Alpha-SAT

Lixuan Lang, Yuda Song, Junchi Zhou, Bingjie Zhou

Advised by Prof. Sicun Gao

What Is the SAT Problem?

The Boolean Satisfiability problem (abbreviated as SAT) is about determining whether or not there are variable assignments that *satisfy* an arbitrary Boolean logic formula.

Satisfiable: (a ∨ b) ∧ (¬ a ∨ c)

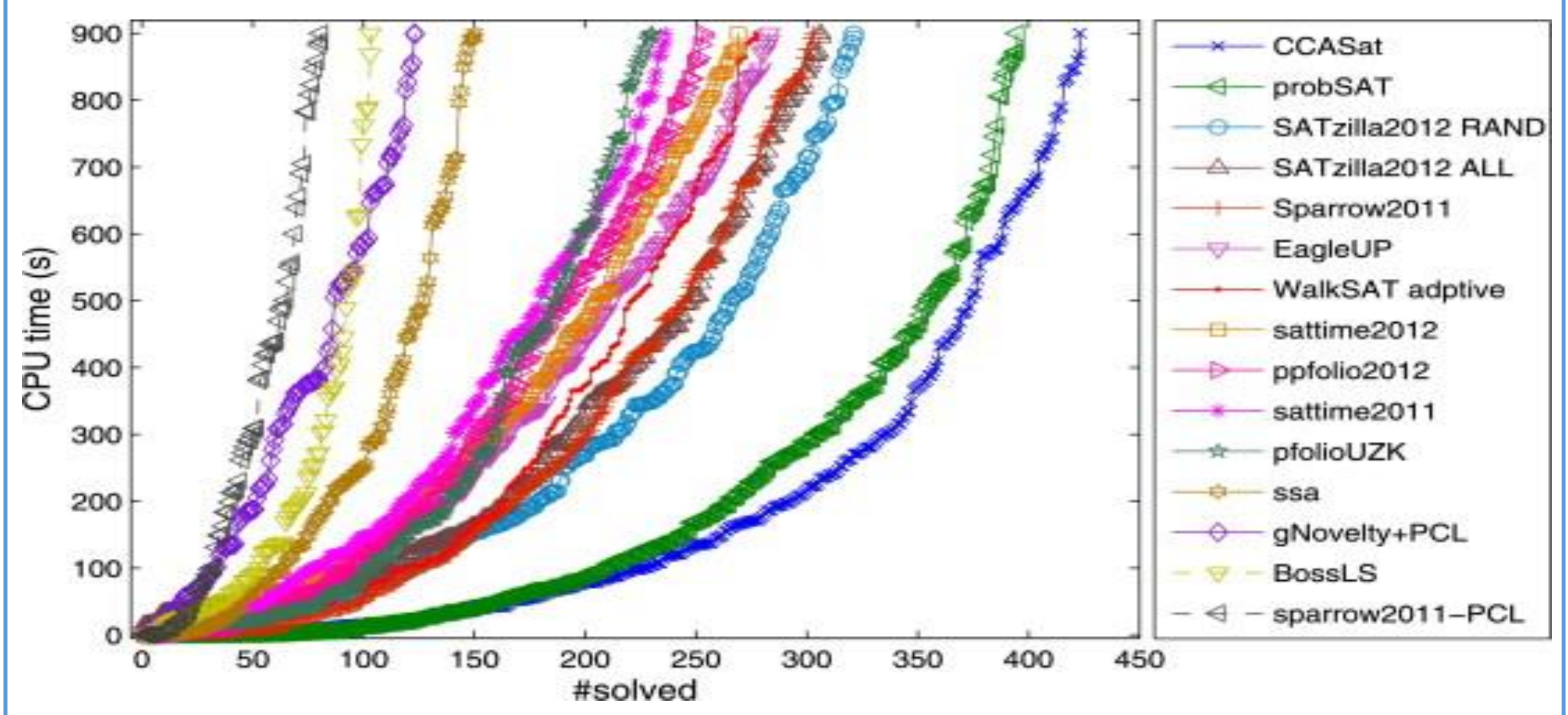
(A satisfying assignment to the formula would be a = TRUE, b = FALSE, and c = TRUE)

Unsatisfiable: (a ∨ a) ∧ (¬ a ∨ ¬ a)

(this formula will always evaluate to FALSE for all possible assignments of a)

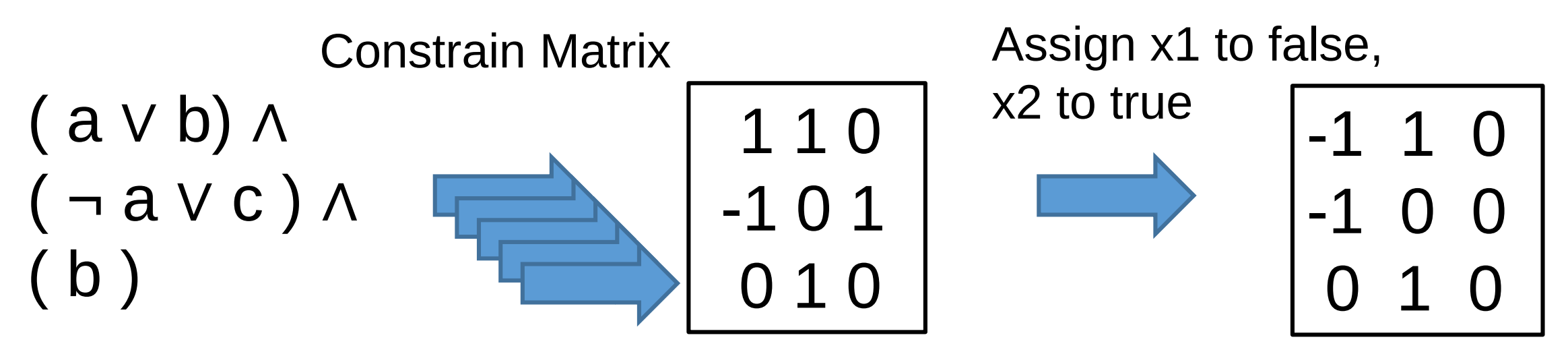
Motivation

In practice, SAT lies at the core of numerous areas in computer science like artificial intelligence, computer security, etc. However, it is of exponential time complexity. Recent trends suggest that over time, SAT solvers have become more time-efficient, which can be seen in the graph below.

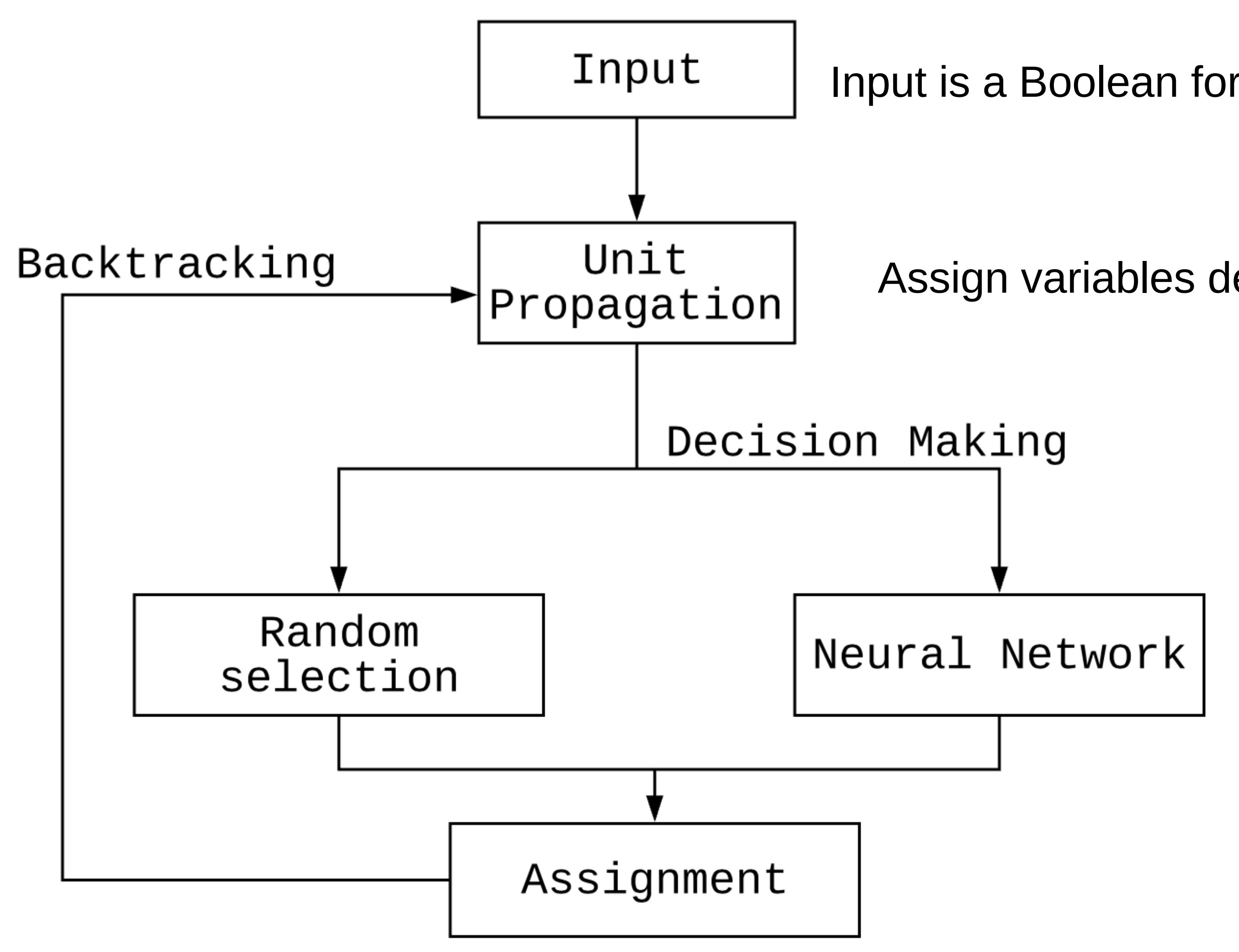


Data Processing

- Since we cannot feed the neural network with a Boolean formula, we convert the Boolean formula into a matrix.
- The shape of the matrix is 2 * number of variables * number of the clauses.
- The first layer is the representation of the constrain matrix. For each clause, we let each non-negated literal to be 1, negated literal to –1, and 0 for non-existing variable.
- The second layer is a representation of the current assignment state. For each clause, we let variable that is assigned to true be 1, -1 for ones assigned false, and 0 for non-existing variable.



SAT solving process



Input is a Boolean formula, such as (x1 ∨ x2) ∧ (¬ x1 ∨ x3) ∧ x3 ∧ …

Assign variables deterministically, such as singletons and negations.

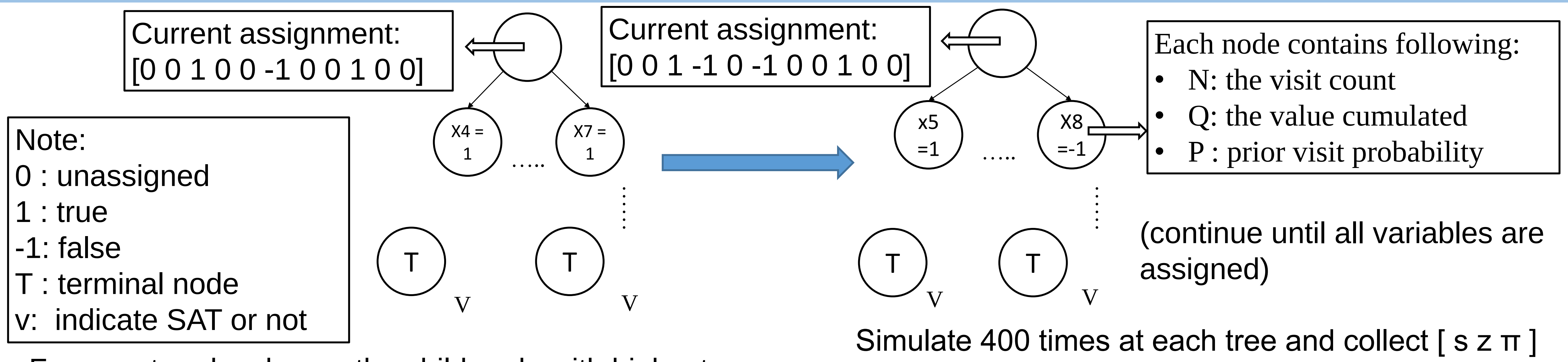
If there is no conflict:
- If all variables are assigned, return
- Else continue making decisions

If there is conflict:
- Backtrack to the last decision made, and change that decision

Traditional SAT solver makes decisions by random selections from unassigned variables or by some heuristics.

Our SAT solver makes decisions by neural networks:
- Input: a matrix of the current assignment of variables and the Boolean expression, as indicated in the data processing section.
- Output: a probability distribution that represents the likelihood of each possible assignment of variables to solve the Boolean formula.
- Greedily choose the variable and assignment with highest probability

Training



Note:
0 : unassigned
1 : true
-1: false
T : terminal node
v: indicate SAT or not

Each node contains following:
- N: the visit count
- Q: the value cumulated
- P : prior visit probability

(continue until all variables are assigned)

From root node, choose the child node with highest

$$Q + c \times P \times \sqrt{\frac{N_{total}}{N}}$$

If meet an unknown node, use existed NN to get the current node value and possible next moves until terminal node

Simulate 400 times at each tree and collect [s z π] each time for training
- s is the current state
- z is the sum of terminal values v
- π is the distribution of children's nodes of root

Sample from π to assign and continue simulation

Result

	# Backtrack	Time(second)
Traditional SAT solver	0.08	0.00168
Alpha-SAT	0	31.8005

We run Alpha-SAT and traditional SAT solver on 1000 random generated CNF each with 10 variables and 10 clauses. As our expectation, the number backtrack of backtrack on Alpha-SAT is 0 which means our model can make "smarter" decision to avoid backtrack.

Conclusion

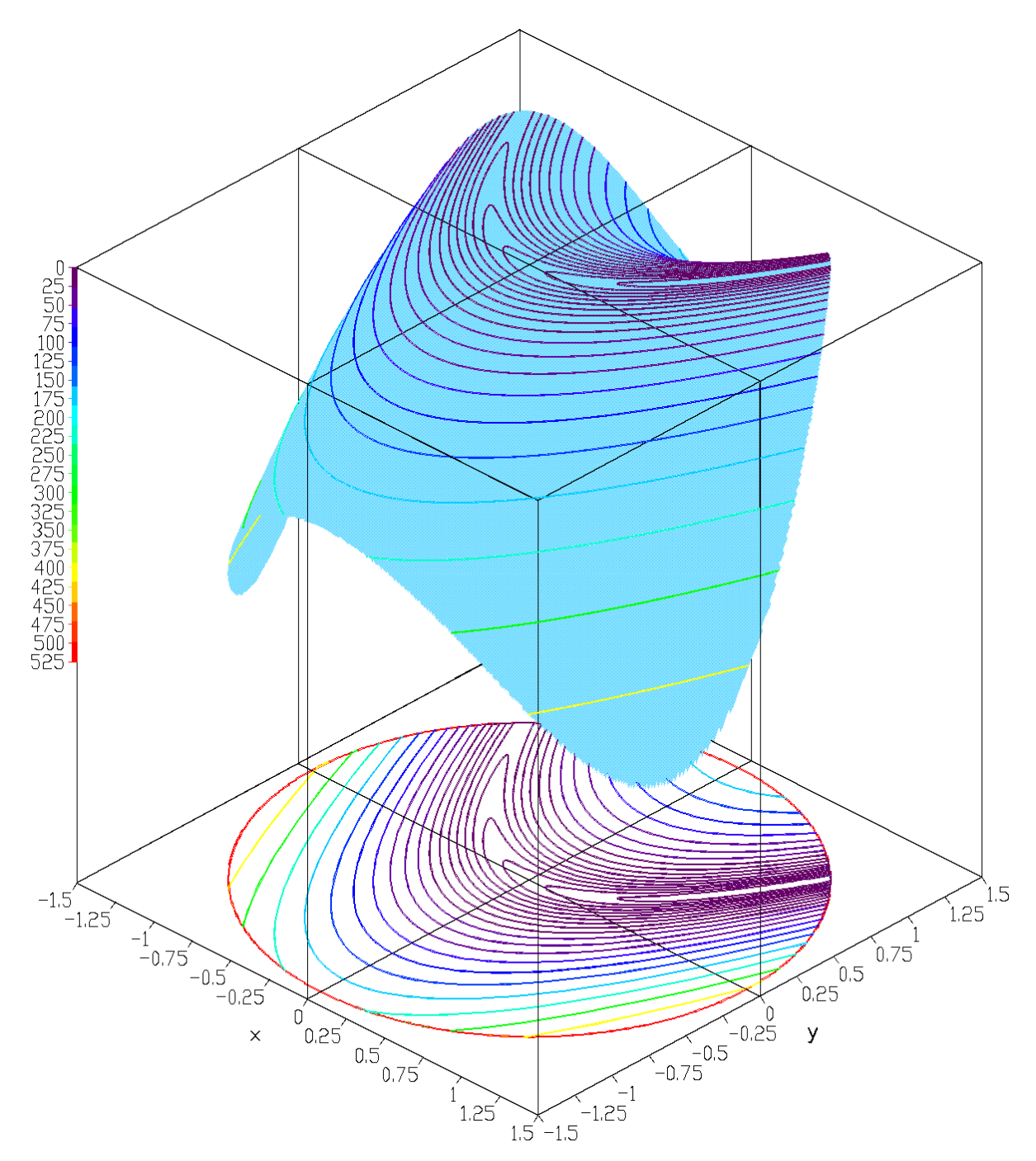
In our approach, we use CNN (Convolutional Neural Network) as our neural network. However, CNN needs a fixed-sized input and it prefers inputs with same size length and width. Thus we restrict our training on 10 * 10 SAT problems, that is, each SAT problem has 10 variables and 10 clauses. It turns out that with the help of the MCTS, our model can very easily find the satisfiable assignment with very few backtracks.

Future Work

Since there is restriction for our approach in SAT, we plan to use the same approach in non-linear optimization.

- Example[3]

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2,$$
$$\text{Subject to} : x^2 + y^2 \le 2$$



- Suppose we want to find the minimum point in this graph. We first divide the domain in several parts and then need to make decision which part to explore, which is similar to the make decision part in our SAT solver. We plan to train neural networks to make decision.

Reference

[1] Bünz B, Lamm M. Graph Neural Networks and Boolean Satisfiability. arXiv preprint arXiv:1702.03592. 2017 Feb 12.
[2] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... & Dieleman, S. (2016). Mastering the game of Go with deep neural networks and tree search. *nature*, *529*(7587), 484-489.
[3] "Test functions for optimization." *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, Inc. 22 July 2004. Web. 10 Aug. 2004, en.wikipedia.org/wiki/Test_functions_for_optimization

Acknowledgement

We want to thank Professor Christine Alvarado and Aditi Mavalankar. We also want to thank Professor Gao for his advising and help in the project.
This material is based upon work supported by the National Science Foundation under Grant No. CNS-1339335.