J Vis (2022) 25:127–141
https://doi.org/10.1007/s12650-021-00786-8

REGULAR PAPER

David Hägele · Moataz Abdelaal · Ozgur S. Oguz · Marc Toussaint · Daniel Weiskopf

Visual analytics for nonlinear programming in robot motion planning

Received: 5 May 2021 / Revised: 29 June 2021 / Accepted: 14 August 2021 / Published online: 13 September 2021

Abstract Nonlinear programming is a complex methodology where a problem is mathematically expressed in terms of optimality while imposing constraints on feasibility. Such problems are formulated by humans and solved by optimization algorithms. We support domain experts in their challenging tasks of understanding and troubleshooting optimization runs of intricate and high-dimensional nonlinear programs through a visual analytics system. The system was designed for our collaborators' robot motion planning problems, but is domain agnostic in most parts of the visualizations. It allows for an exploration of the iterative solving process of a nonlinear program through several linked views of the computational process. We give insights into this design study, demonstrate our system for selected real-world cases, and discuss the extension of visualization and visual analytics methods for nonlinear programming.

Keywords Visual analytics · Nonlinear programming · Optimization · Design study · Loss landscape

1 Introduction

Nonlinear constraint optimization, also known as nonlinear programming (NLP), deals with finding optima within constrained sets of variables. Only little work has been published concerning visualization in this field, which is surprising since a substantial amount of research has been conducted in the closely related fields of discrete and also unconstrained optimization. NLP comprises two separate stages: the modeling stage and the solving stage. In the modeling stage, the problem needs to be expressed in terms of an objective function and associated equality or inequality constraint functions that may be nonlinear. A classical example from economics is to optimize spending for greatest profit (objective) while staying within budget (constraint). In the solving stage, an iterative algorithm moves through the (likely high-dimensional) space of the modeled problem to find the optimal location with respect to the objective while making sure to satisfy all of the imposed constraints.

Due to high dimensionality, large number of constraints, and nonlinearity, which gives rise to local optima and disconnected feasible regions, it is challenging to grasp such optimization problems. Even more challenging is the comprehension of unexpected or unsatisfactory behavior of a solver when applied to a problem.

D. Hägele (✉) · M. Abdelaal · O. S. Oguz · D. Weiskopf
University of Stuttgart, Stuttgart, Germany
E-mail: david.haegele@visus.uni-stuttgart.de

M. Toussaint
Technische Universität Berlin, Berlin, Germany

To help NLP experts in understanding their problems better as well as the corresponding behavior of the applied solvers, we want to leverage visualization of the internal steps of an optimization process. We propose a visual analytics approach for a postmortem analysis of optimization runs of robot motion planning problems that we developed in tight collaboration with domain experts. We focus on visualizing the high-dimensional optimization landscape as seen by the optimizer, to be able to reason about its behavior, and on representing the evolution of the solution throughout the optimization in order to be able to interpret high-dimensional loci as intermediate solutions to the motion planning problem.

Please note that this paper is an extension of our previous work (Hägele et al. 2020) that is verbatim in large parts. The extension consists of a new section on plane orientation strategies for loss landscapes (Sect. 6), elaborations on dataset characteristics, and visualization techniques. The contributions we carry over to this extension are: (1) a visual analytics system for the analysis of constrained optimizations for robot motion planning, and (2) a report of our design study process and lessons learned. In this work we extend our contributions by (3) a novel plane orientation strategy for loss landscape visualizations.

2 Related work

This section is divided into two parts. First, we review related work in the field of visualization of optimization that includes NLP, linear programming, constraint programming, multi-objective optimization, as well as unconstrained optimization. Then, we discuss related work concerned with the visualization of temporal high-dimensional data.

2.1 Visualization of optimization

Despite its strong potential (Messac and Chen 2000; Goodwin et al. 2017), the area of visualizing NLP remains unexplored to a large extent. Androulakis and Vrahatis (1996) proposed OPTAC, a tool for analyzing and visualizing the convergence behavior of unconstrained optimization algorithms. Charalambos and Izquierdo (2001) visualize the geometric shapes of the feasible regions of linear programs. The method uses three-dimensional Cartesian coordinates and, therefore, is limited to three-dimensional optimization problems. To display high-dimensional planes, Chatterjee et al. (1993) use parallel coordinates plots instead. Since the geometry in linear programs is simple and solvers are fundamentally different, these approaches are not applicable for NLP.

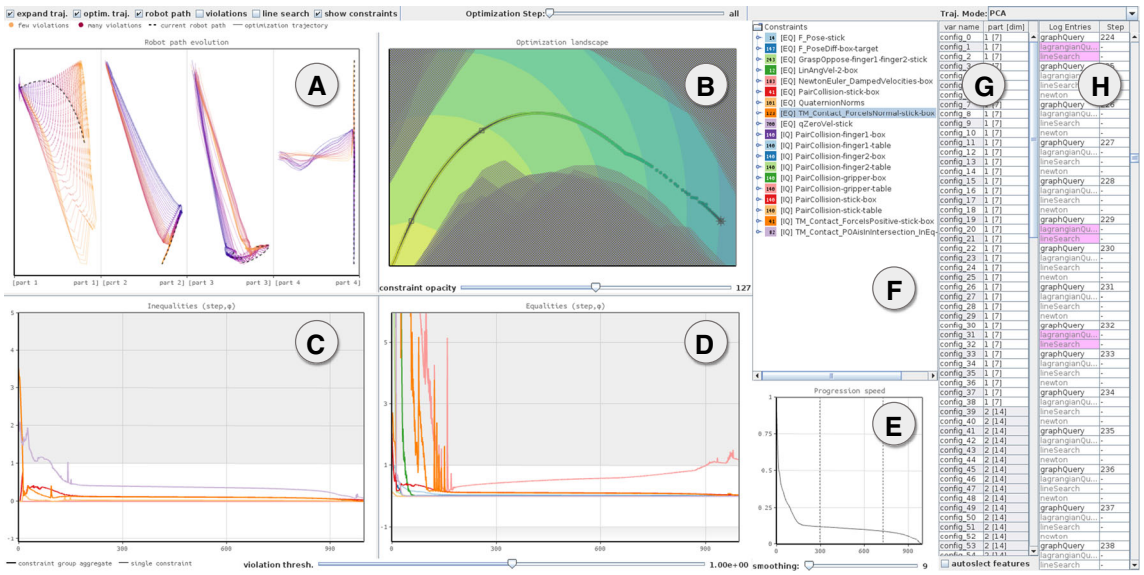


Fig. 1 User interface of our visual analytics system for nonlinear programming processes. Using multiple coordinated views, a user can assess the evolution of a problem's solution, constraint values, and progression speed throughout an optimization. **a** Robot path evolution, **b** optimization landscape, **c**, **d** equality and inequality constraint evolution, **e** optimizer progression speed, **f** constraint groups, **g** robot configurations list, **h** optimization log

The area of constraint programming, in contrast to NLP, received a lot of attention from visualization practitioners. Most of the work done in this area focuses on visualizing the search tree resulting from constraint programs (Carro and Hermenegildo 2000b; Pu and Lalanne 2000; Simonis et al. 2010; Shishmarev et al. 2016; Goodwin et al. 2017). However, these techniques cannot be applied to NLP, due to the differences in modeling and solvers (Heipcke 1999). While solving constraint programs involves tree traversal and dynamic programming, NLPs consist of differentiable implicit surfaces, and their solvers are based on algebraic methods such as gradient descent.

Instead of focusing on visualizing the search tree resulting from constraint programming, others attempt to visualize the evolution of variables, constraints, and the interaction between them using matrix views (Carro and Hermenegildo 2000a; Ghoniem et al. 2005, 2004). While matrices provide good representations to explore the relationship between constraints and variables, they have scalability issues, making them only suitable for exploring optimization problems with a small number of variables and constraints. Despite the differences, we find our work shares the same goal, i.e., exploring the evolution of variables and constraints.

The visualization of multi-objective optimization is yet another neighboring field to NLP. Most of the work in this field is concerned with visualizing the solution set in the objective space. Therefore, different visualization methods for high-dimensional data are used. Other methods apply dimensionality reduction to map the high-dimensional objective space to a two-dimensional space. Tušar and Filipič (2014) provide a comprehensive review of these methods. Although multi-objective optimization and NLP address two different problems, they both share the notion of high dimensionality.

The visualization of high-dimensional unconstrained problem optimization such as in neural networks is discussed by Goodfellow and Vinyals (2015), who use a straight line from initialization to found optimum to sample and analyze the loss function. This gave rise to the loss landscape visualization technique by Li et al. (2018). They use a 2D plane to sample the loss function and obtain a contour plot to analyze a subspace into which the optimizer's trajectory can be projected. We extend this technique for NLP to show constraints within the landscape of the objective function.

2.2 Visualization of temporal high-dimensional data

There are numerous methods for visualizing high-dimensional data (Liu et al. 2017), e.g., scatterplot matrices (Andrews 1972), glyphs (Chernoff 1973), parallel coordinates plots (Inselberg 1985), and star coordinates (Kandogan 2000). These techniques, however, do not scale with a large number of dimensions. Therefore, other approaches project high-dimensional data into low-dimensional space using various dimensionality reduction techniques (Nonato and Aupetit 2018). Introducing the time dimension poses a visualization challenge, as the visual representation needs to convey not only the relation between the different dimensions but also their temporal context and evolution. Aigner et al. (2011) provide a comprehensive overview of such time-oriented visualization.

A rather straightforward approach is to include time as an additional dimension in the traditional high-dimensional data representations. For example, Wong and Bergeron add an axis for the time dimension in parallel coordinates (Wong and Bergeron 1994). Similarly, TimeWheel (Tominski et al. 2004) arranges the other axes in a circular layout around the time axis. These, however, provide a poor representation of the temporal context information and are prone to become cluttered. Other approaches use depth to encode the temporal information, resulting in 3D parallel coordinates (Wegenkittl et al. 1997; Gruendl et al. 2016), 3D star coordinates (Noirhomme-Fraiture 2002), or space-time cubes (Bach et al. 2014, 2015). These methods, however, have scalability concerns regarding the size of the data. Additionally, the use of interaction and animation is a necessity to avoid occlusion problems.

More recent approaches use dimensionality reduction to show temporal information. Jäckle et al. (2016) proposed temporal multidimensional scaling (TMDS) for visualizing multivariate time series data. Bach et al. (2016) and van den Elzen et al. (2016) showed the temporal progression of datasets by projecting the individual snapshots of datasets as points in 2D space. The points of subsequent snapshots are then connected by lines, while color is used to encode time.

In our work, we think of the optimization process as a sequence of intermediate solutions. Each solution is characterized by a set of high-dimensional decision variables. Thus, we can adopt the same techniques (Bach et al. 2016; van den Elzen et al. 2016) to visualize the evolution of the optimization process. In our work, in contrast, we deal with two different notions of time simultaneously: the time of a robot's motion and the time of the optimization process.

Torsney-Weir et al. (2018) studied multi-dimensional shapes using their hypersliceplorer algorithm. Using separate views for pairs of dimensions does not scale well with our problems, however.

3 Background

In this section, we summarize the domain background of robot motion planning and nonlinear programming. We also present the requirements for a visualization system that we elicited from the domain experts. From here on we will use *NLP* as an abbreviation for both nonlinear program and nonlinear programming.

3.1 Robot motion planning

Motion planning is the problem of finding a collision-free path to move an object from an initial state to a desired goal state (Latombe 2012). It is a crucial problem in many fields, including robotics, computational biology (drug design, protein folding), virtual prototyping, manufacturing, and computer graphics. In this paper, we focus on robot motion planning as our application domain. However, we expect that our visual analytics approach will carry over to other applications. An example of such motion planning problems is illustrated in Fig. 3 for a robot arm corresponding to the ones depicted in Fig. 2.

There are different approaches for finding a valid path connecting the source and target configurations. LaValle (2006) provides an overview of the most important ones. Grid- and sampling-based approaches (Kondo 1991; Chen and Hwang 1998; Kavraki and Latombe 1994; Kavraki et al. 1996) discretize the configuration space and use graph search algorithms to find a valid path in the resulting topological space.

Another approach for finding a valid path is based on nonlinear optimization (Toussaint 2015). This involves the formulation and solving of an NLP that describes the desired goal mathematically and imposes constraints to ensure a collision-free path and feasible motion. In contrast to the aforementioned methods, nonlinear optimization does not require any discretization of the configuration space. Therefore, it is possible to find an optimal path if the problem is well-defined.

However, nonlinear optimization requires an initial guess of the solution and the method is prone to get stuck in local minima (Chinneck 2006). To avoid that, it is common to use path finding algorithms as a first step to provide a reasonable initial guess. While this might be enough to find a valid solution, it does not provide insights into the internal mechanism of the optimization algorithms: How fast does the optimizer progress? Or which constraints hinder the algorithm the most? Such questions cannot be answered without a proper investigation of the inner workings of the optimization.

To this end, there is a critical need for visualization tools to debug and troubleshoot the optimization algorithms (Messac and Chen 2000; Goodwin et al. 2017). Having such tools can help domain experts formulate hypotheses about the behavior of the optimizer that could eventually lead to a better reformulation

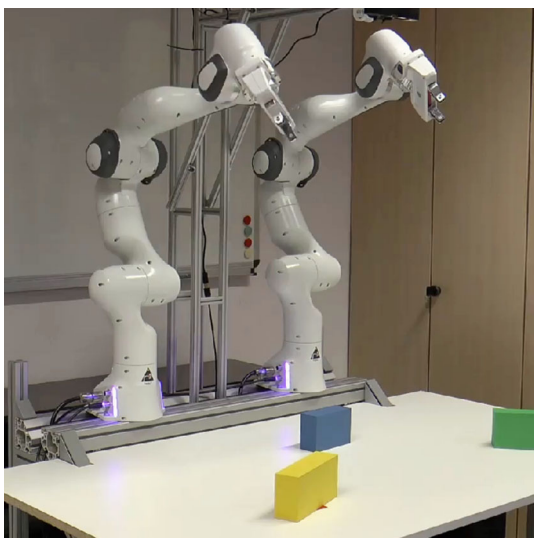


Fig. 2 Photo of two robot arms with 7 degrees of freedom in our domain experts' robotics lab

of the cost function and/or the constraints. In this context, we were approached by robotics researchers. During the course of this design study, we have been collaborating with them for more than 9 months, integrating their expertise and knowledge about their domain-specific problems. To better understand the domain problem we tried to characterize it by eliciting requirements for a visual analytics system and tasks to be achievable from the experts. We were able to identify two high-level information needs:

Q1: When an optimization requires a lot of time to converge, what is taking the optimizer so long?
Q2: When an optimization produces an infeasible solution, what prevents the optimizer from getting to a feasible location?

To be able to analyze an optimization with regard to these questions, we formulated the following analysis tasks that the experts want to perform on optimizations of their robot motion planning framework:

T1: Identify and characterize different phases of the optimization process.
T2: Identify when the optimizer converges faster or more slowly to a local minimum during the optimization process.
T3: Characterize the evolution of the constraint function values during the optimization process.
T4: Identify the shape and boundaries of a constraint.
T5: Identify the shape and boundaries of the feasible regions.

Throughout the project, we followed a design study process (Sedlmair et al. 2012), regularly meeting with our collaboration partners to continuously refine our design and to update requirements.

3.2 Nonlinear programming

Nonlinear programs consist of three parts that allow expressing an optimization problem with auxiliary conditions on the solution, known as the constraints. It typically reads like this:

$$\text{minimize} f(x) \text{subject to} h_i(x) = 0, g_j(x) \leq 0 \tag{3.1}$$

In this, x is the set of decision variables that can be varied. It makes sense, in our case, to think about x as a large vector describing a robot's motion. The objective function f is to be minimized while all of the inequality constraints g_j and equality constraints h_i have to be satisfied at the solution x^*. In an NLP, at least one of these functions is nonlinear. In high-dimensional space, the equalities define hypersurfaces, and inequalities define hypersurface-confined sets in which the optimal x^* has to be located.

Including time into an optimization problem is often done by discretization into several time steps. This also applies to our case of robot motion, where the motion path consists of joint configurations representing the robot's poses over time.

The objective and constraints are used to express the robot's task mathematically and to guarantee that the resulting motion is physically viable. In such time-dependent optimization problems, some constraints will typically apply to individual time steps only to enforce valid states, and others will take several times steps into account to enforce valid state transitions or global properties.

Technical Background: We now discuss a class of iterative algorithms to solve NLPs. We will not go into details but sketch the general mechanic that is common in the log-barrier, squared penalty, and also the augmented Lagrangian methods. All of these want to find a solution that satisfies the Karush–Kuhn–Tucker conditions (Kuhn and Tucker 2014) (a set of conditions that hold at a feasible minimum of an NLP) and thus determine the dual variables. This is done by constructing an unconstrained problem $L_{\lambda,\kappa}(x)$ from the NLP

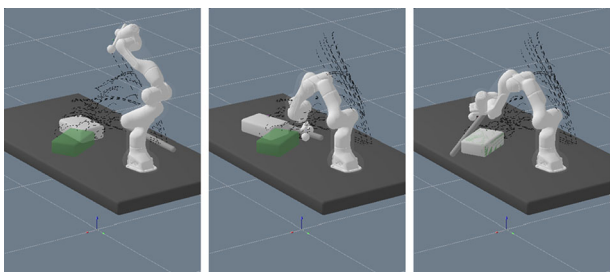


Fig. 3 Animation frames from an optimized robot motion sequence for a box-pushing problem. The robot arm picks up a stick and then moves a box to a target location. The animation was created by our collaborators' motion planning framework

that can be minimized using standard techniques like gradient descent or Newtons's method. For example, in the augmented Lagrangian method, $L_{\lambda,\kappa}(x) = f(x) + \kappa^\top h(x) + \lambda^\top g(x) + ||h(x)||_2^2 + \sum_j [g_j(x) > 0] g_j(x)^2$. The unconstrained problem is then minimized repeatedly while updating the dual parameters λ, κ, and thus changing the impact of constraints, in between.

A pseudocode implementation is shown in 1. A line search mechanism (most inner loop decreasing step size) is leveraged to ensure sufficient decrease in each step to prevent overshooting valleys and enforces satisfaction of the Wolfe condition (Wolfe 1969).

Besides the problem formulation, i.e., functions f, g, and h, this algorithm is the source of our data for visualization. We define the optimization trajectory as the sequence S of arguments $x' = x + a\delta$ that are tested during line search in the `argmin` procedure of 1 (line 21):

$$S = \{x_{\text{init}}, x_1, x_2, .., x^*\} \tag{3.2}$$

Furthermore, we obtain a detailed log of the sequence in which different stages in the algorithm were executed: updates to the dual variables (line 10), evaluations of the loss function and corresponding function values of objective and constraints (line 20), step size decrease during line search (line 22), and updates to x (line 24).

Listing 1: Pseudo Code for NLP Solver

```
1  FUNCTION solveNLP
2  INPUT: objective f, equalities h,
3    and inequalities g
4  OUTPUT: optimal location x
5  BEGIN
6   define unconstrained problem L_{λ,κ} from f, h, and g
7   initialize x randomly (or informed)
8   DO
9    x = argmin of L_{λ,κ} with initialization x
10   update dual parameters λ, κ
11  UNTIL convergence
12 END
13
14 FUNCTION argmin
15 INPUT: loss function L, initialization x
16 OUTPUT: optimal location x
17 BEGIN
18  initialize stepsize a = 1 (or previous value)
19  WHILE a reasonably large
20   set descent direction δ = Newton step for L at x
21   WHILE first Wolfe condition NOT satisfied
22    decrease stepsize a
23   END WHILE
24   x = x + aδ
25   increase stepsize a
26  END WHILE
27 END
```

3.3 Dataset characteristics and sizes

In the previous subsection, we gave an introduction to NLP and how an iterative solver generates all the data that we like to analyze. As this is quite generic, we want to provide more details on the size of typical problem instances, to give a better impression of the challenge at hand. Further details on how motion planning problems are cast to NLPs are given elsewhere (Toussaint 2015).

First, let us examine the dimensionality of the optimization space, which is determined by the motion path represented by the argument x. The path's dimensionality depends on the degrees of freedom of the robot ($\approx$ 7 for a robotic arm as depicted in Fig. 2, $\approx$ 30 for humanoids), the number of objects with which it interacts (1 or 2 in our cases), and the number of time points used to discretize the path (80 to 200 in our cases). For instance, one of our examples with 180 discrete time points, 7 degrees of freedom, and 2 objects has a 2173-dimensional path representation.

The constraints of the NLP are quite diverse, and their number depends on the robot's task and modeling of the motion problem. However, for a *semantic* constraint such as "prevent collision with the wall" or "do

not accelerate while grabbing the object," there will be one or several *effective* constraints in the final NLP for a subset of time points on the path. Some constraints apply only to a single time point, whereas others exist for every time point of the problem. Our examples range from a single up to twenty *semantic* constraints, resulting in 80 to 3000 *effective* constraints.

4 Methods

In this section, we will introduce our visual analytics system and discuss the special techniques used to visualize the optimization process, i.e., the loss landscape visualization and the robot path evolution visualization.

4.1 Visual analytics system

Our visual analytics system combines and links different views to facilitate the exploration of an optimization process. Due to the iterative nature of the algorithm, the optimization process describes the evolution of essentially two things: The evolution of objective and constraint function values, and the evolution of the solution described by the sequence of intermediate solutions $s_i \in S$ on the optimization trajectory.

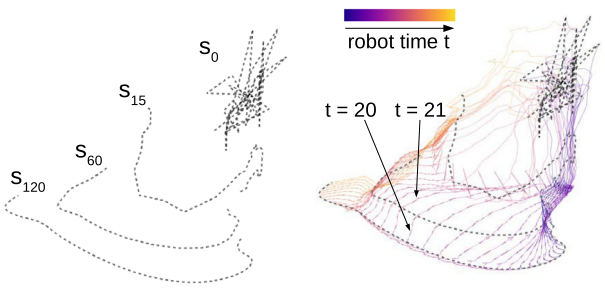
Constraint Value Evolution To show the evolution of function values, which allows for task T3 to be accomplished, we provide line charts for equality and inequality constraints, plotting $h(s_i)$ and $g(s_i)$ with optimization step i on the x-axis (views C and D in Fig. 1). We group constraints by names due to their large number, which allows us to prevent visual clutter. For each constraint group, we plot the maximum value of all its constraints as aggregation. Let H_k denote a group of equality constraints, and G_k an inequality constraint group, respectively. The aggregates of the groups are then defined like this:

$$\bar{h_k}(s_i) = \max_{h_j \in H_k}(h_j(s_i)); \bar{g_k}(s_i) = \max_{g_j \in G_k}(|g_j(s_i)|)$$

Groups are listed in a separate view (F in Fig. 1) and can be expanded if desired, to reveal the individual unaggregated constraints in the plot.

Robot Path Evolution To show the evolution of the problem's solution we employ a time curves (Bach et al. 2016) approach in which we reduce dimensionality of the joint configurations of $s = \{c_1 \ldots c_T\}$, that is the motion path (view A in Fig. 1). In this 2D representation of the solution, we can show its evolution. Connecting subsequent joint configurations of s through line segments yields a time curve describing the robot motion. Connecting the same joint configuration of subsequent intermediate solutions s_i results in a time curve describing the evolution of that particular configuration throughout the optimization. Figure 4 illustrates this technique that is integrated in view A in Fig. 1.

Optimization Landscape To combine constraints and solution evolution we adapt the loss landscape technique (Li et al. 2018) to NLP, which also enables us to observe the behavior of the optimizer as it travels through high-dimensional space (view B in Fig. 1). Using isobands, we can show contours of the objective



(a) path evolution **(b)** per configuration trajectories

Fig. 4 Figures show the motion path evolution over the optimization process using PCA for projection. **a** It can be observed how the path evolves from a chaotic random shape (top) to a smooth curve (bottom). **b** The trajectory from initialization to final solution for each configuration c_t is shown and color-coded by time t. The underlying optimization corresponds to an NLP for pushing a box in a circular motion similar to the illustration in Fig. 3

function and constraints on a 2D plane slice. For inequality constraints, we plot isobands for $h(x) > \epsilon$ and $h(x) < -\epsilon$, where $\epsilon > 0$ is a threshold for constraint violation. The advantage over an isoline of $h(x) = 0$ is that we can get a sense of how badly a constraint is violated.

Similarly, inequalities have isobands $g(x) > \epsilon$. Making this threshold variable allows for examining the proximity of the optimization trajectory to the constraint's surface even when the surface is not intersecting the plane slice. The resulting isobands of the constraints are rendered with a transparent checkerboard pattern to stand out against the isobands of the objective function below; see Figs. 1b or 7 (right) for reference.

The slice plane is spanned by the vector connecting the point of the currently selected optimization step with the minimum x^*, and a perpendicular vector that is a linear combination of the first two principal directions of the optimization trajectory S. We choose this as the default plane orientation strategy for being the quickest to compute. However, we found that using other strategies involving *most prominent directions* leads to significant improvements of trajectory representation. A discussion and evaluation of different plane orientation strategies can be found in 6.

Selecting three optimization steps aligns the slice to coincide with the corresponding points. We draw the optimization trajectory into the same plot and encode proximity to the plane as line thickness to determine when the optimizer moves away from it, as it is crucial for judging the degree of correspondence between landscape and trajectory.

Optimization Progression Speed For quickly identifying steps of slow or fast optimizer progression (Task T2), we introduce a line plot: the *progression speed plot* (see Fig. 1 E). It follows the metaphor of the optimizer walking downhill, has a similar appearance as a loss curve, but displays the descend speed (i.e., optimization step size). The x-axis of the plot shows the optimization steps, the y-axis the remaining optimization trajectory length. Therefore, the slope of the curve is the relative step size for the respective step. This way, fast solver progression can be identified from steep slopes, slow progression from flat slopes, respectively. We allow for a smoothing of the curve by aggregating several steps with a sliding window and using the effective distance traveled within the window. This has the advantage of removing backtracking steps during line search from the total trajectory length.

Interactive Exploration Exploration of an optimization is enabled by interaction through selecting or cycling through optimization steps, constraints, or configurations, as well as through zooming and panning in the different views that respond to the selections made.

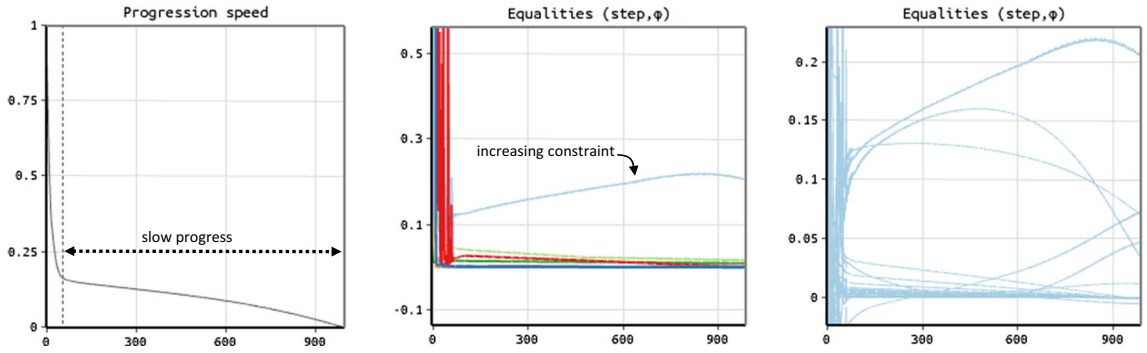
To select an optimization step, the user can either click into the progression speed and constraint plots, use a slider (on top of the GUI), or select a corresponding entry from an optimization log entry list (H in Fig. 1). The log entries give an overview of the inner workings of the optimization algorithm throughout the process, such as Newton steps taken, line search condition checked, or dual parameter updates done. We highlight line search backtracking steps in pink, graph query entries correspond to optimization steps. The naming of these entries stems from the structure of the log, output by the optimizer. Selecting individual joint configurations (robot time instants) can be done from list G in Fig. 1, which results in highlighting the corresponding trajectory in the path evolution view (A).

4.2 Implementation

We implemented our visual analytics system as a desktop application in the Java programming language based on the *AWT/Swing* GUI environment. Libraries employed include *Smile*, *EJML*, *JPlotter*, *jackson*, and *OkHttp*. The solver runs in a separate software (*KOMO* Toussaint 2014), producing optimization log files that our implementation facilitates to explore interactively. To obtain samples of the optimization space, we implemented an HTTP-server-based interface in the motion planning framework that runs simultaneously. We chose this method of data transfer due to its versatility and sustainability for future projects.

5 Case study

To showcase the usefulness of our system we present a case study and insights we could find with respect to a particular motion planning problem. We analyze an optimization run of a typical motion planning problem concerned with information need Q1. The robot in this problem is supposed to pick up and throw a ball so that it bounces of the ground and a wall to finally hit a target area.



(a) progression overview **(b)** constraint value evolution **(c)** expanded constraint group

Fig. 5 Views examined in the first part of our case study. We observe slow progress throughout the majority of the optimization (**a**). We find an equality constraint group that increases over the process (**b**), and on expansion, see that several constraints of the group behave undesirably (**c**), i.e., not converging to zero

When taking a look at the corresponding optimization in our visual analytics system, we would like to first get an overview. We typically start with checking the progression speed plot from which we can see, in this case, the large number of steps taken, which is around 1000 (Fig. 5). Examining the process' progression behavior (T2), we can observe that the optimizer is leaping forward in the first few iterations and then harshly slows down. Clicking the plot where we identified the sudden decrease in speed selects the corresponding optimization step (step 57 in this case). From there on, the optimizer crawls to its final position, also hardly accelerating over the remaining iterations. This behavior is quite suspicious to us since this seems extremely ineffective, so we want to know what is going on.

Next, we take a look at the evolution of constraint values to check for any anomalies there (T3). Zooming in a little on the equality constraint line chart reveals a constraint that is increasing instead of decreasing during the slow part of the optimization (blue line in Fig. 5). The graph shows the maximum absolute value for the constraints sharing the same name, so to identify which constraints exactly are behaving in this way we expand this group of constraints to see the individuals (Fig. 5). From the expanded constraints, we see that most of them actually converge to zero as desired, but some of them rise up in the end and one is located far away from zero.

From the corresponding entries in the constraint tree, we can read off the robot time instants, i.e., the joint configurations to which the strange behaving constraints relate. Three consecutive configurations in the second part of the motion are related. This tells us that it is the part where the robot grabs the ball and throws it. Selecting the entry highlights the trajectories in the path evolution view. We use this to get an idea of the

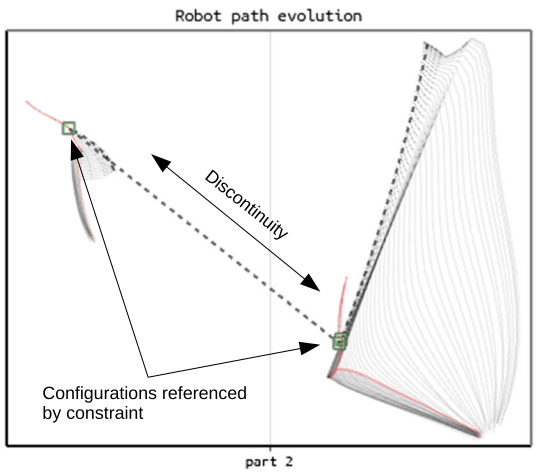


Fig. 6 Examining the part on the motion path that the misbehaving constraints relate to. We see a discontinuity in the path that corresponds to a change in configuration vector semantics. The constraints are related to the moment when the robot releases the ball

region on the path that is affected by the constraint (Fig. 6). Examining this part, we recognize a discontinuity in the motion path. This is due to the semantics of dimensions changing from one configuration to the next, indicating an event on the motion path that requires a different mathematical modeling. It tells us that the constraint is related to the exact moment of releasing the ball from its grip. This information could be valuable to the author of the motion problem when wanting to reformulate the problem to achieve better convergence.

We now want to go back to the original issue of slow progression. Checking the log table for frequent line searches that could cause slow progression reveals that there are no line searches taking place in the largest portion of the process. As a next step, we consult the landscape view to take a look at the optimization trajectory (Fig. 7). We can observe that the optimizer actually moves away from a relatively better location in terms of its objective function indicated by the colored contours. (Yellow is less costly than green). This is not unusual in constraint optimization since the optimizer has to make a compromise to satisfy the constraints. Our hypothesis is that the suspected constraint forces the optimizer to leave this "cozy" place.

To test this hypothesis, we enable the display of constraint boundaries in this view (T4). We also make sure that the projection plane is representative of the slowly progressing part on the trajectory by selecting three points at steps 100, 600, and the final solution (Fig. 7). This orientates the plane to coincide with these three points so that we get a reasonable plane for our landscape. We can observe from this view that the trajectory moves in a parabolic shape and is actually enclosed by boundaries that have a similar shape. The areas shaded in gray correspond to infeasible regions of the suspected constraints for a certain feasibility threshold that we increased from one to six to be able to see a region at this large scale. Our interpretation of this is that the optimizer is trying to walk as closely as possible to the hyper surfaces defined by the constraint equations. The reason it keeps on walking is that it has not yet reached a location where it is close enough to all surfaces to be considered a feasible solution.

Considering the progression speed again and also no line search behavior for the majority of the trajectory, taking so many optimization steps still seems unreasonable. A sharper adaptive increase in step size was tested afterward and led to a shortened time to convergence.

Since a single analysis scenario can only show some of the various issues that can occur in optimizations, we like to point the reader to our supplemental video for further impressions. In the video, we provide two other analysis scenarios while also demonstrating the system's interactive features.

6 Plane orientations for loss landscape

In this section, we will discuss and evaluate different strategies for choosing orientations of the plane slice for the loss landscape visualization. In general, there are infinite possibilities for orienting a 2D plane in high-dimensional optimization space. However, only a few strategies for orientation seem valid if we take the tasks into account that correspond to the visualization, i.e., examining the optimization trajectory and the surrounding landscape. Li et al. (2018), for example, used random directions to examine the general loss landscape of neural nets. They also proposed using PCA of trajectory points for obtaining a reasonable plane

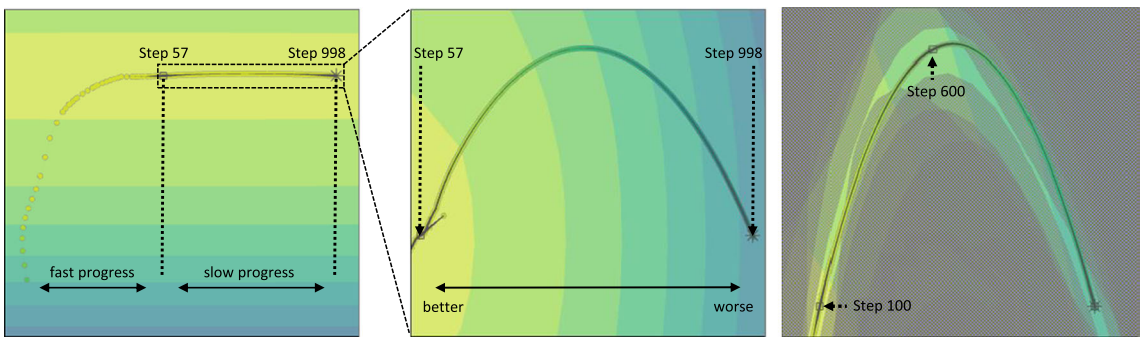


Fig. 7 Optimization landscape view examined in the last part of our case study. We zoom-in on the part of the optimization trajectory corresponding to slow progression we identified earlier (left). We can observe how the optimizer is climbing uphill in terms of the objective function (middle). When displaying the constraint boundaries, we see that the trajectory is moving in a parabola-shaped valley defined by the constraints (right)

for including the trajectory in the loss landscape. Since we also would like to show the trajectory and follow it through high-dimensional space, this strategy is our baseline. We employ a loss landscape in an interactive environment though, giving room for quite some improvement.

6.1 Plane orientation strategies

Let us first state two important properties of the slice plane that are desirable for the visual analysis of optimizer behavior and examination of the optimization space. First, we want the trajectory to coincide with the plane or to be close to it. This is important so that the landscape corresponds to the trajectory points. Second, steps on the trajectory should be well represented in the projection onto the plane, i.e., optimizer movement in nullspace of projection is minimal so that the visualization is not misleading (e.g., showing small steps when there are large jumps orthogonal to the plane).

Due to interaction, we are not limited to a static view of the landscape but can support changing orientations. When cycling through the location on the trajectory of the optimization, we move the origin of the plane to the respective location and can reorient it automatically to satisfy our aforementioned properties. In the following, we will discuss several strategies for this.

Global PCA This strategy serves as our baseline method and uses a static plane orientation for every trajectory point, where the directions of the plane are the first two principal vectors of a PCA on the trajectory points (Li et al. 2018). This strategy is prone to emphasizing early and more spaced out locations of the trajectory, leading to a poor representation of later optimization steps. We propose using weighted PCA to mitigate this: with inverse step size weighting, where each trajectory point s_i is weighted by $\omega_i = ||s_i - s_{i+1}||_2^{-1}$.

argmin Direction + PCA In this strategy, one of the vectors spanning the plane points from initialization to the location of the problem's minimum (i.e., the last location of the trajectory) $v_1 = x^* - x_{\text{init}}$, which is also used by Goodfellow and Vinyals (2015). The second vector is chosen as a linear combination of first and second principal vector of a global PCA of the trajectory that is perpendicular to the argmin direction.

A dynamic variant of this uses the currently selected trajectory point instead of the initialization. This has the advantage that the current location and the minimizer coincide with the plane. In practice, we found that the argmin direction represents the optimization steps better than the principal directions.

Local PCA Trying to improve on the global PCA strategy for individual trajectory points, we tested a localized PCA approach with decaying weights for sequentially distant points. The weights were chosen as $\omega_i = \exp(-(j-i)^2/\sigma^2)$ where s_j is the current trajectory point and σ determines the desired level of locality. However, we could not confirm the expected improvement with this strategy in our evaluation.

Prominent Directions For extracting the most meaningful directions among the optimization steps, we developed an algorithm that determines the most prominent directions from a set of unit vectors δ_i. The idea is to find a linear combination of all vectors in the set $v_1 = \sum_i a_i \delta_i$, so that the resulting direction v_1 represents the majority of them well. We chose the dot product as a measure of degree of representation and choose the weights a_i according to the similarity of δ_i to the other vectors, resulting in $a_i = \sum_j \langle \delta_i,\ \delta_j \rangle$. For calculating the second most prominent direction, we remove the first prominent direction from all vectors of the set $\delta_i^{(2)} = \delta_i - v_1' \cdot \langle \delta_i,\ v_1' \rangle$ with $v_1' = v_1/||v_1||$ and repeat the calculation. A localization as used in local PCA strategy can be introduced easily with $a_i = \omega_i \sum_j \langle \delta_i,\ \delta_j \rangle$.

Using the first and second most prominent directions from the set of length normalized optimization steps shows very good results in our evaluation.

argmin Direction + Prominent Directions Analogously to the argmin direction + PCA strategy, we combine the argmin direction with the aforementioned prominent directions to span the plane.

6.2 Evaluation of strategies

To evaluate the orientation strategies, we test them on three different optimization runs of real motion planning problems: one object pickup problem, as well as a converging and nonconverging optimization of an object relocation problem. We measure how well the optimization trajectories are represented across all optimizer locations. Recalling that we want to minimize invisible optimizer movement in the nullspace of the projection onto the slice plane, we determine how much of each individual step direction is preserved in the projection. For every plane orientation (v_1, v_2), we calculate the length of the projection of each step direction $d_i' = d_i/||d_i||$ with $d_i = (s_{i+1} - s_i)$ on the trajectory. Using a heatmap, we can examine which steps of the trajectory are represented well. Given a plane orientation $(v_1, v_2)^{(j)}$ for step j (plane orientations may

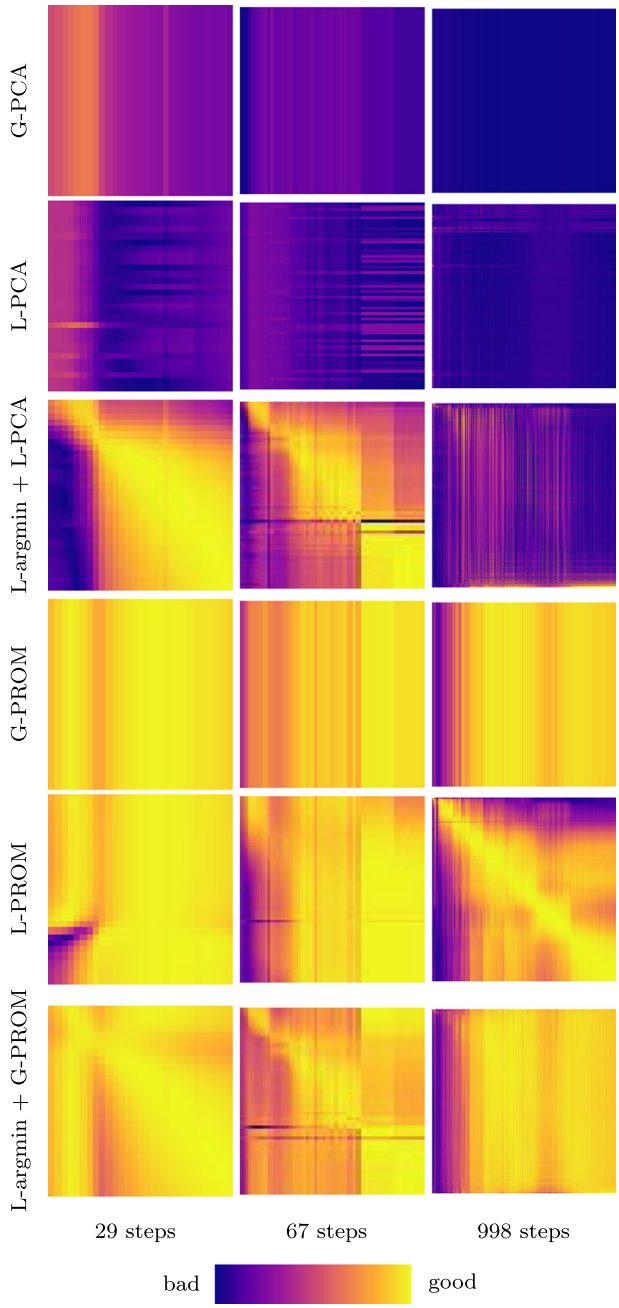


Fig. 8 Heatmaps for different plane orientation strategies. Each row corresponds to a strategy (L = local, G = global, PROM = prominent directions). Columns are different optimization runs with increasing numbers of optimization steps. Each row r in a heatmap shows how well the step directions of the optimization trajectory are represented in the projection onto the plane of trajectory point s_r

vary with examined optimization step), we compute for every step direction d'_i of the trajectory how well it is represented when projecting it onto the plane. Then, each cell of the heatmap represents a $\text{score}_{i,j} = ||v_1 \cdot \langle v_1,\ d'_i \rangle + v_2 \cdot \langle v_2,\ d'_i \rangle||$, where a line shows the scores of all trajectory directions for a fixed plane orientation. Figure 8 shows heatmaps for different strategies on optimization runs with increasing numbers of optimization steps are shown.

For the global PCA strategy (G-PCA), we can observe that it tends to represent late steps badly. The trajectory directions of the long lasting optimization are not represented well at all. The local PCA strategy (L-PCA) performs inconsistently across the individual orientations, which can be seen from the alternating

row pattern. For a local strategy a more diagonal pattern would be expected, as is the case for the local argmin direction + PCA strategy. The strategies involving prominent directions provide better plane orientations in all three optimization runs of our evaluation. For a more thorough assessment of quality consistency, a large amount of test cases will have to be considered in future evaluations.

Apart from the quantification of plane orientation goodness, it is also interesting to notice the block patterns in the heatmaps involving local strategies. These hint at different stages of the respective optimization where the optimizer obviously moves in different subspaces. This kind of insight could also be valuable to our experts and we plan to look into the usefulness of these plots as part of our system in the future. An in-depth evaluation of these strategies with respect to loss landscape visualization for general optimization problems will be left for future work.

7 Discussion

In this section, we reflect on the project and report on the lessons learned during our research. NLP for motion planning is quite complex and comes with many quantities to visualize, such as objective costs, constraints, dual variables, gradient forces, and hypersurfaces—just to name a few. On top of that, it comes in high-dimensional flavors and two notions of time (optimization time and robot time). We chose to focus on the evolutionary aspect, in terms of optimization time, to be able to follow the algorithm along its process. This choice turned out to work well for the assessment of long lasting optimizations and causes for that (Q1).

The use of line charts to examine convergence behavior of constraints or loss is a widely used practice in optimization. They serve well as an entry point to optimization assessment, but experts want to be able to reason beyond the scope of such charts in order to understand what is actually happening. Leveraging linked views to allow for a more thorough exploration of the process worked well in our case.

During the course of this design study, we noticed that slow progression can be connected to ill conditioning and that long lasting optimizations often fail to produce feasible results. To assess optimizations with infeasible solutions (Q2), we also wanted to be able to reason about the behavior of the algorithm, analyze why it goes which way, and ideally see when it takes a “wrong” turn. Through the loss landscape visualization, we were able to provide means to achieve this. However, this technique is most effective if the optimizer only moves in a low-dimensional subspace of the complete optimization space. This is not always the case and plane orientation becomes the deciding factor to effectively use this view.

To understand abstract high-dimensional points of optimization space, we chose a generic solution to display such time-involving intermediate solutions (motion paths in our case) by means of time curves. This works to some degree, however, experts would value a more concrete representation for their domain specific application, e.g., an animation of the robot performing the motion to better connect the abstract and physical world. We expect that integrating the animation capabilities of the motion planning framework into our system will greatly improve optimization space exploration in future work.

8 Conclusion

We presented a visual analytics system for the assessment of optimization processes of nonlinear constraint problems for robot motion planning. Except for the robot path evolution view, displaying the evolution of the motion path, the system provides domain-agnostic views for the analysis of nonlinear programs. Leveraging a loss landscape visualization technique, the system allows users to have a look at the optimization trajectory and surrounding constraints even for the very high-dimensional problems we are facing in the domain. However, we found that the choice of projection plane is crucial for this technique to be effective. To this end, we tested and evaluated different plane orientation strategies for automatically choosing appropriate directions. More interactive ways to alter these planes meaningfully could be explored in future research.

The system’s capabilities were showcased in a case study where we analyzed optimization runs of our collaborators’ NLPs. Together with domain experts, we could confirm that we are able to gain new insights into optimizer behavior, detect flaws in the optimization process, and come up with issue resolving strategies by exploration and analysis with our system. Since most views are applicable to any nonlinear program, we

think that our approach to nonlinear constraint optimization visualization will generalize to other problems. In future work, we plan to investigate respective applications and evaluate our approach in more depth.

Acknowledgements The research has been supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy—EXC 2120/1—390831618, and the DAAD.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

Aigner W, Miksch S, Schumann H, Tominski C (2011) Visualization of time-oriented data. Springer, Berlin
Andrews DF (1972) Plots of high-dimensional data. Biometrics 18(1):125–136
Androulakis G, Vrahatis M (1996) OPTAC: a portable software package for analyzing and comparing optimization methods by visualization. J Comput Appl Math 72(1):41–62
Bach B, Dragicevic P, Archambault D, Hurter C, Carpendale S (2014) A review of temporal data visualizations based on space-time cube operations. EuroVis-STARs, pp 23–41
Bach B, Henry-Riche N, Dwyer T, Madhyastha T, Fekete JD, Grabowski T (2015) Small multiPiles: piling time to explore temporal patterns in dynamic networks. Comput Graph Forum 34(3):31–40
Bach B, Shi C, Heulot N, Madhyastha T, Grabowski T, Dragicevic P (2016) Time curves: folding time to visualize patterns of temporal evolution in data. IEEE Trans Vis Comput Graph 22(1):559–568
Carro M, Hermenegildo M (2000a) Tools for constraint visualisation: the VIFID/TRIFID tool. In: Deransart P, Hermenegildo MV, Małuszynski J (eds) Analysis and visualization tools for constraint programming. Springer, Berlin, pp 253–272. https://doi.org/10.1007/10722311_11
Carro M, Hermenegildo M (2000b) Tools for search-tree visualisation: the apt tool. In: Deransart P, Hermenegildo MV, Mauszynski J (eds) Analysis and visualization tools for constraint programming. Springer, Berlin, pp 237–252. https://doi.org/10.1007/10722311_10
Charalambos JP, Izquierdo E (2001) Linear programming concept visualization. IEEE, pp 529–535. https://doi.org/10.1109/IV.2001.942107
Chatterjee A, Das P, Bhattacharya S (1993) Visualization in linear programming using parallel coordinates. Pattern Recogn 26(11):1725–1736
Chen PC, Hwang YK (1998) Sandros: a dynamic graph search algorithm for motion planning. IEEE Trans Robot Autom 14(3):390–403
Chernoff H (1973) The use of faces to represent points in k-dimensional space graphically. J Am Stat Assoc 68(342):361–368
Chinneck JW (2006) Practical optimization: a gentle introduction. Systems and Computer Engineering. Carleton University, Ottawa
Ghoniem M, Jussien N, Fekete JD (2004) Visexp: visualizing constraint solver dynamics using explanations. AAAI Press, pp 263–268
Ghoniem M, Cambazard H, Fekete JD, Jussien N (2005) Peeking in solver strategies using explanations visualization of dynamic graphs for constraint programming. Association for Computing Machinery, pp 27–36. https://doi.org/10.1145/1056018.1056022
Goodfellow IJ, Vinyals O (2015) Qualitatively characterizing neural network optimization problems. In: Bengio Y, LeCun Y (eds) 3rd International conference on learning representations, ICLR, San Diego, CA, USA, 7–9 May 2015. arxiv:1412.6544
Goodwin S, Mears C, Dwyer T, de la Banda MG, Tack G, Wallace M (2017) What do constraint programming users want to see? Exploring the role of visualisation in profiling of models and search. IEEE Trans Vis Comput Graph 23(1):281–290
Gruendl H, Riehmann P, Pausch Y, Froehlich B (2016) Time-series plots integrated in parallel-coordinates displays. Comput Graph Forum 35(3):321–330
Hägele D, Abdelaal M, Oguz OS, Toussaint M, Weiskopf D (2020) Visualization of nonlinear programming for robot motion planning. VINCI '20. https://doi.org/10.1145/3430036.3430050
Heipcke S (1999) Comparing constraint programming and mathematical programming approaches to discrete optimisation—the change problem. J Oper Res Soci 50(6):581–595
Inselberg A (1985) The plane with parallel coordinates. Vis Comput 1(2):69–91
Jäckle D, Fischer F, Schreck T, Keim DA (2016) Temporal MDS plots for analysis of multivariate data. IEEE Trans Vis Comput Graph 22(1):141–150
Kandogan E (2000) Star coordinates: a multi-dimensional visualization technique with uniform treatment of dimensions. IEEE, pp 9–12

Kavraki L, Latombe J (1994) Randomized preprocessing of configuration for fast path planning. In: Proceedings of the IEEE international conference on robotics and automation, vol 3, pp 2138–2145
Kavraki LE, Svestka P, Latombe J, Overmars MH (1996) Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Trans Robot Autom 12(4):566–580
Kondo K (1991) Motion planning with six degrees of freedom by multistrategic bidirectional heuristic free-space enumeration. IEEE Trans Robot Autom 7(3):267–277
Kuhn HW, Tucker AW (2014) Nonlinear programming. In: Giorgi G, Kjeldsen TH (eds) Traces and emergence of nonlinear programming. Springer, Basel, pp 247–258
Latombe JC (2012) Robot motion planning, vol 124. Springer, Berlin
LaValle SM (2006) Planning algorithms. Cambridge University Press, Cambridge
Li H, Xu Z, Taylor G, Studer C, Goldstein T (2018) Visualizing the loss landscape of neural nets. Curran Associates Inc., New York, pp 6391–6401
Liu S, Maljovec D, Wang B, Bremer P, Pascucci V (2017) Visualizing high-dimensional data: advances in the past decade. IEEE Trans Visu Comput Graph 23(3):1249–1268
Messac A, Chen X (2000) Visualizing the optimization process in real-time using physical programming. Eng Optim 32(6):721–747
Noirhomme-Fraiture M (2002) Visualization of large data sets: the zoom star solution. Int Electron J Symb Data Anal 26–39
Nonato L, Aupetit A (2018) Multidimensional projection for visual analytics: linking techniques with distortions, tasks, and layout enrichment. IEEE Trans Vis Comput Graph 25(8):2650–2673
Pu P, Lalanne D (2000) Interactive problem solving via algorithm visualization. IEEE, pp 145–153. https://doi.org/10.1109/INFVIS.2000.885103
Sedlmair M, Meyer M, Munzner T (2012) Design study methodology: reflections from the trenches and the stacks. IEEE Trans Visu Comput Graph 18(12):2431–2440
Shishmarev M, Mears C, Tack G, De La Banda MG (2016) Visual search tree profiling. Constraints 21(1):77–94
Simonis H, Davern P, Feldman J, Mehta D, Quesada L, Carlsson M (2010) A generic visualization platform for CP. In: Cohen D (ed) Principles and Practice of Constraint Programming—CP 2010, Springer, Berlin, Heidelberg, pp 460–474
Tominski C, Abello J, Schumann H (2004) Axes-based visualizations with radial layouts. In: Proceedings of the 2004 ACM symposium on applied computing, SAC '04. Association for Computing Machinery, New York, NY, USA, pp 1242–1247. https://doi.org/10.1145/967900.968153
Torsney-Weir T, Möller T, Sedlmair M, Kirby RM (2018) Hypersliceplorer: interactive visualization of shapes in multiple dimensions. Comput Graph Forum 37(3):229–240
Toussaint M (2014) Newton methods for k-order Markov constrained motion problems. CoRR abs/1407.0414
Toussaint M (2015) Logic-geometric programming: an optimization-based approach to combined task and motion planning. AAAI Press, Santa Clara, pp 1930–1936
Tušar T, Filipič B (2014) Visualization of Pareto front approximations in evolutionary multiobjective optimization: a critical review and the prosection method. IEEE Trans Evol Comput 19(2):225–245
van den Elzen S, Holten D, Blaas J, van Wijk JJ (2016) Reducing snapshots to points: a visual analytics approach to dynamic network exploration. IEEE Trans Vis Comput Graph 22(1):1–10
Wegenkittl R, Löffelmann H, Gröller E (1997) Visualizing the behaviour of higher dimensional dynamical systems. In: Proceedings visualization '97 (Cat. No. 97CB36155), pp 119–125. https://doi.org/10.1109/VISUAL.1997.663867
Wolfe P (1969) Convergence conditions for ascent methods. SIAM Rev 11(2):226–235
Wong PC, Bergeron RD (1994) 30 years of multidimensional multivariate visualization. In: Nielson GM, Hagen H, Müller H (eds) Scientific visualization, overviews, methodologies, and techniques. IEEE Computer Society, New York, pp 3–33