Learning Feasibility of Factored Nonlinear Programs in Robotic Manipulation Planning

Joaquim Ortiz-Haro[1], Jung-Su Ha[1], Danny Driess[1], Erez Karpas[2], Marc Toussaint[1]

***Abstract*— A factored Nonlinear Program (Factored-NLP) explicitly models the dependencies between a set of continuous variables and nonlinear constraints, providing an expressive formulation for relevant robotics problems such as manipulation planning or simultaneous localization and mapping. When the problem is over-constrained or infeasible, a fundamental issue is to detect a minimal subset of variables and constraints that are infeasible. Previous approaches require solving several nonlinear programs, incrementally adding and removing constraints, and are thus computationally expensive. In this paper, we propose a graph neural architecture that predicts which variables and constraints are jointly infeasible. The model is trained with a dataset of labeled subgraphs of Factored-NLPs, and importantly, can make useful predictions on larger factored nonlinear programs than the ones seen during training. We evaluate our approach in robotic manipulation planning, where our model is able to generalize to longer manipulation sequences involving more objects and robots, and different geometric environments. The experiments show that the learned model accelerates general algorithms for conflict extraction (by a factor of 50) and heuristic algorithms that exploit expert knowledge (by a factor of 4).**

I. INTRODUCTION

Computing values for a set of variables that fulfil all the constraints is a key problem in several applications, such as robotics, planning, and scheduling. In discrete domains, these problems are generally known as Constrained Satisfaction Problems (CSP), which also include classical combinatorial optimization like k-coloring, maximum cut or Boolean satisfaction (SAT). In continuous domains, the dependencies between a set of continuous variables and nonlinear constraints can be modelled with a factored nonlinear program (without cost term or with a small regularization) which have applications in robotic motion planning [1] or simultaneous localization and mapping [2].

When a problem is over-constrained or infeasible, a fundamental challenge is to extract a minimal conflict: a minimal subset of variables and constraints that are jointly infeasible. These conflicts usually provide an explanation of the failure that can be incorporated back into iterative solvers, for example in the conflict-driven clause learning algorithm for Boolean satisfiability problems (SAT) [3], [4], or conflict based solvers for Task and Motion Planning in robotics (TAMP) [5], [6], [7], [8], [9].

In a continuous domain, extracting conflicts is expensive as it requires solving several nonlinear programs adding and removing constraints.

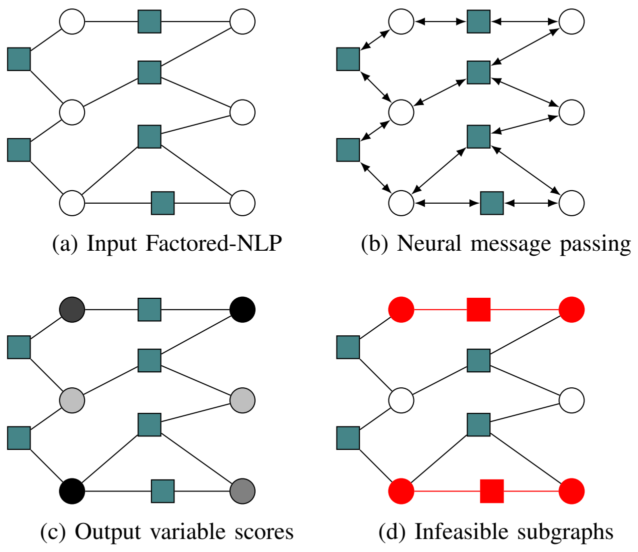

(a) Input Factored-NLP (b) Neural message passing (c) Output variable scores (d) Infeasible subgraphs

Fig. 1: Overview of our approach to detect minimal infeasible subgraphs in a Factored-NLP. *(a)* The input of the model is a Factored-NLP. Circles represent variables and squares are constraints. *(b)* We perform several iterations of neural message passing using the structure of the NLP. *(c)* The network outputs the probability that a variable belongs to a minimal infeasible subgraph. *(d)* We extract several minimal infeasible subgraphs with a connected component analysis.

We propose a neural model to predict minimal infeasible subsets of variables and constraints from a factored nonlinear program. The input to our model is directly the factored nonlinear program, including semantic information on variables and constraints (e.g. a class label) and a continuous feature for each variable (which, for instance can be used to encode the geometry of a scene in robotics). The graph structure of the factored nonlinear program is exploited for performing message passing in the neural network. Finding the minimal infeasible subgraph (i.e. a subset of variables and constraints of the Factored-NLP) is cast as a variable classification problem, and the predicted infeasible subsets are extracted with a connected components analysis. An overview of our approach is shown in Fig. 1. The prediction of the graph neural network can be naturally integrated into an algorithm to detect minimal infeasible subgraphs, providing a significant speedup with respect to non-learning methods, both using general conflict extraction algorithms or expert algorithms with domain knowledge.

Our approach follows a promising trend in robotics to combine optimization and learning [10], [11], [12], [13],

[1] TU Berlin, Germany. [2] Technion, Israel.

This research has been supported by the German-Israeli Foundation for Scientific Research (GIF) grant I-1491-407.6/2019. Joaquim Ortiz-Haro and Danny Driess thank the International Max-Planck Research School for Intelligent Systems (IMPRS-IS) for the support.

[14]. In this paradigm, a dataset of solutions to similar problems is used to accelerate optimization methods, making expensive computations tractable and enabling real-time solutions to combinatorial and large scale optimization. Therefore, we assume that a dataset of labeled factored nonlinear programs (generated offline with a non-learning algorithm) is available. Each example consists of a factored nonlinear program and a set of minimal infeasible subgraphs.

As an application, we evaluate our method in robotic sequential manipulation. Finding minimal conflicts is a fundamental step in conflict-based solvers, usually accounting for most of the computational time.

From a robotics perspective, our novel contribution is to use the structure of the nonlinear program formulation of manipulation planning for message passing with a graph neural network. To this end, we first formulate the motion planning problem that arises from a high-level manipulation sequence (symbolic actions such as pick or place) as a factored nonlinear program [1]. Variables correspond to the configuration of objects and robots at each step of the motion and the nonlinear constraints model kinematics, collision avoidance and grasping constraints. When combined with our learned model, we get strong generalization capabilities to predict the minimal infeasibility of manipulation sequences of different lengths in different scenes, involving a different number of objects and robots.

Our contributions are:

- A neural model to predict minimal conflicts in factored nonlinear programs. We formulate the detection of minimal infeasible subgraphs as a variable classification problem and a connected components analysis.
- An algorithm that integrates the prediction of our neural model and non-learning conflict extraction methods, providing a significant acceleration.
- An empirical demonstration that the formulation of manipulation planning as a factored nonlinear program, together with our neural model, enables scalability and generalization.

II. Related Work

A. Minimal Infeasible Subsets of Constraints

In the discrete SAT and CSP literature, a minimal infeasible subset of constraints (also called Minimal Unsatisfiable Subset of Constraints or Minimal Unsatisfialble Core) is usually computed by solving a sequence of SAT and MAX-SAT problems [15], [16], [17].

In a general continuous domain, a minimal infeasible subset can be found by solving a linear number of problems [18]. This search can be accelerated with a divide and conquer strategy, with logarithmic complexity [19]. In convex and nonlinear optimization, we can find approximate minimal subsets by solving one optimization program with slack variables [20]. In contrast, our method uses learning to directly predict minimal infeasible subsets of variables and constraints, and can be combined with these previous approaches to reduce the computational time.

B. Graph Neural Networks in Combinatorial Optimization

We use Graph Neural Networks (GNN) [21], [22], [23] for learning in graph-structured data. Different message passing and convolutions have been proposed, e.g. [24], [25]. Our architecture, targeted towards inference in factored nonlinear programs, is inspired by previous works that approximate belief propagation in factor graphs [26], [27], [28].

Recently, GNN models have been applied to solve NP-hard problems [29], Boolean Satisfaction [30], Max cut [31], constraint satisfaction [32], and discrete planning [33], [34], [35]. Compared to state-of-the-art solvers, learned models achieve competitive solution times and scalabilty but are outperformed in reliability and accuracy. To our knowledge, this is the first work to use a GNN model to predict the minimal infeasible subgraphs of a factored nonlinear program in a continuous domain.

C. Graph Neural Networks in Manipulation Planning

In robotic manipulation planning, GNNs are a popular architecture to represent the relations between movable objects, because they provide a strong relational bias and a natural generalization to including additional objects in the scene.

For example, they have been used as problem encoding to learn policies for robotic assembly [36], [37] and manipulation planning [38], to learn object importance and guide task and motion planning [39], and to learn dynamical models and interactions between objects [40], [41]. Previous works often use object centric representations: the vertices of the graph represent the objects and the task is encoded in the initial feature vector of each variable. Alternatively, our model performs message passing using the structure of the nonlinear program that corresponds to a manipulation sequence, achieving generalization to different manipulation sequences that fulfil different goals.

III. Formulation

A. Factored Nonlinear Program (Factored-NLP)

A Factored-NLP G is a bipartite graph $G = (X \cup H, E)$ that models the dependencies between a set of variables $X = \{x_i \in \mathbb{R}^{n_i}\}$ and a set of constraints $H = \{h_a : \mathbb{R}^{m_a} \rightarrow \mathbb{R}^{m'_a}\}$. Each constraint $h_a(x_a)$ is a piecewise differentiable function evaluated on a (typically small) subset of variables $x_a \subseteq X$ (e.g. $x_a = \{x_1, x_4\}$). The edges model the dependencies between variables and constraints $E = \{(x_i, h_a)$: constraint h_a depends on variable $x_i\}$. Throughout this document, we will use the indices i, j to denote variables and a, b to denote constraints.

The underlying/associated nonlinear program is

$$\text{find } x_i \text{ s.t. } h_a(x_a) \leq 0 \quad \forall x_i \in X, h_a \in H \ . \tag{1}$$

The constraints $h_a(x_a) \leq 0$ also include equality constraints (that can be written as $h_a(x_a) \leq 0$ and $h_a(x_a) \geq 0$).

A Factored-NLP G is feasible ($\mathcal{F}(G) = 1$) iff (1) has a solution, that is, there exists a value assignment $\bar{x}_i$ for each variable x_i such that all constraints $h_a(\bar{x}_a)$ are fulfilled. Otherwise, it is infeasible ($\mathcal{F}(G) = 0$). This assignment can be computed with nonlinear optimization methods, such as Augmented Lagrangian or Interior Points. A minimal infeasible subgraph $M \subseteq G$ is an infeasible subset of variables and constraints whose any proper subset is feasible,

$$M \subseteq G,\ \mathcal{F}(M) = 0,\ \mathcal{F}(M') = 1\ \forall M' \subset M. \qquad (2)$$

Given a graph G and a subset of variables $X' \subseteq X$, a *variable-induced* subgraph $G[X'] = (X' \cup H', E')$ with $H' = \{h_a \in H : \text{Neigh}_G(h_a) \subseteq X'\}$ is the subgraph spanned by the variables X'. Intuitively, $G[X']$ contains the variables X' and all the constraints that can be evaluated with these variables. In this work, we consider only minimal subgraphs in the form of *variable-induced* subgraphs, i.e. $M = G[X'] \subseteq G$, because it enables a more compact representation (our approach can be adapted to predict general subgraphs if required, changing the proposed variable classification to constraint classification in Sec III-B).

A minimal infeasible subgraph is connected and a supergraph $\tilde{M} \supseteq M$ of an infeasible subgraph M is also infeasible. A factored-NLP G can contain multiple infeasible subgraphs, and a variable $x_i \in X$ can belong to multiple infeasible subgraphs.

B. Minimal Infeasible Subgraph as Variable Classification

Let $\phi(G) = \{M_r \subseteq G\}$ be the set of all minimal infeasible subgraphs M_r of a Factored-NLP G. Instead of learning ϕ directly, we propose to learn an over-approximation $\tilde{\phi}$ that can efficiently be framed as binary variable classification.

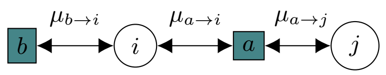
We first introduce the *variable-feasibility* function $\psi(x_i; G)$ that assigns a label $y_i \in \{0, 1\}$ to each variable $x_i \in X$. $y_i = 0$ if x_i belongs to some infeasible subgraph M_r, and $y_i = 1$ otherwise. Given such a labelled graph, we can recover the infeasible subgraphs approximately by computing the connected components on the graph spanned by the vertices with label 0, $G' = G[\{x_i : y_i = 0\}]$. Thus, we define the approximate mapping as,

$$\tilde{\phi}(G) = \text{CC}(G[\{x_i : y_i = 0\}]), \qquad (3)$$

where CC performs a connected component analysis.

The approximate mapping $\tilde{\phi}$ is exact, i.e. $\tilde{\phi} = \phi$, if the infeasible subgraphs $\{M_r\}$ are disconnected. If two or more of the infeasible subgraphs are connected, it returns their union as a minimal infeasible sugraph, i.e. $\cup\tilde{\phi} = \cup\phi$, which over-approximates the size of the original minimal infeasible subgraph. Our neural model will be trained to imitate the labels of the *variable-feasibility* function ψ.

We emphasize that learning the approximate function $\tilde{\phi}$ is not a real limitation. First, because the prediction will be integrated into an algorithm that can further reduce the size of the infeasible subgraph, if it is not already minimal, as shown later in Sec. III-D. Second, because in practice finding small infeasible subgraphs, as opposed to strictly minimal, is already useful in the applications. Finally, note that ϕ can be converted to a multiclass variable classification $f(x_i; G) = y \subseteq \{1, \ldots, R\}$, where each variable can belong to multiple classes – but this would require a complex, and potentially intractable, permutation invariant formulation.



$$[\mu_{a\to i}, \mu_{a\to j}] = \text{Message}_a(z_i, z_j), \quad \mu_{b\to i} = \text{Message}_b(z_i)$$
$$z_i' = \text{Update}\left(\text{AGG}\left(\mu_{a\to i}, \mu_{b\to i}\right), z_i\right), \quad z_j' = \text{Update}\left(\mu_{a\to j}, z_j\right)$$

Fig. 2: Message Passing in a Factored-NLP with two variables (i, j) and two constraints (a, b).

C. GNN with the Structure of a Factored-NLP

A fundamental idea of our method is to use the structure of the Factored-NLP for message passing with Graph Neural Networks (GNN) to learn the *variable-feasibility* $\psi(x_i; G)$.

Each variable vertex $x_i \in X$ has a feature vector $z_i \in \mathbb{R}^{n_z}$ that is updated with the incoming messages of the neighbour constraints. z_i is initialized with z_i^0 to encode semantic and continuous information of the variable x_i (an example on how to initialize the features in manipulation planning is shown in Sec. IV-B). The update rule follows a two-step process: first, each constraint computes and sends back a message to each neighbour variable, which depends on the current features of all the neighbour variables. Second, each variable aggregates the information of the incoming messages from the constraints and updates its feature vector. A graphical representation is shown in Fig 2.

$$[\oplus\mu_{a\to i}]_{i\in N(a)} = \text{Message}_a([\oplus z_i]_{i\in N(a)}), \qquad (4)$$
$$z_i' = \text{Update}(\text{AGG}_{a\in N(i)}\ \mu_{a\to i}, z_i).$$

$\mu_{a\to i} \in \mathbb{R}^{n_\mu}$ is the message from constraint a to variable i. $[\oplus z_i]_i$ denotes concatenation. $N(a) = \text{Neigh}_G(h_a)$ is the ordered set of variables connected to the constraint h_a. $N(i) = \text{Neigh}_G(x_i)$ is the set of constraints connected to variable x_i. AGG is an aggregation function, e.g. max, sum, mean or weighted average. We use max (element-wise) in our implementation. `Update` and `Message`$_a$ are small MLPs (Multilayer Perceptron) with learnable parameters. Likewise the nonlinear constraints in the Factored-NLP are not permutation invariant or symmetric, the features z_i have to be concatenated in a predefined order $N(a)$ when evaluating `Message`$_a$.

The function `Update` is shared by all vertices (which generalizes to Factored-NLPs with additional variables). The function `Message`$_a$ is shared between different constraints of the Factored-NLP that represent the same mathematical function, i.e. `Message`$_a$ = `Message`$_b$ iff $h_a(x) = h_b(x)$ (which generalizes to Factored-NLPs with additional constraints). For example, in manipulation planning, all constraints that model collisions between objects will share the same `Message` MLP.

The message passing update (4) is performed K times, starting from the initial feature vectors z_i^0. The feature vectors after K iterations are used for variable classification with a small MLP.

$$\hat{y}_i = \text{CLASSIF}(z_i^K) \qquad (5)$$

The parameters of all message and update networks are

Algorithm 1: Conflict Extraction with a GNN

```
Input: Factored-NLP $G$, GNN_Model, Solve, Reduce,
Result: $M \subseteq G$   ▷ Minimal infeasible subgraph
1  $[\hat{y}_i]$ = GNN_Model(Factored-NLP)
2  $\delta = .5,\ \delta_r = 1.2, found = false$
3  while not found do
4      $X_{candidate} = \{x_i \in X : \hat{y}_i < \delta\}$
5      for $g \in$ Connected_components($G[X_{candidate}]$) do
6          $feasible$ = Solve($g$)
7          if not feasible then
8              $M \leftarrow$ Reduce($g$)
9              $found \leftarrow true$
10             break
11     $\delta \leftarrow \delta \times \delta_r$
12 return $M$
```

trained to minimize the weighted binary cross entropy loss between $\hat{y}_i$ and the *variable-feasibility* labels y_i.

D. Algorithm to Detect Minimal Infeasible Subgraphs

To account for the approximation of our variable classification formulation, and small prediction errors, we can integrate the learned classifier into a classical algorithm to detect minimal infeasible subgraphs.

We assume the user provides the Solve and Reduce routines, that respectively check if a Factored-NLP is feasible and compute a minimal infeasible subset of an infeasible graph. Reduce is an expensive routine, as it requires solving several nonlinear programs adding and removing constraints. The number of evaluated NLPs (and therefore the computation time) depends on the size of the input graph: linear on the total number of variables using [18], or logarithmic [19].

Our algorithm is shown in Alg. 1. The GNN model is evaluated on the input Factored-NLP and computes a feasibility scores $\hat{y}_i$ for each variable. Iteratively increasing the classification threshold δ, we select the candidate infeasible variables $X_{\text{candidate}}$ using the current threshold δ. We generate candidate infeasible subgraphs with a connected components analysis on the *variable-induced* subgraph $G[X_{\text{candidate}}]$, that are evaluated with Solve. Once an infeasible subgraph is found, we use Reduce to get a minimal infeasible subgraph.

A non-learning approach runs solve and reduce directly on the input Factored-NLP. Therefore, the acceleration in our algorithm comes from evaluating these routines with small (ideally minimal) candidates. Alg. 1 can be extended to compute several minimal infeasible subgraphs by removing the break instruction (line 10) and adding a special check before solving a candidate subgraph (line 6), to avoid solving for a supergraph of a found infeasible subgraph.

IV. Factored-NLP for Manipulation Planning

In this section, we present a factored nonlinear program formulation for robotic manipulation planning that enables our model to generalize to problems with longer manipulation sequences, more objects and robots, and different geometric environments.

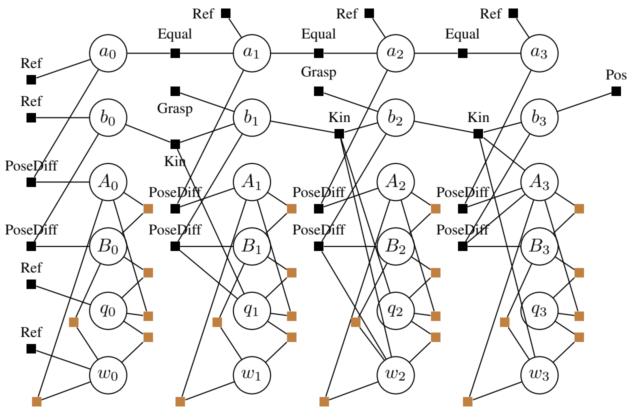


Fig. 3: Factored-NLP in manipulation planning. Circles are variables and squares are constraints (brown is collision avoidance). Each column represents a keyframe of the manipulation sequence. q, w are the configurations of two robots; A, B are the absolute position of two objects, and a, b are the relative pose of these objects with respect to their parent in the kinematic tree.

A. Structure of the Factored-NLP

A Factored-NLP models the motion of robots and objects that is implied by a sequence of high-level actions (such as pick and place) as a nonlinear optimization/feasibility problem. It contains variables that represent the configuration of objects and robots at each time step. We focus on the keyframe or mode-switch problem, that considers only the configurations at the beginning and end of each motion phase (e.g. when picking or placing an object), but not the trajectory between them. This follows a common problem decomposition used in Task and Motion Planning, where path feasibility is evaluated afterwards with trajectory optimization or sampling based motion planning [42], [1].

We include three types of variables: robot configurations, object absolute positions and object relative positions with respect to the parent frame. Nonlinear constraints model grasping, kinematic, stability, and collision avoidance constraints. The structure of the NLP is similar to previous factored formulations [7], [42], [43] but less compact (one can formulate the nonlinear program without explicitly introducing the absolute position of objects). However, this is necessary to formulate Factored-NLPs of diverse manipulation sequences using only few types of nonlinear constraints. Because each type of constraint will correspond to a different Message network, this formulation is crucial to enable generalization of the GNN model.

In Fig. 3, we show the Factored-NLP that corresponds to the manipulation sequence *(pick object B with robot Q from Initial B), (pick object B with robot W from robot Q), (place object B on top of object A with Robot W).*

B. Encoding of the Problem in the Initial Feature Vectors

The structure of the Factored-NLP implicitly encodes the number of objects, robots and the high-level action sequence (e.g. which robots pick which objects). The geometric description of the environment is encoded locally in the initial

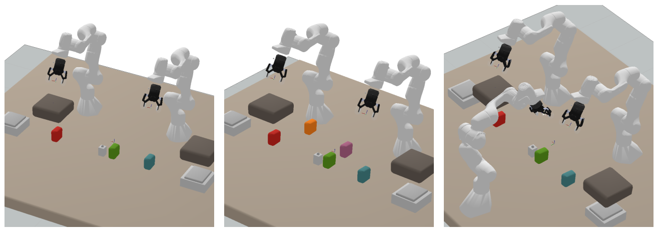


Fig. 4: Manipulation Scenarios. Obstacles are brown, blocks are colorful and tables are white. *Left*: Train Data, *Middle*: + Blocks, *Right:* + Robots.

feature vector of each variable z_i^0. Specifically, the initial feature vector includes the information of unary constraints (i.e. constraints evaluated only on a single variable, which are then not added to the message passing architecture), additional semantic class information (for example, whether the variable represents an object or a robot, but without including a notion of time index or entity), and geometric information that is relevant for the constraints (for example, the size of the objects). The dimension of z_i^0 is fixed, and shorter feature vectors are padded with zeros.

For example, suppose that the Factored-NLP of Fig. 3 is evaluated in a scene where robot Q is at pose $T_Q = [.32, .41, .56 \; .707, 0, 0, 0.707]$, the start position of object A is $T_A = [0.35, 0.4, 0.5, 0.707, 0, 0, 0.707]$ and object A is a box of size $S_A = [.2, .3, .2]$. Then z^0 of variables $\{q_0, q_1, q_2, q_3\}$ is $[1, 0, 0, 0, 0, 0, T_Q]$ where the first 6 components are a one-hot vector to indicate that it is a robot. The z^0 of $\{a_0, a_1, a_2, a_3\}$ is $[0, 1, 0, 0, 0, 0, T_A]$, where first components indicate that it is a relative pose with respect to the reference position. The z^0 of $\{A_0, A_1, A_2, A_3\}$ is $[0, 0, 1, 0, 0, 0, S_A, 0, 0, 0, 0]$ to indicate that it is an absolute position of an object of size S_A.

V. Experimental Results

A. Scenario

We evaluate our model in robotic sequential manipulation. The high-level goal is to build towers and rearrange blocks in different configurations, in scenarios that contain several and varying number of blocks, robots and movable obstacles, at different positions, see Fig. 4 and 5. The following setting is used to generate the train dataset (4800 Factored-NLPs):

• Five movable objects: 3 blocks and 2 obstacles. Both types of objects have collisions constraints but obstacles are bigger and usually block grasps or placements. • Two robots: 7-DOF Panda robot arms, that can pick and place objects using a top grasp. • Different geometric scenes: the position of the objects, robots and tables is randomized. • Manipulation sequences of length 4 to 7.

To evaluate the generalization capabilities of the learned model, we consider three additional datasets:

• `+Robots`: we add an additional robot. • `+Blocks`: we add two additional blocks. • `+Actions`: it contains Factored-NLPs with longer manipulation sequences (length of 8 to 10).

TABLE I: Classification Accuracy. Each pair indicates the accuracy in predicting feasible and infeasible variables.

	Train Data	+ blocks	+ Robots	+ Actions
GNN	(94.7, 95.4)	(96.1, 95.2)	(95.7, 95.3)	(94.6, 94.1)
MLP	(93.0, 82.2)	(93.4, 80.8)	(93.0, 80.8)	(91.0, 48.0)
MLP-SEQ	(83.5, 88.1)	(82.3, 88.8)	(82.1, 88.8)	(74.0, 75.3)

B. Data Generation

For training the GNN model we need a set of Factored-NLPs with labelled variables to indicate whether they belong to a minimal infeasible subset. First, we generate a set of interesting high-level action sequences. Second, we evaluate the manipulation sequences on random geometric scenes to generate Factored-NLPs (including the initial feature vectors). To compute the feasibility labels, we adapt the conflict extraction algorithm of [7] to find up to 10 minimal infeasible subgraphs.

C. Accuracy of the GNN Classifier

We compare our model (*GNN*) against a Multilayer Perceptron (*MLP*) and a sequential model (*MLP-SEQ*), trained with the same dataset.

The *MLP* computes $\hat{y}_i = \text{MLP}(\tilde{z}_i^0, A, C)$. $\tilde{z}_i^0 = [z_i^0, t_i, e_i]$ is the feature vector of the variable we want to classify. It concatenates the feature vector z_i^0 used in the *GNN*, with the time index t_i of the variable, and a parametrization that defines the entity e_i (for instance, we represent an object with its starting pose – one hot encodings could not generalize to more objects). A is the encoding of the whole action sequence, using small vectors to encode each token, e.g. {"pick", "block1", "l_gripper", "table"}. To account for sequences of different length, we fix a maximum length and add padding. C is the global scene parametrization. Since the dimensionality of C is required to be constant, it contains a parametrization that can encode the maximum number of robots and objects. We also evaluate *MLP-SEQ*, a sequential model $\text{MLP}(\tilde{z}_i^0, \text{SEQ}(A), C)$ that encodes the action sequence with a recurrent network (using Gated Recurrent Units (GRU)).

We first evaluate the accuracy of the models to predict if a variable belongs to a minimal infeasible subset, see Tab. I. Our *GNN* model outperforms the alternative architectures, both in the original *Train Data* and, specially, in the extension datasets. Our model keeps a constant ∼95% success rate across all datasets, while the performance of *MLP* and *MLP-SEQ* drops to 48% and 75%. We also evaluate the accuracy of our model to predict infeasible subgraphs, using the proposed method that combines variable classification (with a threshold of $\delta = 0.5$) and connected component analysis. Our model outperforms *MLP* and *MLP-SEQ*, and finds between 70% and 57% of the infeasible subgraphs, and 30%-50% of the predicted subgraphs are minimal, see Tab. II. Between 34%-48% of the predicted graphs are feasible (not shown in the table due to space limitation).

TABLE II: Prediction of infeasible subgraphs. Each pair indicates the ratio "found / total" and "minimal / found".

	Train Data	+ Blocks	+ Robots	+ Actions
GNN	(71.2, 54.1)	(58.9, 33.3)	(70.2, 55.3)	(57.1, 41.9)
MLP	(58.5, 54.6)	(34.5, 53.2)	(55.2, 37.6)	(22.1, 35.5)
MLP-SEQ	(65.7, 26.0)	(28.6, 21.2)	(61.3, 09.5)	(36.3, 11.0)

TABLE III: Finding one minimal infeasible subgraph. Each pair indicates the number of solved NLPs and the computational time in 100 Factored-NLPs, normalized by *GNN+g1*.

	Train Data	+ blocks	+ Robots	+ Actions
GNN+e	(1.57, 2.25)	(1.44, 2.09)	(1.66, 2.14)	(1.50, 2.19)
GNN+g1	(1, 1)	(1, 1)	(1, 1)	(1, 1)
Oracle	(0.83, 0.97)	(0.62, 0.79)	(0.83, 0.84)	(0.71, 0.86)
Expert	(3.66, 4.32)	(3.13, 5.06)	(4.33, 4.62)	(3.33, 4.56)
General 2	(3.50, 64.1)	(3.30, 163)	(3.50, 66.5)	(3.83, 128)

MLP, *MLP-SEQ* and *GNN* have the same information to make the predictions. The Factored-NLP is a deterministic mapping of the action sequence and the geometric scene. Although a *MLP* could learn this mapping, our experiments show that the representation does not emerge naturally – confirming that a structured model yields better generalization.

D. Finding Minimal Infeasible Subgraphs

We analyze the time required to find one minimal infeasible subgraph in an infeasible Factored-NLP with algorithms:

- *Oracle*, which executes a single call to `Solve` and `Reduce` with a minimal infeasible subgraph as input.
- *General* {*1,2*}, which are generic algorithms for conflict extraction: *General 1* uses constraint filtering [18], and *General 2* uses *QuickXplain* [19].
- *Expert* is a heuristic algorithm for conflict extraction in manipulation planning [8]. It exploits the temporal structure, domain relaxations, and the convergence of the optimizer to quickly discover the conflicts.
- *GNN*+{*e,g1*} combines the prediction of our *GNN* model with either *Expert* or *General 1*. *Expert* and *General 1* are used as `Reduce` in Alg. 1.

Results are shown in Table III. *GNN+g1* is 60-120x faster than *General 2* (which is faster than *General 1*). This highlights the benefits of our approach in domains where we can compute a dataset using *General* offline, and train the model to get an order-of-magnitude improvement in new problems. *GNN+g1* is 4-5x faster than the *Expert* algorithm, and only 1.2x slower than an oracle. Moreover, the acceleration provided by *GNN* is maintained in all the datasets. This confirms the good accuracy and generalization of the architecture seen in the classification results. As a side note, *Expert* is faster than *General 2* because it solves a lot of small feasible NLPs first, until it finds one that is infeasible (which is faster than solving infeasible NLPs).

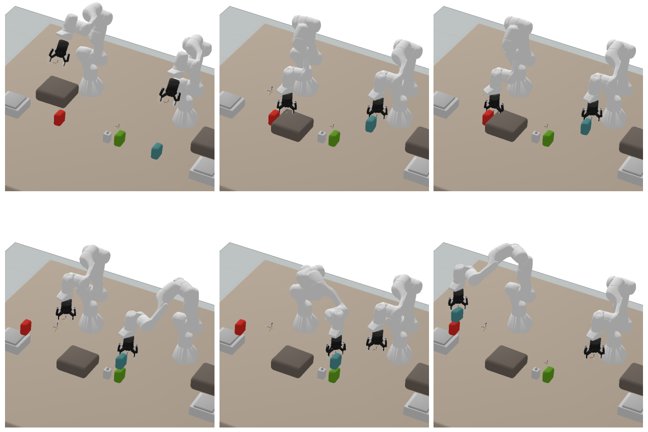


Fig. 5: Manipulation sequence in + *Actions*. Robots build a tower [*red*, *blue*] in the left table, moving first an obstacle.

E. Integration in a Conflict-based TAMP Planner

We demonstrate the benefits of neural accelerated conflict extraction inside the GraphNLP Planner [8] for solving TAMP problems. The planner iteratively generates high-level plans, detects infeasible subgraphs, and encodes this information back into the logical description of the problem.

For this evaluation, we define 10 high-level goals for each setting corresponding to the `+Actions`, `+Robots`, and `+Blocks` datasets, and report the total sum (including the 10 goals) of the number of solved NLPs and the computational time in the conflict extraction component of TAMP solver. *GNN+e* (which is more robust than *GNN+g1* in this setting) takes only (8.33s, 511 NLPs), (9.83s, 603 NLPs), and (63.9s, 1979 NLPs) for each scenario, and is between 2 and 3 times faster than the *expert* algorithm, which requires (24.2s, 731 NLPs), (38.7s, 1116 NLPs), and (137.9s, 2554 NLPs).

VI. Conclusion

In this paper, we have presented a neural model to predict the minimal infeasible subsets of variables and constraints in a factored nonlinear program. The structure of the nonlinear program is used for neural message passing, providing generalization to problems with more variables and constraints.

We have demonstrated our approach in manipulation planning. A single learned model, combined with a suitable NLP representation of the motion sequence, can predict minimal infeasibility of manipulation sequences of different lengths in different scenes, increasing the number of objects and robots. Our model achieves high accuracy, and the predictions can be integrated to guide and accelerate classical and heuristic algorithms for detecting minimal conflicts.

As future work, we would like to apply our neural formulation to detect conflicts in discrete constraint satisfaction problems, such as Boolean Satisfaction or k-coloring. From a robotics perspective, we will further investigate the potential of graph neural networks to combine logic and geometric information for guiding task and motion planning.

REFERENCES

[1] M. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning," in *Robotics: Science and Systems XIV RSS*, 2018.

[2] F. Dellaert, M. Kaess, *et al.*, "Factor graphs for robot perception," *Foundations and Trends® in Robotics*, vol. 6, no. 1-2, pp. 1–139, 2017.

[3] R. J. Bayardo Jr and R. Schrag, "Using csp look-back techniques to solve real-world sat instances," in *Aaai/iaai*. Providence, RI, 1997, pp. 203–208.

[4] J. P. Marques-Silva and K. A. Sakallah, "Grasp: A search algorithm for propositional satisfiability," *IEEE Transactions on Computers*, vol. 48, no. 5, pp. 506–521, 1999.

[5] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "An incremental constraint-based framework for task and motion planning," *The International Journal of Robotics Research*, vol. 37, no. 10, pp. 1134–1151, 2018.

[6] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2014, pp. 639–646.

[7] J. Ortiz-Haro, E. Karpas, M. Toussaint, and M. Katz, "Conflict-directed diverse planning for logic-geometric programming," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 32, no. 1, pp. 279–287, Jun. 2022. [Online]. Available: https://ojs.aaai.org/index.php/ICAPS/article/view/19811

[8] J. Ortiz-Haro, E. Karpas, M. Katz, and M. Toussaint, "Conflict-driven interface between symbolic planning and nonlinear constraint solving," in *IEEE Robotics and Automation Letters (RA-L)*, 2022.

[9] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated task and motion planning," *Annual review of control, robotics, and autonomous systems*, vol. 4, pp. 265–293, 2021.

[10] D. Driess, O. Oguz, J.-S. Ha, and M. Toussaint, "Deep visual heuristics: Learning feasibility of mixed-integer programs for manipulation planning," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2020.

[11] D. Driess, J.-S. Ha, and M. Toussaint, "Deep visual reasoning: Learning to predict action sequences for task and motion planning from an initial scene image," in *Robotics: Science and Systems 2020 (RSS 2020)*. RSS Foundation, 2020.

[12] N. Mansard, A. DelPrete, M. Geisert, S. Tonneau, and O. Stasse, "Using a memory of motion to efficiently warm-start a nonlinear predictive controller," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 2986–2993.

[13] A. Cauligi, P. Culbertson, B. Stellato, D. Bertsimas, M. Schwager, and M. Pavone, "Learning mixed-integer convex optimization strategies for robot planning and control," in *2020 59th IEEE Conference on Decision and Control (CDC)*. IEEE, 2020, pp. 1698–1705.

[14] R. Deits, T. Koolen, and R. Tedrake, "Lvis: learning from value function intervals for contact-aware robot controllers," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 7762–7768.

[15] M. H. Liffiton and K. A. Sakallah, "Algorithms for computing minimal unsatisfiable subsets of constraints," *Journal of Automated Reasoning*, vol. 40, no. 1, pp. 1–33, 2008.

[16] J. Marques-Silva, I. Lynce, and S. Malik, "Conflict-driven clause learning sat solvers," in *Handbook of satisfiability*. ios Press, 2021, pp. 133–182.

[17] F. Hemery, C. Lecoutre, L. Sais, F. Boussemart, *et al.*, "Extracting mucs from constraint networks," in *ECAI*, 2006.

[18] E. Amaldi, M. E. Pfetsch, and L. E. Trotter, "Some structural and algorithmic properties of the maximum feasible subsystem problem," in *Int. Conf. on Integer Progr. and Combinatorial Optimization*, 1999.

[19] U. Junker, "Preferred explanations and relaxations for over-constrained problems," in *AAAI-2004*, 2004.

[20] Y. Shoukry, P. Nuzzo, A. L. Sangiovanni-Vincentelli, S. A. Seshia, G. J. Pappas, and P. Tabuada, "Smc: Satisfiability modulo convex programming," *Proceedings of the IEEE*, 2018.

[21] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[22] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.

[23] Y. Ma and J. Tang, *Deep Learning on Graphs*. Cambridge University Press, 2021.

[24] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International conference on machine learning*. PMLR, 2017, pp. 1263–1272.

[25] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[26] Z. Zhang, F. Wu, and W. S. Lee, "Factor graph neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 8577–8587, 2020.

[27] V. G. Satorras and M. Welling, "Neural enhanced belief propagation on factor graphs," *CoRR*, vol. abs/2003.01998, 2020. [Online]. Available: https://arxiv.org/abs/2003.01998

[28] J. Kuck, S. Chakraborty, H. Tang, R. Luo, J. Song, A. Sabharwal, and S. Ermon, "Belief propagation neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 667–678, 2020.

[29] M. J. A. Schuetz, J. K. Brubaker, and H. G. Katzgraber, "Combinatorial optimization with physics-inspired graph neural networks," *CoRR*, vol. abs/2107.01188, 2021. [Online]. Available: https://arxiv.org/abs/2107.01188

[30] D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, and D. L. Dill, "Learning a sat solver from single-bit supervision," *arXiv preprint arXiv:1802.03685*, 2018.

[31] W. Yao, A. S. Bandeira, and S. Villar, "Experimental performance of graph neural networks on random instances of max-cut," in *Wavelets and Sparsity XVIII*, vol. 11138. SPIE, 2019, pp. 242–251.

[32] J. Toenshoff, M. Ritzert, H. Wolf, and M. Grohe, "Graph neural networks for maximum constraint satisfaction," *Frontiers in artificial intelligence*, vol. 3, p. 580607, 2021.

[33] W. Shen, F. Trevizan, and S. Thiébaux, "Learning domain-independent planning heuristics with hypergraph networks," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, 2020, pp. 574–584.

[34] O. Rivlin, T. Hazan, and E. Karpas, "Generalized planning with deep reinforcement learning," *arXiv preprint arXiv:2005.02305*, 2020.

[35] R. Nir, A. Shleyfman, and E. Karpas, "Learning-based synthesis of social laws in STRIPS," in *Proceedings of the Fourteenth International Symposium on Combinatorial Search, SOCS 2021, Virtual Conference [Jinan, China], July 26-30, 2021*, H. Ma and I. Serina, Eds. AAAI Press, 2021, pp. 88–96. [Online]. Available: https://ojs.aaai.org/index.php/SOCS/article/view/18555

[36] N. Funk, G. Chalvatzaki, B. Belousov, and J. Peters, "Learn2assemble with structured representations and search for robotic architectural construction," in *Proceedings of the 5th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, vol. 164. PMLR, 08–11 Nov 2022, pp. 1401–1411. [Online]. Available: https://proceedings.mlr.press/v164/funk22a.html

[37] S. K. S. Ghasemipour, D. Freeman, B. David, S. S. Gu, S. Kataoka, and I. Mordatch, "Blocks assemble! learning to assemble with large-scale structured reinforcement learning," 2022. [Online]. Available: https://arxiv.org/abs/2203.13733

[38] R. Li, A. Jabri, T. Darrell, and P. Agrawal, "Towards practical multi-object manipulation using relational reinforcement learning," in *2020 ieee international conference on robotics and automation (icra)*. IEEE, 2020, pp. 4051–4058.

[39] T. Silver, R. Chitnis, A. Curtis, J. B. Tenenbaum, T. Lozano-Perez, and L. P. Kaelbling, "Planning with learned object importance in large problem instances using graph neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, no. 13, 2021, pp. 11 962–11 971.

[40] D. Driess, Z. Huang, Y. Li, R. Tedrake, and M. Toussaint, "Learning multi-object dynamics with compositional neural radiance fields," 2022. [Online]. Available: https://arxiv.org/abs/2202.11855

[41] F. Paus, T. Huang, and T. Asfour, "Predicting pushing action effects on spatial object relations by learning internal prediction models," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 10 584–10 590.

[42] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Sampling-based methods for factored task and motion planning," *The International Journal of Robotics Research*, 2018.

[43] F. Lagriffoul, D. Dimitrov, J. Bidot, A. Saffiotti, and L. Karlsson, "Efficiently combining task and motion planning using geometric constraints," *The International Journal of Robotics Research*, vol. 33, no. 14, pp. 1726–1747, 2014.