Learning to Search via Self-Imitation with Application to Risk-Aware Planning

Jialin Song*
Caltech
jssong@caltech.edu

Ravi Lanka*
JPL
ravi.kiran@jpl.nasa.gov

Albert Zhao
Caltech
azzhao@caltech.edu

Yisong Yue
Caltech
yyue@caltech.edu

Masahiro Ono
JPL
masahiro.ono@jpl.nasa.gov

Abstract

We study the problem of learning a good local search policy for solving combinatorial optimization problems such as mixed integer linear programs. To do so, we propose the self-imitation learning setting, which builds upon imitation learning in two ways. First, self-imitation uses feedback provided by retroactive analysis of demonstrated search traces. Second, the policy can learn from its own decisions and mistakes without requiring repeated feedback from an external expert. Combined, these two properties allow our approach to iteratively scale up to larger problem sizes than the initial problem size for which expert demonstrations were provided. We showcase the effectiveness of our approach on the challenging problem of risk-aware planning.

1 Introduction

Many complex prediction and decision-making tasks require solving challenging optimization problems such as mixed integer linear programs (MILPs) [1]. Examples include MAP inference in graphical models [2, 3, 4], path planning [5, 6, 7], transportation scheduling [8] and many others.

Since solving MILPs is NP-hard, one typically resorts to branch-and-bound [9] combined with local search heuristics for making branching and bounding decisions. One key design question in branch-and-bound (and virtually all other local search heuristics) is how to prioritize the search space, e.g., prioritize which integer variables to set first. The conventional approach is to manually design such heuristics to exploit specific structural assumptions (cf. [10, 11]). However, this conventional approach is labor intensive and relies on human experts developing a strong understanding of structural properties of some class of MILPs.

In this paper, we take a learning approach to finding an effective search heuristic over a (focused) distribution of MILP instances. We build upon the imitation learning paradigm [12, 13] and propose the self-imitation learning approach, where the policy can iteratively learn from its own decisions and mistakes without requiring continuous feedback from an external expert. Our approach improves upon previous imitation approaches for solving MILPs [14] in two major aspects. First, our learning approach allows the trained policy to iteratively refine towards new feasible solutions that may be easier for the policy to find than those in the original training set. Second, by learning from its own decisions, our approach can then train on larger problem instances than contained in the original supervised training set. We also provide a mistake bound

*Equal contribution.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

of our policy in a restricted setting of the general problem. Finally, we showcase the practicality of our approach on an application of risk-aware path planning that is built on top of MILP-based path planning [5] and chance-constrained optimization [15], where we demonstrate improvements upon prior imitation learning work [14] as well as commercial solvers such as Gurobi.

2 Related Work

Imitation learning is an increasingly popular paradigm, whereby a policy is trained to mimic the decision-making of a expert or oracle [13, 12]. Existing imitation learning approaches for solving mixed integer linear programs [14] do not iteratively learn from their own mistakes, and are restricted to solving fixed-size problem instances. In contrast, our self-imitation approach is able to learn from its own mistakes as well as train on larger problem instances than contained in the original supervised training set.

Other work on using machine learning to learn a branch-and-bound policy focused on designing a good feature representation for the variable selection procedure [16]. In contrast, we adopt the imitation learning reduction paradigm [12, 14], so that we can leverage powerful function classes such as deep learning in order to avoid requiring a heavily engineered feature representation.

Our self-imitation approach bears some affinity to other imitation learning approaches that aim to exceed the performance of the oracle teacher [17]. One key difference is that we are effectively using self-imitation as a form of transfer learning by learning to solve problem instances of increasing size in the absence of expert demonstrations.

3 Problem Setting & Preliminaries

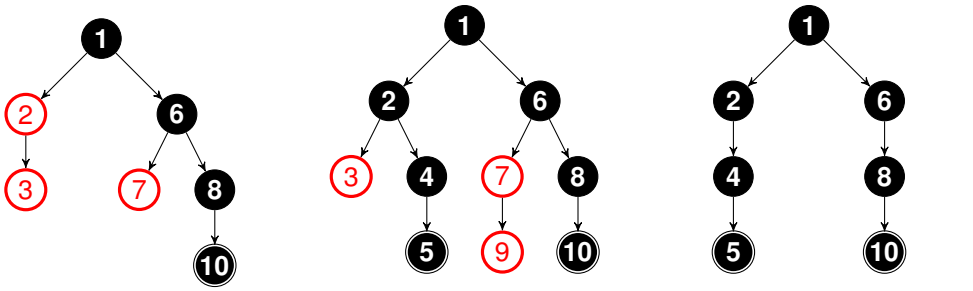
Learning a Search Policy. We consider the following search problem: given a problem instance x (which includes the initial state s_0) drawn from some distribution $\mathscr{D}$, an agent follows a policy $\pi \in \Pi$ which chooses an action $a \in A$ at each non-terminal state s, i.e., $\pi(x,s) \to a$. The agent then transitions to a new state s' after action a. The search ends once the agent arrives at a terminal state. The objective is to train a policy that quickly reaches a terminal state.

Mixed Integer Linear Programs. We focus on policies that solve mixed integer linear programs (MILPs). MILPs are optimization problems of the following form:

$$\begin{array}{ll} \text{minimize} & c^t z \\ \text{subject to} & Az \preceq b \\ & z_i \in \mathbb{Z}, i \in I \end{array}$$

A popular procedure used to solve MILPs is branch-and-bound, which can be visualized as a tree search scheme, such as depicted in Figure 1a. Each node in the search tree corresponds to a state s defined above. Each internal node of the search tree is a partial setting of the integer variables, and the remaining variables are allowed to be real-valued, yielding an LP-relaxation of the MILP. If the solution of the LP-relaxation happens to satisfy all the integer constraints, we have discovered a new feasible solution. Otherwise, we choose a variable z_i in the solution with value z_i^0, which should be an integer but is not, and generate two branches corresponding to constraints $z_i \le \lfloor z_i^0 \rfloor$ and $z_i \ge \lceil z_i^0 \rceil$, respectively, and decide one branch to continue searching. At each new node, the LP-relaxation and branching action are performed again. The search continues until some termination criterion is met, and the best feasible solution found is returned.

Imitation Learning. We build upon the imitation learning paradigm to learn a good policy. In imitation learning, there is typically an expert policy π_{expert} from which a trained policy acquires feedback on its actions. The expert feedback are actions that π_{expert} would have taken in those states encountered by the trained policy. We will build upon the DAgger algorithm [12], where the trained policy iteratively learns to make action decisions more similar to those of an expert. Central to DAgger and other imitation learning algorithm is the availability of π_{expert}, which can be a human expert or another (expensive) solver. However, this assumption does not always hold. As an alternative to expert feedback, we show that our trained policy can derive feedback for itself by examining its past decisions, a process we call *self-imitation*.



(a) Expert Search Trace

(b) Trained Policy Search Trace before Self-Imitation

(c) Trained Policy Search Trace after Self-Imitation

Figure 1: An example where the expert trace is a subset of the policy trace (the numbers on the expert trace are inherited from the policy trace; nodes with double circles are terminal states). The data collected via imitation learning and self-imitation learning differ in two aspects. First, there is no clear expert feedback on nodes missing from the expert trace, e.g. node 4. Second, for nodes common in two traces, sometimes the feedback will be different. In this example, our policy has discovered an additional terminal node 5. Self-imitation guides the training of the policy to more efficiently search for node 5 in the cases where it chooses to branch to node 2 over node 6.

4 Self-Imitation Approach

We now describe the self-imitation learning approach for training a MILP search policy. Our approach builds upon the data aggregation imitation learning framework (DAgger) [12, 14]. We present our overall approach in two steps. First, Algorithm 1 describes our core self-imitation approach for learning a search policy to solve a distribution of fixed-size MILP instances. Secondly, Algorithm 2 explains how self-imitation learning can scale up beyond the original problem size.

Core Algorithm. We assume access to an initial dataset of expert demonstrations to help bootstrap the policy. We enter an iterative self-imitation learning phase. At each iteration, we first run the current policy (potentially blended with an exploration policy), until reaching a terminal state. Then, we identify the retrospective optimal path (from the initial state to the terminal state) and collect data on mistakes made with respect to this path. The reasoning about hindsight path $\pi^*(P)$ is what distinguishes our self-imitation approach from conventional imitation learning. In particular, we do not require an expert to repeatedly provide feedback on the roll-out traces. Instead, self-imitation only relies on a (light-weight) algorithm to compute the optimal path retrospectively. For example, in tree search problems, an optimal path is obtained by tracing parent nodes from a solution node, which takes linear time to compute. In Figure 1b, if the policy is a ranking-based one, the collected feedback will contain pairwise preference such as $4 \prec 3$ and $8 \prec 7$ from this specific search trace. We can thus avoid repeatedly querying the expert by training on hindsight optimal paths of traces generated by our policy – hence self-imitation.

Generalized Self-Imitation & Scaling Up. Another major benefit of self-imitation is that it is not constrained by the problem instance size. Thus, one can apply self-imitation to problem of sizes beyond those in the initial dataset consisting of expert demonstrations. Algorithm 2 describes our generalized self-imitation approach that iteratively learns to solve increasingly larger MILP instances using Algorithm 1 as a subroutine. We start by training on demonstrated feasible solutions from some expert (such as Gurobi) on MILP instances of size S_1. If no such expert is available, we can set S_1 small enough such that exhaustive search is tractable.

5 Theoretical Results

Our theoretical analysis aims to compare self-imitation learning with conventional imitation learning. To do so, we analyze the connection between the feedback derived from self-imitation and expert demonstration. Consider the search trace P_1 generated by a trained policy and P_2 generated by an expert policy. If $P_2 \subseteq P_1$, the self-imitation will collect a dataset that aligns better with the objective than one collected using imitation learning, as explained by an example in Figure 1. In other words, consider a policy π' outside of the policy class Π. π' looks at a successful

Algorithm 1: Self-Imitation Policy Learning

Inputs:
N: number of iterations
π_1: initial policy trained by imitating expert traces
α: mixing parameter
D_0: expert traces dataset
$D = D_0$
for $i \leftarrow 1$ **to** N **do**
  train π_i on D
  $\hat{\pi}_i \leftarrow \alpha\pi_i + (1-\alpha)\pi_{explore}$ (optionally explore)
  run $\hat{\pi}_i$ to generate trace P
  compute the retrospective optimal path $\pi^*(P)$
  collect new dataset D_i for each node based on $\pi^*(P)$
  $D \leftarrow D \cup D_i$
end
return best π_i on validation

Algorithm 2: Generalized Self-Imitation

Inputs:
S_1: initial problem size
S_2: target problem size
π_{S_1}: policy trained on expert data of problem size S_1
for $s \leftarrow S_1 + 1$ **to** S_2 **do**
  generate problem instances P_s of size s
  train π_s via Algorithm 1 by running π_{s-1} on P_s
end

search trace and computes the retrospective optimal path. π' always takes the minimal number of time steps (in the MILP case, the number of integer variables). Since the goal of self-imitation is to minimize the number of time steps, the feedback from the retrospective reasoning is exactly the feedback from π'. Hence we have the following proposition.

Proposition 1. *Assume π_{S_1} is a policy trained using imitation learning on problem size S_1. If, during the scaling-up training process to problems of size $S_2 > S_1$, the trained policy search trace, starting from π_{S_1}, always contains the expert search trace, then the final error rate ϵ_{S_2} on problems of size S_2 is at most that obtained by running imitation learning directly on problems of size S_2.*

Proof. By our assumption on the trace inclusion property, the dataset obtained by self-imitation corresponds to the right loss objective while the dataset collected by imitation learning does not. So the error rate trained on self-imitation learning data will be at most that trained on imitation learning. □

6 Experimental Results

Our empirical result is a case study on the task of risk-aware planning that is built on top of MILP-based path planning [5] and chance-constrained optimization [15].

Risk-Aware Path Planning. We briefly describe the problem setup of risk-aware planning. Appendix A in the supplementary material contains a detailed description. Given a start point, a goal point, a set of polygonal obstacles, and an upper bound of the probability of failure (risk bound), we need to find a path, represented by a sequence of way points, that minimizes an objective function while limiting the probability of failure to the risk bound. This task can be formulated as MILPs.

Experimental Setup. The experiment is conducted on a set of 150 different instances of randomly generated obstacle maps with 10 obstacles each. We split them 50/50/50 for train/validation/test. We used a commercially available MILP solver Gurobi to generate expert solutions. Details on dataset generation can be found in the Appendix. For this study, we set the risk bound $\delta = 0.02$ and vary the number of way points from 10 to 14, in increment of 1. The number of integer variables range from 400 to 560.

Policy Class. Our policy class consists of a node ranking model and a pruning model. We use RankNet [18] as the ranking model, instantiated using a 2-layer neural network. For the pruning model, we train a 1-layer neural network classifier.

Method	Explored nodes (Avg)	Optimality gap (Node Limit)
Imitation Learning [14]	49	48%
Self-Imitation Learning	43	31.2%

Table 1: Comparison of Imitation Learning and Self-Imitation Learning for 10 way points test data

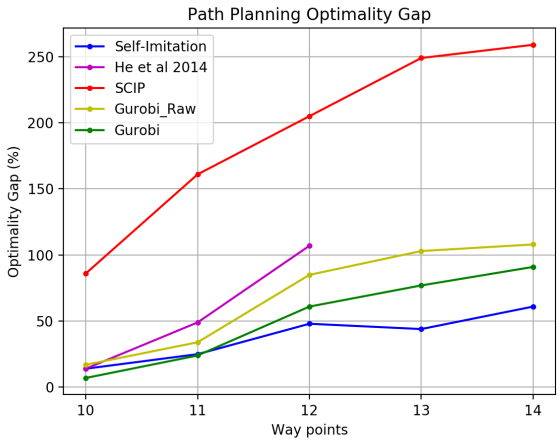


Figure 2: Mean Percentage Optimality Gap on the Risk-Aware Path Planning held-out test data.

Methods Compared. We compared our method with a commercial solver Gurobi, SCIP [19], as well as the previous imitation learning approach by [14]. Due to the difference in the implementation, we use the number of explored nodes as a proxy for runtime, i.e. we compare the optimality gap of the algorithms at the same number of explored nodes. We also compare against a version of Gurobi that de-prioritized primal heuristics, which we call Gurobi_Raw, in order to compare the local search branch-and-bound behavior. Since the method in [14] relies on supervised demonstrations by an expert, it is only trained on the reference problem size.

Main Results. We conduct two sets of experiments to demonstrate: 1) improvement over imitation learning [14] brought by self-imitation learning within the reference problem size; and 2) scaling ability of self-imitation learning to larger problem sizes.

In the first experiment, we compare against [14] within the reference problem size (10 way points). The results are summarized in the Table 1. Our approach provides a 15% relative improvement in search time and 50% relative improvement in solution quality. These results suggest that initial expert demonstrations were sub-optimal for the inductive biases of our policy class, and self-imitation was able to discover new feasible solutions easier for our policy class to learn.

In the second experiment, we validate the ability of self-imitation learning to scale up in a self-supervised fashion. A policy is initially trained on the reference scale (10 way points) using the expert solutions. It is then used to train on the next problem scale (11 way points) via Algorithm 2. We also utilized some exploration schemes into our policy (see Appendix C). Upon convergence, this process is repeated on increasing problem scales up to 14 way points. Figure 2 depicts the optimality gap as we scale up to larger problem instances. We see that self-imitation learning is able to effectively scale up. Note that method in [14] and SCIP failed in finding feasible solutions for $\sim 60\%$ and $\sim 20\%$, respectively, on the test instances, when scaling up beyond 12 way points. We thus did not test [14] beyond 12 way points. Also note that the self-imitation approach achieved a comparable performance with the expert solver (Gurobi) even though it is trained with expert feedback only at the initial problem scale (10 way points).

7 Conclusion

We have presented the self-imitation approach for training search policies to solve MILPs. Our approach extends conventional imitation learning by being able to learn good policies without requiring repeated queries to an expert. Our theoretical analysis shows that, under certain assumptions, the self-imitation learning scheme is provably more powerful and general than conventional imitation learning. We validated our approach on the challenging problem of risk-aware planning, where we demonstrated both performance gains over conventional imitation learning and the ability to scale up to large problem instances not tractably solvable by commercial solvers.

References

[1] Robert S Garfinkel and George L Nemhauser. *Integer programming*, volume 4. Wiley New York, 1972.

[2] Min Sun, Murali Telaprolu, Honglak Lee, and Silvio Savarese. Efficient and exact map-mrf inference using branch and bound. In *Conference on Artificial Intelligence and Statistics (AISTATS)*, 2012.

[3] Alexander Schwing and Raquel Urtasun. Efficient exact inference for 3d indoor scene understanding. *European Conference on Computer Vision (ECCV)*, 2012.

[4] Xian Qian and Yang Liu. Branch and bound algorithm for dependency parsing with non-local features. *Transactions of the Association for Computational Linguistics (TACL)*, 1:37–48, 2013.

[5] Tom Schouwenaars, Bart DeMoor, Eric Feron, and Jonathan How. Mixed integer programming for multi-vehicle path planning. In *In European Control Conference 2001*, pages 2603–2608, 2001.

[6] Masahiro Ono, Brian C Williams, and Lars Blackmore. Probabilistic planning for continuous dynamic systems under bounded risk. *Journal of Artificial Intelligence Research (JAIR)*, 46:511–577, 2013.

[7] Tichakorn Wongpiromsarn, Ufuk Topcu, Necmiye Ozay, Huan Xu, and Richard M Murray. Tulip: a software toolbox for receding horizon temporal logic planning. In *International Conference on Hybrid Systems: Computation and Control*, 2011.

[8] Christopher A Hane, Cynthia Barnhart, Ellis L Johnson, Roy E Marsten, George L Nemhauser, and Gabriele Sigismondi. The fleet assignment problem: Solving a large-scale integer program. *Mathematical Programming*, 70(1):211–232, 1995.

[9] Ailsa H Land and Alison G Doig. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, pages 497–520, 1960.

[10] Rica Gonen and Daniel Lehmann. Optimal solutions for multi-unit combinatorial auctions: Branch and bound heuristics. In *ACM Conference on Economics and Computation (EC)*, 2000.

[11] Kaj Holmberg and Di Yuan. A lagrangian heuristic based branch-and-bound approach for the capacitated network design problem. *Operations Research*, 48(3):461–481, 2000.

[12] Stéphane Ross, Geoffrey Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

[13] Hal Daumé III, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine learning*, 75(3):297–325, 2009.

[14] He He, Hal Daume III, and Jason M Eisner. Learning to search in branch and bound algorithms. In *NIPS*, 2014.

[15] András Prékopa. The use of discrete moment bounds in probabilistic constrained stochastic programming models. *Annals of Operations Research*, 85:21–38, 1999.

[16] Ro Marcos Alvarez, Quentin Louveaux, and Louis Wehenkel. A supervised machine learning approach to variable branching in branch-and-bound. In *European Conference on Machine Learning (ECML)*, 2014.

[17] Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daume, and John Langford. Learning to search better than your teacher. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2058–2066, 2015.

[18] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *International Conference on Machine Learning (ICML)*, 2005.

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

Supplementary Material

A MILP formulation of risk-aware path planning

This section describes the MILP formulation of risk-aware path planning solved in Section 6. Our formulation is based on the MILP-based path planning originally presented by [5], combined with risk-bounded constrained tightening [15]. It is a similar formulation as that of the state-of-the-art risk-aware path planner pSulu [6] but without risk allocation.

We consider a path planning problem in $\mathbb{R}$, where a path is represented as a sequence of N waypoints $x_1, \cdots x_N \in X$. The vehicle is governed by a linear dynamics given by:

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k + w_k \\ u_k &\in U, \end{aligned}$$

where $U \subset \mathbb{R}^m$ is a control space, $u_k \in U$ is a control input, $w_k \in \mathbb{R}^n$ is a zero-mean Gaussian-distributed disturbance, and A and B are n-by-n and n-by-m matices, respectively. Note that the dynamic of the mean and covariance of x_i, denoted by $\bar{x}_i$ and Σ_i, respectively, have a deterministic dynamics:

$$\begin{aligned} \bar{x}_{k+1} &= A\bar{x}_k + Bu_k + w_k \\ \Sigma_{k+1} &= A\Sigma A^T + W, \end{aligned} \tag{1}$$

where W is the covariance of w_k. We assume there are M polygonal obstacles in the state space, hence the following linear constraints must be satisfied in order to be safe (as in Figure 3):

$$\bigwedge_{k=1}^{N} \bigwedge_{i=1}^{M} \bigvee_{j=1}^{L_i} h_{ij} x_k \le g_{ij},$$

where $\bigwedge$ is conjunction (i.e., AND), $\bigvee$ is disjunction (i.e., OR), L_i is the number of edges of the i-th obstacle, and h_{ij} and g_{ij} are constant vector and scaler, respectively. In order for each of the linear constraints to be satisfied with the probability of $1-\delta_{kij}$, the following has to be satisfied:

$$\bigwedge_{k=1}^{N} \bigwedge_{i=1}^{M} \bigvee_{j=1}^{L_i} h_{ij} \bar{x}_k \le g_{ij} - \Phi(\delta_{kij}) \tag{2}$$

$$\Phi(\delta_{kij}) = -\sqrt{2h_{ijk}\Sigma_{x,k}h_{ijk}^T}\,\mathrm{erf}^{-1}(2\delta_{ijk}-1),$$

where erf^{-1} is the inverse error function.

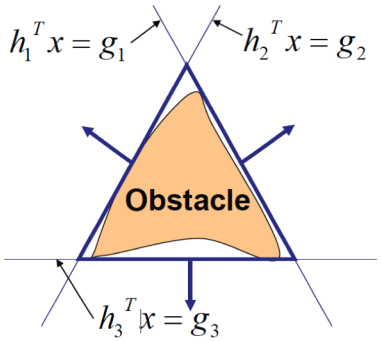


Figure 3: Representation of polygonal obstacle by disjuctive linear constraints

The problem that we solve is, given the initial state $(\bar{x}_0, \Sigma_0)$, to find $u_1 \cdots u_N \in U$ that minimizes a linear objective function and satisfies (1) and (2). An arbitrary nonlinear objective function can be approximated by a piecewise linear function by introducing integer variables. The disjunction in (2) is also replaced by integer variables using the standard Big M method. Therefore, this problem is equivalent to MILP. In the branch-and-bound algorithm, the choice of which linear constraint to be satisfied among the disjunctive constraints in (2) (i.e., which side of the obstacle x_k is) corresponds to which branch to choose at each node.

B Risk-aware Planning Dataset Generation

We generate 150 obstacle maps. Each map contains 10 rectangle obstacles, with the center of each obstacle chosen from a uniform random distribution over the space $0 \leq y \leq 1$, $0 \leq x \leq 1$. The side length of each obstacle was chosen from a uniform distribution in range $[0.01, 0.02]$ and the orientation was chosen from a uniform distribution between 0° and 360°. In order to avoid trivial infeasible maps, any obstacles centered close to the destination are removed.

C Exploration Strategy

For self-imitation learning to succeed in scaling up to larger problem instances, it is important to enable exploration strategies in the search process. In our experiments, we have found the following two strategies to be most useful.

- ϵ-greedy strategy allows a certain degree of random exploration. This helps learned policies to discover new terminal states and enables self-imitation learning to learn from a more diverse goal set. Discovering new terminal states is especially important when scaling up because the learned policies are trained for a smaller problem size; to counter the domain shift when scaling up, we add exploration to enable the learned policies to find better solutions for the new larger problem size.
- Searching for multiple terminal states and choosing the best one as the learning target. This is an extension to the previous point since by comparing multiple terminal states, we can pick out the one that is best for the policy to target, thus improving the efficiency of learning.
- When scaling up, for the first training pass on each problem scale, we collect multiple traces on each data point by injecting 0.05 variance Gaussian noise into the regression model within the policy class, before choosing the best feasible solution.