ISE, The Ohio State University

Parallelized Conflict Graph Cut Generation

Yongzheng Dai, Chen Chen

GitHub Link

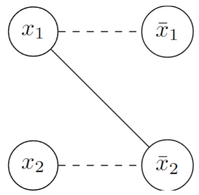
Preprint Link

Abstract

We develop efficient parallel conflict graph management: conflict detection; maximal clique generation; clique extension; and clique merging. We leverage parallel computing to intensify computational effort on the conflict graph, thereby generating a much larger pool of cutting planes than what can be practically achieved in serial. Computational experiments demonstrate our parallel method led to substantial reductions in total MIP solve time.

Conflict Graph (CG)

A conflict [1] is an infeasible assignment of values to binary variables, e.g., $x_1 = 1, x_2 = 0$, (or $\overline{x_2} = 1 - x_2 = 1$).



CG can be used to guide branching decisions, fix variable values, **generate cuts**, etc.

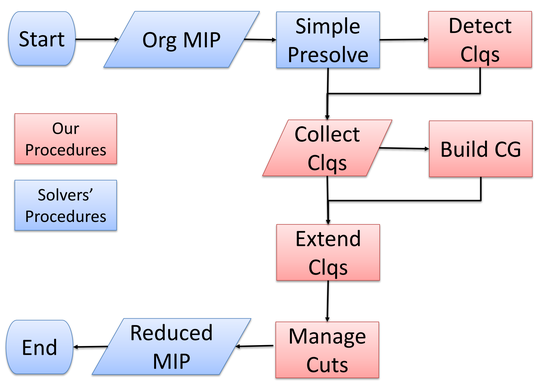
Suppose m constraints (or cliques), n binary variables, k threads, and a probability p that one binary variable appearing in one clique in later complexity results.

Our focus:

We modify the algorithm from [2] to generate even more cuts, set computation limits, and deploy cuts management:

- ~Break-even in serial
- Substantial advantage in parallel

Our Procedure



Clique Detection

Set Pack (Clique): $\boldsymbol{x_1 + x_2 + x_3 \leq 1}$

Knapsack: $\boldsymbol{2x_1 + 4x_2 + 3x_3 \leq 5.5}$

Detect following cliques:

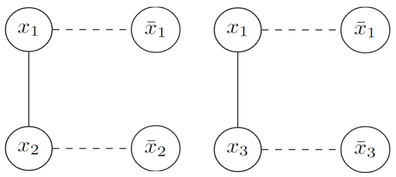
- $x_1 + x_2 \leq 1$,
- $x_2 + x_3 \leq 1$.

Randomly shuffle all knapsacks and assign them to different threads evenly. The parallel algorithm is $O\left(\frac{mn \log n}{k}\right)$ on average, and $O\left(\frac{mn^2}{k}\right)$ at the worst case.

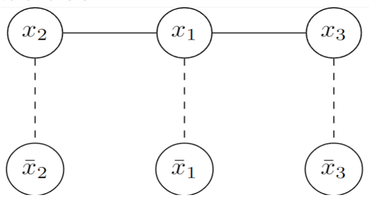
CG Construction

$$x_1 + x_2 \leq 1,$$
$$x_1 + x_3 \leq 1.$$

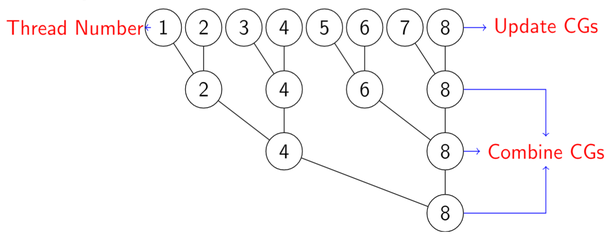
Build the CG with two threads:



Combine to one CG.



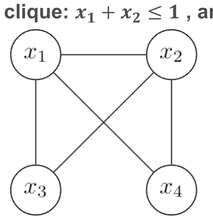
Binary Combination:



Using sparse adjacency matrix to cache CG, and set Max_Clq_Length as 3000. Randomly shuffle all cliques and assign them to different threads evenly. The parallel algorithm is $O\left(\frac{mn^2p^2}{k} + n^2 \log k\right)$ on average, and $O\left(\frac{mn^2}{k} + n^2 \log k\right)$ at the worst case.

Clique Extension

Suppose we have a clique: $x_1 + x_2 \leq 1$, and a CG:



Then we can get:

- $x_1 + x_2 + x_3 \leq 1$, (Brito's method stops here)
- $x_1 + x_2 + x_4 \leq 1$.

Clique extension is time-consuming ($O(n^2)$); thus set Max_Nonzero_Term as 10^6. Randomly shuffle all cliques and assign them to different threads evenly. The worst-case complexity of the parallel algorithm is $O(\frac{mn^2}{k})$.

Clique Merging

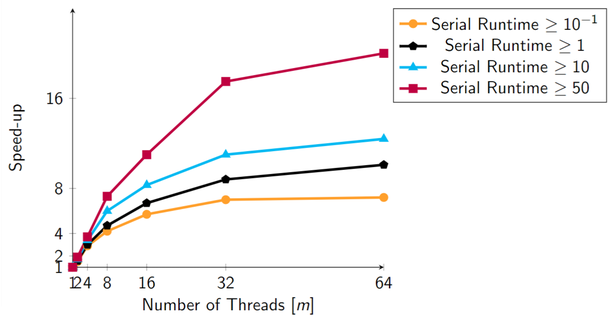
$x_1 + x_2 + x_3 \leq 1$ dominates $x_1 + x_2 \leq 1$, then combine them. The parallel complexity is $O\left(\frac{m^2n}{k}\right)$.

Cut Management

- We add some generated cuts to the root node and place others as user cuts [4].
- Details can be found in Sec. 2.5 in our preprint [3].

Parallel Performance

- Implemented in Julia and a 32-core CPU.
- 173 impacted cases from MIPLIB 2017 Benchmark Collection, and 163 of them can be completed in 600s.
- 108 of them take more than 0.1s.



Gurobi Solver Time Comparison

- 99/173 cases are compared, and 3 of them have memory issues (clique size).
- 8 cases in which Gurobi 11.0.1 cannot solve the original MIPs within 3600 seconds; adding our CG procedure solves 1 of them.
- Faster or slower implies the runtime difference > 5%
- Org Time: Gurobi solver time for the original model
- Reduced Time: Gurobi solver time for the reduced mode
- Runtime Comp.:

  Reduced Time / Org Time.

Bracket	Number of Cases	Runtime Comp.	Nodes Comp.	Faster	Slower
All	88	0.88	0.60	42	24
$\geq$ 1 sec	80	0.83	0.57	42	22
$\geq$ 10 sec	62	0.76	0.46	37	16
$\geq$ 100 sec	35	0.69	0.47	25	8
$\geq$ 1000 sec	7	0.60	0.35	7	0

Table1: Ignore CG procedure runtime

- nThread Time: CG procedure time with n threads
- Runtime Comp.:

  (Reduced Time + nThread Time)/Org Time

Bracket	Runtime Comp.	Faster	Slower
1 Thread	1.10	36	41
2 Threads	1.04	38	40
4 Threads	0.96	38	39
8 Threads	0.93	38	36
16 Threads	0.91	39	35
32 Threads	0.90	39	34
64 Threads	0.90	39	34

Table 2: Consider CG procedure runtime

BIBLIOGRAPHY

1. Alper Atamturk, George L Nemhauser, and Martin WP Savelsbergh. Conflict graphs in solving integer programming problems. European Journal of Operational Research, 121(1):40–55, 2000
2. Brito SS, Santos HG. Preprocessing and cutting planes with conflict graphs. Computers & Operations Research. 128:105176, 2021.
3. Dai Y, Chen C. Parallelized Conflict Graph Cut Generation. arXiv preprint arXiv:2311.03706. 2023.
4. Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL https://www.gurobi.com
5. Tobias Achterberg, Robert E Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. Presolve reductions in mixed integer programming. INFORMS Journal on Computing, 32(2):473–506, 2020

ACKNOWLEDGEMENTS

This work was funded by the Office of Naval Research under grant N00014-23-1-2632.

THE OHIO STATE UNIVERSITY