coding-hw3-1002949

June 16, 2020

1 Homework 3

- Name: **Hong Pengfei**
- student id: **1002949**
- using python version: 3.8

[1]:
```
from csp import *
from random import shuffle
import random
random.seed(123)
import numpy as np
```

[2]:
```
def solve_semi_magic(algorithm=backtracking_search, **args):
    """ From CSP class in csp.py
        vars        A list of variables; each is atomic (e.g. int or string).
        domains     A dict of {var:[possible_value, ...]} entries.
        neighbors   A dict of {var:[var,...]} that for each variable lists
                    the other variables that participate in constraints.
        constraints A function f(A, a, B, b) that returns true if neighbors
                    A, B satisfy the constraint when they have values A=a, B=b
                    """
    # Use the variable names in the figure
    csp_vars = [f'V{d}' for d in range(1,10)]

    #########################################
    # Fill in these definitions

    csp_domains = { v:list(range(1,4)) for i,v in enumerate(csp_vars) }
    csp_neighbors = {
        'V1': ['V2', 'V3', 'V4', 'V5', 'V7', 'V9'],
        'V2': ['V1', 'V3', 'V5', 'V8'],
        'V3': ['V1', 'V2', 'V6', 'V9'],
        'V4': ['V1', 'V7', 'V5', 'V6'],
        'V5': ['V1', 'V2', 'V4', 'V6', 'V8', 'V9'],
        'V6': ['V3', 'V4', 'V5', 'V9'],
        'V7': ['V1', 'V4', 'V8', 'V9'],
        'V8': ['V7', 'V2', 'V5', 'V9'],
        'V9': ['V1', 'V5', 'V3', 'V6', 'V7', 'V8'],
    }

    random.shuffle(csp_vars)
    csp_domains = list(csp_domains.items())
    random.shuffle(csp_domains)
    csp_domains = dict(csp_domains)
    for o in csp_domains.values(): random.shuffle(o)
    csp_neighbors = list(csp_neighbors.items())
    random.shuffle(csp_neighbors)
    csp_neighbors = dict(csp_neighbors)
    for o in csp_neighbors.values(): random.shuffle(o)

    def csp_constraints(A, a, B, b):
        return a != b

    #########################################

    # define the CSP instance
    csp = CSP(csp_vars, csp_domains, csp_neighbors,
              csp_constraints)

    # run the specified algorithm to get an answer (or None)
    ans = algorithm(csp, **args)

#     print('number of assignments', csp.nassigns)
    assign = csp.infer_assignment()
#     if assign: for x in sorted(assign.items()): print(x)
    return csp

n = 10000
print('='*80)
print("Constraint on Roles, columns and the diagonal (with \ direction)")
print('='*80)
print("\033[1mmethod \t \t select unassigned variable \t select domain values␣
 ↪\t\t inference \t\t average assignments\033[0m")
ns = []
for _ in range(n): ns.append(solve_semi_magic(backtracking_search,␣
 ↪select_unassigned_variable = first_unassigned_variable,
                       order_domain_values = unordered_domain_values, inference␣
 ↪= no_inference).nassigns)
print(f"backTracking, \t first unassigned variable, \t unordered, \t\t\t no␣
 ↪inference ,\t\t  {np.mean(ns)}")
ns = []
for _ in range(n): ns.append(solve_semi_magic(backtracking_search,␣
 ↪select_unassigned_variable = mrv,
                 order_domain_values = unordered_domain_values, inference =␣
 ↪no_inference).nassigns)
```

```
print(f"backTracking, \t minimum remaining values, \t unordered, \t\t\t no␣
↪inference ,\t\t  {np.mean(ns)}")
ns = []
for _ in range(n): ns.append(solve_semi_magic(backtracking_search,␣
↪select_unassigned_variable = first_unassigned_variable,
                  order_domain_values = lcv, inference = no_inference).nassigns)
print(f"backTracking, \t first unassigned variable, \t least constraining␣
↪value, \t no inference , \t  {np.mean(ns)}")
ns = []
for _ in range(n): ns.append(solve_semi_magic(backtracking_search,␣
↪select_unassigned_variable = first_unassigned_variable,
                  order_domain_values = unordered_domain_values, inference =␣
↪forward_checking).nassigns)
print(f"backTracking, \t first unassigned variable, \t unordered, \t\t\t␣
↪forward checking ,\t  {np.mean(ns)}")
ns = []
for _ in range(n): ns.append(solve_semi_magic(backtracking_search,␣
↪select_unassigned_variable = first_unassigned_variable,
                  order_domain_values = unordered_domain_values, inference =␣
↪mac).nassigns)
print(f"backTracking, \t first unassigned variable, \t unordered, \t\t\t AC-3␣
↪,\t\t\t  {np.mean(ns)}")
ns = []
for _ in range(n): ns.append(solve_semi_magic(backtracking_search,␣
↪select_unassigned_variable = mrv,
                  order_domain_values = lcv, inference = mac).nassigns)
print(f"backTracking, \t minimum remaining values, \t least constraining value,␣
↪\t AC-3 ,\t\t\t  \033[1m{np.mean(ns)}\033[0m")
```

```
================================================================================
Constraint on Roles, columns and the diagonal (with \ direction)
================================================================================
method                   select unassigned variable      select domain values

inference                   average assignments
backTracking,    first unassigned variable,      unordered,
no inference ,           14.8958
backTracking,    minimum remaining values,       unordered,
no inference ,           14.8054
backTracking,    first unassigned variable,      least constraining value,
no inference ,           14.6666
backTracking,    first unassigned variable,      unordered,
forward checking ,       11.8879
backTracking,    first unassigned variable,      unordered,
AC-3 ,                   9.4539
backTracking,    minimum remaining values,       least constraining value,
AC-3 ,                   9.0
```

1.0.1 Experiment setup

The number of assignments depends largely on the randomness that python is using. So I took the average of 10000 iterations of the problem and do ablation study on effectiveness of each methods. ### Findings - Pure backtracking algorithm give the highest average number of assignments and the worst result. - Minimum remaining values as variable assigning order can slightly improve the result but give negligible difference - Least Constraining values as values assigning order can also slightly improve the result and is more effective than Minimum remaining values on this task. - forward checking to detect failure early can greatly improve the result. - Arc consistency checking have the most positive effect on average assignments and is more effective than forward checking in reducing the number of assignments. - Finally I combined all the methods together, and it gave the best result possible: average of 9 assignments.

1.0.2 My understanding

For semi magic square problem, - minimum remaining values and Least Constraining values showed some effectiveness but did not helped a lot because we have roughly same amount of constraints on each variable, and each value will have roughly the same number of constraints on other variables. Therefore the order in which variables and values are assigned only have a little effect on the result. - However, forward checking algorithm can terminate early when any variable has no legal values and don't have to wait for the value to be assigned. therefore it can improve the result greatly. - AC-3 algorithm is better than forward checking algorithm because it not only check the immediate node under constraint but also check all the neighbors the information proprogates. therefore can early spot the failures. - combination of all these methods can have all adavantages and therefore have the best result.

```
[ ]:
```