Map Coloring

Tips

- The program is written in Python2.7
- I use Resident Set Size (RSS) to value the memory requirement of the process

1. Backtracking with MRV, degree, and LCV (MRV)

This is the first algorithm I implemented. Except for these three heuristics, I also implemented forward checking. I assign class “Node” with property “remaining Color Set”. Initially, each node has color set of *k* (3 or 4), each time when the neighbor of this node is assigned some color, this color will be removed from the remaining color set of this node.

2. Backtracking with MAC (MAC)

Along with MAC, I retained MRV, degree and LCV heuristics. The main difference between forward checking and MAC is that forward checking only checks consistency between assigned and non-assigned states. MAC also checks constraints between two unassigned states.

Although MAC outperforms forward checking, because it needs to check too much nodes at each iteration, the time used for MAC may not be shorter than the first algorithm.

Since MAC will find inconsistency earlier than algorithm 1, the memory usage of MAC will be smaller than algorithm 1.

3. Conflict-directed backjumping (BJ)

I still retained MRV, degree and LCV heuristics. Comparing to the first algorithm, backjumping algorithm maintain a conflict set for each node and backtracks to the most recent node in the conflict set. And conflict-directed backjumping modifies backjumping to use conflict sets that include not only the variables with a direct conflict but also any subsequent variables with no consistent solution.

Let Xi be the current variable, and let $conf(X_i)$ be its conflict set. If every possible value for X_i fails, backjump to the most recent variable X_j in $conf(X_i)$, and set

$$conf(X_j) = conf(X_j) \cup conf(X_i) - \{X_j\}.$$

Conflict-directed backjumping outperforms the first algorithm, and because it backtracks logically, the time it used to find a solution or find no sulotion exists will be shorter than the first algorithm.

The memory usage of conflict-directed backjumping will also be smaller than algorithm 1 and algorithm 2.

Results

Color number	node number	average time for generating map (seconds)	average run time (seconds)			RSS (Resident Set Size)		
			MRV	MAC	BJ	MRV	MAC	BJ
k=3	20	0.48	0.0138	0.0193	0.0062	16020	15920	16060
	40	2.10	0.0221	0.4399	0.0211	21644	22080	21948
	60	5.05	0.0461	0.0876	0.0383	26828	26796	27348
	80	10.48	0.0560	0.1299	0.0481	32324	32644	32288
	100	18.06	0.0762	0.1387	0.0630	37384	37380	36816
	150	50.33	0.1582	0.2356	0.1078	50840	50620	50768
k=4	20	0.43	0.0141	0.0256	0.0007	16308	15888	15644
	40	2.01	0.0070	0.1061	0.0009	21736	21816	21748
	60	5.03	0.0198	0.2722	0.0030	27016	26744	27036
	80	10.34	0.1765	0.5221	0.0328	32076	31748	32200
	100	18.56	0.4969	1.4950	0.0408	36856	36956	36912
	150	51.59	0.2787	2.4642	0.1040	50740	51108	50744

For k = 3, map coloring seldom succeeds. But for k = 4, map coloring can always succeed (four color theorem).

From the results, we found it basically consistent with our conjecture: time used: BJ < MRV < MAC. Here I want to explain that my MRV runs shorter time than MAC is because I implemented forward checking for MRV, and it is a property of the node, so MRV uses very short time for forward checking. But for MAC, it needs to call function AC3() to check consistency, and the function runs longer time than embedded property, besides, Arc consistency checks the consistency between two unassigned nodes. Although MAC may find an unconsistency before simple forward checking, it takes too much time for checking each node. Whereas conflict-directed backjumping backtracks more logically so that it takes the least time to find the solution or to find no solution exists.

As for memory requirements, here I compare the RSS (Resident Set Size) of the processes. And the experiments run out that the memory RSS required for the three algorithms are almost the same. It is possibly because all the algorithms apply depth-first search and the node expanded are not remembered by memory.