GenCO: Generating Diverse Designs with Combinatorial Constraints

Aaron M. Ferber [1 2] **Arman Zharmagambetov** [3] **Taoan Huang** [1] **Bistra Dilkina** [1] **Yuandong Tian** [3]

Abstract

Deep generative models like GAN and VAE have shown impressive results in generating unconstrained objects like images. However, many design settings arising in industrial design, material science, computer graphics and more require that the generated objects satisfy hard combinatorial constraints or meet objectives in addition to modeling a data distribution. To address this, we propose GenCO, a generative framework that guarantees constraint satisfaction throughout training by leveraging differentiable combinatorial solvers to enforce feasibility. GenCO imposes the generative loss on provably feasible solutions rather than intermediate soft solutions, meaning that the deep generative network can focus on ensuring the generated objects match the data distribution without having to also capture feasibility. This shift enables practitioners to enforce hard constraints on the generated outputs during end-to-end training, enabling assessments of their feasibility and introducing additional combinatorial loss components to deep generative training. We demonstrate the effectiveness of our approach on a variety of generative combinatorial tasks, including game level generation, map creation for path planning, and photonic device design, consistently demonstrating its capability to yield diverse, high-quality solutions that verifiably adhere to user-specified combinatorial properties.

1. Introduction

Generating diverse and realistic objects with combinatorial properties is an important task with many applications. For AI-guided design in particular, the goal is to generate a diverse set of combinatorial solutions that are both feasible and high quality with respect to a given nonlinear objective. Diversity is often useful when dealing with uncertain metrics, diverse preferences, or for the sake of further manual scrutiny. For example, in photonic device design, we want to generate a variety of devices that meet foundry manufacturing constraints and optimize physics-related objectives (Schubert et al., 2022). In the design of new molecules, we want to generate a variety of chemically valid molecules with specific properties (Pereira et al., 2021). In other scenarios, we would like to create a diverse set of designs so that their solutions (not necessarily diverse given the design) meet certain objectives. For example, in video game level design, we may want to generate levels that are both realistic and valid/playable (Zhang et al., 2020). Here, "valid" may refer to discrete characteristics of the level that must be satisfied (e.g., a minimum number of enemies, a guaranteed path between the level entrance and exit, etc.).

Mathematically, we seek several solutions $\mathcal{X} := \{\boldsymbol{x}_j\}$ to the following optimization problem:

$$\min_{\mathcal{X}} \mathcal{L}(\mathcal{X}) + \gamma \sum_j \mathcal{D}(\boldsymbol{x}_j) \qquad \text{s.t. } \boldsymbol{x}_j \in \Omega \tag{1}$$

where $\mathcal{L}$ is a *group loss* defined on a population of solutions $\mathcal{X}$ (e.g., minimizing the Wasserstein distance to a dataset), $\mathcal{D}$ is an *individual loss* measuring the quality of each solution $\boldsymbol{x}_j$, γ is the weighing coefficient and Ω is the feasible set. Eqn. 1 can be solved by direct optimization or learning approaches that leverage solution datasets.

Several approaches can attempt to solve this problem. The *Lagrangian* (Frangioni, 2005) can reformulate the problem as $\min_{\mathcal{X}} \mathcal{L}(\mathcal{X}) + \gamma \sum_j \mathcal{D}(\boldsymbol{x}_j) + \lambda \sum_j \text{dist}_\Omega(\boldsymbol{x}_j)$, where $\text{dist}_\Omega(\boldsymbol{x}_j)$ measures the distance between $\boldsymbol{x}_j$ and the feasible set Ω ($\text{dist}_\Omega(\boldsymbol{x}_j) = 0$ if $\boldsymbol{x}_j \in \Omega$). However the final solutions $\boldsymbol{x}_j$ may not be feasible unless we use $\lambda \to +\infty$ which may lead to numerical instability. To reduce the optimization cost, we want to learn a *generative model* to directly generate $\boldsymbol{x}_j$, and avoid on-the-fly optimization. However, standard methods like GAN (Goodfellow et al., 2014) or VAE (Kingma & Welling, 2013) fail to generate feasible and high quality solutions. One approach (Chao et al., 2021) penalizes the generator based on constraint violation but is nontrivial to extend to more complex constraints,

*Equal contribution [1]Department of Computer Science, University of Southern California, Los Angeles, CA, USA [2]Department of Computer Science, Cornell University, Ithaca, NY, USA [3]AI at Meta (FAIR), Menlo Park, CA, USA. Correspondence to: Aaron M. Ferber <aferber@usc.edu>, Yuandong Tian <yuandong@meta.com>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

Approaches	Solution generation	Feasibility guarantees	Data distribution	Nonlinear objective
GAN/VAE (Goodfellow et al., 2014; Kingma & Welling, 2013)	**Fast**	✗	✓	✓
Integer Programming (Deaton & Grandhi, 2014)	Slow	✓	✗	✗
Lagrangian Multiplier (Di Liello et al., 2020)	Slow	✗	✓	✓
SurCo (Ferber et al., 2023)	Slow	✓	✗	✓
Fixing/Rejection (Zhang et al., 2020; Gomes et al., 2006)	Slow	✓	✗	✓
GenCO (proposed)	**Fast**	✓	✓	✓

Table 1. A comparison of relevant directions tackling generative modeling or combinatorial optimization for design problems.

such as logical or general combinatorial constraints that can be readily expressed in MILP. Alternative approaches include postprocessing solutions (Zhang et al., 2020), which may lead to repetitive solutions since the generative network is not trained with the postprocessing, or rejection sampling (Gomes et al., 2006) that can be computationally inefficient due to high rejection rates. Fundamentally, the training procedure does not involve verifiably feasible solutions. Another approach is to leverage integer linear programming (ILP) to find feasible solutions to problems with linear constraints on integer variables. However, such solvers are not geared towards generating diverse solutions and cannot leverage deep learning models to capture the data distribution or quickly adapt to slightly modified settings.

In this work, we propose GenCO. Instead of directly optimizing $\boldsymbol{x}$, we learn a *latent design space* $\mathcal{C}$ so that (1) each element $\boldsymbol{c} \in \mathcal{C}$ yields a guaranteed feasible solution $\boldsymbol{x} \in \Omega$, often through a combinatorial solver $\boldsymbol{g}$: $\boldsymbol{x} = \boldsymbol{g}(\boldsymbol{c})$, and (2) a generative model G_θ can map noise to the latent space $\mathcal{C}$. The entire pipeline is trained in an end-to-end manner via backpropagation through the solver $\boldsymbol{g}$. Intuitively, this latent space $\mathcal{C}$ represents the hidden parameters that lead to the solution. For example, $\boldsymbol{c}$ represents the layout of a 10x10 maze, while $\boldsymbol{x}$ represents the optimal path of the maze.

GenCO addresses many challenges in previous work. GenCO generates solutions efficiently due to its learned generative model G_θ. Additionally, GenCO guarantees feasibility via re-parameterization in the latent design space, which is different from conventional deep generative models like Generative Adversarial Networks (GANs) that directly generate the ultimate object of interest $\boldsymbol{x}$, be it images or any other complex entity. GenCO encourages diverse solutions, thanks to the group loss term $\mathcal{L}$ imposed on the solution population and end-to-end training of the entire pipeline. Finally, GenCO does not require linear objective functions and can handle nonlinear objectives. A comparison is summarized in Table 1.

Our main contributions involve introducing a framework, GenCO, that can: 1) train generative models to generate solutions that are guaranteed to satisfy combinatorial properties throughout training; 2) optimize nonlinear objectives; 3) flexibly handle a variety of generative models and optimizers. We showcase the effectiveness of our approach empirically on various generative combinatorial optimization problems: creating game level (subsection 4.2), generating maps for path planning (subsection 4.3), and designing inverse photonic devices (subsection 4.4).

2. Related Work

The interaction between generative models and combinatorial optimization has drawn attention as practitioners seek to generate objects with combinatorial optimization in mind.

Traditional constrained object generation Traditional constraint optimization methods were modified to search for multiple feasible solutions for problems concerning building layout (Bao et al., 2013), structural trusses (Hooshmand & Campbell, 2016), networks (Peng et al., 2016), building interiors (Wu et al., 2018), and urban design (Hua et al., 2019). Additionally, an approach based on Markov chain Monte Carlo (Yeh et al., 2012) samples objects that satisfy certain constraints. These approaches employ traditional sampling and optimization methods such as mathematical programming and problem-specific heuristics to obtain multiple feasible solutions. These methods often obtain multiple solutions by caching the feasible solutions found during the search process or modifying hyperparameters such as budgets or seed solutions. These methods can guarantee feasibility and optimality; however, they cannot synthesize insights from data such as historical good design examples or generate unstructured objects like images that are difficult to handle explicitly in optimization problems.

Infeasibility penalization Further work has modified deep generative models such as GAN (Goodfellow et al., 2014) to penalize constraint violation. These methods can generate objects respecting constraint graphs (Para et al., 2021), and blackbox constraints (Di Liello et al., 2020). Specialized approaches consider blackbox-constrained graphs such as in the design of photonic crystals (Christensen et al., 2020), crystal structure prediction (Kim et al., 2020), and house generation (Tang et al., 2023; Nauata et al., 2021). Here, feasibility is encouraged through penalties and inductive bias but is not guaranteed.

Optimization-based priors Previous work proposed conditioning VAE (Kingma & Welling, 2013) with combinatorial programs (Misino et al., 2022). Here, the latent information is extracted by the encoder and then fed through DeepProblog, a differentiable logic program (Manhaeve et al., 2018). The result is then fed through the decoder to generate the original image based on logical relationships. At test time, the model generates objects that hopefully satisfy the logical relationship through model conditioning; however, there is no guarantee of constraint satisfaction. Previous work has made various optimization problems differentiable, such as quadratic programs (Amos & Kolter, 2017), probabilistic logic programs (Manhaeve et al., 2018), linear programs (Wilder et al., 2019a; Mandi & Guns, 2020; Elmachtoub & Grigas, 2017; Liu & Grigas, 2021), Stackelberg games (Perrault et al., 2020), normal form games (Ling et al., 2018), kmeans clustering (Wilder et al., 2019b), maximum likelihood computation (Niepert et al., 2021), graph matching (Rolínek et al., 2020), knapsack (Demirovic et al., 2019b;a), maxsat (Wang et al., 2019), mixed integer linear programs (Mandi et al., 2020; Paulus et al., 2021; Ferber et al., 2020), blackbox combinatorial solvers (Pogančić et al., 2020; Mandi et al., 2022; Berthet et al., 2020), nonlinear programs (Donti et al., 2017), continuous constraint satisfaction (Donti et al., 2020), cone programs (Agrawal et al., 2019). General-purpose methods are presented for minimizing downstream regret by learning surrogate loss functions (Shah et al., 2022; 2023; Zharmagambetov et al., 2023). Additionally, previous work has employed learnable linear solvers to solve nonlinear combinatorial problems (Ferber et al., 2023). Finally, there is a survey on machine learning and optimization (Kotary et al., 2021). These approaches focus on identifying a single solution rather than generating diverse solutions. Additionally, many of these approaches can be integrated into GenCO if a specific optimization problem better suits the generative problem.

Enhacing combinatorial optimization with generative models Recently, generative models have been proposed to improve combinatorial optimization. (Zhang et al., 2022) use generative flow networks (gflownet) for robust scheduling problems. (Sun & Yang, 2023) use graph diffusion to solve combinatorial problems on graphs. (Ozair et al., 2021) use vector quantized variational autoencoders to compress the latent space for solving planning problems in reinforcement learning. (Zhao & You, 2020) use generative adversarial networks to generate settings for sample average approximation in robust chance-constrained optimization. Additionally, in (Lopez-Piqueres et al., 2023), the authors generate continuous objects with tensor networks that satisfy linear constraints for optimization problems. These approaches solve fully specified optimization problems rather than generating objects with combinatorial constraints.

3. GenCO: Method Description

In the GenCO framework (Figure 1), instead of direct optimization of the solution $\boldsymbol{x}$, we learn a distribution over a *latent design space* $\mathcal{C}$ so that for any $\boldsymbol{c} \in \mathcal{C}$, we get a solution $\boldsymbol{x} = \boldsymbol{g}(\boldsymbol{c}) \in \mathcal{X}$ that is guaranteed to be feasible. Here $\boldsymbol{g}(\cdot)$ is a combinatorial solver that uses its input $\boldsymbol{c}$ to specify the combinatorial optimization problems that it solves. For example, $\boldsymbol{g}(\boldsymbol{c}) := \arg\min_{\boldsymbol{x}\in\Omega} \boldsymbol{c}^\top \boldsymbol{x}$ outputs the solution to a mixed integer linear program (MILP) with coefficients $\boldsymbol{c}$. $\boldsymbol{g}(\cdot)$ can also output a shortest path (via Dijkstra's algorithm), given a weighted graph represented by $\boldsymbol{c}$, etc.

During the training, we learn a generative model G_θ so that $\boldsymbol{c}_j = G_\theta(\epsilon_j)$ leads to solutions that minimize the loss:

$$\min_{\mathcal{C}:=\{\boldsymbol{c}_j\}} \mathcal{L}(\{\boldsymbol{g}(\boldsymbol{c}_j)\}) + \gamma \sum_j \mathcal{D}(\boldsymbol{g}(\boldsymbol{c}_j)) \quad \text{s.t. } \boldsymbol{c}_j = G_\theta(\epsilon_j) \tag{2}$$

Here ϵ_j is random noise driving generation. This can be written as the following (with $\mathcal{X} := \{\boldsymbol{x}_j\}$ representing a group of generated solutions):

$$\min_\theta \mathcal{L}(\mathcal{X}) + \gamma \sum_j \mathcal{D}(\boldsymbol{x}_j) \quad \text{s.t. } \boldsymbol{x}_j = \boldsymbol{g}(G_\theta(\epsilon_j)) \tag{3}$$

The generative model parameters θ can be learned via backpropagation through the combinatorial solver $\boldsymbol{g}(\cdot)$.

Choices of group loss $\mathcal{L}$. The group loss $\mathcal{L}$ encourages the generated examples to capture the data distribution and can be instantiated using the Wasserstein distance as in Wasserstein GAN (Arjovsky et al., 2017), or Evidence Lower Bound (ELBO) as in VQVAE (Van Den Oord et al., 2017). In the WGAN case, the adversarial discriminator is trained only on examples satisfying the combinatorial constraints ($\boldsymbol{x}$) rather than continuous solutions. Similarly, in VQVAE, the known feasible solutions are passed through the autoencoder followed by the solver $\boldsymbol{g}$ to ensure that the autoencoder loss is only computed on feasible solutions $\boldsymbol{x}$ rather than the continuous latent space $\boldsymbol{c}$.

Choices of individual loss $\mathcal{D}$. We design the individual loss $\mathcal{D}$ to control the quality of individual solutions (e.g., the manufacture cost of a generated molecule $\boldsymbol{x}_j$ should be minimized). In many settings, this individual loss is nontrivial to encode in combinatorial solvers like Gurobi as in (Ferber et al., 2023).

Loss on solution space Ω vs latent space $\mathcal{C}$ Note that to optimize GenCO's objective (Equation 3), we would need to run the combinatorial solver at every iteration, which involves calling MILP solvers and can be slow. On the other hand, if the group loss $\mathcal{L}$ or individual loss $\mathcal{D}$ can be evaluated in the latent space $\mathcal{C}$, then this extra cost is not

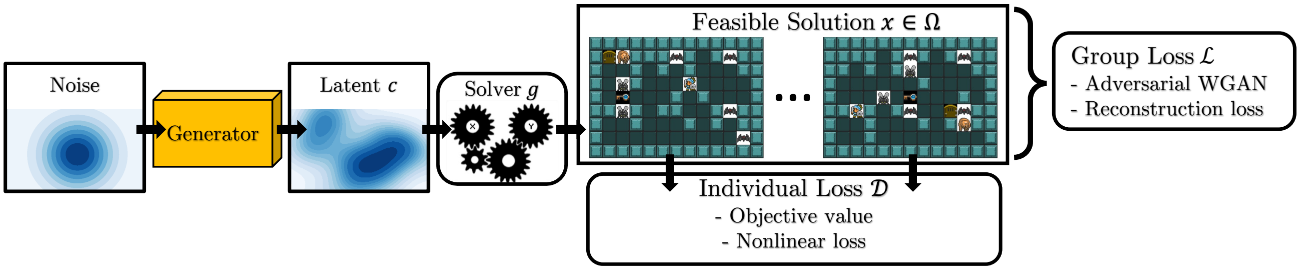

Figure 1. GenCO diagram. A neural generator projects noise to a latent space which then gets used by a solver to generate provably feasible solutions. The solutions are then penalized with generative losses like Wasserstein distance (WGAN) or reconstruction (VQVAE) which can be backpropagated through the full pipeline. Additionally, GenCO can optimize individual objectives.

Experiment	Sol x	Feas Ω	Group Loss $\mathcal{L}$	Ind. Loss $\mathcal{D}$	Latent c	Solver g
Game Design	Game Level	Playability	WGAN (comb)	–	Soft sol.	Gurobi (ILP)
Path Planning	Min Path	Path	WGAN	Min path (comb)	RGB map	Gurobi (LP)
Photonic Device	0/1 Grid	Manufacturing	VQVAE (comb)	Maxwell's sim. (comb)	Soft sol.	Domain spec.

Table 2. Summary of experimental settings highlighting the broad applicability of GenCO to settings with various optimizers, generative models, and loss functions. Losses operating on combinatorial objects rather than continuous latent space are highlighted with (comb).

needed during training. This occurs in the map generation experiments (subsection 4.3), in which an observation $\boldsymbol{c}_i$ is provided (as an RGB image) and the optimal solution $\boldsymbol{x}_i$ can be computed accordingly (e.g., via an edge cost estimator and shortest path solver):

$$\begin{aligned} \min_{\theta} \quad & \mathcal{L}(\mathcal{C}) + \gamma \textstyle\sum_j \mathcal{D}(\boldsymbol{x}_j) \\ \text{s.t.} \quad & \boldsymbol{x}_j = \boldsymbol{g}(\boldsymbol{c}_j) \quad \text{and} \quad \boldsymbol{c}_j = G_\theta(\epsilon_j) \end{aligned} \tag{4}$$

This formulation is related to SurCo-zero (Ferber et al., 2023), in which a latent representation $\boldsymbol{c}$ is optimized so that its inferred solution $\boldsymbol{x} = \boldsymbol{g}(\boldsymbol{c})$ by the solver $\boldsymbol{g}$ minimizes a nonlinear loss function $\mathcal{D}$. Mathematically, the training objective of SurCo is $\min_{\boldsymbol{c}} \mathcal{D}(\boldsymbol{g}(\boldsymbol{c}))$. When there are additional problem descriptions $\boldsymbol{y}_i$ (i.e., SurCo-prior), the latent design vector $\boldsymbol{c}_i = \boldsymbol{c}_\theta(\boldsymbol{y}_i)$ becomes a function of $\boldsymbol{y}_i$ and thus the SurCo objective becomes $\min_\theta \sum_i \mathcal{D}(\boldsymbol{g}(\boldsymbol{c}_\theta(\boldsymbol{y}_i)); \boldsymbol{y}_i)$. The main difference here is that GenCO considers the group loss $\mathcal{L}$ (e.g., matching the data distribution) on newly generated solutions, while SurCo focuses on the quality of solutions to known problem descriptions during training.

In addition to Figure 1, we summarize our proposed method in Algorithm 1. In each iteration, a set of solutions is generated using generator G_θ and a solver $\boldsymbol{g}$ (i.e., forward pass). We then evaluate the loss function using equation 3 (or equation 4). The parameters of G_θ are updated via backpropagation. Here, the differentiation through the combinatorial solver is achieved approximately by employing methods outlined in (Pogančić et al., 2020; Sahoo et al., 2022).

Assumptions GenCO has several assumption on the individual loss $\mathcal{D}$, group loss $\mathcal{L}$, solver g, and latent space c. We assume the individual loss $\mathcal{D}$ is a function that maps a fixed solution to a loss value and provides gradient information useful for gradient descent. Note that if gradient information is not available, a neural network estimator may be learned to approximate the individual loss function. The group loss $\mathcal{L}$ imposes a loss on the distribution of instances generated. With deep generative models, this group loss is generally imposed by using learning-based proxies that act on individual examples. The assumption overall is that $\mathcal{L}$ can be optimized via stochastic gradient descent on the generated examples. We assume the solver g is a function that maps from generated latent information c to a feasible solution $x \in \Omega$. Additionally, we assume that previous work has made g differentiable as a function of c with various techniques proposed such as those described in related work. Finally, we assume the latent space c simply represents values that can be mapped to via a neural network, and that the solver is well defined on that input.

Algorithm 1 GenCO

Output: Trained generator
Initialize generator parameters θ
while not converged **do**
  for $j = 1 \dots K$ **do**
    Sample a noise ϵ_j
    Sample a latent description $\boldsymbol{c}_j = G_\theta(\epsilon_j)$
    Call a solver $\boldsymbol{x}_j = \boldsymbol{g}(\boldsymbol{c}_j)$
  end for
  Compute loss using Equation 3 with $\mathcal{X} := \{\boldsymbol{x}_j\}$ and $\mathcal{C} := \{\boldsymbol{c}_j\}$.
  Backpropagate ∇_θloss to update θ
end while

4. Experiments

We test GenCO on three applications: Zelda game level design, Warcraft path planning map generation, and diverse inverse photonic device design. Settings are summarized in Table 2, and all settings involve combinatorial optimization, making the application of deep generative models nontrivial. In these experiments, GenCO *significantly outperforms the baselines, efficiently finding diverse, realistic solutions that obey combinatorial properties*, paving the way for combining combinatorial optimization with deep generative models.

4.1. Metrics

We evaluate generative performance with density and coverage (Naeem et al., 2020). Density measures, for the average fake data point, how many real examples are nearby. Coverage measures how many real data points have a fake data point nearby. Formally, M fake examples Y_j, are evaluated against N real examples X_i. The metrics use the ball around X_i, with radius $r = \text{NND}_k(X_i)$ being the maximimum distance from X_i to its k nearest real neighbors, to quantify generated density and coverage of the real data distribution. They are expressed as density = $\frac{1}{kM}\sum_{j=1}^{M}\sum_{i=1}^{N} 1_{Y_j \in B(X_i, \text{NND}_k(X_i))}$, and coverage = $\frac{1}{N}\sum_{i=1}^{N} 1_{\exists j \text{s.t.} Y_j \in B(X_i, \text{NND}_k(X_i))}$.

4.2. Explicitly Constrained GAN: Game Level Design

We evaluate GenCO on generating diverse and realistic Zelda game levels. Automatic level generation with playability guarantees helps designers create games with automatically generated content or iterate on development using a few examples. In this setting, we use GenCO to train using a *constrained* WGAN formulation. Here, we are given examples of human-crafted game levels $\mathcal{X}_{\text{train}}$ and are asked to generate fun new levels. The generated levels must be playable in that the player must be able to complete them by moving the character through a route that reaches the destination ($\boldsymbol{x} \in \Omega$). Additionally, the levels should be realistic in that they should be similar to the real game levels as measured by an adversary that is trained to distinguish between real and fake images. This adversary imposes the generative group loss $\mathcal{L}$, effectively approximating the Wasserstein distance (Arjovsky et al., 2017). We use the same 50 Zelda levels as (Zhang et al., 2020), and the same neural network architecture, as we don't tune the hyperparameters for our model in particular but rather compare the different approaches, all else equal.

4.2.1. Baselines

Here, we evaluate GenCO against previous work (Zhang et al., 2020)(*GAN + MILP fix*). This approach trains a standard Wasserstein GAN to approximate human game levels. Specifically, they train the WGAN by alternatively training a generator and a discriminator. The generator is updated to "fool" the discriminator (maximize the loss of the discriminator). The discriminator tries to correctly separate the generated and real game levels into their respective classes. When the practitioner wants to generate a valid level, this approach generates a soft level and fixes it using a MILP that finds the nearest feasible game level in terms of cosine distance (dot product).

We evaluate two variants of GenCO, *GenCO - Fixed Adversary*, which trains against a fixed pretrained adversary from previous work (Zhang et al., 2020), and *GenCO - Updated Adversary*, which updates both the generator and adversary during training. Both approaches are initialized with the fully trained GAN from previous work (Zhang et al., 2020). Importantly, the adversaries act on the generated solution $\boldsymbol{x} \in \Omega$, rather than the latent soft solution $\boldsymbol{c}$.

4.2.2. Results

Results are presented in Table 3 and examples in Figure 2, which includes the performance of the previous approach as well as two variants of GenCO. In these settings, we estimate performance based on sampling 1000 levels. Each level is made out of a grid, with each grid cell having one of 8 components: wall, empty, key, exit, 3 enemy types, and the player. A valid level is one that can be solved by the player in that there is a valid route starting at the player's location, collecting the key, and then reaching the exit. We evaluate the performance of the models using two types of metrics: diversity, as measured by the percentage of unique levels generated, and fidelity, as measured by the average objective quality of a fixed GAN adversary. We evaluate using adversaries from both the previous work and GenCO.

As shown in Table 3, we find that GenCO with an updating adversary generates unique solutions at a much higher rate than previous approaches and also generates solutions that are of higher quality as measured by both the GAN adversary and its own adversary. The adversary quality demonstrates that the solutions are realistic in that neither the adversary from the previous work nor from GenCO is able to distinguish the generated examples from the real examples. This is further demonstrated in Figure 2 with the updated adversary generating realistic and nontrivial game levels. Furthermore, given that the levels are trained on only 50 examples, we can obtain many more game levels. Overall, GenCO - Updated Adversary obtains the best uniqueness and generative loss. However, it obtains neither the best density nor coverage since game levels are generally sparse (the majority of tiles are empty floors), meaning that having a slightly shifted map will result in a higher distance than a completely empty map. As a result, coverage and

Approach	% Unique ↑	Density ↑	Coverage ↑	GAN loss ($\mathcal{L}$) ↓	GenCO adversary ↓
GAN + MILP fix (previous)	0.52	**0.07**	0.94	0.22	0.24
GenCO - Fixed Adversary	0.22	0.05	**0.98**	-1.45	-0.85
GenCO - Updated Adversary	**0.995**	0.06	0.82	**-10.10**	**-4.49**

Table 3. **Game level design** comparison between the previous MILP postprocessing work, GenCO with an updating adversary, and GenCO with a fixed adversary, with all approaches guaranteed to generate feasible levels. We measure % Unique Density and Coverage to estimate diversity and distribution alignment. We also evaluate the adversary's prediction to gauge how well a neural network can distinguish the generated levels from real levels.

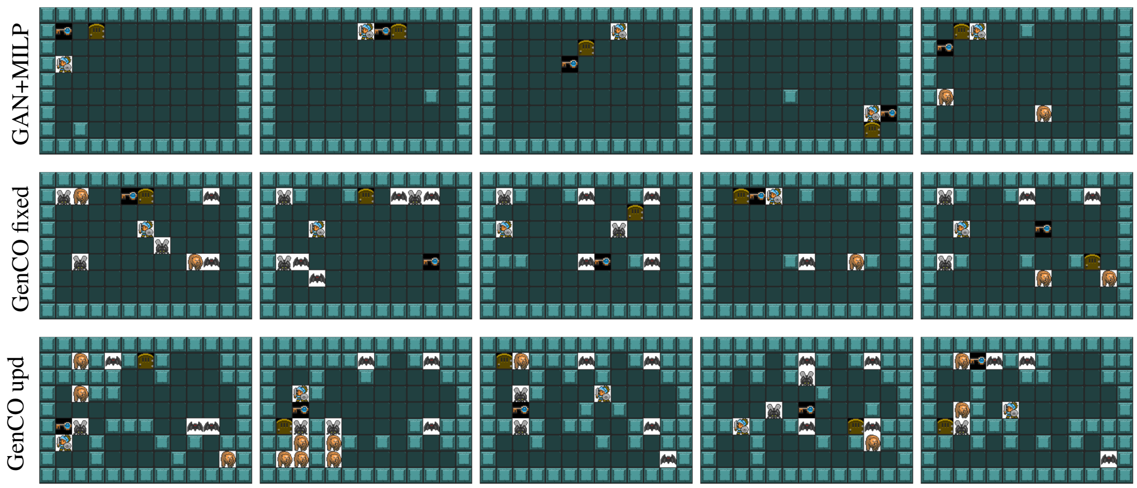

Figure 2. Generated zelda level examples. "GenCO Updated" obtains solutions that seem more realistic than the empty MILP postprocessed GAN levels and GenCO Fixed Adversary levels.

density have counterintuitive behaviors in discrete settings while in standard image domains, samples are continuous and unconstrained. Additionally, density and coverage are relatively low. Note here that both approaches are guaranteed to give playable levels as they are postprocessed to be valid. However, GenCO is able to generate more diverse solutions that are also of higher realism quality.

Uniqueness As shown in Table 3, GenCO, with an updated adversary, obtains the highest percentage of unique solutions, generating 995 unique solutions out of 1000, significantly higher than the 520 unique solutions in previous work. This is likely due to the fact that the generator is trained with the downstream fixing explicitly in the loop. This means that while the previous work may have been able to "hide" from the adversary by generating slightly different continuous solutions, these continuous solutions may project to the same discrete solution. On the other hand, by integrating the fixing into the training loop, GenCO's generator is unable to hide in the continuous solution space and thus is heavily penalized by generating the same solution as the adversary will easily detect those to be originating from the generator. In essence, this makes the adversary's task easier as it only needs to consider distinguishing between valid discrete levels rather than continuous and unconstrained levels. This is also reflected in the adversary quality, where GenCO's adversary is able to distinguish between levels coming from the previous work's generator and the real levels with a much better loss.

4.3. Map generation for path planning

In this experiment, we consider somewhat different and more challenging for GenCO setting of generation of image maps for strategy games like Warcraft. This poses a greater challenge because the generator G is not restricted and can produce any RGB image of a map (group loss $\mathcal{L}$ operates directly on latent space $\mathcal{C}$, i.e., $\mathcal{L}(\mathcal{C})$ in eq. 3). However, we would like to encourage G to generate a map with two main criteria: 1) it should resemble real and diverse game maps, as determined by the group Loss $\mathcal{L}(\mathcal{C})$; 2) *the cost of the shortest path (SP) from the top-left to the bottom-right corners (source and destination) should be minimized* (group loss $\mathcal{D}(\boldsymbol{g}(\boldsymbol{c}))$). Intuitively, the SP corresponds to (mostly) a diagonal part of the image, and a map can include various elements (terrains) like mountains, lakes, forests, and land, each with a specific cost (for example, mountains may have a cost of 3, while land has a cost of 0). To

Approach	Density ↑	Coverage ↑	GAN loss ($\mathcal{L}$) ↓	SP loss ($\mathcal{D}$) ↓
Ordinary GAN	0.81	**0.98**	**0.6147**	36.45
GAN + semantic loss (Di Liello et al., 2020)	**1.09**	**0.98**	0.8994	35.61
GenCO (ours)	0.94	0.93	0.6360	**23.99**

Table 4. **Map generation for Warcraft** performance comparison on 100 instances. The goal is to generate maps which are similar to the data distribution (Density / Coverage), realistic (lower GAN loss or group loss $\mathcal{L}$), and have small shortest paths (SP or individual loss $\mathcal{D}$).

calculate this, we pass the generated image to the fixed ResNet to get the graph representation together with edge costs. This is then followed by the shortest path solver ($\boldsymbol{g}$). The objective is to populate the map with a diverse range of objects while ensuring the shortest path remains low-cost: $\mathcal{L}(\mathcal{C}) + \gamma \sum_j \mathcal{D}(\boldsymbol{g}(\boldsymbol{c}_j))$. We use the same dataset as in (Pogančić et al., 2020) with DCGAN architecture adapted from (Zhang et al., 2020). Note that the maps are unconstrained, representing the generator's latent space. However, the generated feasible solution is a valid path through the map. Additionally, the generative loss acts on the latent space c rather than the feasible solution as we are given maps rather than solutions. The resnet architecture is part of the problem formulation in that it determines costs. More implementation details and experimental settings can be found in Appendix A.

4.3.1. Baselines

We examine various baseline models, including an "Ordinary GAN" that does not incorporate the Shortest Path objective. As in the Zelda experiment, this approach employs a standard Wasserstein GAN architecture closely following (Zhang et al., 2020). However, in this case, we generate images directly instead of encoding game levels. More precisely, we train the WGAN by iteratively training two components: a generator and a discriminator. The generator is fine-tuned to deceive the discriminator to the greatest extent possible, aiming to maximize the discriminator's loss. The discriminator, on the other hand, endeavors to accurately distinguish between generated and authentic game levels and categorize them accordingly.

The next baseline is a "soft" penalty approach: "GAN + semantic loss". It uses a fixed NN to extract representation of the image in the form of cost vector (obtained from the fixed ResNet). This vector is then averaged and added to the final objective as a regularization. The concept is similar to the *semantic loss* appeared in (Di Liello et al., 2020). The difference is that instead of building a circuit (via knowledge compilation) to encode the constraints, we use a fixed pre-trained ResNet as a surrogate to obtain the cost vector. While this baseline considers costs associated with objects, it does not incorporate information about the Shortest Path.

As for the GenCO, it generalizes both of these approaches incorporating combinatorial solver into the pipeline. The detailed algorithm is given in Appendix A. To backpropagate through the solver, we employ the "identity with projection" method from (Sahoo et al., 2022). The remaining settings are similar to "Ordinary GAN".

4.3.2. Results

Quantitative results are showcased in the Table 4. The "Ordinary GAN" focuses exclusively on the GAN's objective (group loss $\mathcal{L}$), without considering the Shortest Path's objective (individual loss $\mathcal{D}$). In contrast, GenCO achieves higher performance with regards to the SP's objective, albeit with a slight reduction in GAN's loss. On the other hand, the "GAN + semantic loss" approach tries to uniformly avoid placing costly objects in any part of the image, whereas we are interested only in the shortest path. While it demonstrates a modest enhancement in the objective $\mathcal{D}$, it experiences a notable decline in terms of group loss ($\mathcal{L}$). This indicates a trade-off between optimizing for the Shortest Path and the GAN's objective. Here, the density and coverage are comparable between all approaches. While GenCO doesn't obtain the highest density and coverage, it does obtain good SP loss and comparable GAN loss with Ordinary GAN, indicating a trade-off between fitting the distribution and obtaining high-quality solutions.

Such quantitative results directly translate into image qualities. In fig. 3, a subset of generated Warcraft map images using various methods is displayed. The "Ordinary GAN" tends to generate maps with elements such as mountains and lakes, which are considered "very costly", particularly along the Shortest Path from the top-left to the bottom-right (which mostly goes through the diagonal). This makes sense since WGAN is trained with no information about grid costs. In contrast, the "GAN + semantic loss" approach produces less costly maps, albeit with reduced diversity. For instance, most part of the image is populated with the same object. On the other hand, GenCO strikes a balance, achieving a cheap Shortest Path while maintaining a diverse range of elements on the map. The Shortest Path is efficient, and the map exhibits a rich variety of features simultaneously.

Additionally, in Table 5 we evaluate sensitivity to generative and shortest path loss trade-off parameter. While the shortest path loss improves with high trade-off parameter, the image quality metrics start to deteriorate rapidly. Eventually, our

γ	SP Loss ($\mathcal{D}$) $\downarrow$	Density $\uparrow$	Coverage $\uparrow$
0	36.45	0.81	0.98
1e-4	27.66	0.93	0.93
1e-3	23.99	0.94	0.93
3e-3	23.76	0.75	0.86
5e-3	18.02	0.49	0.61
1e-2	0.00	0.00	0.00

Table 5. γ ablation, trading off individual and generative loss. High trade-off parameters improve optimization performance while low trade-off values give slightly better generation.

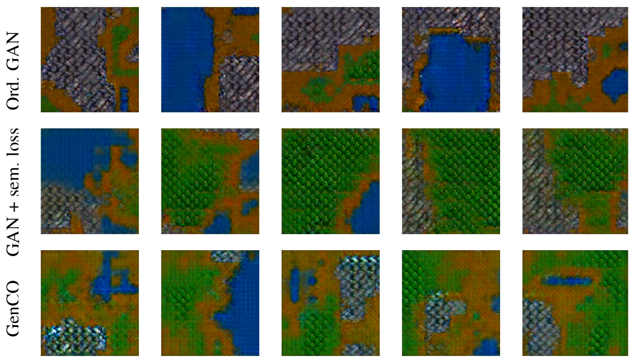

Figure 3. A subset of generated Warcraft map images. "Ordinary GAN" generates "very costly" maps (e.g. mountains, lakes) along the Shortest Path from top-left to bottom-right; "GAN + semantic loss" generates less costly maps but they are *less diverse*; GenCO: SP path is cheap and the map is diverse at the same time.

approach generates blank or empty images, as such images result in the shortest path with zero cost. This behavior could be attributed to the combinatorial nature of the problem.

4.4. Constrained VAE: Photonic Design

We finally evaluate GenCO for training a Vector Quantized Variational Autoencoder (VQVAE) to ensure that all decoded examples satisfy combinatorial manufacturing constraints $\boldsymbol{x} \in \Omega$, are each minimizers of a photonic loss $\mathcal{D}$, and are similar to given solutions via the ELBO loss $\mathcal{L}$. We evaluate GenCO against a generative + postprocess baseline (Zhang et al., 2020) on the inverse photonic design setting.

The inverse photonic design problem (Schubert et al., 2022) asks how to design a device consisting of fixed and void space to route wavelengths of light from an incoming location to desired output locations at high intensity. Here the feasible region consists of satisfying manufacturing constraints that a die with a specific shape must be able to fit in every fixed and void space. Specifically, the fixed and void regions must represent the union of the die shape at different locations. The objective function here consists of a nonlinear but differentiable simulation of the light using Maxwell's equations. Previous work demonstrated an approach for finding a single optimal solution to the problem. However, we propose generating a diverse collection of high-quality solutions using a dataset of known solutions. In practice, there are various applications for generating multiple feasible photonic devices. Firstly, there are many devices that globally minimize the objective function by matching the specification perfectly. With a generator, practitioners can select from a diverse array of solutions that already optimize both specification and satisfy manufacturing constraints, quickly generating more examples if needed. Upon inspecting the solutions, they may select one that satisfies their needs, or derive high-level insights from the diversity of viable solutions to improve the design process. Lastly, optimizing the encoded objective function over the feasible region may not necessarily result in the best solution considering unquantified components like aesthetics, further manual scrutiny, and simply tacit domain expertise. Regardless, the selected design must always be optimal and satisfy the problem constraints. As such, we seek to generate diverse optimal solutions to aid the practitioner when a single solution fails to capture the full range of possibilities.

In this setting, we train a vector quantized variational autoencoder (VQVAE) (Van Den Oord et al., 2017). During training, the encoder encodes a known solution and then decodes an object with the same shape as the input space, using neural networks for the encoder and decoder. The continuous decoded object is then fed into the constrained optimization layer to enforce that the generated solution is feasible. This feasible solution is then used to calculate the reconstruction loss. Furthermore, in this setting we have an optimization-based objective function that consists of the simulation of Maxwell's equations. As such, we ablate the data distribution approximation, and penalty components of GenCO: whether or not to train using the reconstruction loss, and whether or not to penalize generated solutions based on Maxwell's equations. We consider a baseline of training the same generative architecture without combinatorial optimization and then postprocessing generated examples during evaluation using a combinatorial solver. At test time, we sample noise and decode to the original input space. We then obtain a feasible solution using the combinatorial solver. The results in Table 6 include the solution uniqueness rate, the average loss evaluated using Maxwell's equations, as well as the density and coverage. The dataset is obtained by expensively running previous work (Ferber et al., 2023) until we obtain 100 optimal solutions. We evaluate performance on generating 1000 feasible examples.

These results in Table 6 demonstrate that postprocessing obtains very few unique solutions which all have high loss and furthermore don't resemble the data distribution. This is largely due to the method not being trained with the postprocessing end-to-end. Although it closely approximates

Approach	% Unique ↑	Density ↑	Coverage ↑	Avg Solution Loss ↓
VQVAE + postprocess	30.6%	0.009	0.006	1.244
GenCO (reconstruction only)	**100%**	0.148	0.693	1.155
GenCO (objective only)	46.6%	0.013	0.036	**0**
GenCO (reconstruction + objective)	**100%**	**0.153**	**0.738**	**0**

Table 6. **Inverse photonic design** comparison evaluating variants of GenCO using VQVAE against the same model architecture which postprocesses solutions. Solution loss is evaluated using a simulation of Maxwell's equations and is optimal at 0.

the data distribution with continuous and infeasible objects, when these continuous objects are postprocessed to be feasible, they are no longer representative of the data distribution and many continuous solutions collapse to the same discrete solution. Additionally, GenCO obtains high density and coverage compared to the baseline postprocessed VQVAE indicating that training end-to-end helps in this setting.

GenCO - reconstruction only gives many unique solutions that resemble the data distribution. However, the generated devices fail to perform optimally in the photonic task at hand. Disregarding the reconstruction loss and only training the decoder to generate high-quality solutions yields high-quality but redundant solutions. Combining both the generative reconstruction penalty as well as the nonlinear objective, GenCO generates a variety of unique solutions that optimally solve the inverse photonic design problem and are representative of the data distribution.

5. Conclusion

In this paper, we introduce GenCO, a framework for integrating combinatorial constraints in a variety of generative models, and show how it can be used to generate diverse combinatorial solutions to nonlinear optimization problems. Unlike existing generative models and optimization solvers, GenCO guarantees that the generated diverse solutions satisfy combinatorial constraints, and we show empirically that it can optimize nonlinear objectives.

The underlying idea of our framework is to combine the flexibility of deep generative models with the guarantees of optimization solvers. By training the generator end-to-end with a combinatorial solver, GenCO generates diverse and combinatorially feasible solutions, with the generative loss being computed only on feasible solutions.

Currently, many of GenCO's requirements are tied to those of the space of differentiable solvers which is a rapidly evolving field. Some of the key bottlenecks researchers are working to address are scalability, differentiability, and the range of optimization problems that can be differentiated. Another frontier of differentiable solver research is in learning constraints from data. Currently it is assumed that practitioners have access to a mathematical representation of the constraints that can be optimized over. However, in some domains such as molecular design, the constraints such as binding affinity are not available analytically and it is nontrivial to guarantee feasibility using mathematical programs. Learning mathematical representations of these constraints from data is an active area of research.

Additionally, as with many generative models, it is important to keep real-world deployment in mind. Even though GenCO can guarantee feasibility with respect to explicitly encoded constraints, it cannot yet guarantee feasibility with respect to constraints that are desirable but not explicitly encoded. For instance, unless explicitly encoded or leveraging other alignment research, GenCO cannot guarantee fairness, privacy, or positive social impact. Similarly, in image generation, it may be difficult to encode some constraints such as ensuring that pixels representing hands have exactly 5 fingers using a diffusion model in image generation. Overall, GenCO is a step towards enforcing desirable constraints on the outputs of generative models, but it does not immediately solve all problems in ensuring the safety of generative models.

Overall, we propose GenCO, the first generative model that generates diverse solutions, optimizing differentiable nonlinear objectives with guaranteed feasibility for explicitly combinatorial constraints. We test GenCO on three different scenarios: inverse photonic design, game level design, and path planning, showing that GenCO outperforms existing methods along multiple axes. We have tested GenCO on various combinatorial optimization problems and generative settings, including GAN in Zelda game level generation, Warcraft map generation for path planning, and inverse photonic design. These settings require that GenCO handle a variety of generative architectures (GAN / VAE), optimization problems (linear programs, quadratic programs, mixed integer programs), and solvers (SCIP, Gurobi, shortest path solvers, blackbox domain specific solvers), demonstrating the flexibility of our framework for integrating different generative paradigms and optimization methods. Our framework consistently produces diverse and high-quality solutions that satisfy combinatorial constraints, which can be flexibly encoded using general-purpose or domain specific combinatorial solvers.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning, deep generative models, and combinatorial optimization. There are many potential societal consequences of the fields, none which we feel must be specifically highlighted here. Additionally, while our work focuses on guaranteeing the output of generative models satisfies specified constraints, it may be difficult to characterize all types of constraints which may be impossible to specify or enumerate.

Acknowledgment

The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant number 2112533. We also thank the anonymous reviewers for helpful feedback.

References

Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., and Kolter, J. Z. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32, 2019.

Amos, B. and Kolter, J. Z. Optnet: Differentiable optimization as a layer in neural networks. In *Proceedings of Machine Learning Research*, volume 70, 2017.

Arjovsky, M., Chintala, S., and Bottou, L. Wasserstein generative adversarial networks. In *International conference on machine learning*, pp. 214–223. PMLR, 2017.

Bao, F., Yan, D.-M., Mitra, N. J., and Wonka, P. Generating and exploring good building layouts. *ACM Transactions on Graphics*, 32(4), JUL 2013. doi: 10.1145/2461912.2461977.

Berthet, Q., Blondel, M., Teboul, O., Cuturi, M., Vert, J.-P., and Bach, F. Learning with differentiable pertubed optimizers. *Advances in neural information processing systems*, 33:9508–9519, 2020.

Chao, X., Cao, J., Lu, Y., Dai, Q., and Liang, S. Constrained generative adversarial networks. *IEEE Access*, 9:19208–19218, 2021. doi: 10.1109/ACCESS.2021.3054822.

Christensen, T., Loh, C., Picek, S., Jing, L., Fisher, S., Ceperic, V., Joannopoulos, J. D., Soljacic, M., and Jakobovic, D. Predictive and generative machine learning models for photonic crystals. *NanoPhotonics*, 9(13, SI): 4183–4192, OCT 2020. doi: 10.1515/nanoph-2020-0197.

Deaton, J. D. and Grandhi, R. V. A survey of structural and multidisciplinary continuum topology optimization: post 2000. *Structural and Multidisciplinary Optimization*, 49: 1–38, 2014.

Demirovic, E., Stuckey, P. J., Bailey, J., Chan, J., Leckie, C., Ramamohanarao, K., and Guns, T. An investigation into prediction plus optimisation for the knapsack problem. In *16th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*, volume 11494 of *Lecture Notes in Computer Science*, pp. 241–257, 2019a.

Demirovic, E., Stuckey, P. J., Bailey, J., Chan, J., Leckie, C., Ramamohanarao, K., and Guns, T. Predict plus optimise with ranking objectives: Exhaustively learning linear functions. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pp. 1078–1085, 2019b.

Di Liello, L., Ardino, P., Gobbi, J., Morettin, P., Teso, S., and Passerini, A. Efficient generation of structured objects with constrained adversarial networks. *Advances in neural information processing systems*, 33:14663–14674, 2020.

Donti, P., Amos, B., and Kolter, J. Z. Task-based end-to-end model learning in stochastic optimization. In *Advances in Neural Information Processing Systems (NIPS 2017)*, pp. 1320–1332, 2017.

Donti, P. L., Rolnick, D., and Kolter, J. Z. Dc3: A learning method for optimization with hard constraints. In *International Conference on Learning Representations*, 2020.

Elmachtoub, A. N. and Grigas, P. Smart "predict, then optimize". arXiv:1710.08005, October 22 2017.

Ferber, A., Wilder, B., Dilkina, B., and Tambe, M. Mipaal: Mixed integer program as a layer. In *AAAI Conference on Artificial Intelligence*, volume 34, pp. 1504–1511. Assoc Advancement Artificial Intelligence, 2020.

Ferber, A. M., Huang, T., Zha, D., Schubert, M., Steiner, B., Dilkina, B., and Tian, Y. Surco: Learning linear surrogates for combinatorial nonlinear optimization problems. In *International Conference on Machine Learning*, pp. 10034–10052. PMLR, 2023.

Frangioni, A. About lagrangian methods in integer optimization. *Annals of Operations Research*, 139:163–193, 2005.

Gomes, C. P., Sabharwal, A., and Selman, B. Near-uniform sampling of combinatorial spaces using xor constraints. *Advances In Neural Information Processing Systems*, 19, 2006.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In

Advances in Neural Information Processing Systems, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

Guyomarch, J. Warcraft ii open-source map, 2017. URL http://github.com/war2/war2edit.

Hooshmand, A. and Campbell, M. I. Truss layout design and optimization using a generative synthesis approach. *Computers & Structures*, 163:1–28, JAN 15 2016. doi: 10.1016/j.compstruc.2015.09.010.

Hua, H., Hovestadt, L., Tang, P., and Li, B. Integer programming for urban design. *European Journal of Operational Research*, 274(3):1125–1137, MAY 1 2019. doi: 10.1016/j.ejor.2018.10.055.

Kim, S., Noh, J., Gu, G. H., Aspuru-Guzik, A., and Jung, Y. Generative adversarial networks for crystal structure prediction. *ACS Central Science*, 6(8):1412–1420, AUG 26 2020. doi: 10.1021/acscentsci.0c00426.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kotary, J., Fioretto, F., van Hentenryck, P., and Wilder, B. End-to-end constrained optimization learning: A survey. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence, IJCAI 2021*, pp. 4475–4482. International Joint Conferences on Artificial Intelligence, 2021.

Ling, C. K., Fang, F., and Kolter, J. Z. What game are we playing? end-to-end learning in normal and extensive form games. In *Proceedings of The Twenty-Seventh International Joint Conference on Artificial Intelligence*, pp. 396–402, 2018.

Liu, H. and Grigas, P. Risk bounds and calibration for a smart predict-then-optimize method. *Advances in Neural Information Processing Systems*, 34:22083–22094, 2021.

Lopez-Piqueres, J., Chen, J., and Perdomo-Ortiz, A. Symmetric tensor networks for generative modeling and constrained combinatorial optimization. *Machine Learning-Science and Technology*, 4(3), SEP 1 2023. doi: 10.1088/2632-2153/ace0f5.

Mandi, J. and Guns, T. Interior point solving for lp-based prediction+ optimisation. *Advances in Neural Information Processing Systems*, 33:7272–7282, 2020.

Mandi, J., Demirovic, E., Stuckey, P. J., and Guns, T. Smart predict-and-optimize for hard combinatorial optimization problems. In *AAAI Conference on Artificial Intelligence*, volume 34, pp. 1603–1610, 2020.

Mandi, J., Bucarey, V., Tchomba, M. M. K., and Guns, T. Decision-focused learning: through the lens of learning to rank. In *International Conference on Machine Learning*, pp. 14935–14947. PMLR, 2022.

Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., and De Raedt, L. Deepproblog: Neural probabilistic logic programming. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Misino, E., Marra, G., and Sansone, E. Vael: Bridging variational autoencoders and probabilistic logic programming. *Advances in Neural Information Processing Systems*, 35: 4667–4679, 2022.

Naeem, M. F., Oh, S. J., Uh, Y., Choi, Y., and Yoo, J. Reliable fidelity and diversity metrics for generative models. In *International Conference on Machine Learning*, pp. 7176–7185. PMLR, 2020.

Nauata, N., Hosseini, S., Chang, K.-H., Chu, H., Cheng, C.-Y., and Furukawa, Y. House-gan plus plus : Generative adversarial layout refinement network towards intelligent computational agent for professional architects. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 13627–13636, 2021. doi: 10.1109/CVPR46437.2021.01342.

Niepert, M., Minervini, P., and Franceschi, L. Implicit mle: Backpropagating through discrete exponential family distributions. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Ozair, S., Li, Y., Razavi, A., Antonoglou, I., van den Oord, A., and Vinyals, O. Vector quantized models for planning. In *International Conference on Machine Learning, Vol 139*, volume 139 of *Proceedings of Machine Learning Research*, 2021.

Para, W., Guerrero, P., Kelly, T., Guibas, L. J., and Wonka, P. Generative layout modeling using constraint graphs. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6690–6700, 2021.

Paulus, A., Rolínek, M., Musil, V., Amos, B., and Martius, G. Comboptnet: Fit the right np-hard problem by learning integer programming constraints. In *International Conference on Machine Learning*, pp. 8443–8453. PMLR, 2021.

Peng, C.-H., Yang, Y.-L., Bao, F., Fink, D., Yan, D.-M., Wonka, P., and Mitra, N. J. Computational network design from functional specifications. *ACM Transactions on Graphics*, 35(4), JUL 2016. doi: 10.1145/2897824.2925935. ACM SIGGRAPH Conference, Anaheim, CA, JUL 24-28, 2016.

Pereira, T., Abbasi, M., Ribeiro, B., and Arrais, J. P. Diversity oriented deep reinforcement learning for targeted molecule generation. *Journal of cheminformatics*, 13(1): 21, 2021.

Perrault, A., Wilder, B., Ewing, E., Mate, A., Dilkina, B., and Tambe, M. End-to-end game-focused learning of adversary behavior in security games. In *AAAI Conference on Artificial Intelligence*, volume 34, pp. 1378–1386. Assoc Advancement Artificial Intelligence, 2020.

Pogančić, M. V., Paulus, A., Musil, V., Martius, G., and Rolinek, M. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BkevoJSYPB.

Rolínek, M., Swoboda, P., Zietlow, D., Paulus, A., Musil, V., and Martius, G. Deep graph matching via blackbox differentiation of combinatorial solvers. In *ECCV*, pp. 407–424. Springer, 2020.

Sahoo, S. S., Vlastelica, M., Paulus, A., Musil, V., Kuleshov, V., and Martius, G. Backpropagation through combinatorial algorithms: Identity with projection works. In *International Conference on Learning Representations*, 2022.

Schubert, M. F., Cheung, A. K. C., Williamson, I. A. D., Spyra, A., and Alexander, D. H. Inverse design of photonic devices with strict foundry fabrication constraints. *ACS Photonics*, 9(7):2327–2336, 2022. doi: 10.1021/acsphotonics.2c00313.

Shah, S., Wang, K., Wilder, B., Perrault, A., and Tambe, M. Decision-focused learning without decision-making: Learning locally optimized decision losses. *Advances in Neural Information Processing Systems*, 35:1320–1332, 2022.

Shah, S., Perrault, A., Wilder, B., and Tambe, M. Leaving the nest: Going beyond local loss functions for predict-then-optimize. *arXiv preprint arXiv:2305.16830*, 2023.

Sun, Z. and Yang, Y. Difusco: Graph-based diffusion solvers for combinatorial optimization. *arXiv preprint arXiv:2302.08224*, 2023.

Tang, H., Zhang, Z., Shi, H., Li, B., Shao, L., Sebe, N., Timofte, R., and Van Gool, L. Graph transformer gans for graph-constrained house generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2173–2182, 2023.

Van Den Oord, A., Vinyals, O., et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Wang, P.-W., Donti, P. L., Wilder, B., and Kolter, Z. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, 2019.

Wilder, B., Dilkina, B., and Tambe, M. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *AAAI Conference on Artificial Intelligence*, pp. 1658+, 2019a.

Wilder, B., Ewing, E., Dilkina, B., and Tambe, M. End to end learning and optimization on graphs. In *Conference on Neural Information Processing Systems*, volume 32, 2019b.

Wu, W., Fan, L., Liu, L., and Wonka, P. Miqp-based layout design for building interiors. *39th Annual Conference of the European-Association-for-Computer-Graphics*, 37 (2):511–521, MAY 2018. doi: 10.1111/cgf.13380.

Yeh, Y.-T., Yang, L., Watson, M., Goodman, N. D., and Hanrahan, P. Synthesizing open worlds with constraints using locally annealed reversible jump mcmc. *ACM Transactions on Graphics (TOG)*, 31(4):1–11, 2012.

Zhang, D. W., Rainone, C., Peschl, M., and Bondesan, R. Robust scheduling with gflownets. In *The Eleventh International Conference on Learning Representations*, 2022.

Zhang, H., Fontaine, M., Hoover, A., Togelius, J., Dilkina, B., and Nikolaidis, S. Video game level repair via mixed integer linear programming. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 16, pp. 151–158, 2020.

Zhao, S. and You, F. Distributionally robust chance constrained programming with generative adversarial networks (gans). *AICHE*, 66(6), JUN 2020. doi: 10.1002/aic.16963.

Zharmagambetov, A., Amos, B., Ferber, A., Huang, T., Dilkina, B., and Tian, Y. Landscape surrogate: Learning decision losses for mathematical optimization under partial information. *arXiv preprint arXiv:2307.08964*, 2023.

A. Detailed experimental setup for map generation (section 4.3)

A.1. Pseudocode

Algorithm 2 GenCO Penalized generator training for GANs.

```
1:  Initialize generator parameters θ_gen
2:  Initialize adversary parameters θ_adv
3:  Input: distribution of the true dataset (c_true ~ p_data(c)), GAN's objective L(θ_gen, θ_adv)
4:  for epoch e do
5:    for batch b do
6:      Sample a noise ϵ
7:      Sample true examples from dataset c_true ~ p_data(c)
8:      Sample fake examples using c_fake ~ G(ϵ; θ_gen)
9:      Transform c_fake into the coefficients of the optimization problem: c = f(c_fake)
10:     Solve: x* = g(c) = arg min_{x∈Ω} c^T x
11:     Backpropagate ∇_{θ_gen} [L(c_fake; θ_gen, θ_adv) + βc^T x*] to update θ_gen
12:     Backpropagate ∇_{θ_adv} [−L(c_fake; θ_gen, θ_adv) + L(c_real; θ_gen, θ_adv)] to update θ_adv
13:   end for
14: end for
```

Algorithm A.1 provides a detailed description of the GenCO framework in our penalty formulation and utilized in section 4.3. In this process, we sample both real and synthetic data, drawing from the true data distribution and the generator G respectively (lines 7–8). Subsequently, the synthetic data undergoes a fixed mapping (e.g. ResNet in our experiments), called cost neural net (or cost NN), to obtain coefficients for the optimization problem (the edge weights for the Shortest Path). Following this, we invoke a solver that provides us with a solution and its associated objective (lines 10–11). We then proceed to update the parameters of the generator G using both the GAN's objective (group loss $\mathcal{L}$) and the solver's objective (individual loss $\mathcal{D}$). Finally, we refine the parameters of the adversary (discriminator) in accordance with the standard GAN's objective.

A.2. Settings

We employ ResNet as the mapping $f(\cdot)$ from the above Algorithm A.1, which transforms an image of a map into a 12×12 grid representation of a weighted directed graph: $f : \Re^{96\times96\times3} \to \Re^{12\times12}$. The first five layers of ResNet18 are pre-trained (75 epochs, Adam optimizer with lr=$5e-4$) using the dataset from (Pogančić et al., 2020), comprising 10,000 labeled pairs of image–grid (refer to the dataset description below). Following pretraining, we feed the output into the Shortest Path solver, using the top-left point as the source and the bottom-right point as the destination. The resulting objective value from the Shortest Path corresponds to $\boldsymbol{g}$.

Dataset The dataset used for training in the Shortest Path problem with $k = 12$ comprises 10,000 randomly generated terrain maps from the Warcraft II tileset (Pogančić et al., 2020) (adapted from (Guyomarch, 2017)). These maps are represented on a 12×12 grid, with each vertex denoting a terrain type and its associated fixed cost. For example, a mountain terrain may have a cost of 9, while a forest is assigned a cost of 1. It's important to note that in the execution of Algorithm A.1, we don't directly utilize the actual (ground truth) costs, but rather rely on ResNet to generate them.

A.3. Architecture

We employ similar DCGAN architecture taken from (Zhang et al., 2020) (see fig. 3 therein). Input to generator is 128 dimensional vector sampled from Gaussian noise centered around 0 and with a std of 1. Generator consists of five (256–128–64–32–16) blocks of transposed convolutional layers, each with 3×3 kernel sizes and batch normalization layers in between. Discriminator follows by mirroring the same architecture in reverse fashion. The discriminator mirrors this architecture in reverse order. The entire structure is trained using the WGAN algorithm, as described in(Zhang et al., 2020).

B. GANs with combinatorial constraints

In the generative adversarial networks (GAN) setting, the generative objectives are measured by the quality of a worst-case adversary, which is trained to distinguish between the generator's output and the true data distribution. Here, we use the combinatorial solver to ensure that the generator's output is always feasible and that the adversary's loss is evaluated using only feasible solutions. This not only ensures that the pipeline is more aligned with the real-world deployment but also that the discriminator doesn't have to dedicate model capacity to detecting infeasibility as indicating a solution is fake and instead dedicate model capacity to distinguishing between real and fake inputs, assuming they are all valid. Furthermore, we can ensure that the objective function is optimized by penalizing the generator based on the generated solutions' objective values:

$$\mathcal{L}(G(\epsilon;\theta_{gen})) = \mathbb{E}_\epsilon\left[\log(1 - f_{\theta_{adv}}(G(\epsilon;\theta_{gen})))\right] \tag{5}$$

where $f_{\theta_{adv}}$ is an adversary (a.k.a. discriminator) and putting this in the context of equation 1 leads to:

$$\min_{\theta_{gen}} \quad \mathcal{L}(S(\tilde{G}(\epsilon;\theta_{gen}))) = \mathbb{E}_\epsilon\left[\log(1 - f_{\theta_{adv}}(S(\tilde{G}(\epsilon;\theta_{gen}))))\right] \tag{6}$$

where $\tilde{G}$ is unconstrained generator and S is a surrogate combinatorial solver as described above. Here, we also have adversary's learnable parameters θ_{adv}. However, that part does not depend on combinatorial solver and can be trained as in usual GAN's. The algorithm is presented in pseudocode B.

Algorithm 3 GenCO in the constrained generator setting

Initialize generator parameters θ_{gen}
Initialize adversary parameters θ_{adv}
for epoch e **do**
  for batch b **do**
    Sample problem ϵ
    Sample true examples from dataset $x_{true} \sim p_{\text{data}}(x)$
    Sample linear coefficients $c \sim G(\epsilon;\theta_{gen})$
    Solve $x^* = \arg\max_{x\in\Omega} c^T x$
    Backpropagate $\nabla_{\theta_{gen}}\mathcal{L}(x^*;\theta_{gen})$ to update generator (equation 6)
    Backpropagate $\nabla_{\theta_{adv}}\left[\log(f_{\theta_{adv}}(x_{true})) - \mathcal{L}(x^*;\theta_{gen})\right]$ to update adversary
  end for
end for

C. GenCO – VQVAE

The formulation below spells out the VQVAE training procedure. Here, we simply train VQVAE on a dataset of known objective coefficients, which solves the problem at hand. A variant of this also puts the decision-focused loss on the generated objective coefficients, running optimizer g on the objective coefficients to get a solution and then computing the objective value of the solution.

$$\mathcal{L}(\mathcal{X}) = \text{ELBO}(\boldsymbol{x},\theta,E) = \mathbb{E}_{q_\theta(z|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|z)] - \beta\cdot D_{\text{KL}}(q_\theta(z|\boldsymbol{x})||p(z)) + \gamma\cdot\|sg(\mathbf{e}_k) - \mathbf{z}_{e,\theta}\|_2^2 \tag{7}$$

Here z is an embedding vector, c is the objective coefficients, $\log p_\theta(\boldsymbol{x}|z)$ is a loss calculated via the mean squared error between the decoder output and the original input objective coefficients, $q_\theta(z|\boldsymbol{x})$ is the encoder, $p(z)$ is the prior, and $sg(\cdot)$ is the stop gradient operator, E is a discrete codebook that is used to quantize the embedding.

$$\mathcal{D} = \mathbb{E}_{c\sim p_\theta(c|z)}\left[f_{\text{obj}}(g(c;y))\right] \tag{8}$$

The algorithm below maximizes a combination of the losses in Equation equation 7 and Equation equation 8.

Algorithm 4 Constrained Generator Training for VQVAE

```
Given: Training data distribution D over problem info y and known high-quality solutions x, regularization weight β, linear surrogate solver g_solver, nonlinear objective f_objective.
Output: Trained encoder f_enc, decoder f_dec, and codebook E
------------------------------------------------------------
Initialize the parameters of the encoder f_enc, decoder f_dec, and the codebook E = {e_1, e_2, ..., e_K} with K embedding vectors
for t = 1 to T do
    Sample y, x from the distribution D
    Compute the encoder output z_e = f_enc(y, x)
    Find the nearest embedding vector z_q = arg min_{e∈E} ||z_e − e||_2^2
    Compute the quantization loss L_quant = ||z_e − z_q||_2^2
    Decode the embedding c̃ = f_dec(y, z_q)
    Solve x̃ = arg max_{x∈Ω} c^T x
    Compute the reconstruction loss L_recon = ||x − x̃||_2^2
    Compute the optimization loss L_opt = f_objective(x̃)
    Compute the total loss: L_total = L_recon + β_1 L_quant + β_2 L_opt
    Update the parameters of the encoder, decoder, and codebook to minimize L_total
end for
```