Artificial Intelligence & Cybersecurity
Automated Reasoning
A.A 2023-24

Relazione progetto

Salvatore Francesco Riefolo
139970

October 15, 2023

Contents

1 Il problema: PNRR

Il PNRR è un piano nazionale di investimenti, e l'obiettivo del problema è posto è determinare quali richieste effettuate dalle città eligibili al fondo possono essere approvate, in base a dei criteri definiti attraverso dei vincoli.

Scelta una regione od un suo sottinsieme, i comuni eligibili sono quelli con più di 10.000 abitanti. Si determinano inoltre per ogni comune il suo numero di abitanti e se questo è connesso con altri comuni della regione.

Il PNRR consiste in 10 milioni di Euro complessivo per 5 tipologie di interventi: scuole, strade, energie rinnovabili, parchi, sistemazione di torrenti/fiumi. Almeno 1 milione per tipologia va erogato: la soglia minima è quindi di 5 milioni. Ogni comune può effettuare al massimo una richiesta per tipologia di intervento.

Si vuole allocare al meglio la quota di 10 Milioni in modo tale che:

- Se due comuni sono collegati nel grafo l'intervento sulle strade sarà finanziato al massimo a uno dei due.
- Le richieste possono essere approvate come sono o ridotte del 10%, del 20% o del 30%.
- Ad ogni comune sia approvato almeno un progetto.
- I 10 milioni vanno usati il più possibile.
- Per scuole, energie rinnovabili e parchi, si bocciano i progetti troppo esosi. Questi sono i vincoli da definire per l'approvazione delle richieste.
- Minimizza la somma di richieste non erogate (ovvero la cifra dei progetti bocciati e le cifre ridotte del 10, 20, 30%).

1.1 Assunzioni

La regione presa in considerazione è l'Umbria, con 19 città aventi più di 10.000 abitanti.

Dopo aver esaminato la consegna, sono stati definiti i seguenti requisiti:

- Ogni città richiede almeno un intervento.
- Ad ogni città viene approvato almeno un intervento
- La somma delle richieste per una tipologia di intervento deve essere almeno di 1 milione.
- Le cifre minime e massime per la somma degli interventi sono rispettivamente 5 milioni e 10 milioni.
- Se due comuni sono collegati ed entrambi richiedono un intervento per le strade, ad uno dei due viene approvato e all'altro viene bocciato.
- Si rigettano le richieste esose. Sono stati definiti così i progetti "troppo costosi" per ogni tipologia (escluse le strade, per cui esiste già un vincolo):
    1. **Scuole**: data la media di tutte le richieste per tutte le tipologie di intervento ed il rapporto tra la media e gli abitanti per una città, la richiesta per le scuole per quella città è approvata se il costo richiesto moltiplicato per il rapporto tra media ed abitanti è minore di due volte la media.

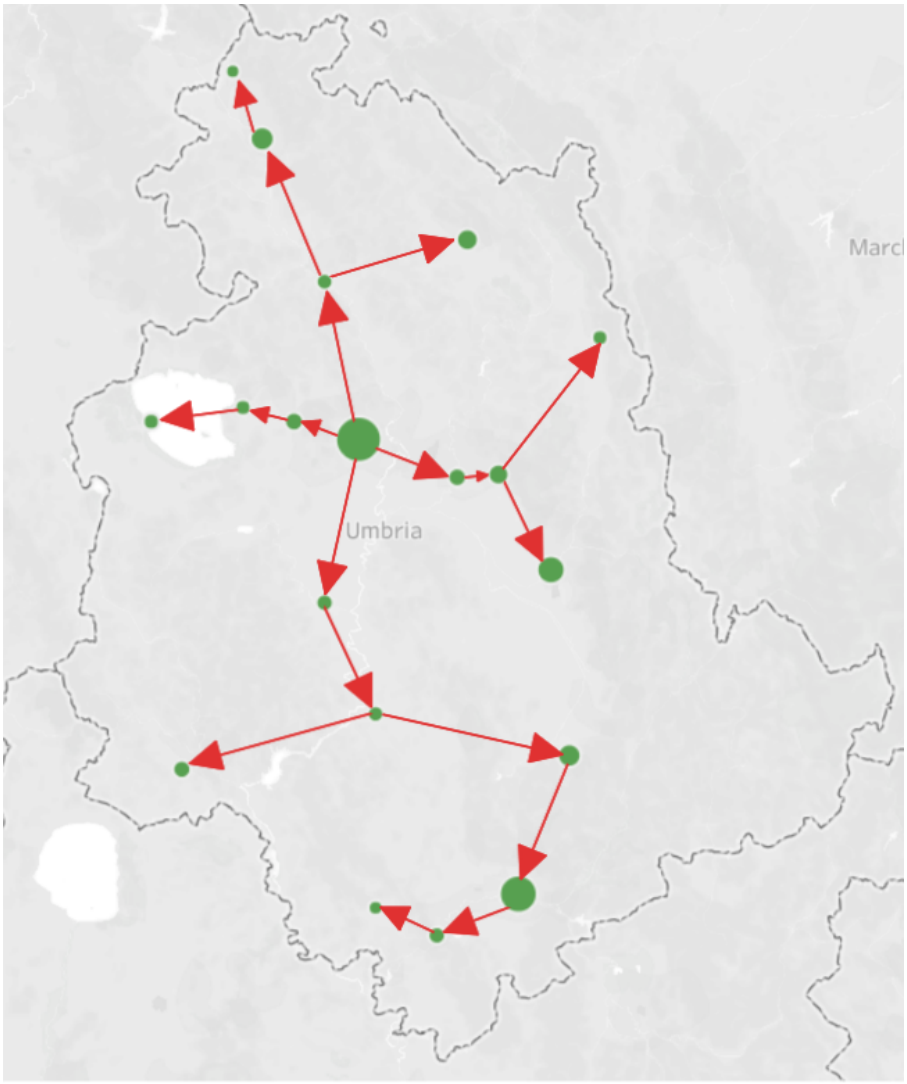

Figure 1: Città della regione Umbria con più di 10.000 abitanti, connesse attraverso il percorso minimo tra l'una e l'altra.

2. **Parchi**: ogni città può richiedere al massimo 10.000.000 diviso il numero di comuni.
3. **Energie rinnovabili**: la richiesta moltiplicata per 2 deve essere minore alla somma delle richieste per gli altri interventi per una città.

Inoltre sono state prese le seguenti scelte riguardo ad alcuni punti:

- Il grafo delle strade che connettono le città non presenta cicli: due città possono essere connesse da uno ed un solo percorso. Ciò risolve un potenziale problema per la determinazione del vincolo per gli interventi sulle strade, che occorre quando un numero dispari di città in un ciclo effettua una richiesta per tale intervento.
- Le richieste vengono approvate o rigettate prima di applicare la riduzione. Se una richiesta è troppo alta quando viene presentata viene cassata subito.
- Le richieste sono effettuate in decine di migliaia di euro. Ad esempio, una richiesta di "30" equivale ad una richiesta di 300.000 euro.
- Gli abitanti sono espressi in migliaia.

Entrambe le scelte sono state prese per al fine di ridurre il dominio delle richieste e ridurre lo spazio di ricerca. Queste scelte sono state prese dopo aver concluso l'implementazione ed averla testata sui benchmark preparati, avendo notato che le istanze più piccole richiedevano minuti per trovare la prima soluzione.

Per quanto riguarda le riduzioni, queste trovano utilizzo quando la somma delle richieste supera i 10.000.000 e le richieste rifiutate non portano il totale delle richieste approvate sotto questa soglia. Si assume quindi che la somma delle richieste possa superare i 10.000.000.

1.2 Obiettivi

Ci si vuole assicurare che:

- Per ogni città viene approvato almeno un intervento.
- La spesa minima per intervento è 1.000.000.
- La spesa minima totale è quindi di 5.000.000.
- La spesa massima è di 10.000.000.

Come obiettivo, ci si pone di:

- Massimizzare la spesa.
- Minimizzare il numero di richieste non approvate o ridotte, o equivalentemente massimizzare il numero di richieste pienamente approvate.

2 Implementazione

2.1 Minizinc

Per scrivere ed eseguire il modello Minizinc è stata usata la versione 2.7.6 del software. Il programma è stato lanciato usando il comando

```
minizinc --solver Chuffed -f --time-limit 300000 --output-time /
pnrr.mzn benchmarks/TEST.dzn
```

dove `TEST` è il nome dell'istanza. Si noti che è stato usato il solver Chuffed per Minizinc. Questo solver risulta essere più rapido, seppur di poco, rispetto a Gecode.

Il programma è diviso in due files: il file `pnrr.mzn` contiene le costanti e le definizioni dei dati usati nel modello, mentre i file `smallN.dzn`, `mediumN.dzn` e `largeN.dzn`, con `N = 0..10` contengono i valori di input creati dinamicamente.

Le enumerazioni `INTERVENTI` e `COMUNI` contengono rispettivamente le tipologie di intervento richiedibili dai comuni ed i nomi di questi. `INTERVENTI` è statica, in quanto gli interventi richiedibili non variano rispetto agli input, mentre i comuni che richiedono gli interventi non sono sempre gli stessi: la complessità dell'istanza è quindi legata al numero di comuni che effettuano le richieste.

```
enum INTERVENTI = {strade, fiumi, scuole, parchi, energia};
enum COMUNI;
```

La variabile NUMERO_COMUNI, dinamica anch'essa indica il numero di comuni dell'istanza, mentre le costanti SPESA_MINIMA_INTERVENTO, SPESA_MASSIMA rappresentano i limiti per la somma delle richieste.

```
var int: NUMERO_COMUNI;
int: SPESA_MINIMA_INTERVENTO = 1000;
int: SPESA_MASSIMA = 10000;
```

Il vettore ABITANTI_PER_COMUNE rappresenta per ciascun comune il suo numero di abitanti, ed il vettore bidimensionale STRADE_TRA_COMUNI contiene valori booleani che indicano se una strada è presente tra due comuni (true) o no (false). Si noti che, per come vengono generati i dati, la diagonale ha sempre valore false: un comune non ha una strada che lo connette a se stesso.

```
array[COMUNI] of var int: ABITANTI_PER_COMUNE;
array[COMUNI, COMUNI] of var bool: STRADE_TRA_COMUNI;
```

I vettori bidimensionali RICHIESTE e RICHIESTE_APPROVATE rappresentano rispettivamente gli interventi richiesti da ogni comune e gli interventi che sono stati approvati per ogni comune. Il valore delle richieste varia tra 0 (intervento non richiesto o rigettato) ed il valore massimo per la somma delle richieste.

```
array[COMUNI, INTERVENTI]
    of var 0..SPESA_MASSIMA: RICHIESTE;

array[COMUNI, INTERVENTI]
    of var 0..SPESA_MASSIMA: RICHIESTE_APPROVATE;
```

Seguono una serie di variabili utilizzate dai vari vincoli per definire se una richiesta è accettabile o meno.

Minizinc permette di definire un solo obiettivo: per questa ragione, si è deciso di massimizzare la somma. Sarebbe possibile esprimere più obiettivi definendo una variabile che racchiude il significato di questi, ma per come è stata strutturata l'implementazione ed avendo a che fare con due unità di misura diverse (soldi per i valori delle richieste, un conteggio per le richieste cassate) si è optato per dichiarare un obiettivo unico.

2.1.1 Input

I file di input contengono i valori delle variabili definite nel file principale. In particolare:

COMUNI contiene i comuni che effettuano richieste, sottoinsieme delle città della regione, NUMERO_COMUNI, il numero di questi, e ABITANTI_PER_COMUNE i loro residenti.

```
COMUNI = {perugia, terni, ...};
NUMERO_COMUNI = 19;
ABITANTI_PER_COMUNE = [161, 106, ...];
```

STRADE_TRA_COMUNI è una matrice contenente valori booleani che rappresentano se due comuni sono connessi. L'ordine dei comuni è quello specificato dall'enumerazione COMUNI.

```
TRADE_TRA_COMUNI = [|
    false, true, ...|
    true, false, ...|
```

```
    ...
|];
```

Infine `RICHIESTE` è una matrice contenente interi rappresentanti le richieste per intervento: ogni riga rappresenta i 5 interventi che un comune può richiedere.

```
RICHIESTE = [|
    1, 2, 0, 10, 13 |
    21, 3, 16, 4, 13 |
    ...
|];
```

2.1.2 Output

Vengono stampati in output le richieste, le richieste approvate, la somma totale dopo la scalatura ed il totale approvato per intervento.

2.2 Answer Set Programming

Per eseguire il modello ASP è stato usato Clingo alla versione 5.6.2. Il programma è stato lanciato usando il comando

```
clingo --outf=1 --time-limit=300 pnrr.lp common.lp benchmarks/TEST.lp
```

dove `TEST` è il nome dell'istanza.

Il programma è diviso in 3 files: `pnrr.lp` contiene le definizioni dei vincoli, predicati necessari per valutare i vincoli e obiettivi; `common.lp` contiene i predicati comuni ad ogni istanza; `inputN.lp` contiene l'istanza stessa.

Vengono definiti i predicati `intervento(I)` che identifica i 5 interventi richiedibili, `abitanti(Citta, N)` che rappresenta il numero di abitanti di una città, `connesse(C1, C2)` che rappresenta la connessione di due città mediante una strada, e `moltiplicatore(M)` che rappresenta il fattore per cui una richiesta può essere scalata dopo essere stata approvata.

```
intervento(strade).
...
abitanti(perugia,161).
...
connesse(perugia,corciano).
...
moltiplicatore(1).
moltiplicatore(0,9).
moltiplicatore(0,8).
moltiplicatore(0,7).
```

Vengono inoltre definite le costanti `richiesta_minima_intervento`, `richiesta_massima` rappresentanti i limiti per la somma delle richieste.

```
#const richiesta_minima_intervento = 100.
#const richiesta_massima = 1000.
```

I vincoli assicurano che i predicati rappresentanti le richieste approvate, definiti come `approvata(Citta, Intervento, Richiesta, Moltiplicatore)`, seguano le regole sopra definite. In particolare, un vincolo di cardinalità assicura che per ogni richiesta possa essere approvata al più una volta.

```
0 {approvata(C, I, R, M): richiesta(C, I, R)} 1
    :- citta(C), intervento(I), moltiplicatore(M).
```

Si definisce l'impossibilità dell'approvazione di una richiesta che viola i vincoli generali di costo e degli interventi attraverso le regole sopra definite, utilizzando delle negazioni (regole con testa vuota).

Si vuole massimizzare la somma delle richieste per gli interventi approvati, scalati per il moltiplicatore associato, e minimizzare il numero di richieste non approvate e scalate: per fare ciò, si calcola il numero di richieste non approvate sottraendo al numero di richieste totali quello delle richieste approvate; viene sommato il numero di richieste ridotte, ottenuto calcolando il numero di richieste approvate con moltiplicatore diverso da 1.

2.2.1 Input

I file di input contengono una lista di predicati `citta(C)`, rappresentanti le città da considerare nell'istanza, ed una lista di predicati `richiesta(Citta, Intervento, Richiesta)`, le richieste per ogni città. Diversamente dal modello Minizinc, non è necessario definire dinamicamente le strade tra i comuni ed il numero degli abitanti di questi, in quanto i predicati la cui componente `Citta` non è specificata nell'input vengono rimossi.

```
citta(perugia).
...
richiesta(perugia, strade, 11).
richiesta(perugia, fiumi, 13).
...
```

2.2.2 Output

Vengono dati in output i predicati rappresentanti le richieste approvate, il totale delle richieste effettuate ed il totale delle richieste approvate.

3 Benchmark e risultati

Per testare i modelli, sono stati generati 30 benchmarks:

- 10 benchmark (istanze "piccole") contengono richieste da parte di 3 comuni (15 richieste totali).
- 10 benchmark (istanze "medie") contengono richieste da parte di 8 comuni (40 richieste totali).
- 10 benchmark (istanze "piccole") contengono richieste da parte di tutt i 19 comuni (95 richieste totali).

3.1 Generazione dei benchmark

La generazione degli input avviene tramite script Python: questo contiene la lista completa delle città, delle strade tra di esse e dei loro abitanti. Lo script, in base a dei flag, seleziona un sottoinsieme di comuni e genera casualmente le richieste per essi. È possibile specificare a quanto le richieste si sommano, se includere zeri nelle richieste e quanti comuni selezionare.

L'output della generazione è formattato in formato `.dzn` o `.lp`, in modo da semplificare la creazione degli input attraverso uno script unico.

Per generare una nuova batteria di test, si usi il comando `python3 benchmark-generator.py --full` dopo essersi spostati nella cartella `src/scripts`. Lo script può essere invocato con diversi flag per generare test specifici.

3.2 Risultati

I vincoli come sono stati definiti risultano un po' troppo stringenti: in particolare, il vincolo che impone che la spesa minima per tipologia di intervento sia maggiore di 1.000.000 causa porta a terminare la quasi totalità dei test con "INCONSISTENT", poichè gli altri vincoli spesso portano ad non accettare le richieste più costose facendo scendere il totale per intervento sotto questa soglia e causando dunque predicati contrastanti. Rilassando i vincoli sugli interventi o il vincolo sulla soglia si ottengono risultati migliori e più facilmente comparabili. Per semplicità, i risultati sotto proposti fanno riferimento ad i test lanciati con il vincolo sulla spesa minima per tipologia di intervento rimosso. Un diverso generatore di dati, che distribuisce il totale in modo tale da rendere i valori per tipologia di intervento simili alla loro soglia, potrebbe alternativamente fornire risultati migliori.

Si è cercato di proporre la stessa implementazione nei due linguaggi diversi. L'implementazione Minizinc risulta notevolmente più rapida, riusciendo a risolvere le istanze più grandi in qualche secondo. L'implementazione ASP risulta altrettanto rapida sui modelli più piccoli ma raggiunge il timeout, seppur trovando delle soluzioni non ottimali, su quelle medie e grandi. Per questo motivo, la dimensione dei test medi per ASP è stata ridotta da 40 richieste a 30 richieste.

	Dimensione test	N. Richieste	Tempo medio
ASP	Small	15	0.453s
	Medium	30	141s
	Large	95	Timeout
Minizinc	Small	15	0.05s
	Medium	40	0.06s
	Large	95	0.10s

Figure 2: Risultati ottenuti dall'esecuzione dei modelli ASP e Minizinc.

Si ipotizza che con una riformulazione dei vincoli nel modello ASP si possano ottenere tempi inferiori, restringendo il dominio degli interi del predicato `approvata/4`.