

Enhanced Lower Bound Computation in Branch-and-Bound for MaxSAT

Jialu Zhang¹ Chu-Min Li¹ Sami Cherif¹ Shuolin Li²

¹UR MIS UR 4290, Université de Picardie Jules Verne, Amiens, France

²Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

CP 2026 — Lisbon, Portugal — July 2026



- 1 Introduction
- 2 BnB MaxSAT Algorithm
- 3 Further Lookahead (FLA)
- 4 Experiment Results
- 5 Conclusion

- 1 Introduction
- 2 BnB MaxSAT Algorithm
- 3 Further Lookahead (FLA)
- 4 Experiment Results
- 5 Conclusion

Definitions: MaxSAT

Let X be a set of Boolean variables.

- A **literal** l is a variable $x \in X$ or its negation $\neg x$.
- A **clause** $c = l_1 \vee l_2 \vee \dots \vee l_k$ is a disjunction of literals.
- A **CNF formula** $F = c_1 \wedge c_2 \wedge \dots \wedge c_m$ is a conjunction of clauses.

Definitions: MaxSAT

Let X be a set of Boolean variables.

- A **literal** l is a variable $x \in X$ or its negation $\neg x$.
- A **clause** $c = l_1 \vee l_2 \vee \cdots \vee l_k$ is a disjunction of literals.
- A **CNF formula** $F = c_1 \wedge c_2 \wedge \cdots \wedge c_m$ is a conjunction of clauses.

SAT Problem

Find an assignment α that satisfies all clauses in F .

MaxSAT Problem

Find an assignment α that **maximizes the number** of satisfied clauses in F .

Partial MaxSAT (PMS)

Clauses are partitioned into **hard** H and **soft** S .

Goal: satisfy **all hard** clauses while **minimizing the cost** of soft clauses,

Partial MaxSAT (PMS)

Clauses are partitioned into **hard** H and **soft** S .

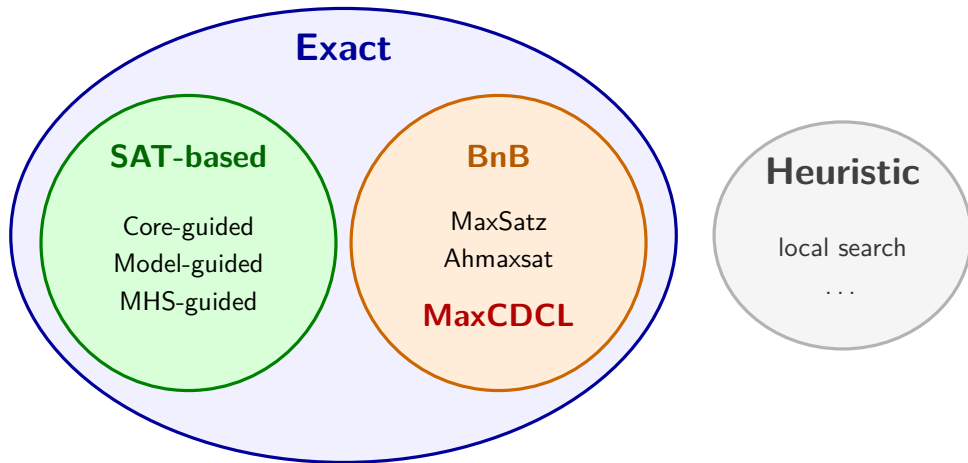
Goal: satisfy **all hard** clauses while **minimizing the cost** of soft clauses,

Weighted PMS (WPMS)

Each soft clause $c \in S$ has a weight w_c :

the **cost** becomes the **total weight** of falsified soft clauses.

MaxSAT Algorithmic Approaches



MaxSAT Algorithmic Approaches

Results: Exact Track

Only one configuration per solver included

Unweighted: 607 instances

Solver	# Solved
VBS	477
UWrMaxSAT	424
UWrMaxSatMaxCDCL	423
WMaxCDCL2026S6O3	417
MaxCDCL2026S2O3	415
CASHWMaxSAT	414

Weighted: 609 instances

Solver	# Solved
VBS	470
UWrMaxSAT	456
WMaxCDCL2026S6 +2	436
EvalMaxRes	434
CASHWMaxSATGoal +2	398
MaxHS	393

MaxCDCL: a state-of-the-art BnB solver enhanced with clause learning

- 1 Introduction
- 2 BnB MaxSAT Algorithm
- 3 Further Lookahead (FLA)
- 4 Experiment Results
- 5 Conclusion

BnB MaxSAT: Lower Bounds

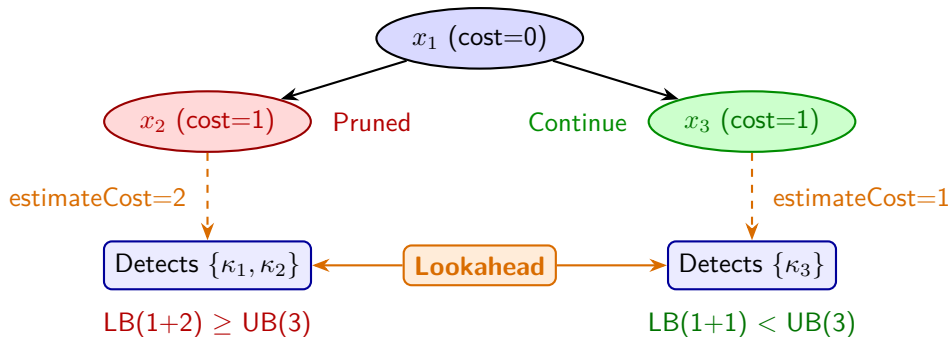
Lower Bound (LB) for a partial assignment α :

$$\text{LB}(\alpha) = \underbrace{\text{cost}(\alpha)}_{\text{falsified soft clauses}} + \underbrace{\text{estimateCost}(\alpha)}_{\text{potentially falsified soft clauses}}$$

BnB MaxSAT: Lower Bounds

Lower Bound (LB) for a partial assignment α :

$$\text{LB}(\alpha) = \underbrace{\text{cost}(\alpha)}_{\text{falsified soft clauses}} + \underbrace{\text{estimateCost}(\alpha)}_{\text{potentially falsified soft clauses}}$$



Local Core

A **local core** κ is a set of **soft literals** that **cannot be satisfied at the same time** under the current partial assignment α .

Local Core

A **local core** κ is a set of **soft literals** that **cannot be satisfied at the same time** under the current partial assignment α .

Core Detection

Lookahead detects local cores by **unit propagation** under α .

Local Core

A **local core** κ is a set of **soft literals** that **cannot be satisfied at the same time** under the current partial assignment α .

Core Detection

Lookahead detects local cores by **unit propagation** under α .

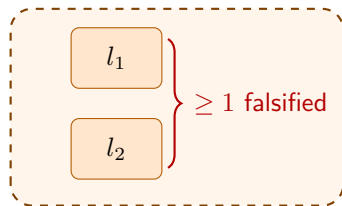
LB Computation

$$\text{LB}(\alpha) = \text{cost}(\alpha) + \sum_{\kappa \in K} w(\kappa)$$

K : the set of **disjoint** local cores; $w(\kappa)$: minimal false soft literals in κ .

BnB MaxSAT: Local Core Example

Core κ_1 , $w(\kappa_1)=1$



Not in core (free)



Any extension of α must **falsify at least one** soft literal in $\{l_1, l_2\}$.

l_1 and l_2 are **locked**, but l_3 is **free**.

BnB MaxSAT: Unlocking Mechanism

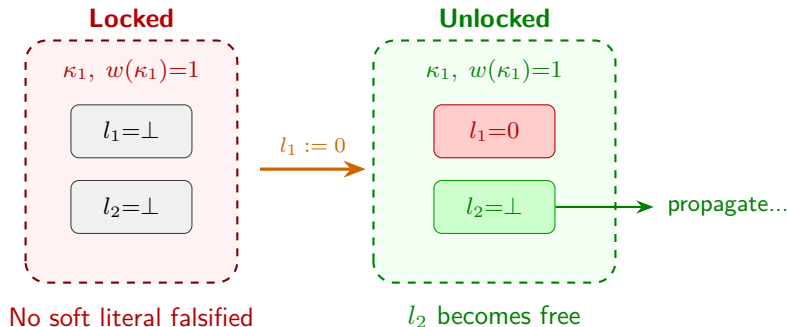
How to unlock a core?

Once $\geq w(\kappa)$ soft literals in core κ are falsified, κ become **unlocked**.

BnB MaxSAT: Unlocking Mechanism

How to unlock a core?

Once $\geq w(\kappa)$ soft literals in core κ are falsified, κ become **unlocked**.



- 1 Introduction
- 2 BnB MaxSAT Algorithm
- 3 Further Lookahead (FLA)**
- 4 Experiment Results
- 5 Conclusion

Drawback of Standard Lookahead

Local cores are unlocked during unit propagation *passively*, **some cores can never be detected** by standard lookahead.

Further Lookahead: Key Idea

Drawback of Standard Lookahead

Local cores are unlocked during unit propagation *passively*, **some cores can never be detected** by standard lookahead.

Key Idea of Further Lookahead

Instead of waiting for a local core to be unlocked *passively*, **actively unlock** some local cores and detect more cores based these **unlocked soft literals**.

Further Lookahead: Example

Example 1

$$S = \{l_1, l_2, l_3\}, \alpha = \emptyset, UB = 2$$

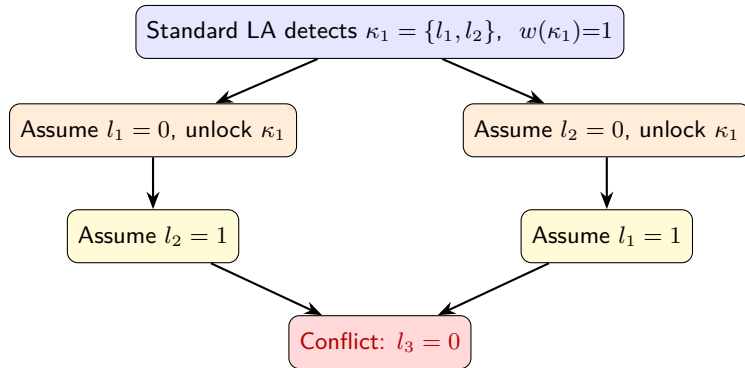
$$H = (\neg l_1 \vee x_1) \wedge (\neg l_2 \vee \neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge (\neg l_1 \vee l_2 \vee \neg l_3) \wedge (l_1 \vee \neg l_2 \vee \neg l_3)$$

Further Lookahead: Example

Example 1

$S = \{l_1, l_2, l_3\}$, $\alpha = \emptyset$, $UB = 2$

$H = (\neg l_1 \vee x_1) \wedge (\neg l_2 \vee \neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge (\neg l_1 \vee l_2 \vee \neg l_3) \wedge (l_1 \vee \neg l_2 \vee \neg l_3)$

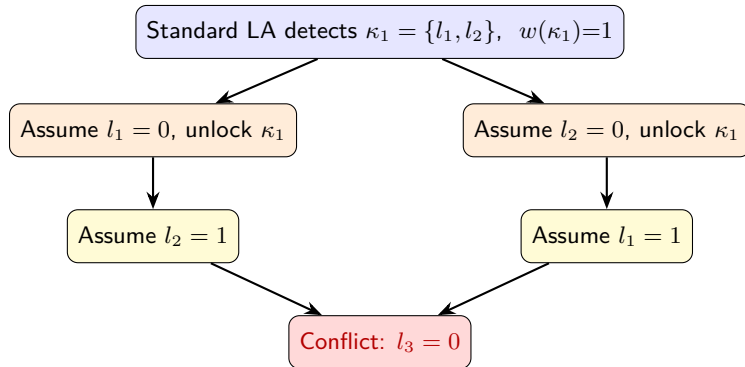


Further Lookahead: Example

Example 1

$S = \{l_1, l_2, l_3\}$, $\alpha = \emptyset$, $UB = 2$

$H = (\neg l_1 \vee x_1) \wedge (\neg l_2 \vee \neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge (\neg l_1 \vee l_2 \vee \neg l_3) \wedge (l_1 \vee \neg l_2 \vee \neg l_3)$



Result

New core: $\kappa_2 = \{l_1, l_2, l_3\}$

Core weight: $w(\kappa_2) = 2$

$LB = 2 \geq UB \Rightarrow$ **Prune!**

Q-learning Strategy: When to Invoke FLA?

FLA is powerful but costly

Invoking FLA at **every** failed standard lookahead is inefficient

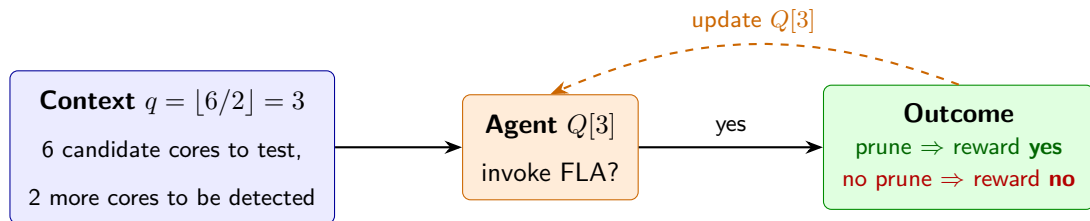
Use Reinforcement Learning (RL) strategy to learn **when** it is worth invoking.

Q-learning Strategy: When to Invoke FLA?

FLA is powerful but costly

Invoking FLA at **every** failed standard lookahead is inefficient

Use Reinforcement Learning (RL) strategy to learn **when** it is worth invoking.



Standard Lookahead vs Further Lookahead

Standard Lookahead

Passively unlock cores
Needs fewer propagations
Worse LB

vs

Further Lookahead

Actively unlock cores
Needs more propagations
Tighter LB

FLA $\Rightarrow$ more cores $\Rightarrow$ tighter LB $\Rightarrow$ more pruning $\Rightarrow$ faster solving

- 1 Introduction
- 2 BnB MaxSAT Algorithm
- 3 Further Lookahead (FLA)
- 4 Experiment Results**
- 5 Conclusion

Experimental Setup

Benchmarks

2209 PMS instances from MSE 2019–2024, spanning 74 diverse families

Experimental Setup

Benchmarks

2209 PMS instances from MSE 2019–2024, spanning 74 diverse families

Solvers

- **enMaxCDCL**: enhanced MaxCDCL2024, the baseline
- **enMaxCDCL+FLA(λ)**: adds RL-based Further Lookahead
- **enMaxCDCL+alwaysFLA**: FLA at every failed lookahead, no RL
- Best MSE 2024 solvers: EvalMaxSAT, UWrrMaxSat, CASHWMaxSAT, ...

Experimental Setup

Benchmarks

2209 PMS instances from MSE 2019–2024, spanning 74 diverse families

Solvers

- **enMaxCDCL**: enhanced MaxCDCL2024, the baseline
- **enMaxCDCL+FLA(λ)**: adds RL-based Further Lookahead
- **enMaxCDCL+alwaysFLA**: FLA at every failed lookahead, no RL
- Best MSE 2024 solvers: EvalMaxSAT, UWrrMaxSat, CASHWMaxSAT, ...

Environment & metrics

1 CPU core, 3600s timeout; #solved and PAR2

Tuning the Hyperparameter λ

λ : the **maximum size** of the cores FLA can unlock ($|\kappa| \leq \lambda$)

	MS19	MS20	MS21	MS22	MS23	MS24	Total	PAR2
#Instances	599	401	448	254	260	247	2209	–
enMaxCDCL	432	319	338	191	181	196	1657	1934.34
enMaxCDCL+FLA($\lambda=2$)	432	319	340	192	182	197	1662	1923.80
enMaxCDCL+FLA($\lambda=3$)	434	322	343	192	182	197	1670	1907.23
enMaxCDCL+FLA($\lambda=4$)	435	321	342	192	182	199	1671	1904.77
enMaxCDCL+FLA($\lambda=5$)	434	322	341	192	181	197	1667	1913.58
enMaxCDCL+alwaysFLA	432	322	341	190	179	199	1663	1926.42

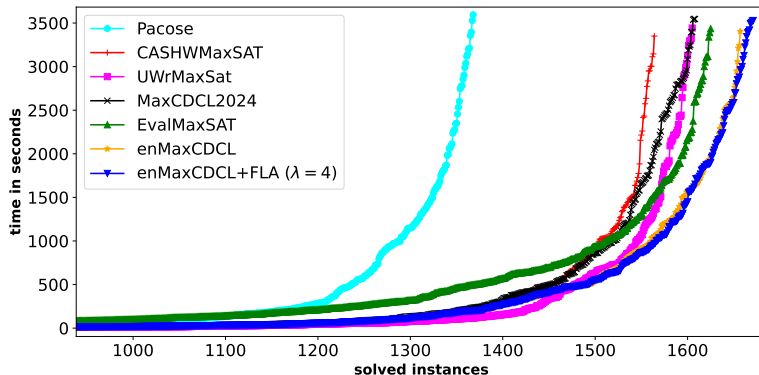
- $\lambda \in \{2, 3, 4, 5\}$ solves **+5, +13, +14, +10** instances over the baseline
- Larger $\lambda \Rightarrow$ more new cores, but also more overhead: $\lambda = 4$ is the sweet spot
- alwaysFLA solves **8 fewer** instances than FLA($\lambda=4$) $\Rightarrow$ the RL strategy pays off

Overall Solver Comparison

	MS19	MS20	MS21	MS22	MS23	MS24	Total	PAR2
#Instances	599	401	448	254	260	247	2209	–
Pacose	347	255	288	167	148	163	1368	2849.95
CASHWMaxSAT	409	307	334	176	170	168	1564	2189.04
UWrMaxSat	421	317	338	179	179	171	1605	2063.29
MaxCDCL2024	433	301	330	177	177	190	1608	2093.27
EvalMaxSAT	440	311	341	181	178	174	1625	2088.30
enMaxCDCL	432	319	338	191	181	196	1657	1934.34
enMaxCDCL+FLA($\lambda=4$)	435	321	342	192	182	199	1671	1904.77

- enMaxCDCL already leads its closest competitor by **32** instances
- FLA adds **+14** solved instances on top, improving *every* benchmark year
- enMaxCDCL+FLA uniquely solves **19** instances (optic, railroad, uaq, maxcut, ...)

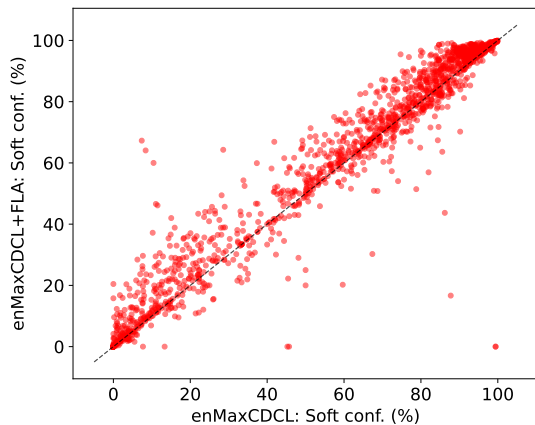
Cactus Plot: Where Does FLA Help?



Results: enMaxCDCL+FLA vs. baseline

- **+14** solved instances, improving every benchmark year; **1.5%** better PAR2
- Gains grow on **hard instances**: at 3000–3600s, **4** $\rightarrow$ **13** solved; **19** uniquely solved

Impact of FLA

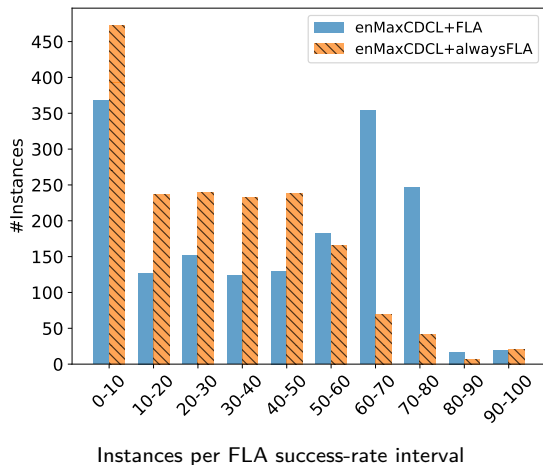


Soft conflict rate: FLA vs baseline

Key observations

- Average soft conflict rate:
52.54% $\rightarrow$ 54.96% (**+2.42%**)
 $\Rightarrow$ stronger pruning
- On the 19 uniquely solved instances, the average soft conflict rate is **79.09%**

Impact of the RL Strategy



Key observations

- **Fewer invocations:**
average FLA CPU time 277.81s → 211.16s (−24%)
- **Higher success rate:**
>50% on **818** instances with RL vs **301** without

- 1 Introduction
- 2 BnB MaxSAT Algorithm
- 3 Further Lookahead (FLA)
- 4 Experiment Results
- 5 Conclusion**

Contributions

- **Further Lookahead (FLA)**: actively unlocks cores to tighten lower bounds
- **Q-learning strategy**: decides per call whether FLA is worth invoking

Conclusion

Contributions

- **Further Lookahead (FLA)**: actively unlocks cores to tighten lower bounds
- **Q-learning strategy**: decides per call whether FLA is worth invoking

Results (2209 PMS instances, MSE 2019–2024)

+14 solved instances and **1.5%** better PAR2 over the strong baseline;
the RL strategy cuts FLA overhead by **24%**

Conclusion

Contributions

- **Further Lookahead (FLA)**: actively unlocks cores to tighten lower bounds
- **Q-learning strategy**: decides per call whether FLA is worth invoking

Results (2209 PMS instances, MSE 2019–2024)

+14 solved instances and **1.5%** better PAR2 over the strong baseline;
the RL strategy cuts FLA overhead by **24%**

Future Work

Extend FLA and the RL strategy to **weighted partial MaxSAT**

Thank you!

`jialu.zhang@u-picardie.fr`



Full paper



Source code