Survival of the Fittest

Evaluating Fitness Functions for Concurrency Testing on the XRPL Consensus Protocol

A. Mousavi Gourabi[1]

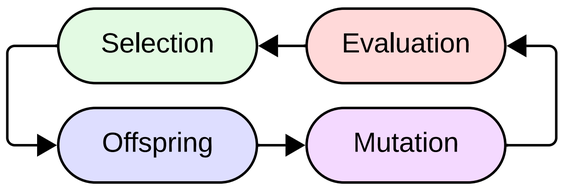
[1] EEMCS, Delft University of Technology

Bachelor's Research Project CSE, 2025

Why?

Goal	Problem	Challenges
Verifying correctness of the approach	Concurrency testing is incredibly difficult	How to effectively find edge cases

What are we doing? Evolutionary search and XRPL



The **evolutionary algorithm** has four steps: (1) **evaluation**, (2) **selection**, (3) reproduction, and (4) mutation.
In (1, 2), the samples in the population are evaluated based on a **fitness function**, after which a selection process decides on the best samples that progress to the next steps.
In (3, 4) a new population is created by first constructing it based on the old one, then creating offspring, and then mutating it.
In testing distributed systems our samples are **test cases**, where the algorithm tries to select the ones most likely to detect bugs.
The fitness function must be indicative of the tests' performance. The exact effects between them is examined.

XRP Ledger

The **XRPL consensus protocol** is a distributed algorithm that constructs and establishes a consensus state between the nodes in the network.
The aim of it is to safeguard the integrity of the **ledger**, which stores the transactions of the tokens in the network, and the network itself.
It is important that the network itself remains whole, it cannot split in two or fork. To avoid this from happening, all nodes must agree to one ledger, to one set of transactions.
The XRPL consensus protocol has **Byzantine fault tolerance**, which means that it can maintain itself even when there are malicious nodes in the network.
During our experiments we made use of a bug-seeded version of the XRPL consensus protocol. Here, we seeded a bug that would make it appear as though consensus would be reached when as few as 40% of the nodes in the network agree, down from the usual 80%.

Yes,	But!
The evolutionary algorithm is known to help find bugs faster than methods based on exhaustive search [2].	The impact of fitness functions and the selection procedure remains poorly understood.

Fitness functions and selection procedures

1. **Time fitness** selects based on the longest runs. The longer it took the algorithm, the more likely the run and associated test case were complex. Guiding the search towards these kinds of cases is known to help in finding certain kinds of bugs [2].
2. **Proposal fitness** selects based on the runs with the longest proposal sequences. The longer this sequence, the more likely the run and associated test case are a complex edge case. Guiding the search towards these kinds of cases is known to help in finding certain kinds of bugs [2].

1. **Elitism** always picks the test case with the best fitness.
2. **Tournament** selection picks the test case with the best fitness from a random subset of the population.
3. **Roulette** proportially picks the test cases based on their fitness values. A case with $f = 5$ will be selected 5× more often than a test case with $f = 1$.
4. **NSGA-II** is a more complex selection method that is used for selection using two or more fitness objectives. It ranks the test cases based on their dominance and crowding distance over the objectives [1].

Results

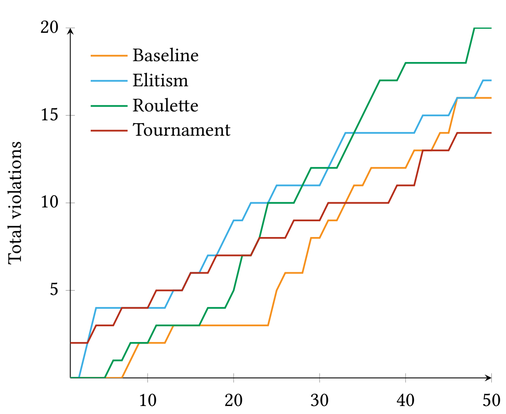


Figure: Cumulative violations in time fitness configurations

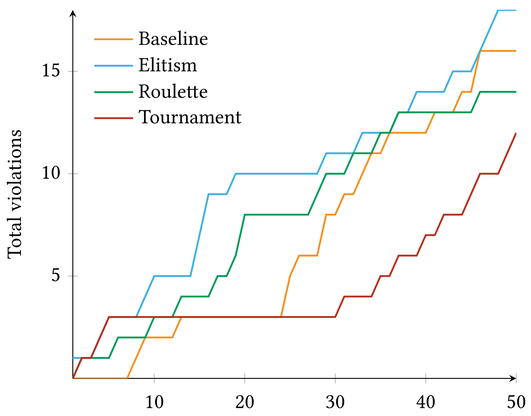


Figure: Cumulative violations in proposal fitness configurations

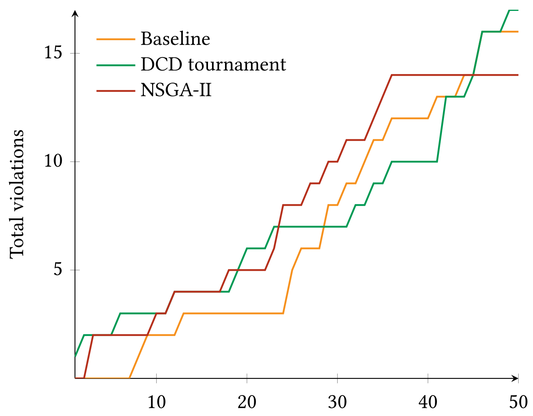


Figure: Cumulative violations in multi-objective configurations

We can see none of the algorithms are clearly beating the baseline on first glance. When we run hypothesis testing, we find that none of the approaches meet the p-value threshold of 0.05 for significance on this either. This means that we cannot prove that our setup with one iteration per test case, the Gaussian mutation operator at $\mu = 0$, $\sigma = 40$, and simulated binary crossover at $\eta = 3.0$ beats the random baseline.

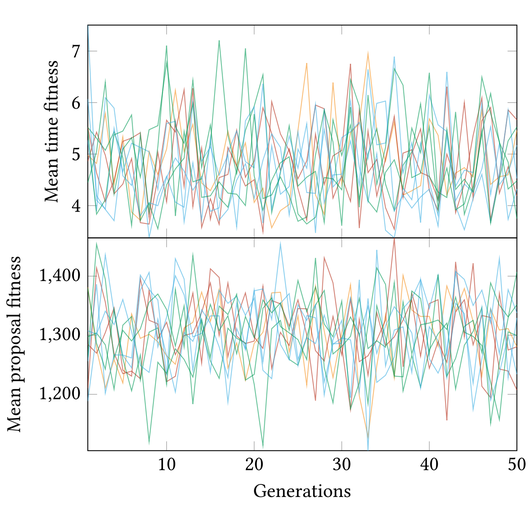


Figure: Failure to converge by the algorithm on either time and proposal fitness

In the preceding figure we can clearly see the failure of the algorithm to converge in any configuration. The proposal and time fitness both show noise patterns over their respective generations. Only one of the configurations does not meet the 0.05 threshold for the p-value under the augmented Dickey–Fuller test for stationarity. It is likely that this one exception is spurious itself as well.

Good fitness objectives

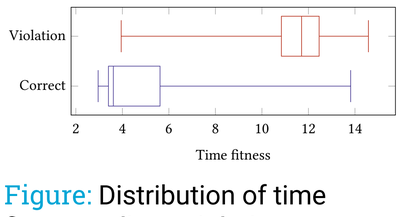


Figure: Distribution of time fitness split on violations

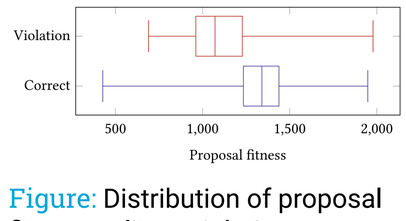


Figure: Distribution of proposal fitness split on violations

We can clearly see the stark difference in time and to some extent also proposal fitnesses between runs that detected violations and those that did not. This indicates that time and proposal fitness are indeed good objectives to optimize for, even with agreement failures.
When faced with the point biserial correlation test, these seeming correlations hold to be statistically significant.
Interestingly, the relation between the proposal fitness and our ability to detect a violation seems to be negative. As previous literature has advocated for proposal fitness to be maximized, this is remarkable.
Further analysis showed that the time and proposal fitness values of a test case are themselves strongly inversely correlated under the Pearson correlation test. When we control for this effect of on the proposal fitness, its correlation to violations becomes positive again.
This shows potential for the usage of multi-objective optimizers such as NSGA-II for finding these bugs. Though it is important to keep in mind that these results show that time and proposal fitness are to some extent conflicting objectives. Potentially this could also be a huge upside for the usage of these kinds of algorithms, NSGA-IIs ability to optimize over two conflicting objectives means that it will be able to find those cases with higher proposal fitness combined with a higher time fitness, while these would generally optimize in inverse directions when paired with a single objective optimizer, due to their correlation.

Conclusions

1. A setup with one iteration on the XRPL consensus algorithm will yield too unstable estimations for the time and proposal fitnesses to be able to optimize over.
2. Time and proposal fitness are sound objectives for finding potential agreement bugs in the XRPLCP.
3. There is great potential for the incorporation of multi objective fitness functions for finding good edge cases.

References

[1] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. "A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II". In: *Parallel Problem Solving from Nature PPSN VI*. Ed. by M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel. Berlin, Heidelberg: Springer, 2000, pp. 849–858. ISBN: 978-3-540-45356-7.

[2] M. van Meerten, B. Kulahcioglu Ozkan, and A. Panichella. "Evolutionary Approach for Concurrency Testing of Ripple Blockchain Consensus Algorithm". In: *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice*. ICSE-SEIP '23. Melbourne, Australia: IEEE Press, 2023, pp. 36–47. DOI: 10.1109/ICSE-SEIP58684.2023.00009.

Paper

A digital version of the paper can be found here:
`https://pure.tudelft.nl/admin/files/247331433/survival-of-the-fittest.pdf`.

TU Delft Delft University of Technology