On Local Elimination Algorithms for Sparse Discrete Optimization Problems

Daria Lemtyuzhnikova[1], Aleksandr Sviridenko[2], Oleg Shcherbina[3]
Tavrian National University, Simferopol, Ukraine
[1]*darabbt@gmail.com*, [2]*oleks.sviridenko@ gmail.com*, [3]*oshcherbina@ gmail.com*

***Abstract*— We discuss local elimination algorithms that compute global information using local computations. Results of benchmarking show real computational capabilities of block elimination algorithms combined with SYMPHONY solver. Strategies for parallelizing a sequential local elimination algorithm for sparse discrete optimization problems are analyzed. We propose to use hybrid Master-Worker scheme where Worker processors (GPUs) solve concurrently subproblems corresponding to super-nodes of extended elimination tree that are generated by a single master process (CPU).**

Keywords*— *discrete optimization; decomposition; nonserial dynamic programming; local algorithms; elimination

I. INTRODUCTION

The use of discrete optimization (DO) models and algorithms makes it possible to solve many practical problems in scheduling theory, network optimization, routing in communication networks, facility location, optimization in enterprise resource planning, and logistics. To meet the challenge of solving large scale DO problems (DOPs) in reasonable time, there is an urgent need to develop new decomposition approaches. Recently, there has been growing interest in graph-based approaches to decomposition; one of them is tree decomposition (TD) [1]. Among decomposition approaches appropriate for solving sparse DO problems we mention local elimination algorithms (LEA) [2] using the special block matrix structure of constraints and nonserial dynamic programming algorithms (NSDP) [3], [4], which can exploit sparsity in the interaction graph of a DOP and allow to compute a solution in stages such that each of them uses results from previous stages.

II. LOCAL ELIMINATION ALGORITHMS IN DISCRETE PROGRAMMING

A. Main Notions

A graph-based unifying framework for the main structural DO decomposition algorithms is provided in [5] that allows to unify and clarify the notation and algorithms of various structural DO decomposition approaches, and to accommodate constraint satisfaction technology transfer. To overcome a lack of common terminology, a framework "Structural decomposition algorithms" is proposed. Structural decomposition algorithms (or local elimination algorithms (LEA)) compute global information using local computations (i.e., computations of information about elements of neighborhoods of variables or constraints – usually, by solving subproblems). This allows describing all the algorithms under this unifying framework. The structure of a discrete problem is defined by dependencies formulated within the framework of a given modeling paradigm, and can be defined both by an interaction graph of initial elements (variables and constraints in DOP), and various derived structures, e.g., block structures, block-tree structures defined by the so called condensed graph.

The algorithmic scheme of LEA is defined by an elimination tree whose vertices are associated with subproblems and whose edges express information on interdependence between subproblems.

Local elimination algorithms (LEA) use the concept of "structural neighborhood" of elements of a set under consideration. In each step of LEA one studies structural neighborhoods instead of the whole set of variables. The information obtained as a result of this study can be used on the next steps of the LEA. Using the special structure of a block matrix of constraints is efficient for solving block DOPs. LEAs are decomposition algorithms, i.e., they reduce the solution of a large-scale DOP to the solution of a sequence of problems with smaller dimensions which can be solved efficiently using modern DO solvers. There are various computational schemes for realizing the local elimination algorithm (LEA), including the LEA elimination of variables, block- elimination algorithm, LEA based on tree decomposition.

LEA is based on the elimination tree [6], which is a transitive reduction of the interaction graph of DOP and is the smallest data structure representing dependencies between operations. The structure of a DO problem can be defined either by an interaction graph of initial elements (variables and constraints), or by various derived structures, e.g., block structures, block-tree structures defined by a so called condensed graph. The elimination tree [6] manages the process of variables or supervariables elimination and indicates data flows. Thus, the elimination tree structure provides information on data dependency in DOP. It captures the essential ingredient for parallel elimination. An ordering of the elimination tree is a topological ordering where each node is numbered higher than all of its descendants.

B. Variable Elimination and Corresponding Graphs

Let us take a detailed look at an NSDP implementation for solving DO problems for the case when the structural graph is an interaction graph of variables.

Definition [3]. Variables x from X and y from X interact in DOP with constraints (we denote $x \sim y$) if they appear both either in the same component of objective function, or in the same constraint (in other words, if variables are both either in a set X^k, or in a set $X_{S[i]}$).

Definition [3]. Interaction graph of the DOP is called an undirected graph G=(X, E), such that

- Vertices X of G correspond to variables of the DOP;
- Two vertices of G are adjacent if corresponding variables interact.

Further, we will use the notion of vertices that correspond one-to-one to variables.

Definition. Set of variables interacting with a variable x from X is denoted as Nb(x) and called a neighborhood of the variable x.

For corresponding vertices of G a neighborhood of a vertex x is a set of vertices of interaction graph that are linked by edges with x. Denote the latter neighborhood as $Nb_G(x)$.

In hypergraph representation of DO problems structure, the set of vertices H of hypergraph, equals to the set of variables X from the DO problems, and hypergraph's hyperedges forms subsets of related variables that are included in constraints, which means the hyperedge defines constraint scope.

The process of interaction graph transformation corresponding to the variable elimination scheme is known as elimination game (EG). The input of the EG is a graph G and an ordering α of G (i.e. $\alpha(v)=i$ if v is i-th vertex in ordering α. The elimination game consists in the following. At each step i, the neighborhood of vertex v_i is turned into a clique, and v_i is deleted from the graph. This is referred to as eliminating vertex v_i.

Following [7], we define an elimination graph G_{vi} as a graph obtained after the elimination of the vertex v_i from the graph G_{vi-1}.

The filled graph $G^+ = (V,E^+)$ is obtained by adding to G all the edges added by the algorithm. The resulting filled graph G_α^+ is a triangulation of G [FulG], i.e., a chordal graph.

C. *Nonserial Dynamic Programming and Tree Decomposition*

Usually, tree decomposition approaches and nonserial dynamic programming (NSDP) are considered in the literature separately, without reference to the close relation between these methods. One of the goals of this paper is to indicate this relation and a way of transforming the elimination tree of computational NSDP procedure into corresponding tree decomposition. These can bridge the gap between these two approaches. It is shown how to build a tree decomposition associated with the corresponding NSDP procedure. As was mentioned above, this elimination tree of local variable elimination is constructed using Elimination Game. A node of the elimination tree is linked with the first variable (accordingly to the given ordering) from neighborhood. Nodes and edges of desired tree decomposition correspond one-by-one to nodes and edges of the elimination tree. Each node of the tree decomposition is indeed a meta-node containing a subset of vertices of the interaction graph. This subset induces a subgraph in the interaction graph that was condensed to generate the meta-node. Restore these subgraphs for each meta-node of the tree decomposition. It is easy to see that a tree decomposition can be obtained from the elimination tree of NSDP procedure; this fact was noted in [8], [9] and [10].

III. COMPUTATIONAL ASPECTS

A. *Benchmarking ordering techniques for nonserial dynamic programming*

NSDP eliminates variables of DOP using an elimination order which makes significant impact on running time. As finding an optimal ordering is NP-complete [11], heuristics are utilized in practice for finding elimination orderings. The literature has reported extensive computational results for the use of different ordering heuristics in the solution of systems of equations. However, no such experiments have been reported for NSDP to date.

1) *Elimination Ordering Techniques.*

An efficiency of the NSDP algorithm crucially depends on the interaction graph structure of a DOP. If the interaction graph is rather sparse or, in other words, has a relatively small induced width, then the complexity of the algorithm is reasonable.

At the same time an interaction graph leads us to another critical factor such as an elimination order which should be obtained from the interaction graph.

From the other side the NSDP algorithm heavily depends on the elimination ordering. A good elimination ordering yields small cliques during variable elimination. There are several successful schemes for finding a good ordering which we will used in this paper: *minimum degree ordering algorithm* (MD), *nested dissection ordering algorithm* (ND), *maximum cardinality search algorithm* (MCS), *minimum fill-in heuristic* (MIN-FILL) and *lexicographic breath-first search algorithm* (LEX-BFS).

a) *Minimum degree ordering algorithm.*

The minimum degree (MD) ordering algorithm is one of the most widely used in linear algebra heuristic, since it produces factors with relatively low fill-in on a wide range of matrices.

In the minimum degree heuristic, a vertex v of minimum degree is chosen. The graph G', obtained by making the neighborhood of v a clique and then removing v and its incident edges, is built. Recursively, a chordal supergraph H' of G' is made with the heuristic. Then a chordal supergraph H of G is obtained, by adding v and its incident edges from G to H'.

b) *Nested dissection algorithm.*

To create an elimination order, we recursively partition the elimination graph using nested dissection. More specifically, we use METIS library to find a nested dissection ordering.

c) Maximum cardinality search algorithm.

The Maximum Cardinality Search (MCS) algorithm [12] visits the vertices of a graph in an order such that at any point, a vertex is visited that has the largest number of visited neighbors. An MCS-ordering of a graph is an ordering of the vertices that can be generated by the Maximum Cardinality Search algorithm.

The visited degree of a vertex v in an MCS-ordering is the number of neighbors of v that are before v in the ordering.

d) Minimum Fill-in algorithm.

The minimum fill-in heuristic works similarly with minimum degree heuristic, but now the vertex v is selected such that the number of edges that is added to make a neighborhood of v a clique is as small as possible.

e) Lexicographic breadth-first search algorithm

Lexicographic breadth-first search algorithm (LEX-BFS) numbers the vertices from n to 1 in the order that they are selected. This numbering fixes the positions of an elimination scheme. For each vertex v, the label of v will consist of a set of numbers listed in decreasing order. The vertices can then be lexicographically ordered according to their labels.

2) Benchmarking

a) NSDP algorithm implementation

The NSDP algorithm was implemented by the second author in Python. The ND and MD algorithms were implemented in C and C++, respectively.

b) Test problems

For benchmarking the DO test problems were generated by using hypergraphs from the CSP (CSP - Constraint Satisfaction Problem) hypergraph library [13]. This collection contains various classes of constraint hypergraphs from industry (DaimlerChrysler, NASA, ISCAS) as well as synthetically generated ones (e.g. Grid or Cliques).

The test problems were generated in the following way. The constraints structure of a linear DO problem with binary variables was described by hypergraph from the library [13]. To build constraint i the next hyperedge of hypergraph was taken, which includes a set of variables $X_{S[i]}$ for a new building constraint. In the next step, the coefficients for appropriate variables of $A_{S[i]}$ were generated using a random number generator. Then the left part of i-th constraint had a form $A_{S[i]}$ $X_{S[i]}$, while the right part was $\sigma \cdot \Sigma$ $A_{S[i]}$, where σ is random number from interval (0, 1). Objective function is linear and includes all variables - vertices of hypergraph, where coefficients c_j of objective function $\Sigma\, c_j\, x_j \rightarrow \max$ were created with help of random number generator.

After the test problems were generated, the ordering algorithms MD, ND, MCS, MIN-FILL and LEX-BFS were applied for obtaining an elimination ordering. Then the problems were solved with the NSDP algorithm by utilizing to the specified elimination ordering.

3) Benchmarking ordering analysis

Five groups of 33 test problems have been taken: 'dubois', 'bridge', 'adder', 'pret' and 'NewSystem'. All experimental results were obtained on a machine with Intel Core 2 Duo processor @ 2.66 GHz, 2 GB main memory and operating system Linux, version 2.6.35-24-generic. For ND algorithm the minimal run-time of the NSDP algorithm was achieved 0 times (0 %), for MD 2 times (6,0 %), LEX-BFS 3 times (9,1 %), MCS 9 times (27,3 %) and MIN-FILL 19 times (57,6 %) [14].

B. Computational Study of Local Elimination Algorithm

Along with the theoretical analysis of performance evaluations, it is of interest to provide a comparison of LEA combined with other DO algorithms by using computational experiments. Providing an exhaustive computational study for all possible combinations of LEA with all existing DO algorithms (or at least the most efficient ones) is extremely laborious. We have done computational comparisons of the LEA combined with two DO algorithms: a) one of the least efficient DO algorithms, which is a simple implicit enumeration algorithm without use of linear relaxation; b) one of the most effective DO algorithm, as the one adopted by the simplex method in the unimodular case.

The computational capabilities of the LEA in combination with a modern solver were tested by using SYMPHONY (https://projects.coin-or.org/SYMPHONY as the implementation framework. SYMPHONY is part of the COIN-OR (http://www.coin-or.org) project and it can solve mixed-integer linear programs (MILP) sequentially or in parallel. We chose this framework since it is open-source and supports warm restarts, which implement postoptimal analysis (PA) of ILP problems.

1) Test problems description.

All the ILP problems with binary variables from a given experiment have artificially generated quasi-block structures. All the blocks from a single problem have the same number of variables, and also the same number of variables in separators between them. This is required in order to evaluate the impact of the PA on the time to solve the problem by increasing the number of variables.

The test problems were generated by specifying the number of variables, the number of constraints and the size of the separators between blocks. The number and the dimensions of the blocks were calculated by using the number of variables and constraints. The objective function and constraint matrix coefficients, and the right-hand sides for each of the block were generated by using a pseudorandom-number generator.

2) Benchmarking results.

Each test problem was solved by using three algorithms, a) the basic MILP SYMPHONY solver with the OsiSym interface, b) the LEA in combination with SYMPHONY, c) the LEA in combination with SYMPHONY and with PA (warm restarts). In all the cases SYMPHONY used preprocessing.

The computational experiments [15] show that LEA combined with SYMPHONY for solving quasi-block problems with small separators outperforms the stand alone SYMPHONY solver. Additionally, by increasing the size of the separators in the problems for the same number of variables and block sizes

LEA becomes less efficient due to the increased number of iteration for solving the block subproblems. LEA's efficiency is improved by using warm restarts. The ILP problems corresponding to the same block for different values of the separator variables differ only in the right-hand side. These problems can be solved partially by using warm restarts and information obtained from other problems, and this was expected to increase the LEA performance. However, the results show a inconsistent behavior. For most problems, the warm restarts don't make a difference. For some problems, they improved the solution time, while for others they did not.

C. Paralleization of Local Elimination Algorithms

The main disadvantage of the structural decomposition techniques, its great computational complexity for DO problems with high size of separators, can be reduced with parallel processing on clusters of workstations. The exponential amount of computations in blocks of a structural decomposition scheme causes an urgency of coping with this drawback. The main problem is the large size of the separator, which separates the various blocks because a volume of enumeration is exponential on size of the separator. In order to reduce the amount of enumeration, postoptimality analysis can be used, since DO problems in the package, that corresponds to a block, differ in right-hand sides. However, our experiment showed a slight decrease in run time when using postoptimality analysis [15]. Another possibility to cope with the challenges posed by the large size separators is to develop approximate versions of the local algorithm, which does not completely enumerate all possible values of the variables included in the separator, but only partially. Thus, the only one way to implement and use exact local elimination algorithms for sparse DO problems with large separators is using of parallelization. Parallelization in DO can help to cope with this drawback using the possibility of parallel solving of DO problems in blocks. We propose to use a hybrid system allowing the concurrent use of CPUs and GPUs. The GPUs, many core parallel machines with shared memory, act as the workers and perform solving DO subproblems for blocks. Synthesis of solution is performed by the CPU which has the role of the master.

The elimination tree serves to characterize the parallelism in solving DOPs with LEA. In particular, the height of the elimination tree gives a rough measure of the parallel computation time.

Two sources of parallelization exist: the dominating one is based on partial problem assignments in separators, the second one comes from the branching of the tree-decomposition itself.

There are two types of parallelism available in structural decomposition algorithm:

1) Parallelism of type 1 introduces parallelism when performing the operations on generated subproblems (in blocks). It consists of solving each DO subproblem in parallel for each block to accelerate the execution.

2) Parallelism of type 2 consists of processing the elimination tree in parallel by performing operations on several subproblems simultaneously. Independent branches of the elimination tree can be processed in parallel, and we refer to this as type 2 parallelism or tree parallelism. It is obvious that in general, tree parallelism can be exploited more efficiently in the lower part of the elimination tree than near the root node.

We propose to develop effective GPU-based parallel algorithms for solving sparse real-world DO problems. This research should analyze and develop new parallel methodology using using structural decomposition combined with linear algebra approaches for solving large-scale DO problems.

REFERENCES

[1] O.A. Shcherbina, "Tree decomposition and discrete optimization problems: A survey," Cybernetics and Systems Analysis, vol.43, pp. 549-562, 2007.

[2] O.A. Shcherbina, "Local elimination algorithms for solving sparse discrete problems," Computational Mathematics and Mathematical Physics, vol. 48, pp. 152—167, 2008.

[3] U. Bertele, F. Brioschi, Nonserial Dynamic Programming. New York: Academic Press, 1972.

[4] O. Shcherbina, "Nonserial dynamic programming and tree decomposition in discrete optimization," in Proc. of Int. Conference on Operations Research "Operations Research 2006", Karlsruhe, 6-8 September, 2006, Berlin: Springer Verlag, 2007, pp. 155-160.

[5] O. Shcherbina, "Graph-Based Local Elimination Algorithms in Discrete Optimization," in Foundations of Computational Intelligence Volume 3. Global Optimization Series: Studies in Computational Intelligence, vol. 203 / A. Abraham; A.-E. Hassanien; P. Siarry; A. Engelbrecht, Eds. Berlin / Heidelberg: Springer. 2009, XII, pp. 235-266.

[6] J.W.H Liu, "The role of elimination trees in sparse factorization," SIAM J. on Matrix Analysis and Applications, vol. 11, pp. 134-172, 1990.

[7] D.J. Rose, "A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations," in Graph Theory and Computing, R.C. Read (ed), New York: Academic Press, 1972, pp. 183—217.

[8] R. Dechter, J. Pearl, "Tree clustering for constraint networks," Artif. Intell., vol. 38, pp. 353-366, 1989.

[9] K. Kask, R. Dechter, J. Larrosa, A. Dechter, "Unifying cluster-tree decompositions for reasoning in graphical models," Artif. Intell., vol. 160, pp. 165-193, 2005.

[10] A. Neumaier, O. Shcherbina, "Multifrontal techniques for sparse problems," unpublished.

[11] M. Yannakakis, "Computing the minimum fill-in is NP-complete," SIAM J Alg Disc Meth, vol. 2, pp. 77-79, 1981.

[12] R.E. Tarjan, M. Yannakakis, Simple linear-time algorithms to test chordality of graphs, test acyclity of hypergraphs, and selectively reduce acyclic hypergraphs, SIAM J Comput, Vol. 13, pp. 566-579, 1984.

[13] N. Musliu, M. Samer, T. Ganzow, G. Gottlob, A CSP hypergraph library, Technical Report, DBAI-TR-2005-50, Technische Universität Wien, 2005.

[14] A. Sviridenko, O. Shcherbina, "Benchmarking ordering techniques for nonserial dynamic programming," 2011, arXiv:1107.1893v1 [cs.DM].

[15] A. Sviridenko, O. Shcherbina, "Block local elimination algorithms for sparse discrete optimization problems," Cybernetics and Systems Analysis (on review). Available online: arXiv:1112.6335v1 [cs.DM].