7
Numerical Results

The code was implemented in C++ and was run in a Core2Quad Q9300 with 8GB of DDR2 RAM, using the CSDP solver for SDP[1].

The code was tested with the instances from the Biq Mac Library[2], and the results were compared to the Biq Mac Solver[3] [rendl2010], which is the best available solver, to our knowledge.

The results for the instances with 60 variables are as following:

Instance	Nodes		Time	
	Biq Mac	ours	Biq Mac	ours
g05_60.0	3	274	7.18s	10.58s
g05_60.1	3	7	4.32s	1.60s
g05_60.2	15	25	26.5s	10.67s
g05_60.3	1	6	0.83s	1.22s
g05_60.4	33	1061	53.12	70.87
g05_60.5	1	4	0.98s	0.98s
g05_60.6	13	20	24.82s	2.01s
g05_60.7	7	291	14.67s	10.93s
g05_60.8	13	244	19.88s	10.40s
g05_60.9	21	536	34.08s	19.01s

As it can be seen, our code generates much more nodes, but solves each node much faster.

This behavior is also observed in the instances with 80 variables:

[1]https://projects.coin-or.org/Csdp/
[2]http://biqmac.uni-klu.ac.at/biqmaclib.html
[3]http://biqmac.uni-klu.ac.at/

Instance	Nodes		Time	
	BiqMac	ours	Biq Mac	ours
g05_80.0	59	169	193.11s	20.00s
g05_80.1	3	36	13.35s	7.54s
g05_80.2	17	811	59.41s	72.43s
g05_80.3	523	7423	1467.26s	567.42s
g05_80.4	39	1750	141.63s	151.35s
g05_80.5	65	1147	207.45s	103.77s
g05_80.6	31	152	107.03s	18.42s
g05_80.7	23	682	75.44s	66.83s
g05_80.8	73	341	225.19s	36.29s
g05_80.9	157	389	453.07s	42.27s

When the number of variables gets larger, our code runs slower than Biq Mac's. This is the table for some big instances:

Instance	size	Nodes		Time	
		BiqMac	ours	Biq Mac	ours
ising3.0-200_6666	200	11	1046	661.02s	1432.27s
ising3.0-200_7777	200	13	1107	790.25s	1325.10s
ising3.0-300_6666	300	23	1421	4236.84s	9123.12s
ising3.0-300_7777	300	39	3121	7298.02s	13214.43s
t3g7_7777	343	81	4023	11072.86s	31235.21s
t2g20_7777	400	13	2751	6605.46s	9927.12s

As it can be seen, our code is substantially slower than the Biq Mac Solver, but we also generate a huge amount of nodes, which can be used in parallel environments, since solving the nodes is almost completely independent. There is no trivial way to parallelize Biq Mac as well as we did with our solution because of the small number of nodes.

The results of the parallelization are shown in the next session.

7.1 Parallel Computation

To experiment with parallelization we used the Amazon EC2[4] cloud computing solution. The virtual cores are equivalent to a 3GHz 2007 Xeon processor. We used 8-cores virtual computers.

To implement the parallelization we modeled the CPUs in a binary tree structure, so that whenever a CPU runs out of nodes to process, it asks for its parent for more. When the root runs out of nodes, it asks for each of its children, who either sends back some of each nodes or pass the request down the tree.

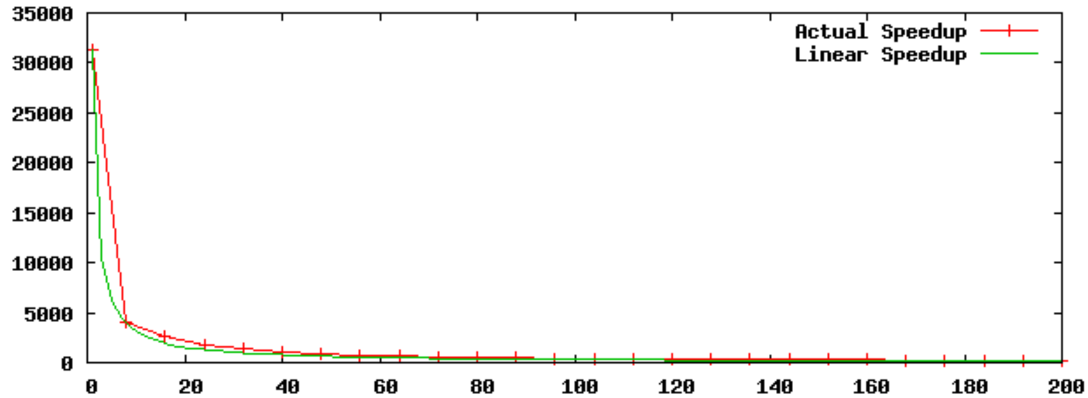
The message passing was implemented using Thrift[5] RPC solution.

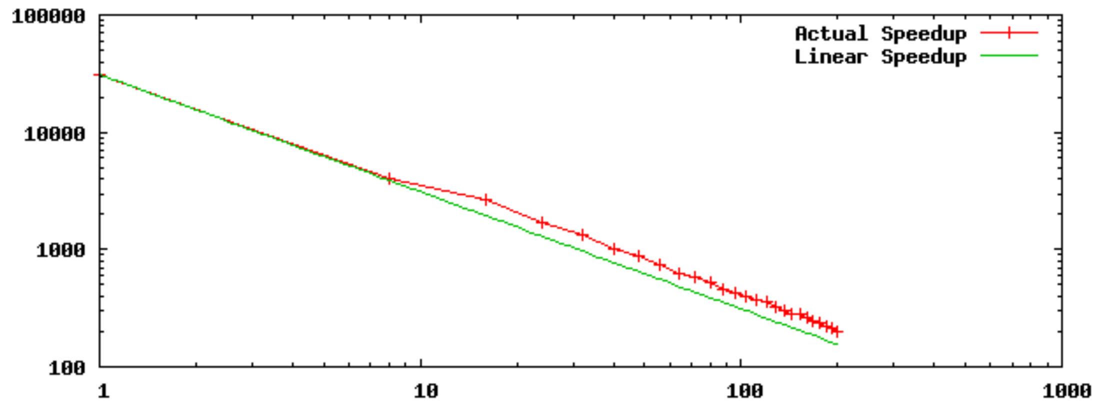
Below we have the result for the instance t3g7_7777:

#cores	time
1	31235.21
8	4023.00s
32	1327.61s
56	733.92s
80	524.92s
104	405.02s
128	322.02s
152	277.17s
176	238.12s
200	200.05s

This result can be better seen in the following graph, which shows the result with a full linear speedup, and the result achieved. The two graphs show exactly the same data, but the second one shows in a log-log scale.

[4] http://aws.amazon.com/ec2/
[5] http://incubator.apache.org/thrift/





If we analyze the graphs, we find out that the time gets a little worse after 8 cores. This is understandable, since until 8 cores we were in a shared memory environment. But it does not get worse if we further increment the number of CPUs.

As we can see, for big problems this method is very parallelizable with a near-linear speedup, as expected.

7.2 Greedy Heuristics

The greedy heuristics described on section 6.4 got some great results. The linear programming solver used was Gurobi[6]

First of all, the running time was below 30 seconds for all 110 instances in the Biq Mac Lib. In 25% of the instances the optimum was reached, and in 90% of them, the result was within 92% of the best known solution. The worst result reached 87% of the best known.

Here is a table with the results for some of the instances:

Instance	Heuristics	Optimum	Heuristics/Optimum
t2g15_5555	13430805	15051133	0.892345
t2g10_7777	5886888	6509837	0.904307
g05_100.9	1423	1430	0.995105
g05_80.5	922	926	0.995680
g05_100.1	1425	1425	1.000000
g05_80.9	923	923	1.000000
pw05_100.9	8099	8099	1.000000

[6] www.gurobi.com