Programming Assignment 2 CSE P 573

Alan Ludwig

Algorithm for conversion to SAT

Conversion to SAT was a fairly straightforward process. Of course, the problem centers on building a mapping. I thought of this mapping as an array. It has the number of rows equal to the number of nodes in the email graph. And it has the number of columns is equal to the number of nodes in the phone graph. Each cell in the array equates to a variable in SAT and a true indicates that the node in the mail graph (row) is mapped to the node in the phone graph (column).

All of the logical constraints in the SAT input come from the map array. For each row, there can be at most one true value in the row, and there must be at least one true value for each row. So, for each row the negation of each cell is OR-ed together (pair-wise) with every other cell in the row and all of them are AND-ed together. This ensures that there is no more than one true value in each row. Finally for each row, the entire row is OR-ed together so that at least one must be true. That takes care of the row constraints.

For each column the algorithm is similar. There can be at most one value in each column. Again, this is achieved by OR-ing together the negation of each cell in pairs. This ensures that for no two pairs can both be true. Unlike the rows, not every column will have a value. So no additional constraint is necessary.

The most interesting constraint comes last, and it comes straight out of the definition of the assignment. As stated in the assignment: There is an edge from v1 to v2 in G if and only if there is an edge from M(v1) to M(v2) in G'. It had to be stated a bit differently to convert well to a CNF. Basically if there is a mapping from x to u, and a mapping from y to v and there is an edge in G from x to y, that implies that there is an edge in G' from u to v. Add to that a case that covers where there is not an edge from x to y, and also the cases where the edges are in the other direction and you get the following four cases.

Mxu ^ Myv ^ Gxy → G'uv	Mxu ^ Myv ^ -Gxy →- G'uv
Mxu ^ Myv ^ Gyx → G'vu	Mxu ^ Myv ^ -Gyx → -G'uv

These are converted into conjunctions. The next step is to consult the first two arrays to determine if the referenced edges are present. If after the edge values are resolved the conjunction is not yet satisfied, then the remaining terms in the conjunction are written into the SAT input file. The final result is that the SAT input file contains conjunctions that depend solely on the values for the mapping array.

Experimental Results for Scalability

One of the themes in the class has been that space is more of a constraint than time is. Generally speaking we can wait a while for a result, but if it requires more memory or drive space than is available there isn't as much you can do about it. The largest consumer of space is the SAT input file, And the SAT input file is a measure of the number of constraints. And the growth of this file (and the number of constraints) is somewhat troubling. In rough terms, an order of magnitude change in the count of the nodes in our graphs from 10/5 to 100/50 resulted in six orders of magnitudes of change in the size of the SAT input file from hundreds of bytes to hundreds of megabytes. My theoretical estimate on the number of clauses in the SAT input file is that it is approximately O((gg')^2) where g is the number of nodes in G and g' is the number of nodes in G'.

Nodes in G	Edges in G	Nodes in G'	Edges in G'	Variables	Constraints	SAT-in Size	Time to Solve
10	20	5	10	50	1,230	128 b	.001 s
25	300	10	50	250	31,565	343 kb	.032 s
50	1,000	25	500	1,250	885,850	10.3 MB	6.77 s
100	2,500	50	1,000	5,000	11,357,700	146 MB	458 s
200	5,000	100	1,500	20,000	96,393,500	1.30 GB	INDETERMINANT

Time in this case appears to be growing similarly but not quite as fast as space. Looking again at the chart an order of magnitude change in node size from 10/5 to 100/50 resulted in five orders of magnitudes of change in time from 1 millisecond to hundreds of seconds.

However, the ultimate barrier to scalability in this case may be minisat itself. The only result I could get for the largest problem was INDETERMINANT. In fact, five different configurations of graphs I attempted with nodes between 100/50 and 200/100 also resulted in KILLED or INDETERMINANT runs. It would appear that 100/50 is near the limit.

Avenues for Continued Progress

I'd like to look at ways to improve the scalability of this problem within the constraints that we have today. Doing that would require that we reduce either the number of variables or the number of constraints. Reducing the number of variables seems difficult. However, It seems reasonable that through some kind of data compression that we could get a reduction. However, this is sure to add some number of constraints.

Reducing constraints seems like the most likely avenue. Many constraints come from the row and column constraints. Perhaps a more efficient encoding than pair-wise logic could be found to eliminate duplicate mappings in rows and columns. This might be accomplished by introducing more variables and building a binary counter to ensure that there is at most one mapping in each row or column.

I have suggested two ways that we might reduce the number of variables or the number of constraints. Each of these resulted in trading one for the other. Such a trade would be valuable if it resulted in an overall encoding of the problem that was somehow more efficient. This leads me to then ask if there is some theory available to help guide when valuable trade-offs exist and what forms they take.