OptMap: Using Dense Maps for Visualizing Multidimensional Optimization Problems

Mateus Espadoto[1,3][a], Francisco C. M. Rodrigues[1,3][b],
Nina S. T. Hirata[1][c] and Alexandru C. Telea[2][d]

[1]*Institute of Mathematics and Statistics, University of São Paulo, Brazil*
[2]*Department of Information and Computing Sciences, University of Utrecht*
[3]*Johann Bernoulli Institute, University of Groningen*
{*mespadot, caiomr, nina*}*@ime.usp.br, a.c.telea@uu.nl*

Keywords: Optimization, Dimensionality Reduction, Dense Maps.

Abstract: Operations Research is a very important discipline in many industries, and although there were many developments since its inception, to our knowledge there are no visualization tools focused on helping users understand the decision variables' domain space and its constraints for problems with more than two input dimensions. In this paper, we propose OptMap, a technique that enables the visual exploration of optimization problems using a two-dimensional dense map, regardless of the number of variables and constraints in the problem and for any kind of single-valued objective function. We show the technique in action for several optimization problems of different types, such as linear, nonlinear and integer, constrained and unconstrained problems.

1 INTRODUCTION

Operations Research (OR), also called Management Science, plays a crucial role in many industries, from logistics to finance. Although its origins as a discipline date from the 1950s, with the development of the Simplex algorithm for Linear Programming (Dantzig, 1990; Kantorovich, 1960), it is a field in constant development since then. The OR practitioner has many tools at their disposal which improve their productivity, such as algebraic modeling languages like GAMS (Brooke et al., 1998), AMPL (Fourer et al., 2003) and more recently, JuMP (Dunning et al., 2017), which enable the use of notation very close to the mathematical definition of optimization problems. Yet, to our knowledge, there are no well-established *visualization* tools that help understand multivariate objective functions with respect to the decision variables and constraints (if any) of the problem. Such tools are important as a complement to more formal tools for getting an overall understanding of how an objective function behaves subject to its many parameters.

We propose a technique called OptMap, which is an image-based visualization tool that enables the OR practitioner to literally see the decision variables and constraint spaces using a two-dimensional dense map, regardless of the number of variables and constraints in the problem. We show that OptMap can be used in several ways, such as a debugging aid to help diagnose errors in the definition of constraints; as a tool to provide insight of the optimizer's inner workings, by plotting the path taken from a starting point to a solution; and as a general tool to visually explore the high-dimensional space of the decision variables in terms of objective function value and constraint feasibility.

OptMap aims to cover the following aspects, which, to our knowledge, are not achieved by existing visualization techniques in the context of optimization:

Quality (C1): We provide high-quality visualizations, that encode information at every available screen pixel, by using a combination of dense maps, direct, and inverse projection techniques;

Genericity (C2): We can handle many kinds of optimization problems for single-valued objective functions. The only requirements we impose are that

[a] https://orcid.org/0000-0002-1922-4309
[b] https://orcid.org/0000-0002-0540-8510
[c] https://orcid.org/0000-0001-9722-5764
[d] https://orcid.org/0000-0003-0750-0502

the user provides implementations of the objective function, constraints (if any), and the range for each variable;

Simplicity (C3): Our technique is based on existing projection techniques which have a straightforward implementation, allowing easy replication and deployment;

Ease of use (C4): Our technique has few hyperparameters, all with given presets. In most cases, users do not have to adjust those to obtain good results;

Scalability (C5): By using a fast projection technique and caching results when possible, our method is fast enough to allow its use during the rapid development-test cycle of optimization models.

We structure our paper as follows. Section 2 presents the notations used and discusses related work on visualization for multivariate functions and optimization problems, Section 3 details our method. Section 4 presents the results that support our contributions outlined above. Section 5 discusses our proposal. Section 6 concludes the paper.

2 BACKGROUND

Related work concerns optimization techniques (Sec. 2.1 and 2.2), dimensionality reduction (Sec. 2.3), and visualization (Sec. 2.4).

2.1 Optimization: Preliminaries

Optimization problems come in many forms with respect to the kind of function to be optimized, the type of decision variables, and the existence of constraints. Functions are typically grouped into linear, convex and non-convex. *Linear* functions are of the form $f(\mathbf{x}) = \mathbf{ax} + b$, which defines a hyperplane; *convex* functions can have many forms, but can be defined as those where the set of points above their graph forms a convex set; *non-convex* functions are neither linear nor convex (linear functions are also convex). Decision variables can be continuous or discrete: Problems with only discrete variables are called Integer Programs (IP) (Guenin et al., 2014), whereas problems with a combination of discrete and continuous variables are called Mixed Integer Programs (MIP). Lastly, problems can be constrained or unconstrained. Constraints can be characterized just as functions (linear, convex, non-convex). Additionally, we have box constraints, which are simple restrictions on the variables' domains. Problems with continuous variables, linear objective functions, and linear constraints are called Linear Programs (LP). Other problems are solved by Nonlinear Programming (NLP) techniques.

In any case, real-world optimization problems typically have many variables and constraints. Without visual aids, the user typically has to rely on numerical analysis to understand if the problem is modeled correctly and if the results make sense. In our opinion, having a visualization tool greatly expands the possibilities of model analysis and debugging, giving the user a quick way to check, for example, if constraints are correctly defined, *i.e.*, not under- or over-constraining by mistake, or, in the case of NLP problems, to check how stable are the optima found, *i.e.*, how close they are to peaks or troughs in the data.

2.2 Optimization: Technicalities

We next define a few notations for optimization problems. Let $f : \mathbb{R}^n \to \mathbb{R}$ be some function to be minimized. Let $\mathbf{x} = (x^1, \ldots, x^n)$, $x^i, 1 \leq i \leq n$, be an n-dimensional vector of n decision variables x^i. Decision variables can be any combination of discrete ($\mathbb{Z}$) and continuous ($\mathbb{R}$). Let O be the optimization problem described as

$$\begin{array}{ll} \text{minimize} & f(\mathbf{x}) \\ \text{subject to} & \mathbf{x} \in S \end{array} \tag{1}$$

where S is the *feasible set* of all points that can be considered as valid for the optimization problem. For unconstrained problems, S is $\mathbb{R}^n$. For constrained problems there is a set of K constraint functions $c_k(\mathbf{x}) \in \{0,1\}, k \in 1,...,K$, where 0 means that the point $\mathbf{x}$ is infeasible with respect to the constraint c_k. That is, for constrained problems, the feasible set is defined as $S = \{\mathbf{x} : \prod c_k(\mathbf{x}) = 1, k \in 1,...,K\}$.

Solvers are algorithms that find one of several (approximate) solutions to a problem O. To do this efficiently, solvers use the characteristics of the problem, such as the type of decision variables, objective function and constraints, and employ adequate heuristics to avoid exploring all possible $\mathbf{x} \in S$, which would be impractical in most cases. Probably the most popular solver algorithm is the Simplex (Dantzig, 1990; Kantorovich, 1960), used for linear problems and implemented by software such as Clp (Forrest et al., 2020a), Cbc (Forrest et al., 2020b) and GLPK (Makhorin, 2008). For non-linear optimization problems there are other algorithms such as Gradient Descent, Nelder-Mead (Nelder and Mead, 1965) and L-BFGS (Liu and Nocedal, 1989), to name a few. Many solvers work iteratively, *i.e.*, start from a given point $\mathbf{x}_0$ and evolve this point until sufficiently close to the solution of O.

A *solution* is an n-dimensional point found by the solver which meets the criteria of being feasible ($\mathbf{x} \in S$) and *optimal*. The definition of optimality depends on the type of problem and solver used: For linear functions with linear constraints, solvers are guaranteed to find a *global optimum* solution, which means that no other n-dimensional point provides a lower value for the objective function f, given those constraints. For non-linear functions, solvers may return different *local optima*, depending on the starting point $\mathbf{x}_0$ used and the shape of the objective function.

Lastly, a solver may provide the user with a *trace*, or *path to solution*, which is the set of all n-dimensional points where it evaluated the objective function, starting from $\mathbf{x}_0$ and ending with the solution, if one was found, else ending with the last point evaluated by the solver.

2.3 Dimensionality Reduction

Dimensionality reduction (DR) is an area of research concerned with representation of high-dimensional data by a low number of dimensions, enabling different tasks to be performed on the data, such as visual exploration (Espadoto et al., 2019a). Probably the best known DR method is Principal Component Analysis (Jolliffe, 1986) (PCA), which has been used in several areas for many decades. It is a very simple algorithm with theoretical grounding in linear algebra. PCA is commonly used as preprocessing step for automatic DR on high-dimensional datasets prior to selecting a more specific DR method for visual exploration (Nonato and Aupetit, 2018).

There are many families of DR methods, such as Manifold Learners, Spring Embedders and Stochastic Neighborhood Embedding (SNE) techniques, among others. Manifold Learners such as MDS (Torgerson, 1958), Isomap (Tenenbaum et al., 2000), LLE (Roweis and Saul, 2000) and more recently UMAP (McInnes and Healy, 2018) try to reproduce in 2D the high-dimensional manifold on which data is embedded, to capture nonlinear structure in the data. Spring Embedders, also called force-directed techniques, such as LAMP (Joia et al., 2011) and LSP (Paulovich et al., 2008), are popular in the visualization literature and have a long history, with uses other than dimensionality reduction, such as graph drawing. The SNE family of methods appeared in the 2000's, and has t-SNE (Maaten and Hinton, 2008) as its most popular member. SNE-class methods produce visualizations with good cluster separation. For extensive reviews of DR methods, and their quality features we refer to (Nonato and Aupetit, 2018; Espadoto et al., 2019a).

We next describe the notation for DR used in the paper. Let $D = \{\mathbf{x}_i\}$, $1 \leq i \leq N$ be a dataset of N points $\mathbf{x}$ with n dimensions each. A dimensionality reduction, or projection, technique is a function

$$P : \mathbb{R}^n \rightarrow \mathbb{R}^q$$

where $q \ll n$, and typically $q = 2$. The projection $P(\mathbf{x})$ of a sample $\mathbf{x} \in D$ is a point $\mathbf{p} \in \mathbb{R}^q$. Projecting a set D yields thus a qD scatter plot, denoted next as $P(D)$. The inverse of P, denoted $P^{-1}(\mathbf{p})$, maps a point in $\mathbb{R}^q$ to a high-dimensional point $\mathbf{x} \in \mathbb{R}^n$, aiming to satisfy that $P^{-1}(P(\mathbf{x})) = \mathbf{x}$. Methods computing inverse projections include iLAMP (Amorim et al., 2012) and NNInv (Espadoto et al., 2019b).

2.4 Visualization

Visualization of high-dimensional data is an active topic for several decades, with many types of methods being proposed (Buja et al., 1996; Liu et al., 2015) and analyzed via several quality metrics (Bertini et al., 2011). Our scope is narrower – we are interested in visualizing multidimensional *functions*, and more particularly, *optimization* processes for such functions.

The visualization of 2D functions $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ is usually done by means of 3D height plots, contour plots, or color (heatmap) plots. For functions $f : \mathbb{R}^n \rightarrow \mathbb{R}$ with more than two variables ($n > 2$), there are far fewer options, with Hyperslice (van Wijk and van Liere, 1993) being a notable one. Hyperslice presents a multidimensional function as a matrix of orthogonal two-dimensional slices, each showing the restriction of f to one of the 2D subspaces in $\mathbb{R}^n$, using the 2D function plotting outlined earlier (contour plots, color plots, 3D height plots).

Visualizing constrained optimization problems is similar to the above, since not only the function has to be visualized but constraint feasibility as well. Most techniques used for this are based on overlaying contour plots with constraint information, with one case where image-based techniques are used (Wicklin, 2018). Still, such techniques cannot work with more than two dimensions ($n > 2$).

The authors of iLAMP (Amorim et al., 2012) used direct and inverse projection techniques applied to non-linear optimization problems, to help users interactively identify good starting points for optimization problems. However, iLAMP is computationally expensive, and has quite a number of free parameters the user needs to set. The NNInv method (Espadoto et al., 2019b) accelerates inverse projections by over two orders of magnitude as compared to iLAMP by deep learning the inverse projection function P^{-1}. The same deep learning idea was also used to accelerate the direct projection P by Neural Network Projections (NNP) (Espadoto et al., 2020). Recently, NNInv was used by an image-based (dense map) technique to visualize the decision boundaries for Machine Learning classifiers (Rodrigues et al., 2019), for problems with arbitrary dimension. Their method can be conceptually seen as the visualization of a function $f : \mathbb{R}^n \to C$, where f is a classifier for nD data and C is a class (label) set. We share the idea of using a dense pixel map to visualize high-dimensional functions with this work. However, we treat *real-valued* functions f rather than classifiers; and our aim is understanding optimization problems rather than understanding the output of a classifier, so we treat a different problem and use-case.

3 METHOD

We next describe the OptMap technique. Figure 1 shows a high-level diagram of OptMap, with each step described in detail next.

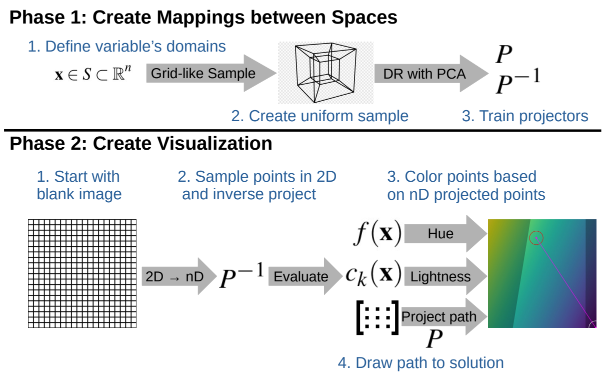
Phase 1: Create Mappings between Spaces
1. Define variable's domains
$\mathbf{x} \in S \subset \mathbb{R}^n$
Grid-like Sample
DR with PCA
P
P^{-1}
2. Create uniform sample
3. Train projectors
Phase 2: Create Visualization
1. Start with blank image
2. Sample points in 2D and inverse project
3. Color points based on nD projected points
2D → nD
P^{-1}
Evaluate
$f(\mathbf{x})$
Hue
$c_k(\mathbf{x})$
Lightness
Project path
P
4. Draw path to solution

Figure 1: OptMap Pipeline.

1. Define variable ranges: the user specifies the domain of each variable x_i for $f(x_1, \ldots, x_n)$. When the range is the entire real axis $\mathbb{R}$, we suggest selecting a reasonable finite range to avoid having a too coarse sampling for that variable;

2. Sample data: We uniformly sample the ranges defined above for each variable, yielding a regular sample grid $G^n \subset \mathbb{R}^n$. We constrain the maximum number of sample points N_{max} to avoid combinatorial explosion. In this paper, we used $N_{max} = 5$M for all experiments. We evaluate f on G^n and call the resulting dataset D.

3. Create mappings: we use PCA (Jolliffe, 1986) trained on D to create the mappings P and P^{-1} from $\mathbb{R}^n$ to $\mathbb{R}^q$, and from $\mathbb{R}^q$ to $\mathbb{R}^n$ respectively.

4. Create a 2D grid: We create an uniform grid $G^2 \subset \mathbb{R}^2$ similar to G^n. Simply put, G^2 is a pixel image of some fixed resolution (set to 800^2 for the experiments in this paper). Next, we use the trained P^{-1} to map each grid point (pixel) $\mathbf{p} \in G^2$ to a high-dimensional point $\mathbf{x} \in \mathbb{R}^n$. Finally, we evaluate the objective function $f(\mathbf{x})$ and optional constraints.

5. Color pixels: We color all pixels $\mathbf{p} \in G^2$ by the values of $f(P^{-1}(\mathbf{p}))$, using a continuous color map, set in this paper to the Viridis color map (Hunter, 2007). Additionally, we set the luminance of $\mathbf{p}$ to reflect $f(P^{-1}(\mathbf{p}))$'s membership of the constraint-set S, thereby indicating constraint feasibility. Note that, strictly speaking, this colormap also has a luminance component. Hence, luminance is actually encoding both f and the constraints. If desired, one can easily select to use other – more (perceptually) isoluminant colormaps. We leave the question of what the optimal colormap is open as part of future work.

6. Draw path to solution: if the solver provides the trace to a solution, as defined in Section 2.1, we can draw it in the 2D grid by projecting them using P.

4 RESULTS

We next present several experiments that show how our OptMap technique performs in different scenarios. First, we use OptMap to visualize high-dimensional functions that have a *known* shape (Sec. 4.1). Since we know the ground truth, we can check how OptMap performs. Next, we test our method on several unconstrained and constrained optimization problems (Secs. 4.2 and 4.3 respectively) and show the added value OptMap provides in these actual use cases.

4.1 Ground-Truth Functions

We use the six functions f listed in Table 1 to test OptMap. The corresponding dense maps, computed as explained in Sec. 3, are shown in Fig. 2. In all cases, the domain used for all variables was $x_i \in [-5, 5]$. All these functions have a predictable shape and also generalize to many dimensions. We created dense maps using increasing numbers of dimensions $n \in \{2, 3, 5, 7, 10, 20\}$. The dense map for $n = 2$ was

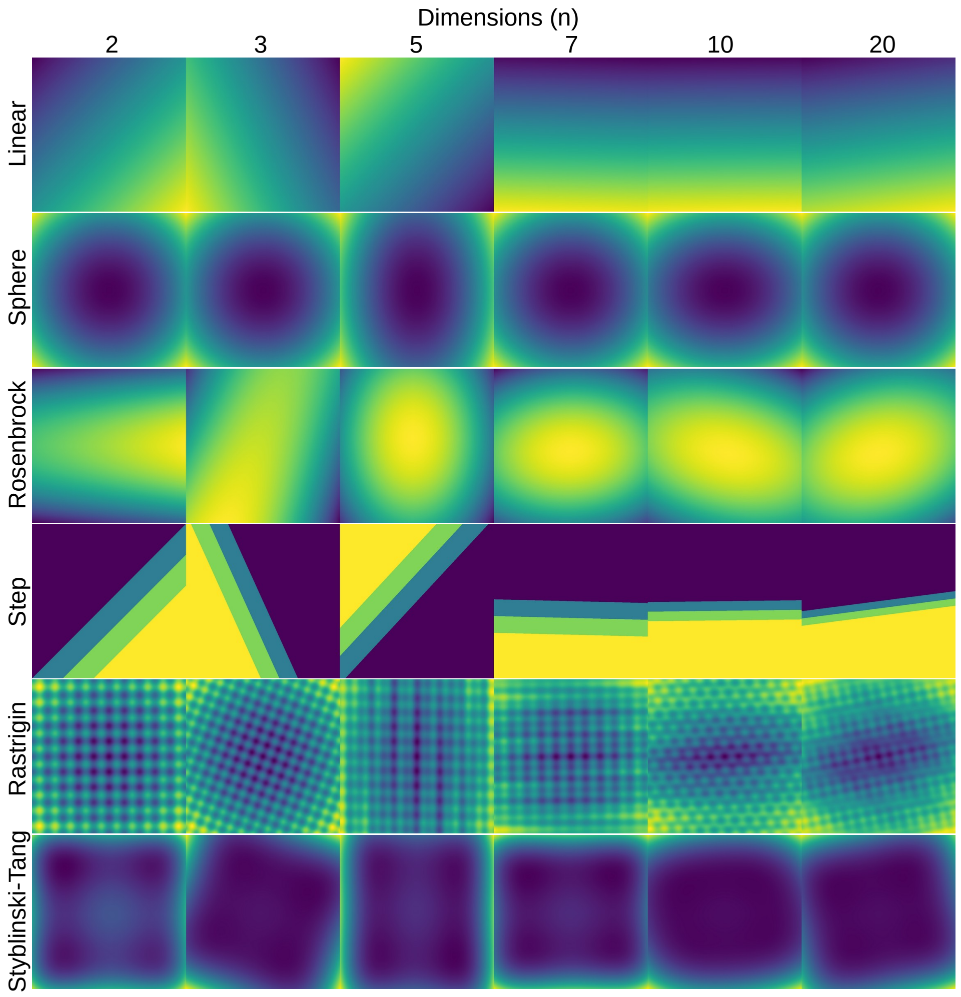

Figure 2: Dense maps for functions with known shape as defined in Table 1, with increasing dimensionality $n > 2$. Compare these with the ground-truth maps for $n = 2$.

created for reference only, without using OptMap. Indeed, for $n = 2$, we can directly visualize f, *e.g.*, by color coding, similar to (van Wijk and van Liere, 1993). Showing these maps for $n = 2$ is however very useful. Indeed, (1) for $n = 2$, we can show f *directly*, without any approximation implied by OptMap; and (2) given the functions' expressions (Tab. 1), we know that they behave similarly regardless of n. Hence, if for $n > 2$ OptMap produces images similar to the ground truth ones for $n = 2$, we know that OptMap works well. And indeed, Fig. 2 shows us exactly this – the OptMap images for $n > 2$ are very similar to the ground-truth ones for $n = 2$. The differences imply some distortion and rotations, which, we argue, are expected and reasonably small, given the inherent information loss when mapping a nD phenomenon to 2D.

4.2 Unconstrained Problems

We next use OptMap to show how different solvers perform when solving different unconstrained problems (that is, variants of Equation 1). For this, we select a subset of the functions defined in Table 1, namely Styblinski-Tang, Rastrigin and Sphere functions, with varying dimensionality n. We use the solvers listed in Table 2, grouped by solver type, namely whether it is gradient-free or if it requires a gradient or a Hessian. In Figure 3 we use OptMap to show the *trace* provided by each solver, *i.e.*, all the points evaluated by the solver to get to the solution. We see that for a simple function with a global optimum (Sphere) most solvers find an optimal solution, except for the gradient-free methods, which seem to struggle with the high-dimensionality of the problem ($n = 20$ dimensions). For the Styblinski-Tang func-

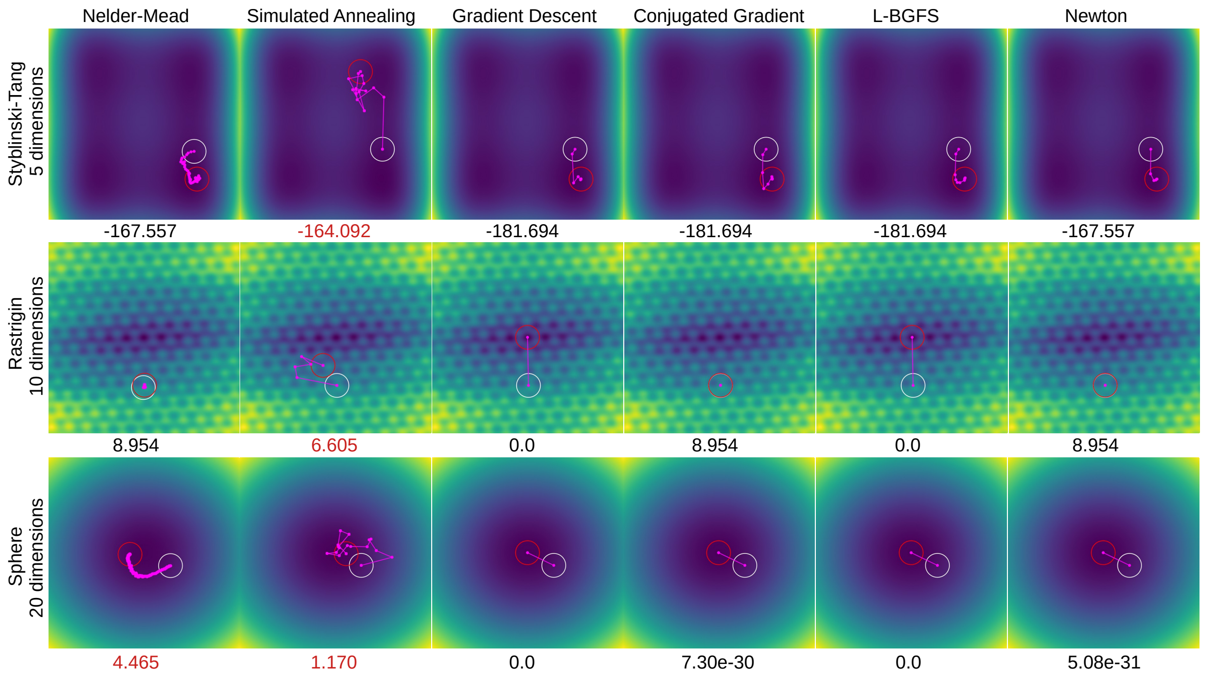

Figure 3: Dense maps created with OptMap for unconstrained problems using the solvers defined in Table 2 and some of the functions defined in Table 1. White circles indicate starting points (random vectors in 5, 10 and 20 dimensions respectively). Red circles indicate optimal points found by the solver. The magenta lines and points show each point evaluated by the solver to get to the solution. The numbers below each image indicate the value of the objective function at the solution; red values indicate that the solver failed to find an optimal solution (converge). In those cases, we list the value the solver stopped at before aborting.

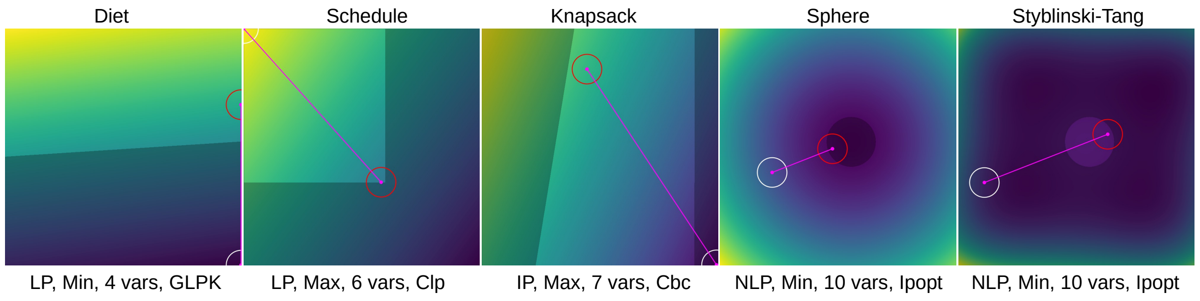

Figure 4: Dense maps created with OptMap for the constrained problems defined in Table 3. White circles indicate starting points (zero vector). Red circles indicate optimal points found by the solver. Magenta lines show the path from the starting point to the solution, and darker areas indicate unfeasible regions. The texts below each image indicate type of problem, direction (minimization or maximization), number of variables, and solver used.

tion, we see different but close optima were found by most solvers. We also see that both gradient-free methods evaluated many more points than the other methods, but that Nelder-Mead kept moving in the right direction. For the same problem, Simulated Annealing had problems converging to an optimal solution and eventually gave up. For the Rastrigin function, which has many optima, we see that only Gradient Descent and L-BFGS managed to find the solution in a straightforward way, while the other methods converged to the wrong solution or did not converge.

4.3 Constrained Problems

We next show how our OptMap performs when dealing with constrained optimization problems – that is, finding the minimum of some n-dimensional function f whose variables are constrained as described in Sec. 2.1. Table 3 shows the definition of constrained problems (objective functions and constraints) we used. The first three problems used are very common in the optimization literature (Guenin et al., 2014). The last two problems use the same Sphere and

Table 1: Definition of *n*-dimensional selected functions for ground-truth testing.

Function Name	Definition
Linear	$f(\mathbf{x}) = \sum_{i=1}^{n} x_i$
Sphere	$f(\mathbf{x}) = \sum_{i=1}^{n} x_i^2$
Rosenbrock (Rosenbrock, 1960)	$f(\mathbf{x}) = \sum_{i=1}^{n-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + \left(1 - x_i\right)^2\right]$
Step	$f(\mathbf{x}) = \begin{cases} 0 & \sum_{i=1}^{n} x_i < 0 \\ 2 & \sum_{i=1}^{n} x_i < 2 \\ 4 & \sum_{i=1}^{n} x_i < 4 \\ 5 & otherwise \end{cases}$
Rastrigin (Rastrigin, 1974)	$f(\mathbf{x}) = An + \sum_{i=1}^{n} \left[x_i^2 - A\cos(2\pi x_i)\right]$ where: $A = 10$
Styblinski-Tang (Styblinski and Tang, 1990)	$f(\mathbf{x}) = \frac{\sum_{i=1}^{n} x_i^4 - 16x_i^2 + 5x_i}{2}$

Table 2: Solvers used for unconstrained problems.

Solver Type	Solver
Gradient-free	Nelder-Mead (Nelder and Mead, 1965) Simulated Annealing
Gradient required	Gradient Descent Conjugated Gradient (Hager and Zhang, 2006) L-BFGS (Liu and Nocedal, 1989)
Hessian required	Newton

Styblinski-Tang functions defined earlier, but with nonlinear constraints added to them. Figure 4 shows how OptMap visualizes the problem space and solution for each problem. Unfortunately, the solvers used in this experiment, namely Clp (Forrest et al., 2020a), Cbc (Forrest et al., 2020b), GLPK (Makhorin, 2008) and Ipopt (Wächter and Biegler, 2006) do not provide trace information to be drawn through the algebraic modeling language we used, JuMP (Dunning et al., 2017), so we only draw the straight-line path from the (randomly chosen) starting point to solution.

In Figure 4, we can see for all problems the relationship between the objective function and the constraints of the problem, which provides insight on how close to boundary conditions the solutions are. For example, in the problems Schedule, Sphere and Styblinski-Tang, we see that the solution found is at the boundary of one or more constraints. This is not the case for the Diet and Knapsack problem, which indicates that some tuning to the solver's settings may be required to obtain better results, or even some adjustments to the problem definition may be done, such as the relaxation of some constraints.

Table 3: Definition of constrained optimization problems.

Name	Definition
Diet	minimize $0.14x_1 + 0.4x_2 + 0.3x_3 + 0.75x_4$ subject to $23x_1 + 171x_2 + 65x_3 + x_4 \geq 2000.0,$ $0.1x_1 + 0.2x_2 + 9.3x_4 \geq 30.0,$ $0.6x_1 + 3.7x_2 + 2.2x_3 + 7x_4 \geq 200.0,$ $6x_1 + 30x_2 + 13x_3 + 5x_4 \geq 250.0,$ $x_1, x_2, x_3, x_4 \geq 0.0$
Schedule	maximize $300x_1 + 260x_2 + 220x_3 + 180x_4 - 8y_1 - 6y_2$ subject to $11x_1 + 7x_2 + 6x_3 + 5x_4 \leq 700.0,$ $4x_1 + 6x_2 + 5x_3 + 4x_4 \leq 500.0,$ $8x_1 + 5x_2 + 5x_3 + 6x_4 - y_1 \leq 0.0,$ $7x_1 + 8x_2 + 7x_3 + 4x_4 - y_2 \leq 0.0,$ $y_1 \leq 600.0,$ $y_2 \leq 650.0,$ $x_1, x_2, x_3, x_4, y_1, y_2 \geq 0.0$
Knapsack	maximize $60x_1 + 70x_2 + 40x_3 + 70x_4 + 20x_5 + 90x_6$ subject to $x_1 + x_2 - 4y \geq 0.0,$ $x_5 + x_6 + 4y \geq 4.0,$ $30x_1 + 20x_2 + 30x_3 + 90x_4 + 30x_5 + 70x_6 \leq 2000.0,$ $x_3 - 10x_4 \leq 0.0,$ $x_1, x_2, x_3, x_4, x_5, x_6, y \geq 0.0,$ $x_1, x_2, x_3, x_4, x_5, x_6 \leq 10.0,$ $y \leq 1.0,$ $x_1, x_2, x_3, x_4, x_5, x_6, y \in \mathbb{Z}$
Sphere	minimize $\sum_{i=1}^{10} x_i^2$ subject to $\sum_{i=1}^{10} x_i^2 \geq 5.0$
Styblinski-Tang	minimize $\frac{\sum_{i=1}^{10} x_i^4 - 16x_i^2 + 5x_i}{2}$ subject to $\sum_{i=1}^{10} x_i^2 \geq 5.0$

4.4 Performance

OptMap's computation time can be divided in two phases (Fig. 1): In phase 1, most of the time is spent while running PCA for the sampled points in the grid G^n to define the mapping between the *n*D and 2D spaces. This is a task that has to be done only once for a given function f and can be reused afterwards when one changes the solver. In phase 2, most of the time is spent evaluating the objective function f and its constraints. To gauge OptMap's performance, we ran the experiments discussed in the previous sections on a 4-core Intel Xeon E3-1240 v6 at 3.7 GHz with 64 GB RAM. Since the evaluation of functions is usually very fast and the pixel grid G^2 is of limited size (800^2 in our experiments), phase 2 takes only a few seconds to run on our platform. Table 4 shows the time it takes to run PCA in phase 1 for $N_{max} = 5$M points, where we see that time increases very quickly with dimensionality. However, since phase 1 is required to be run only once, and since it takes at most a few minutes to run even with a high number of dimensions n (see Tab. 4), we argue that this is not a crucial limitation of OptMap.

Table 4: Time to project $N_{max} = 5M$ points with different dimensionalities n using PCA.

Dimensions n	**Time (sec)**
3	1.19
5	0.85
7	1.45
10	2.38
20	6.62
50	32.74
100	108.71

4.5 Implementation details

We implemented OptMap in Julia (Bezanson et al., 2017). Table 5 lists all open-source software libraries used to build OptMap. The optimization examples in Sec. 4 were implemented using Optim (Mogensen and Riseth, 2018) for the unconstrained problems, and JuMP (Dunning et al., 2017) for the constrained problems, using the solvers Clp (Forrest et al., 2020a), Cbc (Forrest et al., 2020b), GLPK (Makhorin, 2008), and Ipopt (Wächter and Biegler, 2006). Our implementation, plus all code used in this experiment, are publicly available at github.com/mespadoto/optmap.

Table 5: Software used for the evaluation.

Library	**Software publicly available at**
Images	github.com/JuliaImages/Images.jl
ColorTypes	github.com/JuliaGraphics/ColorTypes.jl
ColorSchemes	github.com/JuliaGraphics/ColorSchemes.jl
Luxor	github.com/JuliaGraphics/Luxor.jl
CSV	github.com/JuliaData/CSV.jl
DataFrames	github.com/JuliaData/DataFrames.jl
MultivariateStats	github.com/JuliaStats/MultivariateStats.jl
Optim	github.com/JuliaNLSolvers/Optim.jl
Clp	github.com/jump-dev/Clp.jl
Cbc	github.com/jump-dev/Cbc.jl
GLPK	github.com/jump-dev/GLPK.jl
Ipopt	github.com/jump-dev/Ipopt.jl

5 DISCUSSION

We discuss next how OptMap performs with respect to the criteria laid out in Section 1.

Quality (C1): Figures 2, 3 and 4 show examples of the quality of the visualizations and the kind of insight they can provide for optimization problems. Our dense maps are pixel-accurate, in the sense that they show *actual* information inferred from the nD function f under investigation at each pixel, *without* interpolation. This is in contrast with many other dimensionality reduction methods which either show a *sparse* sampling of the nD space (by means of a 2D scatterplot), leaving the user to guess what happens between scatterplot points; or use interpolation in the 2D *image* space to 'fill' such gaps (Martins et al., 2014; Silva et al., 2015; van Driel et al., 2020), which creates smooth images that may communicate wrong insights, since we do not know the underlying projection is continuous.

Genericity (C2): We show how our technique performs for optimization problems with varying nature, complexity, and dimensionality. We also show that our method can be used simply for visualizing high-dimensional, continuous, functions by a *single* 2D image, in contrast to multiple images that have to be navigated and correlated by interaction (van Wijk and van Liere, 1993). We also show that our technique is independent of the optimization solvers being used;

Simplicity (C3): We use PCA for direct and inverse projections, which is a very well known, simple, fast, and deterministic projection method. OptMap's complete implementation has about 250 lines of Julia code. Note that we also experimented with other methods for the direct projection – namely, t-SNE as learned by NNP (Espadoto et al., 2020) – and inverse projection – namely, NNInv (Espadoto et al., 2019b), obtaining good results. However, for the optimization problems presented in this paper, PCA yielded better results (based on ground truth comparison). Since PCA is also simpler and faster than NNP and NNInv, we preferred it in our work.

Ease of use (C4): Apart from the timing experiment in Section 4.4, we executed all experiments using the same maximum number of sample points N_{max} with good results, which shows that the technique requires little to no tuning to work properly;

Scalability (C5): Section 4.4 shows that our method is highly scalable with the number of sample points and dimensions, which enables its interactive usage during the development cycle of optimization models. On the other hand, scalability is inherently limited by the resolution used to create the dense grid G^n. If the number of dimensions n *and* the sampling rate of each dimension become too high, the total number of samples N becomes prohibitive. This is inherent to the fact that we aim to capture the *dense space* spanned by the variables x_i, rather than the *sparse point cloud* that typical DR methods take as input. To alleviate this, one could (a) consider different sampling rates for x_i, based on prior knowledge on how f depends on each of them; (b) use OptMap interactively by 'zooming in and out' of different variable ranges to explore the high-dimensional space; or (c) use multiresolution techniques, akin to those already present in various optimizers.

Limitations: The projected points, such as the starting, trace, and solution points, are placed in the 2D image space at *approximate* positions, due to the inherent discrete nature of the pixel grid G^2. This can cause situations such as the one in Fig. 4 (Sphere problem), where the optimal point found by the solver – which is obviously feasible – is placed slightly inside the unfeasible region, which can be misleading. Secondly, we noticed that due to the inherently imperfect mapping between nD and 2D spaces, equality constraints that compare against constants might not be satisfied during the evaluation, which will make the drawing of feasible regions fail.

A separate aspect relates to the fact that P can map multiple different points $\mathbf{x} \in D$ to the same pixel $\mathbf{p} \in G^2$. Hence, the color assigned to $\mathbf{p}$ should ideally reflect the *combination* of values $f(\mathbf{x})$ of all these points $\mathbf{x}$. For categorical-valued functions f, this can be done by using voting schemes that compute the confidence of the final coloring (Rodrigues et al., 2019). A low-hanging fruit for future work is to (efficiently) extend such schemes to our real-valued functions f by using aggregation strategies such as average, min, or max.

6 CONCLUSION

In this paper we presented OptMap, an image-based visualization technique that allows the visualization of multidimensional functions and optimization problems. OptMap exploits the idea of constructing dense maps of high-dimensional spaces, by using direct and inverse projections to map these spaces to a 2D image space. Suitable choices for the sampling of these spaces, as well as using efficient and well-understood direct and inverse projection implementations, make OptMap scalable to real-world problems. We show that OptMap performs well in different scenarios, such as unconstrained and constrained optimization, with many examples that demonstrate its genericity and speed. Additionally, we show that OptMap can be used for the visualization of standalone high-dimensional functions, even when these are not part of an optimization problem.

Several future work directions exist. First and foremost, it is interesting to consider using more accurate direct and inverse projections for constructing OptMap. Secondly, we consider using OptMap in concrete applications, and gauging its added-value in helping engineers designing better optimization strategies, as opposed to existing tools-of-the-trade for the same task.

ACKNOWLEDGMENTS

This study was financed in part by FAPESP (2015/22308-2 and 2017/25835-9) and the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

REFERENCES

Amorim, E., Brazil, E. V., Daniels, J., Joia, P., Nonato, L. G., and Sousa, M. C. (2012). iLAMP: Exploring high-dimensional spacing through backward multidimensional projection. In *Proc. IEEE VAST*, pages 53–62.

Bertini, E., Tatu, A., and Keim, D. (2011). Quality metrics in high-dimensional data visualization: An overview and systematization. *IEEE TVCG*, 17(12):2203–2212.

Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98.

Brooke, A., Kendrick, D., Meeraus, A., Raman, R., and America, U. (1998). The general algebraic modeling system. *GAMS Development Corporation*, 1050.

Buja, A., Cook, D., and Swayne, D. F. (1996). Interactive high-dimensional data visualization. *Journal of Computational and Graphical Statistics*, 5(1):78–99.

Dantzig, G. B. (1990). Origins of the simplex method. In *A history of scientific computing*, pages 141–151.

Dunning, I., Huchette, J., and Lubin, M. (2017). JuMP: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320.

Espadoto, M., Hirata, N., and Telea, A. (2020). Deep learning multidimensional projections. *J. Information Visualization*. doi.org/10.1177/1473871620909485.

Espadoto, M., Martins, R. M., Kerren, A., Hirata, N. S., and Telea, A. C. (2019a). Towards a quantitative survey of dimension reduction techniques. *IEEE TVCG*. Publisher: IEEE.

Espadoto, M., Rodrigues, F. C. M., Hirata, N. S. T., Hirata Jr., R., and Telea, A. C. (2019b). Deep learning inverse multidimensional projections. In *Proc. EuroVA*. Eurographics.

Forrest, J., Vigerske, S., Ralphs, T., Hafer, L., jpfasano, Santos, H. G., Saltzman, M., h-i gassmann, Kristjansson, B., and King, A. (2020a). coin-or/clp.

Forrest, J., Vigerske, S., Santos, H. G., Ralphs, T., Hafer, L., Kristjansson, B., jpfasano, Straver, E., Lubin, M., rlougee, jpgoncal1, h-i gassmann, and Saltzman, M. (2020b). coin-or/cbc.

Fourer, R., Gay, D. M., and Kernighan, B. W. (2003). *AMPL. A modeling language for mathematical programming*. Thomson.

Guenin, B., Könemann, J., and Tuncel, L. (2014). *A gentle introduction to optimization*. Cambridge University Press.

Hager, W. W. and Zhang, H. (2006). Algorithm 851: CG_DESCENT, a conjugate gradient method with guaranteed descent. *ACM Transactions on Mathematical Software (TOMS)*, 32(1):113–137.

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90–95. Publisher: IEEE Computer Society.

Joia, P., Coimbra, D., Cuminato, J. A., Paulovich, F. V., and Nonato, L. G. (2011). Local affine multidimensional projection. *IEEE TVCG*, 17(12):2563–2571.

Jolliffe, I. T. (1986). Principal component analysis and factor analysis. In *Principal Component Analysis*, pages 115–128. Springer.

Kantorovich, L. V. (1960). Mathematical methods of organizing and planning production. *Management science*, 6(4):366–422.

Liu, D. C. and Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3):503–528.

Liu, S., Maljovec, D., Wang, B., Bremer, P.-T., and Pascucci, V. (2015). Visualizing high-dimensional data: Advances in the past decade. *IEEE TVCG*, 23(3):1249–1268.

Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-SNE. *JMLR*, 9:2579–2605.

Makhorin, A. (2008). GLPK: GNU Linear Programming Kit).

Martins, R., Coimbra, D., Minghim, R., and Telea, A. (2014). Visual analysis of dimensionality reduction quality for parameterized projections. *Computers & Graphics*, 41:26–42. Publisher: Elsevier.

McInnes, L. and Healy, J. (2018). UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv:1802.03426v1 [stat.ML]*.

Mogensen, P. K. and Riseth, A. N. (2018). Optim: A mathematical optimization package for Julia. *Journal of Open Source Software*, 3(24):615.

Nelder, J. A. and Mead, R. (1965). A simplex method for function minimization. *The computer journal*, 7(4):308–313.

Nonato, L. and Aupetit, M. (2018). Multidimensional projection for visual analytics: Linking techniques with distortions, tasks, and layout enrichment. *IEEE TVCG*.

Paulovich, F. V., Nonato, L. G., Minghim, R., and Levkowitz, H. (2008). Least square projection: A fast high-precision multidimensional projection technique and its application to document mapping. *IEEE TVCG*, 14(3):564–575.

Rastrigin, L. A. (1974). Systems of extremal control. *Nauka*.

Rodrigues, F., Espadoto, M., Hirata, R., and Telea, A. C. (2019). Constructing and visualizing high-quality classifier decision boundary maps. *Information*, 10(9):280.

Rosenbrock, H. (1960). An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175–184.

Roweis, S. T. and Saul, L. L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326. Publisher: American Association for the Advancement of Science.

Silva, R. d., Rauber, P., Martins, R., Minghim, R., and Telea, A. C. (2015). Attribute-based visual explanation of multidimensional projections. In *Proc. EuroVA*.

Styblinski, M. and Tang, T.-S. (1990). Experiments in nonconvex optimization: stochastic approximation with function smoothing and simulated annealing. *Neural Networks*, 3(4):467–483.

Tenenbaum, J. B., Silva, V. D., and Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323. Publisher: American Association for the Advancement of Science.

Torgerson, W. S. (1958). *Theory and Methods of Scaling*. Wiley.

van Driel, D., Zhai, X., Tian, Z., and Telea, A. (2020). Enhanced attribute-based explanations of multidimensional projections. In *Proc. EuroVA*. Eurographics.

van Wijk, J. J. and van Liere, R. (1993). Hyperslice. In *Proc. Visualization*, pages 119–125. IEEE.

Wächter, A. and Biegler, L. T. (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57.

Wicklin, R. (2018). Visualize the feasible region for a constrained optimization. https://blogs.sas.com/content/iml/2018/11/07/visualize-feasible-region-constrained-optimization.html.