EasyChair Preprint

№ 15890

# Applying Genetic Algorithm with Saltations to MAX-3SAT

Ryan Alomair, Hafsa Farooq, Daniel Novikov, Akshay Juyal and Alex Zelikovsky



March 6, 2025

# Applying Genetic Algorithm with Saltations to MAX-3SAT

Ryan Alomair[0009−0007−5605−6790], Hafsa Farooq[0009−0002−6260−5016], Daniel Novikov[0000−0003−2126−0197], Akshay Juyal, Alexander Zelikovsky[0000−0003−4424−4691]

Department of Computer Science, Georgia State University, Atlanta, GA, 30303, USA

**Abstract.** Punctuated evolution, (synonymous with Saltations, Evolutionary Jumps)—the pattern of rapid, significant mutational change—had not been observed in real-time until SARS-CoV-2 viral variants emerged with multiple mutations occurring together. By using Epistasis as a framework to understand this phenomenon, where the effect of one mutation depends on the influence of one or more other mutations (ie. combinations of mutations) we can model the fitness landscape of viral variants with an Epistatic network, and capture this relationship between different combinations of mutations as a result[11]. In exploring these relationships, it has been found that dense subgraphs (where the density of the subgraph increases with the number of edges, given a number of vertices) within the network correspond to emerging saltations, which can uncover high fitness regions seemingly distant from the variant(s) they originally derived from[11]. We incorporate this pattern into the Genetic Algorithm (GA+EJ) as a means to produce new solutions that escape the inherent tendency to produce solutions converging to a local maximum. We applied GA+EJ to the MAX-3SAT problem, and found improvement for satisfiable problem instances with 600 variables and 2550 clauses, as well as 100 variables and 429 clauses, allowing us to find solutions giving a better approximation of the optimum when using jumps, than without them.

**Keywords:** MAX-3SAT Problem · Evolution · Genetic Algorithm · Epistatic network ·

## 1 Introduction

SARS-CoV-2 variants took the world by storm in November 2020, necessitating the exploration of new methodologies to predict new variants. Variants like Alpha and Omicron presented unprecedented challenges to scientists, as the virus had undergone multiple mutations that seemed to occur simultaneously before coming to fruition as Variants of Concern (VOCs)[5][1][2]. While it is not unusual for a virus to mutate multiple times before being passed on, it is uncommon for a variant to retain multiple mutations that, only when present together, translate to phenotypic effects granting higher fitness. Multiple studies have consequently

tried to debunk this phenomenon, explaining it through the framework of epistasis[18][20][17][16][19][14][12], where interactions between multiple mutations can play a role in shaping the viral fitness landscape[11][10]. Rather than evolving through a gradual accumulation of independent mutations with additive effects, SARS-CoV-2 exhibits punctuated evolution, characterized by sudden shifts in viral fitness due to epistatic interactions[11][15].

This dynamic can be effectively modeled using an epistatic network, in which mutations form interconnected nodes, and newly emerging, highly fit variants correspond to dense subgraphs within this network[11]. These dense subgraphs represent clusters of co-occurring mutations that collectively reflect higher viral fitness, facilitating rapid evolutionary jumps. Such changes contrast with traditional models of gradual evolution, where selective pressures incrementally refine individual mutations over time. Inspired by this evolutionary phenomenon, we incorporate punctuated evolution into the genetic algorithm.

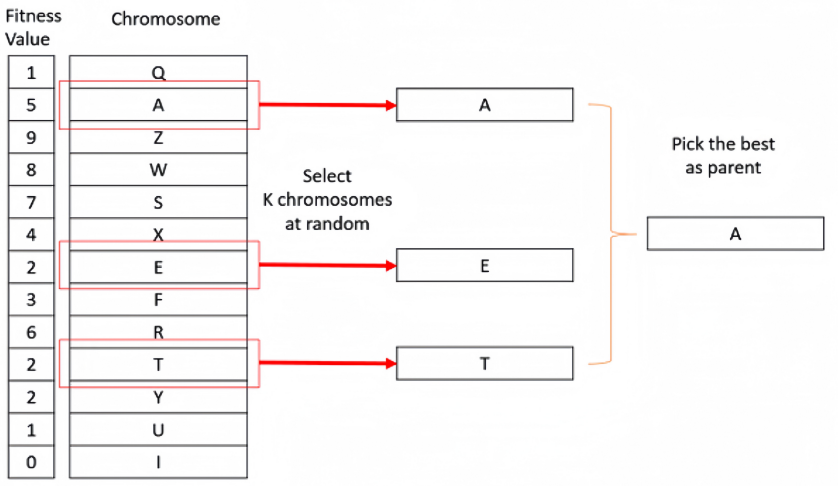
The genetic algorithm (GA) is an optimization technique inspired by natural selection, where a population of candidate solutions evolves through iterative processes of natural selection (based on a fitness function) genetic crossover, and mutation[9]. Once an initial population of randomized solutions is produced, higher-fitness solutions are selected via tournament selection, which will then be used to produce offspring through the process of genetic crossover and mutation. Subsequently, properties (variable assignments) of higher-fitness solutions propagate over successive generations, while properties of lower fitness solutions are eliminated. As a result, each solution's variable assignments become increasingly homogenuous over time, stagnating in local fitness peaks without exploring potentially superior alternatives[7]. To counter this inherent fault, we use Clique-SNV to map solutions to an epistatic network and simulate saltations. These new solutions are then inserted directly into the mating pool, allowing their properties to propagate to the next generation and effectively incorporate a tailored diversity to the gene pool[3]. By incorporating these solutions, GA can break out of its premature convergence, and explore variable assignments corresponding to solutions in other high fitness neighborhoods of the epistatic network[3].

Section 2 discusses the general process of GA and subsequent parts (k-tournament selection, genetic crossover and mutation). Section 3 goes on to discuss SARS-CoV-2/Epistasis/Punctuated Evolution, and how these concepts manifest in our use of Clique-SNV. Section 3 then goes on to describe how we integrate Clique-SNV into simple genetic algorithm, to produce GA+EJ. Section 3 finishes with a discussion of GA+EJ as it is applied to the MAX-3SAT problem. Section 4 highlights more specific details on our implementation and parameter selection. Section 5 covers our problem instances, results/analysis, and finishes with our concluding notes/future plans.

## 2 Genetic Algorithm

### 2.1 Simple Genetic Algorithm

GA begins by initializing a randomized population of solutions for a given population size, where each solution is a binary bit string consisting of 1s and 0s. Each bit represents a boolean value that will be assigned to a corresponding variable in the problem being solved, meaning the length of each solution is equal to the number of variables in the problem instance. Each solution is then passed into a fitness function, which determines how well the solution solves the problem instance. Once a fitness value is determined for each solution, some are chosen to be part of the mating pool based on a k-tournament selection (simulating natural selection). A total of k solutions (input parameter) will be randomly selected, and the one with the highest fitness will be selected for the mating pool. That selection process will repeat until the mating pool reaches a predetermined size (input parameter)[9].




[13]

**Fig. 1.** Tournament selection performed for tournament size k=3. The solution of highest fitness between the k solutions is selected for the mating pool.

Solutions from the mating pool are then randomly selected in pairs to produce a pair of offspring solutions. The parent solutions will first undergo a single point genetic cross-over: a random index p is chosen, and the variable assignments leading up to and including p are then taken from parent1 and given to

child1, and the same for parent2 with child2. Then, the variable assignments beginning at p+1 until the end are taken from parent2 and given to child1, and vice versa with parent1 and child2. As a result, each child solution will have a complementing variable assignment of each parent solution, based on the randomly selected point p (note: this is often extended to a k-point crossover, where k indices are selected, and child solutions contain alternating portions of each parent solution, still forming a complement to each other)[9].

Next, each child solution undergoes mutation, where bits are flipped at random based on a mutation probability (input parameter) as a means to help maintain/control the amount of diversity with each solution. This process of producing two child solutions at a time is repeated until the population size is met, forming a new generation. Once the new generation of offspring is ready, each of their fitness values is calculated, and the process repeats continuously until a threshold number of generations are produced, a threshold number of generations without improvement of global fitness is reached, or a predetermined optimum global fitness is reached[9].

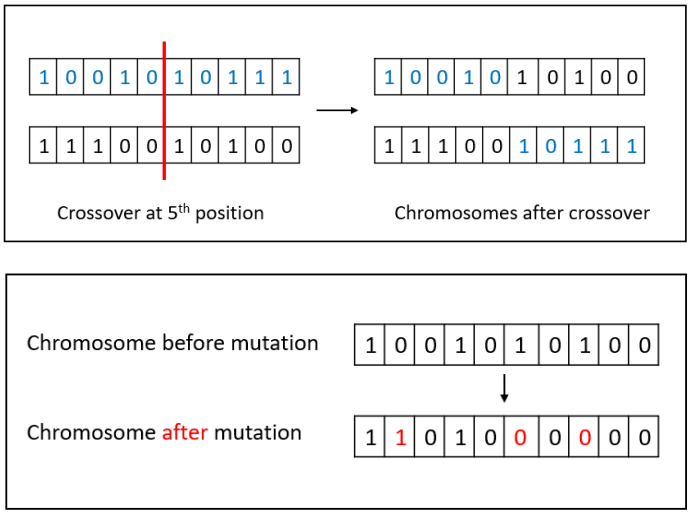



**Fig. 2.** a) Two individuals are performing crossover. Red line represents the point $p$=5 of crossover. b) Mutation is performed on the 2nd, 6th and 8th genes of the individual.

Although the above approaches for parent selection, crossover and mutation are common for GA, these methods can be implemented in different ways, and sometimes have different approaches to producing offspring solutions altogether[9]. Ultimately, the approach must be tailored to the problem which GA solutions are being produced for. In this study, solutions are crafted for the MAX-3SAT problem, warranting a modified approach to our crossover and mutation functions, which are described in section 3.4.

## 3 Genetic Algorithm with Evolutionary Jumps

### 3.1 Epistasis, Punctuated Evolution and SARS-CoV-2

The evolution of SARS-CoV-2 variants can be understood through the lens of epistasis, which describes the non-additive interactions between mutations that influence viral fitness[18][20][17][16][19][14][12]. Epistasis constrains the set of viable genomic variants and, consequently, restricts the potential evolutionary trajectories of the virus. This constraint can be formalized using graph-theoretical frameworks, where viable genotypes form a structured subgraph within a hypercube, known as the viable space. Within this space, a genotype is considered viable if its constituent mutations collectively form a maximal clique in the underlying epistatic network. As a result, viral evolution does not proceed through a smooth and continuous exploration of all possible mutations, but is channeled along a discrete set of constrained pathways defined by these cliques instead[11].

This framework provides a powerful means of predicting high-fitness variants by identifying the restricted evolutionary trajectories accessible to the virus[11]. Importantly, these constraints align with observations of punctuated evolution, wherein viral lineages appear to undergo episodic bursts of adaptation rather than gradual, incremental change[15][2]. Such shifts may correspond to transitions between distinct regions of the viable space, where a set of epistatic interactions must be co-acquired before a new, highly fit genotype emerges. By leveraging this network-based understanding of epistasis, it becomes possible to prioritize specific mutational combinations for functional screening, ultimately aiding in the forecasting of future SARS-CoV-2 variants with high fitness potential[11].

### 3.2 Clique-SNV

The Clique-SNV framework can identify epistatic interactions among single nucleotide variants (SNVs) using a graph-theoretical approach. It first classifies SNV pairs as linked, forbidden, or unclassified based on their observed co-occurrence in sequencing reads. If a 2-haplotype appears significantly more often than expected by sequencing noise, it is classified as linked, meaning the variants likely co-occur in a haplotype. If its occurrence is significantly lower than expected, the pair is forbidden, indicating they are unlikely to be found together. All other pairs remain unclassified until further evaluation[6].

Next, Clique-SNV constructs an SNV graph G=(V,E) where the vertices (V) represent minor variants and the edges (E) connect the linked SNVs. This graph can capture structural constraints on co-occurring mutations. Since many isolated variants arise from sequencing errors, filtering these out refines the SNV graph and improves accuracy[6].

Finally, Clique-SNV identifies maximal cliques within the SNV graph using the Bron–Kerbosch algorithm. A clique represents a set of SNVs that are all pairwise linked, meaning they are likely to co-occur in a viable haplotype [6]. By decomposing the SNV graph into cliques, Clique-SNV can reveal constrained

evolutionary pathways, showing how certain mutations must co-occur for viability. This framework helps predict high-fitness variants and provides insight into the structured nature of viral evolution[3].

### 3.3 Integrating Clique-SNV into GA

Integrating Clique-SNV into the genetic algorithm (GA) allows for the detection of epistatic interactions within evolving solutions. As GA progresses, solutions to accumulate across generations and are converted to FASTA format, where 0s (reference alleles) are encoded as A's and 1s (mutations) as C's. Clique-SNV then analyzes all generated solutions, identifying statistical links between frequently co-occurring variable assignments and constructs an epistatic network.

In this network, nodes represent variable assignments (e.g., a particular position being 0 or 1), while edges form between assignments that frequently co-occur in high-fitness solutions. Note that properties of high-fitness solutions naturally persist through generations, causing them to gain greater influence in the network and reinforcing the statistical associations between their variable assignments. Strongly connected components reveal dependencies between variable states, constraining viable evolutionary pathways. The maximal cliques in this network then naturally emerge as sets of assignments that consistently appear together in viable solutions[3].

Just as dense subgraphs in SARS-CoV-2's epistatic network reveal mutations that co-occur and constrain evolutionary trajectories, Clique-SNV detects maximal cliques in the GA's epistatic network, identifying groups of variable assignments that frequently appear in high-fitness solutions.

By leveraging these maximal cliques, Clique-SNV can construct new candidate solutions by recombining linked variable assignments, simulating evolutionary jumps in the search space—just as epistatic constraints in viral evolution can give rise to saltations[3]. Thus, we use Clique-SNV to identify key epistatic interactions and facilitate punctuated evolution, improving the GA's ability to efficiently explore the search space beyond recombination and random mutation.

### 3.4 GA+EJ Applied to the MAX-3SAT Problem

The MAX-3SAT problem is a well-known NP-hard optimization problem in which the objective is to find a variable assignment that maximizes the number of satisfied clauses in a Boolean formula. The formula is expressed in conjunctive normal form (CNF), where each clause consists of exactly three literals (variables or their negations) connected by logical ORs. The challenge lies in determining an assignment of Boolean values (true or false) to the variables that satisfies as many clauses as possible.

In this implementation, problem instances are provided in CNF files, which specify the set of clauses and variables for a given MAX-3SAT problem. Each candidate solution in the genetic algorithm represents a variable assignment, where every variable in the formula is assigned either 0 (false) or 1 (true). These assignments are then plugged into the CNF formula, and the fitness function

evaluates their quality simply by counting the number of satisfied clauses. Higher fitness values correspond to assignments that satisfy more clauses, guiding the search toward optimal solutions.

The crossover function used in GA differs from the single-point crossover discussed earlier. Since the order of bits in each solution determines the variable assignment, we implement our crossover function differently than how it was described in section 2 (the general case). Instead of splitting the parent solutions at a fixed point, the crossover function preserves all variable assignments that are identical between the two parent solutions in both child solutions. For variables where the parents differ, one child solution inherits the assignment from the first parent, while the other child takes the assignment from the second parent. This approach ensures that shared structure between high-fitness solutions is maintained while still introducing recombination between for some variable assignments.

The mutation function remains the same as previously described, applying small random changes to individual variable assignments to maintain genetic diversity and prevent premature convergence. By incorporating evolutionary jumps through Clique-SNV, the algorithm enhances its ability to escape local optima and explore new high-fitness regions of the search space efficiently.

### 3.5 Incorporating Evolutionary Jumps

To insert solutions corresponding to evolutionary jumps, we wait for 10 generations to be produced without any improvement in global fitness, then call Clique-SNV, which returns several EJ solutions that are inserted directly into the mating pool for the next generation. Since EJ solutions are constructed using statistically linked variable assignments in solutions that have already been produced, it is important to allow enough generations (of no improvement) to be produced prior to calling Clique-SNV, such that these relationships can be discovered. That said, we also must not wait for too many generations to pass, since this allows suboptimal/convergent solutions to occupy an increasingly larger share of the cumulative solution space being used to construct the epistatic network.

## 4 Implementation Specifics

### 4.1 Pygad as our Base Implementation

The implementation of the genetic algorithm is built using the PyGAD library, which provides a flexible and efficient framework for developing evolutionary algorithms[4]. PyGAD offers a wide array of customizable features, making it an effective tool for fine-tuning GA behavior. Its built-in mechanisms for selection, crossover, and mutation allow for seamless experimentation with different evolutionary strategies, but also give us freedom to implement these on our own. Additionally, PyGAD's callback functions, such as on_fitness and on_parents, enabled

the straightforward integration of Clique-SNV into the algorithm. By leveraging these functions, we could dynamically analyze evolving solutions, construct the epistatic network, and introduce evolutionary jumps without disrupting the standard workflow of GA.

### 4.2 Parameters

To ensure optimal performance of the genetic algorithm, I carefully tuned various parameters to maximize the total number of clauses satisfied. The parameters adjusted during this process included population size, mating pool size, tournament size, mutation type, mutation rate, crossover method, and selection strategy. Each of these factors played a crucial role in determining how effectively the algorithm explored the solution space and converged on high-fitness solutions. Additionally, Clique-SNV required meticulous tuning of its own parameters to accurately capture epistatic interactions and facilitate effective evolutionary jumps. These adjustments were essential to balancing exploration and exploitation in the search process. A summary of all finalized parameters is provided in the Table 1, shown below, followed by a description of the tuning process for some individual parameters.1

**Table 1.** Parameters Table

| **GA Parameter** | **Value** | | **C-SNV Parameter** | **Value** |
|---|---|---|---|---|
| Population Size | (1308),(256) | | Jump Threshold | 10 |
| Max. Gens No Improvement | 300 | | Min. Gen. Wait | 20 |
| Num of Parents | (28,38,32,50),(128) | | Jump Type | Global Best |
| Num CPUs | 16 | | C-SNV Timeout | 1200sec |
| Parent Type | Tournament | | Threshold.Freq | 0.01 |
| K-Tournament | (35,26,41,26),(2) | | Threshold.Freq+ | 0.01 |
| Mutation Type | Random | | Memory | 96GB |
| Mutation Pr | 0.01 | | Edge Limit | 1000 |

1. Population size: 1308 Solutions
   - While ensuring the population size was large enough to effectively explore a wide solution space, making it too large came at a cost to overall performance and quality of EJ solutions.
   - By using a solution space too large, many generations would pass before there were enough consecutive generations without improvement. Consequently, the cumulative store of solutions became overloaded with those consisting of a suboptimal variable assignments to which the solutions were converging. As a result, EJ solutions did not introduce enough diversity in order to escape local optimums.

2. Mating pool and tournament size
   - The mating pool size was coordinated with the tournament size to ensure a particular portion of the solution space would be explored. Tables 2 and 3 show which parameters correspond to which population sizes, shown below:

| Mating Pool Size | Tournament Size | % Soln. Space Explored |
|---|---|---|
| 28 | 35 | ∼53% |
| 38 | 26 | ∼54% |
| 32 | 41 | ∼64% |
| 50 | 26 | ∼64% |

**Table 2.** For Population Size of 1308 solutions

| Mating Pool Size | Tournament Size | % Soln. Space Explored |
|---|---|---|
| 128 | 2 | ∼63% |

**Table 3.** For Population Size of 256 solutions

3. Max gens no improvement: 300
   - I allow the GA+EJ and simple GA both to run until they have produced 300 generations without improvement in optimum fitness. This value was chosen as the result of stringent testing to see how many generations GA+EJ can produce without improvement before eventually reaching its optimum. Simple GA generally reaches its optimum faster (due to its premature convergence) but we allow both to run this long to remain consistent.
4. Clique-SNV
   - True Frequency
     - This value represents the assumed lower bound on the actual occurrence rate of a given 2-haplotype (22) in the dataset. It is used to control the probability of falsely identifying linked variable pairs by ensuring that observed read counts are statistically significant.
     - The probability of observing at least $x \geq O_{22}$ reads given that the true frequency is at most t is constrained to be low enough to prevent false-positive associations. This threshold is set such that the expected number of false positives remains below 0.05 / $\binom{L}{2}$, where $L$ is the haplotype length.
     - In my implementation, I initially set true frequency to 0.01, but if the number of edges produced in the epistatic network exceeds the predefined edge limit, I increment true frequency by 0.01 and rerun Clique-SNV. This adjustment controls the density of the network by filtering out weaker associations, preventing an excessive number of edges from forming. Maintaining a strict edge limit is crucial because the time required to find maximal cliques increases exponentially with the number of vertices in the network.

- Edge Limit
  - To manage computational feasibility, I set the edge limit to 1000, as this ensures that maximal cliques can typically be found within 10 minutes.

# 5 Results

## 5.1 MAX-3SAT Problem Instances

In MAX-3SAT problem instances, the number of clauses and variables plays a crucial role in determining the difficulty of finding a satisfying assignment. The challenge of solving all clauses—meaning the instance is fully satisfiable—largely depends on the clause-to-variable ratio. When this ratio is low, the problem tends to be easier because there are fewer constraints on variable assignments. Conversely, as the ratio increases, the constraints become more restrictive, making it significantly harder to find an assignment that satisfies all clauses. Research has shown that for random 3SAT instances, the satisfiability threshold occurs at a clause-to-variable ratio of approximately $\sim$4.267[8]. At this critical ratio, problem instances transition from being mostly satisfiable to mostly unsatisfiable when adding more clauses, making them particularly challenging for both exact and heuristic solvers.

To maintain this complexity, we test two problem instances, one with 2,550 clauses and 600 variables and another with 429 clauses and 100 variables, both closely adhering to the 4.267 ratio. This ensures that the problem remains difficult enough to test the effectiveness of GA while still allowing room for high-fitness solutions to emerge. This is a solvable instance, meaning there exists at least one variable assignment which allows all clauses to be solved (2,550 and 429 clauses is the optimum for each respective instance). By operating near the satisfiability threshold, the instance presents a strong challenge, requiring the genetic algorithm to efficiently navigate the search space to maximize the number of satisfied clauses.

## 5.2 Hard MAX-3SAT Instances

Initially, we aimed to categorize problem instances into "hard" and "easy" based on their structure. However, in the context of Max-3SAT, such a distinction does not hold in a meaningful way. While some instances contain a smaller proportion of satisfiable clauses, our tests showed that both GA+EJ and simple GA performed similarly relative to the optimum (the number of maximum number of satisfiable clauses) regardless of satisfiability constraints. This suggests that the presence of fewer satisfiable clauses does not inherently make an instance harder in a way that affects the comparative performance of our algorithms, rendering this distinction uninformative for our analysis.

### 5.3 GA+EJ vs. Simple GA

The following two tables contain performance scores for simple GA and GA+EJ based on 5 runs, where a score is the difference in the number of clauses satisfied by the algorithm and the optimum (2,550 and 429, since both problem instances are satisfiable). This metric was chosen because it provides the most clear measure of how close each algorithm comes to achieving the optimum. Additionally, the number of generations is reported as the generation at which each algorithm reached its optimum fitness value, ensuring that comparisons reflect the point at which the best solution was found rather than when the algorithm terminated.

| **2550 Clauses / 600 Variables** | | | **GA+EJ** | | **Simple GA** | |
|---|---|---|---|---|---|---|
| **Measure** | **Parents** | **Tourn. Size** | **Clauses Unsat** | **Generations** | **Clauses Unsat** | **Generations** |
| Best | 28 | 35 | 71 | 358 | 72 | 217 |
| Worst | 28 | 35 | 89 | 69 | 84 | 35 |
| Average | 28 | 35 | 78 | 217.2 | 79.4 | 214.6 |
| Best | 38 | 26 | 52 | 85 | 58 | 240 |
| Worst | 38 | 26 | 64 | 196 | 80 | 67 |
| Average | 38 | 26 | 58 | 258.4 | 68 | 161.2 |
| Best | 32 | 41 | 67 | 98 | 63 | 106 |
| Worst | 32 | 41 | 81 | 102 | 87 | 68 |
| Average | 32 | 41 | 75.6 | 117.8 | 78 | 187.4 |
| Best | 50 | 26 | 44 | 323 | 47 | 320 |
| Worst | 50 | 26 | 58 | 272 | 62 | 165 |
| Average | 50 | 26 | 53.8 | 156.6 | 53.8 | 300.2 |

**Table 4.** Performance scores measured by number of clauses left unsatisfied. Scores calculated over 5 runs.

| **429 Clauses / 100 Variables** | | | **GA+EJ** | | **Simple GA** | |
|---|---|---|---|---|---|---|
| **Measure** | **Parents** | **Tourn. Size** | **Clauses Unsat** | **Generations** | **Clauses Unsat** | **Generations** |
| Best | 128 | 2 | 3 | 36 | 5 | 37 |
| Worst | 128 | 2 | 5 | 44 | 7 | 49 |
| Average | 128 | 2 | 4.2 | 40 | 6 | 42 |

**Table 5.** Performance scores measured by number of clauses left unsatisfied. Scores calculated over 5 runs.

### 5.4 GA+EJ vs. Simple GA: Analysis

The results in Tables 3 and 4 demonstrate that the genetic algorithm with evolutionary jumps (GA+EJ) generally outperforms the simple genetic algorithm (GA) in solving the Max3SAT problem. In the larger problem instance , GA+EJ achieves a lower number of unsatisfied clauses across best, worst, and average cases in most configurations. For example, with 38 parents and a tournament size of 26, GA+EJ's best solution leaves 52 clauses unsatisfied, compared to 58

for the simple GA—an improvement of 10.3%. Similarly, in the worst case for this configuration, GA+EJ leaves 64 clauses unsatisfied, while the simple GA leaves 80, a 20% reduction. On average, GA+EJ leaves 14.7% fewer unsatisfied clauses (58 vs. 68).

The smaller problem instance shows an even more pronounced advantage for GA+EJ. In the best case, GA+EJ leaves only 3 clauses unsatisfied, compared to 5 for the simple GA, a 40% reduction. The worst case follows a similar trend, with GA+EJ leaving 5 clauses unsatisfied versus 7 for the simple GA (28.6% fewer). On average, GA+EJ results in 4.2 unsatisfied clauses, compared to 6 for the simple GA, marking an overall improvement of 30%.

Although, there are instances where GA+EJ does not consistently outperform simple GA. For example, with 32 parents and a tournament size of 41, the best-case result for GA+EJ is slightly worse (67 unsatisfied clauses) compared to the simple GA (63 unsatisfied clauses). Additionally, in the same configuration, GA+EJ's worst-case result (81 unsatisfied clauses) is only marginally better than the simple GA's worst case (87), but its average (75.6) is still close to that of the simple GA (78), suggesting less improvement in this scenario.

Additionally, GA-EJ does not consistently reduce the number of generations required to reach high-fitness solutions. In some cases, such as the 28/35 and 32/41 settings, GA-EJ requires more generations on average than the standard GA, despite achieving slightly higher or similar fitness values. This suggests that while evolutionary jumps help maintain a higher fitness ceiling, they do not always accelerate convergence. Instead, they may introduce additional variability, allowing the algorithm to explore alternative solutions rather than prematurely converging.

However, it is important to note that both algorithms have a predefined threshold of 300 generations without improvement in global fitness before they automatically stop running. As a result, the total number of generations before termination is often larger for the genetic algorithm with evolutionary jumps. This occurs because evolutionary jumps periodically reintroduce targeted diversity into the population, sometimes causing the algorithm to run for many additional generations before reaching its final optimum. In contrast, the genetic algorithm without evolutionary jumps tends to converge prematurely; once an optimal solution is reached at a particular generation, no further improvements occur, and the algorithm simply continues for the remaining 300 stagnation generations before termination.

While the benefits of GA+EJ are sometimes configuration dependent, its overall effectiveness in producing solutions closer to the optimal is evident. These results indicate that GA+EJ generally enhances genetic diversity and helps the algorithm escape local optima, with it outperforming simple GA in 12 of the 15 scores reported, demonstrating its consistent ability to find solutions with fewer unsatisfied clauses. Additionally, in the 429-clause problem, GA+EJ outperformed the simple GA in every measured case, achieving up to a 40% reduction in unsatisfied clauses. Even in the larger 2550-clause problem, where solution space complexity increases, GA+EJ produced better results in 9 out of 12 cases.

Although there were a few instances where the simple GA performed similarly or marginally better, GA+EJ's more frequent success in reducing clause violations indicates a stronger ability to approximate the optimal solution. Its advantage is most evident in cases where the standard GA requires more generations to converge, suggesting that evolutionary jumps help escape local optima and drive long-term improvements in fitness. These findings reinforce the conclusion that incorporating evolutionary jumps enhances the genetic algorithm's exploration of the solution space, ultimately leading to superior optimization performance in the Max3SAT problem.

## 6 Conclusion

The results demonstrate a marginal yet consistent improvement in the performance of the genetic algorithm when incorporating evolutionary jumps, both in achieving higher fitness and reaching high-fitness solutions in fewer generations. This enhancement aligns with the principles of punctuated evolution, where populations experience periods of stasis followed by sudden leaps in adaptation. By leveraging Clique-SNV to identify and exploit epistatic interactions, the algorithm introduces structured genetic changes that mimic these evolutionary jumps, reintroducing tailored diversity into the mating pool and helping escape local optima. While the improvement is sometimes limited, it underscores the potential of integrating biological principles into evolutionary algorithms to enhance their efficiency in solving complex optimization problems like MAX-3SAT. Building on these findings, we plan to extend GA+EJ to other optimization challenges, such as the Traveling Salesman Problem, MAX-SAT instances, and the multidimensional knapsack. Additionally, we aim to further optimize performance by parallelizing the algorithm on GPUs, paving the way for even more efficient and scalable implementations.

## References

1. Andre, M., Lau, L.S., Pokharel, M.D., Ramelow, J., Owens, F., Souchak, J., Akkaoui, J., Ales, E., Brown, H., Shil, R., Nazaire, V., Manevski, M., Paul, N.P., Esteban-Lopez, M., Ceyhan, Y., El-Hage, N.: From alpha to omicron: How different variants of concern of the sars-coronavirus-2 impacted the world. Biology (Basel) **12**(9) (Sep). https://doi.org/10.3390/biology12091267
2. Corey, L., Beyrer, C., Cohen, M.S., Michael, N.L., Bedford, T., Rolland, M.: Sars-cov-2 variants in patients with immunosuppression. N Engl J Med **385**(6), 562–566 (Aug 2021). https://doi.org/10.1056/NEJMsb2104756
3. Farooq, H., Novikov, D., Juyal, A., Zelikovsky, A.: Genetic Algorithm with Evolutionary Jumps, pp. 453–463 (10 2023). https://doi.org/10.1007/978-981-99-7074-$2_3$6
4. Gad, A.F.: Pygad: An intuitive genetic algorithm python library. Multimedia Tools and Applications pp. 1–14 (2023)
5. Ingraham, N.E., Ingbar, D.H.: The omicron variant of sars-cov-2: Understanding the known and living with unknowns. Clin Transl Med **11**(12), e685 (Dec 2021). https://doi.org/10.1002/ctm2.685

6. Knyazev, S., Tsyvina, V., Shankar, A., Melnyk, A., Artyomenko, A., Malygina, T., Porozov, Y.B., Campbell, E.M., Switzer, W.M., Skums, P., Mangul, S., Zelikovsky, A.: Accurate assembly of minority viral haplotypes from next-generation sequencing through efficient noise reduction. Nucleic Acids Research **49**(17), e102–e102 (07 2021). https://doi.org/10.1093/nar/gkab576
7. Leung, Y., Gao, Y., Xu, Z.B.: Degree of population diversity - a perspective on premature convergence in genetic algorithms and its markov chain analysis. IEEE Transactions on Neural Networks **8**(5), 1165–1176 (1997). https://doi.org/10.1109/72.623217
8. Mézard, M., Zecchina, R.: Random $k$-satisfiability problem: From an analytic solution to an efficient algorithm. Phys. Rev. E **66**, 056126 (Nov 2002)
9. Mitchell, M.: An introduction to genetic algorithms. MIT press (1998)
10. Mohebbi, F., Zelikovsky, A., Mangul, S., Chowell, G., Skums, P.: Community structure and temporal dynamics of sars-cov-2 epistatic network allows for early detection of emerging variants with altered phenotypes. bioRxiv pp. 2023–04 (2023)
11. Mohebbi, F., Zelikovsky, A., Mangul, S., Chowell, G., Skums, P.: Early detection of emerging viral variants through analysis of community structure of coordinated substitution networks. Nature Communications **15**(1), 2838 (2024). https://doi.org/10.1038/s41467-024-47304-6
12. Moulana, A., Dupic, T., Phillips, A.M., Chang, J., Nieves, S., Roffler, A.A., Greaney, A.J., Starr, T.N., Bloom, J.D., Desai, M.M.: Compensatory epistasis maintains ace2 affinity in sars-cov-2 omicron ba.1. Nat Commun **13**(1), 7011 (Nov 2022). https://doi.org/10.1038/s41467-022-34506-z
13. NA: Tournament$_s$*election*(2023), *https* : *//www.tutorialspoint.com/genetic$_a$lgorithms/images/tournament$_s$election.j*
14. Neverov, A.D., Fedonin, G., Popova, A., Bykova, D., Bazykin, G.: Coordinated evolution at amino acid sites of sars-cov-2 spike. Elife **12** (Feb 2023). https://doi.org/10.7554/eLife.82516
15. Nielsen, B.F., Li, Y., Sneppen, K., Simonsen, L., Viboud, C., Levin, S.A., Grenfell, B.T.: Immune heterogeneity and epistasis explain punctuated evolution of sars-cov-2. medRxiv (Jul 2022). https://doi.org/10.1101/2022.07.27.22278129
16. Rochman, N.D., Faure, G., Wolf, Y.I., Freddolino, L., Zhang, F., Koonin, E.V.: Epistasis at the sars-cov-2 receptor-binding domain interface and the propitiously boring implications for vaccine escape. mBio **13**(2), e0013522 (Apr 2022). https://doi.org/10.1128/mbio.00135-22
17. Rochman, N.D., Wolf, Y.I., Faure, G., Mutz, P., Zhang, F., Koonin, E.V.: Ongoing global and regional adaptive evolution of sars-cov-2. Proc Natl Acad Sci U S A **118**(29) (Jul 2021). https://doi.org/10.1073/pnas.2104241118
18. Rodriguez-Rivas, J., Croce, G., Muscat, M., Weigt, M.: Epistatic models predict mutable sites in sars-cov-2 proteins and epitopes. Proc Natl Acad Sci U S A **119**(4) (Jan 2022). https://doi.org/10.1073/pnas.2113118119
19. Zahradník, J., Marciano, S., Shemesh, M., Zoler, E., Harari, D., Chiaravalli, J., Meyer, B., Rudich, Y., Li, C., Marton, I., Dym, O., Elad, N., Lewis, M.G., Andersen, H., Gagne, M., Seder, R.A., Douek, D.C., Schreiber, G.: Sars-cov-2 variant prediction and antiviral drug design are enabled by rbd in vitro evolution. Nature Microbiology **6**(9), 1188–1198 (2021). https://doi.org/10.1038/s41564-021-00954-4
20. Zeng, H.L., Dichio, V., Rodríguez Horta, E., Thorell, K., Aurell, E.: Global analysis of more than 50,000 sars-cov-2 genomes reveals epistasis between eight viral genes. Proc Natl Acad Sci U S A **117**(49), 31519–31526 (Dec 2020). https://doi.org/10.1073/pnas.2012331117