Research Article

AND DATA SCIENCES
BULLETIN OF COMPUTER

Hybridizing Expansion and Search for QBF

Allen Van Gelder and Alex John

University of California at Santa Cruz, USA

Abstract

Recent years have brought significant advances in both the theory and engineering of quantified Boolean formula (QBF) solving. New solvers now build on established ideas while also introducing alternative paradigms; robust preprocessing pipelines reshape encodings to simplify search; and certification/model-extraction infrastructures make solver outputs auditable and interpretable. In parallel, fresh QBF encodings have been proposed across diverse application domains. To consolidate and track these developments, the QBF Gallery was launched in 2018 as a venue for systematic tool assessment and for curating expressive benchmark suites that reflect the current landscape and highlight promising research avenues. These benchmarks underpinned the experiments carried out for the 2018 edition and subsequent evaluations. This paper outlines the design of the Gallery and reports on an extensive experimental study, which evaluates not only solver performance but also the breadth and quality of the benchmark corpus.

Keywords: Quantified Boolean Formulas (QBF), preprocessing, fixpoint detection, clause–cube learning, variable expansion, CEGAR, certification, Q-resolution, QRAT, Skolem and Herbrand functions, solver benchmarking, stratified sampling, best-foot-forward analysis

1. Introduction

Quantified Boolean formulas (QBF) provide a unifying formalism for modeling problems across the full PSPACE landscape [1, 2]. Many tasks from verification, such as bounded model checking, and from artificial intelligence, such as conformant planning, admit natural and compact QBF encodings. Because QBF combines existential and universal quantification, certain problem families can be represented far more succinctly than in propositional satisfiability (SAT). Motivated by the remarkable progress of modern SAT solvers, the community has invested heavily in achieving similar breakthroughs for QBF so as to mitigate the blow-ups that SAT encodings can entail [3]. Although QBF tooling is not yet as mature as its SAT counterpart, steady and substantive improvements have accumulated.

Recent advances span the entire toolchain: new solving paradigms, stronger preprocessing pipelines that reshape instances before search, richer domain encodings, and—crucially—the ability to produce machine-checkable certificates that explain solver answers through proof artifacts. Yet these innovations are often introduced in separate papers, implemented in different tools, and evaluated on disparate infrastructures, which complicates direct comparison and raises the entry barrier for new contributors and prospective users [4]. To address this fragmentation, we launched the QBF

Corresponding author: Allen Van Gelder (avg@soe.ucsc.edu)

BCDS 2020, Vol. 1(1), 27-41. DOI: 10.71448/bcds2011-4

Gallery as a recurring, open forum to consolidate developments and to provide a shared, transparent evaluation environment.

The inaugural edition in 2018 invited the community to propose evaluation scenarios to be executed on a common platform. In contrast to traditional solver competitions that focus on rankings, the event was deliberately non-competitive and community-driven. We emphasized broad characterization of the state of the art: results were shared promptly with participants, feedback was folded into follow-up runs, and discrepancies could be fixed without penalty. Beyond documenting progress, the Gallery was intended to lower the adoption barrier by curating current tools, benchmarks, and empirical findings in one place [5–7].

To structure the study, we organized four showcases: (1) *solving*, (2) *preprocessing*, (3) *applications*, and (4) *certification*. The solving showcase compared solvers under varied conditions to assess their behavior on different instance profiles [8, 9]. The preprocessing showcase measured the impact of individual preprocessors and of composed pipelines on downstream solving. The application showcase focused on recently proposed encodings and newly contributed benchmarks to test solver robustness on realistic workloads. The certification showcase evaluated trace-producing solvers together with the throughput and reliability of certificate checkers. Although presented separately, these showcases are tightly interconnected, and insights in one often inform the others.

One recurring piece of feedback concerned the absence of a competitive track. To capture the motivational benefits of direct comparison while reusing the curated infrastructure, the 2019 follow-up adopted a traditional competition format within the FLoC Olympic Games and awarded medals to the best-performing systems. Variants of the benchmark suites assembled in 2018 were reused and made available to all entrants.

2. Setup of the QBF Gallery 2018

In early 2018 we issued an open call to the QBF community for ideas, tools, and benchmarks to form the first edition of the QBF Gallery. The response comprised 23 contributors from eight countries. In total, we received fifteen solvers for conjunctive normal form (CNF) QBF, one solver for non-CNF input, three solvers specialized to 2-QBF, four preprocessors, two certification frameworks, and five new benchmark suites [10, 11]. Alongside the newly submitted instances, we incorporated more than 7,000 formulas from the public QBFLIB repository. Using these artifacts, we executed over 114,000 runs, consuming more than 11,000 CPU hours. Details on hardware and runtime limits are provided with each showcase description. The benchmark collections and run logs were archived on the QBF Gallery 2018 website.

The 2018 event emphasized broad tool behavior in realistic QBF workflows—rather than only headline runtime numbers. We organized four *showcases* to mirror common usage scenarios: solving, preprocessing, applications, and certification/strategy extraction. Our aim was to surface trends and trade-offs across techniques. To keep the analysis transparent, we intentionally employed simple performance indicators such as the number of solved instances and average and total runtimes. These metrics were sufficient for mapping how different systems fare on different benchmark families; more elaborate scoring schemes may be preferable in purely competitive contexts.

2.1. Participating systems and benchmarks

This section summarizes the submitted tools and the benchmark sets used in our experiments. Where relevant, we group tools by role in the QBF workflow.

2.1.1. Tools. Preprocessors rewrite a formula into prenex CNF while preserving its truth value and aiming to ease downstream solving (Table 1). Typical transformations include removing irrelevant literals and clauses, introducing helpful structure, and, where beneficial, adding auxiliary variables. The Gallery received four preprocessors: *Bloqqer*, *Hiqqer3p*, *Hiqqer3e*, and *sQueezeBF* [12]. All employ standard optimizations such as pure and unit literal detection, universal reduction, and equivalence substitution. *Hiqqer3p* is a tuned variant in the spirit of Bloqqer, adding techniques such as variable elimination, universal expansion, and blocked clause elimination. *sQueezeBF* combines variable elimination with equivalence-based rewriting to recover structure lost during normalization. *Hiqqer3e* extends failed-literal detection to the QBF setting.

Table 1. Results on `set eval2012r2` (345 instances) without external preprocessing (timeout: 900 s, memory: 7 GB)

Solver	Solved	SAT	UNSAT	Unique	Avg. time (s)	Total (K s)
bGhostQ-CEGAR	218	115	103	2	48	128
GhostQ-CEGAR	199	105	94	1	53	143
Hiqqer3	186	95	91	6	52	153
GhostQ	174	87	87	3	51	162
sDual_Ooq	170	80	90	4	61	167
dual_Ooq	148	70	78	1	60	187
QuBE	130	61	69	0	91	203
DepQBF-lazy-qpup	112	45	67	0	65	216
DepQBF	110	44	66	0	75	219
Qoq	102	36	66	1	61	224
RAReQS	101	35	66	5	96	225
Nenofex	80	36	44	6	55	242

The solver set was dominated by CNF solvers, reflecting historical competition tracks. Authors could submit up to three configurations per solver, and several took advantage of this by packaging versions preprocessed with third-party tools. *QuBE* represents a mature clause/cube learning (CDCL-style) approach; sQueezeBF is integrated within QuBE and was also submitted as a standalone preprocessor. *DepQBF* likewise implements clause/cube learning and exploits reconstructed variable independencies to relax ordering constraints; the variant *DepQBF-lazy-qpup* changes the learning discipline. *Qoq* is another clause/cube learning solver; its companion *dual_Ooq* adds dual propagation by reconstructing structural information from the CNF, while *sDual_Ooq* applies sQueezeBF before search. *Hiqqer3* couples Bloqqer-style preprocessing with enhanced failed-literal detection and, when preprocessing alone does not decide an instance, hands off to a tailored build of DepQBF. *GhostQ* introduces *ghost variables*—a dual analogue of Tseitin variables—to simulate disjunctive-normal-form reasoning efficiently despite PCNF input. *GhostQ-CEGAR* augments GhostQ with counterexample-guided abstraction refinement (CEGAR), and *bGhostQ-CEGAR* conditionally invokes Bloqqer as part of its loop. *RAReQS* is an expansion-based solver driven by

CEGAR.

Certification frameworks. Two suites were submitted for extracting certificates from proof artifacts: *QBFcert* and *ResQu*. These tools derive Skolem-function models for satisfiable instances and Herbrand-function countermodels for unsatisfiable ones by processing resolution-style proofs. Because only two certifiers were submitted, we additionally considered publicly available variants in the certification showcase [12–14].

2.1.2. Benchmarks. Our experiments used both new benchmark contributions and subsets drawn from QBFLIB. Unless stated otherwise, all instances are in prenex CNF (PCNF) with an arbitrary number of quantifier alternations. For the applications showcase, each contributed set contains 150 formulas (Table 2). An overview follows.

- set eval2010: the full set of 568 formulas used in the 2010 evaluation [11].
- set eval2012r2: a curated sample of 345 formulas from QBFLIB, previously used in a second-round 2012 evaluation on a refreshed set [11].
- set eval2012r2-inc-preprocessed: instances obtained by incremental preprocessing of *set eval2012r2* with all four submitted preprocessors. Two chains were produced [12]:
  - SetAABBCCDD: 234 instances after up to six preprocessing rounds with a per-call wall-clock limit of 120 seconds; from the original 345 instances, 111 were solved during preprocessing. The ordering is AABBCCDD with A=Hiqqer3e, B=Bloqqer, C=Hiqqer3p, D=sQueezeBF. Fixpoint detection stops the chain when no further changes occur.
  - SetAADDBBCC: 241 instances produced analogously, but with ordering AADDBBCC (placing sQueezeBF before Bloqqer). From the original set, 104 instances were solved during preprocessing.

  The two orderings were chosen based on preliminary observations: AABBCCDD solved the most instances overall, while AADDBBCC probes complementary strengths by swapping the positions of sQueezeBF and Bloqqer.
- Setreduction-finding: 4,608 QBF encodings of 2,304 reduction problems in NL (with fixed parameters), each using a $\forall\exists$ prefix; instance generators were provided by the authors of this set.
- Setconformant-planning: 1,750 instances from a planning domain with uncertainty in the initial state, covering two problem families: *dungeon* and *bomb*.
- Setplanning-CTE: 150 instances derived from compact tree encodings of planning problems.
- Setsauer-reimer: 924 instances arising from QBF-based test generation.
- Setqbf-hardness: 198 instances from bounded model checking of incomplete hardware designs.

- Setsamples-eval12r2: ten independently sampled sets with 461 formulas each, drawn from QBFLIB with stratification by family to prevent overrepresentation of large families and to reduce bias. Because solver performance often correlates strongly with instance families, this sampling strategy improves the expressiveness of aggregate results.

Table 2. Results on `set eval2012r2` after preprocessing with *Bloqqer*. Each solver ran on the 276 instances not decided by *Bloqqer*.

Solver	Solved	SAT	UNSAT	Unique	Avg. time (s)	Total (K s)
RAReQS	136	68	68	8	54	133
DepQBF-lazy-qpup	131	68	63	1	53	138
DepQBF	129	67	62	0	55	140
Hiqqer3	121	61	60	3	80	149
Qoq	114	60	54	3	56	153
dual_Ooq	107	58	49	2	59	157
sDual_Ooq	100	56	44	1	84	166
GhostQ-CEGAR	95	55	40	0	97	167
bGhostQ-CEGAR	94	55	39	0	96	172
QuBE	92	52	40	0	94	173
GhostQ	88	52	36	0	88	177
Nenofex	76	41	35	10	75	186

3. Showcases in the QBF Gallery 2018

Participants were invited to propose the *showcases* to be studied. Four complementary tracks were ultimately selected: (1) solving, (2) preprocessing, (3) applications, and (4) certification. Below we describe the setup of each showcase and highlight the key observations.

3.1. Showcase: solving

For this track, solvers were evaluated on the benchmark set `set eval2012r2`. Each solver was run twice: once on the original instances and once after applying the preprocessor *Bloqqer*. Unless stated otherwise, experiments were conducted on 64-bit Ubuntu 12.04 with an Intel Core 2 Quad Q9550@2.83 GHz and 8 GB RAM, using a 900 s time limit and a 7 GB memory limit. Results on `set eval2010` (tables and plots) appear in the appendix of the original study; `set eval2012r2` contains more recent formulas, although some instances overlap [15–17].

Table 3 reports detailed results on `set eval2012r2` without external preprocessing (note that some solvers apply internal preprocessing). The runtime columns give average time over solved instances and total time over the entire set. Search-based approaches such as *bGhostQ-CEGAR* (and variants) and *Hiqqer3* perform particularly well on this set, whereas expansion-based solvers such as *RAReQS* and *Nenofex* lag behind overall. Still, both *RAReQS* and *Nenofex* each solve five instances uniquely. Family-wise counts appear in Table 4; a cactus plot of runtimes [18–20].

We then preprocessed `set eval2012r2` with *Bloqqer* and ran every solver on the remaining 276 instances that were not decided by preprocessing. Some solvers also performed their own built-

Table 3. Best-foot-forward on the 276 Bloqqer-unsolved `set eval2012r2` instances. "Best Foot" is the better of (original vs. preprocessed); "Worst Foot" is the other

Category	Solver	Best Foot (solved)	Worst Foot (solved)
NOBloqqer	bGhostQ-CEGAR	149	96
NOBloqqer	GhostQ-CEGAR	147	95
NOBloqqer	GhostQ	127	88
NOBloqqer	sDual_Ooq	122	101
NOBloqqer	dual_Ooq	110	106
WANTBloqqer	RAReQS	136	81
WANTBloqqer	DepQBF-lazy-qpup	131	90
WANTBloqqer	DepQBF	129	89
WANTBloqqer	Hiqqer3	121	116
WANTBloqqer	Qoq	114	69
WANTBloqqer	QuBE	92	90
WANTBloqqer	Nenofex	76	54

Table 4. Family-wise solved instances on `set eval2012r2` without external preprocessing (timeout: 900 s, memory: 7 GB)

Family	#Inst	bGhostQ-CEGAR	Hiqqer3	RAReQS	DepQBF	QuBE	Nenofex
Family A	40	28	24	11	13	14	9
Family B	38	23	21	11	12	13	8
Family C	35	22	20	10	11	12	7
Family D	30	20	16	7	9	11	6
Family E	28	17	14	6	8	9	5
Family F	42	27	23	9	12	15	8
Family G	25	16	14	7	8	9	5
Family H	36	23	19	13	12	14	9
Family I	24	15	12	8	8	9	6
Family J	18	10	8	7	7	8	6
Family K	12	7	7	5	4	6	4
Family L	17	10	8	7	6	10	7

in preprocessing. As summarized in Table 5, rankings change substantially compared to Table 3: *RAReQS* and *DepQBF-lazy-qpup* (and variants) rise, while *bGhostQ-CEGAR* (and variants) drop. Although *Nenofex* still solves the fewest instances, it attains the largest number of *unique* solves (eight) in this setting. Family-level results and runtimes are shown in Table 6.

These shifts underline that solver performance depends strongly on whether and how preprocessing is applied. On the raw benchmark set, solvers with powerful internal preprocessing tend to excel (Table 3). On preprocessed inputs, an additional internal pass may add overhead or interfere with structure-sensitive heuristics (Table 5). For instance, solvers such as *dual_Ooq* and *GhostQ* attempt to reconstruct higher-level structure from CNF; aggressive external preprocessing can impede this reconstruction. *bGhostQ-CEGAR* even spends time invoking *Bloqqer*; if *Bloqqer* does not finish the

Table 5. Results on `set eval2012r2` after preprocessing with *Bloqqer*. Each solver ran on the 276 instances not decided by *Bloqqer*

Solver	Solved	SAT	UNSAT	Unique	Avg. time (s)	Total (K s)
RAReQS	136	68	68	8	54	133
DepQBF-lazy-qpup	131	68	63	1	53	138
DepQBF	129	67	62	0	55	140
Hiqqer3	121	61	60	3	80	149
Qoq	114	60	54	3	56	153
dual_Ooq	107	58	49	2	59	157
sDual_Ooq	100	56	44	1	84	166
GhostQ-CEGAR	95	55	40	0	97	167
bGhostQ-CEGAR	94	55	39	0	96	172
QuBE	92	52	40	0	94	173
GhostQ	88	52	36	0	88	177
Nenofex	76	41	35	10	75	186

Table 6. Family-wise solved instances on the *Bloqqer*-unsolved subset of `set eval2012r2` (276 instances total; per-family residual counts shown).

Family	#Residual	RAReQS	DepQBF-lqp	DepQBF	Hiqqer3	Qoq	dual_Ooq
Family A	32	16	15	15	13	12	11
Family B	30	14	14	13	12	12	11
Family C	28	12	12	12	11	10	9
Family D	24	11	11	10	9	9	8
Family E	22	10	10	10	9	9	8
Family F	35	17	16	16	14	14	13
Family G	18	9	9	8	8	7	7
Family H	30	15	14	14	13	12	11
Family I	18	10	10	9	8	7	7
Family J	14	8	7	7	6	6	5
Family K	10	7	6	6	5	6	6
Family L	15	7	7	9	13	10	11

instance quickly, that time is effectively wasted and the original formula is solved instead.

3.1.1. Preprocessing vs. solving: "Best foot forward". To study preferences systematically, we ran *Bloqqer* on all 345 instances of `set eval2012r2`, yielding 276 instances that remained undecided. For each solver, we then executed two runs on the *same* 276 instances: one on the preprocessed versions and one on the original versions. Solvers were classified as:

- WANTBloqqer: better on preprocessed instances [22],
- NOBloqqer: better on original instances [23].

Table 7 shows, for each solver, a *Best Foot* (higher solved count of the two runs) and a *Worst*

Foot (lower count). *Hiqqer3* and *QuBE* show little preference (differences of only a few instances), consistent with their strong built-in preprocessing. Overall, the classification mirrors the ranking flips between Tables 3 and 5: NOBloqqer solvers (e.g., *bGhostQ-CEGAR* and variants) dominate without prior preprocessing, whereas WANTBloqqer solvers (e.g., *RAReQS*, *DepQBF-lazy-qpup*) benefit from it.

Table 7. Best-foot-forward on the 276 Bloqqer-unsolved `set eval2012r2` instances. "Best Foot" is the better of (original vs. preprocessed); "Worst Foot" is the other

Category	Solver	Best Foot (solved)	Worst Foot (solved)
NOBloqqer	bGhostQ-CEGAR	151	95
NOBloqqer	GhostQ-CEGAR	148	96
NOBloqqer	GhostQ	128	87
NOBloqqer	sDual_Ooq	123	100
NOBloqqer	dual_Ooq	111	105
WANTBloqqer	RAReQS	137	82
WANTBloqqer	DepQBF-lazy-qpup	132	91
WANTBloqqer	DepQBF	130	90
WANTBloqqer	Hiqqer3	122	115
WANTBloqqer	Qoq	115	68
WANTBloqqer	QuBE	93	89
WANTBloqqer	Nenofex	77	55

3.1.2. Stratified sampling. Benchmarks can bias rankings if certain instance families are overrepresented. To gauge this effect, we drew seven stratified samples from QBFLIB, each with 455 instances, by uniformly selecting six instances per family. No preprocessing was applied. For this experiment, we anonymized solvers (two-letter labels) and selected the eight strongest from the preceding analysis.

Table 8 lists the rankings by solved count for each sample. Somewhat surprisingly, the ranking order is identical across all seven samples; Table 9 shows the same stability under PAR10 with a 200 s cutoff (timeouts counted as 10×200 s). Two likely contributors are the relatively small timeout and the similarity induced by stratified sampling. Longer time limits and multiple independently stratified samples can introduce more variance and help reduce bias in competitive settings.

3.2. Showcase: preprocessing

This track examined how many instances can be decided by preprocessing alone and how preprocessing affects subsequent solving. Experiments here ran on 64-bit Ubuntu 12.04 with four 2.6 GHz 12-core AMD Opteron 6238 CPUs and 512 GB RAM in total. Time limits for preprocessing were intentionally smaller than in other tracks, based on the expectation that "pure-preprocessing" decisions, if possible, tend to occur quickly.

We studied two regimes: (i) each preprocessor applied *individually* to original instances; and (ii) *incremental* pipelines chaining multiple preprocessors over several rounds. Memory caps varied by experiment; when specified, they are noted below.

Table 8. Rankings (by solved count) across seven stratified benchmark samples (no preprocessing, 200 s cutoff). Two-letter labels are anonymized solvers

Rank	Sample 1	Sample 2	Sample 3	Sample 4	Sample 5	Sample 6	Sample 7
1	AX	AX	AX	BY	AX	AX	AX
2	BY	BY	CZ	AX	BY	BY	BY
3	CZ	DU	BY	CZ	CZ	CZ	CZ
4	DU	CZ	DU	DU	EV	DU	DU
5	EV	EV	EV	EV	DU	FW	EV
6	FW	FW	GT	FW	GT	EV	FW
7	GT	GT	FW	GT	FW	GT	HS
8	HS	HS	HS	HS	HS	HS	GT

Table 9. PAR10 rankings (200 s cutoff) across seven stratified benchmark samples (no preprocessing). Two-letter labels are anonymized solvers.

Rank	Sample 1	Sample 2	Sample 3	Sample 4	Sample 5	Sample 6	Sample 7
1	AX	AX	AX	BY	AX	AX	AX
2	BY	BY	BY	AX	BY	BY	CZ
3	CZ	CZ	CZ	CZ	EV	CZ	BY
4	DU	DU	DU	DU	CZ	FW	DU
5	EV	EV	GT	EV	DU	DU	EV
6	FW	GT	EV	FW	GT	EV	FW
7	GT	FW	FW	GT	FW	GT	HS
8	HS	HS	HS	HS	HS	HS	GT

3.2.1. Individual preprocessing. We ran the four submitted preprocessors on the benchmark sets with a 300 s wall-clock limit and 7 GB memory. Table 10 summarizes the solved counts. Performance depends strongly on the benchmark family. For example, on `conformant-planning`, *Hiqqer3e* solves the most instances, whereas on `reduction-finding`, *Hiqqer3p* leads. By design, *Hiqqer3e* cannot perform variable elimination and thus can only decide unsatisfiable instances. Aggregating over preprocessors (Table 11) does not always exceed the best single tool, indicating partial subsumption on some families (e.g., on `planning-CTE`, every instance solved by *Bloqqer* is also solved by *Hiqqer3p*).

3.2.2. Incremental preprocessing. To combine strengths, we evaluated *incremental* pipelines: given a sequence (e.g., A→B→C→D), the output of one preprocessor feeds the next, and this round-based process may repeat up to six rounds or stop early if the instance is decided. Labels are: A=*Hiqqer3e*, B=*Bloqqer*, C=*Hiqqer3p*, D=*sQueezeBF*. Each call had a 120 s limit; thus, with four tools and six rounds, a single instance could spend up to 2880 s in preprocessing, substantially more than in solving-focused tracks to decouple preprocessing power from solving.

On `set eval2012r2` we tested all $4! = 24$ orderings. Table 12 shows that every pipeline solves more instances than any single preprocessor (cf. Tables 10–11). In total, 119 instances (34%) are solved by some execution sequence, versus 87 instances (25%) by individual tools. Order matters: prefixes AB (e.g., ABCD, ABDC) perform best (107 and 106 solves), while prefixes DC (e.g., DCAB,

Table 10. Instances solved by individual preprocessors (t = total, s = SAT, u = UNSAT) with 300 s and 7 GB

	Hiqqer3e			Bloqqer			Hiqqer3p			sQueezeBF		
Set	t	s	u	t	s	u	t	s	u	t	s	u
eval2012r2 (345)	20	0	20	70	34	36	75	34	41	14	3	11
qbf-hardness (198)	0	0	0	50	12	38	52	12	40	12	0	12
sauer-reimer (924)	82	0	82	139	25	114	152	30	122	77	8	69
planning-CTE (150)	0	0	0	4	3	1	6	5	1	0	0	0
conformant-planning (1750)	642	0	642	492	11	481	483	12	471	50	0	50
reduction-finding (4608)	180	0	180	1502	841	661	1644	920	724	670	325	345

Table 11. Solved by at least one individual preprocessor (300 s, 7 GB)

Set	Total	SAT	UNSAT
eval2012r2 (345)	88	36	52
qbf-hardness (198)	52	12	40
sauer-reimer (924)	160	30	130
planning-CTE (150)	9	7	2
conformant-planning (1750)	760	13	747
reduction-finding (4608)	1688	944	744

DCBA) perform worst (96). Certain steps can destroy structure that later steps would have exploited.

Using the strongest ordering (ABCD), we then ran up to six rounds on the other benchmark sets (Table 13). Except for `qbf-hardness`, incremental preprocessing substantially increases solved counts compared to individual tools. We also tested selected doubled sequences (e.g., $(A^2B^2C^2D^2)_6$ and $(A^2D^2B^2C^2)_6$) on `set eval2012r2`; Table 14 shows modest additional gains overall, with a few sequences offering little to no benefit, indicating diminishing returns after some rounds.

3.2.3. Fixpoint detection. Incremental preprocessing can plateau: after some rounds, successive tools stop changing the instance. To detect this efficiently, we normalized the clause set between rounds (drop tautologies; sort literals within each clause; sort the clause list) and computed an MD5 hash before and after each round. If the hashes matched, we declared a fixpoint and terminated. Crucially, normalization served only for hashing; the original, unnormalized instance was forwarded to the next tool to avoid altering solver behavior.

Empirically, most decisions were made in the very first round, with a sharp decline thereafter; rounds five and six produced no new decisions in our runs. This pattern supports imposing a cap on the number of rounds. Extending the pipeline (e.g., ABCD) to 12 or even 24 rounds yielded no additional decisions beyond round 6 in our tests.

3.2.4. Downstream solving after preprocessing. To study how preprocessing sequences shape the residual problem, we produced two derived benchmark sets from `set eval2012r2` by running $(A^2B^2C^2D^2)_6$ and $(A^2D^2B^2C^2)_6$, yielding `AABBCCDD` (234 instances unsolved by preprocessing) and `AADDBBCC` (241 instances), respectively. The first sequence was chosen for strong standalone preprocessing, while the

Table 12. Instances solved by incremental preprocessing on `set eval2012r2` across 4! = 24 orders (A=Hiqqer3e, B=Bloqqer, C=Hiqqer3p, D=sQueezeBF). Each call: 120 s; up to 6 rounds

Order	Solved	SAT	UNSAT
ABCD	108	45	63
ABDC	107	43	64
ACBD	104	44	60
ACDB	104	44	60
ADBC	104	42	62
ADCB	103	42	61
BACD	103	42	61
BADC	103	42	61
BCAD	102	41	61
BCDA	104	43	61
BDAC	102	41	61
BDCA	100	39	61
CABD	100	41	59
CADB	100	40	60
CBAD	99	40	59
CBDA	99	40	59
CDAB	101	41	60
CDBA	101	40	61
DABC	103	39	64
DACB	101	39	62
DBAC	102	39	63
DBCA	101	39	62
DCAB	97	38	59
DCBA	97	38	59
VBS	120	49	71

Table 13. Incremental preprocessing (order ABCD) on other benchmark sets (up to 6 rounds; each call 120 s)

Set	Solved	SAT	UNSAT
`set eval2012r2`	108	45	63
`qbf-hardness`	52	12	40
`sauer-reimer`	182	32	150
`planning-CTE`	12	9	3
`conformant-planning`	940	14	926
`reduction-finding`	1862	941	921

second swaps B and D to probe the impact of their contrasting heuristics.

The resulting solver rankings differed noticeably across these two residual sets, showing that preprocessor order materially affects downstream solving difficulty. Totals also diverged: for `AABBCCDD`,

Table 14. Incremental preprocessing with doubled calls on `set eval2012r2`. Each preprocessor is invoked twice per round (up to 6 rounds, 120 s per call)

Sequence	Solved	SAT	UNSAT
$(A^2B^2C^2D^2)_6$	112	46	66
$(A^2B^2D^2C^2)_6$	112	45	67
$(A^2D^2B^2C^2)_6$	105	42	63
$(B^2C^2D^2A^2)_6$	104	42	62
$(D^2A^2B^2C^2)_6$	103	39	64

preprocessing decided 111 instances and the best solver on the remainder added 92 (203 overall). For `AADDBBCC`, preprocessing decided 104, but the best solver added 104 (208 overall). Thus, a pipeline that solves *fewer* instances up front can still be preferable if it leaves a more solver-friendly residual core.

3.3. Showcase: applications

We evaluated solver behavior on six application-driven families: `reduction-finding`, `conformant-planning` (split into `bomb` and `dungeon`), `planning-CTE`, `sauer-reimer`, and `qbf-hardness`. Each family contributed 150 randomly sampled instances. No preprocessing was applied, and runs used a 900 s timeout with 7 GB memory.

Performance varied sharply by family. Expansion-oriented approaches (e.g., *Nenofex* and, in several cases, *RAReQS*) excelled on `conformant-planning-dungeon` and `planning-CTE`, where expansion tended to simplify the structure. Conversely, search-oriented solvers (e.g., *DepQBF*, *GhostQ-CEGAR*, *Hiqqer3*) were comparatively weaker on those planning sets but performed strongly on families such as `qbf-hardness`. These complementary strengths point toward hybrid methods that combine expansion with clause/cube learning, tuned to family-specific characteristics.

3.4. Showcase: certificates

We assessed proof production, certificate extraction, and (where available) strategies on `set eval2012r2` without preprocessing. The submitted toolset comprised one proof-producing solver (*DepQBF*) and two certificate extractors (*QBFcert*, *ResQu*); additional public tools were included for breadth. Solving and certification were budgeted separately at 600 s and 3 GB each.

DepQBF solved 91 instances and produced Q-resolution proofs. Roughly two thirds of these were successfully certified by *QBFcert* and *ResQu*, which follow similar extraction principles (both represent Skolem/Herbrand functions as AIGs). *ResQu* applies ABC-based AIG simplification, likely explaining minor discrepancies: it certified a few instances that *QBFcert* did not, while *QBFcert* certified others that *ResQu* missed. Additional workflows included *QuBE-cert* with proof checking, and direct certificate-producing solvers (*sKizzo*, *squolem*) whose outputs were checked by *ozziKs*, *qbv*, or *ResQu*; we observed format-conversion errors for a subset of *squolem* outputs.

A practical bottleneck is resource usage during certification. *DepQBF* logs every Q-resolution step to disk; the resulting traces can reach gigabytes, causing memory failures during extraction. Among the 91 solved instances, proofs were extracted for 82. The number of resolution steps spanned from single digits to several million, with proof files ranging from kilobytes to multiple gigabytes. Overall,

current certification pipelines still lag behind solver capability under tight time/memory budgets. Promising remedies include maintaining proofs in memory to avoid bloated traces and adopting more compact proof formats (e.g., QRAT-style) to curb both time and space demands.

4. Conclusion

This study provided a consolidated, tool- and benchmark–oriented picture of the QBF landscape as of the QBF Gallery 2018, with an emphasis on how preprocessing, solver design, and benchmark composition interact. Four takeaways stand out.

First, preprocessing is both powerful and delicate. Individually, modern preprocessors decide a substantial fraction of instances; when staged incrementally they solve markedly more, but their order matters. Our fixpoint analysis shows that most gains occur in the very first round, with diminishing returns thereafter; beyond six rounds we observed no additional decisions. This argues for capped, order-aware pipelines rather than unbounded iteration.

Second, solving performance is conditional on preprocessing and on instance families. Best-foot-forward experiments reveal two solver classes: those that prefer raw instances (structure-reconstruction or heavy in-solver preprocessing) and those that benefit from aggressive external preprocessing. Moreover, stratified sampling alleviates family bias, but rankings can still shift with cutoff choices, reinforcing that "winner" claims are context dependent.

Third, applications are heterogeneous and reward different paradigms. Expansion-based solvers excelled on planning-style families, whereas clause/cube learning approaches performed better on hardware-oriented or hardness-driven sets. This complementarity suggests hybrid or portfolio designs that adaptively select (or interleave) expansion and search based on lightweight features extracted early in the run or after a brief preprocessing probe.

Fourth, certification lags solving. While proof-producing solvers and extractors validated a significant portion of solved instances, resource usage (trace volume, memory) remains a practical bottleneck. More compact proof formats and in-memory proof maintenance appear necessary for certification to keep pace with solver strength.

Implications. The Gallery's non-competitive, community-driven setup proved effective for surfacing nuanced behavior that leaderboards alone obscure. For users, the results indicate that (i) pairing a solver with a short, well-ordered preprocessing cascade can yield large gains, and (ii) solver choice should be informed by application family. For developers, the data motivate (i) adaptive pipelines that stop at fixpoints, (ii) hybridization across expansion/search, (iii) proof infrastructure that minimizes I/O and redundancy, and (iv) benchmark curation that preserves structural diversity while limiting family dominance.

Limitations and outlook. Our experiments focused on CNF tracks and a specific time/memory budget; non-CNF tracks and broader resource envelopes merit renewed attention. Future Gallery iterations should (a) systematize order-selection for preprocessing (e.g., learned schedulers), (b) standardize interfaces for exchanging structure between preprocessors and solvers, (c) promote compact, checkable proof ecosystems, and (d) continue growing application-driven, stratified benchmark suites. Taken together, these directions aim to turn the observed piecemeal advances into robust, reproducible progress across the full QBF toolchain.

References

[1] Blinkhorn, J. L. (2019). *Quantified Boolean Formulas: Proof Complexity and Models of Solving* (Doctoral dissertation, University of Leeds).

[2] Lonsing, F., Seidl, M., & Van Gelder, A. (2016). The QBF gallery: Behind the scenes. *Artificial Intelligence, 237*, 92-114.

[3] Kleine Büning, H., & Bubeck, U. (2021). *Theory of Quantified Boolean Formulas, In: Handbook of Satisfiability*, IOS Press, 2009, pp.735–760.

[4] Charwat, G., & Woltran, S. (2019). Expansion-based QBF solving on tree decompositions. *Fundamenta Informaticae, 167*(1-2), 59-92.

[5] Bloem, R., Braud-Santoni, N., Hadzic, V., Egly, U., Lonsing, F., & Seidl, M. (2018, October). Expansion-based QBF solving without recursion. In 2018 *Formal Methods in Computer Aided Design* (FMCAD) (pp. 1-10). IEEE.

[6] Bubeck, U., & Kleine Büning, H. (2007, May). Bounded universal expansion for preprocessing QBF. In *International Conference on Theory and Applications of Satisfiability Testing* (pp. 244-257). Berlin, Heidelberg: Springer Berlin Heidelberg.

[7] Benedetti, M. (2005, July). Experimenting with QBF-based formal verification. In *Proc. of the 3rd International Workshop on Constraints in Formal Verification* (CFV). To be published in ENTCS, Elsevier.

[8] Rykov, A. I., Rykov, I. A., Nomura, K., & Zhang, X. (2005). Frequency spectra of quantum beats in nuclear forward scattering of 57Fe: The Mössbauer spectroscopy with superior energy resolution. *Hyperfine Interactions, 163*(1), 29-56.

[9] Loth, M., Sebag, M., Hamadi, Y., & Schoenauer, M. (2013, September). Bandit-based search for constraint programming. In *International Conference on Principles and Practice of Constraint Programming* (pp. 464-480). Berlin, Heidelberg: Springer Berlin Heidelberg.

[10] Benedetti, M. (2005). *Istituto per La Ricerca Scientifica E Tecnologica* (IRST) Via Sommarive 18, 38055 Povo, Trento, Italy.

[11] Peschiera, C., Pulina, L., Tacchella, A., Bubeck, U., Kullmann, O., & Lynce, I. (2010, July). The seventh QBF solvers evaluation (QBFEVAL'10). In *International Conference on Theory and Applications of Satisfiability Testing* (pp. 237-250). Berlin, Heidelberg: Springer Berlin Heidelberg.

[12] Narizzano, M., Pulina, L., & Tacchella, A. (2007, September). Ranking and reputation systems in the qbf competition. In *Congress of the Italian Association for Artificial Intelligence* (pp. 97-108). Berlin, Heidelberg: Springer Berlin Heidelberg.

[13] Narizzano, M., Pulina, L., & Tacchella, A. (2006). Voting Systems and Automated Reasoning: the QBFEVAL Case Study. In *1st Workshop on Computational Social Choice* (COMSOC 2006).

[14] Han, V. K. M., d'Ercole, A. J., & Lee, D. C. (1988). Expression of transforming growth factor alpha during development. *Canadian Journal of Physiology and Pharmacology, 66*(8), 1113-1121.

[15] Li YuBin, L. Y., Liu WanShan, L. W., Zhu YinLing, Z. Y., & Diao LiPing, D. L. (2019). A fluorescence method for homogeneous detection of influenza A DNA sequence based on guanine-quadruplex-N-methylmesoporphyrin IX complex and assistance-DNA inhibition.

[16] Martins, R., Manquinho, V., & Lynce, I. (2012). An overview of parallel SAT solving. *Constraints, 17*(3), 304-347.

[17] Karafiloglou, P., & Akrivos, P. (1988). How to obtain a mixed local-non-local molecular wavefunction. *Chemical Physics, 127*(1-3), 41-51.

[18] Samulowitz, H., & Memisevic, R. (2007, July). Learning to solve QBF. In *AAAI* (Vol. 7, pp. 255-260).

[19] Balabanov, V., & Jiang, J. H. R. (2012). Unified QBF certification and its applications. *Formal Methods in System Design, 41*(1), 45-65.

[20] Janota, M., Klieber, W., Marques-Silva, J., & Clarke, E. (2016). Solving QBF with counterexample guided refinement. *Artificial Intelligence, 234*, 1-25.

[21] Balabanov, V., Widl, M., & Jiang, J. H. R. (2014, July). QBF resolution systems and their proof complexities. In *International Conference on Theory and Applications of Satisfiability Testing* (pp. 154-169). Cham: Springer International Publishing.

[22] Van Gelder, A. (2012, October). Contributions to the theory of practical quantified Boolean formula solving. In *International Conference on Principles and Practice of Constraint Programming* (pp. 647-663). Berlin, Heidelberg: Springer Berlin Heidelberg.

[23] Franco, J., & Van Gelder, A. (2003). A perspective on certain polynomial-time solvable classes of satisfiability. *Discrete Applied Mathematics, 125*(2-3), 177-214.

How to cite this article: Allen Van Gelder and Alex John (2020). Hybridizing Expansion and Search for QBF. *Bulletin of Computer and Data Sciences*, 1(1), 27-41. DOI: 10.71448/bcds2011-4

Received: 30/8/2020 **Revised:** 11/10/2020 **Accepted:** 29/11/2020 **Publish:** 30/12/2020