APPENDIX B

Evolutionary Model for the EPDP

GENETIC ALGORITHM OPERATORS

In order to solve partitioning problems effectively, it is necessary to design operators specifically for the task. Operators that are designed for one type of partitioning problem (Bin Packing) may be ineffective on another type of partitioning problem (Graph Coloring). This appendix is intended to expose the reader to several design concepts that are fundamental in building operators that work effectively on a problem that is to be solved using a GA. In addition, we describe in detail the operators that have been tailored specifically for solving the EPDP.

GA Design Concepts

To build GA operators that work effectively, one should consider the design principle of "Minimal Redundancy" (Radcliffe 1991). Essentially, this concept simply requires an encoding scheme which provides a unique representation for each possible solution vector in the search space. Consider the following representation of a chromosome string for a grouping GA comprised of three different groups (A - Group 1, B - Group 2, C - Group 3):

P1: AABBCC $\longrightarrow$ 112233

P2: BBCCAA $\longrightarrow$ 223311

Clearly, P1 and P2 represent identical groupings of the elements of a chromosome string. Thus, this simple representation does not conform to the "Minimal Representation" design principle.

Next, consider the design principle of "Context Sensitivity/Insensitivity," which applies to the crossover operator. Using the same representation of a chromosome string for a grouping GA comprised of three different groups (A - Group 1, B - Group 2, C - Group 3) and applying single point crossover:

```
P1:   AAB¦ BCC   112 ¦ 233  ⎫   C1:   AAB¦BBC   112 ¦ 223
                            ⎬
P2:   BBC¦ CAA   223 ¦ 311  ⎭   C2:   BCC¦ CAA  233 ¦ 311
```

The design principle of "Context Sensitivity/Insensitivity" states that a crossover operator must produce progeny, meaning that solution vectors representing the same solution to the problem must crossover to produce the same solution. Clearly, neither child above represents the same solution vector as either of its parents although the parents represent identical groupings. We note however, that this result may be acceptable depending upon the problem being solved.

Finally, "Disruption" of genetic operators should be considered in their design. Disruption can be loosely interpreted as the likelihood that good schema in the chromosome string will be destroyed by the operator. The sensitivity of a chosen representation to the effects of disruption is proportional to the length of the chromosome string. In general a binary representation is substantially longer than an integer representation of the same solution vector. Thus, the affects of classical crossover operators are more disruptive to integer (or alphabet) representations, which are commonly used in grouping GAs.

As evolution takes place and good schema begin to appear, the effects of disruption can stifle evolutionary progress. For example, consider the same representation of a chromosome string for a grouping GA comprised of three different groups (A - Group 1, B - Group 2, C - Group 3) and applying random mutation to a chromosome string:

P1: AABBCC 112233 $\longrightarrow$ C1: AABCCC 112333

When considering the mutation operator alone, it is likely that if mutation takes place in the latter portion of the evolution and if P1 were a fit individual, then C1 is highly likely to be unfit compared to other children produced by P1 (assuming a respectful crossover operator). Some researchers have found that "...the classic mutation operator is 'too destructive' once the GA begins to reach a good solution to the grouping problem" (Falkenauer 1994). Again, it is conceded that that above scenario may be acceptable under certain circumstances. For a complete review of the disruption affects of crossover and mutation operators, the reader is referred to (Spears 1998).

TOWARD A DNA BASED EVOLUTIONARY MODEL

In the classical evolutionary math models put forth by scientists such as Holland, Rechenburg, and Koza, the fundamental component included in the model is a gene. The rationale for this is that a gene is the smallest biological unit which carries a specific trait or characteristic of an organism. While science has shown this to be true, it is surprising to find that very few, if any, biologically inspired mathematical models have taken advantage of the constructs which underlie the gene - namely DNA. . In this appendix, we do not pretend that presentation of the biological composition of DNA is complete. It is our objective to provide rationale for introducing analogies that link the DNA structure to the general graph model. For a comprehensive review of the biology of DNA the reader is referred to (Alberts et al. 1994).

Major DNA Components

Science has revealed that DNA works the same in all forms of life. Without DNA life would not exist - not plants, nor animals, not even bacteria. All living tissue in any life form is comprised of DNA. This element which is so fundamental to life is composed of a very simple four code alphabet referred to as DNA bases. They are adenine (A), thymine (T), cytosine (C), and guanine (G). Together, these four bases create all forms of life depending upon the manner in which they are bonded and sequenced. Surprisingly, the rules which they follow to accomplish this seemingly miraculous task are rather simple. There are only four possible conditions in which the DNA bases can bond: A-T, T-A, C-G, and G-C. Bonded DNA bases are referred to as base pairs which are then sequenced to form a DNA strand. DNA strands are extremely long and herein lies the complexity of living organisms. The human description has been estimated to be approximately 3 billion base pairs long.

If we are to understand the process of evolution and attempt to mimic the reproduction process of living biological organisms, is seems reasonable to study the replication process in DNA. After all, it is not the genes of the organism that reproduce, but the DNA in the genes that reproduce. DNA sequences, just like DNA bonds, also follow a very simple set of rules. They are sequenced as base pairs in groups of exactly three and, in this form, are referred to as codons. The number of possible codons (base pairs sequenced into groups of 3) can then be calculated as 4*4*4=64 where each setting in the series can contain any of the 4 DNA bases (A, T, C, or G). The 64 possible codon arrangements are the fundamental building blocks which determine all of the characteristics of any life form (that we are currently aware of). A series of codons will determine the proteins that can be produced by the cells of the host organism. Thus, it is a particular series (or sequence) of codons which we call a gene.

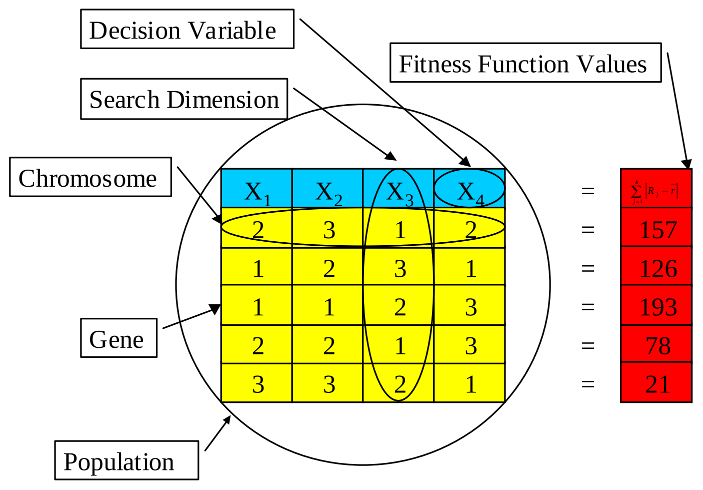
A DNA Based Evolutionary Model

The process by which new sequences in the DNA strand are explored is referred to as DNA replication. An interesting aspect of the recombination process that occurs in all sexually reproducing organisms is that it too follows the simple bonding rules which allows the DNA to replicate. Thus, while new sequences of codons are evolved over time, the fact remains that A-T, T-A, C-G, and G-C still holds before and after the DNA strand has been reproduced in the new host organism.

The Classical Genetic Algorithm Model

The genetic algorithm that was put forth by John Holland is based upon a system of analogies that link a matrix of numbers to biological components. In Holland's original GA the gene values were exclusively binary, however, many useful modifications such as integer representations have since emerged. Figure B.1 shows an integer representation of a classical GA where the decision variables X_i are analogous to the nodes in a graph model and the integer values in the gene represent assignments of the nodes to districts. The fitness function in this model is the absolute revenue deviation function described in (2.1). It is significant to note that contiguity of the nodes assigned to districts must be enforced in this model via penalty functions.

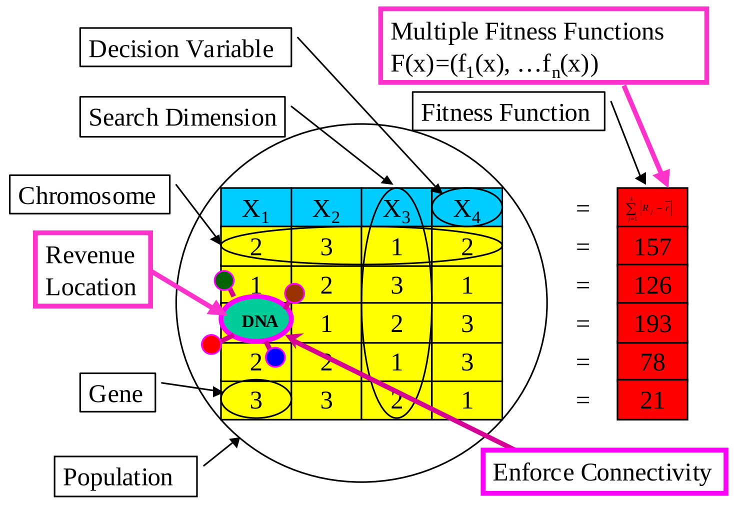
FIGURE B.1
The Classic GA Model



Enforcing Connectivity with DNA Bonds

Enforcing contiguity in a districting plan requires an exponential number of constraints with respect to the number of nodes in the graph model (Mehrotra et al. 1998). In most GA implementations, constraints are enforced via penalty functions. When the size of the constraint set is large, the penalty function method often results in the production of a substantial number of infeasible chromosomes. Thus, a great deal of computational effort is wasted in the process. An alternative approach would be to used the additional constructs illustrated in Figure B.2 based upon DNA bonds.

FIGURE B.2
A DNA based GA Model



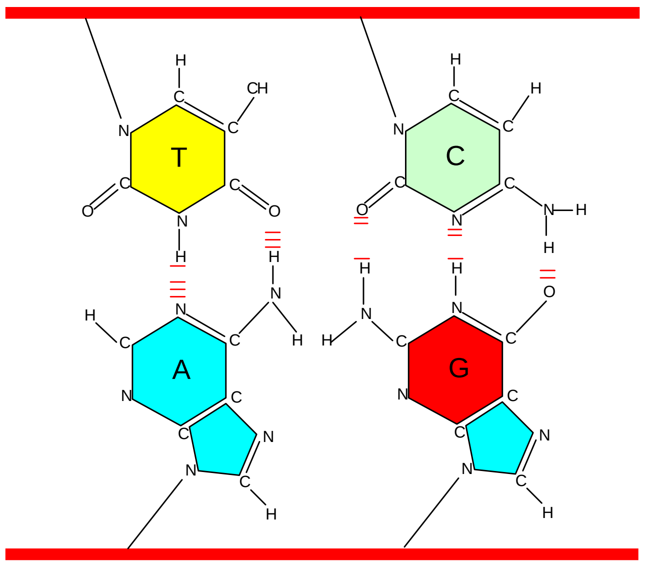
GAs are often regarded as a multi-purpose algorithm because of their widespread applicability. However, a common criticism of GAs is their limited ability to handle highly constrained optimization problems (Ahuja and Orlin 1997). Graph models represent a constrained environment where the GA has competed less favorably than highly specialized techniques (Liepins and Hilliard 1990). A rigorously studied graphical problem where the development of very sophisticated recombination and mutation operators has improved the performance of GAs is the traveling salesman problem (Jog et al.1991, Homaifar et al. 1992, Muhlenbein et al. 1998, Cheng and Gen 1994). A shortcoming of these sophisticated operators is that they are specifically tailored to the problem being solved and are of very limited use when applied to other types of graph modeling problems. It seems only reasonable to expect any modifications to a biologically inspired algorithm to have some biological justification for the change. Without such justification, the modified "genetic algorithm" is really just an algorithm.

We posit that there is biological motivation for the development of new operators that will enhance the capability of the GA to handle graphical problems. However, it requires an additional layer of abstraction to the traditional GA model.

DNA Object Model

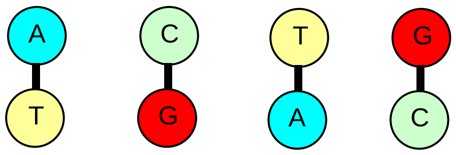
In the preceding section, we described how the gene is comprised of DNA bases. No reasonable person will dispute this. We note that there is a strong resemblance between graph models, which are constructed of nodes connected by edges, and DNA bases which are bonded and sequenced into a DNA strand to provide the gene with the ability to propagate the traits of a host organism. See Figure B.3. The configuration of a graph model can be used as a surrogate for the simple rules with which DNA bases bond into pairs and sequence into codons. In Figure B.3, an example of the bonding configuration in DNA base pairs is shown

FIGURE B.3
DNA Base Structures



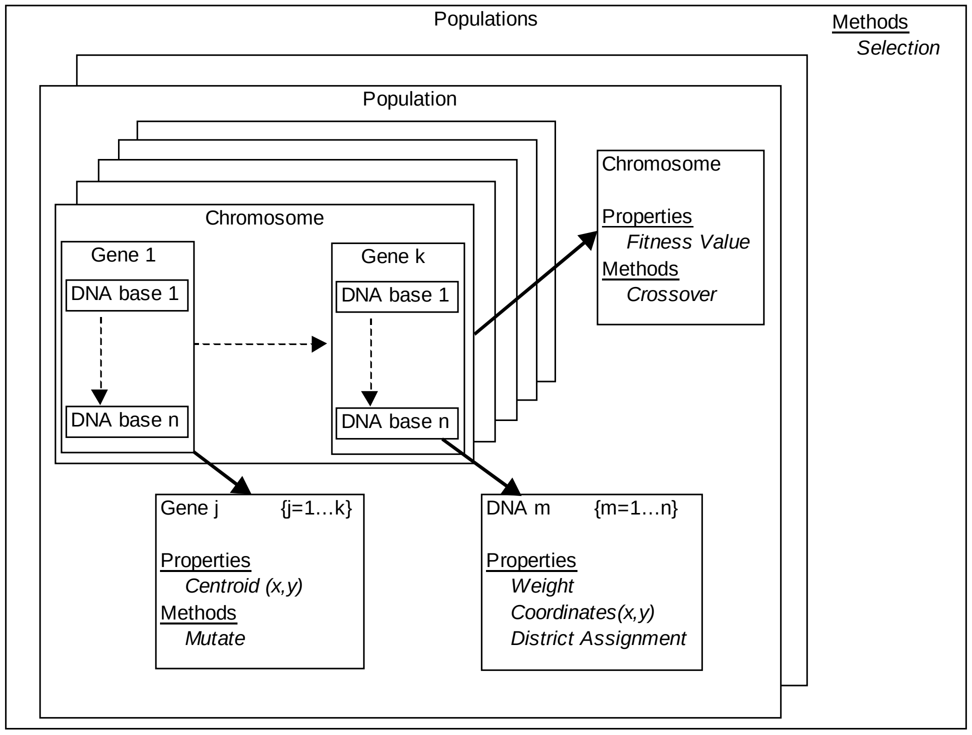
Notice that each DNA base is comprised of molecules of carbon, hydrogen, nitrogen and oxygen. The two thick red lines above and below the bases are the "rails" which sequence the base pairs. In reality, the rails are twisted into a double helix with the DNA base pairs joining the rails together in a physical configuration similar to the rungs on a ladder. The vertical pairs of DNA bases are bonded together with hydrogen molecules to form a specific base pair. The vertical base pairs are not bonded to their adjacent base pairs, but are sequenced together by the upper and lower rails. The base pairs shown in Figure B.3 may also be inverted to form any of the configurations in Figure B.4.

FIGURE B.4
DNA Base Pairs



In our DNA object model we create an additional layer of abstraction beyond the traditional GA model where a collection of DNA bases (nodes) are contained within a gene. It is the unique collection of DNA bases that give the gene a unique characteristic. The DNA bases use their neighbors in the graph model as the simple rules that allow them to bond to other DNA bases. Thus, feasible solutions to highly constrained graphical network problems can be produced without the use of inefficient penalty functions. Figure B.5 depicts the object model hierarchical relationships.

FIGURE B.5
The DNA Object Model



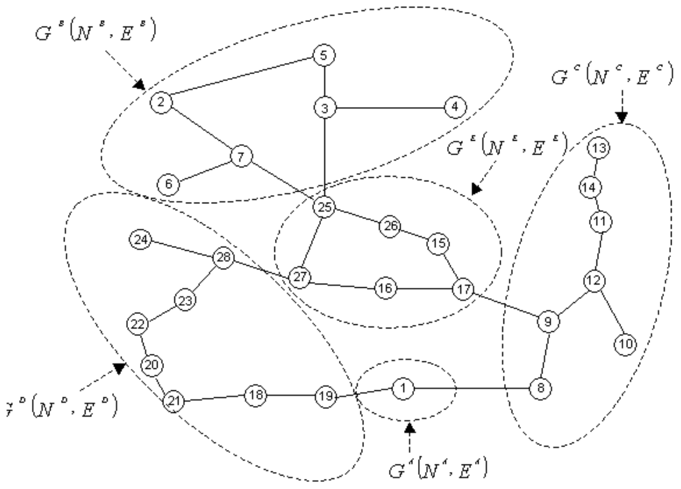
Starting from the top of the hierarchy, the Populations object contains a collection of Population objects. Typically, there exists a parent population and a child population, although the collection can be expanded to include multiple populations in parallel. Note that the Populations object has a selection method that allows it to act upon the Population object. Each Population object contains a collection of Chromosome objects. Each Chromosome object contains a collection of Gene objects. Note that each Chromosome object has a crossover method that allows it act upon the Gene object. Note also the Chromosome object has a property defining its fitness. Each Gene object contains a collection of DNA bases. The Gene object has a mutation method that can act upon the members of the DNA base collection. The Gene object contains a property defining the location of the group centroid on a grid. The DNA base object is at the bottom of the hierarchy. It has properties defining problem specific information such as the weight assigned to a node (revenue potential), the location of the node on a grid, and the district to which it is assigned.

Encoding DNA

The purpose of this section is to describe the operators we use to encode the DNA model for the EPDP. To overcome the pitfalls of context insensitivity and progeny, we propose an encoding scheme that treats Genes (districts) as the fundamental building blocks of the crossover operator and DNA bases (nodes) as the fundamental building blocks of the mutation operator. This results in a collection of respectful crossover operators that effectively make use of the additional layer of abstraction within the genetic structure.

The next step is to determine an appropriate representation for the GA components that are consistent with the above object model. Standard genetic operators are not respectful of good schema when used on graph partitioning problems "as is" and they generally reduce the algorithm to a random search (Hohn and Reeves 1996). To understand the concept of mapping of genetic operators, one must be aware that a "genotype" is the term used to describe the genetic representation and the "phenotype" is the term used to describe the corresponding problem representation. To model the electrical power districting problem, consider the solution shown in Figure B.6 and the subsequent mapping of the solution into a GA representation.

FIGURE B.6
An Example of a Feasible Districting Plan



Mapping of the genotype (Gene Assignment) and phenotype (district) to the object model:

Gene (Object)		*Genotype*		*Phenotype*
$G^A(N^A,E^A)$	=	Gene A	=	District 1
$G^B(N^B,E^B)$	=	Gene B	=	District 2
$G^C(N^C,E^C)$	=	Gene C	=	District 3
$G^D(N^D,E^D)$	=	Gene D	=	District 4
$G^E(N^E,E^E)$	=	Gene E	=	District 5

A chromosome string:

Gene (Object)		*Genotype*		*Phenotype*
$G^AG^BG^CG^DG^E$	=	ABCDE	=	12345

The Crossover Operator

The crossover operator that is implemented using the DNA object model is context insensitive. The DNA bases are bonded according to the bonding rules that are completely defined by the configuration of the graph model. The bonding rules are base pair definitions which correspond to the neighbors connected to each node in the network. Thus, crossover is accomplished without regard to the current label or index assigned to the genotype or phenotype

Progeny is also maintained in the crossover operation by the bonding process of the DNA bases. Genes of parents containing identical sets of DNA bases will always result in a cloned partition. Unbonded members in the selected genes are stored in a temporary gene to be assigned subsequent to the crossover exchange. The result of this representation is an elegant and respectful operator with minimal redundancy in its operation.

Crossover Operator Pseudo-Code for the EPDP

The following is pseudo code describes the crossover procedure implemented via the DNA object model:

```
Crossover Procedure (Pass In: Parent 1, Parent 2)

    Create $G^{temp}(N^{temp},E^{temp})$

    Set Child = new Chromosome

    For Each $G^{'}$ in Parent 1
        Randomly Select DNA base from $G^{j}(N^{j},E^{j})$ from Parent 1 and Seed
        $G^{j}(N^{j},E^{j})$ with the DNA base in Child
    Next

    For Each $G^{'}$ in Child Chromosome

        Let Rnd()=Uniform[0,1]

        If Rnd()< .5 then
            Select $G^{j}(N^{j},E^{j})$ from Parent 1containing the seed in G'
        Else
            Select $G^{j}(N^{j},E^{j})$ from Parent 2 containing the seed in G'
        End if

        Do
            Bond DNA bases in $G^{j}(N^{j},E^{j})$ from Parent with matching DNA
            bases (ie. neighboring nodes) in $G^{j}(N^{j},E^{j})$ in Child
        Loop Until no more matching DNA bases exist in $G^{j}(N^{j},E^{j})$ from Parent

        If $G^{j}(N^{j},E^{j})$ from Parent $\neq \varnothing$ Then
            Place all remaining DNA bases in $G^{temp}$
        End If

    Next

    If $G^{temp} \neq \varnothing$ Then
        For Each DNA base in $G^{temp}$
            Do
                Randomly allocate DNA base to Child $G^{j}(N^{j},E^{j})$
            Loop until $G^{temp} = \varnothing$
        Next
    End If

End Crossover Procedure
```

The Mutation Operator

The mutation operation involves a series of neighborhood moves that exchanges a single node with a neighboring partition. Thus, the mutation operator is limited to nodes that exist on the boundary between two partitions. We note that a boundary node must meet the sufficiency conditions for swapping as described in the SA swap procedure in appendix A. The result is an operator that enumerates all feasible solutions that are exactly one degree of separation away from the current configuration. Following the enumeration of neighboring solution vectors, the best configuration is retained. Thus, the mutation operator is a hill-climbing operator that is designed to optimize the DNA strand of the child chromosome.

Mutation Operator Pseudo-Code for the EPDP

The following is pseudo code describes the mutation procedure implemented via the DNA object model:

```
Mutation Procedure (Pass In: Child)

    For each G' in Child

        Find all Boundary Nodes N'

        For each N' in Child

            Swap N' with neighboring district

                Evaluate Child

            Reverse Swap

        next

    next

    Restore best Swap configuration

End Mutation Procedure
```