Memcomputing-inspired 3SAT Solver
Special Subject: Faster 3-SAT expression evaluation

Ioannis Kallergis

07/5/2025

1. Notation and Indices

- t: Discrete time step.
- M: Total number of clauses.
- N: Total number of variables.
- $m \in \{1, ..., M\}$: Clause index.
- $k \in \{1, ..., N\}$: Variable index.
- $j \in \{0, 1, 2\}$: Literal index within a clause.
- $v_{m,j}$: Variable index of the j-th literal in clause m.
- $q_{m,j} \in \{-1, 1\}$: Sign of the j-th literal in clause m (1 for positive, -1 for negated).
- $v_k^{(t)} \in [-1, 1]$: State of variable k at time t.
- $x_{S,m}^{(t)} \in [0, 1]$: Short-term memory for clause m at time t.
- $x_{L,m}^{(t)} \in [1, 10^4 M]$: Long-term memory for clause m at time t.
- $E_m^{(t)}$: Energy of clause m at time t.
- $\Delta_k^{(t)}$: Effectiveness of variable k at time t.
- $I_k^{(t)}$: Influence of variable k at time t.

- $\tau^{(t)}$: Adaptive time step at time t.
- $\text{sgn}(x)$: Sign function.

2. Clause Properties at Time t

literal's measure dissatisfaction

$$l_{m,j}^{(t)} := 0.5 \cdot (1 - q_{m,j} \cdot v_{\nu_{m,j}}^{(t)})$$

$$l_{\min,m}^{(t)} := \min_{j' \in \{0,1,2\}} (l_{m,j'}^{(t)}), \text{ and } \quad j_m^* := \arg\min_{j'} (l_{m,j'}^{(t)})$$

the clause's measure of dissatisfaction

$$E_m^{(t)} := l_{\min,m}^{(t)}$$

how 'aligned' is the clause (how much the variables align with their respective booleans). In this sense, a perfectly satisfied clause $\exists q_{m,j} \cdot v_k = 1$ but if the other $q \cdot v < 0$ then the clause isn't very 'aligned'.

$$S_m^{(t)} := \sum_{j=0}^{2} q_{m,j} \cdot v_{\nu_{m,j}}^{(t)}$$

A possible solution to the CNF found by the system is $|v|_k = (1, 0, 0)$, ie. the literal that makes the clause stisfied is ± 1 and the others are 0 (don't cares). Then $S_m = 1 + 0 + 0 = 1$. We could reinforce this property into the system. So let's define the following term which grows bigger the more the $S_m <\!\!\longleftarrow\!\!< 1$ to measure the progress. Call it 'disalignment'

$$H_m^{(t)} := \max(0, 1 - S_m^{(t)})$$

3. Variable Influence at Time t

the total disalignment the variable is locally (via the clauses) involved in

$$I'_k{}^{(t)} = \sum_{\substack{m \\ k \in \{\nu_{m,0}, \nu_{m,1}, \nu_{m,2}\}}} \left(H_m^{(t)}\right)^2$$

$$I_k^{(t)} = \ln(1 + I_k'^{(t)} + \epsilon)$$

4. Auxiliary Variable Dynamics

short term and long term dissatisfaction (memories as non-local in time). $\alpha, \beta, \gamma, \delta$ in the original paper [1] where carefully tuned constants, now they are dynamic. $\sin^3(\cdot)$ (non-linearity) is also a new feature.

$$\dot{x}_{S,m}^{(t)} = \beta^{(t)}(x_{S,m}^{(t)} + \epsilon) \cdot \sin^3\left(E_m - \gamma^{(t)}\right)$$

$$\dot{x}_{L,m}^{(t)} = \alpha^{(t)} \cdot \left(E_m - \delta^{(t)}\right)$$

5. Variable Dynamics

5.1 DMM Gradient

gradient term and reinforcement (gradient) term are coupled with the 'memory' variables. Note: In the authors' paper the reinforcement term instead of G was scaled by E_m, but that didn't work as well in this implementation.

$$g_m^{(t)} := x_{L,m}^{(t)} x_{S,m}^{(t)}, \quad r_m^{(t)} := (1 + \zeta^{(t)} x_{L,m}^{(t)})(1 - x_{S,m}^{(t)})$$

gradient term is derived to act as a derivative of E_m with respect to v_k

$$G_{m,j}^{(t)} = q_{m,j} \cdot 0.5 \cdot \min_{p \neq j} l_{m,p}^{(t)}$$

gradient term and reinforcement term (might be conflicting with the desire to reduce 'disalignment')

$$T_{1,m,j}^{(t)} = g_m^{(t)} \cdot G_{m,j}^{(t)}, \quad T_{2,m,j}^{(t)} = \begin{cases} r_m^{(t)} \cdot G_{m,j}^{(t)}, & j = j_m^* \\ 0, & \text{otherwise} \end{cases}$$

regularization term

$$T_{3,m,j}^{(t)} = x_{L,m}^{(t)} \cdot q_{m,j} \cdot (1 - |v_{\nu_{m,j}}^{(t)}|)$$

the derivative of the voltages at timestep t equals the sum of those terms of the clauses in which v_k appears

$$\dot{v}^{(t)}_{v_{m,j},\mathrm{DMM}} += T^{(t)}_{1,m,j} + T^{(t)}_{2,m,j} + T^{(t)}_{3,m,j}$$

5.2 GloSat Gradient (glo as global)

(Figure1: with-GloSat

Figure2: without-GloSat)

constants were picked at *random* and therefore its contribution the the total gradient turned of to be much much smaller (in terms of 'signal' magnitude) , nevertheless it seems to help the system find solutions faster. [Figure1 , Figure2] .

disalignment of the clause , clipped to [-1,1]

$$G^{(t)}_{cl,m} = \mathrm{clip}(\ 1 - S^{(t)}_m,\ [-1, 1]\ )$$

'attention' window, is used to focus dynamics on almost-alignment

$$W^{(t)}_m = \max(0, 1 - |S^{(t)}_m - 0.8| \cdot 5)$$

(scaled) effectiveness of the variable * (times) its total disalignment

$$A^{(t)}_{v_{m,j}} = \tanh(\Delta^{(t)}_{v_{m,j}}) \cdot \min(5, \sqrt{1 + I^{(t)}_{v_{m,j}}})$$

$$T^{(t)}_{B,m,j} = 2 \cdot \eta_{DS} \cdot q_{m,j} \cdot A^{(t)}_{v_{m,j}} \cdot G^{(t)}_{cl,m} \cdot \left(x^{(t)}_{L,m}\right)^{\frac{1}{3}x^{(t)}_{S,m}}$$

entropy term , forces the system to explore and not get stuck at exactly $|v_k| = 1$

$$T^{(t)}_{E,v_{m,j}} = \begin{cases} w_E \cdot (-\mathrm{sgn}(v^{(t)}_{v_{m,j}}))(|v^{(t)}_{v_{m,j}}| - 0.9) \cdot 10, & |v^{(t)}_{v_{m,j}}| > 0.9 \\ 0, & \text{otherwise} \end{cases}$$

(Figure1: is without the q's, and its role is just for symmetry breaking) Also tried the following (without noticable improvement): alignment reinforcer term, we want the system to end up in $|v| = (1, 0, 0)$ solution

$$\mathrm{sim}^{(t)}_{v_{m,j},m} = \sum_{p \neq j : \mathrm{sgn}(q \cdot v^{(t)}_{v_{m,j}}) = \mathrm{sgn}(q \cdot v^{(t)}_{v_{m,p}})} |v^{(t)}_{v_{m,j}}| \cdot |v^{(t)}_{v_{m,p}}|$$

$$T^{(t)}_{D,v_{m,j},m} = -w_D \cdot \mathrm{sgn}(v^{(t)}_{v_{m,j}}) \cdot \mathrm{sim}^{(t)}_{v_{m,j},m}$$

only almost-aligned clauses contribute to v_k derivatives (because of the window)

$$T^{(t)}_{tot,m,j} = (T^{(t)}_{B,m,j} + T^{(t)}_{E,v_{m,j}} + T^{(t)}_{D,v_{m,j},m}) \cdot W^{(t)}_m$$

the derivative of the voltages at timestep t equals the sum of those terms of the clauses in which v_k appears

$$\dot{v}^{(t)}_{v_{m,j},DS} += T^{(t)}_{tot,m,j}$$

5.3 Final Derivative and Update

$$\dot{v}^{(t)}_k = \dot{v}^{(t)}_{k,\mathrm{DMM}} + \ 0.35\ \dot{v}^{(t)}_{k,\mathrm{DS}}$$

simple forward euler integration is chosen, as the authors [1, 3] argue it suffices because of memcomputing ability to handle noise (explained with topological theories) and that including numerical noise. As long as the method doing introduce numerical $v^{(t+1)} = v^{(t)}$ (for example inplicit integration methods could do that)

$$P^{(t)}_k = \mu^{(t)}(v^{(t)}_k - v^{(t-1)}_k), \quad \mathrm{v}^{(t+1)}_k = \mathrm{clip}(v^{(t)}_k + \dot{v}^{(t)}_k \cdot \tau^{(t)} + P^{(t)}_k, [-1, 1])$$

6. Adaptive Time Step and Parameters

Parameters where optimized when designing the without-GloSat system (along with the DMM parameters), with standard bayesian methods.

$$\bar{E}^{(t)} = \frac{1}{M} \sum_m E^{(t)}_m$$

$$s^{(t)}_{\mathrm{avg}} = 1 + \mathrm{clip}(\alpha_{\mathrm{avg}}(\bar{E}^{(t)} - \beta_{\mathrm{avg}}), [0, 2])$$

$$\alpha^{(t)} = \alpha_0 \cdot \sqrt{s^{(t)}_{\mathrm{avg}}} \cdot 1.1, \quad \beta^{(t)} = \beta_0 \cdot s^{(t)}_{\mathrm{avg}}$$

$$\gamma^{(t)} = \gamma_0(1 - \alpha_\gamma \cdot f_{\text{step}}^{(t)} \cdot 1.05), \quad \zeta^{(t)} = \zeta_0(1 + f_{\text{step}}^{(t)} \cdot 1.1)$$

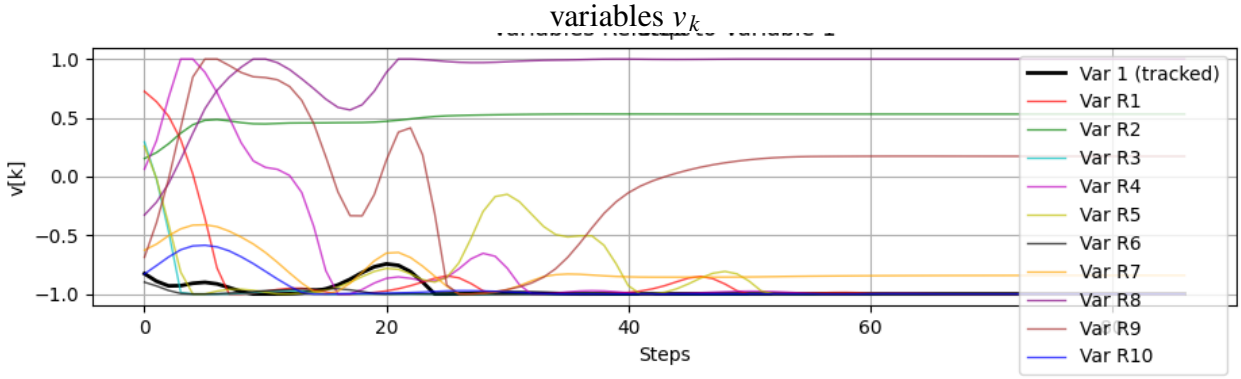
momentum switches on and off depending on progress

$$\mu^{(t)} = \begin{cases} \min(\mu_{\max}, \mu_{\min} + \mu_{\text{growth}} f_{\text{step}}^{(t)} \cdot 1.2), & \text{if momentum on} \\ 0, & \text{otherwise} \end{cases}$$

7. Initial Conditions

- $v_k^{(0)} \sim \mathcal{U}(-1, 1), \quad v_k^{(-1)} = v_k^{(0)}$
- $x_{S,m}^{(0)} = 0.5, \quad x_{L,m}^{(0)} = 1.0$
- $\Delta_k^{(0)} = 0, \quad E_m^{(-1)} = E_m^{(0)}$

The parameters of the first model were optimized with Bayesian methods, The parameters of the model with GloSat gradients were not optimized and were picked ramdomly and kept.



Variables vary continuously and can only rest at a solution.

1. **Gradient Magnitude Ranking**:
   - : $\mathbf{DMM_{Lagrangian}}$, $\mathbf{DMM_G}$, $\mathbf{DMM_R}$ are about equal.
   - GloSat's main gradient ($\mathbf{GloSat_{Main}}$) is generally the smallest but maybe still meaningful.
   - notice around step 110 : $\mathbf{DMM_{Lagrangian}} \to 0$ and GloSat terms get bigger.

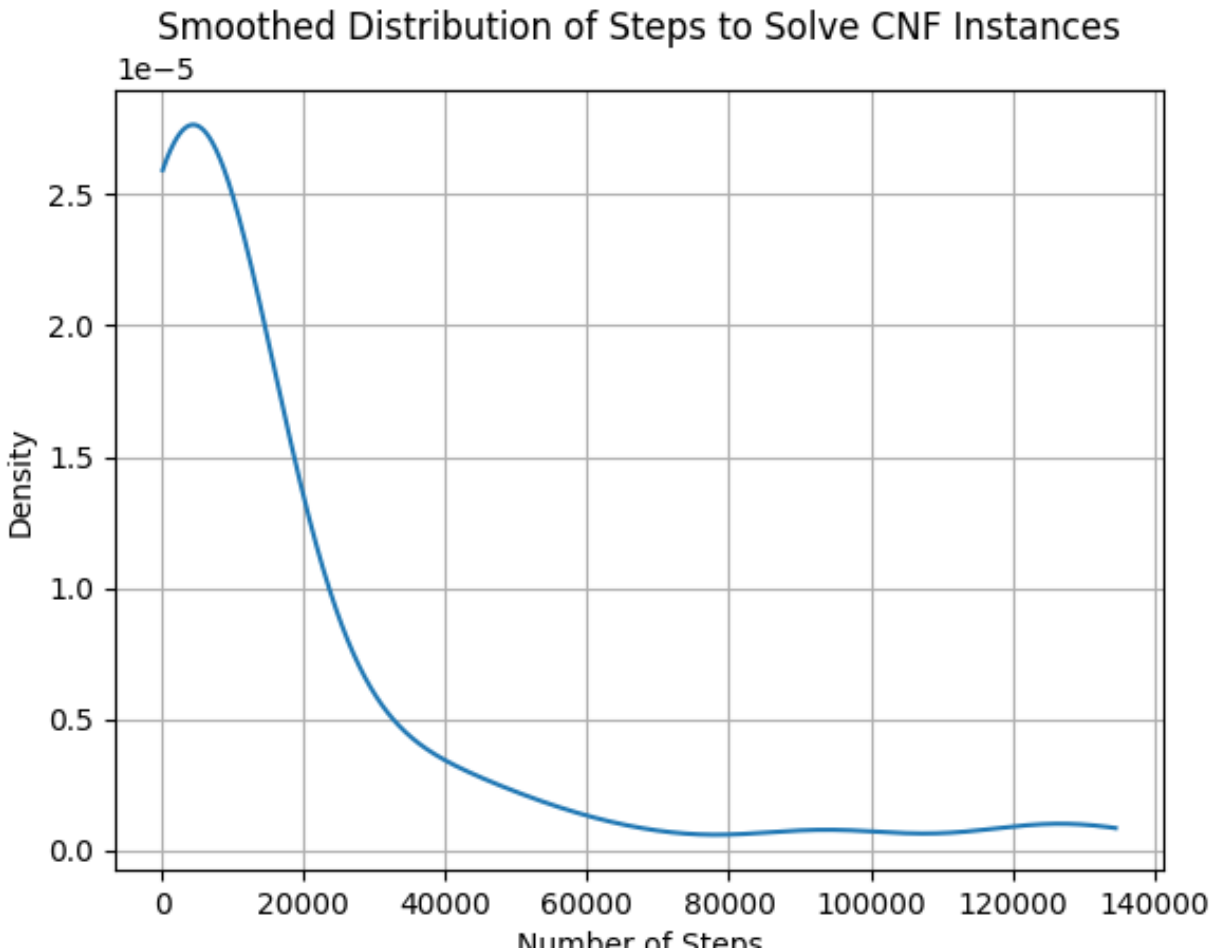


Figure 1: (*) without q in diffaset reinforcement term (so just symmetry breaking): distribution of timesteps it took to solve the CNFs of UF250-1065

, only the 93/100 (solved ones) count .

Figure 2: without GloSat gradients. Notice it takes more steps (and sometimes CNF33,CNF94 gets stuck in a loop indefinitely), but it does have 94/100 solved CNFs

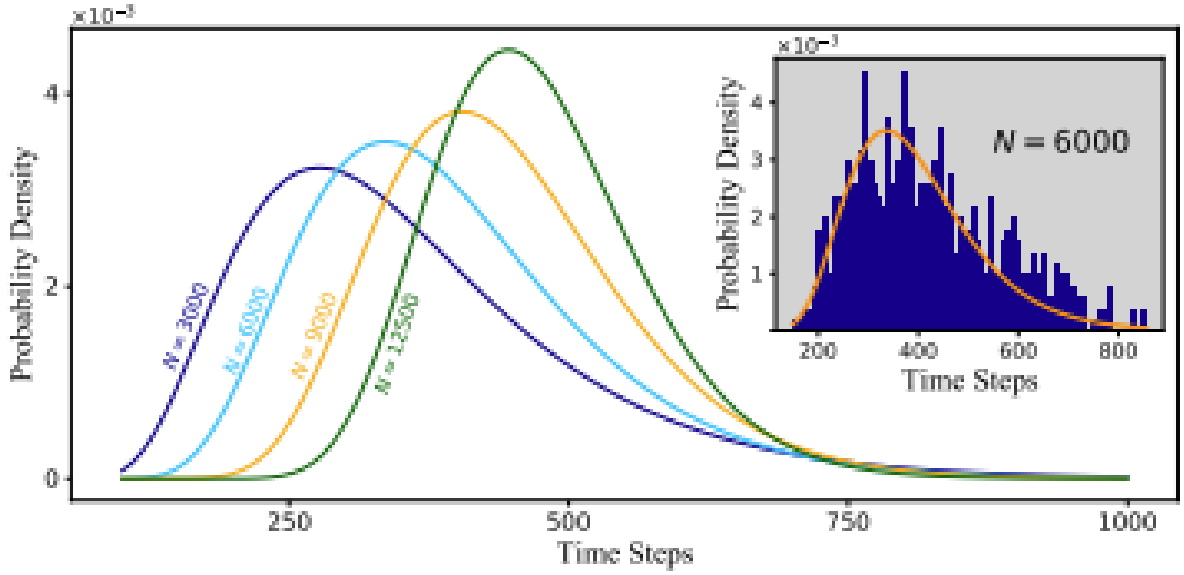


FIG. 2. Various fits (inverse Gaussian distributions) of the TTS (in number of integration steps) of DMMs out of 1,000 3-SAT instances (taken from [9]) of varying number of variables, N, at a fixed clause-to-variable ratio of 7. The noise strength is $\Gamma = 0.12$ (in inverse time units). Inset: the histogram of TTS (in number of integration steps) of the DMM with $N = 6{,}000$ variables and 42,000 clauses. The fit is an inverse Gaussian distribution, Eq. (5), with $\lambda = 3720 \pm 420$ and $\mu = 390 \pm 10$.

Figure 3: original system follows an inverse gaussian distribution source: [3]

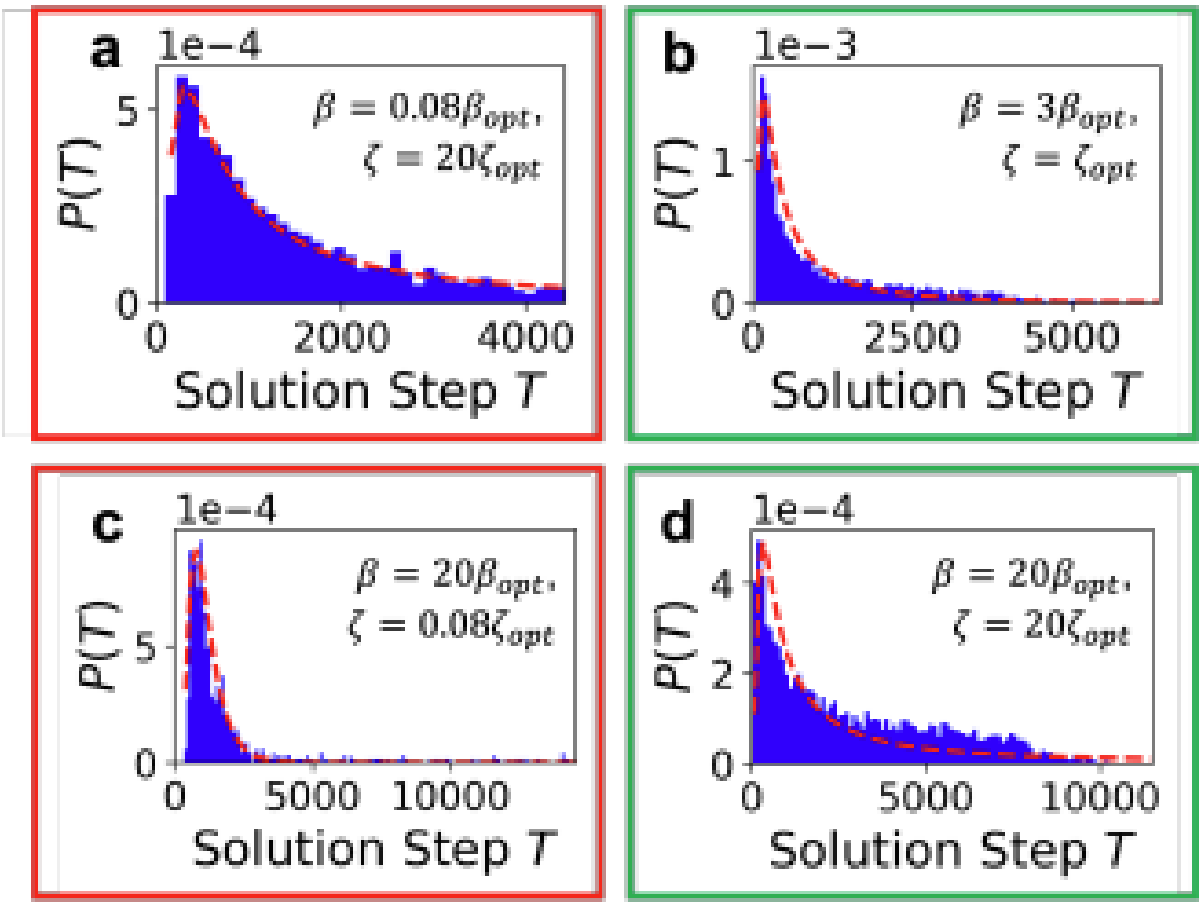


FIG. 4. A comparison of the time-to-solution (TTS) distributions $P(T)$ as a function of number of solution steps T for different $\{\beta, \zeta\}$. Plots (a), (b), (c), and (d) correspond to plots (b), (c), (d), and (e) in Fig. 2. The distributions in all plots are fitted to inverse Gaussians. 100 batches are simulated, and each batch includes 100 instances each with $N = 100$. According to the results of Fig. 1(a), plots (a) and (c) correspond to parameters that lead to inefficient solution search, while the parameters in (b) and (d) lead to an efficient search. For (a), the median solution step is $T_{median} \approx 10^7$. For (c), a median of instances could not be solved within $T = 10^8$ steps.

Figure 4: source: [2]

2. **Correlation Patterns**:

   - $\mathbf{DMM_{reinforcement\_term}} \leftrightarrow \mathbf{DMM_{gradient\_term}}$: High positive correlation (0.6-0.9), showing they work together.
   - $\mathbf{DMM_{gradient\_term}} \leftrightarrow \mathbf{GloSat}$: Generally small negative correlation (-0.01 to -0.09).
   - $\mathbf{DMM_{reinforcement\_term}} \leftrightarrow \mathbf{GloSat}$: Also small negative correlation (-0.01 to -0.07).

Quantitatively:

tablePerformance Metrics Over Training Steps

Step	$\mathbf{C_{mean}}$	$\mathbf{DMM_L}$	$\mathbf{DMM_R}$	$\mathbf{DMM_G}$	$\mathbf{GloSat_{Main}}$	$\mathbf{GloSat_{Ent}}$	$\mathbf{GloSat_{Div}}$	**Corr(Main, G/R)**
0	0.2394	6.77	3.38	2.63	0.0000	0.1068	0.0307	0.0000 / 0.0000
10	0.0269	5.25	5.63	5.02	0.0734	0.1431	0.0848	-0.0909 / 0.0676
20	0.0192	3.29	6.52	5.48	0.0603	0.0918	0.0770	-0.1973 / 0.0654
30	0.0173	5.25	6.94	7.07	0.0433	0.0882	0.0872	-0.1694 / 0.0157
40	0.0117	13.39	8.02	7.01	0.0851	0.0987	0.1007	0.0824 / 0.0832
50	0.0033	0.96	8.14	3.60	0.0556	0.0485	0.0586	-0.0229 / -0.0104
60	0.0045	6.51	8.14	4.73	0.0218	0.0415	0.0347	-0.0408 / -0.0332
70	0.0020	0.94	8.16	2.92	0.0231	0.0428	0.0378	0.0117 / 0.0343
80	0.0010	0.45	9.28	2.08	0.0438	0.0894	0.0716	-0.0743 / 0.0067
90	0.0019	0.04	8.15	2.64	0.0159	0.0441	0.0176	0.0000 / 0.0136
100	0.0030	10.74	8.73	7.37	0.0163	0.0469	0.0176	0.0660 / 0.0096
110	0.0018	6.84	8.27	4.91	0.0237	0.0524	0.0341	-0.0418 / 0.0205
120	0.0024	1.92	8.41	3.16	0.0315	0.0576	0.0438	-0.0124 / 0.0083
130	0.0017	0.87	8.55	2.43	0.0289	0.0603	0.0382	-0.0269 / 0.0127
140	0.0013	0.39	9.10	2.01	0.0321	0.0627	0.0431	-0.0335 / 0.0019
150	0.0009	0.19	9.33	1.82	0.0350	0.0660	0.0450	-0.0216 / -0.0023
160	0.0008	0.12	9.65	1.61	0.0362	0.0681	0.0467	-0.0142 / 0.0038
170	0.0006	0.08	9.84	1.52	0.0379	0.0697	0.0483	-0.0197 / 0.0044
180	0.0005	0.05	10.01	1.45	0.0384	0.0703	0.0490	-0.0150 / 0.0011
190	0.0004	0.03	10.15	1.37	0.0392	0.0714	0.0502	-0.0114 / 0.0029
200	0.0003	0.02	10.28	1.29	0.0400	0.0725	0.0510	-0.0107 / 0.0007
210	0.0003	0.01	10.38	1.22	0.0406	0.0731	0.0516	-0.0092 / 0.0014
220	0.0002	0.01	10.47	1.15	0.0412	0.0738	0.0523	-0.0078 / 0.0021
230	0.0002	0.01	10.55	1.08	0.0417	0.0742	0.0527	-0.0069 / 0.0017
240	0.0002	0.01	10.62	1.02	0.0420	0.0745	0.0530	-0.0063 / 0.0015
250	0.0002	0.01	10.68	0.97	0.0423	0.0748	0.0533	-0.0058 / 0.0012

Step	C_{mean}	DMM_L	DMM_R	DMM_G	$GloSat_{Main}$	$GloSat_{Ent}$	$GloSat_{Div}$	Corr(Main, G/R)
260	0.0001	0.00	10.73	0.92	0.0426	0.0750	0.0536	-0.0053 / 0.0010
270	0.0001	0.00	10.78	0.88	0.0428	0.0752	0.0538	-0.0049 / 0.0009
280	0.0001	0.00	10.82	0.83	0.0430	0.0753	0.0540	-0.0046 / 0.0008
290	0.0001	0.00	10.86	0.79	0.0431	0.0754	0.0541	-0.0043 / 0.0007
300	0.0001	0.00	10.89	0.75	0.0432	0.0755	0.0542	-0.0041 / 0.0006
310	0.0001	0.00	10.92	0.72	0.0433	0.0756	0.0543	-0.0039 / 0.0006
320	0.0001	0.00	10.94	0.69	0.0434	0.0756	0.0544	-0.0037 / 0.0005
330	0.0001	0.00	10.97	0.66	0.0435	0.0757	0.0545	-0.0036 / 0.0005
340	0.0001	0.00	10.99	0.63	0.0435	0.0757	0.0545	-0.0034 / 0.0004

———————- Solution found ———————————————————————————————

Related a minimal system without GloSat terms , with E_m instead of the gradient and constant vanilla momentum, DMM-langrangian terms under tight time constraints achieved (on my machine):

Processed 100 files: - VERIFIED: 51

- FAILED: 49 (didn't progress in few thousand steps)

Average solve time for verified instances: 2.04s

Also, systems such as for example a ones with random purturbations (23 random bit flipping when system gets stuck) was implemented with relatively high success rate but not so great time-to-solution.

Also lstm-inspired dynamical systems were designed, with learning and prediction based mechanisms based on differential equation (initially after noticing $x_{S,m}$ acts as a gate similar grus and lstms). This approach was abandoned, as those systems often struggled escaping local minima, that could be because of bad design, anyways, this direction was deemed non-worth pursuing for technical reasons I don't recall.

Before using benchmarked CNF instances such as the UF250-1065 [Figure1 Figure2] as well as well as back_bone_90 cnfs (even more successfully solved, just not included in the paper)

1 Experiments with custom generated 'hard' (turned out to be easy) CNFs

An custom idea for generating hard (as we'll see those are easy, as they are solved very easily but the systems), satisfiable 3-SAT instances is presented, which involves planting a hidden solution and introducing controlled noise:

1. **Establish a Hidden Solution**: A random truth assignment is generated for all N variables, where each variable is assigned a value of -1 (representing FALSE) or 1 (representing TRUE). This assignment serves as the guaranteed solution to the generated 3-SAT instance.
2. **Clause Construction**: For each clause, three distinct variables are randomly selected from the pool of N variables.
3. **Initial Literal Assignment**: The signs of the three literals in each clause are chosen such that all literals evaluate to TRUE under the hidden solution. Specifically, for a variable v_i with value $s_i \in \{-1, 1\}$ in the hidden solution, the literal is set to v_i (positive) if $s_i = 1$, or $\neg v_i$ (negative) if $s_i = -1$. This ensures that each clause is triply satisfied initially.
4. **Introduce Controlled Noise**: To increase the complexity of the instance, each literal's sign is flipped with a fixed probability, referred to as the *noise level*. For example, a positive literal v_i may be changed to $\neg v_i$, or vice versa, introducing controlled randomness.
5. **Guarantee Satisfiability**: After determining which literals to flip, a check is performed to ensure satisfiability. If flipping all three literals in a clause would result in the clause being unsatisfied by the hidden solution (i.e., all literals evaluate to FALSE), the flip of one randomly selected literal in that clause is reverted. This ensures that at least one literal per clause remains TRUE under the hidden solution.
6. **Final Instance Assembly**: The selected variables and their final, post-noise signs are combined to form the complete 3-CNF instance.

The ratio of clauses to variables is typically set near the 3-SAT critical phase transition threshold, approximately 4.26, as instances in this region are empirically the most challenging for many SAT solvers [2, 3].

The ratio of clauses to variables is set near the "critical ratio" (approximately 4.262 for 3-SAT), which is empirically known to produce the most difficult instances for solvers.

Pseudocode for Instance Generation

The following pseudocode outlines the core logic for generating a 3-SAT instance:

```
0: function GenerateHardSATInstance(N, M, noise_level, seed)
0:     {N: number of variables, M: number of clauses}
0:     Initialize a random number generator with seed
    {1. Create a hidden solution where True=1, False=-1}
0:     solution ← random array of size N with values from {−1, 1}
0:     clauses ← empty list of M clauses
0:     for m = 1 . . . M do
0:         {2. Select three unique variables for the current clause}
0:         v1, v2, v3 ← Select three distinct variables from 1 . . . N
0:         current_clause.variables ← {v1, v2, v3}
    {3. Set initial signs to make all literals true under the solution}
0:         s1 ← solution[v1]
0:         s2 ← solution[v2]
0:         s3 ← solution[v3]
0:         initial_signs ← {s1, s2, s3}
    {4. Determine which literal signs to flip based on noise level}
0:         flip_mask ← empty boolean list of size 3
0:         for i = 1 . . . 3 do
0:             flip_mask[i] ← (random_float(0, 1) < noise_level)
0:         end for
    {5. Ensure the clause remains satisfied by the hidden solution}
0:         if flip_mask[1] and flip_mask[2] and flip_mask[3] then
0:             {If noise would make the clause unsatisfiable...}
0:             k ← randomly choose an index from {1, 2, 3}
0:             flip_mask[k] ← False {...cancel one of the flips.}
0:         end if
    {6. Apply the flips to get the final literal signs}
0:         final_signs ← initial_signs
0:         for i = 1 . . . 3 do
0:             if flip_mask[i] then
0:                 final_signs[i] ← −final_signs[i]
0:             end if
0:         end for
0:         current_clause.signs ← final_signs
0:         Add current_clause to clauses
0:     end for
```

0: **return** *clauses*

0: **end function**=0

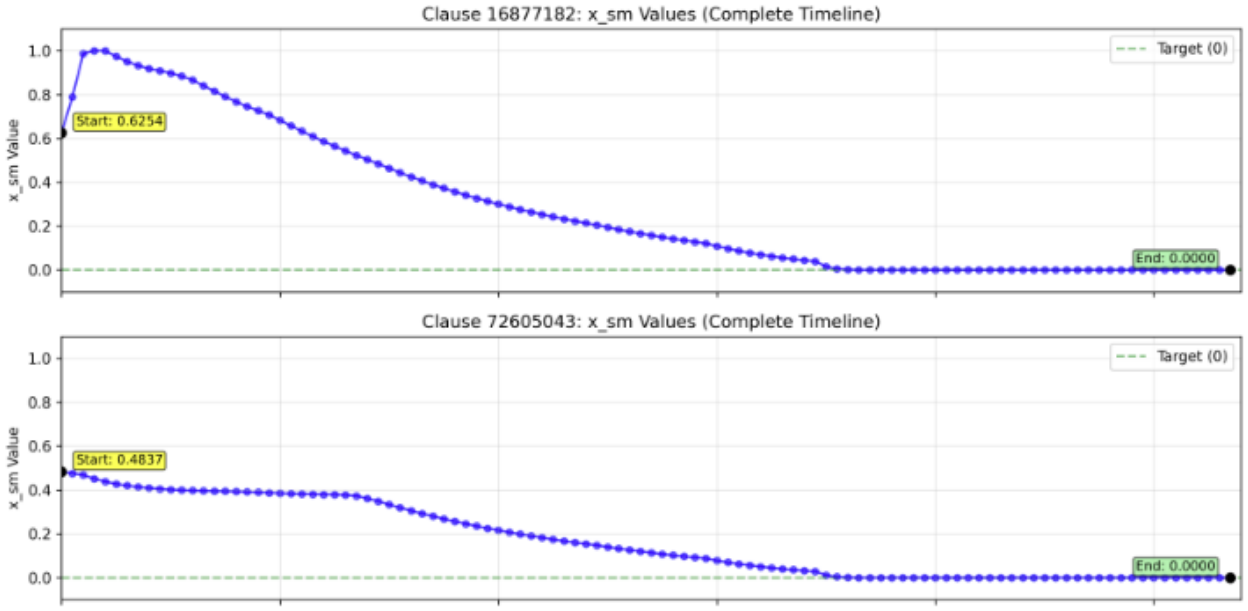


Figure 5: Some $s_{S,m}$ of (an easier large) cnf instance

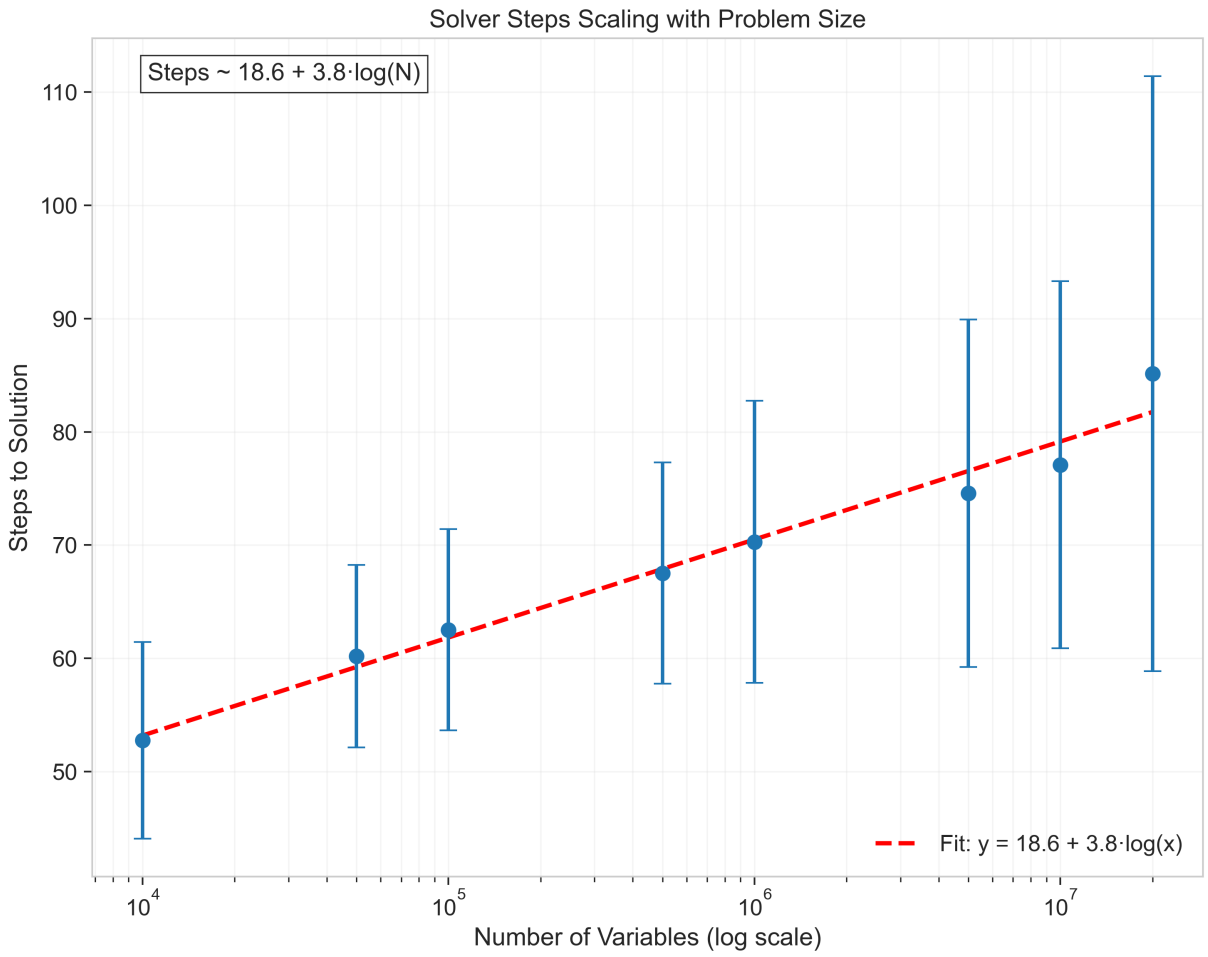


Figure 6: that was custom generated instances (it turned out they were not hard instances)

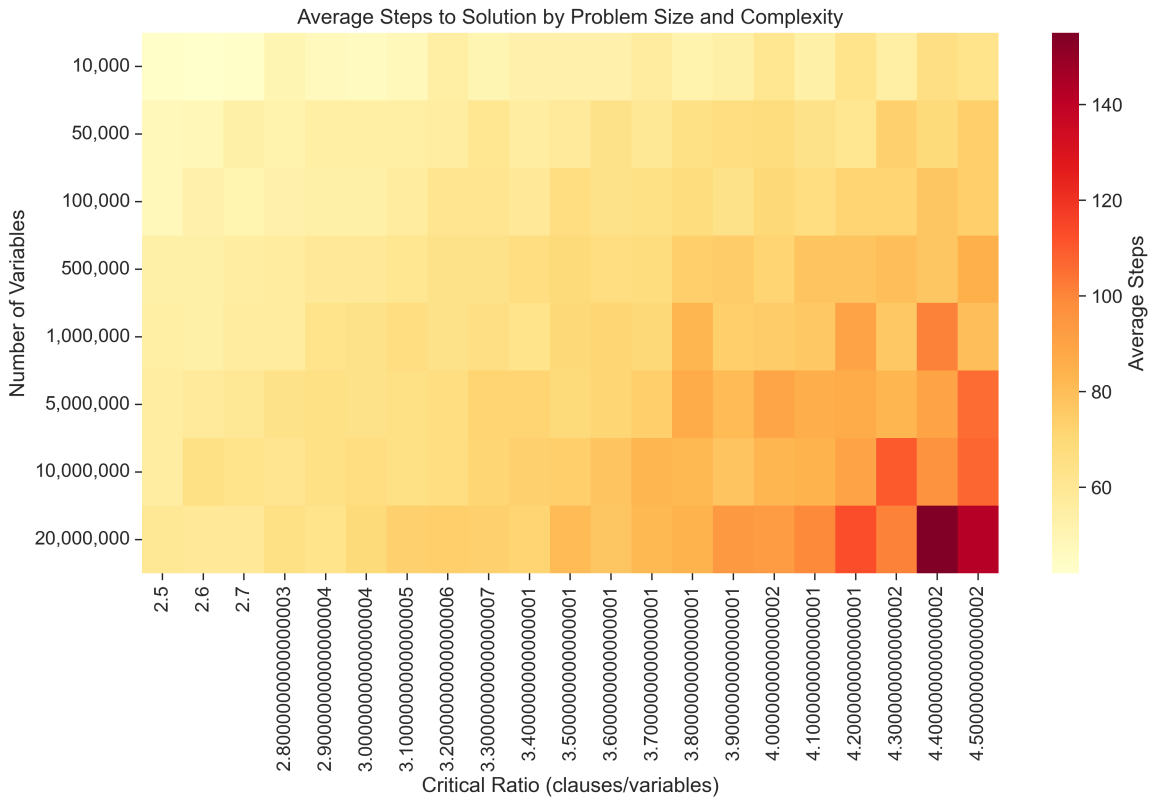


Figure 7: time-to-solution slightly increases at critical ratio (of large but easy instances)

```
root@C.18979334:/$ python3 2s.py
[GPU Active] b'NVIDIA H100 PCIe', Compute Capability: 9.0
Generating 1,406,459,999 clauses with seed 723151...
Initializing solver with deterministic seed 4...
Solving...
Step 0: C_mean=0.249996498227 avg_scale=1.9500
/usr/local/lib/python3.10/dist-packages/numba/cuda/dispatc
pancy.
  warn(NumbaPerformanceWarning(msg))
Step 10: C_mean=0.037377823144 avg_scale=1.0000
Step 20: C_mean=0.016352757812 avg_scale=1.0000
Step 30: C_mean=0.007561686449 avg_scale=1.0000
Step 40: C_mean=0.003183825174 avg_scale=1.0000
Step 50: C_mean=0.000790188089 avg_scale=1.0000
Step 60: C_mean=0.000134185262 avg_scale=1.0000
Step 70: C_mean=0.000006832993 avg_scale=1.0000
Step 80: C_mean=0.000000185265 avg_scale=1.0000
Step 90: C_mean=0.000000005834 avg_scale=1.0000
Step 100: C_mean=0.000000012093 avg_scale=1.0000
Step 110: C_mean=0.000000012133 avg_scale=1.0000
Step 120: C_mean=0.000000003753 avg_scale=1.0000
Solution found at step 123!
Elapsed time: 141.83 seconds
Solution found! Verifying satisfiability...
Solution is verified to satisfy all clauses.
```

Figure 8: Solving a very large (but easy) CNF, accelerated on a cloud-based NVIDIA H100 GPU

References

[1] Sean R. B. Bearden, Yan Ru Pei, and Massimiliano Di Ventra. "Efficient solution of Boolean satisfiability problems with digital memcomputing". In: *Scientific Reports* 10.1 (Nov. 2020). ISSN: 2045-2322. DOI: 10.1038/s41598-020-76666-2. URL: http://dx.doi.org/10.1038/s41598-020-76666-2.

[2] Chesson Sipling. *Phase-Space Engineering and Dynamical Long-Range Order in Memcomputing*. 2025. arXiv: 2506.10149 [physics.comp-ph].

[3] Yuan-Hang Zhang. *Self-averaging of digital memcomputing machines*. 2023. arXiv: 2301.08787 [cs.ET].