

Evolving Through the Looking Glass: Learning Improved Search Spaces with Variational Autoencoders

Peter J. Bentley^{1,2}(✉), Soo Ling Lim¹, Adam Gaier², and Linh Tran²

¹ Department of Computer Science, University College London (UCL), London, UK
p.bentley@cs.ucl.ac.uk

² Autodesk Research, London, UK

Abstract. Nature has spent billions of years perfecting our genetic representations, making them evolvable and expressive. Generative machine learning offers a shortcut: learn an evolvable latent space with implicit biases towards better solutions. We present SOLVE: Search space Optimization with Latent Variable Evolution, which creates a dataset of solutions that satisfy extra problem criteria or heuristics, generates a new latent search space, and uses a genetic algorithm to search within this new space to find solutions that meet the overall objective. We investigate SOLVE on five sets of criteria designed to detrimentally affect the search space and explain how this approach can be easily extended as the problems become more complex. We show that, compared to an identical GA using a standard representation, SOLVE with its learned latent representation can meet extra criteria and find solutions with distance to optimal up to two orders of magnitude closer. We demonstrate that SOLVE achieves its results by creating better search spaces that focus on desirable regions, reduce discontinuities, and enable improved search by the genetic algorithm.

Keywords: Variational autoencoder · Latent variable evolution · Generative machine learning · Genetic algorithm

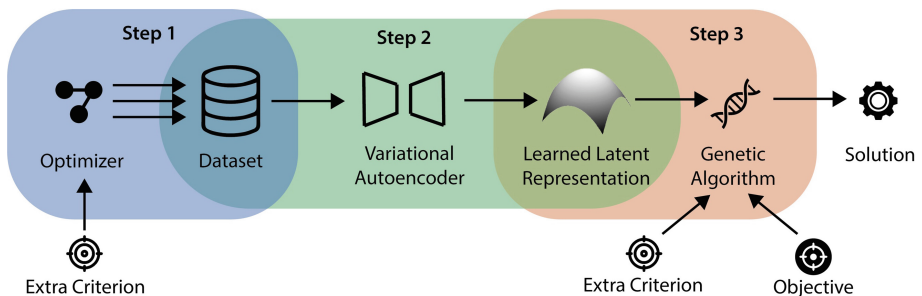


Fig. 1. Search space Optimization with Latent Variable Evolution (SOLVE). An optimizer produces a dataset of random solutions satisfying an extra criterion (e.g., constraint or secondary objective). A variational autoencoder learns this dataset and produces a learned latent representation biased towards the desired region of the search space. This learned representation is then used by a genetic algorithm to find solutions that meet the objective and extra criterion together.

1 Introduction

In nature, the mapping from gene to phenotypic effect is hugely complex. Any new human genetic trait will be propagated through trillions of cells, which via a complex developmental process involving gene regulatory networks, intercellular communication, pattern formation and differentiation, results in altered phenotypic characteristics: an improved ability to taste bitter substances; an increased likelihood of developing an immunity to certain diseases; a reduced propensity to be creative. There is pervasive pleiotropy in the human genome [1] and yet like all living organisms, we have evolved and continue to do so, with most of our offspring remaining viable as healthy functioning organisms. Somehow, nature has learned a genetic representation that is astonishingly expressive, searchable, and despite constant genetic innovation, maps to viable living creatures that satisfy the multiple criteria of survival.

In evolutionary computation our hand-designed genetic representations are usually mapped directly to phenotypic effects so that we have minimal pleiotropy. Yet if we introduce any additional criterion or constraint, our evolutionary algorithms still struggle. From an optimization perspective, the additional criteria distort the search space, adding discontinuities and deceptive regions that result in ineffective optimization [2]. Typical solutions involve modifying the search operators or the optimization algorithm to overcome problems in the search space [3–5]. These specialized algorithms may need tuning for each problem and require expertise, which may not always be available.

Recent work has used autoencoders to learn representations when performing black box optimization. We extend our previous work [6] and propose a variation of this idea: *learn a better search space*. In contrast to previous work which aims to reduce the problem dimensionality, here we investigate the idea that a generative machine learning approach could map a difficult-to-search genotype space into an easier-to-search latent space. This new space would be biased towards solutions that satisfy the additional criteria to the problem, while at the same time smoothing out discontinuities in the space, effectively achieving evolution of evolvability [9] by using deep learning as a shortcut.

To achieve this objective, we introduce SOLVE: Search space Optimization with Latent Variable Evolution (Fig. 1). SOLVE generates a dataset from problem criteria (a constraint, secondary objective, or heuristic). A Variational Autoencoder (VAE) [10, 11] is applied to the dataset to generate a learned latent representation biased towards solutions that satisfy the criteria. A Genetic Algorithm is then used to evolve in the corresponding latent search space and find solutions that also satisfy the overall objective. We provide a step-by-step investigation of this approach, examining improvements provided for optimization in latent space vs. genotype search space through a selection of different types of criterion. To better isolate the effect of the learned search space, we employ only a very simple optimizer.

SOLVE is not a multi-objective optimization algorithm – it aims to find single solutions that meet one objective and one or more extra criteria. SOLVE is also not a constraint satisfaction approach – while constraints can be recast as additional criteria [6], it cannot guarantee that they will always be met. Instead, SOLVE is a search space optimizer. It is suitable for difficult problems that can be broken down into separate objectives, criteria and/or constraints, or that may have important domain knowledge available in the form

of heuristics or required features. SOLVE shows for the first time that a VAE can be used to map a difficult-to-search space into a latent space that is easier to search, without relying on parameter reduction.

2 Background

2.1 Variational Autoencoders

An autoencoder [7] is a neural network originally used for feature learning or dimensionality reduction, but its concept became widely popular for learning generative models of the data. An autoencoder consists of two parts - an encoder p and a decoder q . The encoder maps the observations x to a (lower dimensional) embedding space z , whereas the decoder maps the embeddings back to the original observation space. We denote the reconstructed data as x' . The autoencoder is trained to minimize the reconstruction error between observation and decoded output and simultaneously project the observations into the lower dimensional “bottleneck”.

The Variational Autoencoder (VAE) is a probabilistic autoencoder proposed concurrently by Kingma et al. [8] and Rezende et al. [9]. The architecture of a VAE is similar to the one of autoencoders described above. However, instead of encoding an observation as a single point, VAEs encode it as a distribution over the latent space. Due to its simplicity and the resulting analytical solution for the regularization, the distribution is set to be an isotropic Gaussian distribution. From a Bayesian perspective, VAEs maximize a lower bound on the log-marginal likelihood, which is given by

$$\log p(x) \geq \underbrace{\mathbb{E}_{q_\phi} [\log p_\theta(x|z)]}_{\text{log likelihood}} - \underbrace{D_{KL}(q(z|x)||p(z))}_{\text{latent space regularization}} =: -\mathcal{L}_{VAE}(x; \theta, \phi) \quad (1)$$

where θ are the model parameters of encoder p and ϕ are the model parameters of decoder q . The expected log-likelihood or “reconstruction”, the first term of Eq. (1), is proportional to the mean squared loss between decoded output and input observation if the output distribution is Gaussian. The second term of Eq. (1) denotes the Kullback-Leibler (KL) divergence and measures the similarity between the latent variable distribution q and a chosen prior p . In the common VAE, the latent variable distribution is isotropic Gaussian and parameterized through the neural network q_ϕ and the prior distribution is a Gaussian distribution with zero mean and diagonal unit variance $\mathcal{N}(0, \mathbf{I})$. The KL divergence has an analytical solution. The final minimization objective is

$$\mathcal{L}_{VAE}(x; \theta, \phi) = -\mathbb{E}_{q_\phi} [\log p_\theta(x|z)] + D_{KL}(q(z|x)||p(z)) \propto \|x' - x\|^2 + D_{KL}(q(z|x)||p(z)) \quad (2)$$

2.2 Evolving Latent Variables

The ability of deep learning systems like VAEs to learn representations has not gone unnoticed by the evolutionary optimization community. Latent variable evolution (LVE)

techniques first train generative models on existing datasets, such as video game levels, fingerprints, or faces and then use evolution to search the latent spaces of those models. Game levels can be optimized for a high or low number of enemies [10], fingerprints to defeat biometric security [11], celebrity look-alike faces can be generated with varied hair and eye colors [12], and VAEs can learn alternative search spaces for GP [13]. When no existing dataset of solutions is available, these solution sets must be generated. In [14, 15], solutions were collected by saving the champion solutions found after repeatedly running an optimizer on the problem. A representation learned from this set of champions can then be effective in solving similar sets of problems.

Quality-diversity [16, 17] approaches have been blended with LVE to improve optimization in high dimensional spaces. DDE-Elites [18] learns a ‘data-driven encoding’ by training a VAE based on the current collection, and uses that encoding together with the direct encoding to accelerate search. The related Policy Manifold Search [19] uses a VAE trained in the same way as part of a mutation operator. Standard and surrogate-assisted GAs have also been shown to benefit from having learned encodings “in-the-loop” in order to better tackle high-dimensional search spaces [20, 21].

A learned representation does more than reduce the dimensionality of the search space – it reduces the range of solutions which can be generated [22]. A model trained on Mario levels will never produce Pac-man levels, a model trained on predominantly white faces will only produce white faces. In this work we subvert the biases of models to limit search to desirable regions. Our previous work introduced this idea for constraint handling [6], here we expand the concept to problems with additional criteria.

3 Method

The SOLVE approach comprises three steps: Dataset Generation, Representation Learning, and Optimization.

3.1 Step 1: Dataset Generation

SOLVE decomposes the problem of optimization into stages, similar to [23]: the first step (Fig. 1 left) is the generation of a set of solutions that meet a criterion, without regard to their performance on the objective. When the only requirement is to satisfy a single criterion out of several, the search problem becomes much easier. Where it is feasible to calculate random values that satisfy the criterion using a dedicated algorithm, e.g., a constraint solver, this is typically the fastest method. However, it is sufficient to use a simple genetic algorithm for many criteria.

To produce the dataset of solutions we use the simple genetic algorithm defined in the DEAP framework [22]. We use real encoding, with a real-valued gene corresponding to each variable of the overall problem including variables that may be in the objective function and not in the extra criterion. Fitness is defined only by the criterion with a threshold used to determine acceptability for that criterion. The criterion fitness is zero if the value is under the threshold, otherwise it is set to a linear distance value representing the degree to which the criterion has been met, e.g., for $45 - x < 0$: if $x < 45$ then

fitness $f(x) = 45 - x$; otherwise $f(x) = 0$. If the best individual in the population achieves a fitness of 0 it is added to the dataset and the run terminates.

The genetic algorithm is run repeatedly until a dataset of d values has been found (typical execution times for C1 were no more than 10 ms per run, with a dataset of 5000 values taking less than 30 s on a MacBook Air 2020 M1, 16 GB memory). Each run used different initial populations, producing a distribution of values in the feasible region. These solutions provide a sampling of the valid region that serves as the basis for the learned representation. Should coverage be insufficient, or should a more efficient method be required, alternative algorithms which explicitly search for diversity such as Clustering [24], Clearing [25], Novelty Search [26], or MAP-Elites [27] could be employed for this step. Specialized constraint satisfaction algorithms could be used if the criterion takes the form of a constraint [28, 29].

3.2 Step 2: Representation Learning

Redesigning representations offers an alternative approach to optimization: a representation that has a bias towards the expression of useful or desirable solutions can enable simple optimizers to find good solutions, removing the requirement of needing expert tuning or development of specialized optimization algorithms.

In the second step of SOLVE (Fig. 1 middle), we use a simple VAE¹ with a standard loss function from [8]. We use one latent variable for every variable in the problem. We transform the input from the dataset to -1.0 to 1.0 (although the data-generation GA was limited to this range, its output data can focus on a smaller area of the search space and thus the dataset range may be smaller) and learn for E epochs. We refer the reader to the Supplemental Material² for full details.

3.3 Step 3: Optimization

In the third step (Fig. 1 right), we use the objective function for the first time and use a GA to search for optimal solutions that also satisfy the extra criterion in the learned latent space. Again, we use DEAP with real encoding, each gene corresponding to a latent variable with range -2.0 to 2.0 (typically sufficient to express the full learned range in the latent representation) with the results of all operators bound to this range. The same crossover and mutation operators are used as described in step 1.

Individuals are evaluated by decoding the latent values using the learned VAE model, mapping back to the range of the problem, and applying the fitness function. Fitness is a combination of the criterion, as formulated in Subsect. 3.1, and objective function. To ensure both are treated equally, we use tournament fitness, which awards individuals an average fitness score based on how many times it beats another individual for each objective and criterion over a series of smaller tournaments, similar to [3], chosen to encourage diversity akin to [30]. This approach avoids the need for summing and weighting separate criteria or using penalties for constraints [2, 31] and preliminary experiments using this method on benchmark problems with a standard GA resulted in significant

¹ <https://github.com/pytorch/examples/tree/master/vae>.

² <http://www.cs.ucl.ac.uk/staff/p.bentley/solvesupplemental.pdf>.

improvements to optimization with the settings used here. See the Supplemental Material for the tournament fitness algorithm. Parent selection is then performed using the same tournament selection (t size = 3) as described in step 1. A small population size evolving for few generations is sufficient.

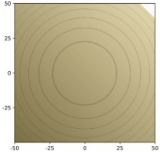
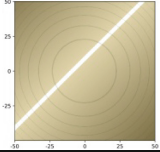
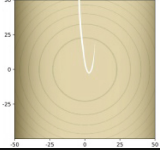
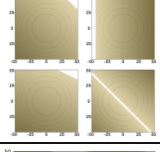
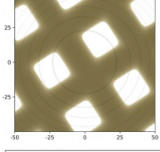
4 Experiments

For all experiments we use a simple objective function to minimize for D variables:

$$f(\bar{x}) = \sum_{i=0}^{D-1} x_i^2 \tag{3}$$

Table 1 provides the additional criteria used to modify the search space of Eq. (3). These represent several commonly observed constraint and optimization functions inspired by the analysis in [34]: the range limit (C1), correlated variables in the form

Table 1. Extra criteria designed to conflict with the objective optimal.

Criterion	Equation	Search space
C1: Range limit	$\sum_{i=0}^{D-1} (45 - x_i) \leq 0$	
C2: Dependency	$\forall i \in \{0, \dots, D - 2\}$ $8 \leq (x_{i+1} - x_i) \leq 10$	
C3: Nonlinear (Shifted, generalised Rosenbrock [32])	$\forall i \in \{0, \dots, D - 2\}$ $100((x_i - 3)^2 - x_{i+1} - 3)^2 + (x_i - 4) \leq 0$	
C4: Multiple dependencies and range limits over different subsets of problem variables	C4.1: $80 - x_0 - x_1 \leq 0$ C4.2: $x_2 + 45 \leq 0$ C4.3: $60 - x_1/2 - x_3 \leq 0$ C4.4: $-7 \leq (x_2 + x_3) \leq -5$	
C5: Discontinuous, $D=2$ (Shifted, rotated DeJong [33])	$1/[\sum_{j=1}^{25} ((x'_0 - A[1, j])^6 - (x'_1 - A[2, j])^6 + j)^{-1} + \epsilon] - 445 < 0$ where: $A \in R^{2 \times 25}$ and $\epsilon = 0.002$ $\begin{bmatrix} x'_0 \\ x'_1 \end{bmatrix} = \begin{bmatrix} \cos(d) & -\sin(d) \\ \sin(d) & \cos(d) \end{bmatrix} \begin{bmatrix} ax_0 \\ ax_1 \end{bmatrix} + b$ $d = -33, a = 0.39, b = [9, -3]^T$	



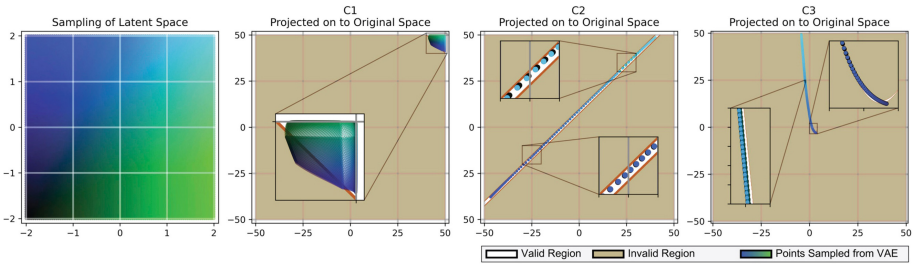


Fig. 2. Visualizing the Learned Landscapes. We can observe how the VAE shapes the space to be searched. Left: x_0 and x_1 values from -2 to 2 are color-coded to their locations in the learned space. *Others*: when these points in the learned space are projected back to the original space they are confined closely to the valid regions as defined by C1, C2 and C3.

of chained inequality (C2), the more complex Rosenbrock function [32] used in optimization competitions (C3) [35], multiple criteria (C4), and a discontinuous, multimodal function (C5). For all experiments, the range of the problem (objective and criteria) is -50 to 50 .

4.1 Single Criterion (C1, C2, C3)

We initially focus on problems with one extra criterion. We investigate C1, C2, and C3, each designed to create a search space that cannot be solved reliably using a standard GA. The criteria achieve this by conflicting with the true optimal of Eq. (3) forcing the optimizer to compromise to meet the objective and criterion equally. We apply each step of SOLVE: generating data, learning new representation, using a GA to find optimal solutions with this representation. First, we examine the VAE and its learned representation.

Visualizing Learned Latent Representations. To understand how the learned latent representation may be beneficial for the GA in SOLVE, we plot the distribution of values returned by the latent representation as we vary the latent variables from -2 to 2 for C1 and C2. Figure 2 shows how the search space is compressed and folded into the small valid region for C1. This gives the dual advantage of reducing the search space to focus on the valid region and improving evolvability through duplication, with different values for the latent variables mapping to the same valid region. For C2 the space is compressed into a straight line, showing that the VAE has learned the correlation between the two variables.

Comparing Learned Latent Representation with Standard Representation. We examine criteria C1 to C3, using SOLVE to generate valid, optimal solutions and compare results to a standalone GA. The number of parameters D for the objective and criterion were varied from 1 to 10 to examine the effects of scaling the problem. The simple GA within step 1 of SOLVE generated data for C1 and C3 (5000 points).

This GA was unable to generate data for C2 for higher dimensions in feasible time, so 5000 random datapoints were calculated directly from the criterion in this case. The algorithm is provided in Supplementary Material.

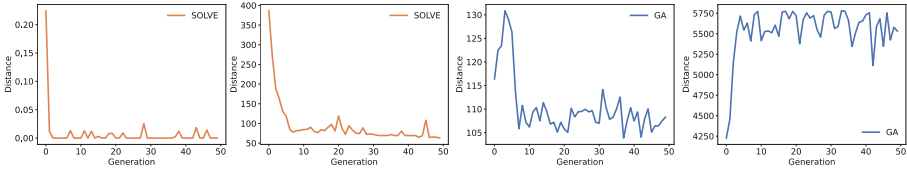


Fig. 3. Example run showing SOLVE criterion error (far left), SOLVE objective error over time (middle left), standalone GA criterion error (middle right) and standalone GA objective error over time (right) for 3 variable C1, and averaged over entire population. Note difference in y-axis scales for SOLVE and standalone GA results.

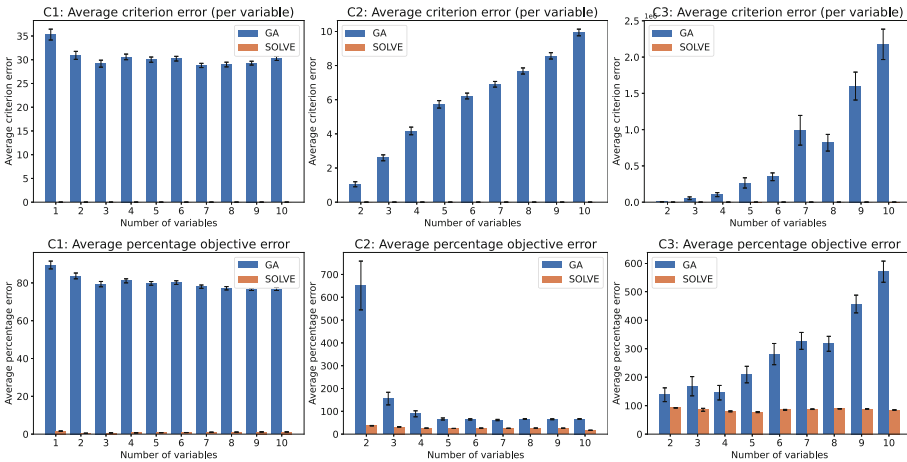


Fig. 4. Average error per criteria and average percentage error from best solution, over 100 runs, for different numbers of variables ($D = 1..10$). SOLVE always met C1 and C2; almost always C3. Error bars: mean \pm SE.

We compare SOLVE with a standalone GA evolving a direct representation of the same problem (each gene real coded and bound to between -1.0 to 1.0 , and using the same fitness calculation, selection, population size of 20 and 50 generations). Every run for both representations was repeated 100 times and average (mean) results reported. In order to enable comparison of achieving different optimal solutions, we calculate objective error as difference between $f(\bar{x})$ (Eq. 3) and optimal, divided by optimal $\times 100$, and criterion error is distance to criterion divided by number of variables.

Figure 3 shows a representative sample run comparing SOLVE evolution vs. standalone GA for C1 with $D = 3$ variables. While the GA struggles to evolve solutions that satisfy C1 and meet the objective, with evolution becoming stuck in a poor local optimal, SOLVE is consistently able to evolve solutions that meet C1 and are of high quality according to the objective. (Similar findings are presented in [6]).

The learned latent representation enables dramatic improvement (two orders of magnitude better) for the evolution of solutions meeting the objective and criteria. Figure 4 shows results for the three criteria with the number of variables increased from 1 to 10.

In C1 the standalone GA performs equally poorly for criteria and objective in all cases, while SOLVE achieves all criteria and a good objective for all numbers of variables. In C2, where variables are correlated, the standalone GA performs worse for larger numbers of variables, but objective values improve as the constrained problem reduces the number of possible good solutions. In contrast, SOLVE consistently nearly always meets all criteria and achieves solutions that reach the objective more closely for all numbers of variables. In the more difficult C3, the standalone GA performs worse for criteria and objectives as the number of variables increases. SOLVE meets the criteria better and consistently achieves good objective values.

4.2 Multiple Criteria (C4)

We next consider problems with multiple criteria (C4 in Table 1). This common form of optimization comprises several criteria and an objective. Each criterion applies only to a subset of the variables in the problem.

While SOLVE may be successfully applied to problems with a single extra criterion as shown for C1 to C3, for multiple criteria the creation of datasets that comprise examples of valid solutions for each criterion is incompatible with the VAE. This incompatibility is caused by the fact that by considering one criterion at a time to keep the problem easily solvable we can only generate valid values for the subsets of the variables belonging to that criterion. If we were to create a dataset for each criterion, we would generate valid values for variables belonging to the criterion in question, and random values for the free variables. For example, C4.1 applies only to x_0 and x_1 , leaving x_2 and x_3 free to take any values; conversely, C4.4 applies only to x_2 and x_3 , leaving x_0 and x_1 free. The result would be datasets comprising, in part, random values for all variables – effectively training the VAE that all values for all variables are valid and removing or even harming its ability to learn a useful representation. (While a GA could generate one dataset for all criteria simultaneously, this partially solves the difficult problem that we wish to transform by the VAE, defeating the objective of the work.)

The solution is to layer SOLVEs. We generate a latent representation for the first criterion, and then use the GA with the learned latent representation to create a dataset for the second criterion, which is used to learn a new latent representation that encapsulates both criteria, and so on, until all criteria have been learned in turn. The final latent representation is then used with all criteria and the objective to evolve optimal solutions that meet all criteria together, Fig. 5.

To assess the advantages of using a learned latent representation for multiple criteria, we used C4 with the same objective and compared a standalone GA against the original SOLVE and the LayeredSOLVE. We grow the difficulty incrementally, first trying C4.1 and C4.2 with a 2-layer SOLVE, then C4.1, C4.2, C4.3 with a 3-layer SOLVE, and finally C4.1, C4.2, C4.3, C4.4 with a 4-layer SOLVE (using 4 variables for all). To determine whether the order of the criteria alters the results, we also perform the same experiment, this time ordered: C4.1, C4.2, C4.4, C4.3. Figure 6 shows the results.

The results show that the original SOLVE offers little advantage compared to the standalone GA, both showing poor performance and large variance over the 100 runs. In contrast, the LayeredSOLVE can meet the criteria better, and enable solutions close to the optimum to be found, both with high consistency. The experiments also show that the

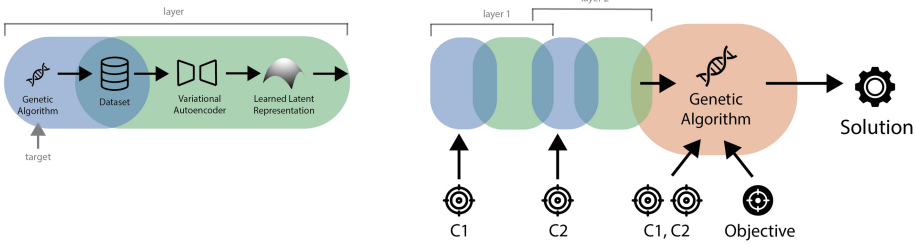


Fig. 5. LayeredSOLVE for two criteria over different subsets of problem variables. The number of stacks = number of criteria in the problem.

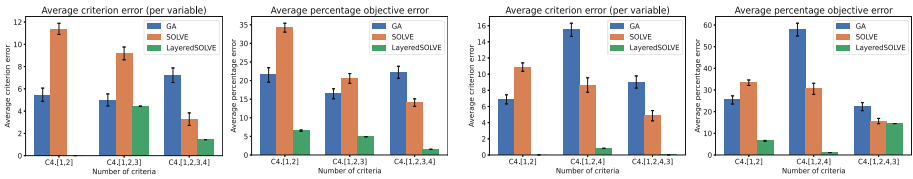


Fig. 6. Average error per criterion (average degree to which each criterion was broken) and average percentage error from best solution, over 100 runs, for a standalone GA, SOLVE and LayeredSOLVE, for 2, 3 and 4 criteria on 4 variables. Left two charts: C4.1, C4.2, C4.3, C4.4; Right two charts: C4.1, C4.2, C4.4, C4.3. Error bars: mean \pm SE.

order in which criteria are presented to LayeredSOLVE has an effect. When presenting the “stricter” C4.4 before C4.3, the criteria are better satisfied. (Although the objective error increases, analysis of the solutions shows the better objective error observed for cases where criteria are not met are caused by the standalone GA cheating – criteria are conflicted to have invalid but deceptively good objective scores.) The notion that criteria more difficult to satisfy should be presented first to an algorithm was exploited in [36] where constraints are ordered according to “the sum of the constraints violations of all solutions in the initial population from the least violated to the most violated”. Our results suggest that using the same strategy in the LayeredSOLVE will also improve its ability to satisfy all criteria.

4.3 Discontinuous Criteria (C5)

Finally, we focus on discontinuous problems. While SOLVE has been successfully applied to single (C1–C3) and multiple (C4) criteria problems, discontinuous criteria provide a challenge for the SOLVE model used in Sect. 4.1. We use an intuitive two-dimensional discontinuous constrained optimization problem (Table 1 C5). The problem is based on the well-known DeJong #5 function [33, 37], designed as a multimodal optimization benchmark, adjusted to create discontinuous valid regions and rotated to make it non-trivial for our optimizer, akin to [35].

This problem is difficult for a standalone GA, but its discontinuous nature is also incompatible with the use of a simple VAE, which attempts to fit a single Gaussian distribution to the set of valid points, i.e., it assumes a Gaussian distribution $N(0, I)$ as

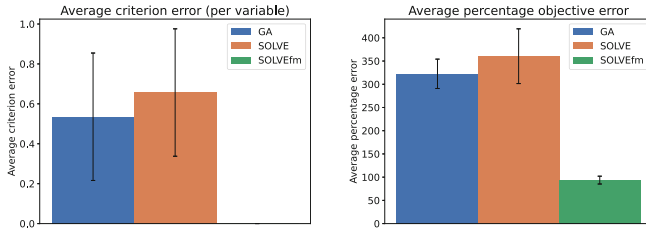


Fig. 7. Average error per criterion (average degree to which each criterion was broken) and average percentage error from best solution, over 100 runs, for original SOLVE, SOLVE with flow model (SOLVEfm), and a standalone GA for criterion C5. Average criterion error for SOLVEfm is zero. Error bars: mean \pm SE.

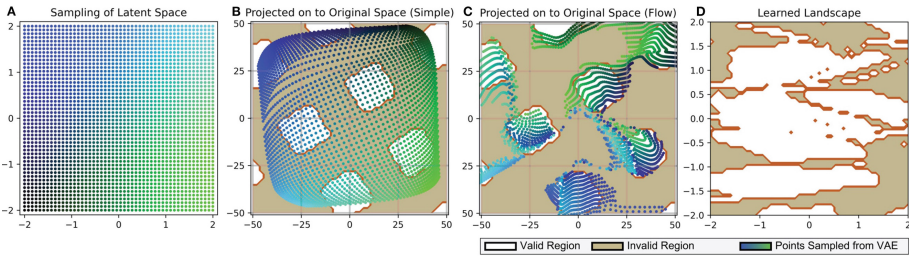


Fig. 8. Visualizing the learned latent representation. **A:** Sampling space from -2 to 2. **B:** Points projected onto space learned by a simple VAE. **C:** Points projected onto space learned by VAE with normalizing flow prior (SOLVEfm). **D:** Latent space colored by valid and invalid regions – this is the transformed landscape searched by SOLVEfm.

prior. The solution to improve expressivity in SOLVE is to increase the complexity of the VAE prior (i.e., “the structure of the sampling space”). While a Gaussian distribution allows for easy sampling and the KL divergence term can be calculated in closed form, for many datasets the representation is much more complicated than Gaussian distribution. For discontinuous criterion C5, we use a **normalizing flow** [38] as prior. A normalizing flow transforms a simple Gaussian distribution into a complex one by applying a sequence of invertible transformations. Given the chain of transformations, we change the latent variable to obtain a more complex target variable according to the change of variables theorem.

We compare the standalone GA with SOLVE and SOLVE using a normalizing flow VAE on C5 plus the usual objective function, reporting the average results of 100 runs. Figure 7 shows that the standalone GA and original SOLVE find valid solutions with large variance but fail to locate the optimum much of the time. SOLVE with normalizing flow model (denoted as SOLVEfm in Fig. 7) finds valid solutions with high consistency and finds better solutions for the objective with smaller variance.

We can understand how this result is achieved by examining Fig. 8. When using a simple VAE within SOLVE, the VAE fails to learn a good representation for the nine unbalanced and separated data modes for valid solutions (Fig. 8B). The generated data from the simple VAE tries to cover the center of the data, however, it misses some

data modes and covers many invalid regions, making this learned representation little different (and potentially inferior) to the original representation. Figure 8C shows the resulting expressed values for the flow-based model. Even though we observe some samples in the invalid regions, the flow-based model is able to cover most modes of the criterion, thus reducing the search space to the valid regions. Figure 8D illustrates how this representation “connects the disconnected” – transforming the discontinuous space into a more connected region, conducive to search.

The advantages of this method still need to be weighed against the generation of datasets, which even when the problem has been simplified by considering just one criteria at a time, requires computation time, see [6] for discussion. A comparison of the quality of SOLVE solutions should also be made with state-of-the-art optimizers.

5 Conclusions

Nature achieved evolution of evolvability in its genetic representations through a computationally expensive process that is infeasible for us to duplicate. Here we have demonstrated a viable alternative: SOLVE, which uses generative machine learning to learn better representations for search. Using this method, not only can we bias the representation so that it focusses mainly on desirable regions of the space according to extra criteria or constraints, but the nature of the Kullback-Leibler (KL) divergence used for regularization within the VAE provides a natural “smoothing” effect on the resulting latent space akin to the notion of evolvability, which can change a discontinuous space into a continuous space enabling highly effective search by the optimizer. We have demonstrated that with zero dimensionality reduction (i.e., using the same number of latent variables as problem variables), SOLVE can map different forms of hard-to-search spaces onto improved latent spaces. These spaces enable even a simple optimizer to achieve substantive performance increases in terms of quality of solution found and effort required to find that solution, as evidenced by the small population sizes and low number of generations required for the GA within SOLVE.

This work used a GA for data generation and optimization and a VAE for representation learning, but other equivalent approaches could be employed within SOLVE. The field of generative machine learning continues to advance at a great pace, so we anticipate that the integration of these newer techniques into optimization for the purposes of generating improved search spaces will be a fruitful area of research going forwards.

Source Code. The source code necessary to reproduce the experiments in this paper is available at: <https://github.com/writingpeter/SOLVE>.

References

1. Watanabe, K., et al.: A global overview of pleiotropy and genetic architecture in complex traits. *Nat. Genet.* **51**, 1339–1348 (2019)
2. Homaifar, A., Qi, C.X., Lai, S.H.: Constrained optimization via genetic algorithms. *SIMULATION* **62**, 242–253 (1994)
3. Deb, K.: An efficient constraint handling method for genetic algorithms. *Comput. Methods Appl. Mech. Eng.* **186**, 311–338 (2000)

4. Yu, T., Bentley, P.: Methods to evolve legal phenotypes. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) PPSN 1998. LNCS, vol. 1498, pp. 280–291. Springer, Heidelberg (1998). <https://doi.org/10.1007/BFb0056871>
5. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **6**, 182–197 (2002)
6. Bentley, P., Lim, S.L., Gaier, A., Tran, L.: COIL: Constrained optimization in learned latent space. Learning representations for valid solutions. In: ACM Genetic and Evolutionary Computation Conference Companion, p. 8 (2022)
7. Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. *Science* **313**, 504–507 (2006)
8. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. In: International Conference on Learning Representation, p. 14 (2014)
9. Rezende, D.J., Mohamed, S., Wierstra, D.: Stochastic backpropagation and approximate inference in deep generative models. In: International Conference on Machine Learning, pp. 1278–1286 (2014)
10. Volz, V., Schrum, J., Liu, J., Lucas, S.M., Smith, A., Risi, S.: Evolving Mario levels in the latent space of a deep convolutional generative adversarial network. In: ACM Genetic and Evolutionary Computation Conference, pp. 221–228 (2018)
11. Bontrager, P., Roy, A., Togelius, J., Memon, N., Ross, A.: DeepMasterPrints: generating masterprints for dictionary attacks via latent variable evolution. In: IEEE 9th International Conference on Biometrics Theory, Applications and Systems, pp. 1–9 (2018)
12. Fontaine, M.C., Nikolaidis, S.: Differentiable quality diversity. In: 35th Conference on Neural Information Processing Systems, pp. 10040–10052 (2021)
13. Liskowski, P., Krawiec, K., Toklu, N.E., Swan, J.: Program synthesis as latent continuous optimization: Evolutionary search in neural embeddings. In: ACM Genetic and Evolutionary Computation Conference, pp. 359–367 (2020)
14. Scott, E.O., De Jong, K.A.: Toward learning neural network encodings for continuous optimization problems. In: ACM Genetic and Evolutionary Computation Conference Companion, pp. 123–124 (2018)
15. Moreno, M.A., Banzhaf, W., Ofria, C.: Learning an evolvable genotype-phenotype mapping. In: ACM Genetic and Evolutionary Computation Conference, pp. 983–990 (2018)
16. Pugh, J.K., Soros, L.B., Stanley, K.O.: Quality diversity: a new frontier for evolutionary computation. *Front. Robot. AI* **3**, 40 (2016)
17. Cully, A., Demiris, Y.: Quality and diversity optimization: a unifying modular framework. *IEEE Trans. Evol. Comput.* **22**, 245–259 (2017)
18. Gaier, A., Asteroth, A., Mouret, J.-B.: Discovering representations for black-box optimization. In: ACM Genetic and Evolutionary Computation Conference, pp. 103–111 (2020)
19. Rakicevic, N., Cully, A., Kormushev, P.: Policy manifold search: exploring the manifold hypothesis for diversity-based neuroevolution. In: ACM Genetic and Evolutionary Computation Conference, pp. 901–909 (2021)
20. Cui, M., Li, L., Zhou, M.: An Autoencoder-embedded Evolutionary Optimization Framework for High-dimensional Problems. In: IEEE International Conference on Systems, Man, and Cybernetics, vol. 2020-October, pp. 1046–1051 (2020)
21. Cui, M., Li, L., Zhou, M., Abusorrah, A.: Surrogate-assisted autoencoder-embedded evolutionary optimization algorithm to solve high-dimensional expensive problems. *IEEE Trans. Evol. Comput.* (2021). <https://doi.org/10.1109/TEVC.2021.3113923>
22. Hagg, A., Berns, S., Asteroth, A., Colton, S., Bäck, T.: Expressivity of parameterized and data-driven representations in quality diversity search. In: ACM Genetic and Evolutionary Computation Conference, pp. 678–686 (2021)
23. Venkatraman, S., Yen, G.G.: A generic framework for constrained optimization using genetic algorithms. *IEEE Trans. Evol. Comput.* **9**, 424–435 (2005)

24. Yin, X., Gernay, N.: A fast genetic algorithm with sharing scheme using cluster analysis methods in multimodal function optimization. In: *Artificial Neural Nets and Genetic Algorithms*, pp. 450–457 (1993)
25. Pétrowski, A.: A clearing procedure as a niching method for genetic algorithms. In: *IEEE International Conference on Evolutionary Computation*, pp. 798–803 (1996)
26. Lehman, J., Stanley, K.O.: Abandoning objectives: Evolution through the search for novelty alone. *Evol. Comput.* **19**, 189–223 (2011)
27. Mouret, J.-B., Clune, J.: Illuminating search spaces by mapping elites. arXiv preprint [arXiv:1504.04909](https://arxiv.org/abs/1504.04909) (2015)
28. Coello, C.A.C.: Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. *Comput. Methods Appl. Mech. Eng.* **191**, 1245–1287 (2002)
29. Tsang, E.: *Foundations of constraint satisfaction: the classic text*. BoD–Books on Demand (2014)
30. Forrest, S., Hightower, R., Perelson, A.: The Baldwin effect in the immune system: learning by somatic hypermutation. *Individual Plasticity in Evolving Populations: Models and Algorithms* (1996)
31. Yeniay, Ö.: Penalty function methods for constrained optimization with genetic algorithms. *Math. Comput. Appl.* **10**, 45–56 (2005)
32. Biscani, F., Izzo, D.: A parallel global multiobjective framework for optimization: pagmo. *J. Open Source Softw.* **5**, 2338 (2020)
33. De Jong, K.A.: *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. University of Michigan. Ph.D. thesis (1975)
34. Hellwig, M., Beyer, H.-G.: Benchmarking evolutionary algorithms for single objective real-valued constrained optimization—a critical review. *Swarm Evol. Comput.* **44**, 927–944 (2019)
35. Suganthan, P.N., et al.: Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. *KanGAL Report 2005005*, 51 (2005)
36. Sallam, K.M., Elsayed, S.M., Chakraborty, R.K., Ryan, M.J.: Improved multi-operator differential evolution algorithm for solving unconstrained problems. In: *2020 IEEE Congress on Evolutionary Computation*, pp. 1–8 (2020)
37. Molga, M., Smutnicki, C.: Test functions for optimization needs. *Test Functions Optim. Needs* **101**, 43 (2005)
38. Rezende, D., Mohamed, S.: Variational inference with normalizing flows. In: *International Conference on Machine Learning*, pp. 1530–1538 (2015)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

