Proceedings of the 1st International Workshop on Robot Learning and Planning (RLP 2016)
in conjunction with 2016 Robotics: Science and Systems
June 18, 2016 – Ann Arbor, Michigan, USA

Learning the Problem-Optimum Map: Analysis and Application to Global Optimization in Robotics

Kris Hauser
Department of Electrical and Computer Engineering
Department of Mechanical Engineering and Materials Science
Duke University
Durham, NC, USA
Email: kris.hauser@duke.edu

***Abstract*—This paper describes a data-driven framework for approximate global optimization in which precomputed solutions to a sample of problems are retrieved and adapted during online use to solve novel problems. This approach has promise for real-time applications in robotics, since it can produce near-globally optimal solutions orders of magnitude faster than standard methods. This paper establishes theoretical conditions on how many and where samples are needed over the space of problems to achieve a given approximation quality. The framework is applied to solve globally optimal collision-free inverse kinematics (IK) problems, wherein large solution databases are used to produce near-optimal solutions in sub-millisecond time on a standard PC.**

I. Introduction

Real-time optimization has been a longstanding challenge for robotics research due to the need for robots to react to changing environments, to respond naturally to humans, and to interact fluidly with other agents. Applications in robotics are pervasive, including autonomous vehicle trajectory optimization, grasp optimization for mobile manipulators, footstep planning for legged robots, and generating safe motions in proximity to humans. Global optimality has long been sought, but is generally computationally complex in the high-dimensional, nonconvex problems typical in robotics. As a result, most researchers resort to local optimization or heuristic approaches, which have no performance guarantees.

A promising approach is to integrate precomputed data (i.e., experience) into optimization to reduce online computation times [1, 2, 6, 14, 17, 19, 20, 11]. This idea is attractive because humans spend little time deliberating when they have seen a similar problem as one solved before, and hence robots may also benefit by learning from experience. But because experience is time consuming to generate, it is important to pose the question, *how much data is needed to learn robot optimal control tasks*?

This paper formalizes this question in a general context of global nonlinear optimization with nonlinear constraints. Consider a structure where problems are drawn from some family of related problems, such that problems can be parameterized by a set of continuous *P-parameters* that modify a problem's constraints and objective function. These parameters are not themselves decision variables, but rather they alter the constraints of the problem and hence affect the value of the optimal solution (Fig. 1). For example, in inverse kinematics, the P-parameter space is identical to the notion of task space. For a robot manipulating objects on a table, the P-parameters may be the source and target pose of the object. In grasp optimization, P-parameters specify some parametric representation of object geometry. In other words, they give a deterministic "feature vector" of the type of optimization problems we would like to learn. We can then speak of learning a *problem-optimum map* (note that some problems may have multiple optima, so a map may be nonunique).

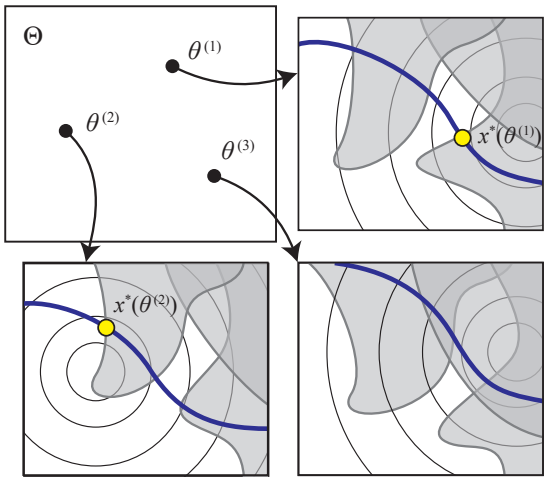


Fig. 1. Illustrating the problem space concept. Instances $\theta^{(1)}$, $\theta^{(2)}$, and $\theta^{(3)}$ drawn from problem space Θ each correspond to different optimization problems. Each problem has an objective function (level sets drawn as circles), inequality constraints (shaded region), and equality constraints (thick curve) dependent on the choice of θ. Each problem $\theta \in \Theta$ has a set of optima $x^\star(\theta)$, which may be empty if the problem has no feasible solution (e.g., $\theta^{(3)}$).

We present a quite general Learning Global Optima (LGO) framework in which a learner is given a database of optimal solutions, called *examples*, that are generated for problems sampled from a given problem distribution, assumed to be representative of problems faced in practice. Since this step is offline, brute force methods can be used to obtain global optima or near-optima with high probability. In the query phase, a solution to a previous problem is adapted to satisfy the constraints of a novel problem, typically via local optimization.

The learner's performance relies on whether *solution similarity* holds across problem space: the optimal solution to a nearby example problem is likely to be close to the global optimum of the query problem, and hence using it to seed local optimization is likely to yield global solution. In an accompanying more detailed paper [10] we show that indeed it is possible to generate a finite database of examples to cover problem space "sufficiently well" to achieve low suboptimality in the query phase. However, the analysis is a mixture of positive and negative results. First, the number of examples needed scales exponentially in problem space dimension. This indicates that learning requires vast experience in high-dimensional problem spaces, unless further strong assumptions are made. Second, the problem-optimum map is in fact only piecewise-continuous, which is challenging to learn for parametric function approximators like neural networks or Gaussian process models [19]. This justifies the use of a k-nearest-neighbor approach that is better suited to capturing these discontinuities [11].

An implementation of the LGO framework is demonstrated on the decades-old — but still surprisingly challenging — problem of inverse kinematics (IK). An general solution would be able to calculate optimal solutions, handle redundant robots, gracefully handle infeasible problems, and incorporate collision avoidance, all done in real-time. Yet current approaches fall short of addressing all of these challenges. Our proposed implementation addresses optimality, redundancy, infeasibility, and collision avoidance in a unified manner. It works with general articulated robots and produces near-optimal, collision-free IK solutions typically in less than one millisecond on a standard PC. It can also predict infeasible queries with high confidence (over 98%), and is amenable to a "lifelong learning" concept that, given a handful of seed queries, automatically populates the IK database in the background. As a result, the solver progressively solves IK queries faster and more reliably as more time is spent using it. This IKDB package is made publicly available at http://motion.pratt.duke.edu/ikdb/.

II. Related work

The results of this work are relevant to the significant body of related literature in robot control learning. Reinforcement learning proposes that robots record their experience interacting with the physical world and then progressively optimize their behavior to improve performance. Other researchers study the approach of learning from demonstration, which asks human teachers to provide robots with instances of "good" motions. In either case, it is quite time consuming to provide physical robots with experience or demonstrations from human teachers, making the question of "how much experience is needed?" of the utmost importance.

Despite decades of research, it is still challenging to quickly compute high quality solutions for high dimensional nonlinear optimization problems. A promising approach is to "reuse" previous solutions to solve similar problems, in which reuse is applied in a variety of forms. Machine learning approaches are popular, and techniques have been used to predict the success rate of initial solutions or trajectories for optimization [3, 19]. Other authors have used databases of grasps to generate grasps for novel objects [2, 6]. Another form of reuse is compositional, in which small solutions are composed to solve larger problems. For example, footstep-based legged locomotion planners often store a small set of optimized motion primitives and replay them to execute footsteps [4, 16]. Rather than directly replaying motions, some other methods apply more sophistication adaptation methods to increase the range of problems to which a given example can be adapted [9]. Experience has also been studied for avoiding recomputation of roadmap structures when obstacles change [17, 20].

The instance-based framework used in this paper is arguably most closely related to the work of Jetchev and Toussaint [14], which select past plans to initialize trajectory optimization. Similarly, this paper uses experience to initialize optimization, but considers the more general context of nonlinear optimization problems, and focuses on the conditions under which such databases provide theoretical guarantees on solution quality. Furthermore, the prior work uses locally optimal examples in the database, whereas our work uses globally optimal ones.

Our automated IK learning framework bears some similarity to the Lightning framework of Berenson et al [1] in which past plans are reused in parallel with planning from scratch. In that work, past examples are retrieved based on nearest feature vector containing the start and end positions of the planning query, whereas our work is more general and accepts any P-parameters of the class of optimization problems. Furthermore, Lightning uses past planning queries to generate new experience, while our framework uses a handful of user-driven queries to seed the database, but can then populate the database automatically thereafter.

In the context of inverse kinematics, researchers from computer animation have used learning from human motion capture data [8, 12]. In robotics, several authors have used machine learning methods to learn local inverse kinematics solutions for manipulator control in Cartesian space [21, 22]. Some researchers are also able to incorporate obstacles [18, 23]. By contrast, our work considers the global IK problem, which is more appropriate for determining goal configurations for motion and grasp planning. Perhaps the closest attempt to ours to address this problem is an approach that tries to discover structure in the IK problem space via clustering [5]. We explore more fully some of the preliminary ideas in that work.

III. Learning Global Optima framework

Let us summarize the intuition behind our main results, which are covered in the auxiliary document. Consider a family of decision problems whose constraints and objectives themselves vary according to external parameters, such as the initial state of a trajectory, obstacle positions, target states, etc. Since these parameters are not under the direct control of the robot, they are not *decision parameters* (denoted in this paper as x) under the control of the agent, but rather *P-parameters*

(denoted in this paper as θ). Since the optimal solution $x^\star$ depends on the current values of the external parameters, such as a changing environmental obstacle causing the robot to take a different path, there is a relationship between P-parameters and optimal solutions. In essence, a P-parameter vector θ can be thought of *producing* an optimization problem that the robot must then solve to generate $x^\star$.

In the context of robotics, P-parameters can include the robot's initial state, the query (e.g., a target location), and characteristics of the environment (e.g., the positions of obstacles). The problem's decision variables may be diverse, such as a robot configuration, a representation of a trajectory to be executed, or the parameters of a feedback control policy.

We intend to study the relationship between θ and $x^\star$ across the space of P-parameters, which is a set we call *problem space*. The concept of problem space is related to the notion of *task space* in IK or operational space control, and *feature space* in machine learning. A problem space gives the range of possible problems encountered in practice, and we are interested in performing well across this entire space. (In practice, it may be more appropriate to speak of a distribution over problems, but in this case we consider the range of likely problems.)

A. Illustrative examples

As an example, let us consider a family of IK problems. *Note that for the next two paragraphs we shall adopt traditional IK notation, which conflicts with the notation used elsewhere throughout this paper.* Suppose a robot is asked to reach the IK objective $x_d = x(q)$, where $x(q)$ is the forward dynamics mapping from configurations q to tasks x. Here, we have q being the decision variable which the IK solver is meant to solve, and $x_d - x(q) = 0$ the constraint. If we are now to think of the robot being asked to reach a range of IK objectives, we now have $\theta \equiv x_d$ being the P-parameter of this problem. If we are now to let x_d range over the space of all possible or all likely tasks, this range of constraints gives the problem space Θ.

If we now have the robot required to simultaneously reach x_d while avoiding an obstacle at position p. We can encode this constraint by a distance inequality $d(q, p) \geq 0$ that measures how far the robot is at configuration q from the object at position p. Now, if p were to also vary, we should add it to the P-parameter vector to obtain $\theta = (x_d, p)$. If, on the other hand, only the z-coordinate of p were to vary, we should only set $\theta = (x_d, p_z)$, treating the x- and y-coordinates of p as constants in the constraint $d(q, p) \geq 0$.

B. Mathematical statement

In the proposed framework, we are given a class of optimization problems over an n-dimensional decision parameter x:

$$\begin{gathered} \text{Given functions } f,\ g, \text{ and } h \\ \text{Find } x \in \mathbb{R}^n \text{ s.t.} \\ f(x) \text{ is minimized,} \\ g(x) \leq 0, \text{ and} \\ h(x) = 0. \end{gathered} \tag{1}$$

Here, the functions are defined with ranges $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^m$, and $h : \mathbb{R}^n \to \mathbb{R}^p$, and the equality and inequality are taken element-wise. The functions f, g, and h are in general nonconvex. However, the feasible set and the objective function are required to be bounded. The goal of global optimization is to produce an optimal solution x if one exists, or to report infeasibility if none exists.

A *problem space* is defined by variations of f, g, and h. In this paper, variations in problems will be encapsulated in the form of a P-parameter variable θ. Specifically, the functions f, g, and h will be themselves function of the parameter $\theta \in \Theta$, where $\Theta \subseteq \mathbb{R}^r$ is known as the problem space. To make the dependence on θ explicit, we write $f(x) \equiv f(x, \theta)$, $g(x) \equiv g(x, \theta)$, and $h(x) \equiv h(x, \theta)$. (Here it is important that f, g, and h are fixed functions of both x and θ and not any external factors — in other words, θ must capture all variability between problems.) We can now rewrite (1) as:

$$\begin{gathered} \text{Given P-parameter } \theta \in \Theta, \\ \text{Find } x \in \mathbb{R}^n \text{ s.t.} \\ f(x, \theta) \text{ is minimized over } x, \\ g(x, \theta) \leq 0, \text{ and} \\ h(x, \theta) = 0. \end{gathered} \tag{2}$$

With this definition, it is apparent that the only external parameter affecting the solution is the P-parameter θ. It is now possible to speak of the set of optimal solutions $x^\star(\theta)$, which is a deterministic function of θ. $x^\star(\theta)$ may be empty if there is no solution. If it is nonempty, an optimal cost $f^\star(\theta)$ exists. We refer to the problem of computing one such solution, or determining that none exists, as the problem $P(\theta)$.

As a final note, in most of this paper, we shall treat m, n, p, and r as fixed. In practice, however, it may be possible to share examples across problems of differing dimension. For example, solutions to motion planning problems may be represented as a variable number of waypoints, with a set of decision variables and constraints for each waypoint. The algorithm presented below still applies, but the analysis would require additional assumptions about problem structure.

C. Database computation phase

We assume the learner has access to compute a database of problem-solution pairs, each of which is known as an *example*. Specifically, the database consists of $D = \{(\theta^{(i)}, x^{(i)}) \mid i = 1, \ldots, N\}$ where $\theta^{(i)}$ is a P-parameter sampled from Θ, and $x^{(i)}$ is an optimal solution in $x^\star(\theta^{(i)})$. Later we will use *nil* to mark that no solution was found, but for now let us assume a database entirely consisting of successfully solved examples. The distribution of $\theta^{(i)}$ should

be relatively diverse, and should approximate the distribution of problems expected to be encountered in practice. The solutions are computed by some global optimization method during an extensive precomputation phase.

In practice, for generating example solutions, we resort to the use of metaheuristic global optimization techniques (e.g., random restarts, simulated annealing, or evolutionary algorithms). However, their approximate nature means that it cannot be guaranteed that a precomputed solution is truly optimal, or that a precomputed failure is truly a failure. As a result the database will contain some noise. We will discuss the effects of noise, and methods for combating it, in subsequent sections.

D. Query phase

The query phase attempts to answer a novel query problem specified by P-parameter θ'. We will analyze a learner that has access to two auxiliary functions:

1) Let $S(\theta)$ be a *selection function* that produces a subset of k examples in D for any problem $\theta \in \Theta$. Usually these are assumed to be close in problem space to θ', such as by finding its k-nearest neighbors given a distance metric $d(\theta, \theta')$ in Θ space.
2) Let $\mathcal{A}(x, \theta, \theta')$ be the *adaptation function* that adapts an optimal solution x for problem θ to another problem θ'. The result of $\mathcal{A}$ is either a solution x' or *nil*, which indicates failure.

The learner is assumed to proceed as follows:

- *Retrieval:* Select the problems $\theta^{(i_1)}, \ldots, \theta^{(i_k)} \leftarrow S(\theta')$
- *Adaptation:* For each $j = 1, \ldots, k$, run $\mathcal{A}(x^{(i_j)}, \theta, \theta')$ to locally optimize the solution for the selected problems. Return the best locally optimized solution, or *nil* if none can be found.

It may be possible to learn a different representation of the map from θ to x, e.g. by using neural networks or Gaussian processes. However, k-NN more readily admits more formal analysis, and also is somewhat better at handling discontinuities.

E. Summary of results

In a more complete paper [10] we derive a number of theoretical results about this framework. Here we summarize these results, and illustrate them on the simple example of Fig. 2(a):

- If the mapping from P-parameters to constraints is continuous, then the mapping from θ to optimal solutions $x^\star(\theta)$ is a piecewise continuous mapping (Fig. 2(b)). (Note that it is only a partial mapping if some θ define infeasible optimization problems.)
- $x^\star(\theta)$ is continuous over connected regions of problem space in which the active set is unique and equal (Fig. 2(c)). Moreover, it satisfies a Lipschitz bound if the active set Jacobian is nonsingular.
- Let a *learner* consist of a database of examples and a subroutine for subselecting and adapting examples to a novel problem. We define a notion of "goodness" of a learner as the worst-case suboptimality of the adapted solution to a randomly drawn problem, relative to its true optimum. Optimal goodness is achieved if the learner generates $x^\star(\theta)$ everywhere.
- An idealized learner achieves optimal goodness if: 1) it stores an example in each contiguous region of problem space in which the active set is constant, 2) for a new problem specified by a P-parameter vector θ, it retrieves an example in the same region as θ, and 3) it uses local optimization, seeded from the example's solution and warm-started from its active set (Fig. 2(d)).
- The worst-case number of examples for an ideal learner is the number of possible combinations of active constraints, which is at worst case exponential in problem size.

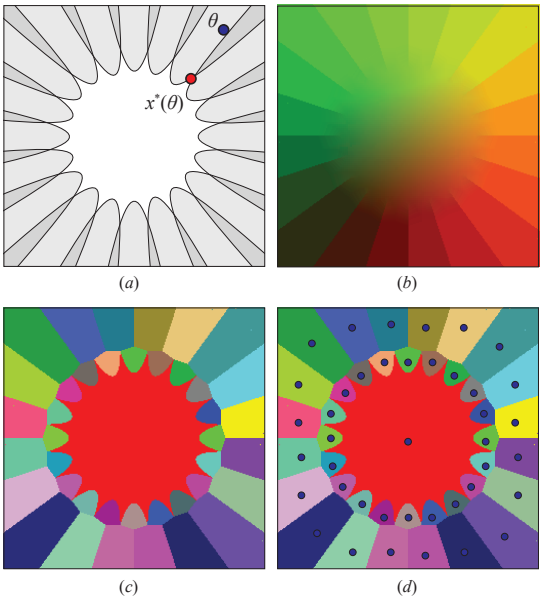


Fig. 2. (a) A simple 2D problem family in which the goal is to minimize the distance between the optimization parameter x and the P-parameter θ subject to concave polynomial constraints. (b) The optimal solution function, with x_1 and x_2 plotted as red and green intensities, respectively. It is piecewise smooth (best viewed in color). (c) The problem space partitioned into regions of unique active set. (d) An ideal learner needs only to store one example in each region (circles). It will globally solve a new problem using local optimization, starting from the example in the same region as that problem.

IV. Query implementation

The basic LGO framework works fairly well as presented in Section III-D. However, several parameters affect query performance in practice:

1) *Problem similarity metric.* Although similar problems have similar optimal solutions, optimal solution quality usually changes anisotropically in problem space. As a result, non-Euclidean distance metrics may be better for retrieving good examples.
2) *Search strategy.* A brute-force NN search has $O(n)$ computational complexity, which becomes slow in large databases. The use of fast NN data structures allows our

method to scale to larger databases. We use a ball tree in our implementation.

3) *Number of neighbors* k affects the robustness of the query phase. Larger values of k help combat the effects of limited database size and noise by allowing additional attempts at solving the problem. This comes at the expense of greater computational cost for challenging queries.
4) *Local Optimization Strategy.* A straightforward approach applies a nonlinear optimizer like sequential quadratic programming (SQP). But we consider faster approximate strategies below.
5) *Perturbations in local optimization.* If feasibility is affected by variables that are not captured in the P-parameters, such as environmental obstacles, it is usually prudent to perturb the retrieved seeds before local optimization. This is also helpful to address non-differentiable inequality constraints.

We present a faster implementation, LGO-quick-query, in which we modify the adaptation step in two ways. First, we sort the $x^{(i)}$'s in order of increasing distance $d(\theta^{(i)}, \theta)$, and stop when the first solution is found. If no solutions are found, failure is returned. This allows the method to scale better to large k, since easy problems will be solved in the first few iterations. Second, rather than using full local optimization, we simply project solutions onto the equality constraints $h(x, \theta) = 0$ and those active inequalities that are met exactly at the prior example. This method is often an order of magnitude faster than SQP, and provides sufficiently near-optimal feasible solutions.

Specifically, given a prior example $(x^{(i)}, \theta^{(i)})$ we detect the set A of constraints in $P(\theta^{(i)})$ active at $x^{(i)}$, and then formulate the active set equality $g_A(x, \theta) = 0$, $h(x, \theta) = 0$ for the new problem. Starting at $x^{(i)}$, we then solve for a root $j(x', \theta') = 0$ via the Newton-Raphson method. Bound constraints, e.g., joint limits, are also efficiently incorporated. Because Newton-Raphson takes steps with least squared norm, it approximately minimizes $\|x - x'\|^2$. Although locally optimizing the objective function f may produce better solutions, the strategy of keeping the solution close to the start will still retain the asymptotic goodness properties of the database.

Pseudocode for this technique is as follows:

Algorithm 1 LGO-quick-query(θ, D)

Find the k-nearest neighbors $\theta^{(i_1)}, \ldots, \theta^{(i_k)}$ in D, sorted in order of increasing $d(\theta^{(i_j)}, \theta)$
for $j = 1, ..., k$ **do**
  $A \leftarrow$ActiveSet$(x^{(i_j)}, \theta^{(i_j)})$
  Simultaneously solve $g_A(x, \theta) = 0$, $h(x, \theta) = 0$ using the Newton-Raphson method, initialized at $x^{(i_j)}$
  if successful **then return** the local optimum x
return nil

A. Handling infeasible problems

When the problem space contains many infeasible problems that may be drawn in practice, queries for infeasible problems are expensive because they always performs k failed local optimizations per query. In some cases it may be preferable to quickly terminate on infeasible problems. LGO can be easily adapted to predict infeasible problems and avoid expending computational effort on them.

We permit the database D to contain infeasible problems, whose solutions are marked as nil. If a large fraction of retrieved solutions for a query problem θ are nil, then it is likely that θ is infeasible as well. More formally, if k' denotes the number of feasible examples out of the retrieved set, we use a confidence score PFeasible(k') that determines how likely the problem is to be feasible given k'. If PFeasible(k') falls below a threshold, then we predict θ as being infeasible and do not expend further effort. Otherwise, it is likely to be feasible but the database does not have sufficient coverage. In this case we fall back to global optimization.

Algorithm 2 LGO-query-infeasible(θ, D)

Find the k-nearest neighbors $\theta^{(i_1)}, \ldots, \theta^{(i_k)}$ in D, sorted in order of increasing $d(\theta^{(i_j)}, \theta)$.
for $j = 1, ..., k$ **do**
  if $x^{(i_j)} \neq nil$ **then**
    Solve $g_A(x, \theta) = 0$, $h(x, \theta) = 0$ starting from $x^{(i_j)}$
    if successful **then return** the local optimum x
$k' \leftarrow |\{j \in \{1, 2, \ldots, k\} \quad | \quad x^{(i_j)} \neq nil\}|$
if PFeasible$(k') > \tau$ **then return** Global-Optimization(θ)
return nil

To generate PFeasible, we use leave-one-out (l.o.o.) cross validation to estimate the empirical probability that the problem is infeasible given that k' out of k nearest neighbors are feasible.

The *feasibility confidence threshold* τ should be chosen to trade off against the competing demands of average query time and incorrect predictions of problem infeasibility. A value $\tau = 1$ will never fall back to global optimization, while $\tau = 0$ always falls back. A high value leads to more consistent running times but slightly lower success rate.

V. Application to Inverse Kinematics

Here we describe an application of our implementation to the classic problem of IK. The resulting solver combines the reliability of global optimization with the speed of local optimization. It improves upon prior techniques by automatically handling optimality criteria, kinematic redundancy, collision avoidance, and prediction of infeasible IK queries.

The solver is highly customizable; it accepts arbitrary robots specified by Universal Robot Description Format (URDF) files, static environment geometries given as CAD models, IK constraints, and additional user-defined feasibility constraints and objective functions. Collision detection is performed using the Proximity Query Package (PQP) [7]. Cost functions

and additional constraints are specified as arbitrary Python functions that accept a configuration and optional additional arguments and return a real number. Each IK problem specifies one or more single-link IK constraints, optional joint limits, an optional array of the movable set, and any additional arguments of custom functions.

A. Problem specification

Specifically, the user provides $s \geq 0$ IK constraints parameterized by $\theta_{IK,1}, \ldots, \theta_{IK,s}$, a cost function $f(q, \theta_f)$, an optional custom inequality $g(q\theta_g) \leq 0$, and joint limits q_{min} and q_{max}. (Note: f and/or g do not need to be parameterized in which case θ_f and/or θ_g are *nil*.) The optimization problem to be solved is:

$$\begin{aligned} &\text{Given } \theta = (\theta_f, \theta_g, \theta_{IK}), \\ &\text{Minimize over } q \ f(q, \theta_f) \text{ s.t.} \\ &g(q, \theta_g) \leq 0 \\ &E_{IK}(q, \theta_{IK}) = 0 \\ &q_{min} \leq q \leq q_{max} \\ &C(q) = \text{ false} \end{aligned} \tag{3}$$

where E_{ik} is the error function for the IK constraints and $C(q)$ returns whether the robot has self-collision or environmental collision at q. C is a nondifferentiable constraint and is implemented during optimization by setting the objective function to ∞ when in collision.

B. Database learning

To generate a database, an axis-aligned range of θ is sampled uniformly at random. To find global optima, a modified random-restart algorithm is used. First, we use a Newton-Raphson iterative IK solver with 100 random initial configurations to find configurations that satisfy IK and joint limit constraints. The IK solution with lowest cost, if one exists, is then used to seed an local SQP-based optimization. The training procedure ranges from approximately 10 ms per example for the robot in Fig. 3 to 100 ms per example for the robot in Fig. 8. Our experiments find that this technique is an order of magnitude faster than naïve random restart local optimization.

To produce a distance metric, we have experimented with both Euclidean distance metric, as well as learned Mahalanobis distances that can correct for poorly scaled problem spaces. Our implementation can be configured to perform online metric learning using the LogDet update method of [13].

C. Experiments

The first experiments consider the effects of database size and selection technique for both redundant and nonredundant IK problems on an industrial robot. The cost function penalizes configurations near joint limits:

$$f(q) = -\sum_{i=1}^{n} \min(q_i - q_{min,i}, q_{max,i} - q_i)^2. \tag{4}$$

For each experiment, we generated a database as described in Sec. V-B, and then independently generated an test set of 1000 problems. We disabled infeasibility prediction (i.e., set $\tau = 0$) for all these experiments.

First we consider a position-constrained problem where the 3 P-parameters of the end-effector position were varied. Fig. 3 compares the performance of the proposed LGO method, with varying numbers of neighbors k and a fixed database of $|D| = 100,000$ examples. LGO is compared against the DIRECT algorithm [15], the metaheuristic global optimization technique differential evolution (DE), and a random-restart method RR(N) with N random restarts. A final "cleanup" local optimization is run to improve the quality of the solution produced by DIRECT and DE. The RR method is implemented as described in Sec. V-B, which is already highly tuned to this IK problem: each restart runs a Newton-Raphson technique to solve for the IK, then if successful, runs SQP.

Clearly, the off-the-shelf global optimizers are not competitive with RR(N). LGO outperforms even RR(1) in speed, and begins to outperform the success rate and solution quality of RR(100) at $|D| = 100,000$ and $k = 10$. Compared to RR(100), LGO is two orders of magnitude faster. Fig. 4 illustrates the learning curves of LGO on this problem.

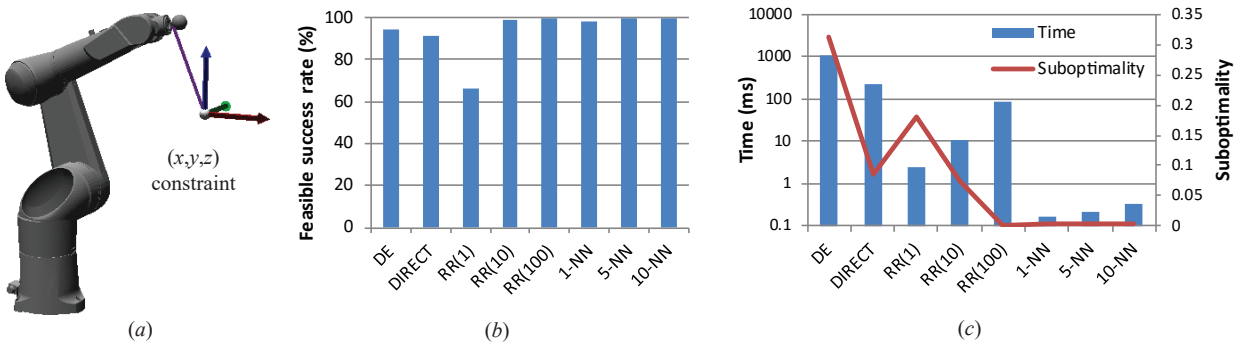
Fig. 5 gives results for a position and orientation-constrained problem. This is a nonredundant problem with a 6-D P-parameter space, and the robot has up to four IK solutions (elbow up/down, wrist up/down). The IK rotation matrix is encoded via a 3D exponential map representation, and for the NN distance metric these P-parameters are scaled by $1/2\pi$. We test LGO with training sets up to $|D| = 1,000,000$ and varying values of k. Again, we see that off-the-shelf methods are not competitive, and LGO is much faster than other techniques while still obtaining close to optimal results.

The learning curves, illustrated in Fig. 6, show that LGO requires more data to get similar success rates to RR(100), only reaching parity at $|D| = 1,000,000$. This result is consistent with the theoretical prediction that more examples are needed in higher dimensional P-parameter spaces to reach a desired level of quality. At this point it is over 10 times faster than RR(100). An unintuitive result is that RR(N) is significantly faster in this case than the redundant case; the rationale is that since the problem is nonredundant, the SQP optimizer quickly terminates because it cannot make progress to improve the objective function.

Another issue is that the problem space includes axes of different units (meters for position vs. radians for orientation). For such poorly-scaled problem spaces, we found that metric learning produced a Mahalanobis distance metric similar to our ad-hoc weighting of $1/2\pi$. Compared to unweighted Euclidean distance, the learned metric produced a consistent, small boost in success rate (Fig. 7). Suboptimality and computation time were not significantly affected.

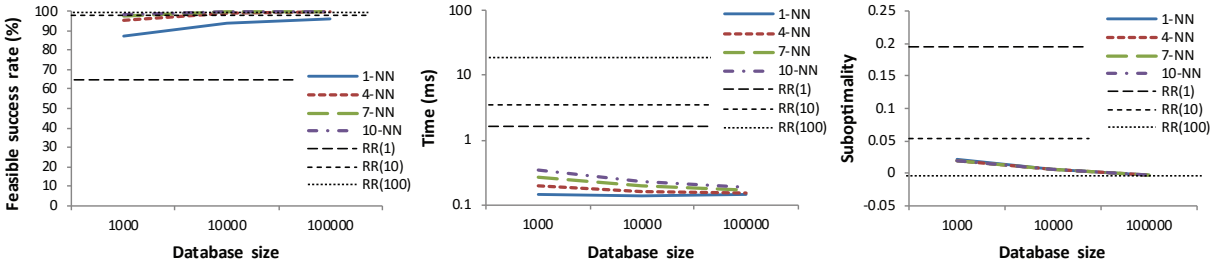
D. Lifelong learning experiment

We also implemented a "lifelong learning" implementation where the system automatically populates the database of



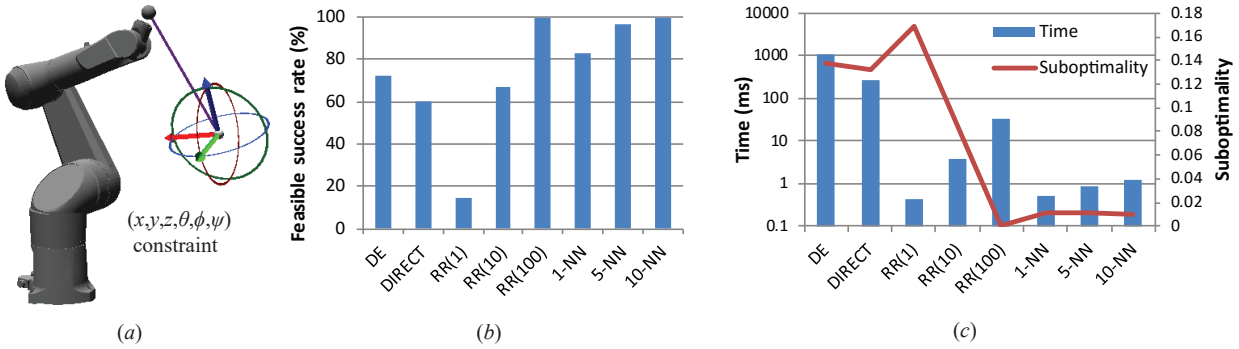
(a) (b) (c)

Fig. 3. (a) A redundant position-constrained IK problem (3 P-parameters) on a 6DOF industrial robot. (b) Comparison of success rate on a test set of 1,000 known feasible problems, between existing global optimization methods DIRECT and differential evolution (DE), the N-random restart method RR(N), and k-NN LGO with 100,000 examples (higher values are better). (c) Comparison on average computation time, in ms, and suboptimality (lower values are better). Time is shown on a log scale.



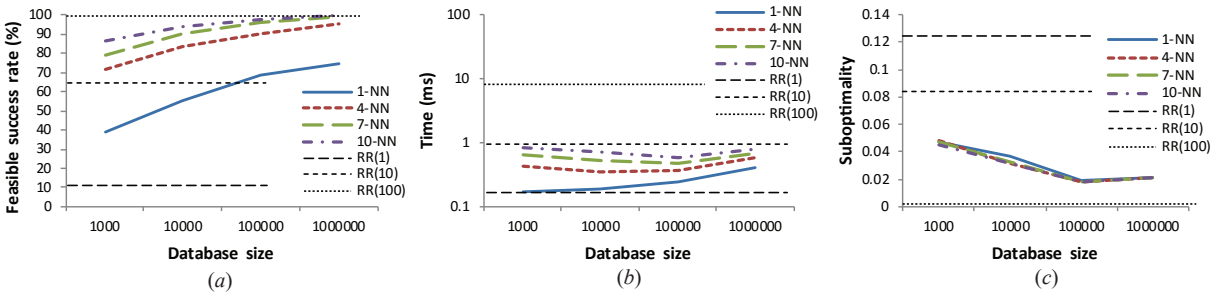
(a) (b) (c)

Fig. 4. Learning curves for LGO on the example of Fig. 3 as the database size and number of neighbors k varies, comparing (a) success rate (b) running time, and (c) suboptimality. For reference, the performance of RR(N) is shown as horizontal lines.



(a) (b) (c)

Fig. 5. (a) A position- and orientation-constrained IK problem (6 P-parameters) on a 6DOF industrial robot. Here LGO is tested with a 1,000,000 example database. (b) Success rate. (c) Running time and suboptimality.



(a) (b) (c)

Fig. 6. Learning curves for LGO on the example of Fig. 5 as the database size and number of neighbors k varies, comparing (a) success rate (b) running time, and (c) suboptimality.

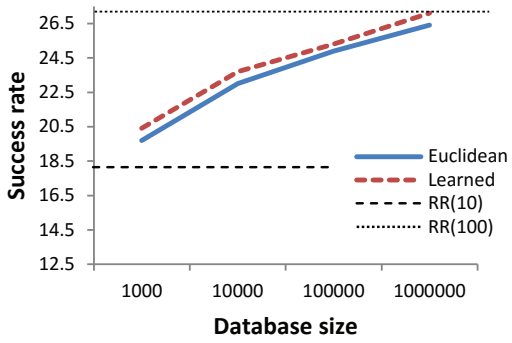


Fig. 7. On the problem of Fig. 5, the use of metric learning produced approximately a 2–3% boost in success rate compared to Euclidean distance, using $k = 4$ neighbors.

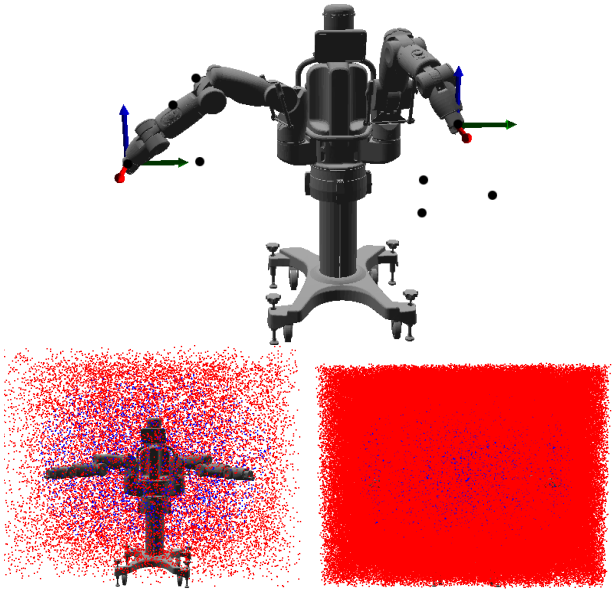


Fig. 8. The automatic database example with a 15-DOF robot and dual-arm position constraints. Four seed IK queries are marked. The IK endpoints of the database after 1h of background computation at 30% CPU usage (approximately 10,000 examples). The IK endpoints after 24h of computation (approximately 250,000 examples).

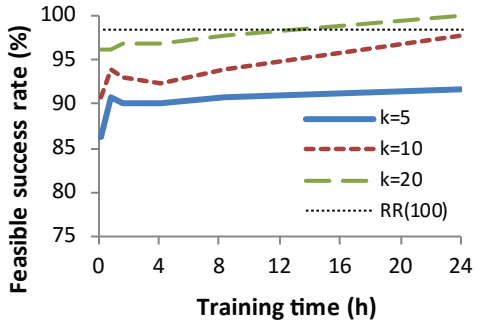


Fig. 9. Learning curves for the automated learning procedure on the example of Fig. 8.

examples in the background, given a handful of seed queries. Fig. 8 shows an example of how this works on a problem with the Rethink Robotics Baxter robot with two position-constrained end effectors, for a 6D P-parameter space. Four seed queries were provided by hand, but no other information was provided to help the system identify a feature representation or feature ranges. It correctly identified the 6 P-parameters, and after 1 hour of background computation, the method generated a distribution of feasible problems shown in Fig. 8, center. The procedure was then left to learn for 24 hours, yielding approximately a quarter million examples. Fig. 9 shows the learning curve for this training process, evaluated on a holdout testing set of 1,000 examples, 130 of which are feasible. The resulting LGO method with $k = 20$ performed more reliably than RR(100), with slightly higher solution quality, and 20 times faster (approximately 5 ms per query compared to 100 ms).

VI. Conclusion

We presented an experience-driven framework for global optimization in families of related problems. Our work is an attempt to answer some fundamental questions relating optimization problem structure to the number of examples needed to attain a given quality. First, we highlight the fact that the problem-optimum map is in general only piecewise continuous, which motivates the use of k-nearest neighbors approaches rather than function approximators in our implementation. Our results suggest that the approach is practically sound in that a finite number of examples is needed to achieve a bounded level of "goodness". However, the required database size depends exponentially on the problem dimensionality in the worst case. Nevertheless, in some domains it is practical to generate databases of millions of examples, and in an inverse kinematics example problem, our implementation yields 1-5 orders of magnitude faster performance than the off-the-shelf global optimization methods, with little sacrifice in quality.

Future work may improve the analysis by uncovering properties of problem spaces that are more easily learned, such as problem families where global optima have larger basins of attraction (e.g., convex optimization problems). It may also be possible to bound the number of possible active sets at global optima based on the order of each constraint, or other properties of the problem. We also have not yet considered the question of practical methods for distributing examples such that each example "covers" a large region of problem space with good adaptations, which would allow us to maximize performance for a given database size.

References

[1] Dmitry Berenson, Pieter Abbeel, and Ken Goldberg. A robot path planning framework that learns from experience. In *IEEE Intl. Conf. on Robotics and Automation*, pages 3671–3678, 2012.

[2] J. Bohg, A. Morales, T. Asfour, and D. Kragic. Data-driven grasp synthesis: A survey. *Robotics, IEEE Trans-*

actions on, 30(2):289–309, April 2014. ISSN 1552-3098. doi: 10.1109/TRO.2013.2289018.

[3] A. Cassioli, D. Di Lorenzo, M. Locatelli, F. Schoen, and M. Sciandrone. Machine learning for global optimization. *Computational Optimization and Applications*, 51(1):279–303, 2010. ISSN 1573-2894. doi: 10.1007/s10589-010-9330-x. URL http://dx.doi.org/10.1007/s10589-010-9330-x.

[4] Joel Chestnutt, James Kuffner, Koichi Nishiwaki, and Satoshi Kagami. Planning biped navigation strategies in complex environments. In *IEEE Intl. Conf. on Humanoid Robotics, Munich, Germany*, 2003.

[5] David DeMers and Kenneth Kreutz-Delgado. Learning global direct inverse kinematics. In *Neural Information Processing Systems*, pages 589–595. Citeseer, 1991.

[6] Corey Goldfeder and Peter K. Allen. Data-driven grasping. *Autonomous Robots*, 31(1):1–20, 2011. ISSN 1573-7527. doi: 10.1007/s10514-011-9228-1. URL http://dx.doi.org/10.1007/s10514-011-9228-1.

[7] S. Gottschalk, M. Lin, and D. Manocha. OBB-tree: A hierarchical structure for rapid interference detection. In *ACM SIGGRAPH*, pages 171–180, 1996.

[8] Keith Grochow, Steven L. Martin, Aaron Hertzmann, and Zoran Popović. Style-based inverse kinematics. In *ACM SIGGRAPH 2004 Papers*, pages 522–531, New York, NY, USA, 2004. ACM. doi: 10.1145/1186562.1015755. URL http://doi.acm.org/10.1145/1186562.1015755.

[9] K. Hauser, T. Bretl, K. Harada, and J.-C. Latombe. Using motion primitives in probabilistic sample-based planning for humanoid robots. In *Intl. Workshop on the Algorithmic Foundations of Robotics(WAFR)*, 2006.

[10] Kris Hauser. Learning the problem-optimum map: Analysis and application to global optimization in robotics. *CoRR*, abs/1605.04636, 2016. URL http://arxiv.org/abs/1605.04636.

[11] L. He, J. Pan, D. Li, and D. Manocha. Efficient penetration depth computation between rigid models using contact space propagation sampling. *IEEE Robotics and Automation Letters*, 1(1):10–17, Jan 2016. ISSN 2377-3766. doi: 10.1109/LRA.2015.2502919.

[12] Edmond SL Ho, Hubert PH Shum, Yiu-ming Cheung, and Pong C Yuen. Topology aware data-driven inverse kinematics. *Computer Graphics Forum*, 32(7):61–70, 2013.

[13] Prateek Jain, Brian J. Kulis, Inderjit S. Dhillon, and Kristen Grauman. Online metric learning and fast similarity search. In *Neural Information Processing Systems (NIPS)*, dec 2008.

[14] Nikolay Jetchev and Marc Toussaint. Fast motion planning from experience: trajectory prediction for speeding up movement generation. *Autonomous Robots*, 34(1-2): 111–127, 2013.

[15] Donald R Jones. Direct global optimization algorithm. In *Encyclopedia of optimization*, pages 431–440. Springer, 2001.

[16] James J Kuffner Jr, Koichi Nishiwaki, Satoshi Kagami, Masayuki Inaba, and Hirochika Inoue. Footstep planning among obstacles for biped robots. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, volume 1, pages 500–505, 2001.

[17] Jyh-Ming Lien and Yanyan Lu. Planning motion in environments with similar obstacles. In *Robotics: Science and Systems*, 2009.

[18] Ziqiang Mao and TC Hsia. Obstacle avoidance inverse kinematics solution of redundant robots by neural networks. *Robotica*, 15(01):3–10, 1997.

[19] Jia Pan, Zhuo Chen, and Pieter Abbeel. Predicting initialization effectiveness for trajectory optimization. In *IEEE Intl. Conf. Robotics and Automation*, 2014.

[20] Mike Phillips, Benjamin Cohen, Sachin Chitta, and Maxim Likhachev. E-graphs: Bootstrapping planning with experience graphs. In *Robotics: Science and Systems*, 2012.

[21] Aaron D Souza, Sethu Vijayakumar, and Stefan Schaal. Learning inverse kinematics. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, volume 1, pages 298–303. IEEE, 2001.

[22] Jun Wang, Qingni Hu, and Danchi Jiang. A lagrangian network for kinematic control of redundant robot manipulators. *IEEE Trans. Neural Networks*, 10(5):1123–1132, 1999.

[23] Yunong Zhang and Jun Wang. Obstacle avoidance for kinematically redundant manipulators using a dual neural network. *IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics*, 34(1):752–759, 2004.