PhD title: "Embedding Machine Learning in Optimization Solvers"

Advisor: Roberto Wolfler Calvo, LIPN (roberto.wolfler@lipn.univ-paris13.fr)
Co-advisor: Antonio Frangioni, Università di Pisa (frangio@di.unipi.it)
Co-advisor: Mathieu Lacroix, LIPN (lacroix@lipn.univ-paris13.fr)

Scope

State-of-the-art solvers for hard optimization problems are highly complex software systems, composed by many sophisticated algorithms that have to work synergistically in order to reach maximum efficiency. Indeed, general-purpose optimization solvers have many algorithmic parameters (up to more than 100), that can be properly tuned to attain the best performances. While the default values of these parameters often lead to acceptable performances (due to experience-induced extensive experimental tuning performed by the solvers developers), significant and sometimes dramatic performance improvements can be obtained by further tweaking them. This is, however, a costly process that requires highly experienced personnel.

More and more modelling frameworks and solvers, such as SMS++[1], Julia/JuMP [2] or SCIP [3], provide ways to decompose hard problems into blocks, whose structure can then be exploited during the solution process by means of specialised solvers. However, this further increases the complexity of the solution process, and therefore the number of algorithmic parameters to tune. Indeed, entirely different solvers, each with its set of parameters, can be used to solve one given block. Furthermore, exploiting the structure usually means that decomposition algorithms have to be used, which have their own set of nontrivial parameters to set in addition to these of "standard" general-purpose solvers.

Machine Learning (ML) has been recently widely used in mixed integer programming (MIP). Indeed, a significant amount of research has focused on improving mathematical programming methods with the use of ML. For instance, ML has been successfully applied to improve branching [4] and to determine when to search for primal bounds [5] in branch-and-bound methods. ML is also used to guess whether a problem can be efficiently tackled by a MIP solver [6, 7], allowing dynamic restarts in branch-and-bound methods [7], whether to apply Dantzig-Wolfe decomposition [8], or to automatically derive useful Gomory cuts to strengthen the linear relaxation [9].

Arguably, most of these efforts can be seen as special cases of the general Algorithm Configuration Problem (ACP): given an algorithm with parameters, and a target instance, find the setting of the parameters that optimise some measure of the algorithm's performances on the instance. Clearly, ACP is a hard problem for several reasons. However, ML may provide a promising approach, and indeed recent work on ACP focus on the use of ML techniques [10, 11]. In particular, a general methodology for ML-based ACP is proposed in [12, 13, 14]. The approach is particularly targeted at "complex" solvers with "many" parameters. This is obtained by first learning a performance function with a ML of choice (SVR, NN, decision trees, ...), and then solve an optimization problems (CSSP) using the learnt model to obtain the optimal configuration. While this idea is natural, the ML technique has typically been considered as a "black box", which implies the use of "derivative-free" methods to solve the CSSP; but these methods may have difficulties when dealing with functions with "many" parameters. Instead, the ML model can be regarded as a "white box", and therefore the CSSP solved by making full use of powerful optimization techniques such as Mixed-Integer NonLinear solvers, which allows the approach to potentially scale to much larger spaces. The alternative approach where the ML technique directly produced the optimal configuration can be considered, but it may have difficulties with "complex" parameter spaces corresponding to the fact that one algorithmic choice constrain potentially many others; these constraints are instead easily represented within the CSSP.

While the approach seem to have potential, its success is highly dependent on two factors:

1. availability of a very large training set of different instances and different algorithmic parameters, as this is instrumental for the ML technique to produce good estimates of the performance function;

2. definition of a proper set of *features* that represent the instance in the ML process.

The first aspect could be considered only a practical limitation, but in fact it is conceptual: not only the set of instances to draw from is typically infinite; the set of configurations is also extremely large, and these two aspects compound each other. The second aspect could also be considered as "easily solvable" by the standard ML techniques of *feature engineering*, possibly followed by *feature selection*. However, while the second admits several algorithmic solutions, the first typically requires considerable work and human expertise to start from.

Thus, it is still unclear to which extent the ML approach to ACP can be made practical. While efforts are underway for "monolithic" general-purpose solvers, as previously mentioned the issues may become even harder for structure-exploiting solvers, in that the scale and the complexity of the parameter space may grow dramatically.

Research proposal

The vast majority of the optimization algorithms present a certain degree of *modularity*, i.e., they consists of several blocks/routines that solves simpler problems repetitively. A well developed class of algorithms capable of exploiting modularity are decomposition methods, among which chiefly Lagrangian relaxation/Dantzig-Wolfe reformulation/Column Generation and Benders' decomposition, although other ways to exploit modularity exist (e.g., structured interior-point methods). As the name suggests, decomposition algorithms rely on decomposing a problem into distinct subproblems linked by a master problem. The resolution method alternates between solving the master problem and the subproblems until the optimum is found. The subproblems are solved a large number of times during the resolution process; each time, they may differ "little" from the problems solved at the previous iterate, although this may depend on the stage of the decomposition algorithm. This means that *reoptimization* can be crucial to the efficiency of the method, which in turn implies that algorithmic parameters working well in one-shot may not be optimal for use during decomposition (e.g., [15]).

Although modelling systems and solvers are gradually starting to incorporate features aimed at decomposition techniques, typically at most one level of decomposition is considered. However, very-large-scale optimization problems may require multiple heterogeneous nested decomposition approaches, whereby the subproblems of the decomposition are in turn solved by a decomposition approach (possibly of a different type). The open-source Structured Modelling System++ (SMS++)[1] has been developed precisely to cater these applications by providing a number of unique features, among which automatic communication of instance changes to all interested solvers and support for asynchronous computation. SMS++ is expected to be eventually endowed with features that allow ML-based ACP methods to be applied, along the lines of [12]. However, whether these features actually make sense, and what their design should ultimately be, depends on the answer to two highly nontrivial research questions that this Ph.D. thesis proposal seeks to address. In particular, we want to determine if the crucial issues of ML-based ACP methodologies exposed above can be effectively tackled by exploiting two characteristics of the subproblems solved during decomposition approached: (i) they are (hopefully) easier to analyse, in particular "at the leaves" of the decomposition tree, and (ii) during one execution of the algorithm they are solved many times, and from one iteration to the other the change is limited. The two mentioned aspects leads to the two research questions described in the following.

RQ1: How to exploit the combination of simple blocks in a decomposition algorithm?

As previously mentioned, a key aspect in ML-based solution approaches to ACP is the identification of the important features of an instance that allows to properly learn the optimal parameter configuration. In the context of a (multi-level, heterogeneous) solution approach, one may reasonably assume that the identification of the features becomes a simpler task "at the leaves" of the decomposition tree, where the subproblems are often "simple" combinatorial problems such as knapsacks, flows, constrained shortest paths. Hence, one may hope that a case-by-case analysis for these simpler problems may provide a reasonable set of features for each, which has the advantage that once this is done it can be re-used in the possibly many decomposition schemes that have that as a subproblem.

However, this arises the question: once the parameters for the different blocks are learned separately, do the still work when combined together? In other words: is the superposition principle valid to the different

blocks of an optimization software? If this is true, once the learning of all the sub-blocks is clear, it might be possible to define a set of features for the whole approach. This, of course, would mean that a set of features can be determined for the "master" block and the corresponding solution approach. If this is possible, repeating this process throughout all the decomposition tree would lead to automatically having a "good" set of features for even a huge-scale problem solved by a very complex algorithm based on a relatively small set of "good" sets of features, each of which may reasonably be obtained "manually". This would go a long way to make ML-based solution approaches to ACP suitable for large-scale complex problems, as opposed to the case where the whole set of features need to be determined at once since there are significant interactions that cannot be reconstructed from the individual parts.

RQ2: How Machine Learning can improve the repetitive resolution of subproblems in decomposition algorithms?

The objective is to exploit the peculiarities of decomposition algorithms, in particular for the solution of subproblems, to overcome the other possible limiting factor, i.e., the need for a large test set to work on. Indeed, decomposition algorithms typically solve the same subproblem many times, with "slight and localised" changes in the data, which naturally allows to construct large data sets. Yet, there are some nontrvial points to be clarified:

1. The decomposition algorithm may itself be a stage of a more complex solution process (say, Column Generation used within a Branch-and-Price approach); hence, there could be rather different kinds of changes, which may have different impacts on the algorithms and therefore require different parameter configurations. For this issue SMS++ may provide convenient since it possible to know *exactly what data has changed*, which may be crucial to identify these cases.

2. If the repeated solution of the subproblem has to be used to construct a test set from which to learn the effect of parameter changes, the parameters need necessarily change. Thus, the process must have a component in which the previously used configuration, even if it has worked "well", may have to be changed in order to explore a new part of the configuration space, which incurs a risk of serious deterioration of performances. For this issue SMS++ may provide convenient since it allows to have several solvers (possibly, copies of the same one with different parameter configurations) attached to the same subproblen and run them in parallel in response to the same data change, which would allow to exploit available processing power to explore new configurations while guaranteeing that the solution is found at lest as efficiently as the initial configuration allows.

All this may allow to use ML-based ACP solution techniques dynamically during the solution process of a decomposition algorithm in order to improve the performances.

It should be noted that this has a nontrivial impact on RQ1, whereby we assume to be statically able to predict the performances of the solution process. While in principle the use of ML-based ACP is "just any other algorithmic technique", and therefore can be analysed with the same tools, it is not obvious whether the use of ML-based ACP within the subproblems makes them easier or harder to predict. On the other hand, a more "global" perspective may also be interesting. That is, in principle algorithmic parameters of the subproblem may not have a significant impact (or even have a negative one) on its performances, but still have a positive impact on the overall decomposition algorithm (say, produce "better" columns that lead to a faster solution of the whole problem). While this may entail a negative response for RQ1, at least partly, it may be an interesting outcome in its own right.

We mention that this research line may naturally lead to a somewhat different direction, which is to make greater use of ML in order to more efficiently solve the subproblems. Many of those are combinatorial problems, which may be solved by heuristics to speed up the resolution of the subproblems. ML is already used to improve algorithms for solving combinatorial optimization problems [16, 17, 18]. Knowing that the problem is solved many times with "small" changes in the data may lead to devise ML techniques that learn from the sequence of previous solutions (or a restriction, down to possibly only the previous one) in order to suggest ways to more efficiently produce the next one. This may be put into the framework of ML-based ACP, but the set of parameters may be so large as to practically become a ML-based heuristic. Whether or not this line of approach is promising will be determined during the course of the research.

Required skills

The PhD student should have a master degree in combinatorial optimization or operational research. A background in machine learning and programming skills in C++ will be much appreciated.

References

[1] `https://smspp.gitlab.io/smspp-project/`.

[2] I. Dunning, J. Huchette, and M. Lubin, "Jump: A modeling language for mathematical optimization," *SIAM Review*, vol. 59, no. 2, pp. 295–320, 2017.

[3] G. Gamrath, D. Anderson, K. Bestuzheva, W.-K. Chen, L. Eifler, M. Gasse, P. Gemander, A. Gleixner, L. Gottwald, K. Halbig, G. Hendel, C. Hojny, T. Koch, P. L. Bodic, S. J. Maher, F. Matter, M. Miltenberger, E. Mühmer, B. Müller, M. Pfetsch, F. Schlösser, F. Serrano, Y. Shinano, C. Tawfik, S. Vigerske, F. Wegscheider, D. Weninger, and J. Witzig, "The SCIP Optimization Suite 7.0," technical report, Optimization Online, March 2020.

[4] M.-F. Balcan, T. Dick, T. Sandholm, and E. Vitercik, "Learning to branch," 2018.

[5] E. B. Khalil, B. Dilkina, G. L. Nemhauser, S. Ahmed, and Y. Shao, "Learning to run heuristics in tree search," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017* (C. Sierra, ed.), pp. 659–666, ijcai.org, 2017.

[6] M. Fischetti, A. Lodi, and G. Zarpellon, "Learning milp resolution outcomes before reaching time-limit," in *Integration of Constraint Programming, Artificial Intelligence, and Operations Research* (L.-M. Rousseau and K. Stergiou, eds.), (Cham), pp. 275–291, Springer International Publishing, 2019.

[7] D. Anderson, G. Hendel, P. L. Bodic, and M. Viernickel, "Clairvoyant restarts in branch-and-bound search using online tree-size estimation," in *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pp. 1427–1434, AAAI Press, 2019.

[8] M. Kruber, M. E. Lübbecke, and A. Parmentier, "Learning when to use a decomposition," in *Integration of AI and OR Techniques in Constraint Programming* (D. Salvagnin and M. Lombardi, eds.), (Cham), pp. 202–210, Springer International Publishing, 2017.

[9] Y. Tang, S. Agrawal, and Y. Faenza, "Reinforcement learning for integer programming: Learning to cut," 2019.

[10] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming," *RCRA Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion at IJCAI*, no. Rcra 2011, pp. 16–30, 2011.

[11] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Automated configuration of mixed integer programming solvers," in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems* (A. Lodi, M. Milano, and P. Toth, eds.), (Berlin, Heidelberg), pp. 186–202, Springer Berlin Heidelberg, 2010.

[12] A. Frangioni and L. Perez Sanchez, "Searching the best (formulation, solver, configuration) for structured problems," in *Complex Systems Design & Management: Proceedings of the First International Conference on Complex Systems Design & Management CSDM 2010*, pp. 85–97, Springer-Verlag, 2010.

[13] G. Iommazzo, C. D'Ambrosio, A. Frangioni, and L. Liberti, "A learning-based mathematical programming formulation for the automatic configuration of optimization solvers," in *Lecture Notes in Computer Science, 6th International Conference on Machine Learning, Optimization and Data science - LOD 2020*, Springer-Verlag, 2020.

[14] G. Iommazzo, C. D'Ambrosio, A. Frangioni, and L. Liberti, "Learning to configure mathematical programming solvers by mathematical programming," in *Lecture Notes in Computer Science, 14th Learning and Intelligent Optimization, LION 14*, Springer-Verlag, 2020.

[15] A. Frangioni and A. Manca, "A computational study of cost reoptimization for min cost flow problems," *INFORMS Journal on Computing*, vol. 18, no. 1, pp. 61–70, 2006.

[16] E. B. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA* (I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, eds.), pp. 6348–6358, 2017.

[17] Z. Li, Q. Chen, and V. Koltun, "Combinatorial optimization with graph convolutional networks and guided tree search," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, (Red Hook, NY, USA), p. 537–546, Curran Associates Inc., 2018.

[18] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," 2020.