Scaling and Validating Set-Based Retrograde Analysis

Isaac Stone[1], Nathan R. Sturtevant[1,2], and Jonathan Schaeffer[1]

[1] University of Alberta
{istone,nathanst,jonathan}@ualberta.ca
[2] Alberta Machine Intelligence Institute (Amii)

Abstract. Retrograde Analysis has been an important algorithm used for solving games by building a database of state-value pairs. The recent Setrograde Analysis algorithm builds a database of set-value pairs. For the game of Bridge, the results for computing the 32-card database using Setrograde Analysis are reported, showing a reduction of six orders of magnitude over Retrograde Analysis. The computation was performed using as many as 20,000 cores over several weeks. This paper discusses the issues of maximizing the parallelism, ensuring the accuracy of the results, and minimizing the human effort required to complete the computation. The results provide new insights into the requirements to complete the 52-card database.

Keywords: Computer Bridge; Endgame databases; Tablebases; Retrograde Analysis; Setrograde Analysis; Partition Search; Parallel and distributed systems

1 Introduction

For many interesting games, it is possible to solve the game using a top-down compute-only approach; it just takes too long to complete. An alternative is to perform a bottom-up compute-and-storage solution. This can also take too long to complete and will quickly run out of storage. A hybrid approach can work best as demonstrated in 8×8 Checkers. A bottom-up computation was performed to build a database of all positions with 10 or fewer pieces on the board (10^{13}). Now, the top-down computation starting with 24 pieces could treat a 10-piece position as a terminal state. Checkers was solved [5].

Given a deal of 52 cards for the game of Bridge, the object of cardplay is to win as many tricks as possible. Calculating the best result for a single deal is easy for modern Bridge programs. Unfortunately, there are 10^{29} possible 52-card (13-trick) deals (10^{28} once symmetries are removed). To construct Bridge strategies, it is beneficial to consider a large portion of the state space, far more than can be computed in an online fashion. Compact databases containing the solutions to all positions would be a wonderful resource. They could be used, for example, to build better Bridge bidding systems. The resources required to build the complete databases are unlikely to be available to us for many years to

come. This paper discusses the steps we have taken toward this eventuality, along with the current state of the databases. In the short term, hybrid approaches are possible that will build on the partial databases described in this paper.

Most game endgame databases (or tablebases) have been constructed using Retrograde Analysis. The algorithm reasons about individual states in the search space. In previous work, we introduced the Setrograde Analysis algorithm, which combines states into sets of states, and reasons about the sets [8]. Setrograde represents a breakthrough for games like Bridge, allowing a very large state space to be solved in a compressed manner by proving results for sets rather than states.

In this paper, we have three contributions to our understanding of retrograde computations:

1. Retrograde Analysis tends to be embarrassingly parallelizable, and therefore is well suited to running on large numbers of cores up to a certain point. Traditional Retrograde computations typically need to read and write from vast regions of memory that cannot fit in RAM at larger scales, resulting in the process being I/O limited. Setrograde Analysis reduces the memory and disk footprint of the calculations, allowing us to create the largest and most compact endgame databases for any game to date [8]. Our previous publications report on the 28-card databases, consisting of 2×10^{17} symmetries, for perfect-information Bridge cardplay. This paper reports on the completion of the 32-card databases that consist of 3×10^{19} symmetries. Setrograde Analysis provided a 5 orders of magnitude reduction in the compute and data needs for this construction.
2. This paper discusses the engineering required to leverage modern computing resources. We cover the challenges of consistency checking and validating this computation. With more than 20,000 processors running in parallel over weeks or months, it is inevitable that some will fail. The monitoring and management of the computations have been automated to minimize the faults that require human intervention.
3. We present new estimations for the size of the 52-card databases, using subsets of the 8- to 32-card databases to improve our understanding of how the problem scales. The results contradict earlier projections for the size of the complete 52-card databases.

The 36-card databases are currently being constructed (roughly 10% complete at the time of paper submission). The 36-card databases may be our limit given our current storage constraints.

2 Background and Related Work

Setrograde Analysis [7, 8] is a Set-Based approach to Retrograde Analysis, using a set representation inspired by the backup-heuristic employed in Matt Ginsberg's Partition Search [3] algorithm. Large-scale computations have been performed to generate endgame databases for other games, notable examples include

Checkers [4–6] and Chess [1]. Both of these efforts used traditional Retrograde Analysis, which at scale tends to be IO-bound. Setrograde analysis is compute-bound, which changes the nature of some scaling challenges.

This section is broken up into three parts. Section 2.1 explains (in minimal detail) the rules of Bridge. Section 2.2 provides some of the motivation and intuition behind Setrograde Analysis. Section 2.3 is provided to help the reader build a high-level understanding of Setrograde Analysis, which motivates the challenges of scaling and validating the algorithm that are discussed in Section 3.

2.1 Rules of Bridge

Bridge is a trick-taking card game played with a 52-card deck in partnerships. Bridge has an initial bidding phase, in which partners communicate by placing bids, and the final bid determines both the number of tricks each partnership needs to win and the *strain* for the cardplay. The strain can be any of the four suits ("trump"), or "no trump".

The cardplay phase consists of 13 "tricks" in which each player plays exactly one card. The first card in any trick is unrestricted, whereas each subsequent card must match the suit of the first card if possible. If not possible, then the play is unrestricted. If there is a trump suit, then the highest trump played in any trick wins the trick. Otherwise, the highest card of the first suit played wins. Whichever player wins the trick plays first in the next trick. Tricks are counted by partnership, not by player, and it is strictly better to win more tricks.

2.2 Partition Search and Retrograde Analysis

Double-dummy (DD) problems are the perfect-information variant of Bridge cardplay. A deal of n cards (with n as a multiple of 4) has an exact integer evaluation under perfect play, ranging from 0 to $\frac{n}{4}$ – the number of tricks won. See Fig. 1 (left) for a one-suit, two-card deal; the four players around the table are referred to as North, South, East, and West.

The idea behind Retrograde Analysis is to solve a game from the end towards the start. In the example of Bridge, this works by enumerating all deals where each player has one card (1 trick), solving them, and then storing the results. Then one can move backwards in the game to consider all deals where each player has two cards (2 tricks). For a given 2-trick deal, taking the maximum result of all successor 1-trick deals (already computed) produces the correct result. Given sufficient computational and storage resources, one could continue to obtain the 3-trick results, and so on. Most often, the algorithm is expressed as solving depth d (where d is the number of tricks in Bridge or the number of pieces on the board as in Chess) given the precomputed results of depth $d-1$. The computational cost and storage requirements typically grow exponentially in d. There are numerous enhancements to the basic algorithm that can improve its performance [9]. Retrograde Analysis is usually used to compute and store the value for all states up to a given depth d, producing a comprehensive database.

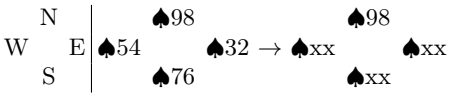

Fig. 1. ♠98 wins two tricks regardless of the locations of lower cards in the perfect-information deal (left) and set of deals (right); x refers to any low card. Play proceeds clockwise starting with North (N) until each player has played one card (East, South, West). The highest card wins. The winning player is next-to-play.

Since evaluations are discrete, n-card deals may be grouped into *consistent* sets, meaning that all states in the set share the same evaluation. Partition Search [3] is a forward minimax solver with a set-based backup heuristic that generates a consistent set for each state in the search. Ginsberg showed a branching factor reduction by using these sets as transposition table entries. DD solvers based on Partition Search remain a state-of-the-art tool for Bridge programs.

Bridge deals are colloquially described by the number of cards in each suit and the ranks of relevant high cards. A low-valued card with a rank that does not affect the result can be referred to as x. This notion is formalized in the set representation we use. Any card denoted x is interchangeable with any other x and is strictly lower than a ranked card. For example, Fig. 1 (right) shows a set containing $6!/(2!)^3 = 90$ states that Partition Search might represent by generalizing Fig. 1 (left).

Partition Search maintains consistent sets by heuristically backing up cards that can be proven to never win by rank. Available DD solvers rely on expert heuristics for move ordering, set generation, and early search cutoffs. For retrograde searching, Set-Based Retrograde Analysis [8] presents a more general algorithm for creating large consistent sets and a framework in which the need for expert knowledge is eliminated.

2.3 Set-Based Retrograde Analysis

Set-based Retrograde Analysis (or "Setrograde") builds on the ideas of Retrograde Analysis and Partition Search. Setrograde repeatedly performs an abstract 1-trick lookahead that operates directly on a set of states to prove the game values for all n-card Double Dummy problems. The 1-trick lookahead currently relies on the availability of an $(n-4)$-card database.

The set of $(n-4)$-card states can be partitioned into disjoint subsets that can be independently calculated. First, partitioning can be done using the strain (clubs, diamonds, hearts, spades, and no-trump). For example, a card deal with spades as trump differs from the same deal with no trump. Second, two states that differ in the distributions of the four suits among the four players (the *shape*) cannot be in the same set. For instance the sets below represent two unique 8-card shapes (they differ in the distribution of hearts and spades).

For the purpose of this paper, a shape is considered an independent and indivisible unit of work.[3] The total number of shapes given n cards are provided in Table 1. The number reported here considers a shape to *include* strain specification.

3 Scaling Setrograde to 32-card Bridge

Section 3.1 presents the first results of the 32-card (8-trick) database computations. Section 3.2 details sources of human and hardware error, and the methods used to ensure the integrity of the databases as they scale in size. Section 3.3 covers our approaches to parallelism within nodes and coordination between nodes to scale the generation process to tens of thousands of CPU cores with minimal wastage. Finally, Section 3.4 details the systems used to automate the process so that the deployment can be managed at scale with minimal human intervention.

3.1 32-card Database Results

Table 1 shows that the 32-card computation has 6×10^{31} states. Symmetry reduction reduces the search space by 12 orders of magnitude, to 3×10^{19}. This is the number of states that Retrograde Analysis would have to compute. In contrast, Setrograde Analysis creates 2×10^{13} sets, a reduction of 6 orders of magnitude. This is still a large computation, but it can be broken down into 100 million shapes that can be computed in parallel.

The computation was distributed across thousands of processors, requiring over 200 CPU years. The number of cores used varied depending on their availability and the amount of work available. Table 2 provides execution results and bounds for the maximum parallelism seen during database generation under the label "cores". For the computations up to and including 16 cards, parallelism was used for testing only, as the single-threaded wall-clock time was less than 10 minutes.

As seen in Table 2, a set-based approach to Retrograde Analysis can scale better than a state-based approach. For 24-card Bridge, it provided between 3 and 4 orders of magnitude speedup and 4 orders of compression. However, for the much larger 28- and 32-card Bridge computations, Setrograde provided an additional 10× speedup, along with 5 and 6 orders of magnitude of compression, respectively. Each byte in the 32-card database represents approximately one million symmetries.

For the 32-card databases, the runtime of a shape can vary from less than 1 millisecond to more than 6 days. The disk footprint can range from as few

[3] In some circumstances, shapes can be further subdivided but in general the benefits are not significant.

Table 1. State-space size and reductions for Bridge.

Cards	States	Symmetries	Sets	Shapes
4	3×10^{7}	2×10^{2}	7×10^{1}	5×10^{1}
8	9×10^{12}	8×10^{4}	8×10^{3}	2×10^{3}
12	4×10^{17}	3×10^{7}	5×10^{5}	3×10^{4}
16	3×10^{21}	1×10^{10}	3×10^{7}	3×10^{5}
20	7×10^{24}	3×10^{12}	1×10^{9}	2×10^{6}
24	5×10^{27}	8×10^{14}	2×10^{10}	1×10^{7}
28	1×10^{30}	2×10^{17}	7×10^{11}	3×10^{7}
32	6×10^{31}	3×10^{19}	2×10^{13}	1×10^{8}
36	1×10^{33}	4×10^{21}	-	2×10^{8}
40	5×10^{33}	4×10^{23}	-	3×10^{8}
44	4×10^{33}	4×10^{25}	-	3×10^{8}
48	3×10^{32}	2×10^{27}	-	1×10^{8}
52	3×10^{29}	1×10^{28}	-	8×10^{6}

as 8 bytes to more than 10 gigabytes before compression. This non-uniformity presents a challenge in distributing the workload efficiently at scale. Early approaches grouped several hundred shapes into task files, each of which would be solved sequentially. This resulted in a large number of task files being quickly depleted, while others were left to run for months. Section 3.3 discussed a better approach, which we used to finish the 32-card databases and to start the 36-card databases. The updated system should allow us to replicate the 32-card databases in one or two weeks of wall-clock time. We felt it would have been unnecessarily wasteful to re-start such a large computation purely to measure wall-clock runtime. No changes were made to the core solver.

When running on a shared cluster with thousands of cores, the mean time to failure is small (e.g., hardware problems, scheduler constraints). There is a need for a robust way of monitoring the computations and recovering from a failure. For example, if a node goes offline while writing to a file, that file can be corrupted, or worse, left silently in an incomplete or incorrect state.

3.2 Consistency Checks and Validation

As the databases scaled, the data structures used began to cause pointer overflows: 64-bit data types were not big enough. Certain forms of overflow cause immediate errors. Others can occur silently and have to be caught by consistency checks. None have yet made it through the consistency checks only to reveal themselves during validation — the last line of defense. At any point in the code where it is inexpensive to verify an assumption, a consistency check is included to verify its correctness.

There are two main forms of consistency checks in the Setrograde code base. Most consistency checks throw errors in cases that should never occur. For instance, any time a set is added to the "solved" list, all states in that set are

Table 2. Database generation time and storage for Bridge. This table reflects the runtimes and database sizes for the latest version of the program. Runtimes have decreased between 10- and 100-fold from those reported in our previous publication [8].

	DB Size (GiB)		Gen Time (CPU days)		Sym/Byte	Max Cores
Cards	**Retrograde**	**Setrograde**	**Retrograde**	**Setrograde**	**Setrograde**	**Setrograde**
4	1×10^{-7}	7×10^{-6}	—	—	2.6×10^{-2}	1
8	4×10^{-5}	3×10^{-5}	—	—	3.2×10^{0}	1
12	2×10^{-2}	1×10^{-3}	3×10^{-3}	5×10^{-5}	3.1×10^{1}	1
16	$\dagger 5 \times 10^{0}$	5×10^{-2}	1×10^{0}	2×10^{-3}	1.9×10^{2}	1
20	$\dagger 2 \times 10^{3}$	2×10^{0}	$\dagger 4 \times 10^{2}$	4×10^{-1}	1.6×10^{3}	48
24	$\dagger 4 \times 10^{5}$	5×10^{1}	$\dagger 1 \times 10^{5}$	2×10^{1}	1.5×10^{4}	192
28	$\dagger 9 \times 10^{7}$	1×10^{3}	$\dagger 2 \times 10^{7}$	7×10^{2}	1.4×10^{5}	$\approx 10,000$
32	$\dagger 1 \times 10^{10}$	3×10^{4}	$\dagger 2 \times 10^{9}$	7×10^{4}	1.0×10^{6}	$\approx 20,000$

†Retrograde values have been extrapolated for 16 cards and beyond.

wiped from the (set-based) open list. Every set that is added to the solved list is generated by selecting a *singleton* state that exists on the open list and solving for a set that includes the singleton state. Therefore, it should be impossible to add a set to the solved list that is exactly identical to a pre-existing set. A consistency check was included to throw an error if it detected that an existing set was being inserted into the data structure. This was critical, as it caught an otherwise silent error during the first attempt at generating the 32-card databases, prompting a long debugging process, heavy validation, and a restart.

A second form of consistency check can be performed after-the-fact. Since the databases contain tight upper and lower bounds on the value of each state in a shape, a scan is performed for each shape to check for conflicts between the bounds of any set. During this scan, the completeness of the database is verified.

Finally, while the hardware we are using employs 256-bit checksums and redundancy to ensure data integrity, we add a 64-bit cyclic-redundancy-check checksum (CRC) to the end of each database segment. This serves two purposes. First, when the databases are transferred between machines, data integrity can be verified against the internal checksum. Second, checksums are verified on first-use during the generation process. These checksums are the last value written in at the end of each database segment, so besides checking for structural integrity (i.e., no pointers point past the end of the range), a missing or inaccurate checksum would allow us to catch silent write failures. In practice, write failures are rarely silent. Most result in corrupted files because the files are written in a compressed format. This is easily detected and common on node failure. We maintain backups of active files to avoid losing work when this happens.

Validation is the last line of defense. The fastest form we use involves iterating over shapes, and for each shape selecting a single deal. This deal is solved using a traditional alpha-beta search, using the set-based databases to truncate to a 1-trick lookahead. A full-depth search would be prohibitively expensive. The value returned by the search is compared to the value returned by a direct

lookup in the Setrograde databases. This process can be used to validate billions of positions. Even that is only a small fraction of the $\approx 3 \times 10^{19}$ positions in the 32-card database. Validation is expensive because it scales similarly to a traditional retrograde approach. Even validating a single position per set can be nearly as expensive as the complete generation process.

3.3 Coordinating Generation at Scale

With hundreds of millions of shapes, it is impractical to queue each shape independently. Currently, a task file consists of approximately 10,000 randomly selected shapes. Tasks files are generated and saved in a "queue" directory. Each independent process will select an item from the queue and move it to the "active" folder. This is an atomic file-system operation and will only succeed for one process. The solutions to the shapes in a given task file are stored together in a single database file. The complete databases consist of tens of thousands of such files.

Through 28 cards this distributed approach was sufficient. For 32-cards, even with as few as 500 shapes per file, the sequential solve time for the worst case task files was a matter of months. Grouping 500 shapes into a task results in more than 200,000 task files. The sheer number of small files in the directories overwhelms the file system. To reduce the wall-clock time for large task files, jobs are started with a main thread and between 48 and 192 worker threads. For larger databases, we limit each node to 48 workers, since this allows us to request more RAM per core. The main thread distributes shapes between the workers allowing parallelism within task files. Even with a maximum parallelism of 48 workers per file, we were able to increase the task file size from 500 to 10,000 shapes, while still achieving dramatically better wall-clock performance. Each job is confined to a single node, and all coordination is done using shared memory regions. All writes to the output file are performed by the main thread. One thread is responsible for monitoring and managing the worker pool; another thread is responsible for picking up task files, assigning shapes to workers, and saving the results of the workers' computations. The main thread handles signals and shutdown sequences.

3.4 Automating Generation at Scale

Several machines have been used for database generation; however, it is impractical to migrate 26+ TiB of data between machines. With the exception of profiling tasks and validation, all computations involved in building 32-card and larger databases are now performed on a single 134,400 CPU core system. This cluster includes 700 standard compute nodes, each having 192 cores and 748 GiB of RAM.

The cluster used for database generation is shared and managed by a batch scheduler. Jobs may not have a wall-clock runtime exceeding 7 days. As database generation is often running on a large number of compute nodes, it is not uncommon for a node to fail without warning. When a job times out or is manually

canceled, the main thread traps a signal. The main thread then asks workers to shut down at the earliest safe opportunity. Once all workers have finished, the main thread re-queues incomplete task files before exiting.

When a node fails, some work is lost. Active database files are backed up roughly every hour during the generation process, limiting the impact. On the other hand, worker computations that have been running for at least a day are checkpointed during a graceful shutdown. When a node failure occurs, checkpoint files cannot be created. The checkpoint files are too large to maintain preemptively. Node failures also leave task files 'locked'. Those task files must be manually re-queued. As this can be performed on a weekly or even monthly basis, it is only a mild inconvenience.

Throughout the generation process, a single job is kept in the scheduler queue at any given time. When a job gets scheduled, it immediately submits a new job to the scheduler, unless the work queue is depleted or a shutdown is set. This lets us schedule a large number of jobs without overwhelming the scheduler, and allows us to scale up to a maximum of 700 jobs (one per node) when the resources are available. To avoid wastage, jobs are scheduled with a timeout of exactly 7 days, using 48 cores (25% of a node's available compute) and 480 GB RAM ($\approx 65\%$ of a node's memory). This allows other users to run compute-intensive workloads using the other 144 cores on the same nodes as our relatively memory-intensive database generation process.

Human intervention is needed when there are insufficient disk resources. This is an increasing problem for us, even though we have more than 100 terabytes of disk. Intervention is also needed when job resubmission fails for any reason, as well as every few weeks to re-queue workloads left behind by node failures. More work needs to be done to improve the robustness of the shutdown sequence, especially for long-running computations. The checkpointing process can be slow. When it takes too long, workers are cut off by a hard shutdown, corrupting the checkpoint. Ideally, workers would checkpoint at regular intervals, rather than at the start of a shutdown sequence; however, this requires more disk than storing the complete databases. Currently, a shutdown signal is sent to the main thread 20 minutes before a job times out; extending the shutdown period increases idle time but reduces checkpoint corruption.

4 Projections for Complete 52-card Bridge

This section discusses recent analysis that contradicts an early assumption regarding the scale of the databases as we approach the complete 52-card case. At the time of this writing, the 36-card databases generation is in progress and is estimated to be between 5 and 10% complete.

In previous work, we predicted that the 40- and 44-card databases may be the hardest to compute, and have a larger disk footprint than the complete 52-card databases. While the average database segment for each shape grows with each additional card, the number of shapes starts to taper off beyond 40 cards. Additionally, the cases where there are exactly $\frac{n}{4}$ cards per suit have been

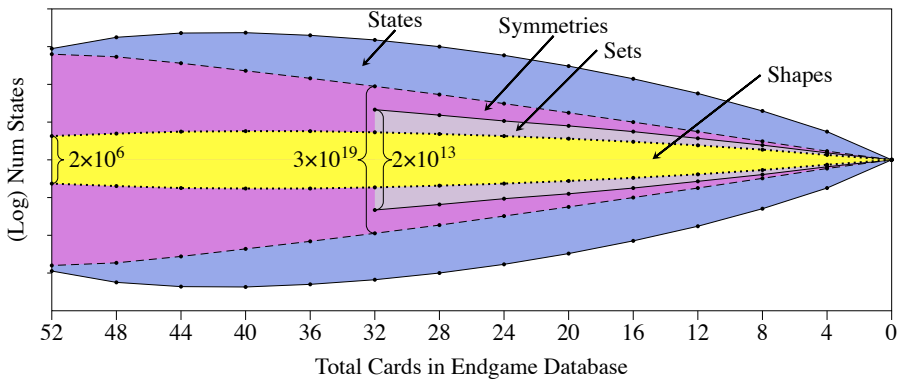

Fig. 2. Statespace, symmetries, sets, and shapes plotted on a log scale.

on average cheaper to compute and store than the cumulative average across all n-card database segments. If this continues to be the case, it would suggest that the 52-card cases (13 cards in each suit for each player) might not be too expensive. Finally, Figure 2 shows the number of sets roughly corresponding to the midpoint on a log plot between the number of shapes and the number of symmetries. The number of sets is nearly linearly correlated with the database disk footprint, so it seemed plausible that we would see an inward trend around 48 cards.

With the 32-card databases complete, we tested this prediction by isolating the database segments for shapes with 2 or fewer cards per suit (2c), then 3 or fewer, and so on, up to 8c. We then measured the size of each partial database with exactly t cards per player (i.e. 1t up to 8t for the 8c case), and graphed them on a log scale, as seen in Figure 3. If our initial prediction was correct, we would expect the lines to be concave downward, and for the last one or two datapoints to trend lower than the second-to-last. While this is technically the case, we also expected the trend to be *stronger* for larger values of c. This is not the case, and in fact, the trend seems to be less pronounced for larger sizes.

We find the recent evidence compelling, and now believe that the 48-card databases, or even the 52-card databases, will have the largest disk footprint. Despite this, as compute continues to scale year over year, without any algorithmic advances, it may be feasible to finish the databases within the next one or two decades. With better compression or algorithmic advances, that timeline may be significantly reduced. In fact, the largest machines today may already have enough storage and computing power to generate the databases. However, given the resources currently available to us, 36 cards is likely the largest case we can complete within a year.

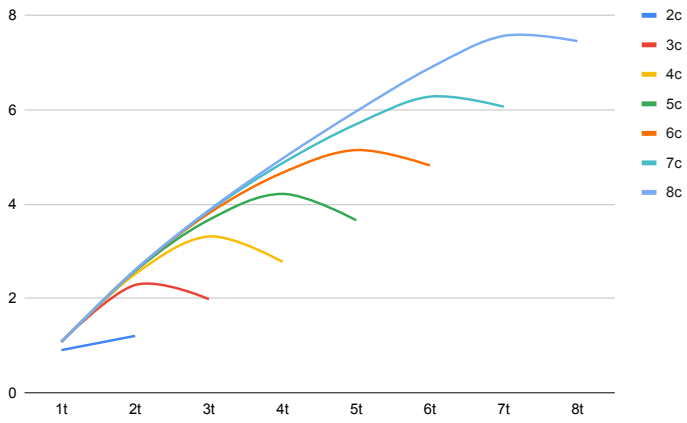

Fig. 3. Log scaling for 4 by t-card decks.

5 Summary and Conclusions

At present, the Bridge databases stand as the largest endgame databases computed to date, with 3×10^{19} symmetries stored in just 28 TiB. These databases have been built using a combination of novel techniques and by effectively leveraging the extreme parallelism of modern high performance computing. The 36-card databases are under construction. The 52-card databases may require significant algorithmic advances or a further scaling of compute resources to build. This long-running computation is set up to run with minimal human intervention, combining parallelism within compute nodes, and distributed coordination between nodes to scale on modern cluster architectures.

Until the 52-card databases can be completed, there is a lot that can be done to leverage the databases. They will be used as part of programs intended to outperform the strongest humans across the complete game. The databases can be used to help find solutions to imperfect-information cardplay [2]. They might also be used to generate puzzles and curricula for furthering human understanding of the game. And finally, the authors have significant interest in using the databases to solve for equilibrium partial-strategies for Bridge bidding under perfect-information cardplay.

Acknowledgments. This research is generously supported by the Digital Research Alliance of Canada and the Canadian Natural Sciences and Engineering Research Council (NSERC) and the Canada CIFAR AI Chairs Program.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Chess: Syzygy endgame tablebases. syzygy-tables.info (2024), Accessed: 2025-06-12
2. Frank, I., Basin, D.: Optimal play against best defence: Complexity and heuristics. In: van den Herik, H.J., Iida, H. (eds.) Computers and Games. pp. 50–73. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)
3. Ginsberg, M.L.: Partition search. In: AAAI Conference. pp. 228–233 (1996)
4. Schaeffer, J., Björnsson, Y., Burch, N., Lake, R., Lu, P., Sutphen, S.: Building the Checkers 10-Piece Endgame Databases, pp. 193–210. Springer US, Boston, MA (2004). https://doi.org/10.1007/978-0-387-35706-5_13, https://doi.org/10.1007/978-0-387-35706-5_13
5. Schaeffer, J., Burch, N., Björnsson, Y., Kishimoto, A., Müller, M., Lake, R., Lu, P., Sutphen, S.: Checkers is solved. Science **317**(5844), 1518–1522 (Sep 2007). https://doi.org/10.1126/science.1144079, https://doi.org/10.1126/science.1144079
6. Schaeffer, J., Culberson, J., Treloar, N., Knight, B., Lu, P., Szafron, D.: A World Championship Caliber Checkers Program. Artificial Intelligence **53**(2-3), 273–290 (1992)
7. Stone, I.: Setrograde Analysis: A Set-Based Approach to Fast Computation and Compact Data Representation. Master's thesis, University of Alberta (January 2025)
8. Stone, I., Sturtevant, N.R., Schaeffer, J.: Set-based retrograde analysis: Precomputing the solution to 28-card bridge double dummy deals. In: International Joint Conference on Artificial Intelligence (IJCAI) (2025), https://webdocs.cs.ualberta.ca/ nathanst/papers/stone2025bridge28.pdf
9. Sturtevant, N.R., Saffidine, A.: A study of forward versus backwards endgame solvers with results in chinese checkers. In: Computer Game Workshop at IJCAI. pp. 121–136 (2017), http://www.cs.ualberta.ca/ nathanst/papers/sturtevant2017ccsolve.pdf