Linear Equality Constraints and Homomorphous Mappings in PSO

Christopher K. Monson and **Kevin D. Seppi**
Computer Science Department
Brigham Young University
{c,kseppi}@cs.byu.edu

Abstract- We present a homomorphous mapping that converts problems with linear equality constraints into fully unconstrained and lower-dimensional problems for optimization with PSO. This approach, in contrast with feasibility preservation methods, allows any unconstrained optimization algorithm to be applied to a problem with linear equality constraints, making available tools that are known to be effective and simplifying the process of choosing an optimizer for these kinds of constrained problems. The application of some PSO algorithms to a problem that has undergone the mapping presented here is shown to be more effective and more consistent than other approaches to handling linear equality constraints in PSO.

1 Introduction

Particle Swarm Optimization (PSO) [6] is a social algorithm that is most naturally applied to unconstrained optimization problems. Potential solutions called 'particles' are initialized within and 'flown' through the target function's domain, searching for the global minimum or maximum. The standard formulas for particle motion are given as follows:

$$\mathbf{v}_{t+1} = \omega\mathbf{v}_t + \phi_1\mathrm{U}_{\mathbf{1}t}\otimes(\mathbf{p} - \mathbf{x}_t) + \phi_2\mathrm{U}_{\mathbf{2}t}\otimes(\mathbf{g} - \mathbf{x}_t) \quad (1)$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{v}_{t+1} \quad (2)$$

where ω is the *inertia weight*, each $\phi_i \approx 2$, each $\mathrm{U}_\mathbf{i}$ is a vector of numbers drawn from a standard uniform distribution, and $\otimes$ performs point-wise vector multiplication [9]. The variables $\mathbf{p}$ and $\mathbf{g}$ are different for each particle and represent the best known position in the particle's own past and the best known position among particles in its neighborhood, respectively.

While unconstrained optimization is the process of finding a vector $\mathbf{g}^\star \in \mathbb{R}^D$ such that $f(\mathbf{g}^\star)$ is the global optimum, constrained optimization involves finding an optimal $\mathbf{g}^\star \in \mathcal{F}$ where $\mathcal{F} \subset \mathbb{R}^D$ is a feasible subspace of the original domain. In other words, the addition of constraints always restricts the space from which an optimal vector may be taken. A number of approaches have been used to add constraint-handling capabilities to PSO, each different depending on the nature of the constraints [3, 4, 12, 14, 15].

Though diverse in detail, the various methods of handling constraints in evolutionary optimization algorithms can be categorized as one or more of the following [7, 12]:

Preserve: All potential solutions are initialized within $\mathcal{F}$ and special operators are applied to search for new solutions without violating the constraints.

Penalize: The fitness of solutions not in $\mathcal{F}$ is artificially reduced in some way to make those solutions less desirable.

Partition: Solutions are partitioned into feasible and infeasible sets and each set is treated differently. This includes techniques such as repair of infeasible solutions and prioritizing solutions based on feasibility.

Preprocess: The problem itself is transformed so that the constraints are either easier to handle or eliminated. The Homomorphous Mapping introduced by Koziel and Michalewicz [7] is in this category.

This paper is concerned with improving the performance of PSO when applied to problems with linear equality constraints. These constraints are generally given in the form $\mathbf{Ax} = \mathbf{b}$. Admittedly, linear equality constraints form a very small subset of possible constraints, but they appear in useful real world problems such as the training of support vector machines [13]. Some interesting work specific to handling linear equality constraints in PSO is found in the Linear PSO (LPSO) and Converging Linear PSO (CLPSO) algorithms introduced by Paquet and Engelbrecht [12].

These algorithms, while simple to implement and empirically effective, have two basic limitations. First, they rely on feasibility preservation, which inherently restricts algorithm design because the constraints must define the set of possible motion operators. Second, the linear restriction placed on the motion equations is known to reduce the effectiveness of PSO in unconstrained problems [11], leading one to ask whether another style of PSO motion may be more effective in linearly constrained problems.

We propose a homomorphous mapping that transforms a space constrained by $\mathbf{Ax} = \mathbf{b}$ into a space that is not only fully unconstrained, but also of lower dimensionality, allowing any unconstrained optimization algorithm to be directly applied to a much easier problem. We begin by discussing LPSO and CLPSO, followed by the introduction of the homomorphous mapping suitable for handling linear equality constraints and some discussion about the motivation for the algorithm. Results comparing existing constrained optimization techniques are then given.

2 LPSO and CLPSO

LPSO (Linear PSO) is much like classical PSO, except that rather than use a different random number for each element of the velocity and position vectors, a single scalar is multiplied by each vector, thus:

$$\mathbf{v}_{t+1} = \omega\mathbf{v}_t + \phi_1\mathrm{U}_{1t}(\mathbf{p}_i - \mathbf{x}_t) + \phi_2\mathrm{U}_{2t}(\mathbf{g}_i - \mathbf{x}_t) \ . \quad (3)$$

This means that the resultant velocity (and therefore position) is a strictly linear combination of other particle po-

sitions. If the particles are all initialized within $\mathcal{F} = \{\mathbf{x}|\mathbf{A}\mathbf{x} = \mathbf{b}\}$, then they will always be within $\mathcal{F}$. CLPSO is similar to LPSO except that its globally best particle has its own motion equation: its next position is calculated as the sum of its personal best and a small random velocity within the null space of the equality constraints. This allows the swarm to do more local exploration and guarantees that at least a local minimum will be found.

These algorithms fill an interesting gap in the constrained PSO literature because they focus solely on linear equality constraints. They also have the advantage of relative implementation simplicity. While CLPSO appears to have much better exploration capabilities than LPSO, however, both are based upon a version of PSO that has observably poor exploration characteristics as the particles near convergence.

Consider for a moment the problem of unconstrained optimization using LPSO. If we think of the positions of particles as vectors, some or all of which participate in a basis set, then linear combinations of these vectors will span a space. Motion that is a result of strictly linear operations of these positions will force particles to always be within that span; this fact is what makes LPSO a *feasibility preserving* method.

This same feature, however, cripples it in terms of exploration capability. If there are fewer particles than effective constrained dimensions, then the algorithm is doomed from the start to explore a space with lower effective dimensionality than the target domain. If there are more particles than effective dimensions, they must be initialized in such a way as to span the entire target domain, something that is fairly likely when using random initialization. Even when this is the case, however, as some particles approach convergence and diversity decreases in the swarm, fewer of the positions will be sufficiently unique to contribute to a full span, and the dimensionality of the searchable space decreases quickly.

In either case, the search space is eventually overconstrained. This behavior of reduced search dimensionality can be observed when watching LPSO near convergence. As some particles become still, the rest will increasingly explore along a periodic straight line trajectory through $\mathbf{g}$. This exploration strategy can work well on some functions like Rastrigin, where the local minima are spread out on a regular grid, but in general it is not effective.

Even the use of some diversity increasing approaches like ARPSO [16] or Spatial Extension PSO [8] does little to solve the problem, as these are commonly implemented to perform a linear change to the particle's motion. As long as the underlying motion overconstrains the search space, these diversity increasing methods are of little help.

The overconstraining of the problem over time results in premature convergence to locations of the target domain that are not even local minima, an issue that motivated the development of CLPSO (Converging Linear PSO), which changes the motion equation for the best particle in the swarm so that it explores in a complete span of the feasible domain using a random velocity component in the null space of $\mathbf{A}$. This idea is mathematically sound and empirically effective, but it is possible that fundamentally changing the underlying motion will produce better results.

3 Homomorphous Mappings in PSO

The homomorphous mapping approach proposed by Koziel and Michalewicz [7] has many advantages over preservation methods like LPSO and CLPSO, not least of which are the ability to use an unmodified unconstrained optimization algorithm and a sometimes significant reduction of the dimensionality of the problem. This idea is especially interesting when using PSO, since it is simple to implement, effective at optimization, and most naturally applied to unconstrained problems.

In general terms, the goal of a homomorphous mapping is to convert a difficult constrained problem into a simpler constrained or unconstrained problem. The burden of constraint handling is thus shifted from the optimization algorithm to an algorithm that creates a *transform* or *decoder* $\mathcal{H} : \mathcal{S} \mapsto \mathcal{F}$ such that $\mathcal{S}$ is a space that is easier to work with than $\mathcal{F}$. The use of the decoder allows an optimization algorithm to work with points $\mathbf{x} \in \mathcal{S}$ while evaluating the target function in its original space: $f(\mathcal{H}(\mathbf{x}))$. For more information on this interesting idea, see Koziel and Michalewicz [7].

3.1 Linear Equality Constraints

Linear equality constraints of the form $\mathbf{A}\mathbf{x} = \mathbf{b}$ always define a hyperplane, assuming that the rows of $\mathbf{A}$ are linearly independent [12]. Since a hyperplane has lower effective dimensionality (D^-) than the space in which it exists (D), it is always possible to reorient the plane such that it is completely contained within $\mathbb{R}^{D^-}$, a space that is spanned by a subset of the axes in $\mathbb{R}^D$. For example, a *plane* in $\mathbb{R}^3$ can always be oriented to lie in the x–y plane, and a *line* in $\mathbb{R}^3$ can be oriented to lie along the x axis.

The size of $\mathbb{R}^{D^-}$ may be easily determined from the linear equality constraints themselves. Each row of $\mathbf{A}$ represents a vector that is normal to a hyperplane in $\mathbb{R}^D$, and the effective dimensionality of this hyperplane is always $D - 1$. To illustrate this idea, it is useful to think of adding constraint hyperplanes into a space one at a time; the first hyperplane reduces the effective dimensionality by 1, the second forms an intersection with the first which drops another effective dimension, and so on. The goal of the homomorphous mapping is to reorient the resulting lower-dimensional hyperplane such that it is contained entirely within $\mathbb{R}^{D^-}$, allowing search to be restricted to that smaller unconstrained space during optimization.

The most obvious such mapping is a *projection* from the larger space into the lower dimensional space, but this has some disadvantages, such as the necessity of selecting out the appropriate dimensions in order to perform a useful (non-degenerate) projection. The mapping on which we will focus our attention in this paper is composed of rotations and translations which are represented in a single homogeneous matrix $\mathbf{H} = \mathbf{T}^{-1}$. The complete method for

Algorithm 1 HHM($pairs$)

1: $\mathbf{T} = \mathbf{I}$
   /* *Rotate Space* */
2: **for** $i = 1$ to len($pairs$) **do**
3: $\quad a = D - (i - 1)$
4: $\quad \mathbf{p}_1, \mathbf{p}_2 = pairs[i]$
5: $\quad$ **for** $j = 1$ to $a - 1$ **do**
6: $\quad\quad \mathbf{n}^+ = \mathbf{T}(\mathbf{p}_2 - \mathbf{p}_1)^+$
7: $\quad\quad \theta = \text{atan2}(n_j, n_a)$
8: $\quad\quad \mathbf{T} = \mathbf{R}_{\theta,j,a}\mathbf{T}$
   /* *Translate Space* */
9: **for** $i = 1$ to len($pairs$) **do**
10: $\quad a = D - (i - 1)$
11: $\quad \mathbf{p}_1, \mathbf{p}_2 = pairs[i]$
12: $\quad \tilde{\mathbf{p}}_1^+, \tilde{\mathbf{p}}_2^+ = \mathbf{T}\mathbf{p}_1^+, \mathbf{T}\mathbf{p}_2^+$
13: $\quad \mathbf{n} = \tilde{\mathbf{p}}_2 - \tilde{\mathbf{p}}_1$
14: $\quad \mathbf{q} = (\tilde{\mathbf{p}}_1 \cdot \mathbf{n})\mathbf{n}$
15: $\quad$ **if** $q_a \neq 0$ **then**
16: $\quad\quad T_{a,D} = -\left\|\mathbf{q}\right\|^2 / q_a$
17: return $\mathbf{T}$

calculating $\mathbf{T}$ is given in Algorithm 1 and a more detailed explanation follows.

3.2 Homogeneous Homomorphous Mapping (HHM)

Because each row of $\mathbf{A}$ and each corresponding element of $\mathbf{b}$ together form the equation of a hyperplane, the constraint system $\mathbf{A}\mathbf{x} = \mathbf{b}$ may be rewritten as a set of equations of the form $\mathbf{n}_i \cdot \mathbf{x} = b_i$, where $\mathbf{n}_i$ is normal to plane i and b_i is a distance parameter. If $\mathbf{n}_i$ is of unit length, then b_i has a convenient geometric interpretation: it is the distance from the origin to the plane in the direction of $\mathbf{n}_i$ as illustrated in Figure 1. The figure also shows a useful alternative definition of a plane using two points:

$$\mathbf{p}_1 = b\mathbf{n} \tag{4}$$

$$\mathbf{p}_2 = \mathbf{p}_1 + \mathbf{n} \ . \tag{5}$$

This two-point definition of a hyperplane is used in Algorithm 1, which requires that each $\mathbf{n}$ has unit length. Since any constraint system $\mathbf{A}\mathbf{x} = \mathbf{b}$ may be trivially rewritten to satisfy this requirement, it will be assumed throughout the rest of this paper that constraints are normalized in this way.

The HHM algorithm is composed of two high-level steps. Starting at line 2 it calculates all of the necessary rotations that will orient the constraint hyperplane so that it is *parallel* to $\mathbb{R}^{D^-}$, but not necessarily contained within it. On line 9 it begins the process of finding the translation that will move the hyperplane so that it has no support outside of $\mathbb{R}^{D^-}$.

Each of these steps will be given special consideration below. The result of the algorithm is a homogeneous matrix of the form

$$\mathbf{T} = \begin{pmatrix} r_{1,1} & \cdots & r_{1,D} & t_1 \\ \vdots & \ddots & \vdots & \vdots \\ r_{D,1} & \cdots & r_{D,D} & t_D \\ 0 & \cdots & 0 & 1 \end{pmatrix} \tag{6}$$

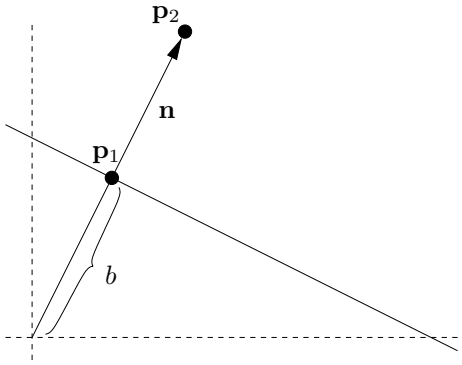


Figure 1: A plane defined by unit normal $\mathbf{n}$ and distance b, or by the points $\mathbf{p}_1$ and $\mathbf{p}_2$

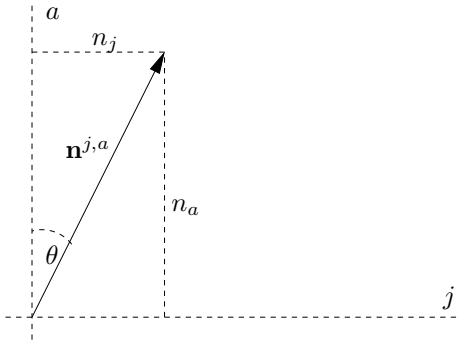


Figure 2: Calculating the rotation for a normal projection

where $r_{i,j}$ participates in rotation and t_i participates in translation. A vector multiplied through this matrix must first be augmented with a terminal 1:

$$\mathbf{p}_1^+ = \begin{pmatrix} p_{1,0} & \cdots & p_{1,D} & 1 \end{pmatrix}^\top \ . \tag{7}$$

The value $pairs$ required by the HHM algorithm is a list of point pairs representing the constraint planes as defined in (4) and (5).

To better describe the HHM algorithm, which applies to arbitrary linear equality constraints in any number of dimensions, it is useful to work through a concrete example where the number of constraints and the dimensionality are fixed. The discussion that follows will assume that $\mathbf{A}$ has two rows and that $D = 3$. Each constraint represents a plane in $\mathbb{R}^3$, and the two constraints together form a line at their intersection. The HHM algorithm will be applied to create a transform $\mathbf{T}$ that orients the entire space so that this line lies along the x-axis.

3.2.1 Rotation

The first step is to rotate each plane so that the intersection is parallel to the x axis. We begin with plane 1, which is defined by two points $\mathbf{p}_1, \mathbf{p}_2 = pairs[1]$. This plane will be rotated so that its normal is parallel to the z axis, effectively eliminating the need to consider that axis during subsequent rotations.

The first plane is realigned by rotating a projection of the normal in two planes, starting with the x–z plane; the

normal is transformed so that its projection in the x–z plane (denoted $\mathbf{n}_1^{x,z}$) lies along the z axis. Once this is done, the projection will be oriented correctly, but the actual normal vector may still have some support along the y axis.

Figure 2 illustrates what the algorithm is doing on lines 6–7: it first gets the normal into the current space and then calculates the rotation angle θ that will cause the projection of the normal into the j–a plane to lie along the a axis. In this example, j is the index of the x component in $\mathbf{n}$ and a is the index of the z component in $\mathbf{n}$.

The angle θ is computed by

$$\theta = \arctan \frac{n_j}{n_a} \tag{8}$$

and is used to construct a rotation matrix $\mathbf{R}_{\theta,j,a}$ that will orient $\mathbf{n}_1^{x,z}$ along the z axis. The rotation matrix is the identity matrix of size $D+1$ with the exception of the following elements:

$$R_{j,j} = \cos\theta \qquad R_{j,a} = -\sin\theta$$
$$R_{a,j} = \sin\theta \qquad R_{a,a} = \cos\theta \ .$$

Again referring to the example, the process is repeated in the y–z plane so that $\mathbf{n}_1^{y,z}$ lies along the z axis. When finished, the unprojected $\mathbf{n}_1 = (0\ 0\ 1)^\top$ and is therefore lined up along the z axis.

Because the first plane now has constant support along the z axis, the intersection of the planes does as well. Therefore, when applying this process to subsequent planes, that axis need not be considered again. When the next plane is considered, a rotation is performed in the x–y plane so that the projection of the second normal into that plane ($\mathbf{n}_2^{x,y}$) lines up with the y axis. When that is done, the intersection of the two planes will be parallel with the x axis, and the rotation step is complete.

3.2.2 Translation

Because the planes may not have crossed through the origin, the rotation step does not limit the constraint hyperplane to $\mathbb{R}^{D^-}$. It does, however, have *constant* support in all but the first D^- dimensions. The last step performed by the HHM algorithm performs a translation so that this constant support is removed, e.g. the intersection of the two planes in $\mathbb{R}^3$ is not merely parallel to the x-axis, but superimposed over it.

This translation step is made more convenient by the two-point definition of a plane shown in Figure 1. Translating so that the first plane contains the origin is very simple: its normal points along the z-axis and therefore it needs to be translated by its (easily calculated) distance from the origin. Once this step is complete, however, the second plane may look something like that shown in Figure 3. Note that $\tilde{\mathbf{p}}_1$ is no longer the point in the plane closest to the origin, and therefore $\|\tilde{\mathbf{p}}_1\| \neq b$.

Fortunately, it is easy to calculate the point in the plane closest to the origin using a dot product: $\tilde{\mathbf{p}}_1 \cdot (\tilde{\mathbf{p}}_2 - \tilde{\mathbf{p}}_1) = b$. The point $\mathbf{q}$ closest to the origin is simply $b(\tilde{\mathbf{p}}_2 - \tilde{\mathbf{p}}_1)$ as calculated in lines 13–14 of Algorithm 1. Given $\mathbf{q}$ it is possible to calculate the amount of y axis translation necessary

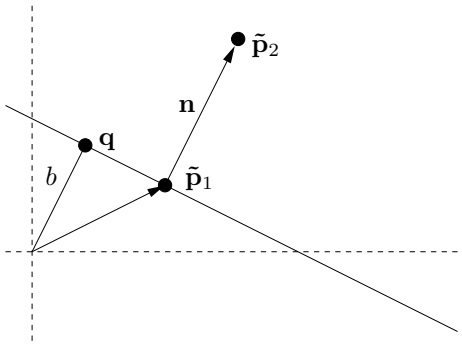


Figure 3: After translation, $\mathbf{q}$ is calculated

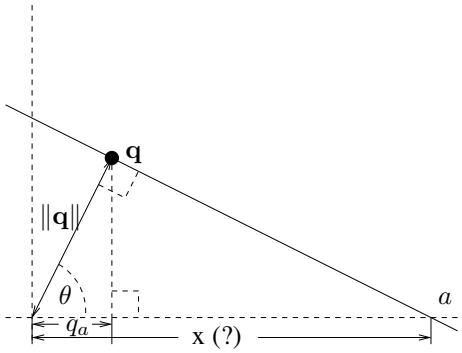


Figure 4: Calculating the final translation for a hyperplane

to ensure that plane 2 contains the origin. As long as no translation occurs in the z axis, the first plane will still contain the origin as well.

Figure 4 illustrates the way in which the translations are calculated. The angle θ is part of two triangles, and can therefore be used in two formulas to find the unknown distance x:

$$\cos\theta = q_a / \|\mathbf{q}\| \tag{9}$$
$$\cos\theta = \|\mathbf{q}\| / x \tag{10}$$

producing

$$x = \|\mathbf{q}\|^2 / q_a \tag{11}$$

which is how the translation is calculated in lines 15–16. This calculation works for every plane to which it is applied, including the first.

3.3 Comments on HHM

The result of applying HHM is a homogeneous matrix $\mathbf{T}$ that transforms *every point* in $\mathbb{R}^D$ to *another point* in $\mathbb{R}^D$. Importantly, points in the feasible region $\mathcal{F}$ are all transformed by this process to be contained within $\mathbb{R}^{D^-}$, e.g. no vector in $\mathcal{F}$ will have a nonzero value for y or z after applying $\mathbf{T}$, effectively reducing the dimensionality of the target function.

To obtain the desired matrix $\mathcal{H} : \mathbb{R}^{D^-} \mapsto \mathbb{R}^D$, one need merely invert $\mathbf{T}$ and appropriately pad vectors in $\mathbb{R}^{D^-}$ with zeros and a trailing 1 before multiplying. Performing a general inverse operation, however, is unnecessary because of

the nature of rotation matrices; it is straightforward to obtain the inverse by splitting out the rotation and translation components of $\mathbf{T}$. The effects of applying $\mathbf{H}$ can then be obtained by first applying the negative of the translation vector followed by the transposed rotation matrix. Other optimizations are possible, but are beyond the scope of this paper.

It is natural to ask why something simple like Gaussian Elimination was not used instead of this rotation/translation mapping. The advantages of the HHM presented here are that it preserves Euclidean distance and it produces an easily-reversed mapping, fulfilling two of the desiderata for homomorphous mappings [7]. Gaussian Elimination, on the other hand, performs a *projection* and is difficult to implement in a numerically stable way in all cases; in order to apply Gaussian Elimination in a way that is guaranteed to be stable, one must choose the appropriate subset of axes on which to do the projection (equivalent to determining the way in which columns of $\mathbf{A}$ are reordered), hopefully in such a way that distances in the projection correspond to similar distances in the original space [7]. The HHM does this automatically by preserving Euclidean distance, and its potential numerical problems inherent in repeated matrix multiplication are easily addressed by infrequent reorthonormalization.

4 Experiments

Given the above algorithm for calculating a mapping, handling linear constraints is as simple a task as finding $\mathbf{H}$ and searching using particles $\mathbf{x} \in \mathbb{R}^{D^-}$ while evaluating $f(\mathbf{Hx})$ in the original space. The approach outlined here is actually more general than its application to PSO, since any unconstrained optimization procedure may be applied after $\mathcal{H}$ has been calculated.

4.1 Experimental Setup

Several benchmark functions were applied with the introduction of LPSO and CLPSO, comparing them against Genocop II, an evolutionary optimization package [10]. These benchmarks are also commonly used to test unconstrained optimization algorithms:

$$\text{Sphere}(\mathbf{x}) = \sum_{i=1}^{D} x_i^2$$

$$\text{Quadratic}(\mathbf{x}) = \sum_{i=1}^{D}\sum_{j=1}^{D} e^{-(x_i-x_j)^2} x_i x_j + \sum_{i=1}^{D} x_i$$

$$\text{Rastrigin}(\mathbf{x}) = \sum_{i=1}^{D} x_i^2 + 10 - 10\cos(2\pi x_i)$$

$$\text{Rosenbrock}(\mathbf{x}) = \sum_{i=1}^{D-1} 100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2$$

$$\text{Griewank}(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \ .$$

Here they are subject to the following linear equality constraints [12]:

$$\mathbf{A} = \begin{pmatrix} 0 & -3 & -1 & 0 & 0 & 2 & -6 & 0 & -4 & -2 \\ -1 & -3 & -1 & 0 & 0 & 0 & -5 & -1 & -7 & -2 \\ 0 & 0 & 1 & 0 & 0 & 1 & 3 & 0 & -2 & 2 \\ 2 & 6 & 2 & 2 & 0 & 0 & 4 & 6 & 16 & 4 \\ -1 & -6 & -1 & -2 & -2 & 3 & -6 & -5 & -13 & -4 \end{pmatrix} \quad (12)$$

$$\mathbf{b} = \begin{pmatrix} 3 & 0 & 9 & -16 & 30 \end{pmatrix}^\top \ . \quad (13)$$

Using the mapping produced by HHM, results were obtained by applying the following *unconstrained* implementations of PSO to the resulting lower-dimensional problems:

Constricted:

$$\mathbf{v}_{t+1} = \chi\left(\mathbf{v}_t + \phi_1 \mathrm{U}_{\mathbf{1}t} \otimes (\mathbf{p} - \mathbf{x}_t) + \phi_2 \mathrm{U}_{\mathbf{2}t} \otimes (\mathbf{g} - \mathbf{x}_t)\right) \quad (14)$$

BareBones:

$$\mathbf{x}_{t+1} = \mathrm{G}\left(\frac{1}{2}(\mathbf{p} + \mathbf{g}), \mathbf{I}\,\|\mathbf{p} - \mathbf{g}\|_2^2\right) \quad (15)$$

PSOGauss:

$$\mathbf{v}_{t+1} = \chi\left(\mathbf{v}_t + \mathrm{G}\left(\mathbf{p} - \mathbf{x}_t, \frac{1}{4}\mathbf{I}\,\|\mathbf{p} - \mathbf{x}_t\|_2^2\right) + \mathrm{G}\left(\mathbf{g} - \mathbf{x}_t, \frac{1}{4}\mathbf{I}\,\|\mathbf{g} - \mathbf{x}_t\|_2^2\right)\right) \quad (16)$$

Constricted PSO [2] used $\phi_1 = \phi_2 = 2.05$ with $\phi = \phi_1 + \phi_2$ and $\chi = 2/|2 - \phi - \sqrt{\phi^2 - 4\phi}|$. BareBones [5] is a simple parameter-free algorithm proposed by Kennedy, and PSOGauss is a version of Constricted PSO with Gaussian noise as proposed by Clerc in his TRIBES paper [1]. In the definitions of both BareBones and PSOGauss, $\mathrm{G}(\cdot,\cdot)$ produces a draw from a multivariate Gaussian distribution with the supplied mean and covariance. In all unconstrained algorithms, a star sociometry is used.

4.2 Results

Tables 1–5 duplicate Paquet and Engelbrecht's results using Genocop II, LPSO, and CLPSO [12]. The tables also provide the results of applying HHM to the three unconstrained algorithms above. Except on the Rastrigin function, the use of the HHM allows *all* of the unconstrained algorithms to outperform not only LPSO and CLPSO, but Genocop II as well. Genocop II has better worst-case performance on Rastrigin but only has better average performance when employing 20 particles.

On every benchmark, including Rastrigin, the unconstrained algorithms find minima that are at least as good as those found by the constrained algorithms. Notably, every unconstrained algorithm has better best and worst-case behavior than the constrained algorithms on Griewank.

In addition to these results, Figure 5 shows the average fitness obtained by the swarm over time. The average is

Table 1: Sphere performance after 250 generations

	10 Particles				20 Particles			
	μ	σ	Min	Max	μ	σ	Min	Max
GC II	304.884	387.746	37.612	1680	54.846	16.939	32.544	107.584
LPSO	445.316	803.006	**32.137**	4505	**32.137**	7×10^{-12}	**32.137**	**32.137**
CLPSO	32.139	0.007	**32.137**	32.183	**32.137**	3×10^{-6}	**32.137**	**32.137**
Constricted	**32.137**	2×10^{-10}	**32.137**	**32.137**	**32.137**	$\mathbf{1 \times 10^{-14}}$	**32.137**	**32.137**
BareBones	**32.137**	$\mathbf{1 \times 10^{-14}}$	**32.137**	**32.137**	**32.137**	$\mathbf{1 \times 10^{-14}}$	**32.137**	**32.137**
PSOGauss	**32.137**	$\mathbf{1 \times 10^{-14}}$	**32.137**	**32.137**	**32.137**	$\mathbf{1 \times 10^{-14}}$	**32.137**	**32.137**

Table 2: Quadratic performance after 1000 generations

	10 Particles				20 Particles			
	μ	σ	Min	Max	μ	σ	Min	Max
GC II	49.945	10.996	35.393	82.221	39.5	9.785	35.41	56.613
LPSO	758.525	1496	35.4	11230	59.762	39.831	**35.377**	246.905
CLPSO	68.57	53.865	**35.377**	196.067	39.832	10.887	**35.377**	71.38
Constricted	**36.165**	**3.117**	**35.377**	**55.538**	35.783	2.394	**35.377**	55.538
BareBones	40.019	9.609	**35.377**	75.147	37.079	5.332	**35.377**	55.538
PSOGauss	38.998	8.59	**35.377**	72.482	**35.589**	**0.528**	**35.377**	**36.892**

Table 3: Rastrigin performance after 1000 generations

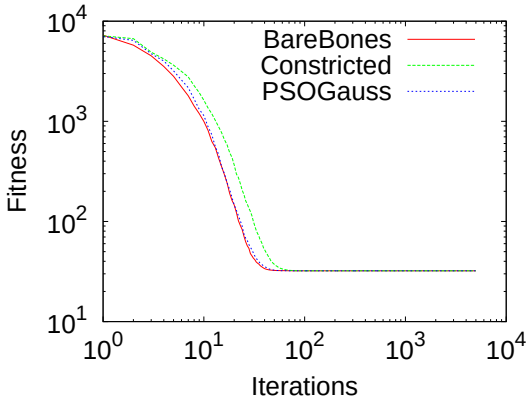
	10 Particles				20 Particles			
	μ	σ	Min	Max	μ	σ	Min	Max
GC II	52.379	**7.498**	37.116	**67.564**	**43.059**	**6.142**	37.011	**59.959**
LPSO	76.487	30.699	**36.975**	232.979	75.011	27.719	38.965	184.226
CLPSO	69.039	21.591	**36.975**	154.379	76.896	27.304	**36.975**	151.394
Constricted	**50.431**	12.314	**36.975**	85.728	46.199	7.477	**36.975**	76.736
BareBones	55.921	16.06	**36.975**	119.556	49.238	10.191	**36.975**	76.774
PSOGauss	55.622	14.826	**36.975**	119.094	47.11	8.136	**36.975**	68.802

Table 4: Rosenbrock performance after 2000 generations

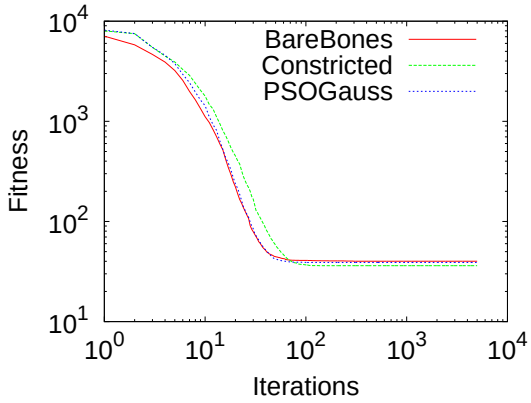
	10 Particles				20 Particles			
	μ	σ	Min	Max	μ	σ	Min	Max
GC II	21630	154.443	21490.8	22031	21485.7	0.4	21485.4	21486.6
LPSO	4×10^6	2×10^7	21554.2	2×10^8	126000	1×10^6	21485.9	1×10^7
CLPSO	744600	7×10^6	**21485.3**	7×10^7	**21485.3**	9×10^{-8}	**21485.3**	**21485.3**
Constricted	**21485.3**	$\mathbf{6 \times 10^{-11}}$	**21485.3**	**21485.3**	**21485.3**	$\mathbf{6 \times 10^{-11}}$	**21485.3**	**21485.3**
BareBones	**21485.3**	$\mathbf{6 \times 10^{-11}}$	**21485.3**	**21485.3**	**21485.3**	$\mathbf{6 \times 10^{-11}}$	**21485.3**	**21485.3**
PSOGauss	**21485.3**	$\mathbf{6 \times 10^{-11}}$	**21485.3**	**21485.3**	**21485.3**	$\mathbf{6 \times 10^{-11}}$	**21485.3**	**21485.3**

Table 5: Griewank performance after 1000 generations

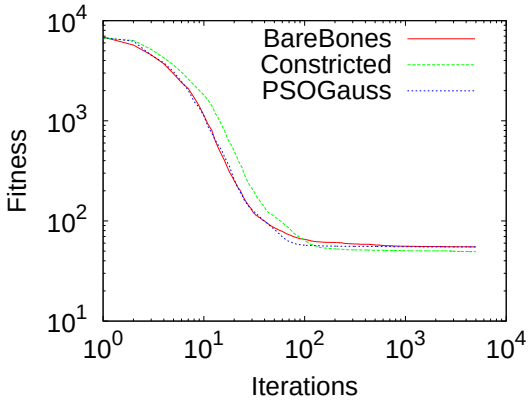
	10 Particles				20 Particles			
	μ	σ	Min	Max	μ	σ	Min	Max
GC II	0.702	0.187	0.417	0.971	0.584	**0.131**	0.201	0.843
LPSO	2.997	2.945	0.387	15.805	1.695	1.921	0.338	14.401
CLPSO	3.049	2.101	0.236	16.427	1.9	2.379	0.236	17.259
Constricted	**0.488**	**0.168**	**0.151**	**0.83**	**0.413**	0.145	**0.151**	**0.792**
BareBones	0.523	0.181	0.203	0.912	0.444	0.158	**0.151**	0.83
PSOGauss	0.53	**0.168**	**0.151**	0.958	0.454	0.174	**0.151**	0.83



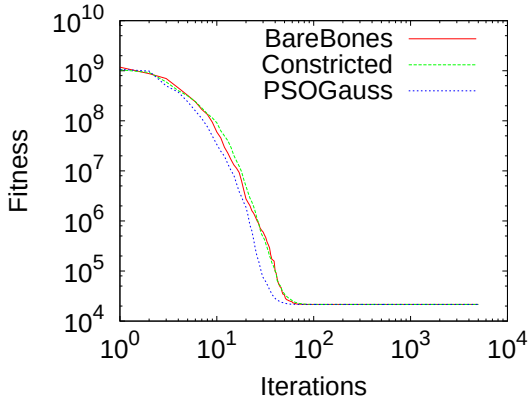
(a) Sphere



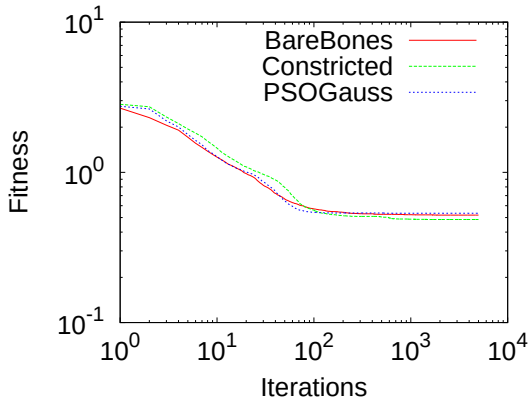
(b) Quadratic



(c) Rastrigin



(d) Rosenbrock



(e) Griewank

Figure 5: Average fitness over time for unconstrained optimizers with HHM and 10 particles

computed over 100 runs using 10 particles. These graphs show that *every* unconstrained algorithm (using HHM) on every benchmark has converged to good values by the time 100 generations have completed. Time did not allow for the creation of similar experiments with LPSO, CLPSO, and Genocop II (this should be done in the future), but it is useful to know that good values may be obtained earlier from the HHM method than the tabulated data suggest.

5 Conclusions

The homomorphous mapping is a useful and effective alternative to feasibility preservation when dealing with linear equality constraints in PSO. The particular mapping developed here, the HHM, is simple to implement, does not suffer from the numeric problems inherent in using Gaussian Elimination, and allows the application of any unconstrained optimization algorithm to a problem of reduced dimensionality. The performance of the unconstrained PSO algorithms chosen here is not only better than that of both LPSO and CLPSO in many instances, it also compares favorably with or outperforms Genocop II.

The ability to apply any unconstrained optimization algorithm to functions with linear equality constraints is a benefit by itself, since there are many more effective unconstrained optimization algorithms than those that handle constraints directly, many of which have been well tuned. Reducing the problem dimensionality provides further benefits that cannot be ignored.

The HHM approach described here may also be useful when working with linear *inequality* constraints; it is possible that it could form the basis for a truly general method of linear constraint handling. Work is ongoing in this area and will be addressed more completely in the future.

It remains to be seen how this approach will fare in real world applications like the training of SVMs, a potentially interesting direction for future research.

Bibliography

[1] M. Clerc. TRIBES - un exemple d'optimisation par essaim particulaire sans paramètres de contrôle. In *Optimisation par Essaim Particulaire (OEP 2003)*, Paris, France, 2003.

[2] M. Clerc and J. Kennedy. The particle swarm: Explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, February 2002.

[3] G. Coath and S. K. Halgamuge. A comparison of constraint-handling methods for the application of particle swarm optimization to constrained nonlinear optimization problems. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2003)*, pages 2419–2425, 2003.

[4] X. Hu and R. C. Eberhart. Solving constrained nonlinear optimization problems with particle swarm optimization. In *Proceedings of the Sixth World Multiconference on Systemics, Cybernetics and Informatics 2002 (SCI 2002)*, Orlando, USA.

[5] J. Kennedy. Bare bones particle swarms. In *Proceedings of the IEEE Swarm Intelligence Symposium 2003 (SIS 2003)*, pages 80–87, Indianapolis, Indiana, 2003.

[6] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *International Conference on Neural Networks IV*, pages 1942–1948, Piscataway, NJ, 1995. IEEE Service Center.

[7] S. Koziel and Z. Michalewicz. Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary Computation*, 7(1):19–44, 1999.

[8] T. Krink, J. S. Vestertroem, and J. Riget. Particle swarm optimisation with spatial particle extension. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2002)*, Honolulu, Hawaii, 2002.

[9] R. Mendes, J. Kennedy, and J. Neves. The fully informed particle swarm: Simpler, maybe better. *IEEE Transactions on Evolutionary Computation*, 8(3), June 2004.

[10] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, Berlin, 3 edition, 1996.

[11] C. K. Monson and K. D. Seppi. The Kalman swarm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 140–150, Seattle, Washington, 2004.

[12] U. Paquet and A. P. Engelbrecht. A new particle swarm optimizer for linearly constrained optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2003)*, pages 227–233, Canbella, Australia, 2003.

[13] U. Paquet and A. P. Engelbrecht. Training support vector machines with particle swarms. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2003)*, pages 1598–1603, 2003.

[14] K. E. Parsopoulos and M. N. Vrahatis. Particle swarm optimization method constrained optimization problems. In *Proceedings of the Euro-International Symposium on Computational Intelligence 2002*, 2002.

[15] G. T. Pulido and C. A. C. Coello. A constraint-handling mechanism for particle swarm optimization. In *Proceedings of the 2004 Congress on Evolutionary Computation (CEC'2004)*, volume 2, pages 1396–1403, Portland, Oregon, June 2004. IEEE.

[16] J. Riget and J. S. Vesterstrøm. A diversity-guided particle swarm optimizer — the ARPSO. Technical Report 2002-02, Department of Computer Science, University of Aarhus, 2002.