regola *combine-certainties* che unisce le *certainty* dei resort producendo il grado di certezza finale secondo cui un determinato albergo debba essere scelto.

3.2 Seconda fase

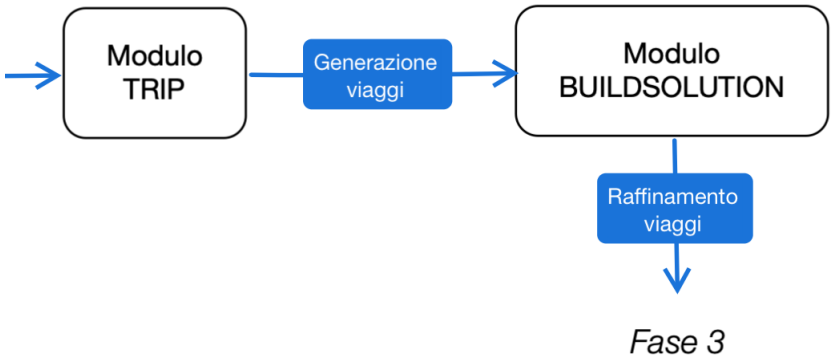


Figura 3: Fase 2

In questa sezione parliamo della generazione vera e propria dei viaggi. La Figura 3 mostra i moduli coinvolti in questa fase e sintetizza i due concetti principali di qui stiamo per parlare:

- Come avviene la generazione dei possibili viaggi
- Come devono essere raffinati questi viaggi in modo da penalizzarle alcuni e avvantaggiarne altri

Un punto molto importante da notare è che, seppur molta della conoscenza dell'esperto del dominio sia stata utilizzata nella fase precedente, il numero possibile di soluzioni è tendenzialmente ancora molto alto e quindi vedremo come, anche in questa parte, la generazione degli itinerari sia guidata dalla dall'uso di alcuni dati empirici.

3.2.1 Generazione viaggi

In questo paragrafo parliamo della creazione dei possibili viaggi da mostrare all'utente. Come abbiamo spiegato nel Paragrafo 2.1.2 un viaggio è composto da diversi multislot ed ogni slot all'interno dei vari multislot serve per contenere le informazioni di una certa meta del viaggio, quindi ad esempio nel momento in cui inseriremo un nuovo resort nel viaggio inseriremo anche:

- La città di provenienza
- La *certainty*
- I giorni in cui si pernotta nel resort
- Il prezzo totale che si deve spendere per rimanere un certo numero di giorni in un resort

La generazione dei viaggi avviene nel modulo TRIP mediante l'utilizzo di 5 regole:

1. **TRIP::first-city**

   Fino a quando esiste un resort, tra quelli selezionati nei passi precedenti, cui non abbiamo ancora creato un viaggio, produciamo un nuovo viaggio che parte proprio da quel resort.

2. **TRIP::next-city**

   Fino a quando il numero di luoghi da visitare non è uguale al numero di luoghi già inseriti nel viaggio, viene richiamata questa regola che aggiunge un nuovo resort al viaggio.

3. **TRIP::distribute-days-fairly**

   Una volta inserito il numero totale di resort del viaggio viene chiamata questa regola che distribuisce i giorni a disposizione per la vacanza nella maniera più equa possibile.

4. **TRIP::add-remaining-days**

   Se, in seguito alla distribuzione equa, sono ancora rimasti dei giorni disponibili creo un nuovo viaggio per ogni resort in cui posso aggiungere i giorni rimanenti (e.g., se il viaggio deve durare 10 giorni, la suddivisione equa su tre resort è <3,3,3>, allora vengono creati tre nuovi viaggio con tre distribuzioni diverse dei giorni rimanenti: <4,3,3>,<3,4,3>,<3,3,4>).

5. **TRIP::cleanup**

   L'ultima regola elimina i viaggi che sono formati da un numero inferiore di città rispetto a quelle richieste e che superano nel costo totale il budget definito dall'utente.

Queste regole si basano tutte più o meno sugli stessi concetti, vediamo come esempio la regola 3.

```
(defrule TRIP::distribute-days-fairly
    (travel-benchmark (name travel-duration) (value ?duration))
    ?p <- (trip (place-sequence $?cities) (price-per-night $?
    price-per-night) (days-distribution $?days-distribution))

    (travel-benchmark (name number-of-place) (value ?k&~unknown))
    (test (eq (length$ ?cities) ?k))
    ;Non deve essere gia' stata calcolata la distribuzione dei giorni
    (test (eq (length$ ?days-distribution) 0))

    =>

    (bind ?city-count (length$ ?cities))

    (bind ?days-distribution (create$))
    (loop-for-count (?cnt1 1 ?city-count)
        ; Distribuisco i giorni uniformemente
        (bind ?days-distribution (insert$ ?days-distribution 1 (div ?
    duration ?city-count)))
        ; Aggiorno i prezzi per notte in base ai giorni appena distribuiti
        (bind ?price-per-night (replace$
                        ?price-per-night ?cnt1 ?cnt1
                        (* (nth$ ?cnt1 ?price-per-night) (nth$ ?cnt1 ?
    days-distribution)))
        )
    )

    (duplicate ?p (price-per-night ?price-per-night) (days-distribution ?
    days-distribution))
    (retract ?p)
```

```
27 )
```

Codice 9: Distribuzione equa dei giorni

La regola 3, mostrata nel Codice 9, distribuisce i giorni disponibili per il viaggio in maniera equa tra le tappe che si devono fare. Nello specifico vengono eseguite le seguenti operazioni:

- **Riga 6-7**. La regola deve essere eseguita su un viaggio formato dal numero di città richieste ed un viaggio in cui non sono ancora stati distribuiti i giorni.
- **Riga 15**. Per ogni città del viaggio:
  - **Riga 17**. Inserisco, nel multislot dedicato alla distribuzione dei giorni, il numero di giorni da passare in quella città, in modo che ogni tappa possa usufruire dello stesso numero di giorni (n.b., viene eseguita una divisione intera tra la durata del viaggio e il numero di luoghi da visitare).
  - **Riga 19**. Viene aggiornato il prezzo del pernottamento nel resort associato alla città che stiamo considerando. Semplicemente viene moltiplicato il numero di notti in cui si rimane per il costo di una notte nell'albergo.
- **Riga 25-26**. Viene creato un viaggio con i nuovi valori aggiornati e viene eliminato il vecchio.

3.2.2 Raffinamento viaggi

Una volta terminato il modulo TRIP abbiamo un insieme di viaggi che rispettano le richieste dell'utente. Al fine di completare la procedura e poter mostrare i viaggi all'utente servono ancora alcune modifiche:

- Sino ad ora abbiamo associato le *certainty* solamente ai resort pertanto è il momento di stabilire quale sia la *certainty* globale di un certo viaggio.
- Dobbiamo penalizzare i viaggi che hanno delle tappe poste a distanza maggiore di 100km.

```
1 (defrule BUILDSOLUTION::find-certainty
2     ?p <- (trip (certainties $?certainties) (days-distribution $?
    days-distribution))
```

```
    (test (> (length$ ?certainties) 1))
    =>
    (bind ?new-cf 0)
    (loop-for-count (?i 1 (length$ ?certainties))
        (bind ?day (nth$ ?i ?days-distribution))
        (bind ?cf (nth$ ?i ?certainties))
        (bind ?new-cf (+ ?new-cf (* ?day ?cf)))
    )
    (bind ?trip-cf (/ ?new-cf (+ (expand$ ?days-distribution))))
    (if (> ?trip-cf 0.2) then (modify ?p (certainties ?trip-cf)) else (
    retract ?p))
)
```

Codice 10: Determinare la *certainty* di un viaggio

Il Codice 10 mostra come apportare la prima modifica. Per ogni viaggio in cui non è stata ancora calcolata la *certainty* finale viene eseguito il corpo della regola che calcola una media delle certainty dei resort pesata sul numero di giorni in cui si rimane nel resort. Se la *certainty* ottenuta è inferiore a 0.2 si elimina il viaggio altrimenti si modifica la *certainty* del viaggio indicandone il valore finale.

La seconda modifica, invece, è stata modellata in modo che potesse rispettare il più possibile i vincoli reali determinati dalle città di provenienza dei resort. Al fine di fare in maniera verosimile gli spostamenti, abbiamo utilizzato le coordinate geografiche di ogni città e con esse abbiamo calcolato la distanza usando la formula dell'emisenoverso [1] che permette appunto di determinare la distanza tra due luoghi utilizzando solamente longitudine e latitudine. Se, svolgendo questi calcoli, si trovano degli spostamenti maggiori di 100km, allora, si va a penalizzare la *certainty* di quel viaggio rendendolo meno plausibile.

3.3 Terza fase

L'ultima fase è davvero molto semplice ed il suo unico scopo è selezionare il migliore viaggio per ogni sottoinsieme di città su cui sono stati creati.

```
(defrule SELECTRESULT::pick-best-trip
    (declare (salience 10))
    ?trip <- (trip (certainties ?certainty) (place-sequence $?
    place-sequence1))
    (trip (place-sequence $?place-sequence2) (certainties ?certainty2&:(>
    ?certainty2 ?certainty)))
    (test (subsetp $?place-sequence1 $?place-sequence2))
  =>
```

```
7     (retract ?trip)
8  )
9
10 (defrule SELECTRESULT::random-choose-equal-trips
11     (declare (salience 5))
12    ?trip <- (trip (certainties ?certainty) (place-sequence $?
      place-sequence1))
13     ?trip2 <-(trip (place-sequence $?place-sequence2) (certainties ?
      certainty2&:(= ?certainty2 ?certainty)))
14     (test (subsetp $?place-sequence1 $?place-sequence2))
15     (test (neq ?trip $?trip2))
16   =>
17    (retract ?trip)
18 )
```

Codice 11: Selezione dei migliori itinerari

Nel corso della computazione possono essere stati creati viaggi diversi basati sulle stesse città (e.g., viaggio1: napoli, roma, torino | viaggio2: torino, napoli, roma) e pertanto, come mostrato nel Codice 11, andiamo ora a selezionare, tra tutti questi, quello con la *certainty* più alta eliminando tutti gli altri (regola *pick-best-trip*), se invece tali viaggi hanno una *certainty* identica allora, attraverso la regola *random-choose-equal-trips*, ne scegliamo uno in maniera casuale.

4 Applicazione mobile

Terminato lo sviluppo del sistema, ci siamo domandati in che modo fosse possibile creare un'interfaccia grafica in grado di mostrare i risultati dell'esecuzione del programma, sia per non dover rispondere da terminale alle domande sui requisti sia per visualizzare in una maniera semplice e lineare i risultati della computazione.

Ricercando in rete abbiamo notato che CLIPS è integrabile all'interno di progetti per applicazioni mobile che sfruttano il sistema operativo iOS e pertanto abbiamo deciso di creare un'applicazione in grado di mostrare le opzioni di preferenza dell'utente ed i viaggi creati dal sistema.

Le applicazioni iOS possono essere scritte in Objective-C e per tale motivo possono facilmente integrarsi con dei framework scritti in C. Sfruttando questa caratteristica abbiamo così potuto sviluppare alcune classi in grado di fare da ponte tra il framework di CLIPS ed il codice vero e proprio di iOS, garantendoci di poter sfruttare al massimo sia le funzionalità di CLIPS che le potenzialità di iOS.

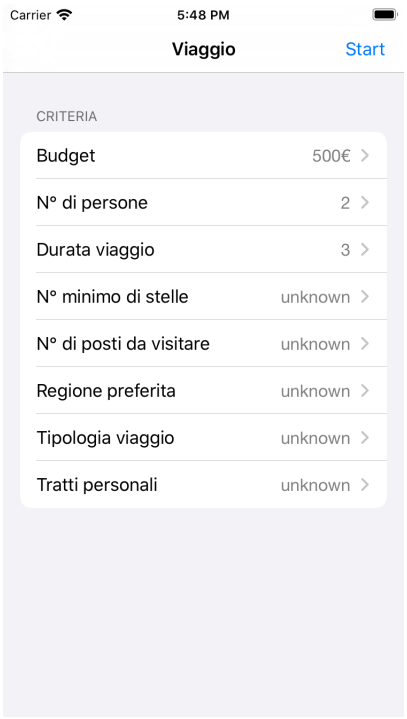
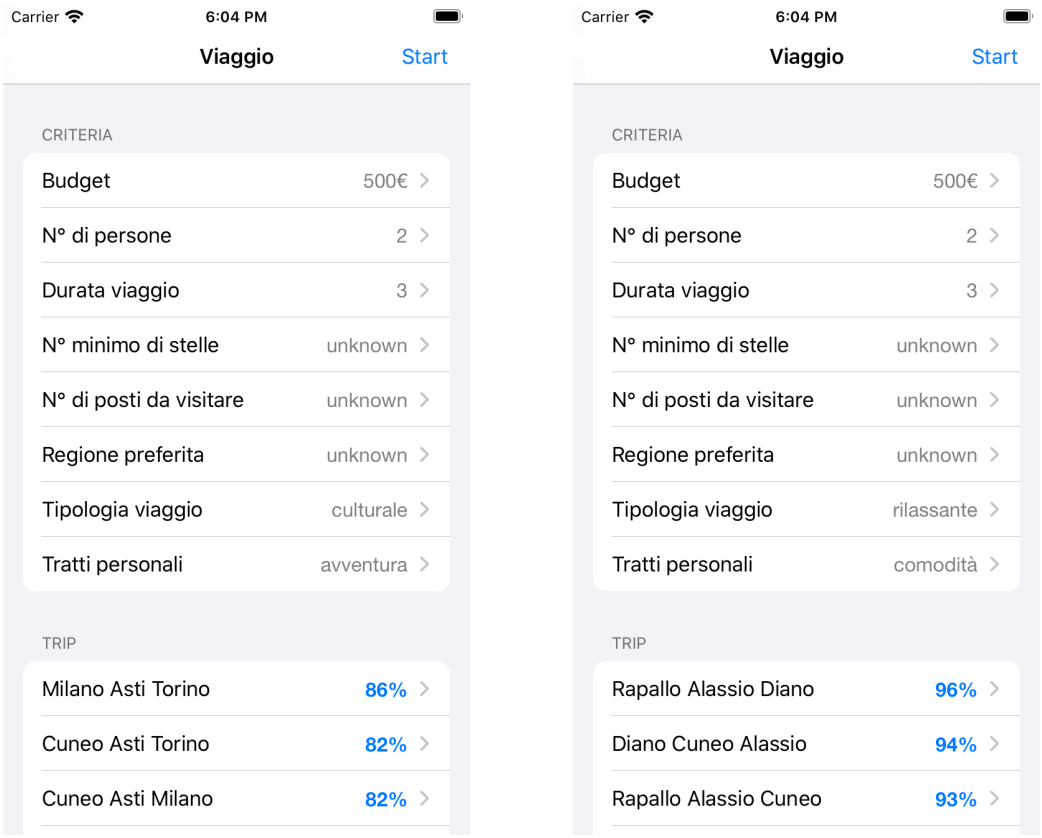


Figura 4: Interfaccia principale dell'applicazione

La Figura 4 mostra l'interfaccia di avvio dell'applicazione ed in essa è possibile scegliere i valori per gli 8 paramatri che guidano la creazione dei viaggi. Un utente che approccia l'applicazione può dunque scegliere liberamente quali valori assegnare alle diverse opzioni (n.b., solamente i primi tre parametri sono obbligatori) e, nel momento in cui ha deciso, può premere il pulsante di "start", posto in alto a destra, ed avviare il sistema esperto che andrà a calcolare i viaggi e glieli mostrerà.

4.1 Test

In conclusione, vediamo alcuni esempi di esecuzione attraverso l'interfaccia grafica così da poter valutare l'esito della computazione.



(a) (b)

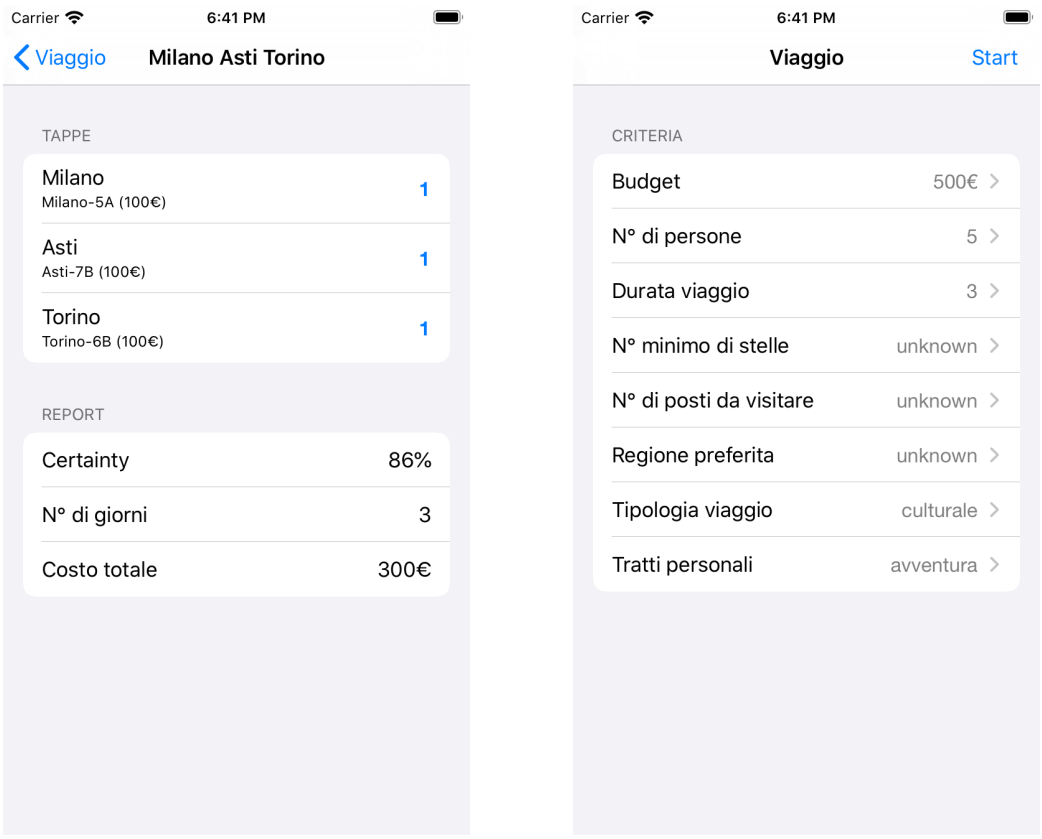
Figura 5: Esempio di due esecuzioni: (a) Utilizza gli attributi "culturale" e "avventura". (b) Utilizza gli attributi "rilassante" e "comodità".

La Figura 5 mostra due possibili esiti della computazione:

- La Figura 5a mostra i risultati che si ottengono impostando gli attributi "culturale" ed "avventura" e come vediamo, i viaggi calcolati sono formati da delle città che tendenzialmente rispettano tali caratteristiche.
- La Figura 5b mostra i risultati che si ottengono impostando gli attributi "rilassante" e "comodità" e come vediamo, i viaggi mostrati sono tipicamente legati a località di mare in cui è più facile scorrere momenti tranquilli e rilassanti.

La doverosa precisazione da fare è che questi risultati sono frutto di come sono state create le regole dell'esperto del dominio e si basano su un dataset limitato di città (rapallo, alassio, diano, cuneo, torino, asti, milano), pertanto la correttezza delle opzioni restituite è limitata sia dall'esigua

dimensione del dataset sia dalla complessa calibrazione dei parametri delle regole dell'esperto.



(a) (b)

Figura 6: (a) Esempio di dettaglio di un viaggio. (b) Esempio di esecuzione senza risultati.

La Figura 6a mostra il dettaglio di un viaggio formato dalle diverse tappe e, come si può notare, ognuna ha un certo resort (e.g., Milano-5A) con un certo costo basato sul numero di giorni in cui si pernotta nell'albergo (e.g., 100€). Nella parte sottostante vi è un report che contiene il riassunto del viaggio ed in particolare specifica le seguenti informazioni: incertezza della proposta, numero totale di giorni del viaggio e costo totale.

La Figura 6b mostra semplicemente che, qualora i vincoli non permettano di trovare alcuna soluzione (e.g., in Figura 6b sono state impostate 5 persone lasciando un budget di 500€), non viene mostrata all'utente alcuna proposta.

5 Considerazioni finali

Questa relazione ha voluto racchiudere le informazioni principali che ci hanno permesso di completare correttamente il progetto. Il sistema esperto sviluppato ci ha dato modo di indagare in maniera approfondita le caratteristiche di un approccio a regole consentendoci di consolidare le competenze teoriche acquisite a lezione.

In conclusione, possiamo ritenerci soddisfatti del programma sviluppato e delle soluzioni ottenute, che sono risultate conformi ai requisti progettuali.

Elenco delle figure

Listings

Riferimenti bibliografici

[1] `https://en.wikipedia.org/wiki/Haversine_formula`