Empirical evalutation of JSetL Ris implementation

Andrea Fois, Gianfranco Rossi, Maximiliano Cristiá

Università di Parma, Parma, Italy
andrea.fois@studenti.unipr.it

Example / Cardinality	0	1	5	10	25	50	100
8 Ris creation	0.65	1.04	0.65	0.67	0.70	0.71	0.82
9 Ris expansion	0.32	0.61	1.03	1.51	3.38	5.61	10.99
11 Ris constraints	0.67	0.82	0.72	0.73	1.02	0.95	1.19
12 Ris constraint solving	0.77	0.95	1.76	2.86	5.30	8.79	17.26
13 Minimum of a set	0.26	0.40	0.53	1.22	1.49	4.54	7.35
15 Prime test	0.00	0.01	0.24	0.89	6.73	28.39	104.39
20 Squares of evens	0.22	0.78	1.26	1,79	3.77	6.83	12.32
21 Domain restriction	0.11	0.20	0.58	1.52	11.74	75.70	525
23 Reachable nodes	-	0.33	1.00	4.82	88.06	815	11136
Example / Cardinality	0	1	5	10	11	12	13
14 Map coloring	0.05	0.47	5.48	713.55	2036	5940	17680
22 Factorial	0.52	1.28	6.46	18.53	22.94	25.21	-
Example / Cardinality	0	1	2	3	4	5	6
7 Permutations	0.11	0.09	0.22	1.77	28.29	625	17153

Table 1. Summary of the empirical evaluation (times are in msec)

Table 1 shows for each major example program the time it takes for the program to complete with the given number of elements in input. The tests have been conducted on a laptop with Windows 10, 8Gb of Ram, an SSD and an i7-7700hq clocked at 2.69GHz. The laptop was connected to the electrical charge during the execution of the examples. The locked CPU frequency was achieved by setting the minimum and maximum frequency at 99% of the base frequency. This setting disables both underclocking for power saving and the boost of the processor frequency above its base frequency for a limited amount of time depending on temperature, power consumption, number of logical cores used. We used default optimisation options for the JSetL Solver. A zip archive with every program used to test the times in the table, along with the excel file with all the data, means and standard deviations is available at the JSetL website (http://www.clpset.unipr.it/jsetl/). All tests were repeated 22 times and then the first two executions were discarded because they showed an increased computational time which dominated the time for the lower cardinalities. This effect is due to the optimisation systems of the JVM like HotSpot which uses a JIT compiler to recompile the most frequently used Bytecode into optimised

code.
Although we would have preferred to test every example with the same cardinalities it was unfeasible: for 23 - Reachable nodes the definition of the problem given is not reasonable for sets of 0 size (graphs with 0 nodes). For examples 7 - Permutations and 14 - Map coloring it was unfeasible to use a cardinality of 25 for the input size. Example 22 - Factorial did not manage to complete for cardinalities of 13 and above because of limited arithmetic rather than time. Moreover, for some examples a pseudorandom number generator was used to create the input. It is important to note that for 14 - Map coloring the input was a (pseudo-)randomly generated bipartite graph and the solver was asked to provide a 2-coloring of it but there is no optimisation in the JSetL code that accounts specifically for bipartite graphs and, in fact, the data shows an exponential growth of computational time.