Conflict Analysis with Minimal Cuts

Nikita Gaevoy

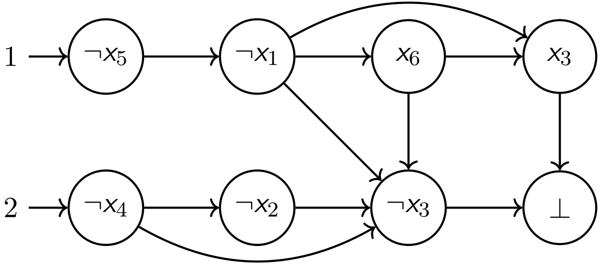
Technion

April 7, 2024

The problem

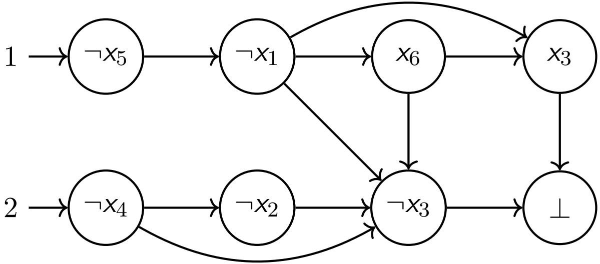
- The goal is study how using different conflict analysis schemes affect the performance of a CDCL solver.
- The work is based on the paper:
  Lintao Zhang et al. "Efficient conflict driven learning in a Boolean satisfiability solver". In: *IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No.01CH37281)*. 2001, pp. 279–285. DOI: 10.1109/ICCAD.2001.968634

The idea



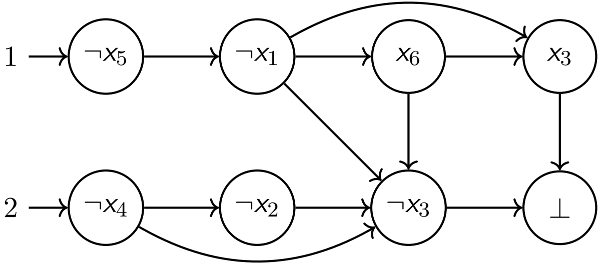
- What is the conflict clause?
- It is a cut between the decisions and the contradiction.
- We want to have a dominator on the last decision level to propagate a unit-clause immediately (UIP).

The idea



- What is the conflict clause?
- It is a cut between the decisions and the contradiction.
- We want to have a dominator on the last decision level to propagate a unit-clause immediately (UIP).
- Among all suitable cuts, we should prioritize closest to the contradiction.

The idea



- What is the conflict clause?
- It is a cut between the decisions and the contradiction.
- We want to have a dominator on the last decision level to propagate a unit-clause immediately (UIP).
- Among all suitable cuts, we should prioritize closest to the contradiction.
- What about minimality?

Previous results

Decision Strategy		Microprocessor Formal Verification[19]			Bounded Model Checking [18]			
		fvp-unsat.1.0(4)	sss.1.0(48)	sss.1.0a(9)	barrel (8)	longmult(16)	queueinvar(10)	satplan(20)
VSIDS	1uip	532.8	24.56	10.63	1012.62	2887.11	6.58	39.34
	2uip	746.87	27.32	16.96	641.64	2734.57	16.37	41.37
	3uip	2151.26	69.12	47.66	656.56	2946.73	19.44	57.16
	alluip	0.68(3)	1746.27(2)	79.09	1081.57(1)	11160.25	18.07	71.86
	rel_sat	2034.09	193.93	82.51	292.33(1)	5719.73	14.4	96.61
	mincut	1612.74(2)	2293.18	11.15	4119.34(1)	7321.69(5)	100.94	43.84
	grasp	2224.44	94.64	33.99	654.54	6196.82	97.82	309.03
	decision*	0(4)	1022.57(17)	227.37(3)	541.96(2)	1421.35(4)	334.39(4)	193.36(3)
FIXED	1uip_f	11.36(3)	15307.13(3)	2997.37	281.48(1)	3141.8	817.07(5)	18(2)
	2uip_f	23.07(3)	18844.51(3)	2646.99	344.34(1)	4279.07	777.2(5)	29.51(2)
	3uip_f	40.75(3)	3985.23(9)	4109.86	432.77(1)	4440.49	860.3(5)	37.62(2)
	alluip_f	0(4)	4063.44(25)	0.28(8)	699.42(1)	11375.32	2025.08(5)	1136.44(2)
	relsat_f	80.94(3)	3114.83(16)	4261.25(4)	293.09(1)	4396.73	478.37(6)	3323.71(3)
	mincut_f	0(4)	5619.4(15)	590.79(4)	3408.28(1)	5232.69(5)	3206.36(4)	373.78(2)
	grasp_f	22.51(3)	6497.99(8)	3382.47	326.46(1)	5597.1	792.12(5)	149.8(2)
	decision_f*	0(4)	415.76(42)	40.57(8)	479.61(1)	1006.58(6)	1.27(8)	600.87(10)

* timeout set to 600s instead of 3600s

Table 1. Run time for different learning schemes

Previous results

Decision Strategy		Microprocessor Formal Verification[19]			Bounded Model Checking [18]			
		fvp-unsat.1.0(4)	sss.1.0(48)	sss.1.0a(9)	barrel (8)	longmult(16)	queueinvar(10)	satplan(20)
VSIDS	1uip	532.8	24.56	10.63	1012.62	2887.11	6.58	39.34
	2uip	746.87	27.32	16.96	641.64	2734.57	16.37	41.37
	3uip	2151.26	69.12	47.66	656.56	2946.73	19.44	57.16
	alluip	0.68(3)	1746.27(2)	79.09	1081.57(1)	11160.25	18.07	71.86
	rel_sat	2034.09	193.93	82.51	292.33(1)	5719.73	14.4	96.61
	mincut	1612.74(2)	2293.18	11.15	4119.34(1)	7321.69(5)	100.94	43.84
	grasp	2224.44	94.64	33.99	654.54	6196.82	97.82	309.03
	decision*	0(4)	1022.57(17)	227.37(3)	541.96(2)	1421.35(4)	334.39(4)	193.36(3)
FIXED	1uip_f	11.36(3)	15307.13(3)	2997.37	281.48(1)	3141.8	817.07(5)	18(2)
	2uip_f	23.07(3)	18844.51(3)	2646.99	344.34(1)	4279.07	777.2(5)	29.51(2)
	3uip_f	40.75(3)	3985.23(9)	4109.86	432.77(1)	4440.49	860.3(5)	37.62(2)
	alluip_f	0(4)	4063.44(25)	0.28(8)	699.42(1)	11375.32	2025.08(5)	1136.44(2)
	relsat_f	80.94(3)	3114.83(16)	4261.25(4)	293.09(1)	4396.73	478.37(6)	3323.71(3)
	mincut_f	0(4)	5619.4(15)	590.79(4)	3408.28(1)	5232.69(5)	3206.36(4)	373.78(2)
	grasp_f	22.51(3)	6497.99(8)	3382.47	326.46(1)	5597.1	792.12(5)	149.8(2)
	decision_f*	0(4)	415.76(42)	40.57(8)	479.61(1)	1006.58(6)	1.27(8)	600.87(10)

* timeout set to 600s instead of 3600s

Table 1. Run time for different learning schemes

- Most learning schemes are well-defined in terms of maximum flow.

Previous results

Decision Strategy		Microprocessor Formal Verification[19]			Bounded Model Checking [18]			
		fvp-unsat.1.0(4)	sss.1.0(48)	sss.1.0a(9)	barrel (8)	longmult(16)	queueinvar(10)	satplan(20)
VSIDS	1uip	532.8	24.56	10.63	1012.62	2887.11	6.58	39.34
	2uip	746.87	27.32	16.96	641.64	2734.57	16.37	41.37
	3uip	2151.26	69.12	47.66	656.56	2946.73	19.44	57.16
	alluip	0.68(3)	1746.27(2)	79.09	1081.57(1)	11160.25	18.07	71.86
	rel_sat	2034.09	193.93	82.51	292.33(1)	5719.73	14.4	96.61
	mincut	1612.74(2)	2293.18	11.15	4119.34(1)	7321.69(5)	100.94	43.84
	grasp	2224.44	94.64	33.99	654.54	6196.82	97.82	309.03
	decision*	0(4)	1022.57(17)	227.37(3)	541.96(2)	1421.35(4)	334.39(4)	193.36(3)
FIXED	1uip_f	11.36(3)	15307.13(3)	2997.37	281.48(1)	3141.8	817.07(5)	18(2)
	2uip_f	23.07(3)	18844.51(3)	2646.99	344.34(1)	4279.07	777.2(5)	29.51(2)
	3uip_f	40.75(3)	3985.23(9)	4109.86	432.77(1)	4440.49	860.3(5)	37.62(2)
	alluip_f	0(4)	4063.44(25)	0.28(8)	699.42(1)	11375.32	2025.08(5)	1136.44(2)
	relsat_f	80.94(3)	3114.83(16)	4261.25(4)	293.09(1)	4396.73	478.37(6)	3323.71(3)
	mincut_f	0(4)	5619.4(15)	590.79(4)	3408.28(1)	5232.69(5)	3206.36(4)	373.78(2)
	grasp_f	22.51(3)	6497.99(8)	3382.47	326.46(1)	5597.1	792.12(5)	149.8(2)
	decision_f*	0(4)	415.76(42)	40.57(8)	479.61(1)	1006.58(6)	1.27(8)	600.87(10)

* timeout set to 600s instead of 3600s

Table 1. Run time for different learning schemes

- Most learning schemes are well-defined in terms of maximum flow.
- The flow algorithm used in the paper is $\mathcal{O}(VE\log(V^2/E))$, which is probably suboptimal.

SK1Flow

- Each vertex has an excess.
- The network has exactly one sink and no sources (simulated by infinite excess).

SK1Flow

- Each vertex has an excess.
- The network has exactly one sink and no sources (simulated by infinite excess).
- The algorithm works in phases.
- In each phase, we run bfs from the sink, traversing reversed unsaturated edges and push all the excess greedily to the lower layers. It works in $\mathcal{O}(E)$.

SK1Flow

- Each vertex has an excess.
- The network has exactly one sink and no sources (simulated by infinite excess).
- The algorithm works in phases.
- In each phase, we run bfs from the sink, traversing reversed unsaturated edges and push all the excess greedily to the lower layers. It works in $\mathcal{O}(E)$.
- Repeat while something happens.

SK1Flow

- Each vertex has an excess.
- The network has exactly one sink and no sources (simulated by infinite excess).
- The algorithm works in phases.
- In each phase, we run bfs from the sink, traversing reversed unsaturated edges and push all the excess greedily to the lower layers. It works in $\mathcal{O}(E)$.
- Repeat while something happens.
- The distance to the sink can only grow. Thus, there are at most $\mathcal{O}(V^2)$ phases.

SK1Flow

- Each vertex has an excess.
- The network has exactly one sink and no sources (simulated by infinite excess).
- The algorithm works in phases.
- In each phase, we run bfs from the sink, traversing reversed unsaturated edges and push all the excess greedily to the lower layers. It works in $\mathcal{O}(E)$.
- Repeat while something happens.
- The distance to the sink can only grow. Thus, there are at most $\mathcal{O}(V^2)$ phases.
- One phase could be optimized to $\mathcal{O}(V + \text{useful work})$.

SK1Flow

- Each vertex has an excess.
- The network has exactly one sink and no sources (simulated by infinite excess).
- The algorithm works in phases.
- In each phase, we run bfs from the sink, traversing reversed unsaturated edges and push all the excess greedily to the lower layers. It works in $\mathcal{O}(E)$.
- Repeat while something happens.
- The distance to the sink can only grow. Thus, there are at most $\mathcal{O}(V^2)$ phases.
- One phase could be optimized to $\mathcal{O}(V + \text{useful work})$.
- Performs well on practice. Although, the best* theoretical bound is $\mathcal{O}(V^3)$.

SK1Flow

- Each vertex has an excess.
- The network has exactly one sink and no sources (simulated by infinite excess).
- The algorithm works in phases.
- In each phase, we run bfs from the sink, traversing reversed unsaturated edges and push all the excess greedily to the lower layers. It works in $\mathcal{O}(E)$.
- Repeat while something happens.
- The distance to the sink can only grow. Thus, there are at most $\mathcal{O}(V^2)$ phases.
- One phase could be optimized to $\mathcal{O}(V + \text{useful work})$.
- Performs well on practice. Although, the best* theoretical bound is $\mathcal{O}(V^3)$.
- Can reuse a part of the graph after backtracking*.

Our plan

1. Implement a CDCL solver. ✓
2. Implement the flow algorithm. ✓
3. Befriend them.
4. Simulate all interesting conflict analysis schemes in terms of the flow.
5. Run the benchmarks from the paper.
6. Publish to the repository.