Multi-label Classification for the Generation of Sub-problems in Time-constrained Combinatorial Optimization

Luca Mossina, Emmanuel Rachelson, Daniel Delahaye

Multi-label Classification for the Generation of Sub-problems in Time-constrained Combinatorial Optimization

Luca Mossina[1], Emmanuel Rachelson[1] and Daniel Delahaye[2]
[1]*ISAE-SUPAERO, Université de Toulouse, Toulouse, France*
[2]*ENAC, Université de Toulouse, Toulouse, France*

Keywords: Multi-label Classification, Mixed Integer Linear Programming, Combinatorial Optimization, Recurrent Problems, Machine Learning.

Abstract: This paper addresses the resolution of combinatorial optimization problems presenting some kind of recurrent structure, coupled with machine learning techniques. Stemming from the assumption that such recurrent problems are the realization of an unknown generative probabilistic model, data is collected from previous resolutions of such problems and used to train a supervised learning model for multi-label classification. This model is exploited to predict a subset of decision variables to be set heuristically to a certain reference value, thus becoming fixed parameters in the original problem. The remaining variables then form a smaller sub-problem whose solution, while not guaranteed to be optimal for the original problem, can be obtained faster, offering an advantageous tool for tackling time-sensitive tasks.

1 INTRODUCTION

In combinatorial optimization, some problems are recurrent in nature and similar instances need to be solved repeatedly over time, with some modifications in the model parameters. We aim at extracting knowledge from the solutions of past instances and learn how to speed-up the resolution process of future instances. When solving a mixed integer linear programming (MILP) problem, for example, one can be faced with a non-negligible computational task. For large instances, resolution methods based on standard Branch-and-Bound (BB) (Schrijver, 1998) techniques or metaheuristics such as genetic algorithms (GA) may fail to deliver a good solution within a restricted time window.

We start by assuming the existence of a reference model, that can be thought of as a system under nominal values, and a corresponding reference optimal solution. If a variation in the reference problem is detected, we may have interest in changing the value of some decision variables. What if we could have an oracle telling us exactly what variable needs to have its value changed, with respect to the reference problem? We try to approximate such an oracle via machine learning. We proceed predicting which of the parts of the reference solution can be left unchanged and which others need to be re-optimized. We obtain a sub-problem that can deliver a good, approximate solution in a fraction of the time necessary to solve the full instance.

We do not aim at predicting the assignment of decision variables directly, but rather at providing some hints to the solver by imposing additional constraints on the search space. The decision variables of the full problem P, individually or in clusters, are seen as the targets in a classification problem whose outcome would be the choice of whether to exclude them or not from the resolution of the new instance of the model. We thus define the subproblem SP as what is left of P after a certain number of its variables are selected and fixed to a reference value, also referred to as blocked variables. These variables become parameters of SP and thus do not affect its resolution, being invisible to the solver. The variable selection can be framed as a multi-label classification (MLC) problem, where one wants to associate a set of labels (variables) from a set of available choices, to an observation P. To accomplish this, we introduce the *SuSPen* meta-algorithm, a procedure that uses MLC to generate sub-problems via a select-and-block step.

1.1 Unit Commitment Example

To better illustrate our framework, let us introduce a practical example. In the context of energy produc-

tion, the operator has to choose how to satisfy the electricity demand with the technology at their disposal. They must decide, among other things, which power units to activate, how to set the output levels for each generator or decide whether it is worth to overproduce and export energy or under-produce and rely on other operators to buy energy at a better price. As the network of units can be assumed to remain mostly unchanged from one period to another, this problem, also known as the *unit-commitment problem* (UC) (Padhy, 2004), requires the resolution of similar instances over time: to minimize the power generation costs under demand satisfaction and operational constraints like network capacity or technical and regulatory limitations. This can be cast as a problem of MILP optimization (Carrión and Arroyo, 2006) where each daily instance can be considered as a random variation on the parameters of the model (demand, scheduled/unscheduled maintenance, etc.).

We consider a reference daily production problem P_{ref}. All daily instances P to be solved are variations of the coefficients defining P_{ref}, due to, for example, some deviations from the expected national or regional energy demand constraints. Solving these instances would require the operator to re-run the optimization performed on P_{ref} which, if given a small notice, may not be achievable within an limited time window.

The idea we develop is to apply a machine learning (ML) procedure that maps a random event that perturbates P_{ref} to a subset of variables more likely to be affected by the variation. We will assume the solution $x^{\star}_{ref}$ of P_{ref} to be feasible for P. Some of the values will be set heuristically as in the original plan $x^{\star}_{ref}$ while the remaining problem is fed to a solver. This is what we refer to as sub-problem generation.

1.2 Our Contribution

We propose an approach based on MLC algorithms and the meta-algorithm *SuSPen* for blocking decision variables. After some of the variables in a problem are *blocked* to a certain value, the remaining problem is processed via a standard solver as if it were a black box. With our work, we want to provide a way to speed up the resolution of hard combinatorial problems independently of the technology employed to solve them, be it an exact or approximate method.

2 RELATED WORK

Combinatorial optimization is a fundamental tool in many fields so it comes at no surprise that contributions originate from the Machine Learning, Artificial Intelligence, or Operations Research communities.

Among the first works explicitly mentioning the concept of combining ML and combinatorial optimization (CO) is that of (Zupanič, 1999), who proposed to couple constraint logic programming with BB algorithms to obtain *value suggestions* for decision variables. One can learn from data how to generate additional constraints to the original problem, restricting the search space and averaging faster resolution time at the cost of losing the optimality guarantee.

The application of ML to solve CO problems has recently seen a surge in interest. Promising results (Kruber et al., 2017) were obtained by learning if and which decomposition method is to be applied to MILP problems, a task that is usually carried out *ad-hoc* with expert domain knowledge for each problem (Vanderbeck and Wolsey, 2010). The authors attempt to automate the task of decomposition identification via classification algorithms, determining whether a decomposition is worth doing and, if that is the case, they then select the most appropriate algorithm. On the same line, recent research (Basso et al., 2017) dealt with clarifying whether such an approach makes sense, that is, whether decomposition can be determined by examining the static properties of a MILP instance, such as the number of continuous and/or binary and/or integer variables, the number of (in)equality constraints or the number of decomposition blocks and their characteristics. In the case of Mixed Integer Quadratic Programming (MIQP), the available solvers offer the user the possibility to linearize the problem. This apparently simple decision problem however, is inherently hard and no method exist to take this decision rapidly. While not definitive, their conclusion is that, on average, this approach can yield good results and is well founded. In (Bonami et al., 2018) the authors deploy ML classification to build a system taking such decisions automatically, adding a prediction step to resolution via the CPLEX solver, obtaining promising results.

In (Fischetti and Fraccaro, 2018) we can find an interesting potential real-world application, in the case of the wind park layout problem. Because of complex interactions among the wind turbines, many different configurations would have to be simulated and evaluated, which would be considerably computationally expensive. As in practice an extensive computational simulation may not be viable, the authors tried to approximate these optimization Via ML, predicting the output of a series of unseen layouts without going thourgh the optimization process, thanks to a dataset built *a priori*. The work of (Larsen et al.,

2018) is of similar nature. Where the problem of choosing the optimal load planning for containers on freight trains cannot be solved online because of time constraints and insufficient information about the instance, a set of offline cases can be collected and exploited to obtain a decription of the solution of a particular instance, at an aggregate level. Such description, the authors affirm, can provide meaningful insights to decision makers in a real-time contexts.

The most prolific field so far has been that of learning to branch in BB. While branching rules were proposed in the past, such as *strong branching*, *pseudocost branching* or *hybrid branching* (Achterberg et al., 2005), these involved no use of learned rules, but rather relied on a set of reasonably good assumptions. Some of the most promising results (Alvarez et al., 2017; Khalil et al., 2016) move towards the supervised approximation of branching rules within the BB procedure. Imitation learning (He et al., 2014), worked on finding a node ordering to guide the exploration of promising branches in the BB tree. A recent article by (Lodi and Zarpellon, 2017) provides a thorough overview on the state-of-the-art of machine learning for branching rules.

Departing from the exact resolutions based on BB, (Dai et al., 2017) propose a greedy algorithm to solve or approximate hard combinatorial problems over graphs, based on a reinforcement learning (RL) paradigm. They are, to the best of our knowledge, the only authors so far focusing explicitly on *recurrent* combinatorial optimization problems.

The main methodological difference between our approach and theirs is the integration of the learned rules in the solver: while they build a greedy algorithm around their learning structure, we build the learning around the solver (BB, metaheuristics, etc.).

In the field of energy production, (Cornelusse et al., 2009) has proposed a method for reacting to deviations in forecasted electricity demand, modeled as a random perturbation of some base case. After running simulations on a set of possible scenarios, the results of re-optimization of such scenarios will form a database to be exploited by the learning algorithm, mapping the current state of the system to a set of time-series describing the adjustments to be operated on each generating unit to respond to the mutated conditions. The problem of responding quickly to a change in the parameters of a model (energy demand) was addressed by (Rachelson et al., 2010) by learning from resolutions of problems within the same family. The aim is to infer the values of some binary variables in a MILP, thus reducing the size of the problem to be solved, via a so-called *boolean variable assignment* method.

Unlike their approach, we do not work towards finding an assignment for a decision variable via ML, but rather we aim at localizing uninteresting variables in a new instance, the ones that we think will be unaffected by the perturbation in our reference problem P_{ref}. Once the interesting zones of our search space are found, we block the non relevant ones to some heuristic value and let the solver optimize the restricted problem.

(Loubière et al., 2017) have worked to integrate sensitivity analysis (Saltelli et al., 2008) into the resolution of nonlinear continuous optimization. After successfully computing a weight for each decision variable and ranking their influence, this piece of information is integrated into a metaheuristic, improving the overall performance. We point out the similarities between their approach and ML-based ones, not in the algorithms but rather in the philosophy of injecting prior knowledge into the resolution of optimization problems.

3 MULTI-LABEL CLASSIFICATION

We will provide a brief introduction to the concepts of multi-label classification (MLC), the ML paradigm of choice in our hybrid approach.

Given a set of targets $\mathcal{L}$, referred to as labels, MLC algorithms aim at mapping elements of a feature space $\mathcal{X}$ to a subset of $\mathcal{L}$, as $h : \mathcal{X} \longrightarrow \mathcal{P}(\mathcal{L})$. The two typical approaches for such problems are known as binary relevance (BR) and label powerset (LP). The first considers each label in $\mathcal{L}$ as a binary classification problem, transforming the main problem into $h_i : \mathcal{X} \longrightarrow l_i, l_i \in \{0,1\}, i = 1,...,|\mathcal{L}|$. The latter transforms a problem of MLC into one of multiclass classification, mapping elements $x \in \mathcal{X}$ directly to subsets of labels $s \in \mathcal{P}(\mathcal{L})$. Its practical use is limited to cases with only a few labels because of the exponential growth of the cardinality of $\mathcal{P}(\mathcal{L})$. In the literature many variations on these two approaches exist, where different classifiers are explored (support vector machines, classification trees, etc.). The fundamental concepts, variations, recent trends and state-of-the-art of MLC have been the object of a recent review by (Zhang and Zhou, 2014). Among the most interesting and effective models we point out to the work on *classifiers chains* (CC) by (Read et al., 2011) and (Dembczynski et al., 2010) and on the *RAndom k-labELsets* (RAkEL) algorithm by (Tsoumakas and Vlahavas, 2007), respectively a variation on BR and on LP. Departing from these approaches is the work of (Mossina and Rachelson, 2017), who proposed an ex-

tension of naive Bayes classification (NBC) into the domain of MLC that can handle problems of high cardinality in the feature and label space $\mathcal{L}$; we will adopt this algorithm for our MLC tasks. Predictions are made following a two-step procedure, first by estimating the size of the target vector, then by incrementally filling up the vector conditioning on the previous elements included in the vector (see Algorithm 1), employing a multiclass naive Bayes classifier. For problems of many labels, one thus does not need to train a model for each label, as in BR for instance.

Algorithm 1: *NaiBX*, Prediction Step.

predict_subset($\mathbf{x}_{new}$):
  $\hat{m} \leftarrow$ predict_size($\mathbf{x}_{new}$)
  $\mathbf{y}_{pred} \leftarrow \emptyset$
  while $length(\mathbf{y}_{pred}) \leq \hat{m}$ **do**
    $\mathbf{y}_{pred} \leftarrow \mathbf{y}_{pred} \cup$ predict_label($\mathbf{x}_{new}, \hat{m}, \mathbf{y}_{pred}$)
end
  return $\mathbf{y}_{pred}$

predict_size($\mathbf{x}_{new}$):
  $\hat{m} \leftarrow \underset{d \in \{0,1,\ldots,L\}}{\text{argmax}} \ \mathbb{P}(m_d) \times \prod_{i=1}^{n} \mathbb{P}(X_i \mid m_d)$
  return $\hat{m}$

predict_label($\mathbf{x}_{new}, \hat{m}, \{y_1, y_2, \ldots, y_i\}$):
  $y_{i+1} \leftarrow \underset{y_{i+1} \in \{\mathcal{L}\}}{\text{argmax}} \ \mathbb{P}(y_{i+1}) \times \mathbb{P}(\hat{m} \mid y_{i+1}) \times \prod_{i=1}^{n} \mathbb{P}(x_i \mid y_{i+1}) \times \prod_{j=1}^{i} \mathbb{P}(y_j \mid y_{i+1})$
  return $\{y_1, y_2, \ldots, y_i\} \cup \{y_{i+1}\}$

$\mathbf{x_{new}}$ is the features vector of a new observation; n is the number of features;
L is number of labels available.

Combinatorial optimization can suffer from problems in symmetry, that is, many optimal solutions exist and they cannot be easily spotted. Generally, when doing MLC, there is only *one* correct label vector and we want to ideally predict exactly each one its elements. In our context, this is less of a concern. We know that there could possibly more than one *correct* vector of labels, depending on the desired *SP* size, because of this symmetry issue. Not only is this not a major problem, but it would guide the search in a specific direction, preventing us from spending computation time in breaking symmetries.

4 COMBINATORIAL OPTIMIZATION

In this section we discuss the formulation of MILP problems and present the model for our case study. We present an application to the domain of mathematical programming (MILP) solved via BB. The UC case introduced in Section 1.1 will serve as a benchmark for MILP, while for the GA we present the well known problem of the traveling salesman (TSP).

4.1 Mixed Integer Linear Programming Modeling

MILP are defined simply as linear programs (LP) with integrality constraints on some of the decision variables, which include binary variables as a special case. These constraints transform linear programs from efficiently solvable problems (polynomial time) into hard combinatorial ones.

Definition 1 Mixed Integer Linear Program (MILP)

$$\begin{aligned} \min_{x_R, x_I} \quad & c_R^t x_R + c_I^t x_I \\ \text{subject to} \quad & A_R x_R + A_I x_I = b \\ & x_R \in \mathbb{R}_+^{n_R}; x_I \in \mathbb{N}^{n_I}; \end{aligned}$$

where $[c_R, c_I] = c \in \mathbb{R}^{n_R+n_I}$, $[A_R, A_I] = A \in \mathbb{R}^{m \times (n_R+n_I)}$, $b \in \mathbb{R}^m$ with n_R and n_I respectively the number of real and integer variables, and m the number of constraints.

MILP is a flexible modeling tool and finds applications in a wide variety of problems. For UC, many refined MILP formulation are available in the literature (Padhy, 2004; Carrión and Arroyo, 2006). We adapted the energy production model proposed by (Williams, 2013) and developed an instance generator. One of the key factors behind our work is that in the industry, recurrent problems are solved continuously and great amounts of data are readily available.

4.2 Genetic Algorithms

For some classes of problems, specialized formulations exist that guarantee very fast resolution times. When no special formulations are available and exact methods are not efficient, heuristics and randomized search heuristics (RSH) (Auger and Doerr, 2011) can prove to be very competitive methods. Genetic algorithms (GA) are a form of randomized exploration of the solution space of a combinatorial problem and can handle linear or nonlinear objective functions; many variations and applications are available in the literature (Gendreau and Potvin, 2010).

We apply this to the *traveling salesman problem* (TSP), one of the most studied problems in combinatorial optimization (Dantzig et al., 1954) (Applegate et al., 2011). A desirable property of this problem is the existence of the Concorde TSP solver (Applegate et al., 2006), the state-of-the-art solver that has been capable of solving instances of up to tens of thousands of nodes. An instance generated by a completely connected 3D graph of 500 cities can be solved in under 7 seconds on a standard laptop computer. We can easily compute exact solutions for our benchmarks instances to better evaluate our experiments.

5 THE *SuSPen* META-ALGORITHM

Given a problem P to solve and its reference problem P_{ref}, we want to detect the differences between the two and detect which parts of the reference solution are more likely to be affected by these differences. The parts predicted to be affected will be run through the optimization process while those thought to be unaffected will be left at their reference value. The decision variables x_B not affected by the optimization process will be blocked while the remaining ones x_{SP} will form the sub-problem SP fed to the solver (see Meta-algorithm 2).

Meta-Algorithm 2: *SuSPen*, Supervised Learning for Sub-problem Generation
(1) Simulate or collect data from past resolutions
(2) Train a MLC classification model on data from (1)
(3) For a new Problem P':
  (3.1) Predict $\{x_B, x_{SP}\}$
  (3.2) Generate SP
  (3.3) Solve SP via external solver

The result of this resolution is not guaranteed to be optimal, although it is highly likely to be of faster resolution (see Section VII). This approach is interesting, for example, in problems that need to be solved repeatedly under random perturbations of the parameters, where one is expected to react promptly to changes in a system. In UC, for example, the search for a feasible solution is hardly ever the core of the problem, as the network of units itself is dimensioned specifically to respond to the demand of the territory on which it operates and one can expect to have redundancy built-in into the grid. The focus, thus, is more on finding a way to react promptly to unexpected problems and limit losses as much as possible. While most of our efforts and the cases reported in the literature have been so far focused on exact methods, working directly with decision variable assignments allows to easily switch from one optimization method to another, while retaining the machine-driven exploration of the search space. When applying our hybrid paradigm, the classification and the optimization steps are carried out by independent pieces of software integrated together, allowing the user to take advantage of state-of-the-art solvers and the computational speed-up granted by the MLC step.

5.1 Building the Classification Model

In our approach, we assume it is possible to access some historical or simulated data on instances of which a problem P is related to. We furthermore assume a reference problem P_{ref} exists, which will serve as the basis to heuristically set the values of the blocked variables.

5.1.1 Feature and Label Engineering

The MLC algorithm will need a representation of our problem in terms of features and labels, depending on the instance and the resolution method, and each family of problems requires specific features and labels engineering. In the case of UC, as in our example, if the only source of variation of interest is the demand, one could use directly the difference between the reference demand and the observed demand as predictors in the MLC model. For the labels, one could use directly binary decision variables as the targets of MLC, thus predicting whether some of the labels could need changing value. For problems with many thousands of variables, a variables-to-labels association can be too hard to handle. We can, however, think of labels as clusters of decision variables with some meaningful structure behind, as for instance with decision variables related to a particular unit in the UC case.

5.2 Generating the Sub-problem: How to Block Variables

Once it has been decided how to formulate the problem, the classification algorithm will be trained on the historical data and employed to predict the set of variables to be blocked. Blocking variables in our approach corresponds to mimicking an expert and exploit some knowledge of the problem to solve it. One could think of *SuSPen* as an *evaluation* tool for expert supervised blocking, that is, to compare the performance of decisions are taken by a human agent to

those where an automated (ML) procedure is adopted.

5.2.1 Blocking Variables in Mathematical Programming

Blocking a variable corresponds to turning it into a fixed parameter of the model. In our implementation, for instance, we achieved this by adding constraints of the type $x_j := Value_j^{ref}$ to the original problem.

Let us consider for example binary decision variables as the targets of our supervised procedure. Blocking x_j and x_k will imply retrieving their values from the solution $x^{\star}_{ref}$ to the reference problem P_{ref} and imposing them as new constraints, say $x_j := x_j^{ref} = 1$ and $x_k := x_k^{ref} = 0$. In this case we would end up adding the constant term $c_j x_j + c_k x_k = c_j \times 1 + c_k \times 0$ to the objective function, $z = \sum_{i \neq \{j,k\}, j=1..p} c_i X_i + c_j$. The same applies for all the occurences of x_j and x_k in the constraints, where the columns $[A_j, A_k]$ of the coefficient matrix A will be moved to the right-hand-side and considered as the constants $A_j x_j$ and $A_k x_k$.

Generalizing, denoting by x_B the selection of variables to be blocked and by x_{SP} the variables to be let free to form our sub-problem, we observe that $x_{SP} \cup x_B = x_R \cup x_I$. What was seen above will apply to all the elements of x_B and we thus get:

$$\begin{aligned} \min_{x_{SP}} \quad & c_{SP}^t \, x_{SP} + c_B^t x_B \\ \text{subject to} \quad & A_{SP} x_{SP} = b - A_B x_B, \end{aligned}$$

In the end, to accomplish blocking variables, we only need to be able to either assign an arbitrary value to the specified blocked variables or to add trivial constraints to the problem.

5.2.2 Blocking Variables in Metaheuristics

Unlike in mathematical programming, genetic algorithms and metaheuristics operate as a black-box, handling directly a vector of solutions and an objective function to be evaluated and optimized. In this case, we will simply fix some of the decision variables via dummy variables, by encoding segments of a TSP path as dummy nodes to be treated by the algorithm as a regular node.

6 EXPERIMENTS

In this section we describe our framework for the numerical experiments carried out, including the sources of data and the feature and label engineering for MLC.

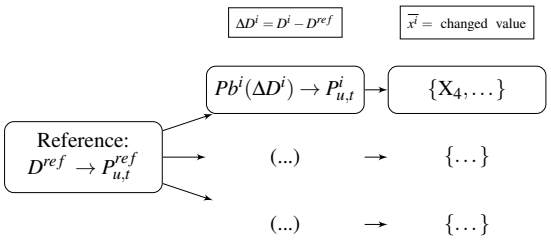

Figure 1: Processing of historical UC data to generate a training dataset.

6.1 Unit Commitment Experiments

We set our reference demand values on a randomly selected calendar day from publicly available demand data[1], for the year 2016. The focus is on the variations of the demand D at the 48 time periods of the day, all the other parameters are held fixed. The labels in our MLC problem were the 1632 binary variables of our MILP formulation, letting free the remaining 576 integer variables and 384 real variables.

We then computed the features (ΔD^j in Figure 1) as the difference between the reference demand D^{ref} and perturbations on it. The MLC step yielded the subset X_{SP} of binary labels affected by the random perturbation, that is, $h_{MLC} : \Delta D \rightarrow X_{SP} \in \{0,1\}^{1632}$. X_{SP} are to be let free in the SP resolution, while all the other binary variables will be blocked to the values found when solving P_{ref}. To test our meta-algorithm, we generated 10 perturbed instances on the reference demand. The 10 instances yielded 10 full problems $\{P\}_{i=1}^{10}$. After running the MLC we produce their relative sub-problems $\{SP\}_{i=1}^{10}$.

The 10 instances were solved in their full formulation P, using the solvers CPLEX[2] and the open source Coin-OR CBC[3] (Forrest, 2012). We then generated the 10 sub-problems SP associated to each new instance and compared the results in terms of relative resolution times and objective values. The results are reported in Table 1.

We observed the SP generation caused a loss of about 2% in terms of objective function, that is, at the end of the minimization the objective value was on average 2.24% greater (i.e. worse) then the true optimal value. As for resolution times, we observed average speed-ups respectively in the order of $\times 5$ and $\times 6.1$

[1] http://www2.nationalgrid.com/uk/Industry-information/Electricity-transmission-operational-data/Data-Explorer/

[2] IBM ILOG CPLEX Optimizer. Version 12.6.3.0. https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

[3] CBC MILP Solver. Version: 2.9.7. https://projects.coin-or.org/Cbc

Table 1: *SuSPen* on the Unit Commitment problem.

Measure	**Speed-up** *SP* is [×*times*] faster			**Objective Loss**∗ (*SP* is worse by [*Loss%*])
	CPLEX †	CPLEX ‡	CBC †	
Min	×1.8	×2.1	×3.3	+2.10%
Average	×5.0	×6.1	×11.9	+2.24%
Max	×8.3	×10.7	×28.7	+2.38%

∗ : CPLEX and CBC converged to the same objective value on all instances.
† : Default values for solver.
‡ : Limited pre-solver, no multi-threading, no dynamic search

when using CPLEX with respectively default parameters and when "tuned down" to limit its heuristic powers. When running CBC the gains were more relevant, on average twice as much as with CPLEX.

6.2 TSP Experiments

We picked a standard problem (*Berlin52*) from the TSPLIB[4]. We selected 10% of nodes at random from the TSP instance *berlin52* and perturbed the weights of the nodes within a certain radius (50% of the highest distance between any two nodes) by a factor decreasing the further apart the nodes are. That is, once a node is picked, the weights within a certain distance are scaled up. If contiguous nodes are selected, then the perturbations will sum up. We produced a dataset of 500 entries, each with comprising the distance matrix and the relative optimal path computed with the Concorde TSP solver. The difference matrix between the reference distance matrix and the new instance matrix form the features. For the labels, we started from the reference optimal solution $x^{\star}_{ref}$, taken to be the actual optimum of *berlin52*. For each entry in the training data, the corresponding optimal TSP path was compared with $x^{\star}_{ref}$, extracting the nodes that were affected by the perturbation. The target prediction vectors contains $\{0,1\}^{52}$ binary labels, or flags, indicating whether the modifications in the weights of the graph resulted in the optimal solution to change at each node.

For example, if a solution contained:

$$(\cdots \rightarrow 22 \rightarrow 31 \rightarrow \boxed{35} \rightarrow \boxed{12} \rightarrow 14 \ldots),$$

but in $x^{\star}_{ref}$ we observed:

$$(\cdots \rightarrow 22 \rightarrow 31 \rightarrow \boxed{12} \rightarrow \boxed{35} \rightarrow 14 \ldots),$$

then our binary labels vector will take value "1" for label "12" and "35" and zero for the others.

On average, for the optimal solutions of the perturbed graphs, 17.2 nodes out of 52 turned out to be affected. On average the predicted label vector should contain 17 non-zero elements, indicating those that

[4]http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/

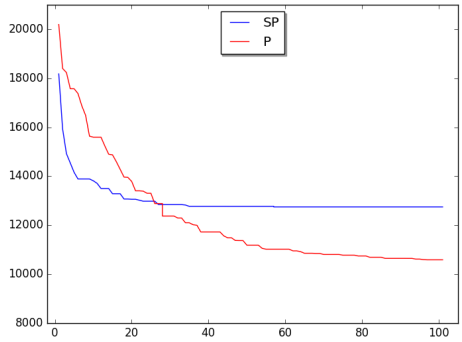

Figure 2: Evolution of the objective function over the 200000 iterations (%-completed *vs* fitness).

deserve the attention of the optimizer and need be re-optimized. After the prediction is obtained, we map these labels back to the $x^{\star}_{ref}$ and find the subsequences of solution not affected by the perturbations. These subsequences are the "blocked" variables in our *SP*, that is, the portions of $x^{\star}_{ref}$ we assume are still good for our *SP*. We encode these subsequences as "fake nodes" deleting from the problem the blocked nodes that become invisible to the solver, yielding a problem *SP* smaller than the original *P*.

Figure 2 reports the typical behaviour registered with our approach: solving *SP* allows us to obtain a good solution rapidly at the cost of some bias (related to the amount of variation in the particular instance). In the case of TSP, the underlying assumption is that after some perturbation is added to a reference instance, the original solution is a good starting point. This is not always the case, and the long-term difference between *SP* and *P* can diverge sensibly, as when we block some variables we inevitably introduce some error. It would be interesting to further expand this approach by freeing previously blocked variables, allowing *SP* to continue diving towards a better solution (see the Conclusion for more on this topic).

7 CONCLUSION

This work proposed a meta-algorithm to speed-up the resolution of recurrent combinatorial problems based on available historical data, working under a *time budget*. Although our approach is highly likely to result in sub-optimal solutions, we have shown that it can yield good approximate solutions within a limited time window. It is also interesting both for exact approaches based on mathematical programming (MILP) and approximate stochastic algorithms treated as black-box solvers. Avenues for future re-

search includes the extension of our methodology to Mixed Integer Nonlinear Programming (MINLP), where an effective variable selection would possibly grant even more sizeable computational gains. Also of interest is the generalization of the blocking phase to an iterative block/unblock step, thus taking full advantage of the time available for the re-optimization. As seen in Section 6.2, *SuSPen* allows us to dive deeply towards a good objective value. Being able to unblock some nodes after the initial iterations would likely result in allowing the solution to keep improving, after the initial boost. In case some of the time budgeted for re-optimization is available, instead of limiting us to solve a single *SP*, one can consider the option of solving more *SP*'s, eventually in parallel, integrating the information derived from the resolution of previous *SP* into the learning framework. Furthermore, in the case of exact BB-based approaches, we find a natural prospective line of work in the integration of branching rules learned from static frameworks (as seen in Section 2) into our problem-specific approach.

ACKNOWLEDGEMENTS

This research benefited from the support of the "FMJH Program Gaspard Monge in optimization and operation research", and from the support to this program from EDF.

REFERENCES

Achterberg, T., Koch, T., and Martin, A. (2005). Branching rules revisited. *Operations Research Letters*, 33(1):42–54.

Alvarez, A. M., Louveaux, Q., and Wehenkel, L. (2017). A machine learning-based approximation of strong branching. *INFORMS Journal on Computing*, 29(1):185–195.

Applegate, D., Bixby, R., Chvatal, V., and Cook, W. (2006). Concorde tsp solver.

Applegate, D. L., Bixby, R. E., Chvatal, V., and Cook, W. J. (2011). *The traveling salesman problem: a computational study*. Princeton university press.

Auger, A. and Doerr, B. (2011). *Theory of randomized search heuristics: Foundations and recent developments*, volume 1. World Scientific.

Basso, S., Ceselli, A., and Tettamanzi, A. (2017). Random sampling and machine learning to understand good decompositions. Technical Report 2434/487931, University of Milan.

Bonami, P., Lodi, A., and Zarpellon, G. (2018). Learning a classification of mixed-integer quadratic programming problems. In van Hoeve, W.-J., editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 595–604, Cham. Springer International Publishing.

Carrión, M. and Arroyo, J. M. (2006). A computationally efficient mixed-integer linear formulation for the thermal unit commitment problem. *IEEE Transactions on power systems*, 21(3):1371–1378.

Cornelusse, B., Vignal, G., Defourny, B., and Wehenkel, L. (2009). Supervised learning of intra-daily recourse strategies for generation management under uncertainties. In *PowerTech, 2009 IEEE Bucharest*, pages 1–8.

Dai, H., Khalil, E. B., Zhang, Y., Dilkina, B., and Song, L. (2017). Learning combinatorial optimization algorithms over graphs. *arXiv preprint arXiv:1704.01665*.

Dantzig, G., Fulkerson, R., and Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410.

Dembczynski, K. J., Cheng, W., and Hüllermeier, E. (2010). Bayes optimal multilabel classification via probabilistic classifier chains. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 279–286.

Fischetti, M. and Fraccaro, M. (2018). Machine learning meets mathematical optimization to predict the optimal production of offshore wind parks. *Computers and Operations Research*.

Forrest, J. (2012). Cbc (coin-or branch and cut) open-source mixed integer programming solver, 2012. *URL https://projects.coin-or.org/Cbc*.

Gendreau, M. and Potvin, J.-Y. (2010). *Handbook of metaheuristics*, volume 2. Springer.

He, H., Daume III, H., and Eisner, J. M. (2014). Learning to search in branch and bound algorithms. In *Advances in Neural Information Processing Systems*, pages 3293–3301.

Khalil, E. B., Le Bodic, P., Song, L., Nemhauser, G., and Dilkina, B. (2016). Learning to branch in mixed integer programming. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*.

Kruber, M., Lübbecke, M. E., and Parmentier, A. (2017). Learning when to use a decomposition. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 202–210. Springer.

Larsen, E., Lachapelle, S., Bengio, Y., Frejinger, E., Lacoste-Julien, S., and Lodi, A. (2018). Predicting solution summaries to integer linear programs under imperfect information with machine learning. *arXiv preprint arXiv:1807.11876*.

Lodi, A. and Zarpellon, G. (2017). On learning and branching: a survey. *TOP*, pages 1–30.

Loubière, P., Jourdan, A., Siarry, P., and Chelouah, R. (2017). A sensitivity analysis method aimed at enhancing the metaheuristics for continuous optimization. *Artificial Intelligence Review*, pages 1–23.

Mossina, L. and Rachelson, E. (2017). Naive bayes classification for subset selection. *arXiv preprint arXiv:1707.06142*.

Padhy, N. P. (2004). Unit commitment-a bibliographi-

cal survey. *IEEE Transactions on power systems*, 19(2):1196–1205.

Rachelson, E., Abbes, A. B., and Diemer, S. (2010). Combining mixed integer programming and supervised learning for fast re-planning. In *2010 22nd IEEE International Conference on Tools with Artificial Intelligence*, volume 2, pages 63–70.

Read, J., Pfahringer, B., Holmes, G., and Frank, E. (2011). Classifier chains for multi-label classification. *Machine learning*, 85(3):333–359.

Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M., and Tarantola, S. (2008). *Global sensitivity analysis: the primer*. John Wiley & Sons.

Schrijver, A. (1998). *Theory of linear and integer programming*. John Wiley & Sons.

Tsoumakas, G. and Vlahavas, I. (2007). Random k-labelsets: an ensemble method for multilabel classification. In *European Conference on Machine Learning*, pages 406–417. Springer.

Vanderbeck, F. and Wolsey, L. A. (2010). *Reformulation and Decomposition of Integer Programs*, pages 431–502. Springer Berlin Heidelberg, Berlin, Heidelberg.

Williams, H. P. (2013). *Model building in mathematical programming*. John Wiley & Sons.

Zhang, M. L. and Zhou, Z. H. (2014). A review on multi-label learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 26(8):1819–1837.

Zupanič, D. (1999). Values suggestion in mixed integer programming by machine learning algorithm. *Electronic Notes in discrete Mathematics*, 1:74–83.