Ottimizzazione Garanti Accademici

Manuel Di Agostino *Università degli studi di Parma*
Parma, Italia
manuel.diagostino@studenti.unipr.it
Leonardo Ongari *Università degli studi di Parma*
Cremona, Italia
leonardo.ongari@studenti.unipr.it

***Sommario*—La gestione dell'assegnazione dei docenti di riferimento o garanti in ambito universitario è un compito complesso che coinvolge vari fattori organizzativi e un gran numero di dati. L'automazione di questo processo può semplificare notevolmente il lavoro amministrativo e migliorare l'efficienza operativa delle università. È stato sviluppato un sistema basato su *Answer Set Programming* (ASP), un paradigma logico adatto alla risoluzione di problemi combinatori complessi. Utilizzando `Python` e `Clingo`, un potente solver ASP, è stata progettata una soluzione capace di identificare automaticamente i docenti di riferimento, a partire da un insieme di dati specifici. Il sistema sviluppato è descritto nei suoi dettagli tecnici, con un focus sulla modellazione logica, l'implementazione e l'analisi della complessità computazionale. I risultati ottenuti sono discussi insieme alle potenzialità di estensione del sistema.**

***Keywords*—Answer Set Programming, Ottimizzazione, Programmazione Dichiarativa, Clingo**

I. Introduzione

L'identificazione dei docenti di riferimento o garanti per uno specifico corso di studi rappresenta un processo che annualmente coinvolge le università italiane. Tale compito può risultare complesso, specialmente in presenza di strutture organizzative articolate e di grandi volumi di dati. La necessità di automatizzare e ottimizzare questa ricerca è quindi cruciale per migliorare l'efficienza delle attività amministrative e accademiche.

In questo progetto, il problema è stato affrontato utilizzando *Answer Set Programming* (ASP), un paradigma di programmazione logica dichiarativa particolarmente adatto alla risoluzione di problemi combinatori complessi. L'implementazione è stata realizzata con l'ausilio di `Python` [1] e `Clingo` [2], un solver open source ASP che combina il modello di programmazione logica con strumenti di ottimizzazione efficienti.

L'obiettivo principale è stato quello di progettare e implementare un sistema che, a partire da un insieme di dati resi disponibili dagli uffici di competenza, consenta di individuare in maniera automatica i docenti di riferimento o garanti in base a criteri specifici. La relazione descrive le fasi del lavoro, dall'analisi dei requisiti del problema alla modellazione logica, presentando i dettagli dell'implementazione concreta e un'analisi sulla sua complessità computazionale. Infine, vengono discussi i risultati ottenuti e le possibili estensioni del progetto.

II. Background

A. Motivazioni

Attualmente, l'Università di Parma gestisce l'assegnazione dei docenti in modo manuale, affrontando il processo in maniera incrementale per ciascun corso di laurea. Il personale incaricato impiega settimane per ottenere una versione soddisfacente della distribuzione, basandosi frequentemente su preferenze informali e criteri non documentati. Questo approccio risulta poco flessibile e difficilmente adattabile a nuove esigenze. Inoltre, presenta significative limitazioni, come la difficoltà nel gestire situazioni complesse e l'incapacità di ottimizzare il processo in tempo reale, rendendo il sistema poco efficiente e reattivo ai cambiamenti.

B. Answer Set Programming

L'Answer Set Programming (ASP) è un paradigma di programmazione logica dichiarativa, particolarmente adatto per risolvere problemi complessi di natura combinatoria che richiedono soluzioni flessibili e ottimizzate. A differenza della programmazione imperativa tradizionale, ASP si concentra sulla descrizione del *cosa* deve essere risolto piuttosto che su *come* farlo, utilizzando una forma di logica che rappresenta le conoscenze del problema e le sue restrizioni (*vincoli*).

ASP si basa sulla teoria degli *answer sets*, ossia un insieme di atomi letterali consistenti con le regole e i fatti che costituiscono il programma. Il compito del solver ASP è quello di trovare gli insiemi di valori che risolvono il sistema di equazioni logiche, fornendo così soluzioni ottimali o ammissibili.

L'uso di questo paradigma di programmazione è particolarmente indicato in ambiti in cui sono presenti vincoli complessi, preferenze multiple e soluzioni che devono rispettare determinati criteri. ASP permette di modellare in modo naturale problemi che coinvolgono l'ottimizzazione, la pianificazione, e la ricerca di soluzioni in scenari combinatori, in cui le variabili e le relazioni tra esse sono numerose e intricate.

C. Il solver

`Clingo` [2] è un solver open-source per ASP, sviluppato dal gruppo Potassco. È uno degli strumenti più potenti e diffusi per risolvere problemi complessi di ottimizzazione e combinazione, combinando un motore di inferenza logica con capacità avanzate di ottimizzazione. Nel nostro progetto, è stato integrato direttamente in Python utilizzando le API Python ufficiali

[3], le quali consentono di interagire facilmente con il solver all'interno di ambienti Python. Questa integrazione permette di automatizzare il processo di invocazione e gestione delle soluzioni, facilitando l'elaborazione dei dati e l'ottimizzazione delle assegnazioni in tempo reale.

D. Provenienza e contenuto dei dati di input

I dati di input sono stati forniti dall'U.O. Progettazione Didattica e Assicurazione della Qualità [4], in collaborazione con il prof. A. Dal Palù dell'Università di Parma. Questi includono un insieme di tabelle e documenti eterogenei, comprendenti le coperture dei corsi per l'anno accademico corrente, l'elenco del personale docente e le informazioni relative alle immatricolazioni nei corsi di laurea. L'elaborazione è stata effettuata utilizzando `Python` e si è rivelata particolarmente complessa a causa dell'assenza di una sorgente dati unica e centralizzata.

III. Modellazione del problema

A. Esempio giocattolo

B. Strutturazione della soluzione

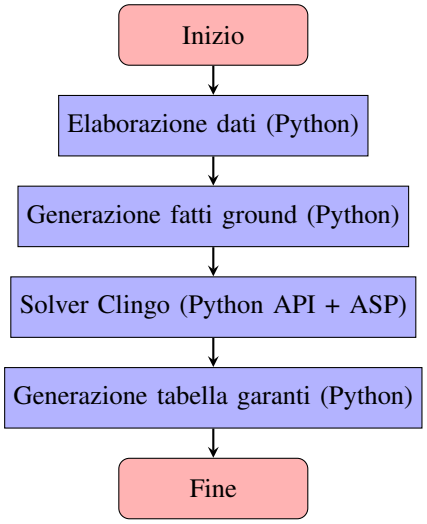

Figura 1. Pipeline del processo di assegnazione dei garanti.

In Fig. 1 è illustrata la pipeline di lavoro. Prima dell'elaborazione automatica dei dati, è stato necessario effettuarne una normalizzazione manuale. Questo passaggio ha permesso di uniformare i formati e risolvere eventuali incongruenze nei dati forniti, facilitando così le fasi successive. La generazione dei fatti ground costituisce la base di conoscenza positiva, utilizzata dal programma ASP principale per calcolare gli answer set. Successivamente, il solver Clingo viene eseguito tramite Python e i risultati ottenuti vengono convertiti in formato tabellare per facilitarne la visualizzazione.

C. Implementazione

L'implementazione è stata progettata separando la logica principale per la generazione dei garanti accademici da quella relativa alle preferenze specifiche. Tra le preferenze considerate figurano:

- assegnazione automatica del ruolo di garante al presidente del corso;
- massimizzazione del numero di docenti con contratto a tempo indeterminato;
- minimizzazione del numero di ricercatori e docenti a contratto coinvolti.

Nel Cod. 1 viene illustrato il codice di `main.lp`. Per questioni di ottimizzazione in fase di grounding si è preferito, ove possibile, utilizzare gli *aggregati* evitando così l'utilizzo della negazione esplicita.

```
% se garante allora insegna almeno una materia
1 {
    insegnamento(I) : insegna(docente(Matricola),
        insegnamento(I), corso(Corso))
} :-
    garante(docente(Matricola), corso(Corso)).

scelta(docente(Matricola), corso(Corso)) :-
        docente(Matricola),
        insegna(docente(Matricola), insegnamento(_)
            , corso(Corso)).

% numero garanti
M{
        garante(docente(Matricola), corso(Corso)) :
        scelta(docente(Matricola), corso(Corso))
}N :-
        min_garanti(M, corso(Corso)),
  max_garanti(N, corso(Corso)),
        corso(Corso),
        afferisce(corso(Corso), categoria_corso(
            Categoria)).

% almeno X garanti a tempo indeterminato
X{
        garante(docente(Matricola), corso(Corso)) :
        garante(docente(Matricola), corso(Corso)),
        indeterminato(docente(Matricola))
}M :-
        min_indeterminato(X, corso(Corso)),
        max_garanti(M, corso(Corso)),
        corso(Corso),
        afferisce(corso(Corso), categoria_corso(
            Categoria)).

% al piu' X garanti ricercatore
% (analogo al precedente, senza minimo)

% al piu' X garanti a contratto
% (analogo al precedente, senza minimo)

% un docente scelto al piu' una volta
1 {
    garante(docente(Matricola), corso(Corso2)) :
    corso(Corso2),
    afferisce(corso(Corso2), categoria_corso(
        Categoria2))
} 1 :-
    garante(docente(Matricola), corso(Corso1)),
    afferisce(corso(Corso1), categoria_corso(
        Categoria1)).

R {
    garante(docente(Matricola), corso(Corso)) :
    garante(docente(Matricola), corso(Corso)),
    afferisce(docente(Matricola), settore(Settore))
        ,
    di_riferimento(settore(Settore), corso(Corso))
} :-
    corso(Corso),
    min_riferimento(R, corso(Corso)).
```

```
#show garante/2.
```

Codice 1. Codice ASP del file `main.lp`.

Le preferenze (Cod. 2) sono state espresse tramite l'utilizzo di *weak constraint*. In particolare, sono stati definiti quattro livelli di priorità[1] sui quali intervenire:

- priorità 4: massimizzare l'assegnazione dei presidenti di corso come garanti;
- priorità 3: massimizzare il numero di corsi per i quali è soddisfatto il predicato `ideale/1`, dove i garanti sono esclusivamente docenti a contratto indeterminato;
- priorità 2: massimizzare l'impiego di docenti a contratto indeterminato come garanti;
- priorità 1: minimizzare il numero di corsi per i quali è soddisfatto il predicato `non_ideale/1`, i cui garanti comprendono almeno un ricercatore e/o un docente a contratto a tempo determinato.

```
#minimize{
        -1@4,
        Matricola :
                garante(docente(Matricola), corso(
                    Corso)),
                presidente(docente(Matricola),
                    corso(Corso))
}.

#minimize{
        -1@2,
        Matricola :
                garante(docente(Matricola), corso(
                    Corso)),
                indeterminato(docente(Matricola))
}.

ideale(corso(Corso)) :-
        N=#count{
                docente(Matricola) :
                garante(docente(Matricola), corso(
                    Corso)),
                indeterminato(docente(Matricola))
        },
        corso(Corso),
        max_garanti(Max, corso(Corso)),
        N=Max.

#minimize{
        -1@3,
        ideale(corso(Corso)) :
        ideale(corso(Corso))
}.

non_ideale(corso(Corso)) :-
        N=#count{
                docente(Matricola) :
                garante(docente(Matricola), corso(
                    Corso)),
                indeterminato(docente(Matricola))
        },
        corso(Corso),
        max_garanti(Max, corso(Corso)),
        N<Max.

#minimize{
        1@1,
        non_ideale(corso(Corso)) :
        non_ideale(corso(Corso))
}.
```

Codice 2. Codice ASP del file `preferenze.lp`.

aggiornare con versione definitiva

[1]Si ricorda che l'ordine di importanza è inverso rispetto al livello di priorità indicato.

D. Spazio delle soluzioni

Si analizza ora la complessità computazionale in termini di possibili configurazioni generate candidate ad essere soluzioni.

```
scelta(docente(Matricola), corso(Corso)) :-
        docente(Matricola),
        insegna(docente(Matricola), insegnamento(_)
            , corso(Corso)).

% possibili garanti
M{
        garante(docente(Matricola), corso(Corso)) :
        scelta(docente(Matricola), corso(Corso))
}N :-
        min_garanti(M, corso(Corso)),
        max_garanti(N, corso(Corso)),
        corso(Corso),
        afferisce(corso(Corso), categoria_corso(
            Categoria)).
```

Codice 3. Frammento del file `main.lp`.

Nel Cod. 3 è illustrata la regola che controlla i garanti generati. Per ciascun corso di studi (95 in totale), esiste una sola versione ground che rende veri i predicati presenti nel corpo della regola; questo significa che il corpo verrà attivato esattamente una volta. Inoltre `M` e `N`, rispettivamente minimo e massimo richiesti, coincidono; quindi l'aggregato presente in testa itera sulle possibili scelte per un determinato corso, costruendo insiemi di esattamente N elementi. Indicati con Δ il numero medio delle possibili `scelta` ottenute da ogni corso e con k il numero medio di garanti richiesti, si ottiene che questa regola genera circa

$$|Corsi| \cdot \binom{\Delta}{k} = 95 \cdot \frac{\Delta!}{k! \cdot (\Delta - k)!}$$

possibili answer set. In media risultano $\Delta = 39$ e $k = 10$, ossia circa

$$95 \cdot \frac{39!}{10! \cdot 29!} = 95 \cdot \frac{39 \cdot 38 \cdot \ldots \cdot 30}{10!} \approx 6 \cdot 10^{10}$$

risultati. Lo spazio delle alternative da considerare cresce dunque fattorialmente rispetto al numero medio di insegnanti per ciascun corso.

Limpiego di vincoli espressi tramite aggregati consente di escludere una significativa quantità di answer set non validi. Ad esempio, la seguente regola:

```
% un docente e' scelto al piu' una volta
0{
        garante(docente(Matricola), corso(Corso2))
            :
        garante(docente(Matricola), corso(Corso2)),
        Corso1!=Corso2
}0 :-
        garante(docente(Matricola), corso(Corso1)).
```

Codice 4. Frammento del file `main.lp`.

esclude tutti gli answer set in cui un docente risulta selezionato come garante per più di un corso.

Le regole riportate nel Cod. 5 stabiliscono invece limiti massimi e/o minimi per le sottocategorie di docenza, differenziate in base al tipo di contratto.

```
% almeno X garanti a tempo indeterminato
X{
        garante(docente(Matricola), corso(Corso)) :
        garante(docente(Matricola), corso(Corso)),
        indeterminato(docente(Matricola))
}M :-
        min_indeterminato(X, corso(Corso)),
        max_garanti(M, corso(Corso)),
        corso(Corso).

% al piu' X garanti ricercatore
{
        garante(docente(Matricola), corso(Corso)) :
        garante(docente(Matricola), corso(Corso)),
        ricercatore(docente(Matricola))
}X :-
        max_ricercatori(X, corso(Corso)),
        corso(Corso).

% al piu' X garanti a contratto
{
        garante(docente(Matricola), corso(Corso)) :
        garante(docente(Matricola), corso(Corso)),
        contratto(docente(Matricola))
}X :-
        max_contratto(X, corso(Corso)),
        corso(Corso).
```

Codice 5. Frammento del file `main.lp`.

IV. Risultati

A. Punti chiave

In questa sezione vengono esposti i principali indicatori di qualità relativi ai risultati ottenuti in varie casistiche.

Un primo fattore caratterizzante è emerso valutando le 4 preferenze espresse tramite i weak constraint. La generazione di Answer Set, tenuto conto di tutti i vincoli, risulta molto esosa e il programma non è in grado di fornire il risultato ottimo in tempo utile. Nonostante non sia possibile garantire la produzione della soluzione migliore a causa dei tempi di esecuzione, le soluzioni fornite dal programma sono da considerare corrette ed esaustive, in termini di configurazioni dei docenti. A questo proposito è stato introdotto un limite di tempo di 5 secondi per fermare il risolutore e procedere con l'analisi dell'output.

B. Preferenze

È possibile ottenere risultati ottimali in tempi brevi (qualche secondo) eliminando alcune preferenze, per facilitare il processo di ottimizzazione. In particolare, si raggiungono prestazioni elevate se vengono tenuti in considerazione solo i vincoli relativi ai presidenti e ai professori a tempo indeterminato. Gli esempi riportati in questa sezione sono stati svolti secondo questa logica.

C. Input ridotto

I primi test sono stati eseguiti ponendo come obiettivo la generazione di configurazioni solo per corsi di studio singoli o a coppie, per verificare rapidamente la correttezza dei risultati. Per fare un esempio pratico, il Corso di Laurea Triennale in Informatica (codice 3027) offre 26 insegnamenti, distribuiti su 20 docenti. La categoria del corso e le numerosità di immatricolazioni richiedono la presenza di almeno 9 garanti, tra cui:

- non meno di 5 professori a tempo indeterminato;
- non più di 4 ricercatori;
- non più di 2 professori a contratto.

Tabella I
Garanti Generati per il Corso 3027.

Docente	Matricola	SSD 2015
Dal Palù A.	6625	INF/01
De Filippis C.	34499	MAT/05
Bonnici V.	34181	INF/01
Guardasoni C.	6801	MAT/08
Benini A.	26131	MAT/03
De Pietri R.	5536	FIS/02
Zaffanella E.	5602	INF/01
Bergenti F.	204741	INF/01
Bagnara R.	5145	INF/01

La soluzione ottimale, in questo caso, viene generata in meno di 1 secondo. Nonostante l'esempio sia banale, i vincoli imposti sono stati rispettati e la soluzione è facilmente consultabile da parte dell'utente.

In alcuni casi, come per il Corso di Laurea Triennale in Chimica (codice 3024), il numero di insegnamenti (e di conseguenza il numero di docenti) è molto alto; in particolare, si riscontra un elevato numero di professori con contratto a tempo indeterminato. In queste casistiche, il programma finisce per valutare una quantità elevatissima di modelli, potenzialmente ottimi, tutti tra loro equivalenti. Per questo motivo il solver potrebbe evolvere in una situazione di "stallo", non riuscendo a dimostrare di aver trovato un modello ottimale. Nel Cod. 6 è riportato l'output in questione, da cui è possibile notare come la maggior parte del tempo di risoluzione sia impiegato nella ricerca (senza successo) di un modello migliore. Il tempo indicato nel campo `Unsat` indica, infatti, il tempo trascorso tra l'istante in cui è stato trovato l'ultimo modello e la terminazione [5]. Sono riportati, cambiati di segno, i valori di ottimizzazione raggiunti, ovvero 1 (presidente) e 9 (professori a tempo indeterminato), che rappresentano i valori migliori per il corso di laurea in questione.

```
Models       : 6
Optimum      : yes
Optimization : -1 -9
Calls        : 1
Time         : 76.021s
                  (Solving: 76.01s
                   1st Model: 0.00s
                   Unsat: 76.00s)
CPU Time     : 75.992s
```

Codice 6. Statistiche clingo per il corso 3024.

Anche per questo motivo, è stato introdotto l'utilizzo di un *timer* all'interno del codice Python, della durata di 5 secondi. Nei test eseguiti, questo limite si è rivelato sufficiente per fornire soluzioni che rispettassero i vincoli prestabiliti.

D. Input completo

La discrepanza tra i due esempi riportati nei test sui corsi singoli mette in mostra la complessità del modello, tipica dei

problemi affrontati tramite il paradigma di programmazione *Answer Set*.

Un'ulteriore conferma si ottiene con l'esecuzione del programma tenendo in considerazione tutti i corsi dell'ateneo. La ricerca del modello ottimo si conclude, infatti, quasi in maniera istantanea, contrariamente a quanto ci si possa aspettare. Il fatto di avere più corsi e professori a disposizione rende alcune soluzioni meno ottimali in maniera naturale e alleggerisce il carico di lavoro del solver.

```
Models       : 196
Optimum      : yes
Optimization : -76 -705
Calls        : 1
Time         : 0.800s
                  (Solving: 0.65s
                   1st Model: 0.36s
                   Unsat: 0.00s)
CPU Time     : 0.714s
```

Codice 7. Statistiche clingo per tutti i corsi.

Preferenze massime: Utilizzando tutti i vincoli presenti nel file delle preferenze, si ottengono configurazioni più specifiche, in base alle proprietà descritte in dettaglio alla sezione III-C. Nel codice 8 viene mostrato un esempio, con un limite di tempo impostato a 5 secondi. Si notano alcune proprietà a riguardo:

- la risoluzione richiede in generale più tempo;
- i livelli di ottimizzazione raggiunti relativi a presidenti e professori a tempo indeterminato non cambiano di molto rispetto all'esempio mostrato nel codice 7;
- i livelli di ottimizzazione relativi ai predicati `ideale/1` e `non_ideale/1` si possono considerare bilanciati.

```
TIME LIMIT   : 1
Models       : 238+
  Optimum    : unknown
Optimization : -68 -43 -702 52
Calls        : 1
Time         : 5.003s
                  (Solving: 4.82s
                   1st Model: 1.38s
                   Unsat: 0.00s)
CPU Time     : 4.870s
```

Codice 8. Statistiche clingo per tutti i corsi con preferenze massime.

V. Conclusione

Riferimenti bibliografici

[1] P. S. Foundation, "Python programming language," https://www.python.org/, 2024, accessed: 2024-12-19. [Online]. Available: https://www.python.org/

[2] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, "clingo: A grounder and solver for answer set programming," https://github.com/potassco/clingo, 2024, accessed: 2024-12-19. [Online]. Available: https://github.com/potassco/clingo

[3] Potassco, "Clingo python api," 2024, accessed: 2024-12-19. [Online]. Available: https://potassco.org/clingo/python-api/5.4/#

[4] U. di Parma, "U.O. Progettazione Didattica e Assicurazione della Qualità," 2024, accessed: 2024-12-19. [Online]. Available: https://trasparenza.unipr.it/ugov/organizationunit/191497

[5] M. Gebser, R. Kaminski, B. Kaufmann, M. Lindauer, M. Ostrowski, J. Romero, T. Schaub, S. Thiele, and P. Wanko, "Potassco user guide," *Institute for Informatics, University of Potsdam, second edition edition*, vol. 69, 2015.