Linear Domino Game Benchmark*

Randal E. Bryant

Computer Science Department
Carnegie Mellon University, Pittsburgh, PA, United States
Randy.Bryant@cs.cmu.edu

1 Description

Two-player games provide a rich set of benchmarks for QBF solvers, with each turn being translated into a quantification level. To encode the game from the perspective of the first player (Player A), A's turns are encoded with existential quantifiers, while the second player's (Player B) turns are encoded with universal quantifiers. The formula will be true if the game has a guaranteed winning strategy for A. The encoding of a game into QBF constrains the two players to only make legal moves. It also expresses the conditions under which A is the winner, namely that the game consist of t consecutive moves, for an odd value of t. Conversely, we can encode the formula where B has a winning strategy by reversing the quantifiers and expressing that the game must consist of an even number of consecutive moves. For a game where no draws are possible, these two formulas will be complementary.

Consider a game played on a $1 \times N$ grid of squares with a set of dominos, each of which can cover two squares. Players alternate turns, each placing a domino to cover two adjacent squares. The game finishes when no more moves are possible, taking at most $\lfloor N/2 \rfloor$ turns. The first player who cannot place a domino loses. This *linear domino placement* game is isomorphic to the object-removal game "Dawson's Kales" [1]. It can be shown that player B has a winning strategy for $N \in \{0, 1, 15, 35\}$ as well as for all values of the form $34\,i + c$ where $i \geq 0$ and $c \in \{5, 9, 21, 25, 29\}$ [11].

The game is encoded as a QBF by introducing a set of $N-1$ input variables for each possible move, each corresponding to the boundary between a pair of adjacent squares. A set of $N-1$ definition variables encodes the board state after each move, and sets of clauses enforce the conditions that 1) each move should cover exactly one boundary, and 2) neither that boundary nor the two adjacent ones should have been covered previously. In all, there are around $N^2/4$ universal input variables, $N^2/4$ existential input variables, and $3N^2/2$ definition variables. The number of clauses grows as $\Theta(N^3)$ due to the quadratic number of clauses to enforce the exactly-one constraints on the input variables for each move.

In generating a QBF problem with definition variables, these variables must be existentially quantified. They can be placed at any level beyond the last universal variable on which they depend, either directly or indirectly. The common practice is to simply place them in an innermost quantifier level. We refer to this as the "end" placement. However, it can be advantageous to place them immediately after any variables used in

* This work was supported by the U. S. National Science Foundation under grant CCF-2108521

their definitions. We refer to this as the "after" placement. In other work, we showed that shifting definition variables earlier can improve the performance of other QBF solvers for many QBF problems [10].

2 Obtaining Benchmarks

A benchmark generator, files, and experimental evaluation are all available at:

`https://github.com/rebryant/linear-domino-game`

For values of N from 15 to 25, there are four QBF files, having both combinations of the formula being true and false and the defintion variable placement following the "after" or "end" conventions.

3 Experimental Evaluation

Experimental results for this benchmark have been presented for the BDD-based QBF solver PGBDDQ [2]. Like other BDD-based solvers [8, 9], PGBDDQ operates by applying *bucket elimination* to the variables [3] from the innermost quantification level to the outermost one. That is, BDD representations of the input clauses are generated and placed into buckets according to their highest variables (in the quantifier ordering). Then the program repeatedly forms the conjunction of all BDDs in the next highest bucket, computes either its universal or existential quantification with respect to the variable, and places the result in the approriate bucket. The final result will be the BDD representation of either true or false. For a QBF encoding of a game, it can be seen that this approach simulates all possible games starting from the end and working backward to the start. PGBDDQ can generate a checkable proof of its results in the standard QRAT proof framework [4]. For a false formula, it will produce a satisfaction proof, consisting of a series of truth-preserving steps leading to the empty clause, indicating falsehood. For a true formula, it will produce a satisfaction proof, consisting of a series of falsehood-preserving steps leading to an empty set of clauses, indicating tautology.

Our previous work shows that the time required for PGBDDQ can have polynomial scaling on for the linear domino game. Doing so requires two refinements. First, the definition variables must be place according to the "after" convention, immediately after the input variables on which they depend. Second, the BDD ordering of the variables must be orthogonal to their quantifier ordering, with all of the variables for a particular board position grouped together, even though they span multiple quantification levels.

We also experimented with two existing QBF solvers: DEPQBF, version 6.0 [7], and GHOSTQ [5,6]. In the following, measurements were performed on a 4.2 GHz Intel Core i7 (I7-7700K) processor with 32 GB of memory running the MacOS operating system. Times are measured in elapsed seconds. None of the solvers were configured to generate proofs, but they all got correct results.

Table 1 shows the times required to run PGBDDQ on the benchmarks. As can be seen, it can readily handle formulas where the definition variables have been placed near their defining variables, but not when they are placed at the innermost level. It is fairly insensitive to whether the formula is true or false.

Table 1. Performance of PGBDDQ on Linear Domino Benchmarks. For each version of N, the formula is either true or false, and definition variables can be placed either after their defining variables or at the end. "TO" indicates a 1000-second timeout.

N	Winner	true after	false after	true end	false end
15	B	3.7	4.7	TO	TO
16	A	7.6	7.9	TO	TO
17	A	12.6	12.9	TO	TO
18	A	16.3	12.9	TO	TO
19	A	11.9	10.6	TO	TO
20	A	15.4	16.1	TO	TO
21	B	22.9	22.8	TO	TO
22	A	33.3	26.7	TO	TO
23	A	27.4	23.8	TO	TO
24	A	23.2	23.6	TO	TO
25	B	35.1	33.0	TO	TO

Table 2. Performance of DEPQBF on Linear Domino Benchmarks

N	Winner	true after	false after	true end	false end
15	B	4.4	4.4	4.6	4.5
16	A	15.2	51.9	15.0	34.0
17	A	7.1	17.5	6.4	11.2
18	A	315.9	128.9	190.0	77.4
19	A	520.7	212.6	500.8	167.7
20	A	TO	TO	TO	TO
21	B	TO	TO	TO	TO
22	A	TO	TO	TO	TO
23	A	TO	TO	TO	TO
24	A	TO	TO	TO	TO
25	B	TO	TO	TO	TO

Table 2 shows the performance of DEPQBF for the benchmark problems. As can be seen, it is relatively insensitive to variable placement. On the other hand, it is sensitive to whether or not the formula is true, but not in a predictable way. It is unable to scale beyond $N = 19$ within the 1000 second time limit.

Table 3. Performance of GHOSTQ on Linear Domino Benchmarks

N	Winner	true after	false after	true end	false end
15	B	1.0	1.2	1.1	1.0
16	A	2.4	1.0	2.7	1.4
17	A	2.7	4.4	3.4	3.5
18	A	6.7	6.1	4.6	3.6
19	A	14.8	16.6	20.6	7.5
20	A	48.1	47.3	55.3	39.3
21	B	117.8	109.2	112.8	95.6
22	A	267.8	287.1	249.8	90.3
23	A	467.6	413.1	544.0	380.9
24	A	600.0	TO	692.7	TO
25	B	TO	TO	TO	TO

Table 3 shows the performance of GHOSTQ for the benchmark problems. Overall, it outpeforms DEPQBF, scaling to $N = 24$ for false formulas. Nonetheless, it cannot achieve the scaling of PGBDDQ.

Overall, we can see that PGBDDQ achieves polynomial scaling, with the time growing slowly as N increases. By contrast, both DEPQBF and GHOSTQ scale exponentially. Over a short range of sizes, they shift from running in just a few seconds to exceeding the time limit.

References

1. Berlekamp, E.R., Conway, J.H., Guy, R.K.: Winning Ways for your Mathematical Plays: Volume 1, Second edition. CRC Press (2001)
2. Bryant, R.E., Heule, M.J.H.: Dual proof generation for quantified Boolean formulas with a BDD-based solver. In: Conference on Automated Deduction (CADE). LNAI, vol. 12699, pp. 433–449 (2021)
3. Dechter, R.: Bucket elimination: A unifying framework for reasoning. Artificial Intelligence **113**(1–2), 41–85 (1999)
4. Heule, M.J.H., Seidl, M., Biere, A.: A unified proof system for QBF preprocessing. In: International Joint Conference on Automated Reasoning (IJCAR). LNCS, vol. 8562, pp. 91–106 (2014)
5. Janota, M., Klieber, W., Marques-Silva, J.a.P., Clarke, E.M.: Solving QBF with counterexample guided refinement. Artificial Intelligence **234**, 1–25 (2016)
6. Klieber, W.: GhostQ system description. Journal on Satisfiability, Boolean Modeling, and Computation **11**, 65–72 (2019)

7. Lonsing, F., Egly, U.: DepQBF 6.0: A search-based QBF solver beyond traditional QCDCL. In: Conference on Automated Deduction (CADE). LNCS, vol. 10395, pp. 371–384 (2017)
8. Olivo, O., Emerson, E.A.: A more efficient BDD-based QBF solver. In: Principles and Practice of Constraint Programming (CP). LNCS, vol. 6876, pp. 675–690 (2011)
9. Pan, G., Vardi, M.Y.: Symbolic decision procedures for QBF. In: Principles and Practice of Constraint Programming (CP). LNCS, vol. 3258, pp. 453–467 (2004)
10. Reeves, J.E., Heule, M.J.H., Bryant, R.E.: Moving definition variables in quantified boolean formulas. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS). LNCS (2022)
11. Sloane, N.J.A., The OEIS Foundation: The on-line encyclopedia of integer sequences (2012), `http://oeis.org/A215721`, sequence A215721