Subproblem Optimization by Gene Correlation with Singular Value Decomposition

Jacob G. Martin
University of Georgia
Computer Science
Athens, GA 30602
martin@cs.uga.edu

ABSTRACT

Several ways of using singular value decomposition (SVD), a linear algebra technique typically used for information retrieval, to decompose problems into subproblems are investigated in the genetic algorithm setting. Empirical evidence, concerning document comparison, indicates that using SVD results both in a savings in storage space and an improvement in information retrieval. Combining theoretical results and algorithms discovered by others, several problems are identified that the SVD can be used with to determine a substructure. Subproblems are discovered by projecting vectors representing the genes of highly fit individuals into a new low-dimensional space, obtained by truncating the SVD of a strategically chosen gene $\times$ individual matrix. Techniques are proposed and evaluated that use the subproblems identified by SVD to influence the evolution of the genetic algorithm. By restricting the locus of optimization to the substructure of highly fit individuals, the performance of the genetic algorithm was improved. Performance was also improved by using SVD to genetically engineer individuals out of the subproblems.

Categories and Subject Descriptors

I.5.3 [**Computing Methodologies**]: Pattern Recognition-Clustering[algorithms, similarity measures]

General Terms

Theory, Algorithms, Experimentation

Keywords

Genetic algorithm, singular value decomposition, graph bisection, graph partitioning, gene decomposition, spectral clustering, linkage learning, probabilistic model building, genetic engineering, graph clustering, reduced rank approximation

GECCO'05, June 25–29, 2005, Washington, DC, USA.

1. INTRODUCTION

The technique of singular value decomposition (SVD) has proven itself valuable in several different problem domains: data compression [13], image recognition and classification [16], chemical reaction analysis [31], document comparison [10, 6], cryptanalysis [30], and genetic algorithms [27]. Although these domains are quite different in some aspects, each can be reduced to the problem of ascertaining or ranking relevance in data. Intuitively, the concept of relevance depends critically on the nature of the problem at hand. SVD provides a method for mathematically discovering correlations within data.

The focus of this work is to investigate several possible methods of using SVD to guide the search process of a Genetic Algorithm (GA). SVD helps expose the most striking similarities between genes in the most highly fit individuals of the optimization history. The GA's optimization operators are then restricted to the locus of the genes corresponding to these striking similarities. Individuals are also *engineered* out of the discovered similarities between genes across highly fit individuals.

Patterns identified in the theoretical results are used as a basis for creating an artificial problem that serves as a benchmark for the types of problems that will benefit from this research. The genetic algorithm's subproblem determination performance on several formulations of the NP-Complete Minimum Graph Bisection problem are also presented, giving insight into the structural discovery abilities of SVD. Results from the application of this process to several problems indicate a significant improvement in the GA's performance. In addition, the subproblems were usually determined early in the optimization process. Furthermore, using the discovered subproblems to genetically *engineer* individuals yielded additional performance improvements.

Linear algebra background is provided in Section 2, followed by a description of the integration of SVD into a genetic algorithm in Section 3. Section 4 describes the results achieved by using the proposed methods in several different problem domains. Finally, Section 5 provides some promising opportunities for future research.

2. BACKGROUND

2.1 Singular Value Decomposition

2.1.1 Theorem Statement

THEOREM 1. *Let A be an $m \times n$ real matrix with rank r. Then there exists an $m \times n$ diagonal matrix*

$$\Sigma = \begin{bmatrix} D & 0 \\ 0 & 0 \end{bmatrix} \quad (1)$$

where the diagonal entries of D are the first r singular values of A, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$, and there exist an $m \times m$ orthogonal matrix U and an $n \times n$ orthogonal matrix V such that

$$A = U\Sigma V^T \quad (2)$$

The existence and theory of singular value decomposition was established by several mathematicians [35]: Beltrami [4], Jordan [24], Sylvester [36], Schmidt [34], and Weyl [38]. Horn and Johnson provide a succinct proof of its existence [23].

2.1.2 Summary

As Theorem 1 states, singular value decomposition expresses an $m \times n$ matrix A as the product of three matrices, U, Σ, and V^T. The matrix U is an m x m matrix whose first r columns, u_i $(1 \leq i \leq r)$, are the orthonormal eigenvectors that span the space corresponding to the row auto-correlation matrix AA^T. The last $m - r$ columns of U form an orthonormal basis for the left nullspace of A. Likewise, V is an n x n matrix whose first r columns, v_i $(1 \leq i \leq r)$, are the orthonormal eigenvectors that span the space corresponding to the column auto-correlation matrix A^TA. The last $n - r$ columns of V form an orthonormal basis for the nullspace of A. The middle matrix, Σ, is an m x n diagonal matrix with $\Sigma_{ij} = 0$ for $i \neq j$ and $\Sigma_{ii} = \sigma_i \geq 0$ for $\forall i$. The σ_i's are called the singular values and are arranged in descending order with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$. The singular values are defined as the square roots of the eigenvalues of AA^T and A^TA. The singular value decomposition can equivalently be expressed as a sum of rank one matrices

$$A = \sigma_1 u_1 v_1^T + \cdots + \sigma_r u_r v_r^T = \sum_{i=1}^{r=rank(A)} \sigma_i u_i v_i^T \quad (3)$$

The u_i's and v_i's are the columns of U and V respectively. Using the Golub-Reinsch algorithm [21, 18], U, Σ, and V can be calculated for an m by n matrix in time $O(m^2n + mn^2 + n^3)$.

2.1.3 Reduced Rank Approximations

The magnitudes of the singular values indicate the weight, or importance, of a dimension. To obtain an approximation of A, all but the $k < r$ largest singular values in the decomposition are set to zero. This results in the formation of a new low-dimensional matrix A_k, of rank k, corresponding to the k most influential dimensions.

$$A_k = U_k \Sigma_k V_k^T \quad (4)$$

Here, U_k and V_k are the matrices formed by keeping only the eigenvectors in U and V corresponding to the k largest singular values. Equivalently,

$$A_k = \sigma_1 u_1 v_1^T + \cdots + \sigma_k u_k v_k^T = \sum_{i=1}^{k} \sigma_i u_i v_i^T \quad (5)$$

The reduced rank matrix A_k amplifies the most important similarities and suppresses the insignificant correlations between the vectors represented in the matrix A. Exactly how much of the original space is preserved is directly related to the amount of reduction performed. A theorem by Eckart and Young states, informally, that the new low-dimensional matrix obtained is the closest matrix, among all matrices of its rank or less, to the original matrix [14, 18]. Formally, it states that among all m x n matrices C with rank at most k, A_k is the one that minimizes

$$\|A - C\|_F^2 = \sum_{i,j} (A_{ij} - C_{ij})^2 \quad (6)$$

Eckart and Young's paper was a rediscovery of this property, first proved by Schmidt [34]. Although the theorem may explain why the reduction *does not deteriorate too much* in performance over conventional vector-space methods, it fails to justify the observed improvement in precision and recall [32]. However, several papers have made positive steps towards a rigorous proof that, given an appropriate structure for the matrix A, the benefit is achieved with high probability [32, 11].

2.2 Clustering

2.2.1 Spectral Guarantees

Previous work has shown that when a block diagonal matrix with k blocks is slightly perturbed, the k largest of ***all*** of the eigenvalues of the nearly block diagonal matrix A^TA are the maximum eigenvalues of each block $B_i^TB_i$, for $i = 1, \cdots, k$, with high probability [32]. Therefore, when projecting onto the k largest eigenvectors of V, an *individual* query vector created mainly from a block $B_i \in A$ will likely be projected only in the direction of the maximum eigenvector of the i'th block of the individual-individual autocorrelation matrix $B_i^TB_i$. Likewise, a *gene* query vector from a block $B_i \in A$ will only be projected in the direction of the maximum eigenvector of the i'th block of the gene-gene autocorrelation matrix $B_iB_i^T$. These results are based on several papers that prove bounds between the conductance and the spectrum of a graph [3, 1, 2].

2.2.2 Block Diagonal Forms

Analyses in this paper are restricted to problems whose ideal solutions can be arranged to form a block diagonal or near block diagonal matrix. By the definition of block diagonal, two solution vectors with ones in different blocks will never have the same component equal to one. Thus, the dot product of any two rows of two solutions from separate blocks will be zero. Furthermore, the dot product between two columns from different blocks will be zero and the vectors will be perpendicular. If the two columns or rows are in the same block, the dot product will always be one. Due to the spectral clustering guarantees presented in the previous section, these relations will also hold true for reductions of solution matrices A down to A_k with k equal to the number of blocks.

If some sequence of row and column interchanges can make a matrix be close to block diagonal, then SVD will

find the sub-blocks of similarly used rows and columns. Sequences of row and column interchanges do not affect the bases produced when computing the SVD because a matrix's row and column space remain the same after any sequence of interchanges. Therefore, if the matrix is not in a block diagonal order but some sequence of row and column interchanges can make it block diagonal, then SVD will still recover the blocks. Notice that swapping the columns of individuals into block diagonal order does not affect the rows of genes that are used similarly across individuals.

$$\begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \xrightarrow{\text{(swap columns 2 and 3)}} \begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

This idea is important in the context of genetic algorithms because solutions are typically not ordered in a block form.

It should be noted that more general forms of matrices have been shown to benefit from clustering with reduced rank SVD [11]. Therefore, it is extremely likely that the SVD operators to be described will prove beneficial for many different problems and representations.

3. GENETIC ALGORITHM

3.1 Background and Terminology

Genetic Algorithms (GAs) are search and optimization methods that mimic natural selection and biological evolution to solve optimization and decision problems. The books by Goldberg [19] and Michalewicz [28] provide a thorough introduction to Genetic Algorithms. A *chromosome* is a sequence of gene values. In this paper, each gene has a value of zero or one. A potential solution to a problem is represented by a chromosome. For graph problems, the number of vertices is the size of the chromosome. A *schema* is a pattern of genes consisting of a subset of genes at certain gene positions. If n is the size of a chromosome, a *schema* is an n-tuple $\{s_1, s_2, \cdots, s_n\}$ where $\forall i, s_i \in \{0, 1, \star\}$. Positions in the schema that have a $\star$ symbol correspond to don't-care positions. The non-$\star$ symbols are called *specific symbols*, and represent the defining values of a schema. The number of specific symbols in a schema is called the *order*, and the length between the first and last specific symbols in a schema is called the *defining length* of the schema.

Although genetic algorithms do not specifically work with schemas themselves, schemas are a fundamental concept when analyzing the exploratory process of a genetic algorithm. According to the *building block hypothesis* [19, 22], GAs implicitly favor low-order, high-quality, schemas. Furthermore, as evolution progresses, the GA creates higher order, high-quality schemas out of low-order schemas. This is partially due to the nature of the crossover operator.

3.2 SVD Incorporation

The goal is to discover the genes that are used *similarly* across the best individuals. Determining genes that are used similarly with SVD yields accurate identification of subproblems in optimization problems whose solutions have a block representation. The SVD of a matrix containing the best few individuals in the entire optimization history was computed. Instead of aiming for the sole fittest individual, the GA used SVD to decompose the few fittest individuals and therefore directed the search towards a *combination* of the best individuals. Tests using large sets of individuals were not as beneficial. Perhaps this was because the SVD could not discover a single pattern for which to aim during operator restriction.

The computational complexity of computing the SVD may outweigh the complexity of the problem being solved. However, problems with a computationally expensive fitness function may benefit from the methods to be described. In particular, if complex problems can be decomposed into smaller and simpler subproblems, then the benefit will outweigh the cost of computing the SVD. Several time optimizations can also be made to decrease the amount of time used computing the SVD. For example, existing SVDs can be updated using special algorithms for adding or removing rows and columns [5]. Also, random projections are a fast alternative to singular value decomposition [32].

3.3 Subproblem Genetic Operators

3.3.1 Restricted Mutation and Crossover

At every other generation, the mutation operator was restricted to a specific subset of the genes. This isolated the search process to the blocks in highly fit solutions, facilitating the determination of the local optimum. Similarly, the crossover operator was restricted to a specific group of genes. After crossover was applied to the reduced gene set, the unrestricted genes were replaced in their corresponding positions in the generated children. In both techniques, the restriction only happened every other generation. This enabled the mutation and crossover operators to fully explore the entire space of possible chromosomes.

3.3.2 Genetic Engineering

A simple genetic engineering approach was tested at every generation. First, the rank 2 SVD of the top 50 best individuals was computed. Then, using a process to be described in Section 3.4, a set of subproblems was generated. Next, a random subproblem with the correct size (the parameters of the problem were known) was selected and a new individual constructed by placing ones in the corresponding positions of the genes in the subproblem, and zeros everywhere else. For example, given the subproblem $\{1, 3\}$ the individual constructed would be $[1010 \cdots 0]$. If no subproblem had the correct size, then no individual was engineered during that generation. Future research could easily develop problem dependent heuristics to engineer good individuals out of subproblems with an arbitrary size.

3.4 Subproblem Determination

After the formation of a matrix of good individuals, the following steps were taken to group genes into subproblems. For every gene, the cosines of the angles between it and every other eigenvector of AA^T were put into a matrix. In order for gene i and gene j to belong to the same subproblem, the cosines of the angle between the i^{th} and j^{th} eigenvectors of the gene-gene autocorrelation matrix had to be greater than 0.92. That is, the vectors had to be close to parallel. The cosine of an angle is an appropriate function to use because its value approaches one as two vectors become more parallel. Likewise, as two vectors become more perpendicular, the cosine of the angle between them approaches zero. The bound of 0.92 was chosen apriori by testing values be-

tween zero and one. However, strategies could be produced to vary this amount in a heuristic manner. For example, if the problem's solutions are required to have subproblems of genes with a particular size, then the parameter could be adjusted to favor retrieving subsets of genes with the correct size.

If ω_{ij} is the angle between the i^{th} and j^{th} gene vector then,

$$\cos\omega_{ij} = \frac{(e_i^T U\Sigma V^T)(V\Sigma U^T e_j)}{\| e_i^T U\Sigma V^T \|_2 \| V\Sigma U^T e_j \|_2} \quad (7)$$

For gene i and gene j to be clustered into the same subproblem, the following relation had to hold

$$\cos\omega_{ij} > 0.92 \quad (8)$$

Here, e_i denotes the i^{th} standard vector, which contains all zeroes except for a one in the i^{th} position. $\| \cdot \|_2$ denotes the Euclidean vector norm. The U, Σ, and V are the matrices found by the SVD.

In the document comparison domain, reduction of rank actually *improves* the quality of the information retrieved [5, 32]. Using reduced rank versions on problems with a block diagonal representation gives an approximation of what it means for two genes to be used similarly across a group of individuals. Therefore, various rank reductions were also tested. To calculate the cosines between genes in a reduced rank model, A_k is substituted for A in all of the above calculations.

$$\cos\omega_{ij} = \frac{(e_i^T U_k\Sigma_k V_k^T)(V_k\Sigma_k U_k^T e_j)}{\| e_i^T U_k\Sigma_k V_k^T \|_2 \| V\Sigma_k U_k^T e_j \|_2} \quad (9)$$

Berry, Drmač and Jessup provide an in depth explanation of how to efficiently compute the rank-k cosines between a query vector and the vectors contained in a reduced rank-k model [5].

3.5 Subproblem Selection Strategies

Two methods were tested for determining the subproblems the GA should work on. In the first method, *Maximum Subproblem*, the largest sized subproblem was selected. In the second method, *Subproblem Rotation*, a subproblem was chosen at random.

3.6 Low Rank Approximations

Two forms of the SVD were tested. The first was the full rank version of the SVD. The second was based on the reduced rank version, where all but the first k largest singular values are set to zero, giving A_k. As expected, the reduced rank strategies generally discovered the subproblems more efficiently than the full rank versions. This is due in part to the theoretical results mentioned in Section 2.2. The performance may also have improved because, in the application domains tested, the GA was only seeking one block in the solution space. Reduction to a lower rank correctly directs the search towards the correct block because a lower value of k in A_k increases the cosines of the angles between vectors of similar types [7]. Another reason may be that in comparison with higher rank reductions, lower rank reductions are less restrictive and will identify larger subsets of related genes as the rank is reduced. Therefore, lower rank reductions allow the restrictive mutation and crossover operators to have more freedom during exploration. However, lowering the rank too far may not always increase the performance because all genes will be seen as similar to all other genes.

4. EXPERIMENTAL RESULTS

As mentioned previously, SVD should perform well when analyzing problems that have a solution space that can be made block diagonal. The first problem tested was the Block Sum Partitioning problem. This problem was created and tested to provide a benchmark for the types of problems that will benefit from this research. The solution vectors of this problem can be arranged to form a block diagonal matrix. When two genes are used similarly across the solution individuals, they often contain the same value across their rows. When the SVD subproblem clustering process is applied to a matrix of correct solutions for this problem, the clusters returned are exactly the subproblems that define where a solution should have the value one.

A problem's individual type will be referred to as *symmetric* if whenever the vector obtained by applying the Boolean NOT to every gene in an individual represents the same solution to the problem. For example, the Minimum Graph Bisection problem's individual type is *symmetric* because $\{1, 0\}$ represents the same bipartition as $\{0, 1\}$. SVD is not confused by solutions that are symmetric. In other words, SVD is not affected by the possible namings of a partition. This is because in problems with a *symmetric* individual type, similar genes are still used similarly across individuals drawn from the same block. Furthermore, subproblems will be correctly discovered regardless of the order the rows or columns of the matrix are in. The Minimum Graph Bisection problem's solution vectors are symmetric and can be arranged to form a block diagonal matrix. When two genes are used similarly across the solution individuals in this problem, it means that the vertices that the genes represent are frequently placed in the same partition. SVD helps obtain an approximate *consensus* from the best individuals as to which vertices should be placed in the same partition.

4.1 Implementation Details

Tests were performed using a custom GA, implemented entirely in Java^{TM}. The source code and documentation for the GA may be obtained by e-mailing the author. The SVD was computed using Visual Numerics' Java Numerical Library. An approach similar to the $(\mu + \lambda)$ evolution strategy was used with populations of size 100 generating 100 candidate individuals. Reinsertion was achieved by picking the best 100 individuals out of the 200 total parents and children. The results are based on the average of the best individual at each generation, over 100 different random initial populations. Let $f(x)$ be the value of the function that is being optimized when applied to an individual x. The fitness of an individual is defined as

$$fitness(x) = \ln \frac{1}{1 + |f(x) - target|} \leq 0 \quad (10)$$

In this fitness function, the function value $f(x)$ approaches its target (for example the minimum) as the fitness function approaches zero. Individuals with higher fitness represent better solutions than those with lower fitness. An individual with a fitness equal to zero is an exact solution because only then will $f(x) = target$.

4.2 Block Sum Partitioning

4.2.1 Problem Statement

Let $(x)_{ik}$ denote the i^{th} block of size k in a binary string. Furthermore, let $(x)_{ik}^{m}$ denote the numeric value of the m^{th} position in the i^{th} block. The (n,k)-block partition problem is defined on binary strings of length n as follows:

$$\max_{i=1,2,\ldots,\frac{n}{k}} \left(\sum_{m=1,2,\ldots,k} (x)_{ik}^{m} \quad - \sum_{l\neq i, m=1,2,\ldots,k} (x)_{lk}^{m} \right) \quad (11)$$

In words, the (n,k)-block partition problem is the maximum, over all the $1, 2, \cdots, \frac{n}{k}$ blocks of k genes, of the sum of the elements in the block minus the sum of the elements not in the block. This problem will be called the Block Sum Partitioning problem (BSP). An individual is considered a solution when its function value is equal to k, the length of every block. By the problem's construction, this can only happen when an individual contains all ones in one block and all zeroes in every other block. Therefore, the set of individuals that are solutions form a block diagonal matrix. From the analysis presented in Section 2.2, the SVD should perform well on this type of problem because it will be able to accurately categorize the similar genes of highly fit individuals. The highly correlated genes of good individuals will correspond to the genes that should belong to the same block.

4.2.2 Implementation Details

Tests were performed on the binary valued version of the (100, 10)-Block Sum Partitioning problem. That is, the problem of structuring 100 genes into a form with one of the 10 blocks containing 10 ones, and the rest of the 9 blocks containing all zeroes. The genetic operators do not know the value of k. Hence, the full representation space is used and not restricted to individuals with k ones during optimization. The mutation rate was set at 12%. Restricted mutation was performed by flipping a gene in a subproblem to its opposite value of either one or zero. Restricted one point crossover was used in both of the genetic subproblem strategies. The plain GA used both one point crossover and mutation without restrictions. Genetic engineering was not tested with this problem.

4.2.3 Results

Figure 1 is a plot of the average best individual at every generation for this problem using various ranks. The maximum subproblem's performance was very similar to the corresponding variations of the subproblem rotation's performance. Both of the subproblem methods outperformed the plain GA. The subproblem rotation strategy using a rank equal to 2 performed best. Furthermore, rank reduction increased the performance of the genetic algorithm in all cases.

Call a set of genes *involved* in a solution if setting each gene in the set to one and each gene out of the set to zero yields a correct solution to the problem. The following tables compare the first generation the GA discovered a solution and the first generation that each subproblem determination method correctly identified at least $\frac{2}{3}$ of a set of genes that is *involved* in a correct solution. In addition, the subproblems were only counted as being found when their size was at least $\frac{2}{3}$ of the size of a correct subproblem. They were not counted when their size was greater than the correct subproblem's size. The results in the following tables were obtained by collecting the average over 100 runs, using full rank, restricted crossover, and restricted mutation. Notice that the subproblems were discovered much earlier than the first solution.

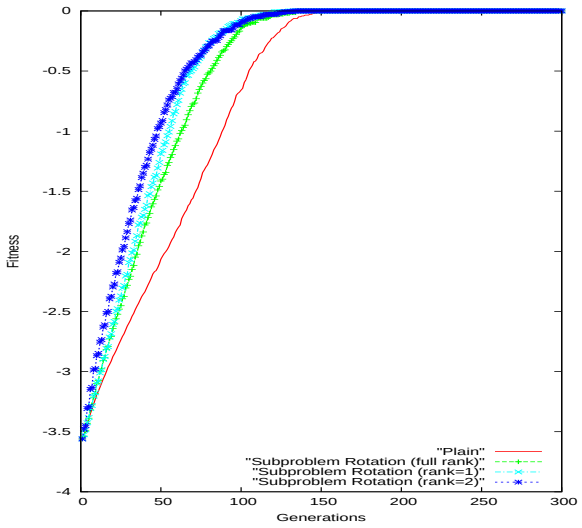

Figure 1: The average best individual per generation for the BSP problem using the Subproblem Rotation Strategy.

BSP (full rank)	Solution	Subproblem found
Maximum	103.87	**63.64**
Rotation	86.92	**75.76**

Although the overall first solution performance was better with the rank 1 reduction for this problem, the subproblems were not typically discovered until after the first solution was found. A possible explanation for this is that the size of the subproblem found when the rank was reduced was much larger than the size of the subproblem when using full rank. This is because under a reduced rank model, genes are more likely to be similar to other genes. While the correct subproblem is likely still represented in the set, the size of the set is usually much larger than the size of a correct subproblem. In these cases the subproblem was not counted as being found because it was too big.

BSP (rank 1)	Solution	Subproblem found
Maximum	87.27	**263.74**
Rotation	75.59	**231.5**

As the following table indicates, the rank 2 reductions resulted in the best overall performance.

BSP (rank 2)	Solution	Subproblem found
Maximum	77.79	**45.02**
Rotation	73.4	**78.06**

4.3 Minimum Graph Bisectioning

4.3.1 Problem Statement

A bisection of a graph $G = (V, E)$ with an even number of vertices is a pair of disjoint subsets $V_1, V_2 \subset V$ of equal size. The cost of a bisection is the number of edges $(a, b) \in E$ such that $a \in V_1$ and $b \in V_2$. The Minimum Graph Bisection problem takes as input a graph G with an even number of vertices, and returns a bisection of minimum cost. The Minimum Graph Bisection problem has been shown to be NP-Complete [17]. Many heuristics have been developed for this problem. Perhaps the best known is the Kernighan-Lin heuristic [25, 8]. Graph partitioning with genetic algorithms has been studied extensively [26, 9]. Singular value decomposition has also proved to be a useful tool when clustering graphs [12, 37]. However, this paper is the first to combine these results, providing strategies for using singular value decomposition in a genetic algorithm for the Minimum Graph Bisection problem.

4.3.2 Implementation Details

Individuals were represented in binary. If the i^{th} component of an individual was one, then the i^{th} vertex was put in the set V_1. Otherwise, if the i^{th} component of an individual was zero, then the i^{th} vertex was placed in the set V_2. Notice that individuals are symmetric in this representation. The mutation rate was set at 12%. A modified mutation method of switching two random genes was implemented to keep the number of ones and zeroes in an individual equal. In the case of subproblem evolution, a gene from the subproblem area was flipped and an opposite gene from the non-subproblem area was also flipped. In plain GAs, the mutation operator simply exchanged the values of two opposite genes. The crossover operator was adapted from an earlier paper on graph bisection with GAs [9]. It is a modified five point crossover that attempts to account for the symmetric nature of graph bisection solutions. No restriction on the locus of crossover was used in this problem. The highly correlated genes correspond to vertices that the current population believes should be clustered into the same partition.

4.3.3 Random Graphs

Figure 2 contains the average fitness of the best individual at each generation over 100 different random graphs. An edge between two vertices was created with a 5% chance. In this problem, the subproblem methods outperformed the plain GA, but only by a slight margin. Graphs with higher chances of an edge occurring between vertices produced very similar results. The decrease in performance in the engineering results after the first 100 generations indicate that the populations may have become genetically stale.

4.3.4 Highly Clustered Random Graphs

The random graphs for this problem were created by first randomly dividing all of the vertices into two disjoint and equal sized sets. Next, edges within a set were created with a 98% probability. Then, edges between vertices in different sets were created with probability equal to 5%. This problem will be called the Minimum Graph Bisection Cluster problem. Presumably, SVD will perform remarkably better in the cases where the random graph contains two main clusters. Accordingly, tests performed on random graphs

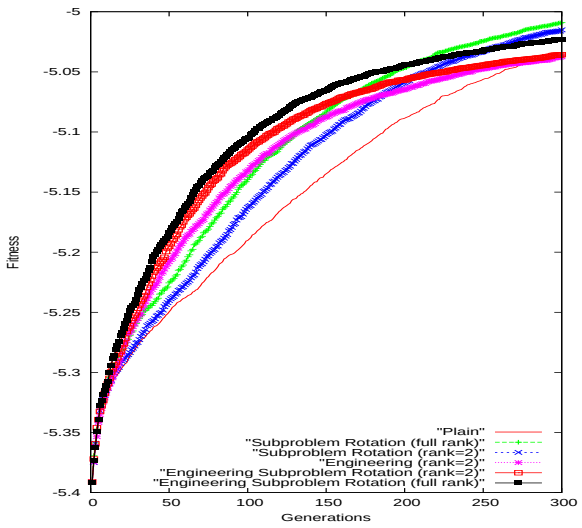

Figure 2: The average best individual per generation for the Minimum Graph Bisection problem on random graphs with 100 vertices and 5% edge probability.

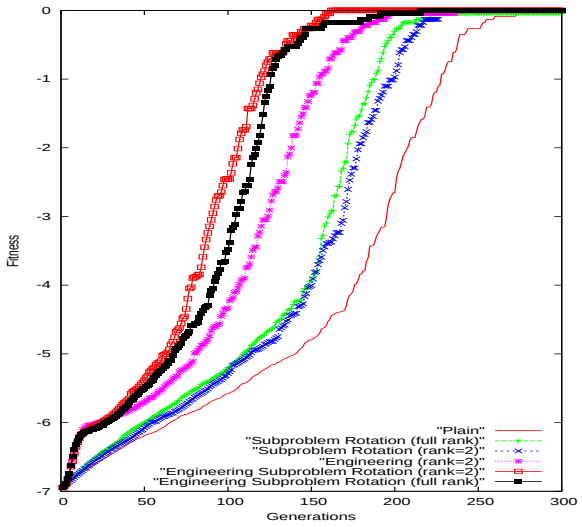

Figure 3: The average best individual per generation for the Minimum Graph Bisection Cluster problem on random graphs with 100 vertices and two main clusters.

that were explicitly constructed to contain most of their weight in two clusters, indicated an increase in the performance of the SVD subproblem and engineering methods. Figure 3 is a plot of the results from these highly clustered random graphs. Once again, all SVD methods outperformed the plain GA. Furthermore, the combination of restricted operators with genetic engineering yielded better results than using either restricted operators or genetic engineering alone. Engineering consistently gave a significant performance boost during the first fifty generations of optimization. Additionally, engineering was improved by reducing the rank to 2.

The following table lists the average first generation the GA subproblem determination methods identified at least $\frac{2}{3}$ of the genes *involved* in a solution. The results in the following table were obtained by collecting the average over 100 runs. The subproblems were discovered much earlier than the first solution was obtained.

GBC (full rank)	Solution	Subproblem found
Plain	206.58	NA
Maximum	176.22	**99.92**
Rotation	154.64	**97.38**

For this problem, rank one reduction did not improve the GA's overall solution performance for either subproblem strategy. This indicates that the reduction to rank one may have deteriorated the solution space too much. On the other hand, the subproblems were typically found earlier than the full rank version found them.

GBC (rank 1)	Solution	Subproblem found
Maximum	181.29	**74.46**
Rotation	204.45	**86.7**

5. FUTURE DIRECTIONS

5.1 Additional Problem Domains

Different graph types, such as geometric and clustered graphs, should also be investigated. In addition, both the (n,k)-graph partitioning problem and the problem of clustering vertices into differently sized parts should be studied using the engineering and restricted genetic operators described. This research will be important to help further unify and generalize the types of problems to which SVD can be successfully applied in a GA.

5.2 Schema Reordering

Due to the nature of the problems addressed, good schema are apt to be destroyed during crossover if the locations forming the schema are scattered apart on the chromosome. To combat the disruptive nature of crossover, chromosomes could be reordered to group the similar genes closer together on a chromosome. This would help to create higher-quality schemas with shorter defining lengths. SVD could be used to define the reordering during optimization. The reordering would group similar genes together, allowing the GA to benefit from the building block hypothesis. This is in contrast to a strategy that only performs an initial schema preprocessing once before the GA for the Minimum Graph Bisection problem starts [9]. As the building block hypothesis suggests, the computational power of genetic algorithms largely comes from manipulating the solutions of subproblems, i.e., building blocks. Hence, identifying subproblems has been a center of many subfields within genetic and evolutionary computation. Three examples of related fields that should be studied to better connect the use of SVD to current GA research are Linkage Learning [20], Probabilistic Model Building Genetic Algorithms [33], and Learnable Evolution Models [29].

5.3 Further Work

Future work should concentrate on several issues. First, more aggressive SVD subproblem heuristics could be performed in the graph bisection problems by using the Kernighan-Lin algorithm [25] in a manner investigated by Bui and Moon [9]. Second, there have been several papers that generalize the categorization powers of reduced rank SVD to situations that are not specifically transformable to block diagonal form [15]. Problem types with structures other than a block diagonal matrix need to be considered to determine additional representations that the SVD can be used with to benefit a genetic algorithm. Third, heuristics for rank choice should be identified to improve the overall subproblem determination performance. Finally, it would be interesting to create heuristics for choosing different subsets of individuals that determine the subproblems at each generation. For example, the worst, the best, or even the most diverse solutions in the optimization history could each be valid choices for the subsets of individuals that direct the optimization process.

6. CONCLUSION

In conclusion, singular value decomposition is useful in genetic algorithms if the solution space can be made nearly, or entirely, out of blocks. SVD was shown to discover subproblems for problems with block diagonal representations by approximating the gene $\times$ gene autocorrelation matrix of highly fit individuals. The SVD operators are useful in GAs when knowing which genes are used similarly across good individuals helps to solve the problem. The GA's performance on the NP-Complete Minimum Graph Bisection problem was improved. Also, the SVD methods were shown to perform better on random graphs with highly clustered parts than general random graphs. Therefore, SVD is beneficial in GAs. In particular, engineering and restrictive genetic operators using SVD will discover gene similarity well if the solution space is either blocked or nearly blocked.

7. ACKNOWLEDGEMENTS

I would like to thank Rod Canfield and Khaled Rasheed for many enlightening discussions and insights.

8. REFERENCES

[1] Alon. Eigenvalues and expanders. *COMBINAT: Combinatorica*, 6, 1986.

[2] N. Alon and V. D. Milman. λ_1, isoperimetric inequalities for graphs, and superconcentrators. *Journal of Combinatorial Theory. Series B*, 38:73–88, 1985.

[3] Y. Azar, A. Fiat, A. Karlin, M. McSherry, and J. Saia. Spectral analysis of data. In ACM, editor, *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing: Hersonissos, Crete, Greece, July 6–8, 2001*, pages 619–626, New York, NY, USA, 2001. ACM Press.

[4] E. Beltrami. Sulle funzioni bilineari. *Giornale di Matematiche ad Uso degli Studenti Delle Universita*, 11:98–106, 1873. An English translation by D. Boley is available in Technical Report 90–37, Department of Computer Science, University of Minnesota, Minneapolis, 1990.

[5] M. W. Berry, Z. Drmač, and E. R. Jessup. Matrices, vector spaces, and information retrieval. *SIAM Rev.*, 41(2):335–362 (electronic), 1999.

[6] M. W. Berry, S. T. Dumais, and G. W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Review*, 37(4):573–595, Dec. 1995.

[7] M. Brand and K. Huang. A unifying theorem for spectral embedding and clustering. citeseer.ist.psu.edu/669050.html.

[8] T. N. Bui, F. T. Leighton, S. Chaudhuri, and M. Sipser. Graph bisection algorithms with good average case behavior. *Combinatorica*, 7(2):171–191, 1987.

[9] T. N. Bui and B. R. Moon. Genetic algorithms and graph partitioning. *IEEE Transactions on Computers*, 45(7):841–855, July 1996.

[10] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[11] C. Ding. A similarity-based probability model for latent semantic indexing. In *Proceedings of 22nd ACM SIGIRConference*, 1999.

[12] Drineas, Frieze, Kannan, Vempala, and Vinay. Clustering large graphs via the singular value decomposition. *MACHLEARN: Machine Learning*, 56, 2004.

[13] R. Duda and P. Hart. *Pattern Classification and Scnene Analysis*. Wiley, 1973.

[14] C. Eckart and G. Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1:211–218, 1936.

[15] A. Fiat, A. R. Karlin, F. Mcsherry, J. Saia, and Y. Azar. Spectral analysis of data, Dec. 05 2000.

[16] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by image and video content: The QBIC system. *Computer*, 28(9):23–32, Sept. 1995.

[17] M. R. Garey, D. S. Johnson, and S. Stockmeyer. Some simplified *NP*-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[18] G.Golub and C. Reinsch. *Handbook for Matrix Computation II, Linear Algebra*. Springer-Verlag, 1971.

[19] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Mass., 1989.

[20] D. E. Goldberg and G. R. Harik. Learning linkage, Aug. 29 1996.

[21] G. H. Golub and C. F. Van Loan. *Matrix Computation*. John Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, Maryland, third edition, 1996.

[22] J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan, 1975.

[23] R. Horn and C. R. Johnson. *Topics in Matrix Analysis*. Cambridge University Press, Cambridge, UK, 1991.

[24] C. Jordan. Mémoire sur les formes bilinéaires. *Journal de Mathématiques Pures et Appliquées, Deuxième Série*, 19:35–54, 1874.

[25] B. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *Bell Systems Journal*, 49:291–307, 1972.

[26] H. S. Maini, K. G. Mehrotra, M. Mohan, and S. Ranka. Genetic algorithms for graph partitioning and incremental graph partitioning. Technical Report CRPC-TR94504, Center for Research on Parallel Computation, Rice University, Houston, TX, 1994.

[27] J. G. Martin and K. Rasheed. Using singular value decomposition to improve a genetic algorithm's performance. In *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003*, pages 1612–1617, Canberra, 8-12 Dec. 2003. IEEE Press.

[28] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1996. Contains introductory chapter on LCS.

[29] R. S. Michalski. Learnable evolution model: Evolutionary processes guided by machine learning. *Mach. Learn.*, 38(1-2):9–40, 2000.

[30] C. B. Moler and D. Morrison. Singular value analysis of cryptograms. *American Mathematical Monthly*, 90(2):78–87, 1983.

[31] B. Nobel. *Applied Linear Algebra*. Prentice-Hall, 1998.

[32] C. H. Papadimitriou, H. Tamaki, P. Raghavan, and S. Vempala. Latent semantic indexing: A probabilistic analysis. In ACM, editor, *PODS '98. Proceedings of the ACM SIGACT–SIGMOD–SIGART Symposium on Principles of Database Systems, June 1–3, 1998, Seattle, Washington*, pages 159–168, New York, NY 10036, USA, 1998. ACM Press.

[33] K. Sastry and D. E. Goldberg. Probabilistic model building and competent genetic programming. In R. L. Riolo and B. Worzel, editors, *Genetic Programming Theory and Practise*, chapter 13, pages 205–220. Kluwer, 2003.

[34] E. Schmidt. Zur Theorie der linearen und nichtlinearen Integralgleichungen. I Teil. Entwicklung willkürlichen Funktionen nach System vorgeschriebener. *Mathematische Annalen*, 63:433–476, 1907.

[35] G. W. Stewart. On the early history of the singular value decomposition. *SIAM Rev.*, 35(4):551–566, 1993.

[36] J. J. Sylvester. On the reduction of a bilinear quantic of the n^{TH} order to the form of a sum of n products by a double orthogonal substitution. *Messenger of Mathematics*, 19:42–46, 1889.

[37] A. Vetta, R. Kannan, and S. Vempala. On clusterings: Good, bad and spectral, July 31 2001.

[38] H. Weyl. Das asymptotische Verteilungsgesetz der Eigenwerte linearer partieller Differentialgleichungen (mit einer Anwendung auf die Theorie der Hohlraumstrahlung). *Mathematische Annalen*, 71:441–479, 1912.