Puzzle Solver (Part 3)

By Applying Problem Solving Techniques

Abdullah Al Raqibul Islam

University of North Carolina at Charlotte | Id# 801151189

Project Goal

The goal of this project is to build a generic interactive pluggable application for solving puzzles (i.e. 8-puzzle, 15-puzzle, n-queen, k-coloring, etc.) using different problem-solving techniques (i.e. informed search, uninformed search, hill-climbing search, backtracking search etc.). Here by word "pluggable" we mean in solving puzzles user can independently decide the search strategy along with the custom heuristic functions. If needed, user can extend this project to device their own solution with a very low effort. Besides, this application is designed in a way that, it will be an easy-going platform for benchmarking any puzzles w.r.t. different state space search strategy, optimization techniques and heuristic functions.

Project Structure

The project has several independent parts that we combine to work as a whole. Directory "core" contains two factory methods that produce the puzzle solver and heuristic instance based on the parameter passed. Puzzle solutions can be implemented in separate directory as we have done for "k-coloring" here. Directory "utils" contains all the utility methods that help other functions to operate.

Please keep in mind, this is an active project and project architecture may change without any prior announcement. But all the future updates will be maintained in a way so that the project integrity will ke kept, meaning you will not face any difficulty in building or running the project.

How to Build

```
# go to project directory
$ cd puzzle-solver

# build the project
$ make clean && make
```

k-coloring

Problem Formulation

In graph theory, graph coloring is an assignment problem which is computationally a NP-complete problem. The basic formulation of this problem is, we need to assign a color to every vertices of a graph in such a way that no two adjacent vertices are assigned to the same color. It is also well known as vertex coloring.

Graph coloring problem can be transformed to a Constraint Satisfaction Problem (CSP). A CSP is composed of the following components:

1. A set of variables X1, X2, ..., Xn with domains (possible values) D1, D2, ..., Dn
2. A set of constraints C1, C2, ..., Cm
3. Each constraint Ci limits the values that a subset of variables can take, e.g., V1 != V2

For graph coloring problem, we can consider,

1. Variables: vertices of the graph
2. Domains: different colors (i.e. red, green, blue)
3. Constraints: no two countries sharing a border have the same color

A k-coloring of a graph thus is an assignment of k colors, one to each vertex, in such a way that no two vertices sharing a edge have the same color. In this project, I will experiment with graph coloring techniques and compare the observed results in the context of USA and Australia maps. We can either find the chromatic number of a graph or test whether a graph can be colored with a MAX_COLOR number. The chromatic number of a graph G, written as $\chi(G)$, is the smallest number of colors needed to color G. To keep it flexible, I considered both of them.

Input Format

We consider the constraint graph as a directed one, meaning if there is a bi-directional constraint between A & B then you need to put both the edges A-B and B-A in your input.

The first line of the input file will contain an integer N, representing the number of vertices in the input graph. The next N lines will contain the adjacency list of the vertices. Every vertex will be a single word (without any space). Adjacency list elements will be separated by the space. A graph representing Australian map is given here,

```
7
Western-Australia Northern-Territory South-Australia
Queensland Northern-Territory South-Australia New-South-Wales
New-South-Wales Victoria South-Australia Queensland
Victoria South-Australia New-South-Wales
Tasmania
Northern-Territory Western-Australia South-Australia Queensland
South-Australia Western-Australia Northern-Territory Queensland New-South-Wales Victoria
```

Implementation Domain

To solve k-coloring problem, currently I have implemented the following algorithms:

Graph Coloring Algorithms	Parameter token for this algorithm
Depth first search only	bt
Depth first search + forward checking	bt-fc
Depth first search + forward checking + propagation through singleton domains	bt-fc-st

Currently I have implemented the following heuristic functions to support the above algorithms:

Heuristic Function	Parameter token for this heuristic function
Minimum Remaining Values (MRV)	-mrv
Degree Heuristic	-degree_heu
Least Constraining Value (LCV)	-lcv

MRV and Degree heuristic is applied on the variable (i.e. vertex) selection and LCV is used to select the value (i.e. color). I have implemented the MRV and Degree Heuristic as a filter that can work individually over a list of available vertices. Details about this heuristic function has been discussed in separate section.

Please keep in mind that, for the current implementation all the parameter token specified above is expected to be exact matched.

How to Run

General run command:

```
> ./puzzle -problem {PUZZLE_NAME} \
-algo {ALGORITHM_NAME} \
-file {INPUT_FILE} -mcolor {MAX_ALLOCABLE_COLOR} \
-print_path {PRINT_INITIAL_TO_GOAL} \
-mrv -degree_heu -lcv
```

Here is the parameter definition,

1. -problem: Specify the puzzle name to solve, for example, "n-queen".
2. -algo: Specify the search strategy to solve the puzzle, for example, "bt", "bt-fc", "bt-fc-st", etc.
3. -file: Input file containing constraint graph.

4. -mcolor: Maximum allowed color, value -1 means we need to find the chromatic number, value (>0) means we need to color the graph by using this number of colors.
5. -print_path: Flag to indicate printing the color assignment after running the coloring algorithm, accepts boolean string, i.e., "true" or "false".
6. -mrv: Flag to indicate whether we will apply heuristic Minimum Remaining Values (MRV) for selecting the next vertex to color.
7. -degree_heu: Flag to indicate whether we will apply heuristic Minimum Remaining Values (MRV) for selecting the next vertex to color.
8. -lcv: Flag to indicate whether we will apply heuristic Minimum Remaining Values (MRV) for selecting the next vertex to color.

Basic Algorithm Description & Run Commands

For k-coloring, here is the list of run commands for different implemented variations of the backtrack search algorithm –

- **Default Backtrack Search**

Implemented standard Backtrack search algorithm, picked a random unassigned vertex if no heuristic specified (i.e. MRV, Degree Heuristic, etc).

Run command:
> ./puzzle -problem map-color -algo bt -file aus.in -mcolor -1 -print_path false

- **Backtrack Search + Forward Checking**

Improvised Backtrack search algorithm with forward checking. While assigning a color to a vertex, removed this color from the neighboring vertices' color domain. This will thus reduce the color choices for the future vertices.

Run command:
> ./puzzle -problem map-color -algo bt-fc -file aus.in -mcolor 3 -print_path false

- **Backtrack Search + Forward Checking + Singleton Propagation**

Further improvised Backtrack search algorithm with singleton propagation. Besides reducing the color choices for the future vertices, we further check whether there is any vertex that has a single available color in its domain. Those vertices are called singleton vertex. If such a single vertex is found, push that vertex to assign color. The intuition is, as singleton vertex has only single color available in its color domain, assigning that color for those vertices will reduce the number of failure attempts.

Run command:
./puzzle -problem map-color -algo bt-fc-st -file aus.in -mcolor 3 -print_path false

Heuristics for Improving the Performance of Backtracking

There are several heuristic function exist that can help the backtracking algorithm

To choose a vertex:

- Choose the vertex with the minimum remaining values ("MRV" heuristic)
- In case of a tie, choose the vertex that is involved in the most constraints with other variables ("degree" heuristic).

In this project, I implemented these heuristics as a filter that pass through the available (i.e. unassigned color) vertex set. By this way we can use these filters independently if we like to. For maintaining the color domain for each vertex, I used bitmask. Bit value "1" means color of this bit position is available for this vertex, and bit value "0" means unavailable. Degree can easily be calculated by checking the number of neighboring vertices in the constraint graph.

To order the values:
Suppose a vertex has been chosen to be assigned a color. Heuristic "least-constraining-value" helps ordering the values for that node. It prefers the value that rules out the fewest choices for the neighboring variables in the constraint graph.

Algorithm of Least Constraining Value:

```
-  Suppose, we are trying to assign value for variable "o"
-  available values for "o" is 0-9.

       sums = {0:0, 1:0, 2:0}
       For i from 0 to 3:
           plug "o=i" into the constraint formulas
           For each neighbor "N" of "o" in the constraint graph:
               sums[i] += the number of values remaining for "N"

       It picks "i" so that:
           sums[i] = MAX{sums[i] | for all "i" that is a member of "o"'s valid
   values}
```

I used a C++ STL supported priority queue to maintain the order of the colors.

Performance Characterization

All the tests performed on the map data of USA and Australia. For all the tests listed in this section, I measured the execution time (in micro seconds) and number of backtracking happened. In the performance table, “X” means result for that run doesn’t come up within a satisfactory time (~5 minutes).

How to reproduce

To reproduce the performance evaluation, please run the following commands:

```
> cd puzzle-solver

#### run without any heuristic

# Do map color with finding chromatic number (dataset Australia):
> ./puzzle -problem map-color -algo bt        -file src/k-coloring/input/aus.in -
```

```
mcolor -1 -print_path false
> ./puzzle -problem map-color -algo bt-fc     -file src/k-coloring/input/aus.in -
mcolor -1 -print_path false
> ./puzzle -problem map-color -algo bt-fc-st  -file src/k-coloring/input/aus.in -
mcolor -1 -print_path false

# Do map color with finding chromatic number (dataset USA):
> ./puzzle -problem map-color -algo bt        -file src/k-coloring/input/usa.in -
mcolor -1 -print_path false
> ./puzzle -problem map-color -algo bt-fc     -file src/k-coloring/input/usa.in -
mcolor -1 -print_path false
> ./puzzle -problem map-color -algo bt-fc-st -file src/k-coloring/input/usa.in -
mcolor -1 -print_path false

# Do map color with the given color (dataset Australia):
> ./puzzle -problem map-color -algo bt        -file src/k-coloring/input/aus.in -
mcolor 3 -print_path false
> ./puzzle -problem map-color -algo bt-fc     -file src/k-coloring/input/aus.in -
mcolor 3 -print_path false
> ./puzzle -problem map-color -algo bt-fc-st  -file src/k-coloring/input/aus.in -
mcolor 3 -print_path false

# Do map color with the given color (dataset USA):
> ./puzzle -problem map-color -algo bt        -file src/k-coloring/input/usa.in -
mcolor 4 -print_path false
> ./puzzle -problem map-color -algo bt-fc     -file src/k-coloring/input/usa.in -
mcolor 4 -print_path false
> ./puzzle -problem map-color -algo bt-fc-st  -file src/k-coloring/input/usa.in -
mcolor 4 -print_path false

#### run with all the heuristics

# Do map color with finding chromatic number (dataset Australia):
> ./puzzle -problem map-color -algo bt        -file src/k-coloring/input/aus.in -
mcolor -1 -print_path false -mrv -degree_heu -lcv
> ./puzzle -problem map-color -algo bt-fc     -file src/k-coloring/input/aus.in -
mcolor -1 -print_path false -mrv -degree_heu -lcv
> ./puzzle -problem map-color -algo bt-fc-st  -file src/k-coloring/input/aus.in -
mcolor -1 -print_path false -mrv -degree_heu -lcv

# Do map color with finding chromatic number (dataset USA):
> ./puzzle -problem map-color -algo bt        -file src/k-coloring/input/usa.in -
mcolor -1 -print_path false -mrv -degree_heu -lcv
> ./puzzle -problem map-color -algo bt-fc     -file src/k-coloring/input/usa.in -
mcolor -1 -print_path false -mrv -degree_heu -lcv
> ./puzzle -problem map-color -algo bt-fc-st -file src/k-coloring/input/usa.in -
mcolor -1 -print_path false -mrv -degree_heu -lcv

# Do map color with the given color (dataset Australia):
> ./puzzle -problem map-color -algo bt        -file src/k-coloring/input/aus.in -
mcolor 3 -print_path false -mrv -degree_heu -lcv
> ./puzzle -problem map-color -algo bt-fc     -file src/k-coloring/input/aus.in -
mcolor 3 -print_path false -mrv -degree_heu -lcv
> ./puzzle -problem map-color -algo bt-fc-st  -file src/k-coloring/input/aus.in -
mcolor 3 -print_path false -mrv -degree_heu -lcv

# Do map color with the given color (dataset USA):
> ./puzzle -problem map-color -algo bt        -file src/k-coloring/input/usa.in -
mcolor 4 -print_path false -mrv -degree_heu -lcv
> ./puzzle -problem map-color -algo bt-fc     -file src/k-coloring/input/usa.in -
mcolor 4 -print_path false -mrv -degree_heu -lcv
> ./puzzle -problem map-color -algo bt-fc-st  -file src/k-coloring/input/usa.in -
mcolor 4 -print_path false -mrv -degree_heu -lcv
```

Performance Evaluation Without Heuristic

Input Graph	Graph Coloring Algorithms	Run 1	Run 2	Run 3	Run 4
USA	Depth first search only	X	X	X	X
	Depth first search + forward checking	92576 / 281984 us	828784 / 3378414 us	1637795 / 4631376 us	238686 / 769989 us
	Depth first search + forward checking + propagation through singleton domains	0 / 331 us	0 / 280 us	128747 / 170057 us	0 / 275 us
Australia	Depth first search only	454 / 226 us	157 / 143 us	566 / 273 us	454 / 429 us
	Depth first search + forward checking	0 / 40 us	0 / 41 us	0 / 39 us	0 / 31 us
	Depth first search + forward checking + propagation through singleton domains	0 / 25 us	0 / 27 us	0 / 39 us	7 / 34 us

Table 1: Performance of graph coloring algorithms without heuristics.
All the results showing in {number of backtrack happened} / {execution time needed} format.

Observations:

1. Coloring of USA map using basic Backtrack search is not possible within a satisfactory timeout, as the program complexity is nearly O(3^50).
2. Impact of forward checking and propagation through singleton domains is clearly visible for small graphs (i.e. Australian map).
3. Backtrack search better perform while using combination of forward checking and propagation through singleton domains, comparing with forward checking only. As we just record the results from four runs, it is very hard to give a straightforward comparison. It could be better if we can manage it from 1000 runs and generate the percentile results.

Performance Evaluation with Heuristic

Input Graph	Graph Coloring Algorithms	Run 1	Run 2	Run 3	Run 4
USA	Depth first search only	X	X	X	X
	Depth first search + forward checking	0 / 303 us	0 / 235 us	0 / 189 us	0 / 272 us
	Depth first search + forward checking + propagation through singleton domains	0 / 182 us	0 / 246 us	0 / 164 us	0 / 181 us
Australia	Depth first search only	680 / 366 us	599 / 323 us	680 / 365 us	703 / 375 us
	Depth first search + forward checking	0 / 30 us	0 / 40 us	0 / 27 us	0 / 41 us
	Depth first search + forward checking + propagation through singleton domains	0 / 27 us	0 / 26 us	0 / 26 us	0 / 27 us

Table 2: Performance of graph coloring algorithms with Vertex Selection heuristics (i.e. MRV & Degree).
All the results showing in {number of backtrack happened} / {execution time needed} format.

Input Graph	Graph Coloring Algorithms	Run 1	Run 2	Run 3	Run 4
USA	Depth first search only	X	X	X	X
	Depth first search + forward checking	89511 / 1973316 us	8820 / 168692 us	46354 / 34670 us	580161 / 290577 us
	Depth first search + forward checking + propagation through singleton domains	0 / 311 us	1316432 / 1063312 us	781374 / 1751548 us	0 / 335 us
Australia	Depth first search only	533 / 269 us	551 / 444 us	694 / 339 us	650 / 435 us
	Depth first search + forward checking	0 / 31 us	0 / 29 us	0 / 30 us	9 / 48 us
	Depth first search + forward checking + propagation through singleton domains	7 / 210 us	0 / 26 us	0 / 30 us	0 / 27 us

Table 3: Performance of graph coloring algorithms with Value Selection heuristics (i.e. LCV).
All the results showing in {number of backtrack happened} / {execution time needed} format.

Input Graph	Graph Coloring Algorithms	Run 1	Run 2	Run 3	Run 4
USA	Depth first search only	X	X	X	X

	Depth first search + forward checking	81 / 383 us	81 / 282 us	81 / 553 us	99 / 359 us
	Depth first search + forward checking + propagation through singleton domains	207 / 185 us	207 / 199 us	207 / 305 us	207 / 214 us
Australia	Depth first search only	680 / 371 us	599 / 337 us	599 / 328 us	680 / 457 us
	Depth first search + forward checking	0 / 31us	0 / 31 us	0 / 44 us	0 / 31 us
	Depth first search + forward checking + propagation through singleton domains	0 / 27 us	0 / 27 us	0 / 28 us	0 / 27 us

Table 4: Performance of graph coloring algorithms with all the possible heuristics (i.e. MRV, Degree, LCV).
All the results showing in {number of backtrack happened} / {execution time needed} format.

Observations:

1. Even with any heuristic, it is still hard to retrieve a result for USA map by simple backtrack search (obviously within a reasonable time i.e. ~5 minutes). My hypothesis is, the basic backtracking search doesn't perform any domain reduction while assigning a color to a vertex. Thus, heuristics like MRV, LCV can't able to make any intelligent choice for the unassigned vertices. Thus, the impact of the heuristic functions ultimately nullifies and hence made huge tried before any successful assignment.
2. The value selection heuristic (i.e. LCV) perform poorly without the support of vertex selection heuristic. It is quite expected as the Least Constraining Value (LCV) only rule out the fewest value in the remaining variables. Without intelligent selection of vertex, it is hardly retrieving some useful information.
3. Propagation through singleton always provide useful assistance and hence outperform the other algorithms.
4. Applying both the vertex and value selection heuristic outperform all the other combination of algorithm and heuristic function.

Future Works Apply propagation through domain reduction (AC-2/AC-3) with backtrack search.Try implementing other advance algorithms like hill-climbing search, simulated annealing search, genetic algorithm, local and stochastic beam search, etc.

Resources

1. [Web] List of Neighboring States: https://state.1keydata.com/bordering-states-list.php
2. [Web] US State Information: https://people.sc.fsu.edu/~jburkardt/datasets/states/states.html
3. [Web] Heuristics for use in backtracking constraint-satisfaction search: https://people.cs.pitt.edu/~wiebe/courses/CS2710/lectures/constraintSat.example.txt