A BDD-engine for computations on monomial ideals

ISSAC'26 — 51st International Symposium on Symbolic and Algebraic Computation

Laura Moreno-Resa Eduardo Sáenz-de-Cabezón
lamorer@unirioja.es esaenz-d@unirioja.es
University of La Rioja, Logroño, Spain

July 13–17, 2026 · Oldenburg, Germany

Outline

Motivation

Two Representations

Boolean Operations on Ideals

Information from $\mathcal{B}_I$

Binary Decision Diagrams

Implementation & Experiments

Conclusions & Future Work

Motivation

Why monomial ideals?

Monomial ideals are a **central object** in combinatorial commutative algebra, connecting:

- **Stanley–Reisner correspondence**
  square-free ideals $\longleftrightarrow$ simplicial complexes
- **Gröbner basis theory**
  commutative algebra and algebraic geometry
- **Hochster's formula**
  Betti numbers $\leftrightarrow$ simplicial cohomology
- **Edge / cover ideals**
  graphs and networks
- **Coherent systems**
  reliability theory

Key observation

Any improvement in *representing* and *computing* with monomial ideals has **broad impact** across all these areas.

Standard tool: computer algebra systems (CoCoALib, Macaulay2, ...)

Question: can we do better using Boolean data structures?

The central idea of this work

Main thesis

Square-free monomial ideal $\longleftrightarrow$ Monotone Boolean function

Use Binary Decision Diagrams as the data structure $\rightarrow$ **faster** for several key operations.

What we do:

- **Compare Representations:** $G(I)$ vs. Monotone Boolean function $\mathcal{B}_I$.
- **Information Retrieval:** Inspect what algebraic data is best read from each representation.
- **BDD-Engine:** Implement a new computational framework via BDDs.

Some results:

- ✓ Intersection
- ✓ Standard monomials
- ✓ Alexander dual
- − Sum (competitive)
- ✗ Colon ideal (M2 wins)

Two Representations

Representation 1: Minimal generating set $G(I)$

Setup

$R = k[x_1, \ldots, x_n]$, $I \subseteq R$ a square-free monomial ideal.
$G(I)$ = unique minimal monomial generating set (Dickson's lemma)

- **Membership:**

$$\mu \in I \iff \exists \nu \in G(I),\ \nu \mid \mu$$

- **Radical:** If I is square-free,

$$\sqrt{I} = I.$$

In general, the radical of any monomial ideal is square-free.

Standard approach in Computer algebra systems:

Most computations reduce to *combinatorial operations* on $G(I)$

— well-studied, highly optimised — but can suffer from **intermediate expression swell**

Representation 2: Monotone Boolean function $\mathcal{B}_I$

Each square-free monomial $\boldsymbol{x^a}$ corresponds to its Boolean exponents vector $\boldsymbol{a}$. It induces:

$$\mathcal{B}_{\boldsymbol{x^a}} : \{0,1\}^n \to \{0,1\}, \text{ where } \mathcal{B}_{\boldsymbol{x^a}}(\boldsymbol{b}) = 1 \iff \boldsymbol{b} \geq \boldsymbol{a}, \text{ then } \mathcal{B}_{\boldsymbol{x^a}} = \bigwedge_{a_i \neq 0} x_i$$

Boolean function of I

$$\mathcal{B}_I : \{0,1\}^n \to \{0,1\}, \text{ where } \mathcal{B}_I(\boldsymbol{b}) = 1 \iff \boldsymbol{b} \in I$$

$$\mathcal{B}_I(\boldsymbol{x}) = \bigvee_{\boldsymbol{x^a} \in G(I)} \bigwedge_{a_i \neq 0} x_i$$

Bijection: square-free monomial ideals $\longleftrightarrow$ monotone Boolean functions

Membership: $\boldsymbol{x^a} \in I \iff \mathcal{B}_I(\boldsymbol{x^a}) = 1$

The DNF of $\mathcal{B}_I$ is a *direct rewriting* of $G(I)$ — this is the key **bridge** between the two perspectives.

Switching between representations

$G(I) \to \mathcal{B}_I$ (easy)

Direct transcription of generators
$\to$ irreducible DNF of $\mathcal{B}_I$

Cost: essentially free

$\mathcal{B}_I \to G(I)$ (hard)

Given $\mathcal{B}_I$ as CNF: this is the **monotone Boolean dualization** problem — quasi-polynomial time (Fredman–Khachiyan 1996)

Important remark

Satisfability in monotone CNF is **NP-complete**; in monotone DNF, satisfiability is **polynomial** — which is why we work with DNF.

Boolean Operations on Ideals

Boolean operations on monomials

Monomial op.	Boolean expression	Native BDD?
$\mathrm{lcm}(\boldsymbol{x}^{\boldsymbol{a}}, \boldsymbol{x}^{\boldsymbol{b}})$	$\mathcal{B}_{\boldsymbol{x}^{\boldsymbol{a}\vee\boldsymbol{b}}} = \mathcal{B}_{\boldsymbol{x}^{\boldsymbol{a}}} \wedge \mathcal{B}_{\boldsymbol{x}^{\boldsymbol{b}}}$	✅
$\gcd(\boldsymbol{x}^{\boldsymbol{a}}, \boldsymbol{x}^{\boldsymbol{b}})$	$\mathcal{B}_{\boldsymbol{x}^{\boldsymbol{a}\wedge\boldsymbol{b}}}$	❌
$\boldsymbol{x}^{\boldsymbol{a}}/\boldsymbol{x}^{\boldsymbol{b}}$	$\mathcal{B}_{\boldsymbol{x}^{\boldsymbol{a}}\oplus\boldsymbol{x}^{\boldsymbol{b}}}$	❌

Boolean expression for division

For a variable to be in $\boldsymbol{x}^{\boldsymbol{a}}/\boldsymbol{x}^{\boldsymbol{b}}$, it must be in $\boldsymbol{x}^{\boldsymbol{a}}$ but not in $\boldsymbol{x}^{\boldsymbol{b}}$ ($\neg(\boldsymbol{x}^{\boldsymbol{a}} \to \boldsymbol{x}^{\boldsymbol{b}})$). Since $\mathrm{supp}(\boldsymbol{x}^{\boldsymbol{b}}) \subseteq \mathrm{supp}(\boldsymbol{x}^{\boldsymbol{a}})$, this simplifies to:

$$\mathcal{B}_{\boldsymbol{x}^{\boldsymbol{a}}/\boldsymbol{x}^{\boldsymbol{b}}} = \mathcal{B}_{\boldsymbol{x}^{\boldsymbol{a}}\oplus\boldsymbol{x}^{\boldsymbol{b}}}$$

Sum and intersection of ideals

Sum of ideals

$$\mathcal{B}_{I+J} = \mathcal{B}_I \vee \mathcal{B}_J$$

Algebraic cost: Concatenate and minimize the union by checking divisibility:

$$G(I + J) \subseteq G(I) \cup G(J)$$

BDD cost: $\mathcal{O}(1)$ non-algebraic cost (exactly one native OR operation).

Intersection of ideals

$$\mathcal{B}_{I \cap J} = \mathcal{B}_I \wedge \mathcal{B}_J$$

Algebraic cost: The initial set has $r \cdot s$ elements:

$$I \cap J = \langle \mathrm{lcm}(m_i, n_j) \mid 1 \leq i \leq r, 1 \leq j \leq s \rangle$$

BDD cost: $\mathcal{O}(1)$ non-algebraic cost (exactly one native AND operation).

Colon Ideal of Square-Free Ideals

1. Principal monomial ideals

$$\langle \boldsymbol{x}^{\boldsymbol{a}} \rangle : \langle \boldsymbol{x}^{\boldsymbol{b}} \rangle = \left\langle \frac{\boldsymbol{x}^{\boldsymbol{a}}}{\gcd(\boldsymbol{x}^{\boldsymbol{a}}, \boldsymbol{x}^{\boldsymbol{b}})} \right\rangle$$

Boolean representation:

$$\mathcal{B}_{\langle \boldsymbol{x}^{\boldsymbol{a}} \rangle : \langle \boldsymbol{x}^{\boldsymbol{b}} \rangle} = \mathcal{B}_{\boldsymbol{x}^{\boldsymbol{a}} \wedge \neg \boldsymbol{x}^{\boldsymbol{b}}}$$

Logic: For a variable to remain, it must be in the numerator ($\boldsymbol{x}^{\boldsymbol{a}}$) but *not* in the gcd ($\boldsymbol{x}^{\boldsymbol{a}} \wedge \boldsymbol{x}^{\boldsymbol{b}}$), simplifying to $\boldsymbol{x}^{\boldsymbol{a}} \wedge \neg \boldsymbol{x}^{\boldsymbol{b}}$.

2. General square-free ideals

$$I : J = \bigcap_{\boldsymbol{x}^{\boldsymbol{b}} \in G(J)} \sum_{\boldsymbol{x}^{\boldsymbol{a}} \in G(I)} \langle \boldsymbol{x}^{\boldsymbol{a}} \rangle : \langle \boldsymbol{x}^{\boldsymbol{b}} \rangle$$

Boolean representation:

$$\mathcal{B}_{I:J} = \bigvee_{\boldsymbol{x}^{\boldsymbol{a}} \in G(I)} \bigwedge_{\boldsymbol{x}^{\boldsymbol{b}} \in G(J)} \mathcal{B}_{\boldsymbol{x}^{\boldsymbol{a}} \wedge \neg \boldsymbol{b}}$$

Non-monotone: Cannot be expressed using only $\mathcal{B}_I$ and $\mathcal{B}_J$. Requires expansion over the generator sets.

Information from $\mathcal{B}_l$

Information about an ideal

Standard monomials

$\boldsymbol{x^b}$ is *standard* (not in I)

$$\boldsymbol{x^b} \notin I \iff \mathcal{B}_I(\boldsymbol{b}) = 0$$

If I is 0-dimensional:

$$\dim_k(R/I) = \left|\{\boldsymbol{b} : \mathcal{B}_I(\boldsymbol{b}) = 0\}\right|$$

Count paths to $\bot$ in the BDD — no enumeration needed!

Alexander dual

$$I^\vee = \bigcap_{\boldsymbol{x^a} \in G(I)} \mathfrak{m}_{\boldsymbol{a}}$$

where $\mathfrak{m}^{\boldsymbol{a}} = \langle x_i \mid i \in \operatorname{supp}(\boldsymbol{x^a})\rangle$.

$$\boldsymbol{x^a} \in I^\vee \iff \forall \boldsymbol{x^b} \in G(I), \operatorname{supp}(\boldsymbol{x^a}) \cap \operatorname{supp}(\boldsymbol{x^b}) \neq \emptyset$$

Computing $G(I^\vee)$ = CNF $\to$ DNF
= *monotone dualization*

Binary Decision Diagrams

Binary Decision Diagrams (BDDs)

Shannon's decomposition:

$$f(\boldsymbol{x}) = \left(x_i \wedge f_{x_i=1}\right) \vee \left(\bar{x}_i \wedge f_{x_i=0}\right)$$

Applied recursively → **directed acyclic graph**:

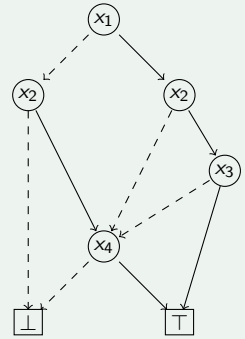
- **Root node**: top node
- **Branch nodes**: labeled by name or index
  $j = V(\textcircled{j})$
  LO (dashed, $x_j = 0$) / HI (solid, $x_j = 1$)
- **Terminal nodes**: $\top$ (TRUE) / $\bot$ (FALSE)

ROBDD = canonical form

Reduced (no redundant nodes)
+ **Ordered** (fixed variable order)
→ *canonical representation* + node sharing

Example: $I = \langle x_1x_4, x_2x_4, x_1x_2x_3 \rangle$

$\mathcal{B}_I = (x_1 \wedge x_4) \vee (x_2 \wedge x_4) \vee (x_1 \wedge x_2 \wedge x_3)$



Implementation & Experiments

CoCoALib

Free C++ library for commutative algebra. Exception-safe, thread-safe, C++14, open-source.

TeDDy

Templated Decision Diagram library. BDD & MDD support, high-performance C++, open-source.

Macaulay2

Computer algebra system for algebraic geometry. Used both for ideal generation and as independent benchmark.

Code & experiments: `github.com/lamorer/MonomialIdeal-BDD`

Results: Intersection

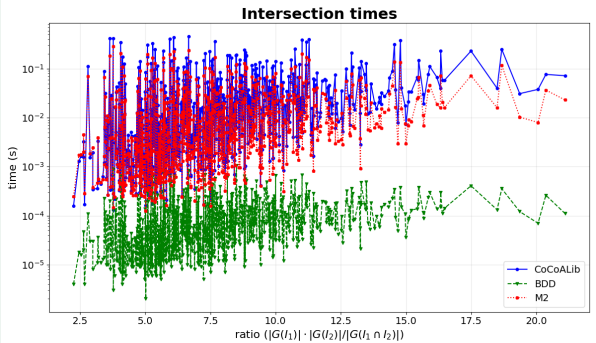
Key result

BDD is **2–3 orders of magnitude** faster than CoCoALib and Macaulay2,
stable below 10^{-3} s.

Why?

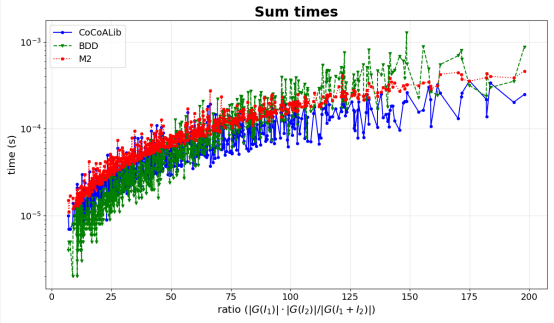
- **Algebraic overhead:** Minimising huge non-minimal generator sets wastes time.
- **Node sharing:** ROBDDs (TeDDy) share memory for identical sub-functions.

Intersection times



Results: Sum

Sum times



- Sum generates *less redundancy* in the algebraic approach
- The cost of building the BDD structure becomes comparable to symbolic minimization
- All three frameworks converge between 10^{-5} and 10^{-3} s

Stability test: staircase ideals

Ideals with many generators in the intersection but few in the sum.

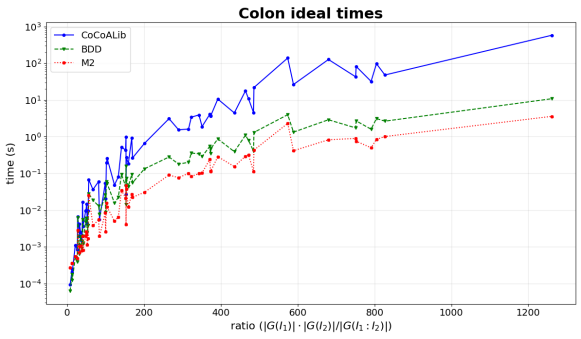
CoCoALib & M2: gap between sum/intersection.

BDD: essentially no gap.

Results: Colon ideal, Standard monomials & Alexander dual

Operation	Performance	Reason
Colon $I : J$	M2 **best**; BDD middle; CoCoALib slowest	Colon not native in BDD: iterate over $G(J)$; M2 has mature algebraic algorithms
Standard monomials	BDD 2–4 orders faster	Count paths to $\perp$ *without enumeration*; algebraic methods enumerate linearly
Alexander dual	BDD consistently best, highly stable	Canonical BDD prevents growth of intermediate generators typical in symbolic methods

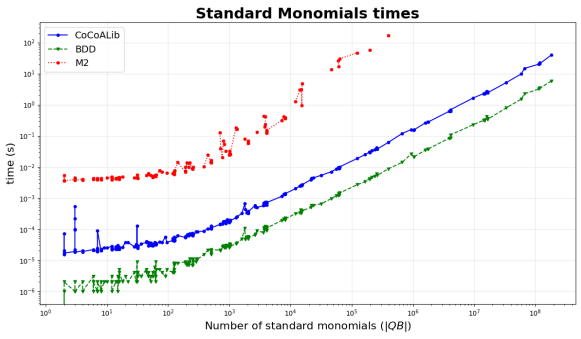
Results



Summary pattern

The BDD-based approach offers an **effective** and **scalable** alternative for the manipulation of square-free monomial ideals, particularly when the algebraic operations involve combinatorial complexity.

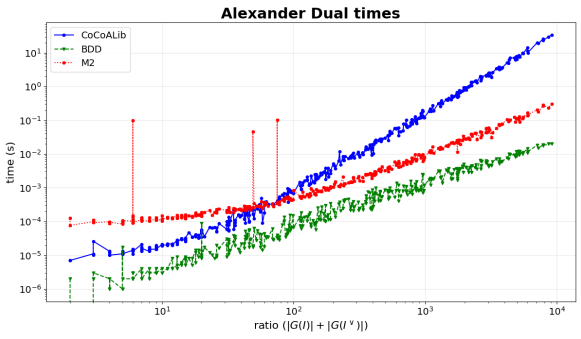
Results



Summary pattern

The BDD-based approach offers an **effective** and **scalable** alternative for the manipulation of square-free monomial ideals, particularly when the algebraic operations involve combinatorial complexity.

Results



Summary pattern

The BDD-based approach offers an **effective** and **scalable** alternative for the manipulation of square-free monomial ideals, particularly when the algebraic operations involve combinatorial complexity.

Conclusions & Future Work

Conclusions

Main take-away

Representing a square-free monomial ideal as a monotone Boolean function and using BDDs as the data structure gives a **practical, measurably faster** engine for several key operations — not just a theoretical curiosity.

Clear advantages:

- ✓ Intersection
- ✓ Standard monomials
- ✓ Alexander dual
- ✓ Stability across op. types

Current limitations:

- – Sum — competitive, not dominant
- ✗ Colon — Macaulay2 wins
- BDD size: *worst case* exponential in n (open theoretical question)
- Variable ordering matters

The Boolean engine **complements** CoCoALib / Macaulay2;
it does not replace them.

Future work

1. **More Boolean algorithms**
   Efficient implementations of further monomial operations via Boolean ops
2. **Algebraic Normal Form (ANF)**
   Zhegalkin polynomials as an alternative Boolean representation
3. **Primary / irreducible decompositions**
   I prime $\iff$ each clause of $\mathcal{B}_I$ has exactly one literal
4. **Non-square-free ideals**
   via polarization + MDD (Multi-valued Decision Diagrams)
5. **Comparison of BDD engines**
   BuDDy, CUDD, Sylvan vs. TeDDy

Code & experiments

`github.com/lamorer/MonomialIdeal-BDD`

All ideals, benchmarks, and plotting scripts are available.

Funding

This work is partially supported by grant:
PID2024-157733NBI00
MCIN/AEI/10.13039/501100011033 /FEDER EU.

Thank you!

lamorer@unirioja.es
esaenz-d@unirioja.es

github.com/lamorer/MonomialIdeal-BDD

Questions welcome!