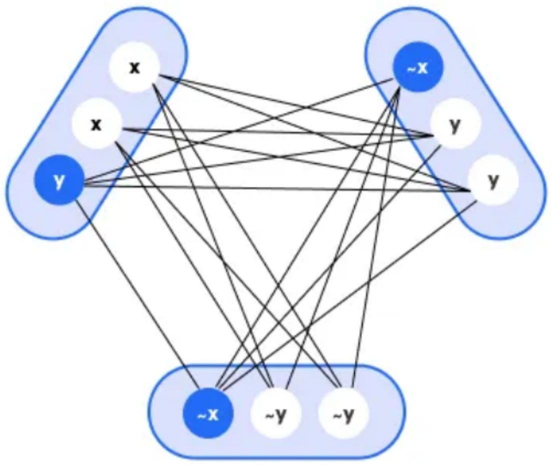
INFO—F413
Randomized Algorithms — Maximum Satisfaction
Project Report

Berthion Antoine — 566199

November 29, 2025



Abstract

This project presents the implementation of a *Las Vegas* algorithm for the **Maximum 3-SAT problem**, which guarantees finding an assignment that satisfies at least $\frac{7}{8}$ of the clauses. We provide both **theoretical** and **experimental** analyses of its performance, and we establish its guarantees regarding runtime. Additionally, we discuss **design choices** and **implementation details**.

1 Introduction

The **Boolean satisfiability problem** (SAT) is a cornerstone of computational complexity and combinatorial optimization. In its most general form, **SAT** asks whether there exists an **assignment of truth values** to variables that satisfies a given Boolean formula. Among its variants, `Maximum 3-SAT` (MAX-3SAT) seeks an assignment that maximizes the number of satisfied clauses, where each clause contains **at most three literals**. Efficiently solving MAX-3SAT is crucial in applications ranging from hardware verification to combinatorial optimization problems.

Exact algorithms for MAX-3SAT are typically intractable for large instances, motivating the use of **randomized algorithms**. These methods trade **deterministic guarantees** for **probabilistic ones**, allowing faster solutions while ensuring high-quality **approximations**.

In particular, **Las Vegas algorithms** provide a compelling approach: they always return a solution satisfying at least a guaranteed fraction of clauses (here, **at least** $\frac{7}{8}$), while the runtime remains random. This makes them especially suitable for MAX-3SAT, where finding near-optimal assignments efficiently is often more practical than exact solutions.

2 Formalization and Algorithm

We now formalize the `Maximum 3-SAT` problem and present the **Las Vegas algorithm** used in this project.

2.1 Maximum 3-SAT Problem

Let $X = \{x_1, \ldots, x_n\}$ denote a set of Boolean variables, and let $\mathcal{C} = \{C_1, \ldots, C_m\}$ denote a set of clauses, each containing at most **three literals**. A literal is either a variable x_i or its negation $\neg x_i$. An assignment $A : X \to \{\text{true}, \text{false}\}$ satisfies a clause C_j if at least one literal in C_j evaluates to true. The MAX-3SAT problem consists in finding an assignment that maximizes the number of satisfied clauses.

2.2 Las Vegas Algorithm

The Las Vegas algorithm generates **random assignments** until one satisfies at least 7/8 of the clauses:

1. Assign each variable **independently** to `True` or `False`.
2. Count the number of satisfied clauses.
3. If at least $\frac{7}{8}m$ clauses are satisfied, return the assignment.
4. Otherwise, repeat from step 1.

Assignments are stored as mappings from variables to Boolean values, and clause evaluation checks whether at least one literal in each clause is true. This structure ensures correctness while keeping the implementation simple and efficient. See Appendix A: for the implementation of this algorithm.

3 Theoretical Analysis and Bounds

3.1 Expected Fraction of Satisfied Clauses

For a random assignment A, each clause C_j is unsatisfied only if all three of its literals are false. Therefore, the probability that C_j is satisfied is:

$$\Pr[C_j \text{ satisfied}] = 1 - \frac{1}{2^3} = \frac{7}{8} \tag{1}$$

By **linearity of expectation**, the expected number of satisfied clauses is:

$$E[\texttt{sat}(A)] = \frac{7}{8}m \tag{2}$$

as first observed by **Karloff and Zwick** [1].

3.2 Concentration via McDiarmid's Inequality

The number of satisfied clauses is a function of independent random variables (the truth values of each variable). Changing a single variable affects **at most d clauses**. **McDiarmid's inequality** [2] then ensures that $\texttt{sat}(A)$ is sharply concentrated around its expectation:

$$\Pr\left[\left|\texttt{sat}(A) - E[\texttt{sat}(A)]\right| \geq \varepsilon\right] \leq 2\exp\left(-\frac{2\varepsilon^2}{nd^2}\right), \tag{3}$$

where n is the number of variables. This implies that a random assignment satisfying at least 7/8 of clauses occurs with **high probability**.

3.3 Runtime Analysis

Let p denote the probability that a random assignment satisfies at least 7/8 of clauses. The number of iterations T until success follows a **geometric distribution** with success probability p, giving expected iterations

$$E[T] = \frac{1}{p}. \tag{4}$$

Each iteration evaluates all m clauses, giving per-iteration cost $O(m)$ and expected total runtime $O(m)$. While the worst-case runtime is **unbounded**, the probability of needing many iterations is **exponentially small** due to concentration, making the algorithm practically **efficient**.

3.4 Summary

The Las Vegas algorithm for `MAX-3SAT`:

— Guarantees an assignment satisfying at least 7/8 of clauses.

— Has expected runtime $O(m)$ due to high probability of success per iteration.

— Exhibits strong concentration around the expected number of satisfied clauses, ensuring predictable performance in practice.

4 Experiments

This section presents the **objectives**, **methodology**, and results of our experimental analysis for the Las Vegas `MAX-3SAT` algorithm. The experiments aim to validate the theoretical guarantees regarding the fraction of satisfied clauses and the **expected runtime** of the algorithm.

4.1 Methodology

To empirically analyze the algorithm, we performed the following procedure:

1. **Multiple runs:** Each `MAX-3SAT` instance from a benchmark set of **100 formulas** was **solved 100 times** using the Las Vegas algorithm to collect robust statistics.
2. **Iteration count tracking:** For each run, we recorded the number of random assignments generated until an assignment satisfying at least 7/8 of the clauses was found.
3. **Satisfaction proportion tracking:** We also recorded the fraction of clauses satisfied by the assignment returned by the algorithm.

The collected data are represented using two visualizations:

— **Histogram of iterations:** Illustrates the distribution of the number of iterations required to reach the 7/8 threshold, with a KDE curve overlay for smoothness.

— **Distribution of satisfied-clause proportions:** Shows the fraction of clauses satisfied across all runs, again with a KDE overlay to highlight the overall distribution.

4.2 Results

The results of our experiments are presented in Figures 1 and 2.

In the **iteration analysis**, the histogram shows how many random assignments were required to reach the 7/8 satisfaction threshold across all instances and runs. Most runs converge quickly, with only a few requiring more iterations, in line with the expected runtime from Section 3.

The CDF provides a complementary view, illustrating the probability that an assignment meeting the threshold is found within a given number of iterations. Both visualizations confirm that the expected

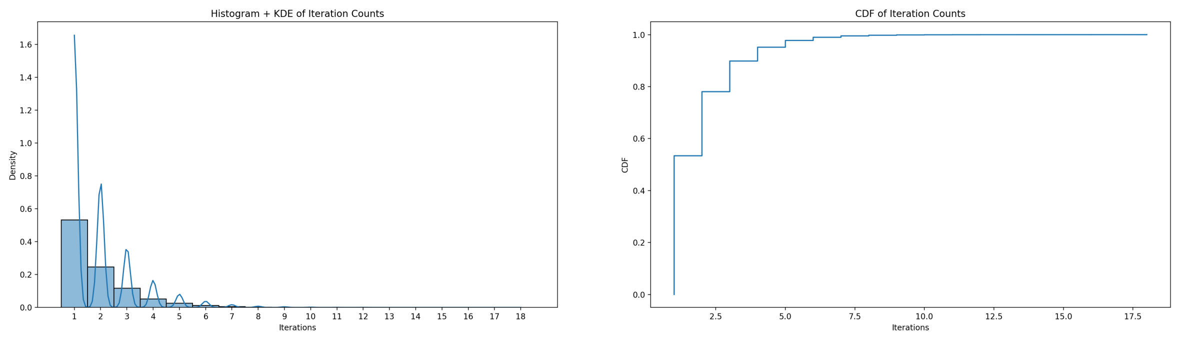


Figure 1: Empirical results for iteration counts: histogram (left) and cumulative distribution function (CDF, right).

number of iterations is small, as predicted by the geometric distribution argument and the concentration results using McDiarmid's inequality [2].

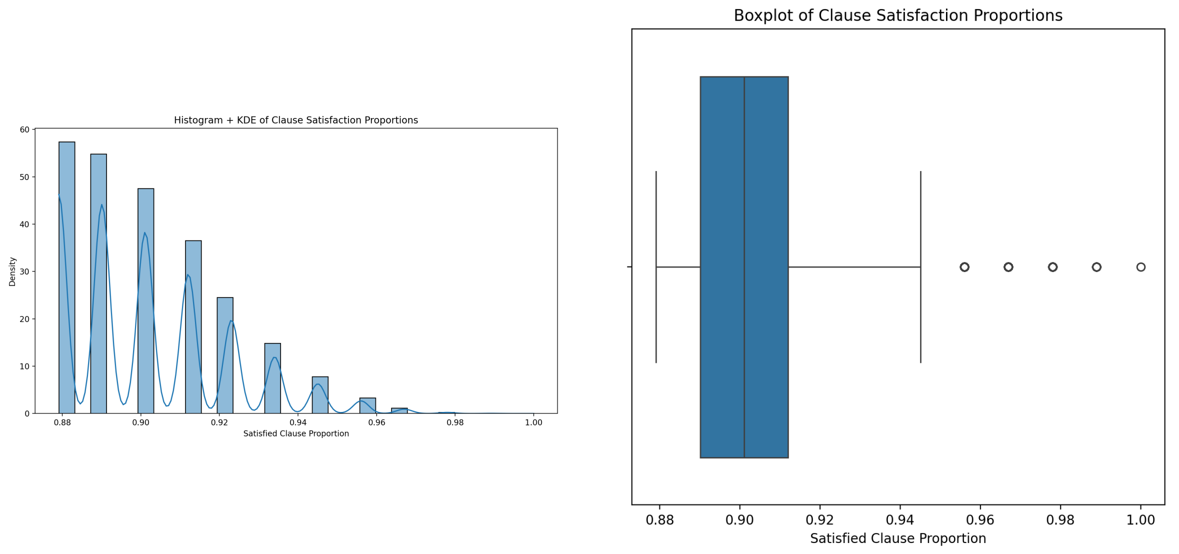


Figure 2: Distribution of satisfied-clause proportions: histogram (left) and boxplot (right).

In the **satisfaction proportion analysis**, the histogram and boxplot summarize the fraction of clauses satisfied by the returned assignments. The distributions are tightly concentrated around the theoretical 7/8 bound, validating the expected fraction of satisfied clauses discussed in Section 3. These results demonstrate that the Las Vegas algorithm reliably produces near-optimal assignments and that the variance across runs is minimal.

Overall, the experiments empirically confirm the theoretical predictions: the algorithm consistently achieves at least 7/8 clause satisfaction, and the number of iterations required is generally small, supporting the expected $O(1)$ runtime behavior per Section 3.

5 Use of LLMs

In this brief section, we discuss the use of large language models (LLMs) in the context of this project. No LLM was used for the implementation or for understanding the project itself. However, this report was reviewed for syntax and grammar by DeepL as well as a GPT model. It should be noted that none of the information contained in this report was generated by anyone other than the author.

6 Conclusion

This project presented the design, implementation, and evaluation of a Las Vegas algorithm for the Maximum 3-SAT problem. Both theoretical analysis and empirical results confirm the key properties of the algorithm: it consistently produces assignments satisfying at least 7/8 of clauses, and the expected number of iterations required is small. Concentration results using McDiarmid's inequality further justify that deviations from this bound are rare, supporting the practical efficiency of the method.

The experiments demonstrate that the algorithm reliably achieves near-optimal satisfaction across diverse instances, with low variance in both iteration counts and fraction of satisfied clauses. This confirms the theoretical guarantees regarding both correctness and expected runtime.

Overall, the Las Vegas approach illustrates the effectiveness of randomized algorithms for combinatorial optimization problems: it delivers strong probabilistic guarantees, simple implementation, and predictable performance, making it a practical choice for large MAX-3SAT instances.

Appendix A: Las Vegas Algorithm Implementation

Python implementation of the Las Vegas algorithm discussed in this report.

```python
import random
from . import Literal, Clause, Formula, Max3SAT

def random_assignment(literals):
    assignment = dict()
    for l in literals:
        value = random.choice([True, False])
        assignment[l.id] = value
    return assignment

def run(max3sat):
    it: int = 0
    literals = list(max3sat.all_literals()) # yields all literals of instance
    while True:
        assignment = random_assignment(literals)
        satisfied_clauses = max3sat.count_satisfied_clauses(assignment)
        it += 1
        # uses the karloff-zwick bound of 7/8
        if satisfied_clauses >= (7 / 8) * max3sat.formula.size():
            return satisfied_clauses, assignment, it
```

Appendix B: Max3SAT Implementation

Python implementation of the Literal, Clause, Formula and finally Max3SAT instance.

```python
class Literal:
    def __init__(self, id, negative):
        self.id = id
        self.negative = negative
    def evaluate(self, assignment):
        if self.id not in assignment:
            raise ValueError()
        value = assignment[self.id]
        return not value if self.negative else value
    def __hash__(self):
        return hash((self.id, self.negative))
```

```python
class Clause:
    def __init__(self, literals):
        self.literals = literals
    def size(self):
        return len(self.literals)
    def is_satisfied(self, assignment):
        for l in self.literals:
            if l.evaluate(assignment): return True
        if all(not l.evaluate(assignment) for l in self.literals): return
           False
```

```python
class Formula:
    def __init__(self, clauses):
        self.clauses = clauses
    def size(self):
        return len(self.clauses)
    def is_satisfied(self, assignment):
        for c in self.clauses:
            if not c.is_satisfied(assignment): return False
        if all(c.is_satisfied(assignment) for c in self.clauses): return True
```

```
class Max3SAT:
    def __init__(self, formula):
        assert all(clause.size() <= 3 for clause in formula.clauses)
        self.formula = formula
    def all_literals(self):
        literals_set = set()
        for clause in self.formula.clauses:
            for literal in clause.literals:
                literals_set.add(literal)
        return literals_set
    def count_satisfied_clauses(self, assignment):
        return sum(1 for clause in self.formula.clauses if clause.is_satisfied
           (assignment))
```

References

[1] H. Karloff and U. Zwick. *A 7/8-Approximation Algorithm for Max 3SAT.* SIAM Journal on Computing, vol. 26, no. 6, pp. 1536–1547, 1997.

[2] C. McDiarmid. *On the Method of Bounded Differences.* Surveys in Combinatorics, London Mathematical Society Lecture Note Series, vol. 141, pp. 148–188, 1989.