A DETAILED EXPERIMENTAL SETUP FOR MAP GENERATION

A.1 PSEUDOCODE

Algorithm 3: 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 ($c_{true} \sim p_{\text{data}(c)}$), GAN's objective $\mathcal{L}(\theta_{gen}, \theta_{adv})$;
4: **for** epoch e **do**
5: **for** batch b **do**
6: Sample a noise ϵ;
7: Sample true examples from dataset $c_{true} \sim p_{\text{data}}(c)$;
8: Sample fake examples using $c_{fake} \sim G(\epsilon; \theta_{gen})$;
9: Transform c_{fake} into the coefficients of the optimization problem: $c = g(c_{fake})$;
10: Solve: $x^* = \arg\min_{x \in \Omega} c^T z$;
11: Backpropagate $\nabla_{\theta_{gen}} \left[\mathcal{L}(c_{fake}; \theta_{gen}, \theta_{adv}) + \beta c^T x^*\right]$ to update θ_{gen};
12: Backpropagate $\nabla_{\theta_{adv}} \left[-\mathcal{L}(c_{fake}; \theta_{gen}, \theta_{adv}) + \mathcal{L}(c_{real}; \theta_{gen}, \theta_{adv})\right]$ to update θ_{adv};
13: **end for**
14: **end for**

Algorithm 3 provides a detailed description of the GenCO framework in our penalty formulation and utilized in section 4.2. 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, specifically 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 and the solver's objective. 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 $g(\cdot)$ from Algorithm 3, which transforms an image of a map into a 12×12 grid representation of a weighted directed graph: $g : \Re^{96 \times 96 \times 3} \to \Re^{12 \times 12}$. 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 f.

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 3, 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:

$$\mathcal{L}_{\text{gen}}(G(\epsilon;\theta_{gen})) = \mathbb{E}_{\epsilon}\left[\log(1 - f_{\theta_{adv}}(G(\epsilon;\theta_{gen})))\right] \tag{4}$$

where $f_{\theta_{adv}}$ is an adversary (a.k.a. discriminator) and putting this in the context of equation 2 leads to:

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

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 4.

Algorithm 4: GenCO 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 ~ p_data(x);
    Sample linear coefficients c ~ G(ϵ; θ_gen);
    Solve x* = arg max_{x∈Ω} c^T x;
    Backpropagate ∇_{θ_gen} L_gen(x*; θ_gen) to update generator (equation 5);
    Backpropagate ∇_{θ_adv} [log(f_{θ_adv}(x_true)) − L_gen(x*; θ_gen)] 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_{solver} on the objective coefficients to get a solution and then computing the objective value of the solution.

$$L_{\text{ELBO}}(c,\theta,E) = \mathbb{E}_{q_\theta(z|c)}[\log p_\theta(c|z)] - \beta \cdot D_{\text{KL}}(q_\theta(z|c)||p(z)) + \gamma \cdot \|sg(\mathbf{e}_k) - \mathbf{z}_{e,\theta}\|_2^2 \tag{6}$$

Here z is an embedding vector, c is the objective coefficients, $\log p_\theta(c|z)$ is a loss calculated via the mean squared error between the decoder output and the original input objective coefficients, $q_\theta(z|c)$ 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.

$$\mathcal{L}_{\text{optimization}} = \mathbb{E}_{c\sim p_\theta(c|z)}\left[f_{\text{obj}}(g_{\text{solver}}(c;y))\right] \tag{7}$$

The algorithm below maximizes a combination of the losses in Equation equation 6 and Equation equation 7.

Algorithm 5: Constrained generator Training for VQVAE

Input:Training data distribution D over problem info y and known high-quality objective coefficients c, regularization weight β, linear surrogate solver g_{solver}, nonlinear objective $f_{\text{objective}}$.

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 $E = \{e_1, e_2, \ldots, e_K\}$ with K embedding vectors;

for $t = 1$ **to** T **do**

- Sample y, c from the distribution D;
- Compute the encoder output $z_e = f_{\text{enc}}(y, c)$;
- Find the nearest embedding vector $z_q = \arg\min_{e \in E} \|z_e - e\|_2^2$;
- Compute the quantization loss $L_{\text{quant}} = \|z_e - z_q\|_2^2$;
- Compute the reconstruction $\tilde{c} = f_{\text{dec}}(y, z_q)$;
- Compute the reconstruction loss $L_{\text{recon}} = \|c - \tilde{c}\|_2^2$;
- GenCo Variant: Compute the optimization loss $L_{\text{opt}} = f_{\text{objective}}(g_{\text{solver}}(\tilde{c}, y))$;
- Compute the total loss: $L_{\text{total}} = L_{\text{recon}} + \beta_1 L_{\text{quant}} + \beta_2 L_{\text{opt}}$;
- Update the parameters of the encoder, decoder, and codebook to minimize L_{total};

end