Fiona Heaney & Pat Lebold

CS 4341 C16

Assignment 5 – CSP

**Instructions for compilation**

* Our program require Java 8
* Run: java – jar BagsAndWeights.jar <input\_file.txt>

**Our Approach**

* Create list of sub-lists of objects that are casted to constraint objects, weights, and bags
* Create a flag, isValid that if false, terminates the DFS backtracking search for a node. If true, we know that the currently placed items satisfy all constraints. Additionally create a global flag for validity (allFinal) that ensures all bag placements so far are valid.
* Recursively call solve method on the remaining list of unplaced items (weightsToTest).
  + This is the list we modify with the LCV and MRV algorithms, ultimately reordering the list of remaining items to optimize placement
* Begin placing items with DFS

(For each bag in the list of bags)

(For each of the unplaced items)

(If item can be placed in this bag)

Remove from unplaced item list

Add item to this bag

Recursively call solve on remaining items

Remove item from bag and place back into unplaced item list if recursive call returns invalid or no solution.

* Adding heuristics
  + Minimum remaining value heuristic

Because we decided to iterate through the bags and then items, we adapted the MRV algorithm to remove items that were too heavy for the given bag capacity rather than looking for the item with the fewest legal values. We decided that we would instead remove any items from the local copy of unplaced items such that

(While we still have items to place && bagWeight + currentItemWeight > bagCapacity)

Remove that item from the potential items for placement

* + Least Constraining Value / Forward Checking

We ranked each item based on the number of valid sub-placements by iterating through the unsorted items, checking if they could be placed in the given bag, and mapping the number of valid sub moves after placement to that item in a hashmap. We then sorted the list of items based on the number of valid sub moves.

For each unsorted item

Put in bag

Remove if not valid, otherwise

Make a sublist of unsorteditems – the item we just placed

For each remaining unsorted item, test if valid in this bag && validCount++ for each successful addition.

Create hashmap entry with validCount and item

Sort based on number of sub valid moves

**Testing**

* After implementing just our DFS, we ran our program with all 26 sets of test data. We found that it correctly placed the first 24 input files correctly but could not efficiently compute the placement of items for file 25 and 26 despite adding memoization, LCV, MRV, and forward checking. The average number of comparisons for input files 20, 21, 22, 23, and 24 can be seen in the table below.
* We found that our program performance for was greatly impacted by the implementation of combining our heuristics with forward checking and memoization. Forward checking allowed us to remove invalid moves sooner than our DFS search method, thus making our program more efficient. This can especially be seen in file 24 where we cut our comparison count from a consistent 3591 (file 20-23) to 12 comparisons with forward checking.

**Strengths and Weaknesses**

* Our code is very efficient when implementing our LCV/ forward checking with the first 25 cases, however even our most efficient case cannot solve the input file 26
* We decided to use Java’s .sort() after implementing our LCV or MRV heuristic which handles tie breaking, which we would have liked to develop more if given more time.

**Comparison of CSP Algorithms**

*Please see attached .txt files for comparison checking*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *BT* | *BT + MRV* | *BT + LCV* | *BT + Heuristics + FC* |
| *Avg # of Comparisons Needed for Solution* | 7936 | 7844 | 7844 | 45 |

I believe my biggest challenge as a community advisor would be