CSE537 Artificial Intelligence

Assignment-3 Report

CSP using Backtracking, Forward Checking and Constraint Propagation

Authors:

Vishal Nayak: 109892702: vnayak@cs.stonybrook.edu

Ashish Chaudhary: 109770154: ashchaudhary@cs.stonybrook.edu

Contents

1. Intro and details

***NOTE**: Graphs are not added in this report because running times were insignificant.*

1.1. Compilation and machine details

Compiler: ***python 2.7.9***

1) Backtracking: ***python bt.py input.txt***
2) Backtracking + Forward Checking: ***python bt_fc.py input.txt***
3) Backtracking + Forward Checking + Constraint Propagation: ***bt_fc_cp.py input.txt***

Machine: Windows 8 64-bit

1.2. Approach

Created a class CSP to hold the following information.

- List of all course names.
- List of lists containing timings for all the courses.
- List of lists containing recitation timings.
- List of lists containing domains of each course.
- List of number of students required for courses.
- List of lists containing the TA skills required by courses.
- List containing ta presence requirement in courses.
- List of TA objects.
- List of lists containing TAs assigned for courses.
- List of lists containing courses assigned to TAs.

Parsed the input file and filled in the above structures.

Performed the below filtering procedures before invoking the search.

- Filter out recitation overlaps
- Filter out lecture timings overlaps
- Filter out skill set mismatches

Wrote a common recursive solver for all the strategies: Backtracking, Forward checking and Constraint Propagation. Selection of which strategy to use is parameterized.

2. Backtracking Search

After applying all the filtering procedures mentioned above, all the courses will only hold valid TA domains in the appropriate data structures.

Each instance of recurrence fetches an unassigned course and from its domain, picks a domain and assigns it.

The search is continued to assign other courses.

The TA capacity is maintained by having an object reference for each TA and the variable 'availability' inside it.

If the search is not complete, the recursion will try to find other assignments.

Analysis:

Branching factor is too big. As the number of courses and corresponding domain size increases, the search complexity increases exponentially.

3. Forward Checking

At each instance of the recursion, when a TA is assigned to a course, forward checking procedure is invoked. As mentioned earlier, the TA capacity is kept track of by 'availability' variable in TA object instances.

So, after assigning TA to a particular course, a linear search is invoked to look for courses containing the same TA in their domain and if the availability of that TA becomes zero, the courses cumulative 'availability' is calculated. And if that becomes empty, it can be concluded that the search is not going to succeed. Hence the search is stopped at that point.

Analysis:

A linear time overhead at each iteration of the problem but provides a safety check to avoid probable unnecessary recursions. If the domains are too huge, initial overheads will be high. Since the data sets with which this assignment is done is remarkably small, the advantages of forward checking is barely noticed. However, the course containing with the contention is noticed before the search reaching it eventually.

4. Constraint Propagation

After forward checking is completed at each recursive step, along with it is the constraint propagation procedure which checks for arc consistency. Here all the neighbors of all the domain values of all the unassigned nodes are checked for inconsistencies. If inconsistencies are found, then that particular domain is removed from the current course.

After removing all the inconsistencies, if the domains of any of the course loses availability of all the resources, the search is terminated at that point.

Analysis:

This is a significant overhead at each recursion instance. However, it is still employable since backtracking in itself is exponential by a branch factor proportional to domains. Hence it can be considered as a good add on. Again, since the data sets were too small, the running time provided no significant values for analysis. But, in this case, the inconsistency in the data were identified even before the search procedure was started. i.e, after filtering of domains were completed, if consistency check is run on the domains before invoking the recursive search, I was able to detect failures, even before the search was invoked!

5. Traces of Search Strategies

5.1. Backtracking

==Course Domains after filtering==
CSE101 : MRS._LAUREN_SMITH(1)
CSE102 : TA4(1)
CSE103 : TA4(1)
CSE104 : TA5(1) TA8(1)
CSE105 : TA2(1)
CSE106 : TA6(1)
CSE107 : TA7(1)

Required TA: [1.5, 0.5, 0.5, 1.5, 2, 1.5, 2]

TimeTaken: 0.000143691716528

Backtracking Successful!

== Courses assignted to TAs ==
MRS._LAUREN_SMITH: [('CSE101', 1)]
TA2: [('CSE105', 1)]
TA4: [('CSE102', 0.5), ('CSE103', 0.5)]
TA5: [('CSE104', 1)]
TA6: [('CSE106', 1)]
TA7: [('CSE107', 1)]

== TAs assigned to Courses ==
CSE101: [('MRS._LAUREN_SMITH', 1)] and Need: 0.5
CSE102: [('TA4', 0.5)] and Need: 0.0
CSE103: [('TA4', 0.5)] and Need: 0.0
CSE104: [('TA5', 1)] and Need: 0.5
CSE105: [('TA2', 1)] and Need: 1
CSE106: [('TA6', 1)] and Need: 0.5
CSE107: [('TA7', 1)] and Need: 1

5.2. Forward Checking

==Course Domains after filtering==
CSE101 : MRS._LAUREN_SMITH(1)
CSE102 : TA4(1)
CSE103 : TA4(1)
CSE104 : TA5(1) TA8(1)
CSE105 : TA2(1)
CSE106 : TA6(1)
CSE107 : TA7(1)

Required TA: [1.5, 0.5, 0.5, 1.5, 2, 1.5, 2]

Forward checking for course: CSE101 after assigning TA: MRS._LAUREN_SMITH
Forward checking for course: CSE102 after assigning TA: TA4
Forward checking for course: CSE103 after assigning TA: TA4
Forward checking for course: CSE104 after assigning TA: TA5

Forward checking for course: CSE105 after assigning TA: TA2
Forward checking for course: CSE106 after assigning TA: TA6
Forward checking for course: CSE107 after assigning TA: TA7

TimeTaken: 9.71103358952e-05

Backtracking Successful!

== Courses assignted to TAs ==
MRS._LAUREN_SMITH: [('CSE101', 1)]
TA2: [('CSE105', 1)]
TA4: [('CSE102', 0.5), ('CSE103', 0.5)]
TA5: [('CSE104', 1)]
TA6: [('CSE106', 1)]
TA7: [('CSE107', 1)]

== TAs assigned to Courses ==
CSE101: [('MRS._LAUREN_SMITH', 1)] and Need: 0.5
CSE102: [('TA4', 0.5)] and Need: 0.0
CSE103: [('TA4', 0.5)] and Need: 0.0
CSE104: [('TA5', 1)] and Need: 0.5
CSE105: [('TA2', 1)] and Need: 1
CSE106: [('TA6', 1)] and Need: 0.5
CSE107: [('TA7', 1)] and Need: 1

5.3. Constraint Propagation

==Course Domains after filtering==
CSE101 : MRS._LAUREN_SMITH(1)
CSE102 : TA4(1)
CSE103 : TA4(1)
CSE104 : TA5(1) TA8(1)
CSE105 : TA2(1)
CSE106 : TA6(1)
CSE107 : TA7(1)

Required TA: [1.5, 0.5, 0.5, 1.5, 2, 1.5, 2]

Checking consistency for course: CSE101
Forward checking for course: CSE101 after assigning TA: MRS._LAUREN_SMITH
Checking consistency for course: CSE101
Forward checking for course: CSE102 after assigning TA: TA4
Checking consistency for course: CSE102
Forward checking for course: CSE103 after assigning TA: TA4
Checking consistency for course: CSE103
Forward checking for course: CSE104 after assigning TA: TA5
Checking consistency for course: CSE104
Forward checking for course: CSE105 after assigning TA: TA2
Checking consistency for course: CSE105
Forward checking for course: CSE106 after assigning TA: TA6
Checking consistency for course: CSE106

Forward checking for course: CSE107 after assigning TA: TA7
Checking consistency for course: CSE107

TimeTaken: 0.000521079851145

Backtracking Successful!

== Courses assignted to TAs ==
MRS._LAUREN_SMITH: [('CSE101', 1)]
TA2: [('CSE105', 1)]
TA4: [('CSE102', 0.5), ('CSE103', 0.5)]
TA5: [('CSE104', 1)]
TA6: [('CSE106', 1)]
TA7: [('CSE107', 1)]

== TAs assigned to Courses ==
CSE101: [('MRS._LAUREN_SMITH', 1)] and Need: 0.5
CSE102: [('TA4', 0.5)] and Need: 0.0
CSE103: [('TA4', 0.5)] and Need: 0.0
CSE104: [('TA5', 1)] and Need: 0.5
CSE105: [('TA2', 1)] and Need: 1
CSE106: [('TA6', 1)] and Need: 0.5
CSE107: [('TA7', 1)] and Need: 1