AI documentation - 3 Assignment

Piccirilli - Addario

Novembre 2023

Introduzione

Considerando la griglia sottostante, ci è stato chiesto di rispondere a delle domande. In particolare, la griglia rappresenta un cruciverba da riempire con un insieme di parole **words** = {'add', 'age', 'aid', 'aim', 'air', 'are', 'arm', 'art', 'bad', 'bat', 'bee', 'boa', 'dim', 'ear', 'eel', 'eft', 'lee', 'oaf'}. Il problema viene codificato in termini di *"Constraint Satisfaction Problem"*, dove, difatto, le intersezioni tra le parole rappresentano i vincoli da rispettare.

Scendendo più nel dettaglio, il problema può essere rappresentato attaverso due form:

1. La prima rappresentazione consta di un insieme di **variabili** che modellano le righe e le colonne da riempire (in particolare one_across, one_down, two_down, three_down, four_across, five_across).

```
words = {'add', 'age', 'aid', 'aim', 'air', 'are',
    'arm','art', 'bad', 'bat', 'bee', 'boa', 'dim'
    , 'ear', 'eel', 'eft', 'lee', 'oaf'}
one_across = Variable('one_across', words,
    position=(0, 0))
one_down = Variable('one_down', words, position
    =(0, 0.3))
two_down = Variable('two_down', words, position
    =(0, 0.6))
three_down = Variable('three_down', words,
    position=(0, 0.9))
four_across = Variable('four_across', words,
    position=(0, 1.2))
five_across = Variable('five_across', words,
    position=(0, 1.5))
```

   Inoltre, sono presenti un insieme di **vincoli** che indicano le intersezioni delle parole

```
Constraint([one_across, one_down], meet_at(0, 0),
    position=(1, 0.1)),
```

2. La seconda rapresentazione presenta un insieme di variabili che modellano le celle della griglia da riempire, le quali possono assumere come valori l'insieme delle lettere dell'alfabeto (letters = "a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z"). Inoltre, queste lettere non possono essere messe a casaccio, ma devono concatenarsi in modo da formare delle parole presenti nell'insieme *words* presente precedentemente.

```
p00 = Variable('p00', letters, position=(0.1,
    0.85))
...
Constraint([p00, p10, p20], is_word, position
    =(0.3, 0.95)), # 1-across
```

Infine, per trovare uina soluzione si utilizza la classe **Searcher** a cui viene passato il tipo di problema in esame:

```
"""Found solution with Arc Consistency"""
CSP_Searcher_with_AC = Searcher(
    Search_with_AC_from_CSP(crossword1)).search()
print("Solutions: ", CSP_Searcher_with_AC.end())

"""Found solution without Arc Consistency"""
"""Don't do this, is very slow (a run expanded 73k
    path on my laptop) """
CSP_Searcher = Searcher(Search_from_CSP(crossword1)).
    search()
print("Solutions: ", CSP_Searcher.end())
```

Punto 1

Give an example of pruning due to domain consistency using the first representation (if one exists).

Non sono presenti esempi di pruning del domain a causa della consistenza del dominio.

Punto 2

Give an example of pruning due to arc consistency using the first representation (if one exists).

Per quanto riguarda la prima rappresentazione, inizialmente il domain di one_across = {'age', 'oaf', 'bee', 'arm', 'bat', 'boa', 'eft', 'art', 'dim', 'ear', 'lee', 'bad', 'aim', 'are', 'add', 'air', 'eel', 'aid'}

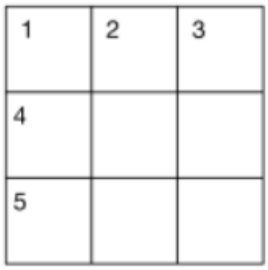

Figure 1: Immagine della griglia da riempire

Dal domain di two_down vengono tagliate fuori delle parole a causa del constraint di *"meet_at"* tra **one_across** e **two_down**: infatti notiamo che, se scegliessimo le paroli del domain di one_across, a quel punto potremmo avere come parole legali di two_down solo quelle che iniziano per 'g', 'a', 'e', 'r', 'o', 'f', 'i', 'd'. Quindi vengono rimosse le parole:

- bee
- bat
- boa
- lee
- bad

```
Domain pruned dom(two_down) = {'age', 'aim', 'oaf', 'are', 'aid', 'arm',
'eft', 'add', 'air', 'eel', 'art', 'ear', 'dim'} due to meet_at(1,0)
[one_across, two_down]
adding nothing to to_do
```

Punto 3

Are domain consistency plus arc consistency adequate to solve this problem using the first representation? Explain

Si, il domain consistency con l'arc consistency trova una soluzione al problema (invece che ridurre semplicemente i domini). Una di queste è {five_across: {'art'}, one_across: {'bee'}, two_down: {'ear'}, three_down: {'eft'}, one_down: {'boa'}, four_across: {'oaf'}} (cost: 2)

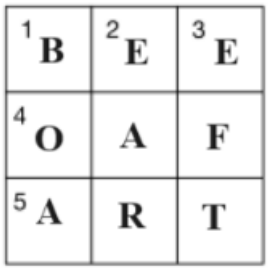

Figure 2: Immagine della griglia con la soluzione nella prima rappresentazione

Punto 4

Give an example of pruning due to domain consistency using the second representation (if one exists).

Per quanto riguarda la seconda rappresentazione, anche in questo caso non sono presenti domain pruning a causa di consistenza del dominio, però si può togliere dai domini di ciascune variabili, con una sorta di pre-processamento, quelle lettere che non compaiono mai nelle parole.

Infatti, i domini vengono ridotti a: {'m', 'e', 'i', 'g', 't', 'r', 'l', 'o', 'b', 'a', 'd', 'f'}

```
def pre_process_literals(problem: CSP):

    # Build the useful literals dictionary
    useful_letters = set()

    for word in words:
        for letter in letters:
            if letter in word:
                useful_letters.add(letter)

    for variable in problem.variables:
        variable.domain = useful_letters

    return problem
```

Media temporale su 100 casi con preprocessing: 0.17936779499053956 s
Media temporale su 100 casi senza preprocessing: 0.3666560196876526 s

Soltanto rimuovendo delle lettere non presenti nelle parole date, si ottiene un miglioramento del 110%

Punto 5

Give an example of pruning due to arc consistency using the second representation (if one exists).

```
Processing arc (p02, is_word[p02, p12, p22])
Arc:  (p02, is_word[p02, p12, p22]) is inconsistent
Domain pruned dom(p02) = 'o', 'l', 'a', 'b', 'e', 'd' due to is_word[p02,
p12, p22]
adding nothing to to_do
Arc:  (p02, is_word[p02, p12, p22]) now consistent
```

In questo esempio controlla che con le lettere del dominio di **p02** si possa costruire una parola, e nota che queste possono iniziare solo per le lettere 'o', 'l', 'a', 'b', 'e' e 'd' (trovadosi p02 all'inizio della riga da scrivere dal momento che sta prendendo in considerazione l'arco tra p02, p12, e p22).

Punto 6

Are domain consistency plus arc consistency adequate to solve this problem using the second representation?

Anche in questo caso, viene trovata una soluzione al problema: infatti {p11: {'a'}, p00: {'b'}, p21: {'r'}, p02: {'e'}, p12: {'f'}, p22: {'t'}, p10: {'o'}, p20: {'a'}, p01: {'e'}}

1 B	2 O	3 A
4 E	A	R
5 E	F	T

Figure 3: Immagine della griglia con la soluzione nella seconda rappresentazione

Punto 7

Which representation leads to a more efficient solution using consistency-based techniques? Give the evidence on which you are basing your answer.

Media temporale prima rappresentazione: 0.1440407967567444 s
Media temporale seconda rappresentazione: 0.2993096303939819 5 s

Tra le rappresentazioni proposte, la più efficiente è senza ombra di dubbio la prima, ma con il preprocessing, i risultato sono confrontabili:

Media prima rappresentazione: 0.1531416344642639 s
Media seconda rappresentazione: 0.2963556218147278 s
Media seconda rappresentazione con preprocessing: 0.22978545904159545s

**Tutti i test sono stati eseguiti sulla stessa macchina, eseguirli altrove potrebbe portare a misure diverse ma con intervalli confrontabili.*