CMPT 310 Assignment 2
Documentation

Search procedure

In this assignment, I used **backtracking search** to assign a list of domains(i.e, colours) to the variables(i.e., countries), and ensure that each variable is assigned with a colour, and no adjacent variables share the same colour. The backtracking search is based upon the depth-first search. The search procedure proceeds with an arbitrary unassigned variable v (without most remaining value and degree heuristics) each time. For each colour domain c, the algorithm checks the validity of (v, c), see whether the assignment (v, c) conflicts with the current solution(i.e., the current state of the graph). If valid, we exert depth-first search on node v, that is, recursively call $backtracking\_search(v)$. Otherwise, the algorithm traces back to the nearest assigned node u and assign u with a new colour. The iteration proceeds so on and so forth until either each node has been assigned with a colour and no adjacent nodes share the same color, or the search algorithm has tried out all possible colour assignments to each node and no assignment satisfy the constraints(CSP can not be solved).

High level description of the program

My program consists of three major parts: $build()$, $solve()$, and $check\_failure()$. $build()$ converts the given adjacency list into a adjacency matrix in which each matrix entry represents the adjacency of two vertices. For example, $adjacency[2][3] = 1$ represents that node 3 and 4 are adjacent to each other while $adjacency[2][3] = 0$ indicates that there is no edge between node 3 and 4. The backtracking search inside the $solve()$ check if the color assignment (v, c) is valid based on this adjacency matrix.

$solve()$ contains a $while\ loop$ in which the backtracking search takes place. Before entering the $while\ loop$, the function calls $select\_by\_mrv()$ and $lcv()$ respectively to apply the minimal remaining values, degree heuristic, and the least constraining value heuristic to the backtracking search. $select\_by\_mrv()$ chooses the optimal unassigned variable, and $lcv()$ orders the colour domains. Inside the $while\ loop$, the selected variable tries out each colour value in ordered domains until the assignment does not conflict with the current solution. If all variables have been assigned with a legal colour, the function prints the solution and calls $sys.exit()$ to exit the program. Otherwise, $solve()$ fills up the solution list and wait $check\_failure()$ to prints an empty list.

$check\_failure()$ is just used to handle the case where there is no solution, and prints an empty list.

My test case

$n = 14$
$k = 4$
$G = [[1,2,3,4,6,7,10],\ [2,1,3,4,5,6,11,12,14],\ [3,1,2,11,14],\ [4,1,2,11,12],\ [5,2,6,13],$
$[6,1,2,5,7,8,13],\ [7,1,6,8,9,10],\ [8,6,7,9,12,14],\ [9,7,8,10,14],\ [10,1,7,9,14],$
$[11,3,2,4],\ [12,4,8,2],\ [13,5,6],\ [14,2,8,3,9,10]]$

Solution = [(2, 1), (1, 2), (6, 3), (7, 1), (8, 4), (14, 2), (9, 3), (10, 4), (3, 3), (4, 3), (12, 2), (5, 2), (11, 2), (13, 1)]

If $k = 3$, the search will fail. That is, no solution.

Improvements by mrv, degree, and the least constraining value heuristics

Minimum remaining value heuristic can be combined with the backtracking search. It chooses an unassigned variable with the fewest legal values, that is, selecting a variable that is most likely to cause the failure, therefore pruning the search sub-trees. **Degree heuristic** is also a strategy used for speeding up the search process. It chooses an unassigned variable which has the largest number of constraints on other unassigned variables. In my program, degree heuristic is used as the tie-breaker for the variables during the mrv selection, that is, if there are multiple minimum item in mrv list, choose the one that has the highest degree.

Least constraining value heuristic applies when decide in what order the domain values should be assigned to the selected variable. It prefers to the domain value that rules out the fewest choices for the neighboring variables. Therefore, the variable leaves the most flexibility to its neighboring variables and maximize the chance to find the solution.

The following table shows the performance of backtracking search with and without the mrv, degree, and least constraining value heuristics in **my test case above with 3-colourings** in which failure will be detected. The performance is measured in **millisecond(ms)**.

	run_1	run_2	run_3	Average runtime
Backtracking	3.49ms	3.02ms	3.61ms	**3.37ms**
Backtracking + **mrv heuristic**	2.45ms	2.65ms	2.01ms	**2.37ms**
Backtracking + **mrv** + **degree heuristics**	2.08ms	1.92ms	2.06ms	**2.02ms**
Backtracking + **mrv** + **degree** + **lcv heuristic**	1.25ms	1.26ms	1.40ms	**1.30ms**

To conclude, mrv, degree, and least constraining value heuristics can significantly improve the efficiency of backtracking search. Combining **mrv** with backtracking search can reduce the total search time by **29.7%**. The combination of **mrv** and **degree heuristic** averagely reduce the search time by **40.1%**. The implementation of **mrv**, **degree** and **least constraining value heuristics** reduce the total search time by **61.4%**. Hence the performance of the backtracking solver has been significantly improved comparing to the normal backtracking search without implementing the heuristics.