Continuous Latent Search for Combinatorial Optimization

Sergey Bartunov, Vinod Nair, Peter Battaglia, Timothy Lillicrap
DeepMind
London, UK
{bartunov, vinair, battaglia, countzero}@google.com

Abstract

Combinatorial optimization problems are notoriously hard because they often require enumeration of the exponentially large solution space. Both classical solving techniques and machine learning-based approaches usually address combinatorial optimization problems by manipulating solutions in their original discrete form. In contrast, we propose a framework that consists of reparametrizing the original discrete solution space into a continuous latent space in which the problem can be (approximately) solved by running continuous optimization methods. We achieve this by learning a surrogate function that is shaped to correlate with the original objective when the latent solution is decoded back to the original solution space. We show that this approach can learn efficient solution strategies and is useful as a primal heuristic inside the widely-used open-source solver SCIP.

1 Introduction

Combinatorial optimization encompasses a class of optimization problems with at least a subset of variables taking values from a discrete set. Without any knowledge about the nature of underlying objective function or constraints, the optimization becomes NP-hard and to find the exact optimum one has to resort to exhaustive search in the solution space which grows in size exponentially as the dimensionality of the problem increases. Fortunately, for many practical applications the exact optimal solution is not necessary, so one is usually concerned with finding a good solution under certain computation limits.

A vast amount of research has been devoted to more efficient search algorithms and heuristics that exploit the problem structure. One way of making progress on this front is to develop general-purpose algorithms that make assumptions about the functional form of objective and constraints. A good example is the class of mixed-integer programming (MIP) problems [1] on which we focus in this paper where both objective and constraints have are linear in each of the variables. For MIP a powerful search method called branch-and-bound has been proposed that utilizes a powerful connection between the original MIP problem and its LP-relaxation.

Another path to tackling combinatorial optimization problems that has actualized with the rapid development of machine learning is to, indeed, *learn* a search algorithm or a search-assisting heuristic that is tailored to a particular distribution of problems in mind. In this paper, we take this path and propose a method for learning efficient approximate search algorithms for combinatorial optimization problems. Unlike many previous attempts, we do not train a model that directly outputs a solution. Instead, we resort to *latent-space optimization* and learn useful embedding of the discrete solution space that allow us to improve solutions by optimizing their continuous representation. This allows us to exploit the problem structure such as sparsity patterns in constraints and potentially address the curse of dimensionality inherent in MIP solving by detecting correlations between variables. We

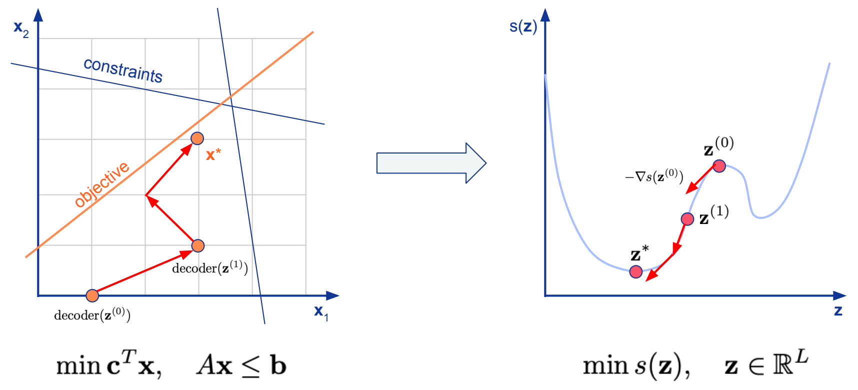


Figure 1: Graphical illustration of Continuous Latent Search.

experimentally show that our model is complementary to general-purpose solvers and when used as a primal heuristic can significantly speed up branch-and-bound search.

2 Continuous Latent Search

The general idea behind *Continuous Latent Search* or CLS is to translate the problem of solving an optimization problem over a *discrete* or *mixed-integer* variable $\mathbf{x}$ into minimizing a *differentiable surrogate* $s(\mathbf{z})$ over a *continuous* variable $\mathbf{z} \in \mathcal{Z} \subseteq \mathbb{R}^L$ from which the discrete solution can be recovered using a decoder $g(\mathbf{z})$. Another way to think about this is that we want to learn a solution improvement operator similar to the gradient that is not available in discrete problems. This scheme is outlined on Figure 1.

The surrogate $s(\cdot)$ and the decoder $p(\cdot|\mathbf{z})$ are trained on a certain distribution of problems such that an optimization procedure such as gradient descent in the latent space leads to an improvement on the original objective $f(\cdot)$ and, ideally, in the end match the optimal solution $\mathbf{x}^*$. While there could be different ways to achieve this, in this work we specifically use recent advances in gradient-based meta-learning [2, 3] because they allow end-to-end training of the whole system.

We first start by sampling a random latent solution $\mathbf{z}^{(0)} \sim \mathcal{N}(\mathbf{0}, I)$ and then unroll K steps of gradient descent on the surrogate producing $\mathbf{z}^{(k+1)} = \mathbf{z}^{(k)} - \eta^{(k)} \nabla s_\theta(\mathbf{z}^{(k)})$. Denoting the problem description as $\tau = \{\mathbf{c}, A, \mathbf{b}\}$, we define the learning loss as

$$\mathbb{E}_{p(\tau)} \sum_{k=1}^{K} \mathbb{E}_{\mathbf{x} \sim p_{\tau,\theta}(\cdot|\mathbf{z}^{(k)})} \left[\underbrace{\gamma(\mathbf{x})}_{\text{objective}} + \alpha \underbrace{||\max\{0, A\mathbf{x} - \mathbf{b}\}||^2}_{\text{feasibility}} + \beta \underbrace{\max\{0, s_{\tau,\theta}(\mathbf{z}^*) - s_{\tau,\theta}(\mathbf{z}^{(k)})\}}_{\text{latent supervision}} \right]. \tag{1}$$

This loss is optimized with respect to parameters of the surrogate θ, also taking into accound second-order derivatives incurred by backpropagation through the inner gradient optimziation. Further we omit the functional dependence on τ for notational clarity. For definition of $\gamma(\cdot)$ please refer to equation (3), one can think of this function as a re-scaled version of the original objective $f(\mathbf{x}) = \mathbf{c}^T\mathbf{x}$ which takes values only in $[0, 1]$ interval and by definition outputs 1, i.e. its maximal value, when $\mathbf{x}$ is infeasible.

As one can note, our training loss does not contain explicit reconstruction-based supervision with the optimal solution(s) $\mathbf{x}^*$. While adding such a learning signal does look natural, it is in fact not trivial to handle because it essentially requires the model to amortize the NP-hard process of solving the original problem. Moreover, in many combinatorial problems the hamming loss or other losses decomposable across dimensions of $\mathbf{x}$ might not behave adequately as the discrepancy in just one bit can render the whole solution infeasible. So, instead, we employ a much more nuanced way of training the model.

First, we encourage decoded solutions to decrease on the original objective and also to enter the feasible region as soon as possible (see (3) for details on relationship between $\gamma(\mathbf{x})$ and $f(\mathbf{x})$). Secondly, we do optionally use information about the optimal solution (or another good solution that can serve as a target), by encoding it into the latent space and ensuring that the surrogate value of its embedding $\mathbf{z}^* \approx \arg\max_{\mathbf{z}} p(\mathbf{x}^*|\mathbf{z})$ is less than for any other point in the optimization trajectory. We refer to this as *latent supervision* and experimentally we found it quite useful for improving the model's results.

Another important detail is the implementation of the learnable surrogate and the decoder which is crucial for success of the method. For MIPs we use a well-proven graph neural network (GNN) construction [4] on a bipartie MIP representation where variables and constraints form nodes in a graph and edges connect variables and the constraints in which they participate [5]. In addition to the basic problem description τ we also provide the solution of the linear programming relaxation to the model as it is often very important for MIP solving. The described GNN architecture is used for jointly computing the surrogate value and modelling the decoding distribution: the latent variable $\mathbf{z}$ is passed to the network as a global input and after a few rounds of message passing, we use the global output to compute the surrogate value through a linear projection and the variable node embeddings to produce logits for the decoder.

The decoder $p(\mathbf{x}|\mathbf{z}) = \prod_{i=1}^{N} p(x_i|\mathbf{z})$ assumes conditionally-independent factorization and models each discrete dimension with a Categorical distribution. To simplify implementation, we assumed that all variables are binary, but it hardly presents any conceptual limitation to the model. In addition to (1), we also train the encoder using the following loss:

$$-\sum_{k=1}^{K} \log p_\theta(\mathbf{x}^{(k)}|g_\theta(\mathbf{x}^{(k)})) - \log p_\theta(\mathbf{x}^*|g_\theta(\mathbf{x}^*)), \quad g_\theta(\mathbf{x}) \approx \arg\max_{\mathbf{z}} \log p_\theta(\mathbf{x}|\mathbf{z}), \tag{2}$$

where $\mathbf{x}^{(0)}$ is just a random solution and all the subsequent $\mathbf{x}^{(k)}$ are decoded from the intermediate latent solutions $\mathbf{z}^{(k)}$. We describe the decoder $g(\mathbf{x})$ in more detail in Appendix A.2.

3 Experiments

We evaluate CLS on three combinatorial problems established by [5]: Set Cover, Combinatorial Auction and Maximum Independent Set problems. We had to exclude the Capacitated Facility Location problem due to presence of continuous variables which are not supported in the current implementation. In order to best demonstrate the practical importance of our work, we integrate CLS into state-of-the-art open-source solver SCIP [6] as a *primal heuristic* whose purpose is to provide or improve an *incumbent solution*. CLS was trained on training instances from each dataset using Adam optimizer with learning rate 10^{-5} and default values for other hyperparameters and batches of just a single instance.

Primal heuristics are well known to be one of the most important solver components because they assist the branch-and-bound search by pruning suboptimal branches and thus may drastically decrease the solving time. To put CLS in the most practically relevant setting, we measure the relative impact of the CLS heuristic estimated *on top of other SCIP heuristics*. We only exclude the RENS heuristic [7] that is based on fixing a subset of variables and recursively solving the resulting sub-problem, because it is unclear how to fairly account for such functionality when measuring convergence speed in number of nodes and because such meta-heuristics address orthogonal research questions to the ones we are concerned in this paper. For simplicity of integration with SCIP, we only call CLS at the root node of the search tree using random initialization, however, initializing the search with an incumbent will likely improve results. In order to best assess the impact of primal heuristics we also disabled other kinds of solver plugins such as cuts and presolves.

When evaluating primal heuristics we are ultimately interested in the speed of objective improvement. [8] proposed a way of measuring via *primal gap* and *primal integral* metrics. Denoting $\tilde{\mathbf{x}}(t)$ as incumbent solution available at time t, primal gap is computed as

$$\gamma(\tilde{\mathbf{x}}) = \begin{cases} 1, & \text{if } f(\tilde{\mathbf{x}})f(\mathbf{x}^*) < 0, \\ 0, & \text{if } f(\tilde{\mathbf{x}}) = f(\mathbf{x}^*) = 0, \\ \frac{|f(\tilde{\mathbf{x}}) - f(\mathbf{x}^*)|}{\max\{|f(\tilde{\mathbf{x}})|, |f(\mathbf{x}^*)|\}}, & \text{else,} \end{cases} \qquad p(t) = \begin{cases} 1, & \text{if no incumbent} \\ \gamma(\tilde{\mathbf{x}}(t)), & \text{else.} \end{cases} \tag{3}$$

And then primal integral is defined as area under the primal gap curve: $P(T) = \int_{t=1}^{T} p(t)dt$.

Measuring wall-clock time in a fair way is arguably difficult in systems involving machine learning models due to a number of contributing factors such as usage of specialized hardware and the great potential to parallelization, so further we refer to time as the number of processed nodes in a search tree.

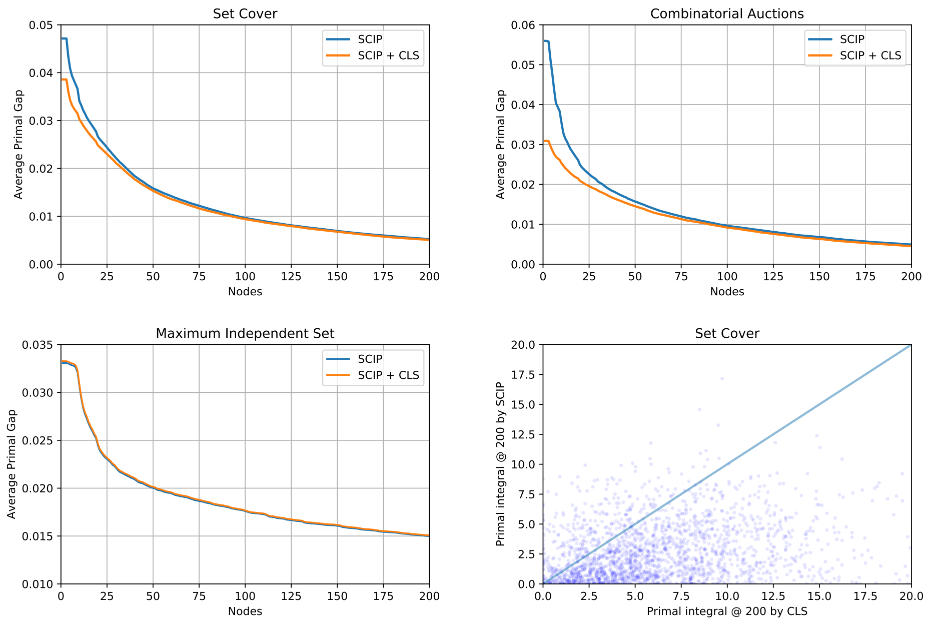


Figure 2: Average primal gap curves on different datasets computed on test instances. Right-bottom: SCIP vs CLS alone comparison in terms of primal gap across all test Set Cover instances.

The quantitative comparison between SCIP and SCIP enhanced with CLS can be found on Figure 2. In this experiment, we did not test generalization to larger instances than seen during training and report results for test instances of the same complexity. One can note faster convergence provided by CLS to the optimal solution on two of the three considered datasets, with neutral impact on Maximum Independent Set problems. On Set Cover, latent supervision did not bring significant improvement, so we present results with $\beta = 0$ on this dataset, but on Combinatorial Auctions it was crucial. It is interesting to see that even when SCIP is used with CLS *alone* (i.e. with other heuristics not being able to improve solutions found by CLS or branch-and-bound search), some instances can be solved significantly faster than with the rich ensemble of SCIP heuristics. We invite readers to the Appendix for additional experimental and implementation details.

4 Related work

Combinatorial optimization has recently attracted significant attention in machine learning, being primarily addressed by reinforcement learning methods [9, 10, 11, 12]. While this approach is very general, reinforcement learning even on moderately large problems is challenging. A more practical approach consists of integration with classical solvers and using supervision from various heuristics [5, 13]. Our work is similar in this respect, but focuses on primal instead of branching heuristics.

The general idea of performing optimization in the latent space has a number of incarnations in the literature. In reinforcement learning it is found useful to perform planning in the latent space of action sequences or goals [14, 15, 16]. [17, 18] use latent space learned by variational auto-encoders to explore and optimize over rich combinatorial structures. A similar idea is explored in [19] where an objective function acting on a latent space is learned to be specifically useful for gradient-based optimization of desired properties in the decoded, potentially discrete object. The recent work [20] also investigates latent-space optimization and proposes a method for iterative re-training of the latent space-producing VAE to better facilitate optimization. Finally, there is an array of research specifically interested in applying latent-space optimization to black-box problems [21, 22]. CLS is different in the joint training of the latent space-inducing model and the optimization procedure which, we believe, is important. Otherwise, the general smoothness properties of the continuous latent space may be not enough to efficiently explore and to learn both useful and easy to optimize surrogate (i.e. convex, Lipschitz continuous etc).

5 Conclusion

We presented a novel approach for learning approximate combinatorial solvers based on latent-space optimization. While we obtained some interesting empirical results, there is plenty of research directions to explore. First, the latent supervision can and probably should be extended to account for many available solutions, not just for the (near-)optimal one. One could also imagine more elaborate supervision scenarios, for example, distillation of powerful but computationally expensive heuristics. Second, extending the basic gradient-based optimization to a more complicated, multi-particle algorithms that make connections to sampling can potentially enable much more interesting search strategies such as exploring different local minima and more efficient parallel search (see the Appendix B.1 for some initial results). Perhaps, departure from the fully end-to-end differentiable scheme could extend the existing relatively short roll-outs to much longer ones, likely necessary in more difficult optimization problems. Finally, more experimentation on larger-scale real-world datasets that possess enough structure should really demonstrate the benefit of continuous latent-space optimization.

References

[1] L.A. Wolsey. *Integer Programming*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 1998.

[2] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

[3] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, pages 3981–3989, 2016.

[4] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[5] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 15580–15592, 2019.

[6] Tobias Achterberg. Scip: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

[7] Timo Berthold. Rens–the optimal rounding. 2012.

[8] Timo Berthold. Measuring the impact of primal heuristics. *Operations Research Letters*, 41(6):611–614, 2013.

[9] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[10] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6348–6358, 2017.

[11] Wei Zhang and Thomas G Dietterich. Solving combinatorial optimization tasks by reinforcement learning: A general methodology applied to resource-constrained scheduling. *Journal of Artificial Intelligence Reseach*, 1:1–38, 2000.

[12] Luca M Gambardella and Marco Dorigo. Ant-q: A reinforcement learning approach to the traveling salesman problem. In *Machine Learning Proceedings 1995*, pages 252–260. Elsevier, 1995.

[13] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems*, pages 539–548, 2018.

[14] Masataro Asai and Alex Fukunaga. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. *arXiv preprint arXiv:1705.00154*, 2017.

[15] Leonardo Amado, Ramon Fraga Pereira, Joao Aires, Mauricio Magnaguagno, Roger Granada, and Felipe Meneguzzi. Goal recognition in latent space. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2018.

[16] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks. *arXiv preprint arXiv:1804.00645*, 2018.

[17] Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

[18] Jonas Mueller, David Gifford, and Tommi Jaakkola. Sequence to better sequence: continuous revision of combinatorial structures. In *International Conference on Machine Learning*, pages 2536–2544, 2017.

[19] Jesse Engel, Matthew Hoffman, and Adam Roberts. Latent constraints: Learning to generate conditionally from unconditional generative models. *arXiv preprint arXiv:1711.05772*, 2017.

[20] Austin Tripp, Erik Daxberger, and José Miguel Hernández-Lobato. Sample-efficient optimization in the latent space of deep generative models via weighted retraining. *arXiv preprint arXiv:2006.09191*, 2020.

[21] Xiaoyu Lu, Javier Gonzalez, Zhenwen Dai, and Neil Lawrence. Structured variationally auto-encoded optimization. In *International Conference on Machine Learning*, pages 3267–3275, 2018.

[22] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *Advances in neural information processing systems*, pages 7816–7827, 2018.

[23] Yurii E Nesterov. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. akad. nauk Sssr*, volume 269, pages 543–547, 1983.

[24] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

A Additional model details

The GNN used in the surrogate architecture consists of 4 layers of message-passing with distinct weights. Nodes and edges have embeddings of size 128 and 64 units respectively, the global embedding size if 64. We set the dimensionality of the latent variable to 128 which is linearly projected to 64 units as it is processed by the GNN.

Since we rely on gradients with respect to the network input, we found it important to use well-behaved bounded activation functions and so the only used nonlinearity in the GNN is *tanh*.

The decoder $p_\theta(\mathbf{x}|\mathbf{z})$ is based on the same GNN as the surrogate function, so both the surrogate value and the decoded discrete solution can be computed in one forward pass.

A.1 Gradient search

We experimented with many different gradient-based optimization procedures and while even the standard gradient descent with fixed (but trainable) learning rate schedule performed quite well, in our experiments we rely on a learned optimizer with a Nesterov momentum [23]:

$$\begin{aligned}
&\hat{\mathbf{z}}^{(k)} = \text{project}(\mathbf{z}^{(k)} + \psi^{(k)}\mathbf{v}^{(k-1)}), \quad \mathbf{v}^{(k)} = \psi^{(k)}\mathbf{v}^{(k-1)} - \eta^{(k)}\nabla s(\hat{\mathbf{z}}^{(k)}) \\
&\mathbf{z}^{(k)} = \text{project}(\mathbf{v}^{(k)}).
\end{aligned} \tag{4}$$

The learning rate $\eta^{(k)}$ and the momentum weight $\psi^{(k)}$ are predicted by LSTM with 256 hidden units that on each step takes the current value of the surrogate $s(\mathbf{z}^{(k)})$ and its squared gradient $[\nabla_{\mathbf{z}} s(\mathbf{z}^{(k)})]^2$.

We also found it occasionally helpful to project iterates of the latent search into the $[-5, +5]^L$ box. During training we use $K = 16$ steps of optimization.

A.2 Encoder

As we mention in the main text, our model does not contain an explicit encoder. Instead, we implement the implicit approximate encoder which we denote as $g_\theta(\mathbf{x})$ as a truncated optimization procedure similar to (4) and only different in the initialization at the zero vector and the use of the decoder likelihood as the optimization objective. We use just 4 optimization steps when encoding.

The role of the encoder in CLS is two-fold. First, the encoder training loss (2) ensures that the encoder indeed preserves information about the discrete solution via minimization of the reconstruction error. Secondly, it also prevents the latent space from collapsing, i.e. when all latent solutions $\mathbf{z}$ are decoded into the same discrete solution, by ensuring that any random solution ($\mathbf{x}^{(0)}$ in (2)) can be adequately auto-encoded. This is similar to the standard entropy regularization found in many auto-encoder models but which is intractable to optimize directly in the framework of CLS.

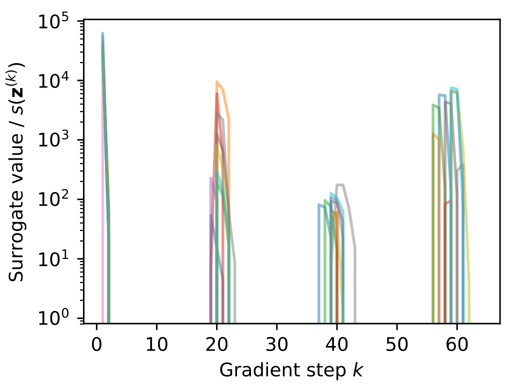
A.3 Gradient estimation

The first two terms in (1) are non-differentiable and we rely on Gumbel-Softmax Straight-Through technique [24] for obtaining low-variance gradient estimators with respect to the corresponding terms. No temperature annealing was found necessary and we conducted our experiments with the temperature parameter set to 0.5. We used single sample from the decoder at each step of the gradient descent for estimating gradients.

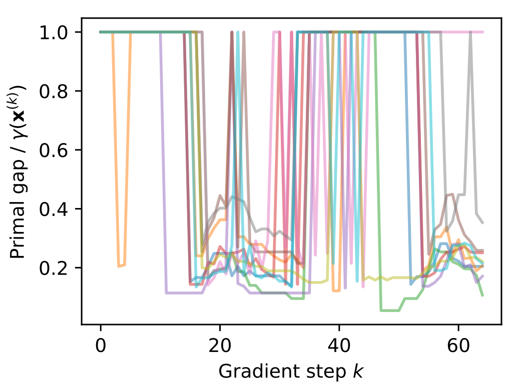
B Additional experimental details

B.1 Latent trajectories behavior

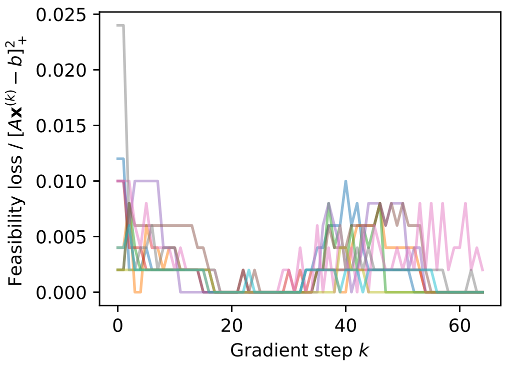
Perhaps, the intrinsic behaviour of CLS is as interesting as its direct evaluation inside a solver. Here we provide a detailed demonstration of the latent particles evolution in CLS as the optimization unrolls, on the example of the Set Cover model.



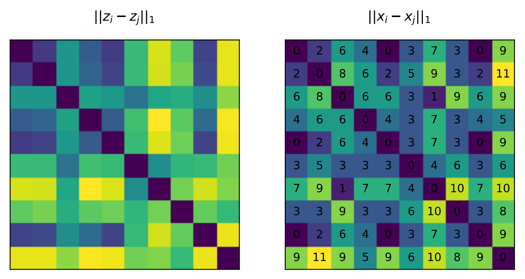
(a) Surrogate values.



(b) Primal gap.



(c) Mean constraint violation.



(d) Left: pairwise L_1 distances in the latent space, right: pairwise L_1 instances in the discrete space.

Figure 3: Evolution of 10 latent solutions under optimization of the same Set Cover instance.

While we trained CLS with only 16 optimization steps, at the test time we take 64 steps. As we show, this not only allows the optimizer to reach a potentially lower value of the surrogate, but, in some cases, also escape local minima, hitting a better objective of the original MIP.

Figure 3 visualizes changes in different important quantities during optimization. Different colors represent different independent runs of CLS initialized randomly. One can see, that depending on the initialization, latent trajectories pass different regions of the latent space, some hitting better discrete solutions than others and, as we mentioned, reaching better objective values when escaping local minima. Quite importantly, when measuring pairwise distances between latent solutions corresponding to the best found objective values, difference in latent representations does translate into different decoded discrete solutions (see Figure 3d). This is an evidence that CLS indeed learns an interesting representation that partially preserves distances in the discrete space, however, discarding some, presumably unimportant, information.

Finally, Figure 4 shows evolution of the decoded solution probabilities corresponding to one of the trajectories.

B.2 Utilizing multiple trajectories in the evaluation

As discussed in the previous section, there is a benefit from using multiple optimization trajectories from different initializations. In our evaluation, we ran 16 parallel trajectories and picked the overall best solution to report as found by the primal heuristic.

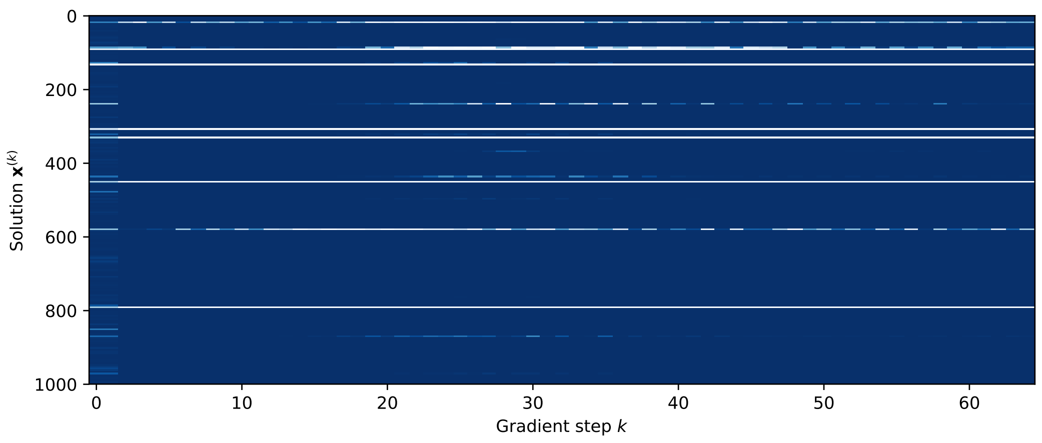


Figure 4: Evolution of the decoded solutions on one of the Set Cover instances. Each of the 1000 rows is probability of inclusion of the corresponding set in the solution.