Principled Evolutionary Algorithm Search Operator Design and the Kernel Trick

Fergal Lane, R. Muhammad Atif Azad and Conor Ryan
CSIS Department, University of Limerick
Ireland
Email: {Fergal.Lane,Atif.Azad,Conor.Ryan}@ul.ie

***Abstract*—Configuring an Evolutionary Algorithm (EA) can be a haphazard and inefficient process. An EA practitioner may have to choose between a plethora of search operator types and other parameter settings. In contrast, the goal of EA *principled design* is a more streamlined and systematic design methodology, which first seeks to better understand the problem domain, and only then uses such acquired insights to guide the choice of parameters and operators.**

We introduce a new approach to principled design of EAs based on *kernel methods*. Several popular machine learning and data analysis paradigms, which have been successfully applied to a wide range of difficult real world problems, would fall under this kernel umbrella. We demonstrate how kernel functions, which capture useful problem domain knowledge, can be used to directly construct EA search operators.

We test two kernel search operators on a suite of four challenging combinatorial optimization problem domains. These novel kernel search operators exhibit superior performance to some traditional EA search operators.

I. Introduction

EA configuration continues, in practice, to be a largely empirical affair. Practitioners can often be confronted with a bewildering array of possible configuration options. Typically, some subset of configuration possibilities is chosen and, then, compared and tuned using many EA runs. The broad approach of EA *principled design* is to first better understand the *problem domain* (PD) being optimized. Then, insights gained can be potentially used in a more focused, informed and efficient design process.

This paper introduces a novel approach to principled EA design. Our method begins by first finding a *kernel function* that matches the inherent statistical characteristics of the PD at hand. We then use this kernel function, which captures this PD knowledge, to directly construct EA search operators. The core strategy of kernel methods is the so-called *kernel trick* [1]. This allows primarily linear algorithms, which principally operate using inner products, to be extended to implicitly and cheaply operate in richer and higher-dimensional kernel feature spaces $\mathcal{V}$ where the original problem is easier to linearly separate and/or model.

A. Layout

Section 2 begins by giving a brief overview of the literature on the principled design of EAs and, then, relates our new kernel approach to this previous work.

Section 3 provides a basic introduction to kernel methods. We lay out the key concepts behind the "kernel trick". Merely having a kernel automatically allows us, for search space points, to calculate inner products, norms, distance and squared distance operations in the implicit transformed kernel space. These basic kernel operations are already sufficient to allow us to construct kernel equivalents of standard Gaussian and other mutation operators.

Somewhat more sophisticated kernel operations are needed for constructing crossover operators. The point of the kernel trick is that we can avoid working with huge feature space coordinate vectors. However, crossover operators often involve mixing or recombining parent coordinate values in various ways. In section 4, we construct, using basic linear algebra, partial coordinate systems for hyperplanes intersecting the n parent points in kernel feature space $\mathcal{V}$.

These kernel parent coordinate systems are then employed in section 5 to construct kernel *crossover-mutation* operators.

In section 6 we describe our basic experimental setup and detail the suite of combinatorial optimization problem domains we used to test our search operators. Section 7 gives results and an analysis of these experiments. Finally, in section 8, we set out our conclusions and describe some potential avenues for future work.

II. Principled EA Design

This section surveys past work relating to the principled design of EAs. A broad recent overview of work in this area would be [2], which gathers together representative papers from most of the major strands of work in this area.

The so-called *No Free Lunch* (NFL) results [3] would form the fundamental backdrop for this whole field emphasizing that EA analysis and design cannot take place in a vacuum, but only have meaning in the specific context of a given problem domain.

A. Exploiting Problem Domain Structure

Several past authors have sought to exploit problem domain structure to produce better search operators and/or improve general EA performance.

Peter Stadler's pioneering the use of *graph Laplacians* [4] and related tools for analyzing combinatorial optmization fitness landscapes has resulted in significant insights into why

some locality structures (and corresponding search operators) perform better than others [5].

Radcliffe and Surry's work on *forma theory* [6] defined and developed a generalized and extended type of schema called a *forma*. Once one has found a suitable set of basic formae for a search space, a set of intuitive design principles: respect, transmission, and assortment [7] can be applied. This results in natural generalizations of traditional EA bitstring search operators to a wide variety of non-standard problem representation types.

Competent GAs [8] would be another good example of a systematic attempt to exploit problem specific knowledge in EAs, in this case in large part by learning dependencies and linkages between variables.

B. Moraglio's Geometric Theory of EAs

The most closely related approach to our work would be Moraglio's "geometric theory of EAs" [9] where, once one has a suitable search space metric for a problem domain, balls and line segments generated by the associated topology can be used to construct geometric mutation and crossover operators. Moraglio successfully applied this approach to a wide variety of problem and search space types. He has also developed geometric generalizations of Particle Swarm Optimization, Differential Evolution and the Nealder-Mead algorithms [10].

Moraglio has also previously considered his "geometric theory of EAs" from a Gaussian Random Function (GRF) viewpoint [11]. He speculated that GRFs (kernel-based regression models) could be a useful bridge between EA theory and practice. While Moraglio did not use GRFs to actually design EAs or construct search operators, on several GRF test problem examples he showed that the smoothness of the problem kernel functions was useful in predicting how well some traditional search algorithms and search operators would perform.

Our kernel-based approach broadly follows the same general design philosophy. Moraglio used appropriate metrics to design search operators; we use suitable kernel functions. Kernel functions do indeed result in a metric space. However, they also generate additional inner product space mathematical machinery such as norms, bases and coordinate systems. A further advantage is that kernel functions can be used to perform inference. Significantly, rigorous evidence-based Bayesian and other ML frameworks can be used to learn suitable kernels for particular problem domains.

C. Gaussian Random Functions and EAs

GRFs have almost entirely been used to augment rather than directly design EAs. For example, GRFs have been one of the more popular models to be used as EA fitness function surrogates, particularly for expensive-to-evaluate fitness functions. Recent state-of-the-art examples would include [12] and [13]. Typically, a GRF surrogate model (with associated kernel function) is used to augment an existing standard EA. In contrast, our work seeks to incorporate the knowledge represented by such models into the design of the search operators themselves. Eventually, in future work, this will lead to a more seamless integration between EAs and kernel-based surrogate models being used to augment them.

Another family of optimizers that use GRFs are *Bayesian Optimization* (BOPT) algorithms within the global optimization field (this subfield goes back over 40 years). BOPT algorithms are optimization algorithms that maintain a Bayesian GRF machine learning model of the fitness function being optimized, which is continually updated as new points are evaluated (the EGO algorithm [14] would be the most popularly used example in this field).

There seems to be only a single previous example of work that directly used a kernel method for principled EA design. This research [15] dates back more than 15 years. In this, William Macready looked at designing mutation operators for a Gaussianized form of NK-landscapes, which closely approximated a GRF with an exponential kernel function.

Macready's approach, at each step, estimated the likely payoffs (using a GRF model) of sampling the remaining budget of N points at the various possible Hamming distances from the single parent point (predicting an "optimal search distance" for the resulting mutation operator). His mutation operator then simply always randomly sampled a point at this "optimal search distance". This distance would vary as the run progressed, typically becoming shorter-range and more exploitative.

Macready's work, though quite different to our current approach, was preliminary but interesting. His payoff estimation method was cheap but very approximate (more accurate predictors would be possible) and only mutation was considered (extensions to multi-parent search operators would not be that difficult). Surprisingly, this promising research was never developed further. We hope to build upon this approach in future work.

III. Basic Kernel Operations

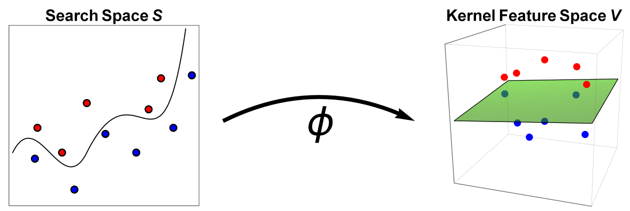
This section gives an introduction to basic kernel concepts.

A. The Kernel Trick

Algorithms such as linear regression and *principal component analyis* (PCA) primarily operate using inner products in the original representation space and assume an underlying linear model for the problem/data being tackled. In principle, such linear algorithms can be extended to cope with non-linear problems/data using an explicit transform $\Phi : \mathcal{S} \rightarrow \mathcal{V}$ from the original search space $\mathcal{S}$ to some (usually higher dimensional) kernel feature space $\mathcal{V}$ where the original problem is better linearized. This approach means one has to explicitly calculate inner products for potentially huge (even infinite dimensional) and cumbersome coordinate feature vectors

The *kernel trick* [1] is where one instead uses only a kernel function $k(s, t)$ (with general form $k : \mathcal{S} \times \mathcal{S} \rightarrow \Re$) that directly gives the inner product between any two search space points s and t in some transformed kernel feature space $\mathcal{V}$. One only implicitly works in these higher dimensional spaces via cheaply-computed kernel functions (see Figure 1). For

Figure 1. The Kernel Trick



example, a standard linear classifier that could not effectively separate data in the original space might successfully separate these in some higher dimensional feature space via a kernel; this is the basis of *Support Vector Machine* techniques [16] in classification and *Gaussian Random Functions* (GRFs) [17] in machine learning (also known as *Gaussian Processes*).

B. What are Kernel Functions?

Kernel functions are functions with the form $k : \mathcal{S} \times \mathcal{S} \rightarrow \Re$. Associated with every kernel function is a transform $\Phi : \mathcal{S} \rightarrow \mathcal{V}$ from the original search space $\mathcal{S}$ to some (usually higher or infinite dimensional) kernel feature space $\mathcal{V}$ (which is a Hilbert space):

$$\Phi(s) = \{\phi_1(s), \phi_2(s), \cdots\}, s \in \mathcal{S}, \phi_i(s) \in \Re \, \forall i$$

The kernel function $k(s,t)$ gives the inner product between any two search space points s and t in this transformed kernel feature space $\mathcal{V}$:

$$k(s,t) = \langle \Phi(s), \Phi(t) \rangle_V = \sum_i \phi_i(s)\phi_i(t)$$

Not any function can be used as a kernel function. Such functions must be continuous, symmetric and also satisfy a technical non-negative definiteness condition. Once this is satisfied, however, Mercer's theorem [18] (and its generalizations) guarantee that there will always be a transform Φ from $\mathcal{S}$ to some associated Hilbert space $\mathcal{V}$ in which $k(s,t)$ calculates the inner product. However, generally with kernel methods, there is no need to explicitly know about or actually use the kernel feature space and mapping that is implicitly associated with a kernel function.

As well as being a type of non-linear computational short-cut, kernel functions can also be viewed as being fitness similarity functions. Many stochastic search algorithms are based on the notion that *similar* search space points are likely to have *similar* fitness values. A kernel function is a particular concrete and explicit model of such fitness similarity for a problem domain. A kernel function, for any two search space points, gives a value indicating how similar their fitness values are likely to be. To be more precise, a kernel function should capture the fitness covariances between points in the search space. The ideal kernel function would equal:

$$k(s,t) = \text{cov}(f(s), f(t)) \quad (1)$$

where the covariance is calculated by averaging over all fitness functions f sampled from the problem domain under consideration.

Theory (the Karhunen-Loeve theorem [18]) indicates that such a choice of kernel function best linearizes the problem domain when mapped to the associated kernel feature space. With such an accurate kernel model, we can take any two points s and t in the search space and predict whether their fitnesses are likely to be similar (highly correlated), unrelated (independent) or dissimilar (negatively correlated).

Rigorous evidence-based machine learning techniques can be used to efficiently learn suitable kernel functions that capture such PD behaviour [17]. Typically, samples of points and their fitnesses are taken from fitness functions from the problem domain being studied. Bayesian or maximum likelihood techniques are then used to find a kernel function that most closely matches this evidence (*DiceKriging* [19] is a good example of a software library, written in R, implementing many such techniques). Usually, the first step in this process is to find a kernel function family that broadly matches general PD characteristics, for example, the degree of smoothness or ruggedness of the PD's fitness functions. Then, associated kernel function hyperparameters are tuned to give a final closely-fitting kernel function.

In some cases, where the actual form of the fitness function generator is known, the kernel function can be explicitly calculated from first principles (conveniently this is the case for the combinatorial optimization problem domains we use later in this paper).

In practice there are common families of popular kernel functions in the GRF and SVM fields, which seem to suffice for most practical problems. See [20] for a discussion of these issues and a fairly comprehensive compendium of almost all commonly-used kernel function families and their properties.

If the problem domain mathematical form is explicitly known (as is the case for the combinatorial optimization problem domains we use later), then sometimes the associated kernel functions can be directly calculated from first principles.

C. Basic Kernel Operations

As we've just seen, a kernel function calculates the inner product between two search space points transformed into some kernel feature space. However, several other basic operations in the feature space; norms, squared distances and distances, can also be expressed in terms of inner products. Table I lists these, their form in terms of kernel feature space coordinates, how to calculate them using kernel functions, and finally how they relate to actual fitness values and their moments.

$d(s,t)$ is, in effect, Euclidean distance in the kernel feature space $\mathcal{V}$. This is a well know metric in GRF theory, known as the *canonical metric* [18].

D. Kernel Mutation

With these basic kernel operations, we already have enough tools to kernelize some well-known mutation operators. We

Table I
KERNEL OPERATIONS

Operation	Symbol	Kernel Feature Space View	Kernel Function Implementation	Fitness Representation
Inner Product	$\langle s,t\rangle$	$\langle\Phi(s),\Phi(t)\rangle=\sum_i\phi_i(s).\phi_i(s)$	$k(s,t)$	$\mathrm{cov}(f(s),f(t))$
Norm	$\parallel s\parallel$	$\sqrt{\langle\Phi(s),\Phi(s)\rangle}=\sqrt{\sum_i\phi_i(s)^2}$	$\sqrt{k(s,s)}$	$\sqrt{\mathrm{var}(f(s))}=\sqrt{\mathrm{cov}(f(s),f(s))}$
Squared distance	$d^2(s,t)$	$\sum_i(\phi_i(s)-\phi_i(t))^2=\sum_i\phi_i(s)^2+\phi_i(t)^2-2\phi_i(s)\phi_i(t)$	$k(s,s)+k(t,t)-2k(s,t)$	$(f(s)-f(t))^2$
Distance	$d(s,t)$	$\sqrt{\sum_i(\phi_i(s)-\phi_i(t))^2}$	$\sqrt{k(s,s)+k(t,t)-2k(s,t)}$	$\sqrt{(f(s)-f(t))^2}$

need to do little more than substitute the usual Euclidean or squared Euclidean distance with their kernel distance counterparts.

Since the kernel feature space $\mathcal{V}$ is a real-valued Hilbert space (essentially a vector of real numbers), search operators from real-coded GAs are particularly relevant for our purposes. Perhaps the most common mutation operator for this type of GA is Gaussian mutation [21], where a multivariate Gaussian density centred on the current parent point p is used to generate a new child point c. The kernel equivalent of this mutation operator is constructed by simply replacing the ordinary squared Euclidean distance $\mathcal{E}(p,c)$ between p and c with its kernel equivalent $d(p,c)$ (as defined in Table I). Hence, the probability density for *Gaussian Kernel Mutation* (GKM) would:

$$\mathcal{P}_{\mathrm{GKM}}(c)\propto\exp\left(-\frac{d^2(p,c)}{2\sigma^2}\right)$$

where:

$$d^2(p,c)=k(p,p)+k(c,c)-2k(p,c)$$

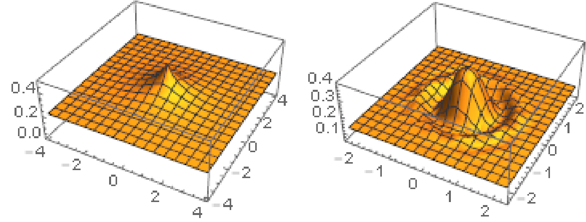
Kernel versions of Cauchy and other similar mutation distributions can be defined in an analogous fashion.

1) Kernel Mutation Examples: The probability densities of these mutation operators will very much reflect the shapes of the kernel functions being used. Many commonly used kernel functions simply decay monotonically with distance. Hence, unsurprisingly, the corresponding kernel mutation operators also do likewise (as can be seen in the example using the popular exponential kernel function $k(s,t)=\exp(-\mathcal{E}(s,t))$ in the leftmost graph in Figure 2). However, kernel functions are not always monotonic. The rightmost graph in Figure 2 gives a similar example using the more complex oscillating kernel function $k(s,t)=\exp(-\mathcal{E}(s,t))\cos(3\,\mathcal{E}(s,t))$. The resulting kernel mutation operator still manages to incorporate and mirror this oscillating pattern.

IV. KERNEL COORDINATE SYSTEMS

With the help of some basic matrix manipulation, kernel operations even more complex than those that appear in Section III-C are possible (involving linear combinations, hyperplanes and spanning sets). A key advantage of kernel methods is not having to explicitly deal with (potentially huge) kernel feature space coordinate vectors. However, many standard

Figure 2. Gaussian Kernel Mutation with Exponential and Oscillating Kernel Functions



crossover search operators work by recombining/mixing the genetic material of existing parent points using some coordinate system. We can have the best of both worlds, though, by using kernel functions to construct *parent coordinate systems*, which give locations relative to some set P of n parent points $P=\{p_1,...p_n\}$, $p_i\in\mathcal{S}$. These are $O(n)$-dimensional coordinate systems in $\mathcal{V}$ for hyperplanes that intersect the n parent points.

For search operator design, two particularly useful hyperplane constructs are the *parent spanning set* $\mathcal{S}_P$ (the set of all possible linear combinations of the parent set coordinates in $\mathcal{V}$) and the parent hyperplane $\mathcal{H}_P$ (the hyperplane in $\mathcal{V}$ that intersects the n parent points).

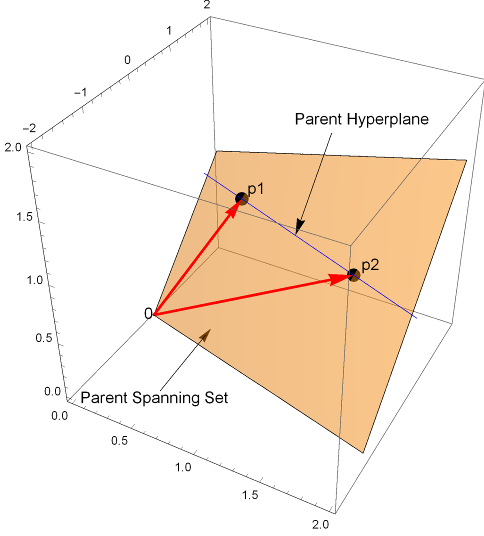
We can construct orthonormal coordinate systems for such hyperplanes and spanning sets, and also calculate the distances of arbitrary points to these sets. We develop in this section n-dimensional partial coordinate systems that span the *parent spanning set*:

$$S_P=\mathrm{span}(P)=\left\{\sum w_ip_i\in\mathcal{V}\,|\,p_i\in P,\,w_i\in\Re\right\}$$

A. Parent Hyperplanes and Simplices

S_P can also be viewed as the unique n-dimensional hyperplane that intersects the n parents points and the origin 0 coordinate in the kernel feature space. The *parent hyperplane* H_P is the $(n-1)$-dimensional hyperplane that intersects just the n parent points. Figure 3 illustrates this for the two parent case. Here, S_P is in 2D and $\mathcal{H}_P$ is a line through the two parent points. The *parent simplex* $\triangle_P$ is then, in this case, just the line segment connecting the two parent points.

Figure 3. Parent hyperplane and spanning set for two parents



B. Parent Coordinate Systems

The simplest valid coordinate system for S_P can be constructed by using the vector of inner products with the parent points in P. Suppose $K_{P,P}$ is the $n \times n$ matrix of kernel inner products between the parents: $(K_{P,P})_{i,j} = k(p_i, p_j)$, and $K_{P,c}$ is the n-dimensional column vector of their inner products with some arbitrary point $c \in \mathcal{S}$: $(K_{P,c})_i = k(p_i, c)$.The coordinates for some $c \in S_P$ would then be given by the n-dimensional column vector: $[c]_K = K_{P,c}$ where $(K_{P,c})_i = k(p_i, c)$. We term this the *kernel coordinate system.*

A potentially more useful alternative would be the *parent coordinate system*, which uses the parent vectors themselves as the basis vectors. The coordinates for a point $c \in S_P$ given by this system $[c]_P = \{c_1, \ldots, c_n\}^T$ correspond to its weights when expressed as a linear combination of parent points: $c = \sum_{i=1}^{n} c_i \, p_i$ in $\mathcal{V}$. Some linear algebra is needed to solve for these c_i. It turns out that $[c]_P = K_{P,P}^{-1} K_{P,c}$.

How do we compute inner products $k_{S_P}(u, v)$ between points u and v in S_P when using their *parent* coordinates $[u]_P$ and $[v]_P$? These can be computed using weighted sums of inner products between parents:

$$k_{S_P}(u, v) = k(\Sigma_i u_i p_i, \Sigma_j v_j p_j) = \Sigma_{i,j} u_i v_i k(p_i, p_j) = [u]_P^T K_{P,P} [v]_P$$

What about when we are dealing with points x and y when using their *kernel* coordinates $[x]_K$ and $[y]_K$? We've already seen that $[x]_P = K_{P,P}^{-1} [x]_K$. Hence:

$$k_{S_P}(x, y) = [x]_P^T K_{P,P} [y]_P = \left(K_{P,P}^{-1} [x]_K\right)^T K_{P,P} \left(K_{P,P}^{-1} [y]_K\right) = [x]_K^T K_{P,P}^{-1} [y]_K$$

When both points x and y are in S_P, these $k_{S_P}(x, y)$ calculations will always coincide exactly with $k(x, y)$. The overall $k(x, y)$ kernel inner product consists of two components: $k(x, y) = k_{S_P}(x, y) + k_{S_P^\perp}(x, y)$. $k_{S_P}(x, y)$ gives the inner product *inside* of S_P. $k_{S_P^\perp}(x, y)$ gives the inner product *outside* of S_P in the parent spanning set's *orthogonal complement* $S_P^\perp$. Suppose the kernel feature space $\mathcal{V}$ is N-dimensional. We've effectively re-oriented its coordinate system, reserving n coordinates for S_P. $(N - n)$ extra coordinates will still be needed to span $\mathcal{V}$. For points in S_P these extra coordinates are always 0 and can be ignored. Outside of S_P, however, these extra coordinates may be non-zero.

Fortunately, for our purposes, we don't actually need to explicitly work with these (potentially huge number of) extra coordinates. For example, we can easily calculate c's squared norm in $S_P^\perp$ as the squared norm over the full kernel feature space $||c||^2 = k(c, c)$ minus its squared norm within our partial coordinate system $||c||_{S_P}^2 = k_{S_P}(c, c)$, which equals[1]:

$$\left\|c\right\|_{S_P^\perp}^2 = k(c, c) - K_{P,c}^T K_{P,P}^{-1} K_{P,c} \tag{2}$$

This actually gives the squared kernel distance from any point c to S_P. Calculations for other basic $S_P^\perp$ kernel operations can be performed similarly via subtraction.

1) An Orthonormal Coordinate System: The most attractive types of bases are orthonormal ones. For this, we seek a basis set:

$$\Psi_E = \{e^{(1)}, \ldots, e^{(n)}\} = ZP = \left\{ \begin{array}{c} z_{1,1} p^{(1)} + \cdots + z_{1,n} p^{(n)} \\ \cdots \\ z_{n,1} p^{(1)} + \cdots + z_{n,n} p^{(n)} \end{array} \right\}$$

that spans S_P but whose elements have norms of 1, i.e. $\left\|e^{(i)}\right\| = k(e^{(i)}, e^{(i)}) = 1 \, \forall i$, and are mutually orthogonal/perpendicular, i.e. $\left\langle e^{(i)}, e^{(j)} \right\rangle = k(e^{(i)}, e^{(j)}) = 0 \, \forall i \neq j$.

The most obvious (though not only) choice of matrix Z that satisfies these requirements is $Z = K_{P,P}^{-\frac{1}{2}}$. Calculating coordinates in this *standard orthonormal* parent coordinate system can then be done as follows:

$$[c]_E = K_{P,P}^{-\frac{1}{2}} K_{P,c}$$

Z is calculated using the eigendecomposition of $K_{P,P}$: $K_{P,P} = QDQ^{-1}$where Q contains the eigenvectors as column vectors and D is a diagonal matrix of eigenvalues. Z then is $QD^{-\frac{1}{2}}Q^{-1}$. All vectors in the basis set must be linearly independent. (Near-)zero eigenvalues indicate (near-)collinearity between the parents. In such cases we can also use this decomposition to first remove any parents with such associated (near-)zero eigenvalues before constructing our coordinate system.

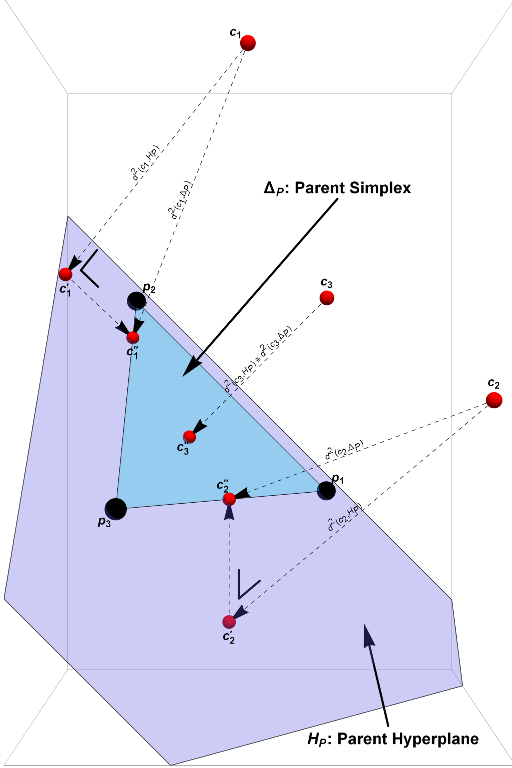
One advantage of such orthonormal coordinate systems is that, for points in S_P, they are always consistent with the basic kernel operations described in section III-C. If we have two points q and r, which are in the parent spanning set S_P, and their coordinates $[q]_E = \{q_1, \ldots, q_n\}$ and $[r]_E = \{r_1, \ldots, r_n\}$, then calculations of distances and dot

[1]This is actually the standard GRF formula for calculating predicted variance.

Table II
PARENT COORDINATE SYSTEMS

Coordinate System	Notation	Basis Set	Calculating Coordinates
Kernel	$[.]_K$	$\Psi_K = K_{P,P}^{-1} P$	$K_{P,c}$
Standard Orthonormal	$[.]_E$	$\Psi_E = K_{P,P}^{-\frac{1}{2}} P$	$K_{P,P}^{-\frac{1}{2}} K_{P,c}$
Parent	$[.]_P$	$\Psi_P = P$	$K_{P,P}^{-1} K_{P,c}$

Figure 4. Parent hyperplane and simplex for three parents



products etc. with these coordinates will match the equivalent ones using kernel functions given in table I. For example, the squared norm $||q||^2$ of point q in S_P can be calculated using its actual coordinate vector $[q]_E^T[q]_E = \sum_{i=1}^{n}(q_i)^2$ or equivalently:

$$(K_{P,P}^{-\frac{1}{2}} K_{P,q})^T (K_{P,P}^{-\frac{1}{2}} K_{P,q}) = K_{P,q}^T K_{P,P}^{-1} K_{P,q} \tag{3}$$

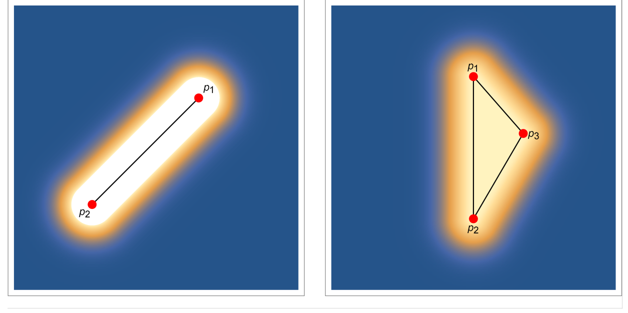
and will agree with the value given by the equivalent kernel function calculation: $||q||^2 = k(q,q)$.

See Table II for a summary of some of the properties of these three coordinate systems.

V. KERNEL CROSSOVER

Such coordinate systems allows straightforward kernelized generalizations of some real-coded GA crossover operators. We implemented two such possible kernel crossover-mutation operators based on the distance from a child point c to either the parent hyperplane $\mathcal{H}_{\mathcal{P}}$ or parent simplex $\triangle_P$ in $\mathcal{V}$ (see figure 4 for an illustration of these concepts for the three parent case).

Figure 5. Kernel simplex crossover-mutation densities in S_P with two and three parents.



Our Kernel Hyperplane (KH) operator was based on the squared kernel distance $d^2(c, \mathcal{H}_P)$ between a child point c and the parent hyperplane $\mathcal{H}_P$ in $\mathcal{V}$. This had the following search operator density form:

$$\mathcal{P}_{\mathcal{KH}}(c) \propto \exp\left(-\frac{d^2(c,\mathcal{H}_P)}{\sigma^2}\right), \sigma > 0 \tag{4}$$

The σ parameter determines how severely departures from the parent hyperplane within $\mathcal{V}$ are penalized.

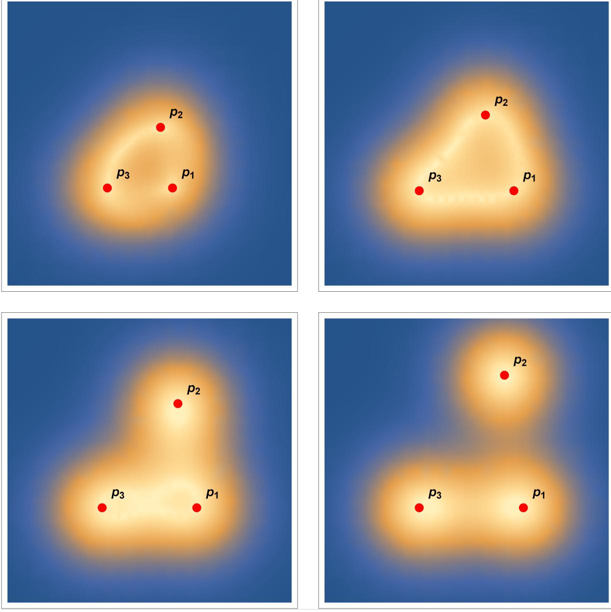
Our Kernel Simplex (KS) operator was based on the kernel distance $d^2(c, \triangle_P)$ between a child point c and the parent simplex $\triangle_P$ in $\mathcal{V}$. This has a similar search operator density form to equation 4 (except distance to $\triangle_P$ rather than $\mathcal{H}_P$ was used). These two kernel search operators combined both crossover and mutation in a single operator. In our experiments we tested the two-parent ($n = 2$) case.

$d^2(c, \triangle_P)$ can be calculated as sum of the squared distance $d^2(c, \mathcal{S}_p)$ from any point c to its closest equivalent in $\mathcal{S}_P$ and then the squared distance within $\mathcal{S}_P$ from this to $\triangle_P$. $d^2(c, \mathcal{S}_p)$ can be calculated using equation 2 as $d^2(c, \mathcal{S}_p) = k(c,c) - K_{P,c}^T K_{P,P}^{-1} K_{P,c}$. Once within $\mathcal{S}_P$, we can use the orthornomal coordinate system Ψ_E, described in section IV-B1. Standard geometric techniques can be used to calculate the distance of a point to a simplex within $\mathcal{S}_P$ (a suitable algorithm can be found in [22]). $d^2(c, H_P)$ can similarly be decomposed as the sum of the squared distance from c to $\mathcal{S}_P$ and then the squared distance, within $\mathcal{S}_P$, to $\mathcal{H}_P$.

Figure 5 gives concrete examples in S_P of some such combined crossover-mutation densities (implementing these operators with higher numbers of parents is possible). The underlying simplex nature of these densities can be very strongly seen in S_P and the kernel feature space. Points outside of these simplices, however, can still be generated.

This simplex shape may not always be as evident when the search operator density is mapped back to the original search space. Figure 6 gives an example of what a density for these search operators (with three parents) actually looks like in the search space itself. A Gaussian kernel function $k(s,t) = \exp(-\frac{5}{2}\mathcal{E}(s,t)^2)$ is used with $\sigma = \frac{1}{5}$. When the three parent points are close together and interacting strongly, a simplex shape can actually be seen. As the parent points draw further away from each other, though, interactions between

Figure 6. Search space densities for kernel simplex crossover-mutation using a Gaussian kernel function.



the parents become weaker, essentially eventually defaulting to just mutation around each parent.

These operators were implementing by constructing search operator density tables, partitioning the binary search spaces according to Hamming distances between and from the set of parent points. This allowed us to efficiently sample points according to the desired search operator density.

VI. Experimental Setup

A. EA Parameter Settings

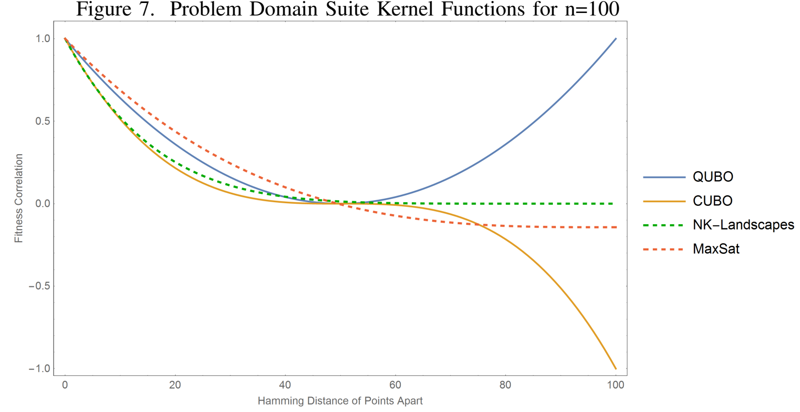
We used an EA with a population size of 150 running for 100 generations. Selection was carried out using tournament selection (with size 2). Several traditional EA bit string crossover operators were tested, 1-point, 2-point and uniform crossover. We also used bit flip mutation. Batches of 100 runs were used to produce all experimental results given here.

B. Problem Domain Set

Four well-known combinatorial optimization problem domains with bit string search spaces were used; these are described in more detail in Table III. One key attraction of the problems chosen for this suite is the actual kernel functions for these problems can be directly calculated from first principles, avoiding the need for introducing kernel fitting methodologies. These kernel functions are all isotropic, that is their values only depend on the Hamming distance $h = H(x, y)$ between bit strings x and y. Three different values for n, the bit string length, were used in simulations: $n =$ 25, 50 and 100.

These kernel functions are plotted (for the case n=100) in Figure 7. We can see from these plots that, for all our problem domains, the average fitness correlation between search space points tends to weaken at various rates (even becoming negative in the CUBO case) as Hamming distance

Figure 7. Problem Domain Suite Kernel Functions for n=100



between them increases. In the QUBO case, though, fitness correlation eventually begins to increase again because, for this problem domain, bitstrings and their complements always have identical fitness values.

Both our KS and KH crossever-mutation operators can actually be viewed, in a certain sense, as a generalization of uniform crossover and bit-flip mutation. The simple but popular toy binary weighted "counting ones" (OneMax) problem can be shown to have a linear kernel function: $k(x, y) = 1 - \frac{2\,H(x,y)}{n}$ when it has the form: $f(x) = \sum_{i=1}^{n} w_i(2x_i - 1)$, $w_i \sim \mathcal{N}(0, \frac{1}{n})$. Applying our kernel design methodology to this case, the KS and KH search operators turn out to be both equivalent to a combination of uniform crossover and bit-flip mutation (with the σ parameter determining the bit-flip mutation rate). Hence, our new kernel operators can be viewed as generalizations of traditional uniform crossover-bitflip mutation operators that have been tailored to the problem domain at hand using the knowledge encapsulated in a kernel function. Using these traditional search operators as a baseline comparison is, therefore, rather appropriate.

VII. Results

We aimed for a fair like-for-like comparison of the kernel simplex operator with some traditional EA search operators (uniform, 1-point and 2-point crossover; each of these used in combination with bit flip mutation). All EA configuation settings used in our experiments were identical except for the bit flip mutation rate and crossover rate for the standard EAs, and for the σ parameter associated with our Kernel Simplex (KS) and Kernel Hyperplane (KH) search operators.

All of the standard search operators exhibited similar performance levels. Of these, uniform crossover was found to be marginally the most competitive; hence, we have included this as our main comparator here (summaries of test suite performances for the other crossover operators are given in Table IV).

Much effort was devoted to finding a single good all-round setting for these parameters that would optimize performance over the test suite. A crossover rate of 0.65 was found to maximize performance for uniform crossover. A bit flip rate

Table III
PROBLEM DOMAIN SET DETAILS

Problem Domain	References	Fitness Function Formula/Details	Kernel Function
QUBO (Quadratic Unconstrained Binary Optimization)	[23]	$f(x)=\sum_{i,j=1}^{n} w_{ij}(2x_i-1)(2x_j-1)$, $w_{ij}\sim\mathcal{N}(0,\frac{1}{n^2})$.	$\left(1-\frac{2\,h}{n}\right)^2$
CUBO (Cubic Unconstrained Binary Optimization)	[24]	$f(x)=\sum_{i,j,k=1}^{n} w_{ijk}(2x_i-1)(2x_j-1)(2x_k-1)$, $w_{ij}\sim\mathcal{N}(0,\frac{1}{n^3})$	$\sum_{u=0}^{3}(-1)^u\binom{3}{u}\left(\frac{h}{n}\right)^u\left(1-\frac{h}{n}\right)^{(3-u)}$
NK-Landscapes	[25]	"Random neighbourhood" model (without replacement); $K=5$	$\frac{\binom{n-h}{K+1}}{\binom{n}{K+1}}$
K-Uniform MAX-SAT (with $20\,n$ random clauses)	[26]	$K=3$ (variables per clause)	$\frac{\binom{n}{K}-2^K\binom{n-h}{K}}{(1-2^K)\binom{n}{K}}$

of $\frac{2}{n}$, where n is the bit string size, was found to gave the best all-round test suite performance for bit flip mutation.

A setting of $\sigma=\frac{1.75}{n}$ produced the best average test suite performances for both the KS and KH operators.

The relative performances of the KS operator over Uniform Crossover (UX) can be seen in Table V. The fitness distributions of these problem domains were all scaled to have zero means and unit variances; hence, it is not surprising that typical maximum fitness values are similar across our test suite. Generally, the KS operator performed better than the UX operator. The mean performance advantage over the entire test suite was 7.2%. This advantage becomes more marked with increasing dimensionality. The QUBO and MAX-SAT problem domains are smoother, less rugged and probably inherently easier than the CUBO and NK-landscape ones. This most likely explains the performance gap, particularly at lower dimensions for these problems (all search algorithms seem to perform well for these easier cases).

Table VI gives the relative performances of the KH search operator. Its overall test suite performance was slightly worse than that of the KS operator. Its performance margin over the UX operator was 6.8%. The overall test suite performance gaps between both KH and KS operators over the UX operator *were* significant at a 99% confidence level using the two-tailed Student's t-test.

VIII. CONCLUSIONS AND FUTURE WORK

These preliminary results show that this novel kernel-based EA principled design approach is worthy of further study. These new operators exhibit a moderate but significant performance gain. We will, of course, need to test this method on a much wider range of problem domain types.

Many other kernel search operator forms and generalizations of standard EA search operators are possible. Kernel functions automatically come with a plethora of very useful and problem domain tailorable vector space machinery. Our kernel search operators did not exploit parent fitness information. Scaling up kernel search operators to use multiple parents should also be straightforward. The resulting exploitation of far more parent position and fitness information will hopefully lead to even better performances.

Other EA components will also be amenable to kernelization, e.g. diversity preservation mechanisms such as sharing.

A potentially very promising avenue of research is the application of the kernel trick to *Estimation of Distribution Algorithms* (EDAs). Early EDA algorithms, e.g. PBIL [27], used simple linear probabilistic population models. Later algorithms, e.g. [28], used more complex Bayesian network models to go beyond simple linearity. Kernel methods, however, represent an unexplored alternative route to a whole new class of non-linear EDAs.

REFERENCES

[1] A. Aizerman, E. M. Braverman, and L. Rozoner, "Theoretical foundations of the potential function method in pattern recognition learning," *Automation and remote control*, vol. 25, pp. 821–837, 1964.

[2] Y. Borenstein and A. Moraglio, *Theory and Principled Methods for the Design of Metaheuristics*. Springer, 2014.

[3] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 1, no. 1, pp. 67–82, 1997.

[4] K. Klemm and P. F. Stadler, "Rugged and elementary landscapes," in *Theory and Principled Methods for the Design of Metaheuristics*, pp. 41–61, Springer, 2014.

[5] P. F. Stadler, "Landscapes and their correlation functions," *Journal of Mathematical chemistry*, vol. 20, no. 1, pp. 1–45, 1996.

[6] P. D. Surry and N. J. Radcliffe, "Formal search algorithms+ problem characterisations= executable search strategies," in *Theory and Principled Methods for the Design of Metaheuristics*, pp. 247–270, Springer, 2014.

[7] N. J. Radcliffe, "Forma analysis and random respectful recombination.," in *ICGA*, vol. 91, pp. 222–229, 1991.

[8] D. E. Goldberg, *The design of innovation: Lessons from and for competent genetic algorithms*, vol. 7. Springer Science & Business Media, 2013.

[9] A. Moraglio and R. Poli, "Topological interpretation of crossover," in *Genetic and Evolutionary Computation–GECCO 2004*, pp. 1377–1388, Springer, 2004.

[10] A. Moraglio and C. G. Johnson, "Geometric generalization of the nelder-mead algorithm," in *Evolutionary Computation in Combinatorial Optimization*, pp. 190–201, Springer, 2010.

[11] A. Moraglio and Y. Borenstein, "A gaussian random field model of smooth fitness landscapes," in *Proceedings of the tenth ACM SIGEVO workshop on Foundations of genetic algorithms*, pp. 171–182, ACM, 2009.

[12] D. Lim, Y. Jin, Y.-S. Ong, and B. Sendhoff, "Generalizing surrogate-assisted evolutionary computation," *Evolutionary Computation, IEEE Transactions on*, vol. 14, no. 3, pp. 329–355, 2010.

[13] B. Liu, Q. Zhang, and G. Gielen, "A gaussian process surrogate model assisted evolutionary algorithm for medium scale expensive optimization problems," *Evolutionary Computation, IEEE Transactions on*, vol. 18, pp. 180–192, April 2014.

Table IV
STANDARD SEARCH OPERATOR PERFORMANCES (MEAN BEST RUN FITNESS)

Problem Bit Length	$n = 25$			$n = 50$			$n = 100$		
Search Operator	UX	1-point Crossover	2-point Crossover	UX	1-point Crossover	2-point Crossover	UX	1-point Crossover	2-point Crossover
QUBO	4.80	4.77	4.71	6.77	6.67	6.81	9.26	8.90	9.05
CUBO	5.01	4.98	5.08	6.59	6.48	6.58	8.69	8.66	8.57
NK (K=5)	4.99	4.98	5.00	6.63	6.50	6.54	8.66	8.46	8.66
MAX-SAT (K=3)	4.45	4.39	4.45	6.24	6.15	6.15	8.55	8.18	8.30
Mean Test Suite Performance	4.81	4.78	4.80	6.56	6.45	6.52	8.79	8.55	8.62

Table V
KERNEL SIMPLEX SEARCH OPERATOR PERFORMANCES (MEAN BEST RUN FITNESS)

Problem Bit Length	$n = 25$		$n = 50$		$n = 100$	
Search Operator	Kernel Simplex	Uniform Crossover	Kernel Simplex	Uniform Crossover	Kernel Simplex	Uniform Crossover
QUBO	4.78 ± 0.03	4.80 ± 0.04	7.16 ± 0.03	6.77 ± 0.04	10.10 ± 0.03	9.26 ± 0.05
CUBO	5.33 ± 0.04	5.01 ± 0.04	7.44 ± 0.04	6.59 ± 0.04	10.07 ± 0.03	8.69 ± 0.05
NK (K=5)	5.30 ± 0.03	4.99 ± 0.04	7.26 ± 0.03	6.63 ± 0.05	9.75 ± 0.03	8.66 ± 0.06
MAX-SAT (K=3)	4.47 ± 0.03	4.45 ± 0.03	6.37 ± 0.03	6.24 ± 0.04	8.79 ± 0.03	8.55 ± 0.04
Mean Test Suite Performance	4.97 ± 0.02	4.81 ± 0.02	7.05 ± 0.02	6.56 ± 0.02	9.68 ± 0.02	8.79 ± 0.02

Table VI
KERNEL HYPERPLANE SEARCH OPERATOR PERFORMANCES (MEAN BEST RUN FITNESS)

Problem Bit Length	$n = 25$		$n = 50$		$n = 100$	
Search Operator	Kernel Hyperplane	Uniform Crossover	Kernel Hyperplane	Uniform Crossover	Kernel Hyperplane	Uniform Crossover
QUBO	4.80 ± 0.05	4.80 ± 0.04	6.95 ± 0.03	6.77 ± 0.04	10.03 ± 0.03	9.26 ± 0.05
CUBO	5.28 ± 0.03	5.01 ± 0.04	7.15 ± 0.05	6.59 ± 0.04	9.93 ± 0.03	8.69 ± 0.05
NK (K=5)	5.33 ± 0.03	4.99 ± 0.04	6.92 ± 0.05	6.63 ± 0.05	9.66 ± 0.03	8.66 ± 0.06
MAX-SAT (K=3)	4.43 ± 0.04	4.45 ± 0.03	6.53 ± 0.04	6.24 ± 0.04	8.72 ± 0.03	8.55 ± 0.04
Mean Test Suite Performance	4.96 ± 0.02	4.81 ± 0.02	6.99 ± 0.02	6.56 ± 0.02	9.58 ± 0.02	8.79 ± 0.02

[14] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *Journal of Global optimization*, vol. 13, no. 4, pp. 455–492, 1998.

[15] W. G. Macready, "Tailoring mutation to landscape properties," in *Evolutionary Programming VII*, pp. 355–364, Springer, 1998.

[16] N. Cristianini and J. Shawe-Taylor, *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.

[17] C. E. Rasmussen and C. Williams, "Gaussian processes for machine learning. 2006," *The MIT Press, Cambridge, MA, USA*, 2006.

[18] R. J. Adler and J. E. Taylor, *Random fields and geometry*, vol. 115. Springer, 2009.

[19] O. Roustant, D. Ginsbourger, and Y. Deville, "DiceKriging, DiceOptim: Two R packages for the analysis of computer experiments by kriging-based metamodeling and optimization," 2012.

[20] P. Abrahamsen, *A review of Gaussian random fields and correlation functions*. Norsk Regnesentral/Norwegian Computing Center, 1997.

[21] S. Tsutsui and D. E. Goldberg, "Search space boundary extension method in real-coded genetic algorithms," *Information Sciences*, vol. 133, no. 3, pp. 229–247, 2001.

[22] O. Golubitsky, V. Mazalov, and S. M. Watt, "An algorithm to compute the distance from a point to a simplex," *ACM Commun. Comput. Algebra*, vol. 46, pp. 57–57, 2012.

[23] E. Boros, P. L. Hammer, and G. Tavares, "Local search heuristics for quadratic unconstrained binary optimization (qubo)," *Journal of Heuristics*, vol. 13, no. 2, pp. 99–132, 2007.

[24] F. Glover, J.-K. Hao, and G. Kochenberger, "Polynomial unconstrained binary optimisation–part 2," *International Journal of Metaheuristics*, vol. 1, no. 4, pp. 317–354, 2011.

[25] L. Altenberg, "Nk fitness landscapes," *Handbook of evolutionary computation*, vol. 7, pp. 5–B2, 1997.

[26] H. H. Hoos and T. Stützle, *Stochastic local search: Foundations & applications*. Elsevier, 2004.

[27] S. Baluja, "Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning," tech. rep., DTIC Document, 1994.

[28] M. Pelikan, *Hierarchical Bayesian optimization algorithm*. Springer, 2005.