MatSat: a matrix-based differentiable SAT solver

Taisuke Sato(NII)○ Ryosuke Kojima(Kyoto univ.)

Matricized 3-SAT

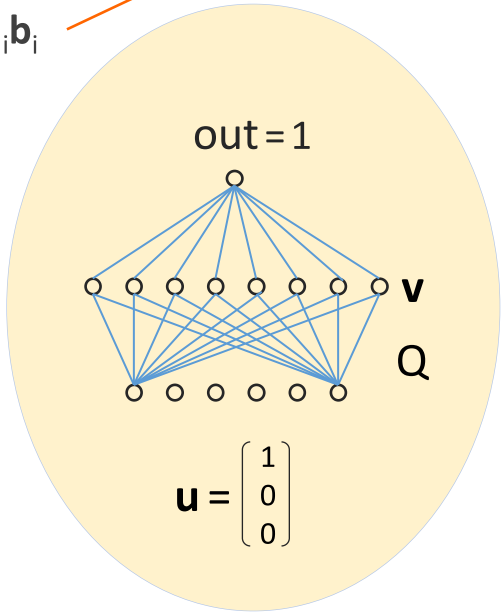
- $S = \{ \overbrace{a \vee b \vee \sim c}^{C_1},\ \overbrace{a \vee \sim b}^{C_2} \}$

$$\begin{array}{cc} & \begin{array}{cccccc} a & b & c & \sim a & \sim b & \sim c \end{array} & \\ Q = & \left[\begin{array}{cccccc} 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{array}\right] & \begin{array}{l} :C_1 \\ :C_2 \end{array} \\ & \begin{array}{cc} \underbrace{}_{Q_1} & \underbrace{}_{Q_2} \end{array} & \end{array}$$

$$\mathbf{u} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \begin{array}{l} :a \\ :b \\ :c \end{array} \Rightarrow \sim\mathbf{u} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \Rightarrow \mathbf{u}^d = \begin{bmatrix} \mathbf{u} \\ \sim\mathbf{u} \end{bmatrix}$$

$$\mathbf{v} = \min_1(Q\,\mathbf{u}^d)$$
$$= \min_1((Q_1 - Q_2)\mathbf{u} + Q_1\mathbf{1})$$
$$\text{out} = (\mathbf{1} \bullet (\mathbf{1} - \mathbf{v})) < 1$$

$(\mathbf{a} \bullet \mathbf{b}) = \Sigma_i\, \mathbf{a}_i \mathbf{b}_i$



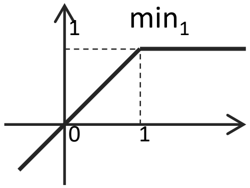
feed-forward NN

$$\min_1(Q\mathbf{u}^d) = \min_1\left(\begin{bmatrix} 2 \\ 2 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

⇒ C_1 is true

⇒ C_2 is true

⇒ S is satisfiable



$\min_1(x) = \min(x,1)$

SAT solving by minimizing J^{sat}

u: real vector

#false clause

$(\mathbf{u}\odot(\mathbf{1}\text{-}\mathbf{u}))_i = \mathbf{u}_i(1\text{-}\mathbf{u}_i)$

- $J^{sat} = \boxed{(\mathbf{1} \bullet (\mathbf{1}\text{-}\min_1(\mathbf{Q}\mathbf{u}^d)))} + \ell \, \| \mathbf{u}\odot(\mathbf{1}-\mathbf{u}) \|^2$

- $J^{sat} = 0$ ↔ $\mathbf{1} = \min_1(Q\mathbf{u}^d)$ & **u** is 0-1 vector

  ↔ $Qu^d \geq \mathbf{1}$ & **u** is 0-1 vector

  ↔ every clause has at least one true literal

  ↔ S is satisfied by **u**

All solutions are captured as a root of J^{sat}

Backpropagation for J^{sat}

- Jacobian $\mathbf{J}_{\mathbf{a}}^{\text{sat}}$ of J^{sat}
  - $\partial J^{\text{sat}}/\partial \mathbf{u}_p = (\mathbf{1} \bullet (-[\mathbf{c}<\mathbf{1}]\odot((Q_1-Q_2)\mathbf{I}_p))) + 2\ell\, ((\mathbf{u}\odot(\mathbf{1}-\mathbf{u})\odot(\mathbf{1}-2\mathbf{u})) \bullet \mathbf{I}_p)$

    $= ((-\,(Q_1-Q_2)^T[\mathbf{c}<\mathbf{1}] + 2\ell\, (\mathbf{u}\odot(\mathbf{1}-\mathbf{u})\odot(\mathbf{1}-2\mathbf{u})) \bullet \mathbf{I}_p)$

    where $\mathbf{I}_p = [0..1..0]^T$, $\mathbf{c} = Q\mathbf{u}^d$, $[\mathbf{c}<\mathbf{1}]_p = 1$ if $\mathbf{c}_p<1$, else $= 0$
  - $\mathbf{J}_{\mathbf{a}}^{\text{sat}} = -\,(Q_1-Q_2)^T[\mathbf{c}<\mathbf{1}] + 2\ell(\mathbf{u}\odot(\mathbf{1}-\mathbf{u})\odot(\mathbf{1}-2\mathbf{u}))$ ≈cubic polynomials

- Iterate (and binarize **u** as a solution) until $J^{\text{sat}} = 0$

$$\mathbf{u} = \mathbf{u} - \alpha\mathbf{J}_{\mathbf{a}}^{\text{sat}} = \mathbf{u} \boxed{+\, \alpha Q_1^T[\mathbf{c}<\mathbf{1}]} \boxed{-\, \alpha Q_2^T[\mathbf{c}<\mathbf{1}]} + \cdots \quad O(mn)$$

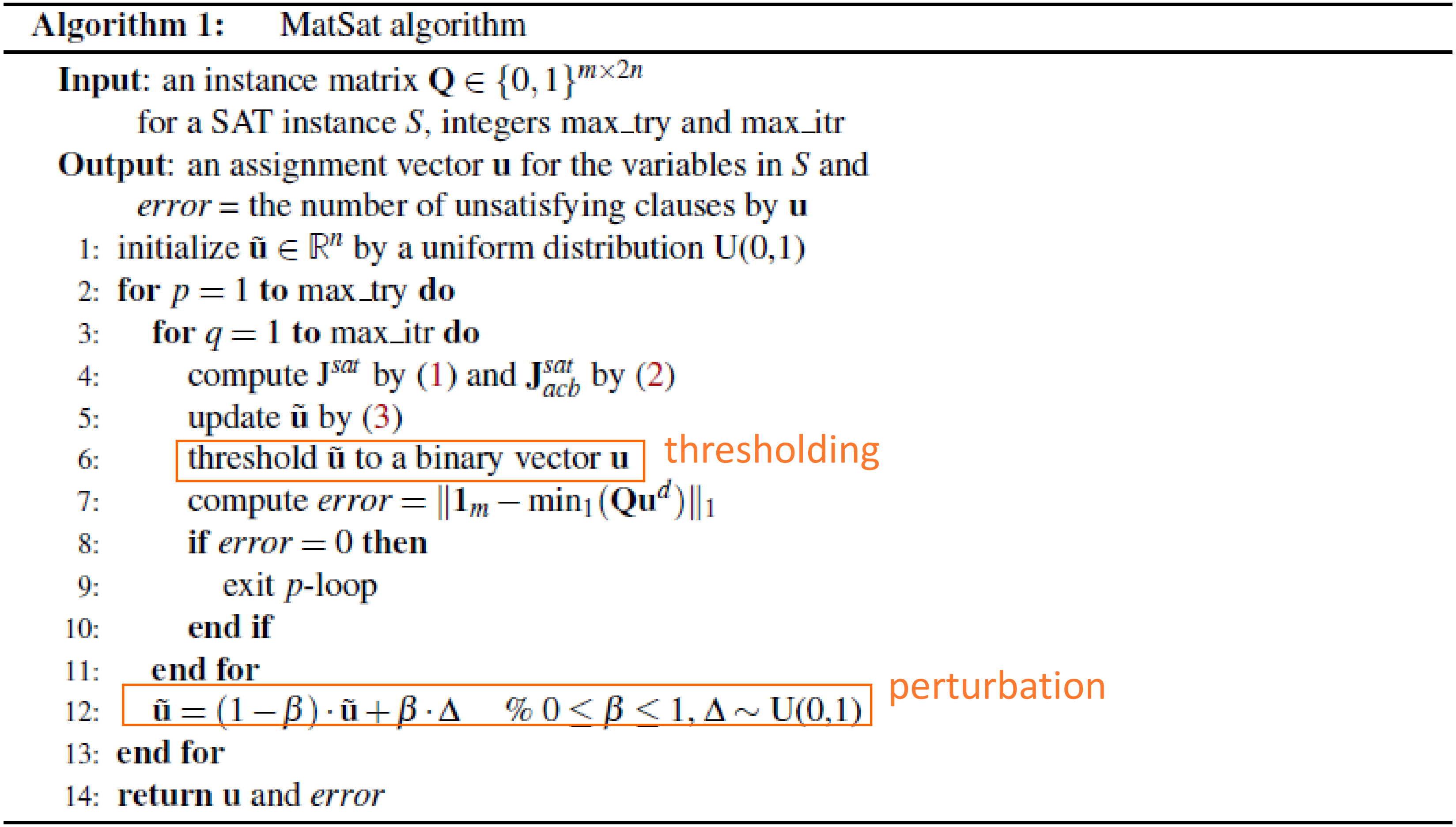
- $\mathbf{u}_p$ is increased ∝ |clauses falsified by **u** having positive literal p|
- $\mathbf{u}_p$ is decreased ∝ |clauses falsified by **u** having negative literal ~p|

MatSat

- MatSat: differentiable SAT solver based on matrix
  - neither CDCL nor SLS but NN
  - finds satisfying assignments in a vector space by minimizing a cost function J^{sat} to zero (by Newton's method)
  - incomplete, cannot solve unsat problems
  - scalable by multi-threads, GPU → parallel SAT solver
  - some similarity to LP approach and SDP approach to MAX 2-SAT in continuous relaxation but much simpler and direct

MatSat algorithm

- MatSat is a new type of SAT solver



Algorithm 1: MatSat algorithm

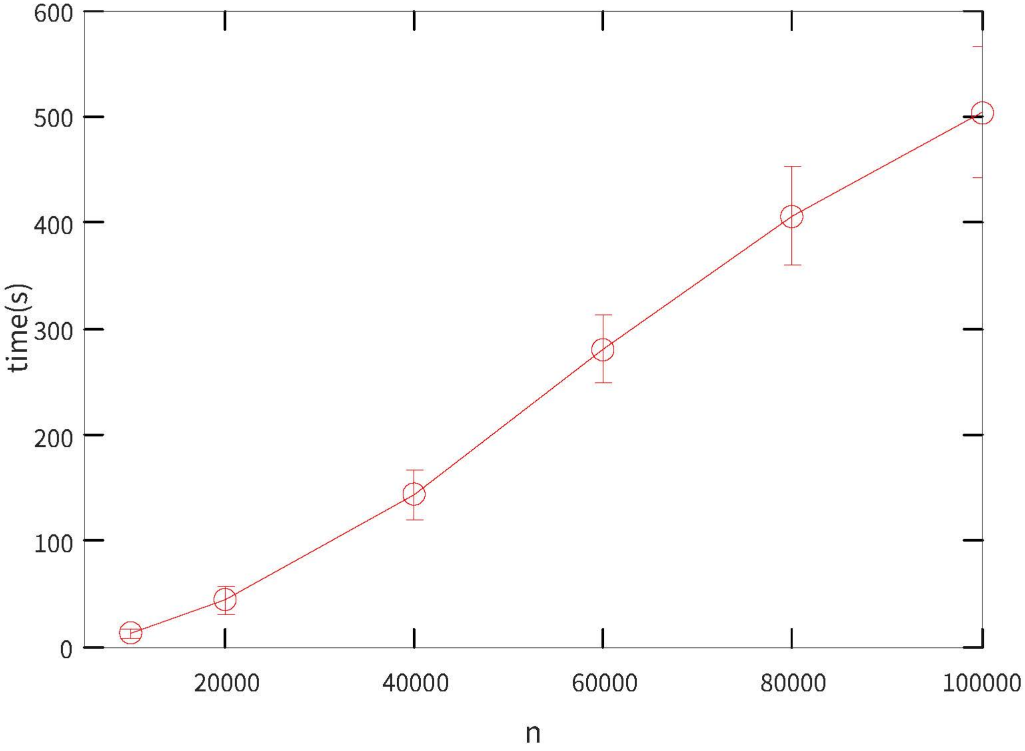
```
Input: an instance matrix $\mathbf{Q} \in \{0,1\}^{m \times 2n}$
       for a SAT instance $S$, integers max_try and max_itr
Output: an assignment vector $\mathbf{u}$ for the variables in $S$ and
       $error$ = the number of unsatisfying clauses by $\mathbf{u}$
initialize $\tilde{\mathbf{u}} \in \mathbb{R}^n$ by a uniform distribution U(0,1)
for $p = 1$ to max_try do
    for $q = 1$ to max_itr do
        compute $\mathbf{J}^{sat}$ by (1) and $\mathbf{J}^{sat}_{acb}$ by (2)
        update $\tilde{\mathbf{u}}$ by (3)
        threshold $\tilde{\mathbf{u}}$ to a binary vector $\mathbf{u}$
        compute $error = \|\mathbf{1}_m - \min_1(\mathbf{Q}\mathbf{u}^d)\|_1$
        if $error = 0$ then
            exit $p$-loop
        end if
    end for
    $\tilde{\mathbf{u}} = (1-\beta)\cdot\tilde{\mathbf{u}} + \beta\cdot\Delta$     % $0 \le \beta \le 1, \Delta \sim U(0,1)$
end for
return $\mathbf{u}$ and $error$
```

thresholding

perturbation

MatSat_sp: Scalability

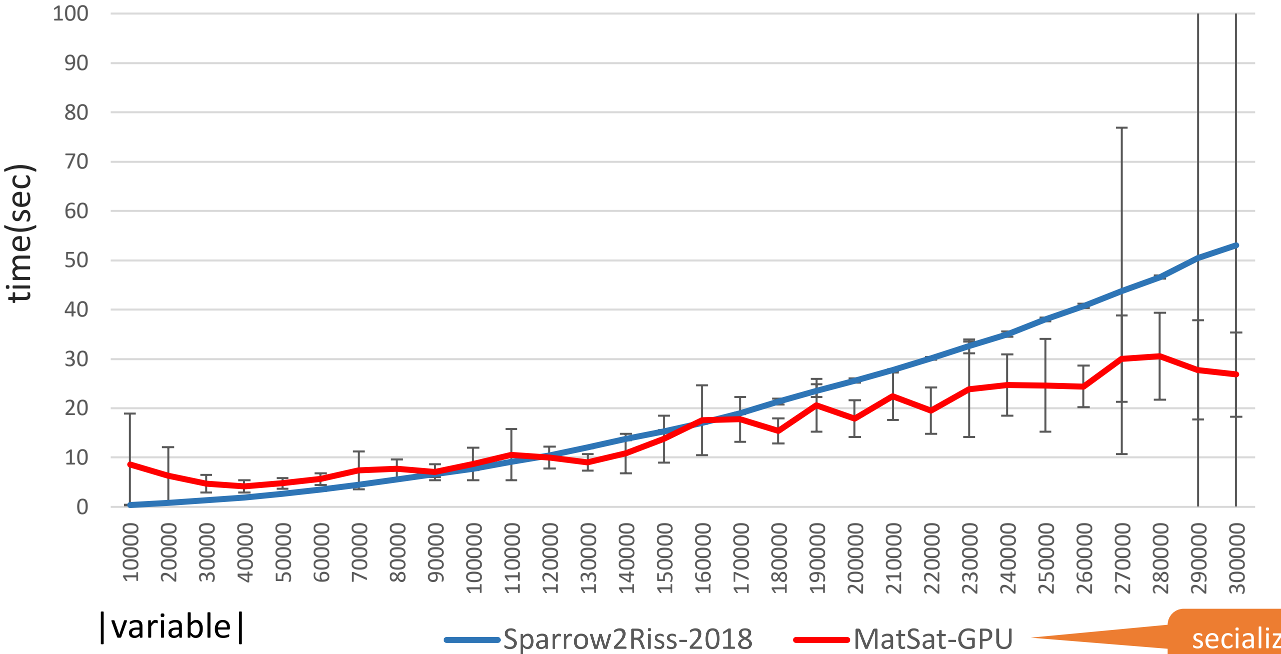
- Random 3-SAT (|var| = n, |clause| = 4.26*n)



n	10000	20000	40000	60000	80000	100000
#instance	5	5	5	5	5	5

MatSat_sp (OpenMP) on PC with Intel Core i7-10700 CPU@2.90GHz, 16-threads
max_try = 100, max_itr = 1000, 5 trials on each instance (all error = 0)

MatSat-GPU: Large scale



secialized implementation for C+GPU (CUDA)

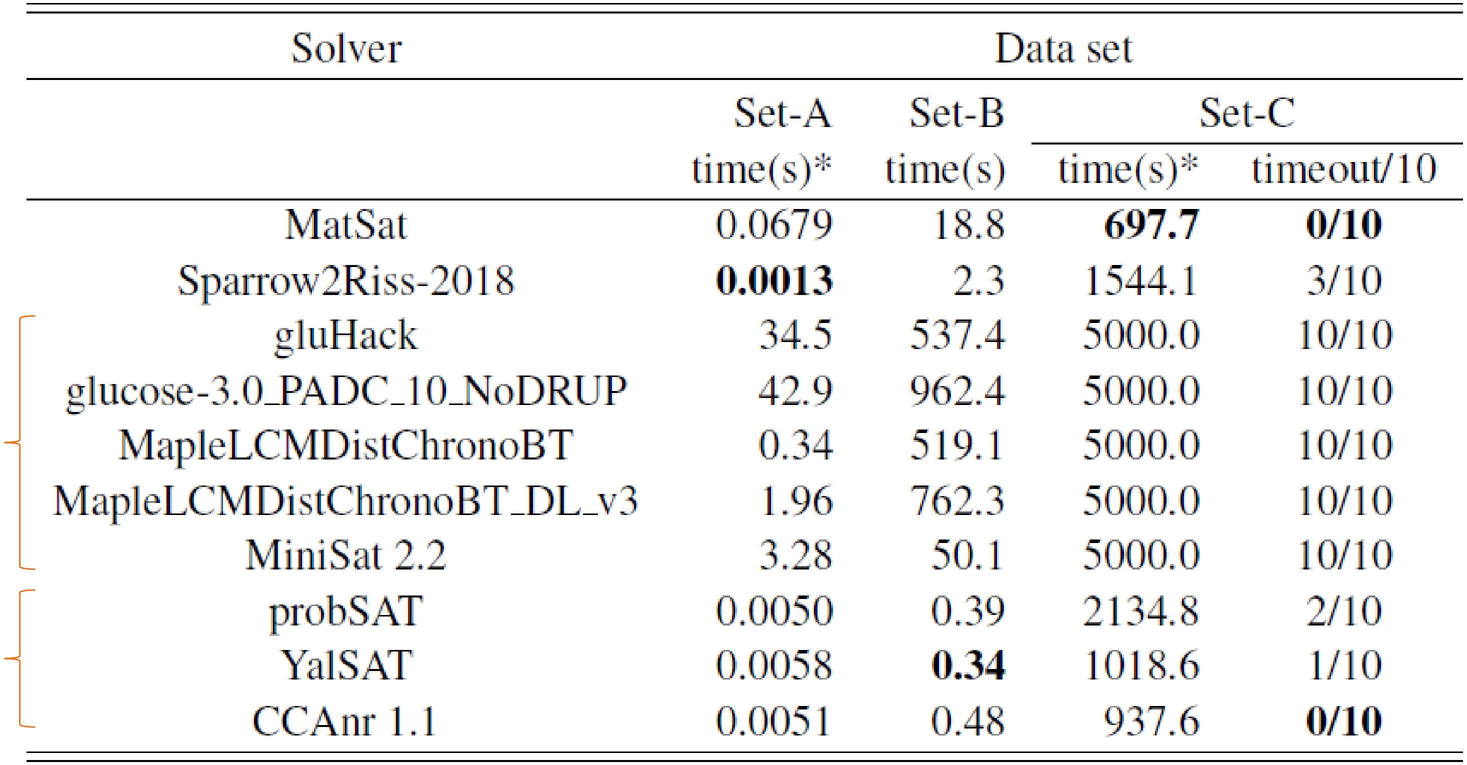
GPU: GeForce® GTX 1080 Ti

|variable| = num. of variables
|clause|/|variable| = 4.26 fixed
Ave. time measured with ±σ over 10 trials with different seed
under the setting max_itr = 1000, max_try = 100
* 7 trials not converged in the whole trials

Comparison with SAT solvers

Random SAT

timeout=5000s



	Solver	Data set			
		Set-A	Set-B	Set-C	
		time(s)*	time(s)	time(s)*	timeout/10
	MatSat	0.0679	18.8	**697.7**	**0/10**
	Sparrow2Riss-2018	**0.0013**	2.3	1544.1	3/10
CDCL	gluHack	34.5	537.4	5000.0	10/10
	glucose-3.0_PADC_10_NoDRUP	42.9	962.4	5000.0	10/10
	MapleLCMDistChronoBT	0.34	519.1	5000.0	10/10
	MapleLCMDistChronoBT_DL_v3	1.96	762.3	5000.0	10/10
	MiniSat 2.2	3.28	50.1	5000.0	10/10
SLS	probSAT	0.0050	0.39	2134.8	2/10
	YalSAT	0.0058	**0.34**	1018.6	1/10
	CCAnr 1.1	0.0051	0.48	937.6	**0/10**

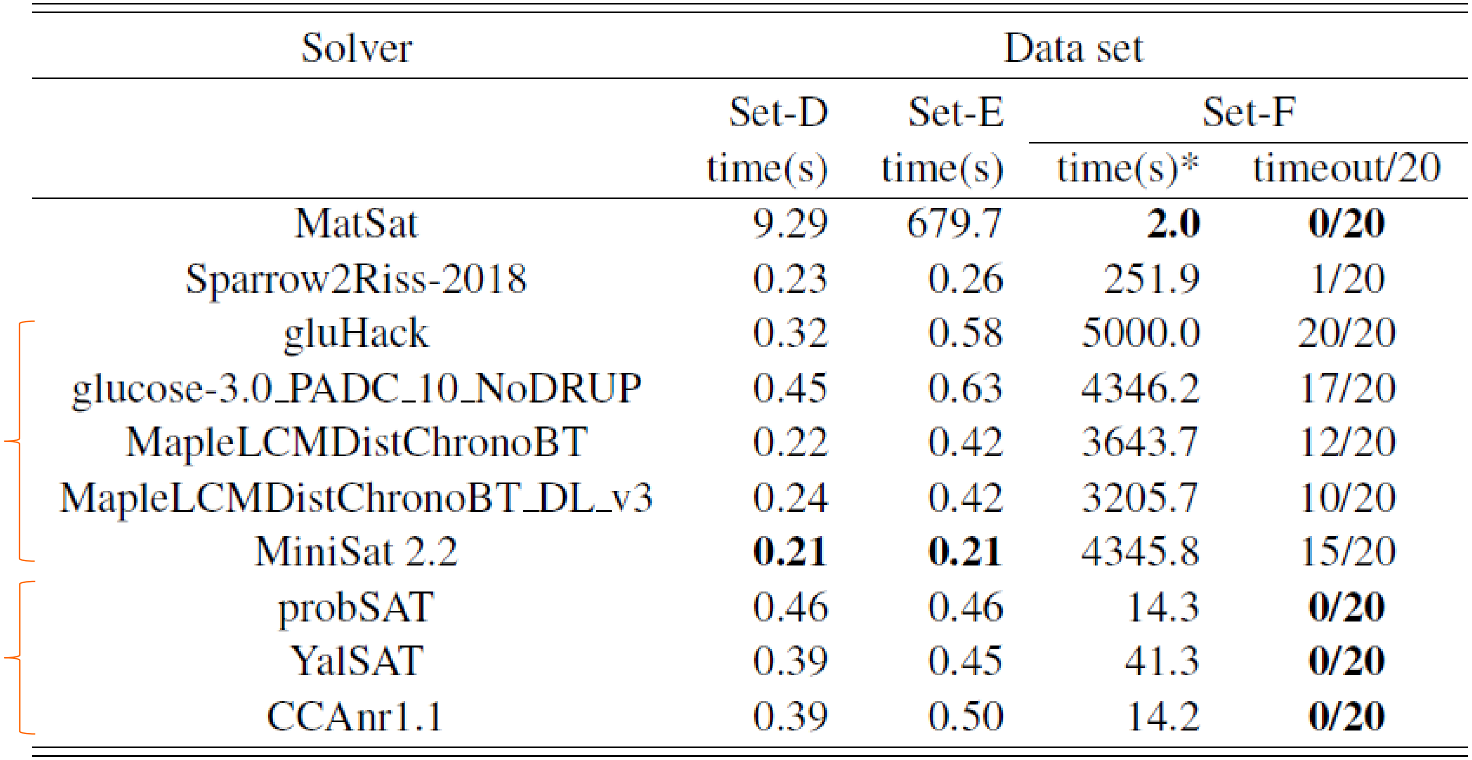
Set-A: 500 3-SAT instances generated as uniform random SAT
Set-B: /rnd-barthel (http://sat2018.forsyte.tuwien.ac.at/benchmarks/Random.zip)
Set-C: /Balint (http://sat2018.forsyte.tuwien.ac.at/benchmarks/Random.zip)

SAT Competition 2018
Random SAT Track: learning time(s)

	Set-B **rnd-barthel** **3-SAT (55 inst)**	**Set-C** **Balint** **5-SAT (10 inst)**		**Set-H** **rnd-qhid** **3-SAT (55 inst)**		**Set-I** **rnd-komb** **3-SAT (55 inst)**	
	time(s)/inst.	time/inst.	timout (5000s)	time(s)/inst.	timout (200s)	time(s)/inst.	timeout (200s)
MatSat_sp	18.8	**697.7**	**0/10**	**59.6**	**12/55**	193.6	51/55
max_(itr try)	(500 100)	(2k 5k)		(1k 100)		(2k 100k)	
S2R	2.3	1544.1	3/10	127.7	42/55	**1.0**	**0/55**
probSAT	0.39	2134.8	2/10	156.3	43/55	172.8	47/55
YalSAT	**0.34**	1018.6	1/10	156.3	43/55	178.1	50/55
CCAnr11	0.48	937.6	0/10	777.9	38/55	55.0	8/55

http://sat2018.forsyte.tuwien.ac.at/benchmarks/Random.zip

Non-random SAT(Set-{D,E})



	Solver	Data set			
		Set-D	Set-E	Set-F	
		time(s)	time(s)	time(s)*	timeout/20
	MatSat	9.29	679.7	**2.0**	**0/20**
	Sparrow2Riss-2018	0.23	0.26	251.9	1/20
CDCL	gluHack	0.32	0.58	5000.0	20/20
	glucose-3.0_PADC_10_NoDRUP	0.45	0.63	4346.2	17/20
	MapleLCMDistChronoBT	0.22	0.42	3643.7	12/20
	MapleLCMDistChronoBT_DL_v3	0.24	0.42	3205.7	10/20
	MiniSat 2.2	**0.21**	**0.21**	4345.8	15/20
SLS	probSAT	0.46	0.46	14.3	**0/20**
	YalSAT	0.39	0.45	41.3	**0/20**
	CCAnr1.1	0.39	0.50	14.2	**0/20**

timeout=5000s

	Set-D	Set-E	Set-F
k-SAT	3-SAT (100 inst)	5-SAT (100 inst)	5-SAT (20 inst)
(n m)	(90 300)	(500 3100)	(320 1120)

Set-D: SATLIB_FlatGraph/Flat30-60 (https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html)
Set-E: SATLIB_MorphGraph/SW100-8-0 (https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html)
Set-F: SAT2018_benchmark_Main_cnf/Jingchao_Chen (http://sat2018.forsyte.tuwien.ac.at/benchmarks/index.html)

Weighted variables and clauses

- MatSat performs poorly on non-random SAT
- Introduce variable weight **w**(v) and clause weight **w**(c) to simulate prioritized optimization of variables (just like unit propagation)

w(v) = (num. of variable v's occ. in the SAT instance)/(ave. var. weight)

➔ $\mathbf{w}_v = [\mathbf{w}(v_1) \cdots \mathbf{w}(v_n)]^T$

w(c) = sum of variable weights in a clause c

➔ $\mathbf{w}_c = [\mathbf{w}(c_1) \ldots \mathbf{w}(c_m)]^T$

- Define weighted $J^{\text{sat-w}}$ and compute Jacobian $\mathbf{J}_{\mathbf{a}}^{\text{sat-w}}$ as

$$J^{\text{sat-}w} = (\mathbf{1} \bullet \mathbf{w}_c \odot (\mathbf{1}\text{-min}_1(\mathbf{Q}\mathbf{u}^d))) + \ell \parallel \mathbf{w}_v \odot \mathbf{u} \odot (\mathbf{1} - \mathbf{u}) \parallel^2$$

$$\mathbf{J}_{\mathbf{a}}^{\text{sat-w}} = -\,(\mathbf{Q}_1\text{-}\mathbf{Q}_2)^T[\mathbf{w}_c \odot (\mathbf{Q}\mathbf{u}^d \leq 1)] + 2\ell(\mathbf{w}_v \odot \mathbf{w}_v \odot \mathbf{u} \odot (\mathbf{1}\text{-}\mathbf{u}) \odot (\mathbf{1}\text{-}2\mathbf{u}))$$

Weighted MatSat: learning time(s)

- The weighted version runs much faster (as far as Set-{D,E,G} concerned)

MatSat_sp	Set-D 3-SAT (100 inst)	Set-E 5-SAT (100 inst)	Set-G 5-SAT (100 inst)
non-weighted	9.29	679.7	1277.3
max_(itr try)	(500 100)	(2k 100)	(5k 100)
weighted	2.18	82.4	123.7
max_(itr try)	(100 300)	(1k 100)	(200 100)

Set-G: SATLIB MorphGraph/sw100-8-lp1-c5 (https://www.cs.ubc.ca/˜hoos/SATLIB/benchm.html)

Conclusion

- MatSat is a new type of SAT solver based on matrix
  - solution search by cost minimization in a vector space
  - NN with a single layer for logical operations
  - seems competitive w.r.t. random SAT
  - declarative, simple and scalable (by many-cores and GPUs)
  - under development for improvement
    - variable, clause weight, dynamic weighting
    - ℓ_1 norm, activating function etc
- Structured problems are hard → future challenge
- Extending to weighted MAX-SAT is straightforward

$$J^{\text{max-sat}} = (\mathbf{w} \bullet (\mathbf{1}\text{-}\min_1(\mathbf{Q}\mathbf{u}^d))) + \cdots$$