J. Appl. Numer. Optim. 4 (2022), No. 2, pp. 143-159
Available online at http://jano.biemdas.com
https://doi.org/10.23952/jano.4.2022.2.03

AVOIDING TRAPS IN NONCONVEX PROBLEMS

SEAN DEYO*, VEIT ELSER

Department of Physics, Cornell University, Ithaca, NY, USA

Abstract. Iterative projection methods may become trapped at non-solutions when the constraint sets are nonconvex. Two kinds of parameters are available to help avoid this behavior and this study gives examples of both. The first kind of parameter, called a hyperparameter, includes any kind of parameter that appears in the definition of the iteration rule itself. The second kind comprises metric parameters in the definition of the constraint sets, a feature that arises when the problem to be solved has two or more kinds of variables. Through examples we show the importance of properly tuning both kinds of parameters and offer heuristic interpretations of the observed behavior.

Keywords. Dominating sets; Fixed-point algorithms; Logical satisfiability; Machine learning; Projection methods; Nonconvex problems.

1. INTRODUCTION

Iterative algorithms whose elementary operations are projections to constraint sets perform well on many nonconvex problems for which one does not have the convergence guarantees one has in convex problems. For some combinatorially hard problems these algorithms routinely outperform state-of-the-art algorithms that find solutions by exhaustive search [1]. This success, or the apparent ability of the projection-based search to very significantly reduce the size of the space being searched, is poorly understood.

The standard criteria for evaluating iterative projection algorithms in the convex case do not apply in the nonconvex case. The amount of time the algorithm spends refining a solution, once its locally convex basin has been encountered, is negligible compared to the time needed to find the basin. Efficiency in basin discovery completely overshadows the benefits of good convergence within the basin.

Iterative projection algorithms usually follow a fixed-point principle, where fixed-points imply a solution, but these algorithms may still get trapped on non-solutions in a dynamic sense [2]. When this happens, the iterations meander indefinitely within a small domain far from a true fixed point. Eliminating or mitigating this behavior by tuning the parameters of the algorithm is the focus of this paper.

Most iterative projection algorithms have parameters that apply to the general case, independent of the application's constraint sets. For such parameters we use the machine learning term "hyperparameter." Relaxation parameters are examples of hyperparameters. Another type of

*Corresponding author.
E-mail addresses: sjd257@cornell.edu (S. Deyo), ve10@cornell.edu (V. Elser).
Received May 10, 2021; Accepted April 19, 2022.

parameter has only come to light more recently, in applications that require multiple kinds of variables [3]. These applications have variable-scaling freedom that is not a geometrical isometry and therefore changes the algorithm through its effect on the projections. We refer to such parameters as "metric parameters."

2. Hyperparameters and metric parameters

One of the best known hyperparameters is the relaxation that is often applied to the standard Douglas-Rachford iteration

$$x \mapsto (1-\beta/2)\,x + (\beta/2)\,R_B(R_A(x)), \tag{2.1}$$

with $\beta \in]0,2[$. This is called the generalized Douglas-Rachford method [4]. Here R_A and R_B are the reflectors for the constraint sets A and B: $R_i(x) = 2P_i(x) - x$, where $P_i(x)$ is the projection of x to set i. In the case where one of the sets A or B is nonconvex and there exist points x where the projection is not unique, we use the fact that such points have measure zero [5] and are not expected to arise in computations with sufficiently high floating point precision. The hyperparameter β is the same parameter that independently was deemed important when this iteration was proposed — by an engineer unaware of Douglas-Rachford — for the phase retrieval application [6]. To make the point that the scope of hyperparameters is broad, we give an example in section 3 of a very different generalization of Douglas-Rachford, in which tuning a hyperparameter is critical for success.

In imaging applications [7], or even sudoku [8], where there is just one kind of pixel or cell, the question of metric parameters never came up. An application where introducing metric parameters makes sense is non-negative matrix factorization (NMF) [9]. In NMF one seeks a low-rank factorization of a rectangular non-negative matrix $Z = XY$, where the factors are themselves non-negative. The rows of Z may be interpreted as data vectors that can be expressed as non-negative mixtures, given by X, of a set of non-negative features, the rows of Y. NMF is non-unique with respect to rescaling ($X \to XD$, $Y \to D^{-1}Y$, arbitrary diagonal and positive D). To remove this ambiguity and also to make the problem compact, we can choose to impose a norm on the columns of X or the rows of Y. If we decide to normalize the features (Y), what setting of the norm do we choose and why is that a choice of metric?

To motivate the term "metric," consider the standard distance that would be used in defining the projections for the NMF application:

$$d\left((X,Y),(X',Y')\right) = \sqrt{\|X-X'\|_2^2 + \|Y-Y'\|_2^2}. \tag{2.2}$$

Now, if we chose to impose normalization on each row vector y of Y, say $\|y\|_2 = \eta$, we could alternatively work with the rescaled variables $\tilde{Y} = Y/\eta$, normalization constraint $\|\tilde{y}\|_2 = 1$, and the distance

$$d\left((X,\tilde{Y}),(X',\tilde{Y}')\right) = \sqrt{\|X-X'\|_2^2 + \eta^2\|\tilde{Y}-\tilde{Y}'\|_2^2}. \tag{2.3}$$

This rewriting by a parameterized distance provides an interpretation of the norm parameter η. Consider the projection to the bilinear constraint, $X\tilde{Y} = Z/\eta$. From the distance (2.3) we see that for small η the $\tilde{Y}$ variables (features) are more compliant than X (mixtures), and we should expect that product-constraint inconsistencies are resolved mostly by changing the features. The opposite, or more reliance on changing the mixtures, is expected for large η. Whether one chooses to normalize y to η and use the standard metric (2.2) or keep y normalized to 1 and use

the modified metric (2.3) is a matter of implementation. In this paper our equations assume the former choice.

NMF is one of the earliest techniques of machine learning, and we anticipate that variable-type metric sensitivity will grow in relevance as projection methods find their way into this domain[1]. In particular, splitting methods lend themselves naturally to optimization on networks, where variables appear on both nodes and edges of the network and are clearly dissimilar.

3. HYPERPARAMETERS

A tuneable double-reflector algorithm. In this section we consider a generalization of the Douglas-Rachford iteration that is very different from the standard relaxation (2.1) :

$$x \mapsto \mathrm{DR}_n[\delta](x) = \frac{1}{1+n} \sum_{r=0}^{n} (R_B[\delta] \circ R_A[\delta])^r(x) \; . \tag{3.1}$$

The number of double reflections, n, is one of the algorithm's hyperparameters. However, we will see that this algorithm only succeeds when the reflectors are themselves parameterized,

$$R_i[\delta](x) = (2-\delta) P_i(x) - (1-\delta) x \; ,$$

with $\delta \in [0,1]$. The original Douglas-Rachford iteration is recovered for $n = 1$ and $\delta = 0$ and will be referred to as $\mathrm{DR}_1[0]$.

The idea behind (3.1) is that the double reflector acts as the identity when x is near a feasible point, and therefore the average of any number of double reflections fixes x. On the other hand, multiple ($n > 1$) applications of the double reflector might be better at ejecting x from a trap when it is not near a feasible point. Through elementary analysis and numerical experiments we will argue that there is an optimal δ^* such that when $\delta > \delta^*$ the reflections are in effect contractive, trapping the iterations at non-solutions, while for $\delta < \delta^*$ the iterate diffuses too freely to notice even the true solutions. The algorithm works best when δ is tuned to the dynamical transition point δ^*.

Hard feasibility problems, including the one we feature below, can often be formulated where one constraint set, say A, is finite and the other, B, is a hyperplane. Traps arise when a point $a \in A$ is very close to B, that is, when the distance $\Delta = \|a - b\|$, to the proximal point $b = P_B(a)$ on the hyperplane, is very small. We analyze the local trapping/escaping behavior by replacing the sets A and B by proximal points $a \in A$, $b \in B$ and the ambient space by the line passing through these points. Let x be the (1D) coordinate along this line with a corresponding to $x = 0$ and b corresponding to $x = -\Delta$. In this simplified model of the local behavior one finds

$$\mathrm{DR}_n[\delta](x) = \gamma x + c \; , \tag{3.2}$$

1. The term "parameter" is potentially confusing in the machine learning context. For example, the *weight parameters* of a neural network that are learned from data are variables from the perspective of the optimization algorithm. The latter may use *metric parameters* to more efficiently optimize the *weight variables*.

where

$$\gamma = 1 - \frac{n(1-q) - q(1-q^n)}{(1+n)(1-q)},$$
$$q = (1-\delta)^2,$$
$$c = (1-\gamma)\left(\frac{1-\delta}{\delta}\right)\Delta\,.$$

For comparison,

$$\mathrm{DR}_1[0](x) = x + \Delta$$

represents a step-wise escape, where $O(1/\Delta)$ iterations are needed before x has changed by $O(1)$. This is an estimate of the number of iterations, in the original problem, for $P_A(x)$ to be significantly different from the point a of the trap.

Since $0 < \gamma < 1$ for $0 < \delta < 1$, iteration (3.2) is contractive and looks problematic because there is always a (non-solution) fixed point:

$$x^* = \frac{c}{1-\gamma} = \left(\frac{1-\delta}{\delta}\right)\Delta\,. \tag{3.3}$$

However, (3.3) should be seen as instructions on the proper use of the algorithm. Since one constraint set is finite there will be a smallest Δ that poses the greatest trapping risk. But by setting $\delta = \delta^* = O(\Delta)$, the fixed point x^* of the 1D dynamics is sufficiently far from the trap that $P_A(x^*)$ will likely be different from the original trapping point a. Since $c \sim n\Delta$ for $\delta \to 0$, a single iteration of DR_n is roughly the same as n iterations of DR_1, although both schemes require n double-reflector computations.

Experiments with logical satisfiability. To illustrate the effect of the hyperparameter δ in a setting with potentially many traps, we turn to the logical satisfiability problem (SAT). In SAT we have a set C of *clauses* and a set V of *variables*. Thinking of these as vertices of a bipartite graph G, the search variables E in our constraint formulation correspond to edges $c \to v$, where $c \in C$ and $v \in V$. In the SAT interpretation, the variable-vertices v incident on a particular c in G correspond to the Boolean variables that participate in one clause of a logical formula in conjunctive normal form. The clause itself is a disjunction

$$y_c = \bigvee_{v \in V:\, c \to v \in E} n_{c\to v} \circ x_{c\to v},$$

where the x are Boolean variables, $n\circ$ specifies whether to apply negation, and y_c is the Boolean value of clause c. The object in SAT is to find an assignment to the x such that the conjunction

$$\bigwedge_{c \in C} y_c \tag{3.4}$$

is true. The set of Boolean variables $x_{c\to v}$ incident on the same v (but appearing in different clauses c) should all be equal. However, in the two-constraint formulation [10] to which we turn next, these are treated as independent in one of the constraints.

The two constraint sets live in a space of dimension $|E|$. Set A is finite and imposes the truth of each clause (otherwise the conjunction (3.4) is false). We encode TRUE and FALSE for the

δ	successes/trials	iterations/solution
0.001	0/100	—
0.002	0/100	—
0.005	90/100	3.23×10^3
0.010	86/100	4.57×10^3
0.020	35/100	2.27×10^4
0.050	0/100	—

TABLE 1. Performance statistics for several values of δ as algorithm (3.1) tries to solve an instance of the 3-SAT problem with 500 Boolean variables and 2100 clauses. Each trial was capped at 10^4 iterations.

Boolean variables as respectively $x = +1$ and $x = -1$ and use multiplication by $n = -1$ for negation:

$$\begin{aligned} A: \\ &\forall\, c \to v \in E: \quad x_{c\to v} \in \{-1, 1\}\,, \\ &\forall\, c \in C: \qquad +1 \in \bigcup_{v\in V:\, c\to v \in E} n_{c\to v}\, x_{c\to v}\,. \end{aligned}$$

Because the A constraint imposes the discreteness of the Boolean variables we are free to use the following relaxed, continuous constraint for B:

$$B: \quad \forall\, v \in V: \exists\, \bar{x}_v \in \mathbb{R}: \quad \forall\, (c \in C: c \to v \in E): x_{c\to v} = \bar{x}_v\,.$$

We consider a hard instance of 3-SAT, where each clause involves exactly three variables and there are altogether $|V| = 500$ Boolean variables in the logical formula. The number of clauses $|C| = 2100$ was tuned so that a typical random instance has roughly even odds of being satisfiable [11] (our instance is satisfiable). All our results will be for iteration DR_3, for which the number of double reflections ($n = 3$) is large enough that a transition in behavior with δ is easy to discern. Each trial starts with a random initial point x, with each of the $|E|$ variables chosen in the range $[0, 1]$.

The performance statistics are given in Table 1. The smallest values of δ are completely ineffective. There is a sharp onset of good performance at $\delta = 0.005$, but the larger values of δ are also ineffective. To understand why, it is helpful to look more closely at a few individual trials. To visualize the behavior, we store the time series of the $|E|$ search variables in a matrix, where each row specifies a point in $\mathbb{R}^{|E|}$. We then find the principal component axes for this matrix and project each row onto the first two principal components to obtain the 2-dimensional plots in Figure 1 of the search as a function of time. Since we are projecting points with root-mean-square distance $O(\sqrt{|E|})$, we also divide the principal components by $\sqrt{|E|}$ to normalize the length scale in the 2D plots. Alongside each PCA time series we provide the time series of the constraint error

$$\varepsilon = |P_A(x) - P_B(R_A(x))|$$

for the points x generated by the DR_3 iteration.

In Figure 1 we see that the search with $\delta = 0.001$ (top plots) jumps around aimlessly with large ε and never finds a solution. Our interpretation is that when δ is small, each reflection $R_i(x)$ is almost the same as the pure reflection $2P_i(x) - x$ and the algorithm never allows itself

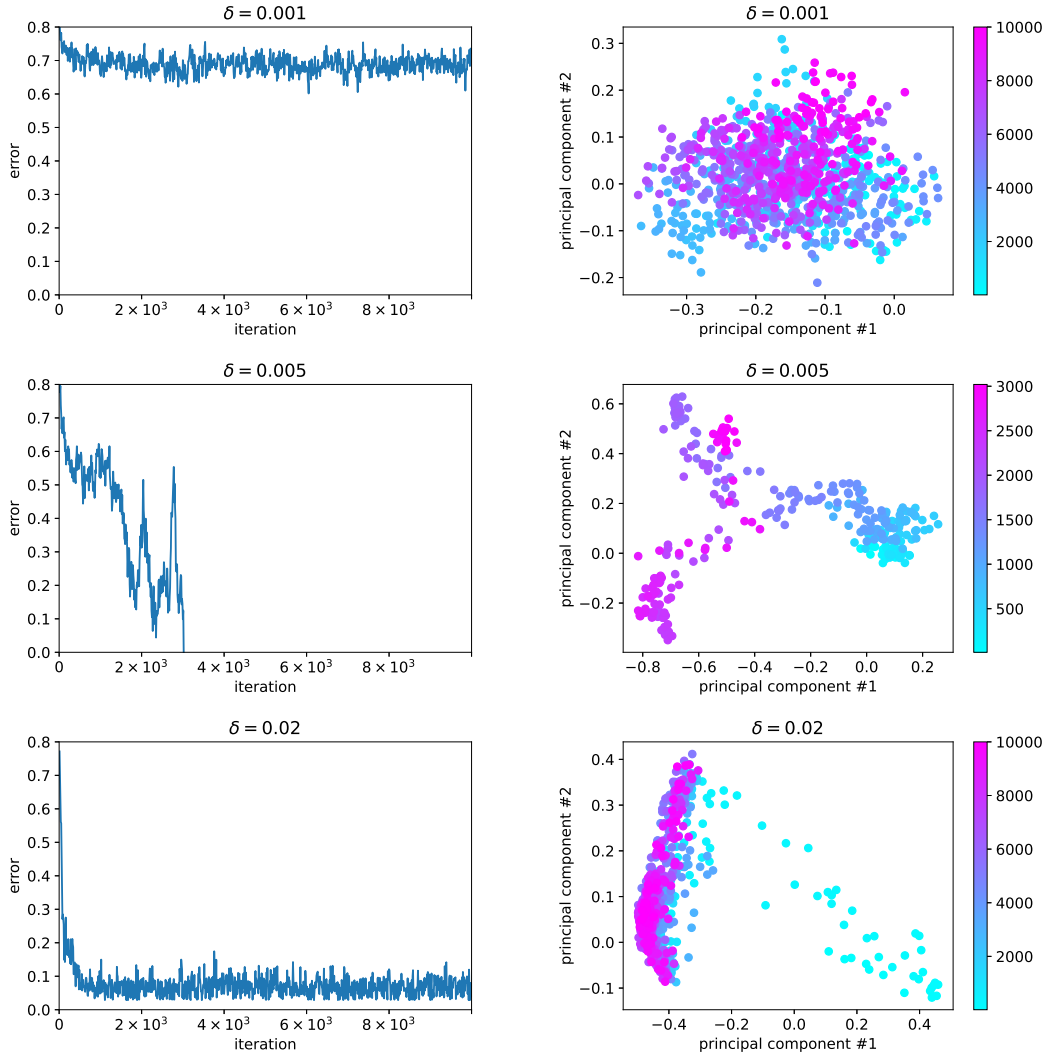

FIGURE 1. Plots of the constraint error ε (left column) and the iterate x (right column) of the DR_3 algorithm applied to an instance of 3-SAT. The plots on the right show the iterate x projected into a 2D space by PCA. These data are plotted in increments of 10 iterations up to a maximum of 10^4 iterations, with the color bar indicating the iteration number.

to fall into a basin of lower ε. When $\delta = 0.02$ (bottom plots of Fig. 1), the algorithm quickly gets trapped on the first basin it finds, corresponding to a non-solution, and remains there for the rest of the search. The intermediate choice $\delta = 0.005$ (middle plots of Fig. 1) seems to be just right. The algorithm now appears to start exploring a basin once or twice but only stays for a few hundred iterations before wandering to another basin. After about 3×10^3 iterations it finds a basin that has a solution and converges toward it.

4. METRIC PARAMETERS

Metric parameters, unlike hyperparameters, are special for each intended application and should always be considered when there is more than one type of variable. Below we give

two examples, both involving applications where the variables take discrete values and live on a network. As with hyperparameters, the settings of the metric parameters make all the difference between an algorithm that never finds solutions and one that does so consistently.

Often a metric parameter will not have an obvious interpretation, or will have non-obvious interactions with the other metric parameters. Tuning these metric parameters can be done by hand and is informed by appropriate diagnostics that go beyond the standard "success rate" statistic. However, an automated procedure to expedite this process is desirable, especially when the number of metric parameters is large. We use a scheme where the current status of the search informs the rule for the parameter updates, and apply these updates adiabatically so as not to upend the fixed-point properties of the algorithm being used.

Our metric parameter update rule is based on the following heuristic. We want to prevent the algorithm from getting stuck on a partial solution, wherein some variables hardly change between the A and B projections while others are changing very much. We avoid this by giving a smaller metric parameter to the variables that are hardly changing, thereby lowering the penalty for changing them when the next projection comes, and vice versa for variables that change too much.

To be precise, suppose we partition the variables into k types, with metric parameters η_1, η_2, ... η_k. Let l_i denote the number of variables of type i, so that the complete variable vector can be written as a concatenation $x = (x_1, x_2, ..., x_k)$, where x_1 is a vector of length l_1 and so on. The projections are made using the distance function

$$d(x,x') = \sqrt{\sum_{i=1}^{k} \|x_i - x_i'\|^2}.$$

In each iteration, we compute the normalized rms error

$$\varepsilon_i = \frac{1}{\eta_i} \sqrt{\|P_A(x)_i - P_B(R_A(x))_i\|^2 / l_i}$$

for each variable type. We then compare each ε_i to the average $\varepsilon = \sqrt{\frac{1}{k}\sum_i \varepsilon_i^2}$ and adjust the metric parameters according to

$$\eta_i \to \eta_i \left(1 + \alpha \left(\varepsilon_i/\varepsilon - 1\right)\right),$$

where $\alpha \ll 1$ is a small but positive tuning parameter. Alternatively, since only the relative weights of the variable types matter, one can set $\eta_1 = 1$ and take

$$\eta_i \to \eta_i \left(1 + \alpha \left(\varepsilon_i/\varepsilon_1 - 1\right)\right)$$

for $i > 1$.

Automatically updating the metric parameters in this fashion encourages the smaller ε_i's to become larger and vice versa, which leads to the ε_i's being highly correlated, which in turn generally leads to more successful searching. This approach also replaces the problem of tuning some (possibly large) number of metric parameters with the simpler question of choosing a value of α. One must have $\alpha \ll 1$ in order to make the metric parameter updates adiabatic and thereby preserve the local convergence properties of the algorithm. One also wants $\alpha \gg 1/N$, where N is the total number of iterations to be run, so as to accomplish the desired tuning within

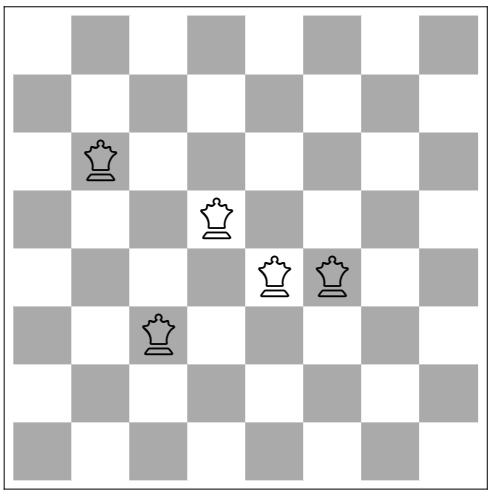

FIGURE 2. Five queens "dominating" the 8×8 chess board; that is, each unoccupied square is attacked by a queen. The domination number of the order 8 queens' graph is 5 because this is not possible with fewer queens [12].

the intended length of the run. Accordingly, for large N there can be a rather generous range of α that will work.

Experiments. For the following experiments we use the relaxed Douglas-Rachford algorithm (2.1). All variables are initialized to random real values between 0 and 1. In each example we first choose a value of the hyperparameter β that works for that particular problem ($\beta = 0.5$ for the dominating sets example, $\beta = 0.8$ for the Boolean generative networks example), then choose a challenging instance of the problem and demonstrate the benefits of tuning the metric parameters while keeping the hyperparameter fixed. The first experiment will demonstrate how the tuning works and why it is important to have a small value of α. The second experiment will demonstrate the main goal of tuning: preventing the algorithm from getting stuck on problems where such trapping is common for the untuned algorithm.

Experiments with dominating sets. Consider a graph G with vertices V and (undirected) edges E. A subset $D \subset V$ *dominates* G if for every vertex $i \in V$ either $i \in D$ or there exists an adjacent vertex $j \in D$ such that $(i, j) \in E$. Finding dominating sets of minimum size $|D|$ is a well known NP-hard problem. Chess players will recognize the configuration shown in Figure 2 as an instance of a dominating set. Here the squares of the board are graph vertices and two squares are "connected" by an edge if a queen placed on one attacks the other. The *domination number* of this particular board/graph is $|D| = 5$ because the five queens attack all the other squares and this is not possible with fewer queens [12].

That a vertex may be dominated either by itself or an adjacent vertex calls for two types of variable in a constraint formulation. A metric parameter should be introduced to control their relative "weight." Our formulation uses vertex variables y_i, $i \in V$ and two variables for each edge $(i, j) \in E$ denoted $x_{i\to j}$ and $x_{j\to i}$ corresponding to the set of doubled (directed) edges E_2. One can think of $x_{i\to j}$ as a copy of vertex i that vertex j uses to express its domination status. Since there are only two variable types, a single metric parameter η suffices to characterize the weight of the vertex variables relative to the edge variables. As explained in section 2, we choose to

absorb the metric weight by variable rescalings, specifically, in different discrete settings for the edge and vertex variables.

Constraint A demands that every vertex is either "dominating" or is "dominated" by at least one adjacent vertex via an incident edge, with at most $|D|$ vertices being dominating:

$$
\begin{aligned}
&A: \\
&\quad \forall\ j \to i \in E_2: \qquad x_{j\to i} \in \{0,1\}\ , && (A1)\\
&\quad \forall\ i \in V: \qquad y_i \in \{0,\eta\} \quad \wedge \quad y_i/\eta\ + \sum_{j\to i} x_{j\to i}\ \geq 1\ , && (A2)\\
&\quad \sum_{i\in V} y_i/\eta\ \leq |D|\ . && (A3)
\end{aligned}
$$

Constraint B demands that all edge variables agree with their associated vertex variable:

$$B: \quad \forall\ j \to i \in E_2: \qquad x_{j\to i} = y_j/\eta\ .$$

We project to the constraint sets using the metric $d((x_1,y_1),(x_2,y_2)) = \sqrt{||x_1-x_2||^2 + ||y_1-y_2||^2}$. The following is pseudocode for projecting to the discrete set A:

- For all $j \to i \in E_2$, set $P_A(x_{j\to i}) = 0$ if $x_{j\to i} < 1/2$ and set it to 1 otherwise. Also, set $P_A(y_i) = 0$ for all $i \in V$. Call this the base state. This state satisfies constraint $(A1)$.
- Determine which j has the largest $x_{j\to i}$ for each $i \in V$. Call this $j_{max}(i)$. Also, for each $i \in V$ determine the change in squared distance $d_1(i)$, relative to the base state, for i to be dominating, or $y_i = \eta$, as well as the change in squared distance $d_0(i)$, relative to the base state, for i to be dominated, or $y_i = 0$, $x_{j_{max}(i)\to i} = 1$. Since only y_i or $x_{j_{max}(i)\to i}$ might need to change, these are easy computations. The results are:

  $$
  \begin{aligned}
  d_1(i) &= \eta^2 - 2\eta y_i,\\
  d_0(i) &= \max(0, 1 - 2x_{j_{max}(i)\to i}).
  \end{aligned}
  $$

  Setting $y_i = \eta$ whenever $d_1(i) < d_0(i)$ or $x_{j_{max}(i)\to i} = 1$ whenever $d_1(i) > d_0(i)$ would be distance minimizing to constraint $(A2)$. However, this may not satisfy $(A3)$.
- To determine the distance minimizing set of vertices that should be dominating rather than dominated, we consider the differences $d_0(i) - d_1(i)$ for all $i \in V$. While vertices with $d_0(i) < d_1(i)$ should be dominated ($y_i = 0$), of the vertices with $d_0(i) > d_1(i)$ at most $|D|$ may be dominating ($y_i = \eta$) by constraint $(A3)$. If this number exceeds $|D|$, the distance minimizing subset of dominating vertices is those $|D|$ with the largest values of $d_0(i) - d_1(i)$.

Because of the vertex ranking in the last step, constraint A is slightly non-local. Projection to set B is much simpler:

- For all $j \in V$, compute the weighted average $\left(\eta y_j + \sum_{j\to i} x_{j\to i}\right)/(\eta^2 + \text{degree}(j))$, where degree$(j)$ is the number of edges leaving node j. Set $P_B(x_{j\to i})$ equal to this average, and set $P_B(y_j)$ to η times this average.

Table 2 gives performance statistics for several values of α in searches for a dominating set of size $|D| = 6$ (the smallest possible [12]) for the queens' graph of order 12. The initial conditions were chosen randomly for each of the 100 trials, but for the sake of fair comparison the 100 sets of initial conditions were the same for each value of α. In all cases we initialize $\eta = 1$.

α	successes/trials	iterations/solution
0	82/100	4.18×10^5
10^{-5}	87/100	3.59×10^5
10^{-4}	89/100	3.45×10^5
10^{-3}	57/100	1.18×10^6

TABLE 2. Performance statistics for several values of α in searches for a size 6 dominating set in the 12×12 queens' graph. Each trial was capped at 10^6 iterations.

The main point of metric tuning is to avoid traps, not necessarily to speed up the algorithm for applications such as this in which trapping is rare; nonetheless, it is worth noting that the algorithm actually found solutions somewhat more quickly and reliably with $\alpha = 10^{-5}$ and $\alpha = 10^{-4}$ than with $\alpha = 0$. However, more tuning is not necessarily better: $\alpha = 10^{-3}$ does worse than $\alpha = 0$.

To understand the influence of *alpha*, it is helpful to plot the metric parameter η as a function of iteration number. Figure 3 does this for ten trials of the three nonzero settings of α. Each curve in the figure terminates when the algorithm finds a solution. For $\alpha = 10^{-5}$, η gently slides toward a value of around 0.6, terminating early if a solution is found. With $\alpha = 10^{-4}$ it reaches something near 0.6 more quickly, though it tends to fluctuate more. A value somewhere between $\alpha = 10^{-5}$ and $\alpha = 10^{-4}$ might be optimal for this problem, but there is a generous range of α that works just fine: As long as α is significantly greater than $1/N$, where N is the maximum number of iterations, there will be plenty of time for the tuning to take place. However, larger α is not always better, as we see with $\alpha = 10^{-3}$. Here η has large amplitude oscillations, almost periodically on a time scale of 10^4 iterations. The metric is no longer evolving adiabatically and making a quasi-monotonic approach to a steady value. It is now changing so quickly that the local convergence properties of the Douglas-Rachford algorithm are compromised, resulting in the poor performance of $\alpha = 10^{-3}$ in Table 2. This is why it is important to keep α small.

Experiments with Boolean generative networks. In the previous experiment metric parameter tuning provided marginal benefits, slightly reducing the number of iterations needed to find a solution. In this next experiment we shall see an application in which trapping is common and the benefits of tuning are much more dramatic. In fact, tuning is essential to have any hope of finding solutions at all.

Our second example of automated metric parameter tuning is unsupervised machine learning with a Boolean generative network, or BGN [13]. The data in this application consist of a set of D Boolean strings of length N, and the network is tasked with discovering a Boolean circuit that generates the strings from a smaller number $M < N$ of Boolean "latent variables" whose values for each data string are unknown. Figure 4 compares a network before and after training. Nodes of the network are arranged in layers and initially each node can potentially receive input from any node in the layer below. However, the data come with the promise or hypothesis that they can be generated with only NOT and 2-input OR gates. Through training the network must therefore discover which of the many edges are utilized in the circuit, that is, the "wires" of the circuit, and whether the Boolean value is to be negated when traversing the wire.

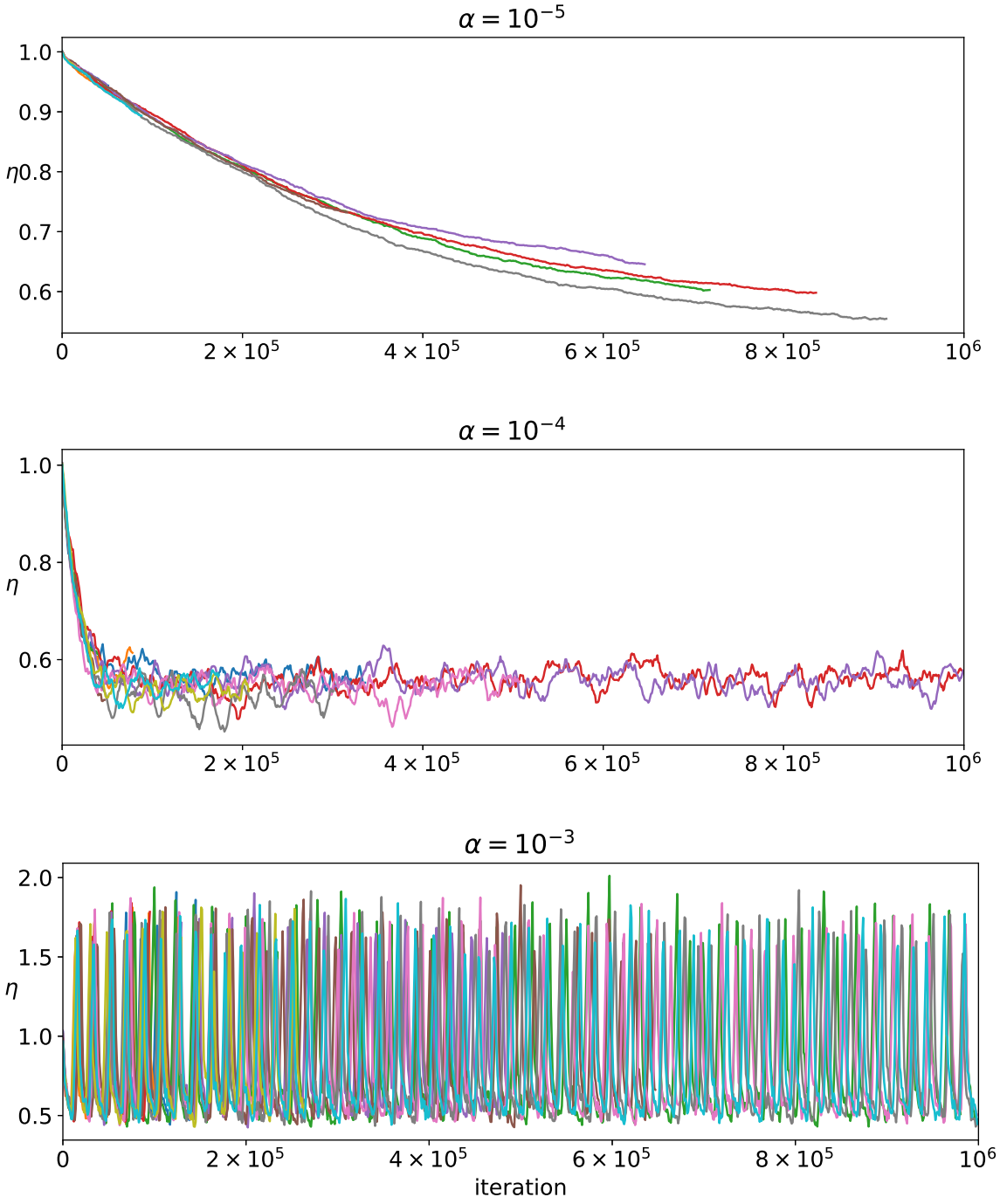

FIGURE 3. Time series of the metric parameter η for ten trials at three difference values of α, searching for a dominating set for the order 12 queens' graph. Each curve terminates when the algorithm finds a solution.

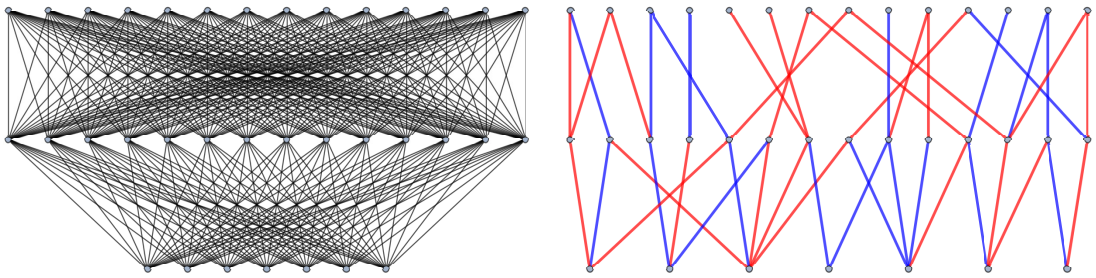

FIGURE 4. Network architecture (left) that takes 7 Boolean inputs and outputs 14 Boolean data; there is also an intermediate layer that holds 14 Boolean values. The trained BGN (right) is a logical circuit that uses relatively few of the available network edges as wires. All the nodes not in the input layer apply OR to the wires incident from below. Training also determines the placement of the NOT gates, which are indicated by red wires.

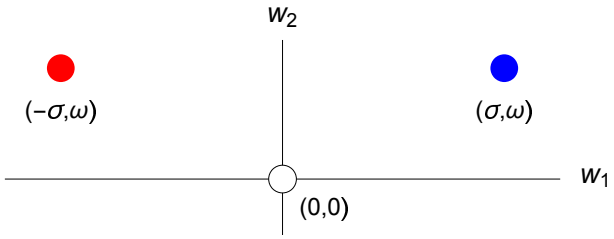

FIGURE 5. A 2D space (w_1, w_2) is required to encode the three possible states of edges in a BGN. The red and blue points correspond to edges utilized as wires with and without negation; the open circle encodes the absence of a wire. Two metric parameters, σ and ω, define the isosceles geometry of the states.

The truth value at a node i of the BGN is encoded by a variable y_i at that node and also by copies of that truth value on all its out-edges, $x_{i\to j}$. These two variable types have the same interpretation they had in the dominating set application and serve to localize the constraints. However, by the uni-directional nature of the BGN logic, there is no need to have a second x variable on each edge. Just as with dominating sets, the semantic equivalence of the x and y variables does not imply metrical equivalence and we control their relative scale with metric parameters θ for x and η for y.

The relevance of metric parameters in BGNs is made especially obvious when we also consider the variables $w_{i\to j}$ used to encode whether an edge $i \to j$ of the network is a wire and if so, its negation state. Not only is the wire-status of an edge semantically distinct from the truth states it operates on, the existence of a wire (on the edge) should have independence, metrically, over its two negation states. To encode the three states of an edge (no wire, negating wire, non-negating wire) as three points we give each w variable two independent components $w = (w_1, w_2)$ and represent the wire states with the vertices of an isosceles triangle as shown in Figure 5. This is the most general metrical relationship among the states that respects the symmetry between negated and non-negated wires. The metric parameter ω now controls the distance between presence and absence of a wire, while σ controls the distance between the presence and absence of a NOT.

Here is a brief overview of the two constraint sets; see [13] for further discussion of the projections in the BGN problem. In order to solve the problem we must create one copy of the network for each of the D data strings. The A constraint demands that every node, in each network copy, makes sense locally:

- Each incoming w variable takes on one of the three values in Figure 5 (representing a negating wire, non-negating wire, or no wire).
- Each incoming x variable is either 0 or θ (signifying that it carries value FALSE or TRUE).
- The y variable at the node is 0 if all of the incoming wires pass FALSE or η if at least one wire passes a TRUE.

The B constraint makes the various copies agree:

- The x variable on an edge represents a copy of the truth value at its lower node in the network, and all the outgoing x's from that node must agree with the y variable at that node. These variables need not take discrete values in this equality constraint because the A constraint sees to that. The distance-minimizing way to make the x's and y's agree is a weighted average, just like in the B projection for the dominating sets problem.
- The w variables represent the wire states, which, in a solution, must be the same across all D copies of the network. Again, in this equality constraint the variables may take continuous values, so we simply average each edge's w variable over all copies of the network.

We will refer to tuning the four metric parameters ω, σ, θ, and η as tuning by type. Now, it may happen that certain nodes and edges in the network are more important than others. For instance, nodes in different layers or having different in- or out-degrees could play markedly different roles. It is natural then to let the metric parameters be different for every node and edge in the network, promoting ω, σ, θ, and η to $\omega_{i\to j}$, $\sigma_{i\to j}$, $\theta_{i\to j}$, and η_i. We will refer to this as tuning by type and location. One might also argue that the D data strings have differing inherent difficulty and deserve suitably tuned metric parameters applied to their copies of the network. Instead of a single $\omega_{i\to j}$ parameter for edge $i \to j$ there would then be parameters $\omega_{k,i\to j}$ for $k = 1,\ldots,D$ and likewise for the other variable types. We will refer to this as tuning by type, location, and data item.

We compared the different degrees of metric parameter tuning on a synthetic data set generated by the circuit in Figure 4 with $M = 7$ inputs and $N = 14$ outputs. For training we used all $D = 87$ unique data strings generated by this circuit, initialized all metric parameters to 1, and initialized the variables (on nodes and edges of the complete architecture) to random values between 0 and 1. Solutions (circuits) were only required to generate all D data strings for some setting of the M inputs. Typically the solution circuits additionally generated strings not among the D data strings, though never a set of size 2^M.

Table 3 gives performance statistics for the four degrees of tuning, each run for 100 trials capped at 10^5 iterations per trial. As in the previous experiment, the random initial conditions were the same for each of the four tuning approaches, but different from one trial to the next. Clearly no tuning at all is not a viable option. Tuning by type is better, but still rarely succeeds within the limit of 10^5 iterations per trial. Evidently it is necessary to tune by type and location, meaning that there is a $\theta_{i\to j}$ for each edge, an η_i for each node, and so on. If tuning by type alone had been more effective, one might have considered doing so by hand: trying some choice

tuning	successes/trials	iterations/solution	iterations/second
none	0/100	—	337.77
by type	2/100	5.07×10^6	396.56
by type and location	58/100	1.85×10^5	427.73
by type, location, and data item	56/100	1.89×10^5	198.03

TABLE 3. Performance statistics for four degrees of metric parameter tuning in an instance of the BGN problem (Figure 4). Each trial was capped at 10^5 iterations.

of metric parameters for one trial, tweaking them according to which variable types appear to be getting stuck, and repeating. But tuning by type and location involves hundreds of metric parameters, making it infeasible to tune by hand. This is what makes our automated approach to tuning helpful.

The additional refinement of tuning by type, location, and data item does not make the search any more successful, and indeed it comes with a considerable slowdown in iterations performed per second. This is the drawback of tuning a large number of metric parameters. If the metric parameters are so numerous that tuning-work becomes an issue, one should first ask if having so many is truly necessary. In the present example, tuning by data item did not add any benefit that was not already seen with tuning by type and location. In an application in which the many metric parameters are all necessary one should consider ways of reducing the work, such as applying the metric updates less frequently than after every iteration.

For insight on how the four degrees of tuning affect the behavior of the Douglas-Rachford algorithm, it is helpful to look at plots of constraint error vs. iteration number. This is shown in Figure 6 where the error is broken down into contributions from each of the four variable types: ε_{w_1}, ε_{w_2}, ε_x, ε_y. The ideal behavior would be all four types fluctuating more or less in concert and with about the same amplitude, indicating that the difficulty of solving the problem is being shared equitably among the variable types.

What we see in the top left plot is far from the ideal behavior. This plot is for no tuning ($\alpha = 0$) with the naive parameter choice $\sigma = \omega = \theta = \eta = 1$. The constraint error for w_2 (wire vs. no-wire) quickly drops to zero, but the others stay at roughly constant nonzero values. Clearly this is an instance of the burden of constraint satisfaction not being shared among all the variable types. The result is that the search is stuck on a non-solution.

In the top right plot, we again initialize the metric parameters to 1 but now we set $\alpha = 10^{-4}$ to allow the four parameters to be slowly tuned as the search progresses. One can see that the search does not get stuck the way it did with $\alpha = 0$, but the four variable types do not always move in concert. The $w_{1,2}$ and x errors become highly correlated, but the y error vacillates between much larger values and much smaller values while the others stay relatively stable. This could be in part because there are fewer y variables than there are w's or x's, since y's are associated with nodes whereas w's and x's are associated with edges. It is natural to expect the relative fluctuations to be larger for the smaller group of variables.

Nonetheless, we can still improve the correlation. In the bottom left plot, we promote ω to $\omega_{i\to j}$, σ to $\sigma_{i\to j}$, and so on, to allow the metric parameters to vary also by location within the network. We still initialize the metric parameters to 1 and use $\alpha = 10^{-4}$. Here the four

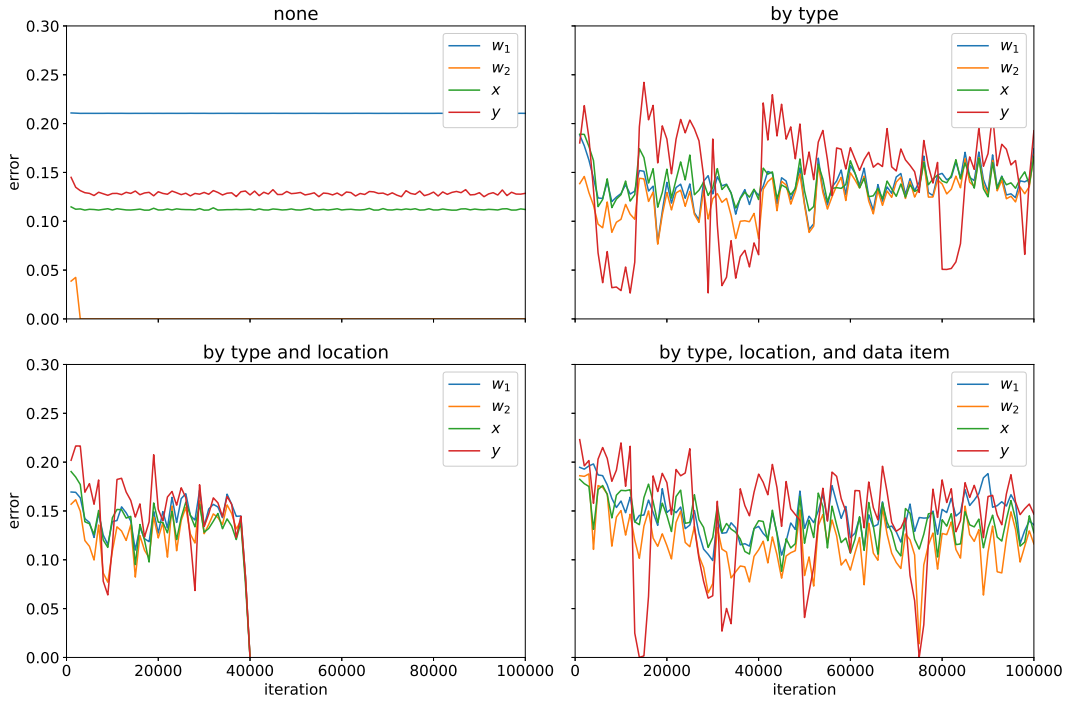

FIGURE 6. Time series of the constraint errors for a single run of the BGN problem with four levels of metric parameter tuning. *Top left:* no tuning. *Top right:* tuning by variable type. *Bottom left:* tuning by type and location within the network. *Bottom right:* tuning by type, location, and data item. The four errors in each plot are those of the four variable types described in the text.

constraint errors appear more strongly correlated and the y error no longer fluctuates with much greater amplitude than the others. The search terminates early because it finds a solution after about 4×10^4 iterations.

Finally, in the bottom right we go one step further, promoting $\omega_{i\to j}$ to $\omega_{k,i\to j}$, $\sigma_{i\to j}$ to $\sigma_{k,i\to j}$, and so on, to allow the metric parameters to vary by data item as well. As always we initialize the metric parameters to 1, and we still use $\alpha = 10^{-4}$. The correlation is certainly better than it was with no tuning and somewhat better than with tuning by type alone, but no better than tuning by type and location.

We have also tried other tuning combinations; for instance, tuning by type and data item but not location performed no better than tuning by type alone. Among the many possible combinations, we conclude that tuning by type and location makes the greatest difference in the algorithm's performance.

One possible interpretation of these results is that metric parameters have tangible effects only when they target systematic or structural properties of the variables. The nodes of the BGN distinguish themselves by their in- and out-degrees, and also their distance from data constraints (imposed only at the output layer). We should therefore expect a metric-tuning benefit based on network location just as for the more obvious case of variables distinct by type (node vs. edge). In the case of variables differing only by the data index, the structural bias is much weaker. For

most of these variables the structural difference is only indirect, through the fixed-value data constraints at the output nodes.

An alternative viewpoint is that metric parameters facilitate symmetry-breaking behavior. For example, it may be important that nodes within the same layer are able to develop unique characteristics, even while the associated node and edge variables have a permutation symmetry. Likewise, some data items might pose a greater challenge to the circuit than others, and the metric parameters of their associated variables would serve to break that form of permutation symmetry. Although our survey of results is far from comprehensive, the absence of a noticeable benefit from tuning by data index leads us to believe that the structural hypothesis is better supported than symmetry breaking.

5. Conclusions

We suspect that quite a few applications of iterative projection methods on nonconvex problems may have been abandoned because of a failure to recognize a sensitivity to parameters. Even in the case of hyperparameters, a setting in a range that is "safe" with respect to local convergence (where the constraints may be approximated as convex) overlooks the fact that these parameters also have a profound effect on the global characteristics of the search. We saw an example of this in the Douglas-Rachford generalization of section 3, where δ needs to be tuned to a sweet spot for effective search.

The tuning of metric parameters is equally important in applications where there is a rescaling freedom among variables not subject to symmetry. Such applications, without the translation or permutation symmetries of phase retrieval or a sudoku puzzle, are relatively new for iterative projection methods. Recognizing the role of the metric in such applications, and not arbitrarily assigning 1 as the relative scale, is an important first step. One can tune the metric parameters by hand, but here we have introduced a method for updating them automatically based on constraint errors. Actively updating the metric during the search does not disturb the convergence of the algorithm, as long as the updates are adiabatically slow. The automatic update approach is particularly useful in applications in which the naive metric fails and there are too many metric parameters to tune by hand.

Acknowledgments

The authors thank Avinash Mandaiya for noticing the two variable types in the dominating set problem and Jim Sethna for useful conversations.

References

[1] V. Elser, The complexity of bit retrieval, IEEE Trans. Info. Theory 64 (2017), 412-428.
[2] F.J.A. Artacho, J.M. Borwein, M.K. Tam, Global behavior of the Douglas–Rachford method for a nonconvex feasibility problem, J. Global Optim. 65 (2016), 309-327.
[3] V. Elser, Matrix product constraints by projection methods, Journal of Global Optimization 68.2 (2017), 329-355.
[4] F.J.A. Artacho, R. Campoy, M.K. Tam, The Douglas–Rachford algorithm for convex and nonconvex feasibility problems, Math. Meth. Oper. Res. 91 (2020), 201-240.
[5] T. Zamfirescu, The nearest point mapping is single valued nearly everywhere, Archiv der Math. 54 (1990), 563-566.
[6] J.R. Fienup, Phase retrieval algorithms: a comparison, Appl. Opt. 21 (1982), 2758-2769.

[7] V. Elser, T.-Y. Lan, T. Bendory, Benchmark problems for phase retrieval, SIAM J. Imaging Sci. 11 (2018), 2429-2455.
[8] F.J.A. Artacho, J.M. Borwein, M.K. Tam, Recent results on Douglas–Rachford methods, Serdica Math. J. 39 (2013), 313-330.
[9] V. Elser, Learning without loss, Fixed Point Theory Algorithms Sci. Eng. 2021 (2021), 1-51.
[10] S. Gravel, V. Elser, Divide and concur: A general approach to constraint satisfaction, Phys. Rev. E 78 (2008), 036706.
[11] D. Mitchell, B. Selman, H. Levesque, Hard and easy distributions of SAT problems, AAAI Vol. 92, 1992.
[12] E. J. Cockayne, Chessboard domination problems, Discrete Math. 86 (1990), 13-20.
[13] V. Elser, Reconstructing cellular automata rules from observations at nonconsecutive times, Phys. Rev. E 104 (2021), 034301.