# Solving Hard Combinatorial Problems in Parallel Using Lift-and-Project Preconditioning

1st Bogdan Zavalnij
*Department of Combinatorics and its Applications*
*HUN-REN Alfréd Rényi Institute of Mathematics*
Budapest, Hungary
bogdan@renyi.hu

***Abstract*—The original Lift and Project method was a useful tool for solving some ILP problems. Encouraged by this idea we propose a new method for preconditioning graphs for the $k$-clique problem. After a blow-up phase some standard preconditioning applied, then the results projected back to the original graph. Because the blow-up may produce a graph too big we deal with a series of smaller blown-up graphs that can be processed independently and thus parallelly. Numerical experiments back up the usefulness of this approach.**

## I. Introduction

In the present paper we would like to introduce a novel approach for preconditioning hard combinatorial problems. As an example we will use the NP-hard maximum clique and NP-complete $k$-clique problem [5], [8], [13], the later asking if in a given graph a clique of size $k$ is present. Equivalent to the first problem are the maximum independent set problem (MIS) or the maximum vertex cover problem (MVC).

The approach for solving these problems is twofold. First, one can use extended preconditioning methods, called also kernelization, to make the problem easier. Second, there are specialized solvers for these problems. For some time it seemed that kernelization is effective for huge but easy problems, while specialized solvers are better for the small hard instances [1]. In 2019 The MVC problem was the problem chosen for the PACE2019 competition, and the results clearly showed, that the above two approaches need to be used together, combined [15].

Although the usual kernelization methods proved useful for such problems [14], these are still preconditioning methods tailored for huge and easy graphs, and specially designed approach of preconditioning is scarce for hard instances. In our previous work we took some steps into this direction [16] that proved crucial for solving some hard problems modeled by graph and solved by finding a $k$-clique in that graph [17]. The main idea behind this approach, if one would believe in $P \neq NP$, that the exact solution of the NP problem is extremely slow, probably taking exponential on near exponential time compared to input. While preconditioning, even the most sophisticated and time consuming, would take polynomial time transforming the original problem into a faster solvable one for the exact solver gaining on the exponential time of the solver. Indeed, the proposed preconditioning methods are time consuming but still able to reduce most of the problems so the added running time of the preconditioner and the solver is reduced compared to the original solution time.

This project was supported by the National Research, Development and Innovation Office – NKFIH Fund No. SNN–135643

The present work aims to extend such method and make use of even more computational resources for preconditioning. As the original preconditioning in some cases runs for hours or even days, this calls for using an HPC environment.

One would raise the question, how is this approach compared to parallelization of the underlying solver for the clique problem. There are very few of such approach, for example [3], [12], and those use shared memory approach, have specific problems of scaling, and show problematic behavior as superlinear speedup, see [4], [9]. Programs using distributed parallelization even more rare [18]. One one hand, compared to the complex parallelization methods for clique solvers the parallelization of the preconditioning program is mostly straightforward. On the other hand one would wonder if the two approaches even comparable, as the algorithms differ radically.

The structure of the present paper as follows. First, we breefly introduce the Lift and Project method. Second, we describe the way we would like to use it in our case. Third, we will show some limited numerical examples and draw the conclusions.

## II. Lift and Project

### A. The original lift-and-project method

The original Lift and Project Method [2], [7], [11] is about integer programming formulation of some combinatorial problem. It advises us to write the problem in quadratic form, and then replace products with novel linear variables, thus lifting the dimensionality. After adding novel constrains (like symmetry breaking), one solves the problem and projects back to the original lower dimensional space.

In essence, this method tells us to increase the size of the problem (lift), where some structure can be exploited. Add new constrains or symmetries, and finally reduce it back (project)

This approach also proved useful is different areas outside combinatorial optimization, such as studying and optimizing nonlinear control and decision system [10].

### B. A Lift and Project method for preconditioning clique problems

Our goal is to apply the abovementioned approach for preconditioning. In our case we will use graph transformations that preserve the clique number property, but instead of reducing the graph we will blow it up. It was already shown, that a blow-up can be useful as following reductions may be more effective in the end [6]. But in our case we are not interested to reduce the blown-up problem below the original graph, we are only interested in the fact, that in this new dimension more effective preconditioning may take place.

The crucial step is, that after performing the preconditioning on the blown-up graph, we project back the results into the original graph, thus weakening it for the solver or further preconditioning steps.

### C. Preconditioning

To keep the present paper more understandable we limit our preconditioning methods to some more simple graph transformations. These are those preconditioning steps that will be used in our discussion. For more detailed preconditioning see [16] where we take our following preconditioning methods from. We omit proofs as well, the reader may find them in the abovementioned citation, although these are simple enough that can be seen in straightforward manner. Basically we are aiming at deleting nodes or edges from the graph, such that no $k$-clique is deleted at the same time. So the if we search for a $k$-clique in the original or in the transformed graph, we will get the same "present" or "not present" answer for both.

For the sake of simplicity and because of the later following numerical examples we consider the graph be $k$-partite, and the problem is to decide if a $k$-clique present in this graph. More on this problem class see [16]. (For general graphs Property 1 and 3 can be applied the same way, and Property 2 and 4 needs a slight modification to be applicable.)

*Definition 1:* The color index of a node $v$ of $G$ (with respect to a legal coloring of the nodes of $G$) is the number of color classes $C_i$ that contain at least one node adjacent to $v$.

*Property 1:* If the colors index of $v$ is at most $k-2$, then $v$ can be deleted from $G$ without losing any $k$-clique.

*Property 2:* If there are two color classes of neighbors of $v$, $C'_r$ and $C'_q$, such as there is no edge between them, that is, $\forall x \in C'_r, \forall y \in C'_q, \{x,y\} \notin E$, then node $v$ can be deleted from $G$ without losing any $k$-clique.

*Definition 2:* The color index of an edge $\{u,v\}$ of $G$ (with respect to a legal coloring of the nodes of $G$) is the number of color classes $C_i$ that contain at least one node adjacent to $u$ and $v$ simultaneously.

*Property 3:* If the color index of an edge $\{u,v\}$ is less than $k-2$, then the edge $\{u,v\}$ can be deleted from $G$ when one is looking for a $k$-clique in $G$. (We do not delete the nodes $u$ or $v$.)

*Property 4:* If there are two color classes of neighbors of $\{u,v\}$, $C'_r$ and $C'_q$, such as there is no edge between them, that is, $\forall x \in C'_r, \forall y \in C'_q, \{x,y\} \notin E$, then edge $\{u,v\}$ can be deleted from $G$ without losing any $k$-clique. (We do not delete the nodes $u$ or $v$.)

### D. The Blow-up Transformation

Our goal is to lift the problem, that is blow up the graph up for subject to more effective preconditioning. Possibly this can be performed by different methods, here we propose a simple one, also from [16]. This rule takes two color classes and contracts the edges running between them into new nodes. The original two color class deleted afterwards.

Let $G=(V,E)$ be a finite simple graph. We assume that the nodes of $G$ are legally colored using $k$ colors and we are looking for a $k$-clique in $G$. We construct a new graph $G'$ from $G$. Let $C_1$, $C_2$ be two distinct color classes of the nodes of $G$. Let $e_1,\ldots,e_s$ be all the edges of $G$ such that the end nodes of the edges are all in $C_1 \cup C_2$. We delete each element of the set $C_1 \cup C_2$ from $G$ and we add new nodes $u_1,\ldots,u_s$ to $G$ to get $G'$. If $e_i=\{x_i,y_i\}$, then we connect the node $u_i$ to each element in $N(x_i) \cap N(y_i)$ with an edge. We color the nodes of $G'$. The nodes in the set $V \setminus (C_1 \cup C_2)$ will inherit the colors from the coloring of the nodes of $G$. We assign a new color $c$ to the new nodes $u_1,\ldots,u_s$. In this way we get a legal coloring of the nodes of the graph $G'$ using $k-1$ colors.

*Lemma 1:* Using the notations above the equation $\omega(G)=k$ is equivalent to the equation $\omega(G')=k-1$.

One can choose different color classes to contract them. In our original work we aimed for the one that reduced the graph in the end, or if it increased the size, the growth was small. For a blow-up we aim at we shall use all possible $\binom{k}{2}$ pairs of color classes, that is transform all edges of the original graph $G$ into nodes of $G'$

### E. Parallelization

Clearly, this graph would be too big, possibly even not fitting into the memory, and also the clique size to search for will be increased to $\binom{k}{2}$ which is also problematic. So one would divide this problem into smaller ones and use a parallel approach to tackle this problem. Our solution is to make not one, but a series of blown-up graphs, each transforming edges between different pairs of color classes. A Round Robbin Tournament scheduling was used, that is $k-1$ graphs were build each using $k/2$ pairs of color classes, to transform the edges inside a pair into nodes of a blown-up graph where the clique size to search for is also $k/2$. By this transformation each edge of the original graph will appear as a node in one graph out of the series of the blown-up graphs.

Note, that a node in this blown-up graph is represents an edge in the original graph. The edges in the blown-up graph are $K_4$ sub-graphs of the original graph. In our view this is the higher dimension we lifted the problem, and this is the reason the proposed simple reduction rules can reduce the problem more aggressively in these blown-up instances.

So next we performed the above preconditionings (see Property 1–4) to delete edges and nodes in the blown-up graphs, which nodes represent edges of the original graph, and edges represent $K_4$ sub-graphs in the original graph. The

reason behind that we used a limited rules for preconditioning, that other preconditioning methods – such as dominance – may lead to loss of $k$-cliques, and in parallel work such may lead to circular deletion, and thus to wrong answer. Obviously, with more careful and complex program one can avoid such circular deletion, but for our present work we chose to make the algorithm as simple as possible. After preconditioning as the nodes of a blown-up graph represent edges in the original graph in the project phase we use the information of such deleted nodes to delete edges in the original graph. (As the edges in the blown-up graph are $K_4$ sub-graphs of the original graph a "deletion" of such cannot be easily applied to the original graph, so we just discarded this information in the end. Again, with a more complex approach the information of such a deletion of a $K_4$ sub-graph can be routed to another worker where by this information an edge can be deleted, but again we left this for future work.)

The main goal of such parallel work is not exactly gaining speed, but to achieve preconditioning for extremely hard problem that cannot be attacked otherwise.

## III. Numerical Experiments

In the present preliminary work for our basic experiments we choose small but notoriously hard problems from [17] based on being hard for the serial solver. This obviously won't show the full potential, as in our opinion this method is more useful for even larger and harder instances. But we keep our examples simple for clear presentation. The original problem is graph coloring, namely if a given graph can be colored with $c$ colors. This problem is translated to a special $k$-partite graph, where the goal is to find a $k$-clique. If such a clique is found, then the graph in question can be colored with $c$ colors. If there is no $k$-clique present, then the graph cannot be colored by $c$ colors.

We choose 4 graphs, that were already reduced by our preconditioner which uses all possible preconditioning rules in serial manner not just the ones we presented in this paper. The auxiliary graphs for 1) if the myciel6 graph can be colored with 6 colors ($G_1$); 2) if the 1-FullIns_5 graph can be colored with 5 colors ($G_2$); 3) if the myciel7 graph can be colored with 6 colors ($G_3$); and 4) if the myciel7 graph can be colored with 7 colors ($G_4$). We used a computer with two AMD EPYC 7643 48-core processors and 1 TB memory. We turned off processor boosting, so all cores run on fixed 2.3GHz speed. All programs were written in `C++` using gcc v12.1 with the switch settings `-O3 -arch=znver3`.

The workflow was built up as follows. First, we started $k-1$ independent programs each constructing a blown-up graph according the Round-Robin Tournament scheduling contracting $k/2$ pairs of color classes. Second, these programs perform the minimal preconditioning on these graphs and save the list of deleted nodes, that is the edges to be deleted from the original graph. As the number of blown-up graphs were less then the number of cores these programs could run independently in parallel manner. Third, the full preconditioner read the list of edges to be deleted, deleted them from the original graph, and performed preconditioning on this graph. Fourth, the reduced graph is solved by the clique solver.

The auxiliary graph $G_1$ for myciel6 and color number 6 had 598 nodes, and could be solved in 1346 sec. The preconditioning of the blown-up graphs could delete 7% of the edges, and the 51 instances were solved in 50–238 seconds each. After deleting these edges from the original graph a new full preconditioner reduced the graph even more deleting 55 nodes and many edges, and the resulting graph could be solved in 992 second.

The auxiliary graph $G_2$ for 1-FullIns_5 and color number 5 had 747 nodes, and could be solved in 12180 seconds. The preconditioning of the blown-up graphs could delete 11% of the edges, and the 63 instances were solved in 228–1520 seconds each. After deleting these edges from the original graph a new full preconditioner reduced the graph so much, that the problem was solved by only using preconditioning.

The auxiliary graph $G_3$ for myciel7 and color number 6 had 1222 nodes, and could be solved in 19354 seconds. The preconditioning of the blown-up graphs could delete 9% of the edges, and the 99 instances were solved in 2000–17000 seconds each. After deleting these edges from the original graph a new full preconditioner reduced the graph even more deleting 158 nodes and many edges, and the resulting graph could be solved in 17829 second.

The auxiliary graph $G_4$ for myciel7 and color number 7 had 1576 nodes, and could not be solved in the given time limit. The preconditioning of the blown-up graphs could delete 5% of the edges, and the 103 instances were solved in 5050–25100 seconds each. After deleting these edges from the original graph a new full preconditioner reduced the graph even more deleting 140 nodes and many edges, but the solver still could not solve this instance.

We summarized the above information in Table I.

TABLE I
SIZE AND SOLUTION TIME FOR THE EXAMPLE GRAPHS

| | $G_1$ | $G_2$ | $G_3$ | $G_4$ |
|---|---|---|---|---|
| description | myciel6 6 colors | 1-FullIns_5 5 colors | myciel7 6 colors | myciel7 7 colors |
| original size (nodes) | 598 | 747 | 1222 | 1576 |
| original solution time $t_o$ (sec) | 1346 | 12180 | 19354 | nd |
| number of parallel workers | 51 | 63 | 99 | 103 |
| preconditioning time $t_p$ (sec) | 49–283 | 228–1520 | 2164–16994 | 5064–25111 |
| new size after full preconditioning | 543 | 0 | 1064 | 1436 |
| new solution time $t_n$(sec) | 992 | 0 | 17829 | nd |
| $t_o/(t_p+t_n)$ | 1.1 | 8 | 0.55 | nd |

### A. Discussion of the Results

Although chosen without any consideration apart from original solving time, the four examples proved to be interesting and show different behavior of the proposed method.

All examples but the unsolvable one showed reduction of the solution time, although in different magnitudes. One was reduced by less than 10%, one by almost 30%, and one was reduced to zero time. But of course the parallel preconditioning of the blown-up graphs also took its time. For one the parallel time was comparable to the solution itself, so we can consider it as not really useful. For the other two the parallel running time was considerably smaller than solution time.

The question of speedup can be raised at this point, and one would argue, that the number given by $t_o/(t_p + t_n)$ is just the speedup we achieved. But the author would disagree, as the algorithms, in this case the preconditioning method, differs substantially and as such they can be compared but cannot be named speedup. As for the results we can observe that for the three examples that could be solved there is one considerable faster (8 times), one runs the same time (1.1 times), and one runs slower (0.55 times).

The results are clearly show that the proposed method can be useful and can reduce the running time of the solver. We would like to emphasize that this method is designed for solving especially hard problems and for those any parallel method that can help may be useful, especially because the parallelization of the original problem is problematic. It is upon more detailed experiments to show in detail the usefulness of the proposed method, at this point we consider it promising.

## IV. Conclusions and Future Work

We applied the idea of the Lift and Project method for preconditioning graphs for the $k$-clique problem. We used a series of blown-up graphs and used the resulting information for deleting edges in the original graph. We could use 50–100 independent workers for parallelization of such preconditioning and the reduced original graph could always be solved in shorter time. Comparing the overall running time – the time for the longest preconditioning of all workers plus the time for final solution – we got mixed results. We could observe considerable decrease in overall time, no change, and also some increase as well. Our presumption is, given the polynomial time for preconditioning and possibly exponential time for clique search, that there should be a clear advantage for the proposed method after some point if we increase the size and thus the complexity of the problem. Note, that with more complex problems the number of workers we use can be even higher.

To extend our preliminary test we would like to implement this approach in MPI environment, so that information of deleting edges can reach the master process, which can perform extended preconditioning using this edge deletion information, and the information of the new reduction can be transferred back to the worker. So the blown-up preconditioning would be even more effective.

Also, the present results were achieved using the most simple reduction rules. With proper considerations other preconditioning transformations could be used and we hope for even better reduction of the graph and thus more clear advantage in solution time. In truth, it is more surprising that such results could be achieved with those simple rules we used.

## References


[1] Akiba T., Iwata, Y. Branch-and-reduce exponential/fpt algorithms in practice: A case study of vertex cover. *Theoretical Computer Science.* 609 pp. 211–225, 2016.

[2] Balas, E., Perregaard, M. Lift-and-project for Mixed 0–1 programming: recent progress. *Discrete Applied Mathematics.* Volume 123, Issues 1–3, 2002, pp. 129–154

[3] Depolli, M., Konc, J., Rozman, K., Trobec, R., Janezic, D. Exact Parallel Maximum Clique Algorithm for General and Protein Graphs. *Journal of Chemical Information and Modeling.* 53, 9 2217–2228. 2013.

[4] Faber, V., Lubeck, O.M., White, A.B. Jr. Superlinear speedup of an efficient sequential algorithm is not possible. *Parallel Computing.* Volume 3, Issue 3, pp. 259–260. 1986.

[5] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, Freeman, New York, 2003.

[6] Gellner, A., Lamm, S., Schulz, Ch., Strash, D., Zavalnij, B. "Boosting Data Reduction for the Maximum Weight Independent Set Problem Using Increasing Transformations." In: 2021 Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX) Society for Industrial and Applied Mathematics (SIAM) pp. 128–142. 2021.

[7] Grötschel, M., Padberg, M.W. On the symmetric travelling salesman problem II: Lifting theorems and facets. *Mathematical Programming.* 16, 281–302. 1979.

[8] Karp, Richard M. (1972). "Reducibility Among Combinatorial Problems." In: Complexity of Computer Computations. New York: Plenum. pp. 85–103. 1972.

[9] Lai, T.-H., Sahni, S. Anomalies in parallel branch-and-bound algorithms. *Communications of the ACM.* 27, 6, 594–602. 1984.

[10] Lasserre, J.B., Prieur, Ch., Henrion, D., Trélat, E. Nonlinear optimal control via occupation measures and LMI-relaxations.' *SIAM Journal on Control and Optimization.* 47(4):1643–1666, 2008.

[11] Lovasz, L., Scrijver, A. Cones of matrices and set-functions, and 0-1 optimization. *SIAM Journal on Optimization.* 1:166–190, 1991.

[12] McCreesh, C. and Prosser, P. The shape of the search tree for the maximum clique problem, and the implications for parallel branch and bound. *ACM Transactions on Parallel Computing.* 2(1), 8. 2015.

[13] C. H. Papadimitriou, *Computational Complexity*, Addison-Wesley Publishing Company, Inc., Reading, MA 1994.

[14] Strash, D. On the power of simple reductions for the maximum independent set problem. In: Computing and Combinatorics (COCOON'16), volume 9797 of LNCS, pages 345–356. 2016.

[15] Szabó, S., Zavalnij, B. "Combining algorithms for vertex cover and clique search." In: Proceedings of the 22nd International Multiconference INFORMATION SOCIETY – IS 2019, Volume I : Middle-European Conference on Applied Theoretical Computer Science Ljubljana, Slovenia pp. 71–74. 2019.

[16] Szabó S., and B. Zaválnij, Clique search in graphs of special class and job shop scheduling. *Mathematics.* 10(5), 697. 2022.

[17] Szabo, S. and Zavalnij, B. Graph Coloring via Clique Search with Symmetry Breaking. *Symmetry.* 14 : 8 Paper: 1574 , 16 p. 2022.

[18] Zavalnij, B. "Speeding up Parallel Combinatorial Optimization Algorithms with Las Vegas Method." In: Large-Scale Scientific Computing. LSSC 2015. Lecture Notes in Computer Science, vol 9374. 2015.