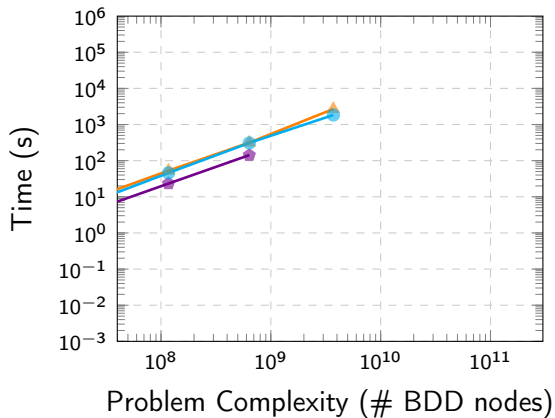


Predicting Memory Demands of BDD Operations using Maximum Graph Cuts

Steffan Christ Sølvesten and Jaco van de Pol

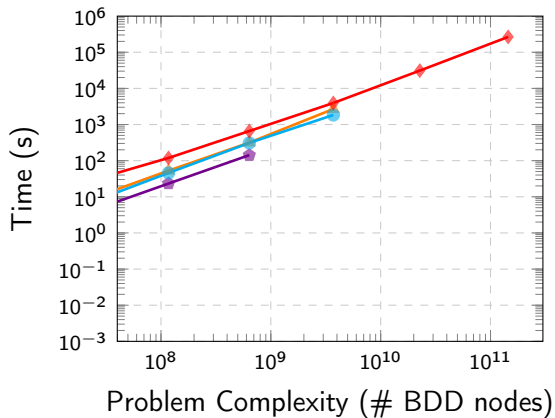
ATVA 2023





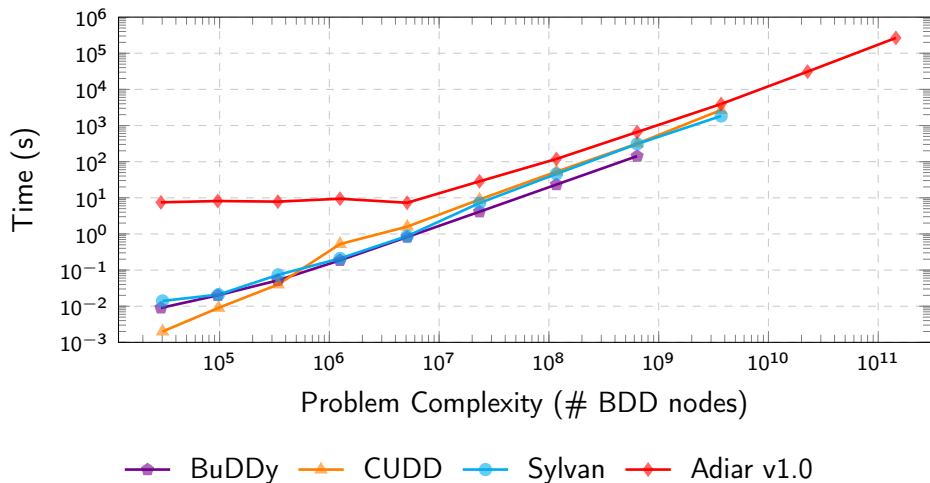
—■— BuDDy —▲— CUDD —●— Sylvan —◆— Adiar v1.0

Running Time to solve N -Queens problems.

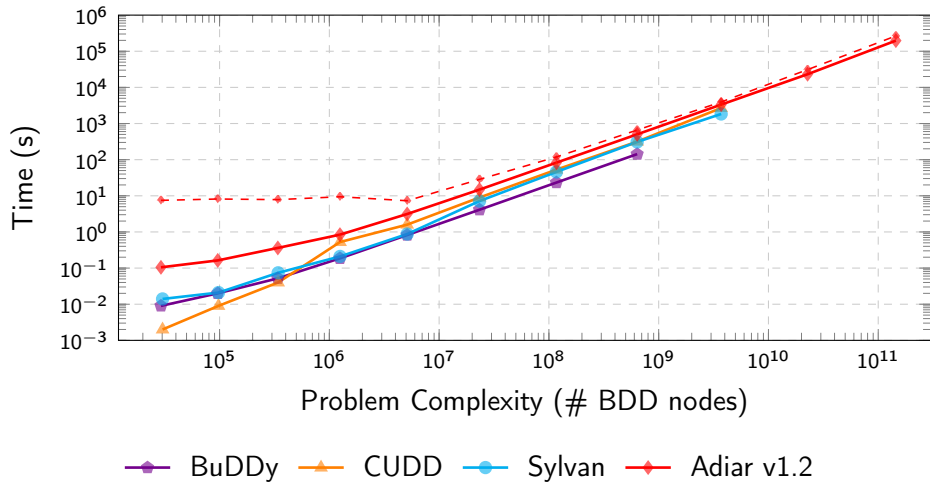


—◆— BuDDy —▲— CUDD —●— Sylvan —◆— Adiar v1.0

Running Time to solve N -Queens problems.



Running Time to solve N -Queens problems.



Running Time to solve N -Queens problems.













i -level cut



i -level cut



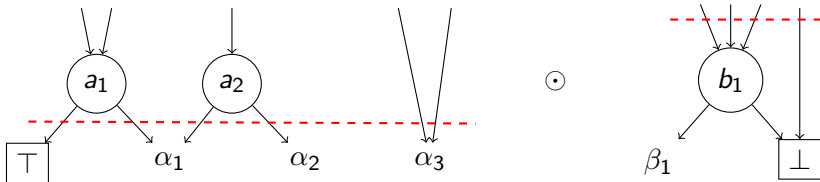
Lemma (Sølvsten, Van de Pol 2023)
The maximum i -level cut problem is in P for $i \in \{1, 2\}$.

Theorem (Lampis, Kaouri, Mitsou 2011)
The maximum i -level cut problem is NP-complete for $i \geq 4$.

Theorem (Sølvsten, Van de Pol 2023)

Given maximum 2-level cuts size C_f for f and C_g for g , the maximum 2-level cut for $f \odot g$ is less than or equal to $C_f \cdot C_g$.

Proof.



□

Theorem (Sølvsten, Van de Pol 2023)

Given maximum 2-level cuts size C_f for f and C_g for g , the maximum 2-level cut for $f \odot g$ is less than or equal to $C_f \cdot C_g$.

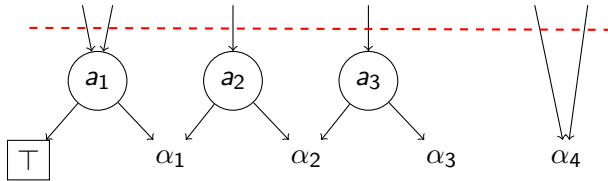
Proof.



Lemma (Sølvsten, Van de Pol 2023)

The maximum 2-level cut for f is at most $\frac{3}{2}$ larger than its maximum 1-level cut.

Proof.

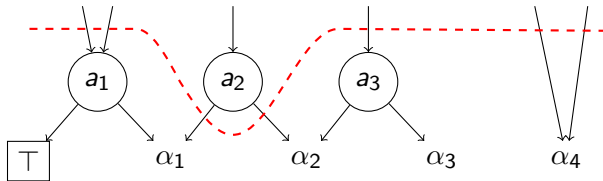


□

Lemma (Sølvsten, Van de Pol 2023)

The maximum 2-level cut for f is at most $\frac{3}{2}$ larger than its maximum 1-level cut.

Proof.



□

Lemma (Sølvsten, Van de Pol 2023)

The maximum 2-level cut for f is at most $\frac{3}{2}$ larger than its maximum 1-level cut.

Proof.



□



Overhead

Precision

1-level cut	:	1.0%	69.2%
2-level cut	:	3.3%	86.3%

Possible to process a
1.1 GiB BDD
with only
128 MiB Memory



Running Time

Adiar v1.0 : 56.5 hours

Verification of the 15 smallest EPFL circuits.



Running Time

Adiar v1.0 : 56.5 hours

Adiar v1.2 : 4.0 hours (−93%)¹

Verification of the 15 smallest EPFL circuits.

¹ 52.1 of these hours were saved on just verifying the `sin` circuit alone.

Steffan Christ Sølvsten

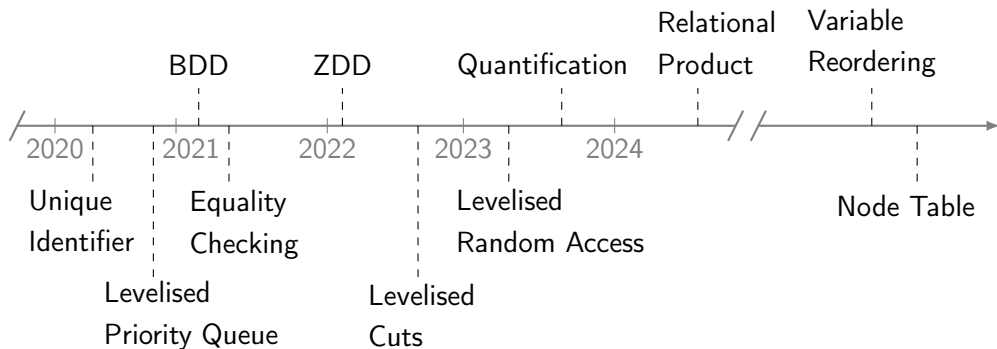
✉ soelvsten@cs.au.dk














🌐 ssoelvsten.github.io

Adiar

🔗 github.com/ssoelvsten/adiar

📖 ssoelvsten.github.io/adiar



					
	Sufficient?	Overhead	Memory ²	Disk R/W	Transition Cost
DF ▶ Adiar ( ▶ )	✗	3×	—	2×	—
DF Adiar ( )	✓	—	3×	2×	—
DF → Adiar 1.0	✗ ¹	—	—	—	$\Omega(N \log N)$
State Pattern ( → )	✓ ⁴	$\sim 20\%$ ³	2×	—	$\Omega(N)$
<i>i</i> -level cut ( / )	✓ ⁴	1%	—	—	—

Comparison of possible solutions.

¹There can be a gap between when depth-first runs out of memory and Adiar 1.0 has no overhead.

²Decreasing the memory dedicated to an external memory data structure impacts its performance.

³Runtime polymorphism adds a 20% to 30% overhead [Stroustrup].

⁴This solves the gap¹; a *non-trivial* integration with depth-first algorithms can cover tiny cases.



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Priority Queue: Q_{count} :

[

]



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Priority Queue: Q_{count} :

[

]



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Priority Queue: Q_{count} :

[$((0,0) \xrightarrow{\top} (1,0), 1)$,
 $((0,0) \xrightarrow{\perp} (2,0), 1)$,

]



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
$(1, 0)$	0	0

Priority Queue: Q_{count} :

[$((0, 0) \xrightarrow{\top} (1, 0), 1)$,
 $((0, 0) \xrightarrow{\perp} (2, 0), 1)$,

]



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
$(1, 0)$	0	0

Priority Queue: Q_{count} :

[$((0, 0) \xrightarrow{\top} (1, 0), 1)$,
 $((0, 0) \xrightarrow{\perp} (2, 0), 1)$,

]



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
$(1, 0)$	1	0

Priority Queue: Q_{count} :

[
 $((0, 0) \xrightarrow{\perp} (2, 0), 1)$,
]

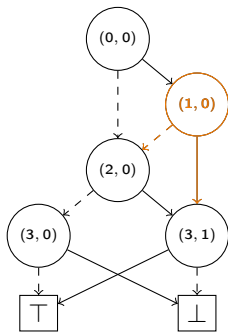


(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
$(1, 0)$	1	0

Priority Queue: Q_{count} :

[
 $((0, 0) \xrightarrow{\perp} (2, 0), 1)$,
 $((1, 0) \xrightarrow{\perp} (2, 0), 1)$,
 $((1, 0) \xrightarrow{\top} (3, 1), 1)$,
]



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
$(2, 0)$	0	0

Priority Queue: Q_{count} :

[
 $((0, 0) \xrightarrow{\perp} (2, 0), 1)$,
 $((1, 0) \xrightarrow{\perp} (2, 0), 1)$,
 $((1, 0) \xrightarrow{\top} (3, 1), 1)$,
]



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
(2, 0)	0	0

Priority Queue: Q_{count} :

[

$((0, 0) \xrightarrow{\perp} (2, 0), 1)$,
$((1, 0) \xrightarrow{\perp} (2, 0), 1)$,
$((1, 0) \xrightarrow{\top} (3, 1), 1)$,

]



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
(2, 0)	1	0

Priority Queue: Q_{count} :

[
 $((1, 0) \xrightarrow{\perp} (2, 0), 1)$,
 $((1, 0) \xrightarrow{\top} (3, 1), 1)$,
]



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
$(2, 0)$	2	0

Priority Queue: Q_{count} :

[

$((1, 0) \xrightarrow{\top} (3, 1), 1)$,
]



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
(2, 0)	2	0

Priority Queue: Q_{count} :

[

$((2, 0) \xrightarrow{\perp} (3, 0), 2)$,
$((1, 0) \xrightarrow{\top} (3, 1), 1)$,
$((2, 0) \xrightarrow{\top} (3, 1), 2)$]



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
(3, 0)	0	0

Priority Queue: Q_{count} :

[

$((2, 0) \xrightarrow{\perp} (3, 0), \quad 2)$,
$((1, 0) \xrightarrow{\top} (3, 1), \quad 1)$,
$((2, 0) \xrightarrow{\top} (3, 1), \quad 2)$]



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
(3, 0)	0	0

Priority Queue: Q_{count} :

[

$((2, 0) \xrightarrow{\perp} (3, 0), 2)$,
 $((1, 0) \xrightarrow{\top} (3, 1), 1)$,
 $((2, 0) \xrightarrow{\top} (3, 1), 2)$]



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
(3, 0)	2	0

Priority Queue: Q_{count} :

[

$((1, 0) \xrightarrow{\top} (3, 1), 1)$,
 $((2, 0) \xrightarrow{\top} (3, 1), 2)$]



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
(3, 0)	2	2

Priority Queue: Q_{count} :

[

$((1, 0) \xrightarrow{T} (3, 1), 1)$,
 $((2, 0) \xrightarrow{T} (3, 1), 2)$]



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
(3, 1)	0	2

Priority Queue: Q_{count} :

[

$((1, 0) \xrightarrow{T} (3, 1), 1)$,
 $((2, 0) \xrightarrow{T} (3, 1), 2)$]



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
(3, 1)	0	2

Priority Queue: Q_{count} :

[

$((1, 0) \xrightarrow{T} (3, 1), 1)$,
 $((2, 0) \xrightarrow{T} (3, 1), 2)$]



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
$(3, 1)$	1	2

Priority Queue: Q_{count} :

[

$((2, 0) \xrightarrow{\top} (3, 1), \quad 2) \quad]$



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
(3, 1)	3	2

Priority Queue: Q_{count} :

[

]



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek
(3, 1)

Sum
3

Result
5

Priority Queue: Q_{count} :

[

]



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Result
5

Priority Queue: Q_{count} :

[

]

Steffan Christ Sølvesten

✉ soelvsten@cs.au.dk

🌐 ssoelvsten.github.io

Adiar

🔗 github.com/ssoelvsten/adiar

📖 ssoelvsten.github.io/adiar