A Crossover Operator for the k-anonymity Problem

Monte Lunacek Darrell Whitley Indrakshi Ray

Department of Computer Science
Colorado State University
Fort Collins, CO 80521
{lunacek,whitley,iray}@cs.colostate.edu

ABSTRACT

Recent dissemination of personal data has created an important optimization problem: what is the minimal transformation of a dataset that is needed to guarantee the anonymity of the underlying individuals? One natural representation for this problem is a bit-string, which makes a *genetic algorithm* a logical choice for optimization. Unfortunately, under certain realistic conditions, not all bit combinations will represent valid solutions. This means that in many instances, useful solutions are sparse in the search space. We implement a new crossover operator that preserves valid solutions under this representation. Our results show that this reproductive strategy is more efficient, effective, and robust than previous work. We also investigate how the population size and uniqueness can affect the performance of genetic search on this application.

Categories and Subject Descriptors

I.2.8 [**Problem Solving, Control Methods, and Search**]: [heuristic methods]; G.1.6 [**Optimization**]: [global optimization]

General Terms

Performance

Keywords

Genetic algorithms, problem representation, K-anonymity

1. INTRODUCTION

Recent dissemination of personal data has raised an important privacy concern: how can privately held data be released in a way that is useful, while protecting the identity of the underlying individual? Unfortunately, simply removing the attributes that explicitly identify a person, such as name and social security number, does not always provide sufficient privacy. Surprisingly, Sweewey discovered that 87% of the United States population showed a high probability of being uniquely identified based solely on their zip-code, gender and data of birth [7]. These attributes, called *quasi-identifiers*, can be used in conjunction with other publicly available information to re-identify an individual.

Sweeney performed the following experiment, which exemplifies the use of *quasi-identifiers* to re-identify an individual [6]. First, Sweeney acquired a medical dataset from the Group Insurance Commission (GIC) of Massachusetts that contained patient's *zip code*, *date of birth*, and *gender*. This was assumed to be anonymous, since explicit attributes were removed. Sweeney also obtained the voter registration list for Cambridge Massachusetts. The governor of Massachusetts, William Weld, was one of only six people that lived in Cambridge whose medical records where in the GIC database. Surprisingly, only three of these people were male and Governor Weld was the only person in his *zip code* area.

One way of protecting an individuals privacy is to ensure that the released data adheres to the $k-$anonymity property [6]. Each record in a $k-$anonymous dataset is indistinguishable from at least $k-1$ other records. In other words, every unique record in the original dataset is transformed such that it is identical to at least $k-1$ other records. This ensures that there are at least k individuals that correspond to the same record. Intuitively, a higher k value provides a greater level of anonymity. In the previous example, the GIC medical database of Massachusetts would not be considered anonymous because the row containing Governor Weld's quasi-identifiers (*zip code*, *date of birth*, and *gender*) is unique.

Generalization and *suppression* are two techniques used to create $k-$anonymous datasets. Generalization takes an attribute value and replaces it with a more general – less specific – value. For example, an individual's birth date could be generalized from a specific day to a more general form that only includes the month and year. When a specific value is not released, it is suppressed (denoted by *). Applying generalization to the original dataset should decrease the number of unique records.

Information is lost when values are generalized. Furthermore, datasets that contain more information are presumably more useful. This creates the following optimization problem: what generalization results in the least amount of information loss, while satisfying the $k-$anonymity property? While several methods have been proposed to answer this question, there is currently no single best choice. This is partly due to the amount of attention this problem has received; there are only a few comparative studies and quality

GECCO'06, July 8–12, 2006, Seattle, Washington, USA.

of empirical evidence is limited. Currently, only one practical benchmark function has been identified and used.

Problem representation is a key component in successful genetic search. One natural representation for the *k-anonymity* problem is a bit-string; ordered elements are generalized based on the position and value of a single bit. We argue that it is realistic to constrain some of the generalizations that exist under this representation. That is, not all of the bit combinations represent solutions that are actually meaningful. Iyengar previously applied a genetic algorithm to the k-anonymity problem using Booker's *reduced surrogate* 2-point crossover operator [4]. Each offspring was potentially modified to ensure it was a valid solution.

In this paper, we implement a new crossover operator that preserves valid solutions under this problem representation. We show that this reproductive heuristic results in more efficient, effective, and robust overall solutions than previous work. In Iyengar's previous work, the performance of his genetic algorithm was not the main objective [4]. We extend and compliment this work by investigating how the population size and uniqueness can affect performance.

The remainder of the paper is organized in the following way. In the next section we review different approaches to generalization and briefly discuss related algorithms. We argue that it is often necessary and sometimes desirable to constrain the number of valid generalizations associated with a given attribute. In Section 3, we discuss how the parameters of a genetic algorithm can affect its performance and introduce our new crossover operator. Then, in Section 4, we show that our crossover operator is both more effective and efficient on the problem we tested. Finally, we conclude the paper.

2. BACKGROUND

The exact way in which attributes are generalized is an important factor in finding effective k-anonymous dataset. Sweeney defines a *value generalization hierarchy* that maps each unique value within an attribute domain to more general value [6]. Consider the following example, where the domain of the *country* attribute is {US, CAN, PERU, BRAZIL}. The next level of generalization could be the continent that each country belongs to. This results in the more general domain {NORTH AMERICA, SOUTH AMERICA}. Figure 1 shows the resulting hierarchy. When the attribute is generalized using this scheme, every attribute value is changed to a more general domain.

Iyengar points out that forcing each attribute value to have the same *level* of generalization is too restrictive and can result in more information loss than is necessary [4]. As an alternative, Iyengar proposes a more flexible model that defines generalizations based on partitions of an imposed ordering of the attribute values [4]. This method maps to Sweeney's approach, but is less restrictive. In the previous example, Sweeney's more general domain could be modeled as {US, CAN}, {PERU, BRAZIL}, where a partition between CAN and BRAZIL separates the four countries into two distinct classes. However, there are no restrictions that prevent US and CAN from being generalized while BRAZIL and PERU remain individually partitioned. This has the advantage of allowing a finer granularity of generalization that potentially preserves more information. Unfortunately, the number of generalizations significantly increases the search space.

Generalization can also be enforced at the *cell level* [6]. A cell is the intersection of an attribute and a row. Unlike attribute level generalization, cell level generalization does not guarantee a unique mapping between the original values and their more general form. This implies that some of the BRAZIL values in the country attribute can be mapped to the SOUTH AMERICA label, while others cannot. Iyengar argues that this creates confusing and complex relationships within a given attribute [4]. Furthermore, the added flexibility of cell level generalization dramatically increases the search space. Meyerson and Williams showed that cell level suppression (the original value or *) is NP-hard [5].

2.1 Measuring Fitness

In order to decide which generalization is best, it is necessary to quantify how much information has been lost. The assumption is that datasets that retain more information will be the most useful. In this paper, we use the general loss metric proposed by Iyengar [4]. Each cell associated with a given attribute is assigned a penalty based on the extent to which it has been generalized. The penalty assigned to each cell value is proportional to the size of its partition. A non-generalized value belongs to a partition size of one. When BRAZIL and PERU are generalized, the partition size is two. Let $P_{a,r}$ be the number of values in a partition associated with attribute a for a value in row r. Let N_a be the number of distinct values in the attribute domain. The penalty for the information loss associated with this cell is:

$$\text{loss} = \frac{P_{a,r} - 1}{N_a - 1}$$

Subtracting one ensures that no penalty is assigned to a non-generalized value. A suppressed attribute will have a value of one because $P_{a,r} = N_a$. Referring to the country example in Figure 1, a cell that generalizes BRAZIL to SOUTH AMERICA will receive a penalty of $(2-1)/(4-1) = 1/3$. The loss associated with a given attribute is the sum of all its cells. The total loss for a given generalization is the sum of all the attribute loss.

$$\text{generalization loss} = \sum_a \frac{1}{N_a - 1} \sum_r P_{a,r} - 1$$

Some rows in a database may be extremely unique and require a larger degree of generalization in order to achieve the k-anonymity property. The loss associated with suppressing these "outlier" rows is probably less than the overall loss due to the excessive amount of generalization required. For this reason, we use Bayardo and Agrawal's model for row suppression [1]. First, we construct a transformed dataset based on a specific generalization. We then suppress all rows that violate the k-anonymity property. Specifically, we suppress any rows that do not belong to an equivalence class of size k. A suppressed cell receives a penalty of one. Therefore, all suppressed rows receive a penalty equal to the number of attributes (assuming each row has N cells, where N is the number of attributes). The overall information lost is calculated by adding the loss due to generalization and the loss due to suppression.

There is a subtle bias in the loss estimation that is more pronounced as the distribution of domain values becomes less uniform. Consider two domains. Assume the first has five distinct values and the second has ten. One generalization may combine the first two values from each domain into

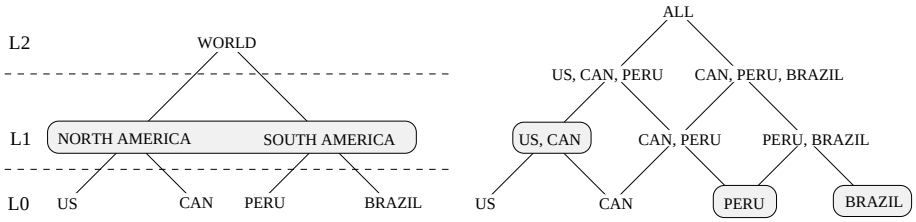

Figure 1: A value generalization hierarchy (left) requires every attribute value to be generalized to the same level. For example, US and CAN are generalized to L1. This implies that BRAZIL and PERU must also be mapped to L1. Iyengar's more flexible generalization model (right) does not force this constraint. For example, US and CAN may be generalized while BRAZIL and PERU remain individually partitioned.

a single partitions. Generalized values associated with the first domain will be penalized $(2-1)/(5-1) = 1/4 = 0.25$, while values from the second domain will receive a penalty of $(2-1)/(10-1) = 1/9 = 0.11$. Under a uniform distribution, we would expect about 2/5 of the rows to be generalized from the first attribute and only 2/10 from the second. However, when the distribution is not uniform, the second attribute could be penalized significantly for essentially the same amount of loss. This bias comes from the number of distinct elements in each domain. Attributes with more elements will tend to receive a lower penalty for a similar level of generalization independent of the underlying attribute distribution. As a result, search may tend to focus more on the attributes that have a greater affect on the overall loss metric. Despite this concern, we use the standard metric outlined above. For the remainder of the paper, we will refer to the overall loss as *fitness*.

2.2 Algorithms for the k-anonymity Problem

Several practical algorithms have been proposed as a means for finding effective solutions to the k-anonymity problem. The μ-argus algorithm breaks the problem into smaller parts that are constructed from different combinations of *quasi-identifying* attributes [3]. The rows from these smaller combined sets that do not meet the k-anonymity property are marked as outliers. Sweeney asserts that not all the solutions found by μ-argus are actually k-anonymous; there may exist larger combinations of quasi-identifiers that are actually unique [6]. It is unclear how efficient or effective the μ-argus algorithm would be if it were revised to address this shortcoming.

Sweeney's datafly algorithm looks at the frequency of each distinct row and generalizes the entire attribute containing the most distinct values. Although the algorithm is efficient, Sweeney shows that it performs more generalization than is actually optimal [6].

Iyengar used a genetic algorithm to explore the larger search space that resulted from his more flexible generalization model [4]. Iyengar's paper is unique because it looked at how useful transformed data sets actually are by comparing their classification error with that of the original. However, the performance of genetic algorithms was not the papers main focus. Iyengar points out that, "For most applications, we do not see any real time requirements for the task of transforming the data to satisfy privacy" [4]. While efficiency may not be paramount, we are still concerned with the effectiveness of the solutions we do find. And, finding them quicker seems to be universally more desirable.

Bayardo and Agrawal comment that all of the above approaches are incomplete. Incomplete algorithms do not guarantee that the solutions they find will be optimal, or even near optimal [1]. As a result, Bayardo and Agrawal propose a complete search method that iteratively constructs more specific (less generalized) solutions starting from a completely generalized dataset. In this way, each node in their search tree represents a valid solution. In a depth-first manner, they traverse the tree and prune away regions that are determined to be sub-optimal. This algorithm is less efficient for small k values [1].

There is no empirically obvious best approach. While Bayardo and Agrawal's algorithm appears to be very efficient and complete for large k, it is unclear how useful datasets constructed with large k are. That is, large k values may require significantly more generalization (and information loss), potentially render a transformed dataset less useful. Our work extends Iyengar's paper by directly addressing the performance of a genetic algorithm applied to this problem. We focus on a small k value of 15.

2.3 Constrained Value Generalization

Iyengar's generalization method is a reasonable compromise between search space size and the granularity at which each value can be generalized. Furthermore, some attribute domains have a logical ordering, which make this method a natural choice for representing generalizations. For example, consider an age attribute with domain values {20, 30, 40, 50, 60, 70, 80}. The way in which these values may be generalized is constrained in two ways. First, the imposed ordering prohibits two non-adjacent values from ever appearing exclusively in a generalization. That is, 50 cannot be generalized with 70 *unless* 60 is included. Second, valid generalizations force a unique mapping between each value and its generalization. Therefore, 30 cannot be mapped to both of the more general nodes {20-30} and {30-40}. Because a logical ordering exists, the constraints imposed by this method generalize these values in a meaningful way.

Categorical data may not always have a logical ordering. In this case, the user must specify how the data should be ordered. Consider the *work-class* attribute displayed in Figure 2. In this example, if FEDERAL, STATE and LOCAL were to be generalized, the least generalized value would be the GOVERNMENT label. There is a subtle inefficiency here. Not all the partitions defined under this categorical ordering are valid. For example, although NOT INC and FEDERAL are adjacent neighbors, there does not exist a node in the tree that generalizes these two values. The only way to make these

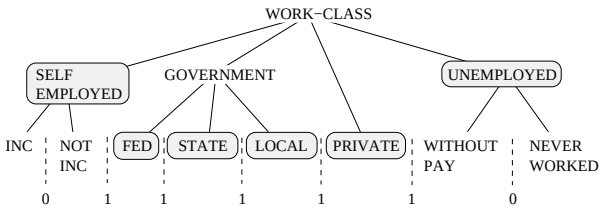

Figure 2: The taxonomy tree for the workclass attribute. Not all partitions are valid.

values indistinguishable is to generalize all the values in the attribute to the WORK CLASS label. Although the number of possible generalizations is $2^7 = 128$, the actual number of valid partitions is only 9. This type of constrained generalization can dramatically reduce the search space of possible solutions. In this example, the search space decreases 93%.

One drawback to this representation is that an imposed ordering for some attribute values can be arbitrary or bias. In other words, some domain values exhibit a higher dimensional relationship and there is not a fair and unbiased way to map them to a one dimensional ordering. To make this point more clear, consider the following countries, VIETNAM, LAOS, CAMBODIA and THAILAND, that belong to the country attribute. Figure 3 shows a map of these countries the corresponding two dimensional graph representation, where an edge indicates that two nodes border each other. The question we ask is: how can we order these countries such that neighboring countries can be generalized together? Unfortunately, we cannot. If we order the countries {THAILAND, CAMBODIA, LAOS, VIETNAM}, then neighboring countries THAILAND and LAOS cannot be generalized *unless* CAMBODIA is included. Arguably, the most fair and unbiased way to generalize these countries is to impose a constraint such that the least generalized value is a label that groups all these countries together.

Even when a logical ordering of values exists, it may be desirable to apply additional constraints such that search spends more time exploring the generalizations that are actually useful. For example, the education attribute is categorical, yet it has a logical ordering based on grade level. However, distinguishing between 9th and 10th grade may be undesirable if 11th and 12th grade are generalized together. In this case, we may want to constrain the valid partitions so that HIGH SCHOOL is the least generalized label. This forces the transformed data to be produced in a constrained way based on what we predetermine to be meaningful. The loss associated with this additional constraint may allow other attributes, that we are less certain about, to have more specific values.

In summary, it is often necessary to constrain the values of categorical data. This happens when either the taxonomy tree for the attribute is sparse, or when there is no fair and unbiased way of determining a logical ordering. Sometimes it may even be desirable to constrain numeric or categorical values even when a logical ordering exists.

3. GENETIC ALGORITHMS

Genetic algorithms are population-based search methods that use an evolution-like heuristic to find competitive solutions within a search space. Most genetic algorithms represent individuals using finite bit strings, called *genotypes*.

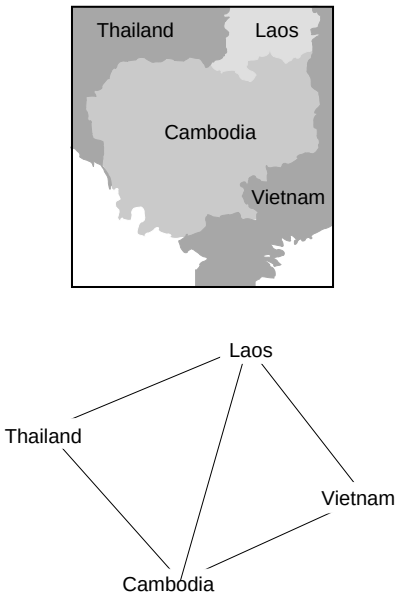

Figure 3: A map of VIETNAM, LAOS, CAMBODIA and THAILAND (left) and the corresponding graph (right). There is no fair way to create a one dimensional ordering.

Through a process of *selection* and *reproduction* the current population of individuals is transformed into a new generation. Selection decides which individuals in the current population will have a chance to reproduce. Reproduction usually involves both crossover (recombination) and mutation. In most genetic algorithms, the *crossover rate* is set much higher than the *mutation rate*, making the *crossover* operator responsible for more of the transformation. The crossover operator combines two selected parents by splitting their strings and concatenating specific pieces back together. Mutation randomly changes part of an individual as a means of exploration and diversification.

Each component in the above description can effect the performance of a genetic algorithm. Specifically, *selection*, *reproduction*, and the *population* can all have a dramatic impact on the effectiveness of genetic search. The effect of reproduction on performance is the most intuitive of the three. If the crossover operator fails to construct more effective solutions, it will be difficult for a genetic algorithm to be very efficient or effective. The way in which selection and population alter performance is less obvious.

The goal of selection is to allocate more reproductive opportunities to above average individuals in the population. The bias selection exhibits is called *selective pressure*. When selective pressure is too strong, highly fit individuals are selected more often, which can cause the population to prematurely converge. That is, high selective pressure tends to *exploit* the best individuals in the population. Over time, this focuses search in a restricted, and potentially sub-optimal, region of the search space. On the other hand, selective pressure can be too low and cause search to stagnate. In this case, genetic algorithms tend to *explore* the search space too much and cannot decide which regions of the search space tend to have most effective solutions. Unfortunately,

maintaining a consistent selective pressure is difficult when parents are selected with a probability that is proportional to their fitness. Whitley shows that allocating reproductive trials according to an individuals *rank* is one way of controlling the variability in selective pressure that occurs when *fitness proportional* selection is used[8].

The size of the population can also effect search performance. Smaller populations tend to have less genetic diversity. That is, it is difficult for a small population to adequately sample the search space. Having less genetic diversity can cause a bias in *where* a genetic algorithm will converge. On the other hand, larger populations tend to have more genetic diversity and a lower selective pressure. These two properties tend to encourage exploration in a larger portion of the search space. However, large populations are not necessarily diverse; too much selective pressure can cause an initially diverse population too converge to quickly.

GENITOR is one of the more successful variations of the traditional genetic algorithm [8]. GENITOR is different from the canonical GA in that it only selects two parents at a time according to rank. This helps control selective pressure. Instead of creating an intermediate population, a newly created offspring is immediately placed back into the current population if its fitness is better than that of the worst individual. As a result, offspring must compete with the current population in order to survive. This also implies that the best individuals found during search will remain in the population.

3.1 Crossover and Mutation

The crossover operator is responsible for most of the exploration that occurs during search. Booker noticed that crossover often becomes less effective as the population of bit strings become more similar[2]. As a result, Booker proposed using 2-point crossover on *reduced surrogates* instead of the entire parent strings. The parent strings are reduced in the sense that the crossover points only fall on bits where the two parents disagree. For example, consider the following two parents:

1 0 1 1 0 1 0 1 1 1 and 1 0 0 0 1 0 0 1 1 1

If we cross out the bit segments where the two parents agree, we are left with their reduced surrogates. Choosing two crossover points from these reduced parents, denoted by a dot ($\cdot$), results in the following new child. For clarity, x and y are substitute for 0 and 1 in the second string.

~~1 0~~ 1 · 1 0 · 1 ~~0 1 1 1~~

~~x y~~ y · y x · y ~~y x x x~~

1 0 1 · y x · 1 0 1 1 1

Whitley points out that reduced surrogates tend to focus search towards those regions where the two parent strings disagree [8].

Notice that crossover points occurring in four rightmost bit positions will render the child string identical to either of the parents. In other words, the child cannot be different from one of its parents when crossover exchanges identical segments. There are two important implications if this were to happen. First, crossover has failed to explore any new areas of the search space. This is inefficient because the crossover operator has used up an evaluation, but the population has not gained any new information. Second, and perhaps more serious, the new child string will replace a lower fitness individual and create a duplicate in the population. This will increase selective pressure and also reduce the diversity in the population. As discussed previously, this can lead to premature convergence.

When a population begins to converge, there tends to be less difference between the individual bit-strings in the population. In the extreme case, each member may agree on a single bit position. For example, every individual could have a one in the rightmost position (**...**1). When this happens, crossover is confined to exploring half of the search space defined under this instance. Adding a small degree of mutation can restore some of the diversity lost by convergence.

3.2 Constrained Generalization

Valid partitions of an attribute value ordering can naturally be represented by a bit string, where a single bit divides the ordered elements. If the bit value is equal to zero, then the two elements are generalized together. A one bit value implies that no generalization occurs. This representation also forces each value to have a unique mapping between itself and its generalization. That is, under this representation, it is impossible for any path extending from a leaf node to the root of the tree to pass through more than one generalized node[4].

A bit-string generalized representation also makes defining an overall solution easy; the bit-strings from each individual attribute are simply concatenated together. The result is a bit-string that exactly represents the generalized solution. Unfortunately, when an ordered generalization is constrained, not all bit combinations represent valid solutions (e.g. the work class attribute). The implication here is that crossover and mutation will likely create invalid solutions. As a result, Iyengar adds an additional step that replaces invalid offspring with the closest valid solution. In our implementation, we define closest to mean minimal Hamming distance.

To clarify this point, consider the following *work class* example. For simplicity, we have replaced the leaf nodes with single letters. For instance, INC = A, NOT INC = B *ect.* Given the following two valid parent solutions, reduced surrogate crossover proceeds to identify the bit segments that are different. In this example, the crossover point occurs between the fourth and fifth bit and produces the following offspring.

AB–C–D–E–F–GH	=	~~0 1~~ 1 · 1 ~~1 1~~ 0
AB–CDE–F–G–H	=	~~0 1~~ 0 · 0 ~~1 1~~ 1
	=	0 1 1 · 0 1 1 1
	=	AB–C–DE–F–G–H

Notice the offspring contains the segment C–DE. Looking at the work class taxonomy tree in Figure 2, it is clear that D = STATE and E = LOCAL cannot be generalized together without C = FEDERAL. As a result, this solution must be transformed into the closest valid solution. The two valid solutions that are Hamming distance one from the offspring are: AB–C–D–E–F–G–H, and the original parent AB–CDE–F–G–H. Consequently, the effectiveness of crossover is somewhat random. Either crossover has no affect, or it creates a new offspring in an unintended way.

We implemented a crossover operator that preserves valid solutions when generalizations are constrained. It differs from the reduced surrogate two point operator in some counter intuitive ways. First, instead of choosing crossover points that fall on the bits that disagree, we specify that crossover must occur in places where the bits are same and are equal to one. Second, not all crossover points are accepted under the previous definition. We constrain the number of crossover points in the following way. We identify segments in each of the parents where the bits agree and disagree. This is similar to reducing the parent strings. Only crossover points that occur after a segment of agreement followed by a segment of disagreement are considered. This ensures that crossover will combine different blocks from each parent. Using the previous example, our crossover operator would produce the following valid offspring.

$$\begin{aligned} \text{AB–C–D–E–F–GH} &= 0\,1\,1\,1\,1 \cdot 1\,0 \\ \text{AB–CDE–F–G–H} &= 0\,1\,0\,0\,1 \cdot 1\,1 \\ &= 0\,1\,1\,1\,1 \cdot 1\,1 \\ &= \text{AB–C–D–E–F–G–H} \end{aligned}$$

Notice that crossover occurs at the first position where 1) the two strings agree, 2) have a value of one, and 3) the previous blocks have agreement and disagreement. To understand why this preserves valid solutions, consider the character strings in the previous example. Assuming we have a valid solution, a partition break, indicated by a dash (–), implies that generalizations on either side are also valid. Because the constraints are ordered, and generalizations must include adjacent values, we know that the generalizations on one side of the partition have no effect on the other side. This means that combinations occurring at partition points shared by both character strings will always create valid solutions.

Mutation on a constrained generalized attribute often has no effect or is very disruptive. If flipping a single bit on a valid solution renders the offspring invalid, the closest valid solution is the original string. However, this may not be the only valid solution that is Hamming distance one from the invalid offspring. If the original solution is chosen, mutation has effectively done nothing. If another valid solution is chosen, it must be Hamming distance two from the original string, which means that we have applied more mutation than desired. We perform crossover on a random individual as a way of introducing diversity and still preserving solution validity.

4. EMPIRICAL RESULTS

We have argued that constrained generalizations are necessary and often desirable. The advantage of our crossover operator will be more pronounced when a greater number of attributes are constrained. If none of the attributes are constrained, then all bit strings are valid and crossover and mutation do not disrupt solution validity at all. In order to demonstrate the advantage of our crossover operator, we constrained half of the eight attributes in the adult census database (described by Iyengar [4]). In particular, we constrained the WORK CLASS, EDUCATION, MARITAL STATUS, and COUNTRY attributes in a logical and meaningful way. The AGE, RACE, GENDER, and OCCUPATION attributes were left unconstrained. Each solution required 92 bits (24 unconstrained and 68 constrained). The constraints reduced the search space from 2^{92} to a size on the order of 2^{42}. We prepared the adult database as described by both Iyengar [4] and Baryardo and Agarwal [1]. That is, we removed any rows with missing values, which left 30,162 unique records. Because the experiments take a considerable amount of time, we randomly selected 15,000 rows to be our test database.

We ran two versions of GENITOR: one used our new crossover operator and the other used the traditional 2-point reduced surrogate operator. We refer to these as NEW and TRAD respectively. Each version ran for 15 trials on our test database using three population sizes: 200, 500, and 1,000. Every trial was allocated exactly 30,000 evaluations. Duplicate values were excluded from the population. We also tested our new crossover operator on the test database, but did not explicitly exclude duplicates.

Figure 4 compares the NEW and TRAD crossover operators for the various population sizes. In each case, the NEW crossover operator converges faster to more effective solutions. The box plots on the right indicate that the NEW crossover tends to find the same local optima, independent of population size. On the other hand, TRAD crossover converges to a wider range of local optima. When the population sizes are 200 and 1,000, the TRAD crossover operator often converges to a local optima that are similarly in quality to those found by the NEW crossover operator. That is, the median solution values are fairly close. The key distinction here is that our NEW crossover operator is more robust. Then the population size is 500, the median value is much closer to the average. We do not have an explanation for this.

Figure 4 also compares the effects of population size. Notice as the population size increases, the convergence time also increases. One explanation for this is that larger populations tend to have lower selective pressure that results in more exploration of the search space. However, there is no relationship between population and the solution effectiveness. The the NEW crossover operator converges to nearly the exact same solution each time, independent of the population size. While the solutions vary using TRAD crossover, there does not appear to be an obvious pattern between population size and the effectiveness of the solution.

Excluding duplicates from the population helps prevent selective pressure from increasing. Figure 5 shows the effects of duplicates on the convergence properties for various population sizes. When duplicates are not explicitly excluded, the convergence properties are somewhat expected; smaller populations converge faster to less effective solutions. This is because higher selective pressure in smaller populations tends to exploit the less diverse population. One of the trials with a population size of 200 converged to a solution outside the range of the box plot on the right. Disallowing duplicates in the population appears to result in a faster convergence to solutions of similar quality.

5. CONCLUSIONS AND FUTURE WORK

Constrained generalizations are often necessary when using Iyengar's more flexible generalization model. Even when a logical ordering of values exists, it is often desirable to apply additional constraints. Sometimes adding constraints is the least biased way of representing more complex relationships that do not directly map to a one dimensional ordering.

In this paper, we have looked at the factors that affect genetic algorithm performance on the *k-anonymity* problem.

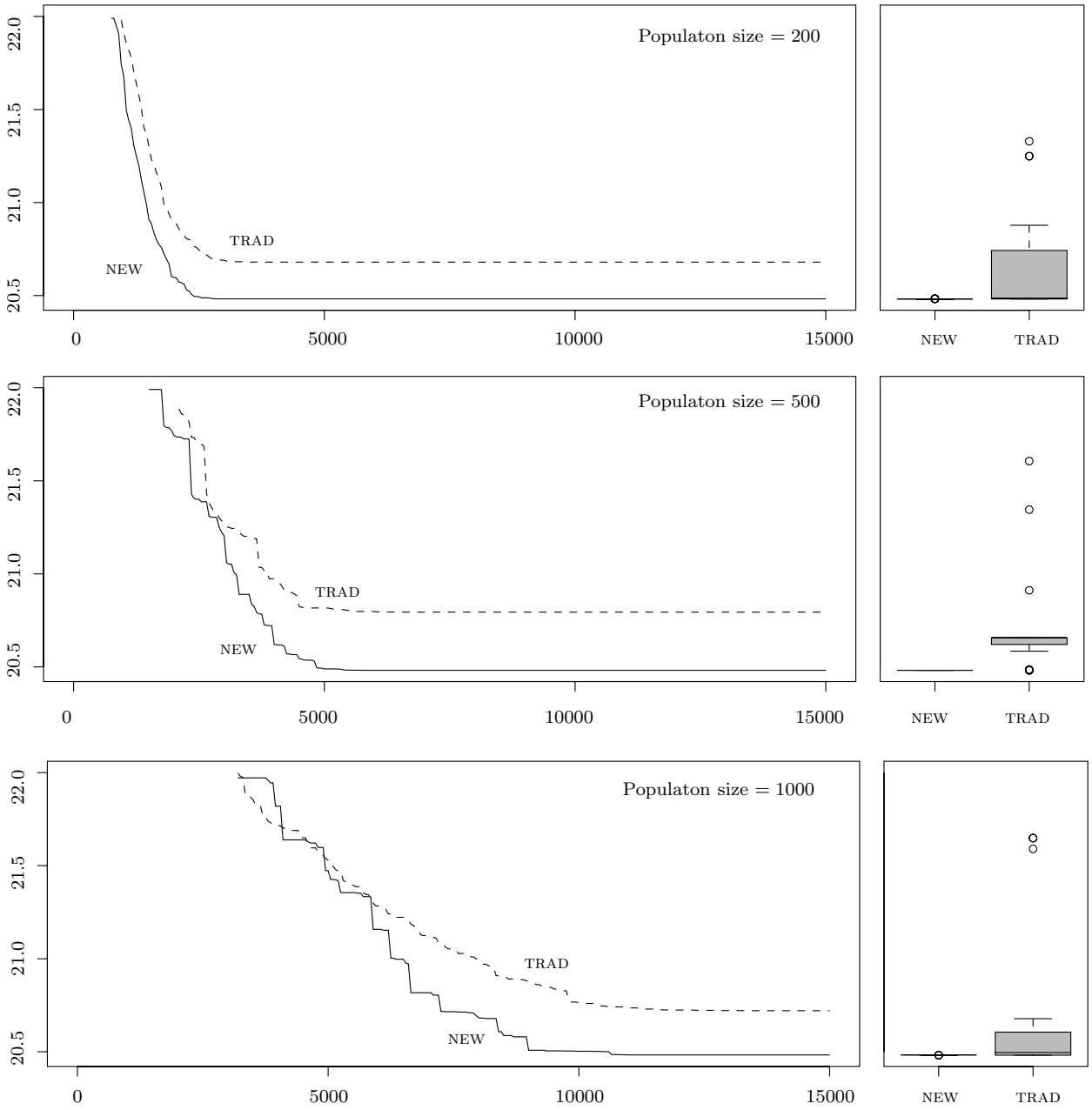

Figure 4: Fitness (x1000) vs. Evaluations: The effects of crossover for different populations sizes. On average, our NEW crossover operator finds more effective solutions than the standard 2-point reduced surrogate crossover operator. The box plots on the right contain the best solutions from all 20 trials. The flat nature of the box plot for the NEW crossover operator suggests that finding solutions to this problem is not very difficult. Larger populations converge to the nearly the same solutions when using the NEW crossover operator.

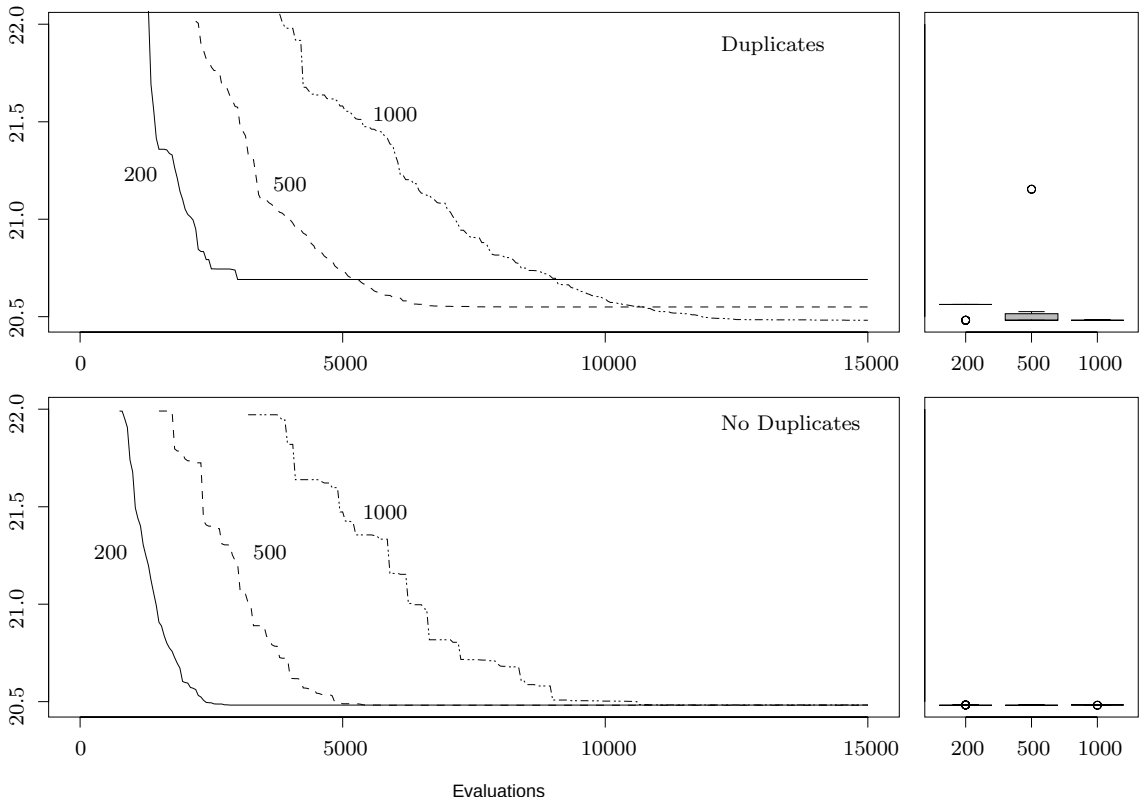

Figure 5: Fitness (x1000) vs. Evaluations: The effect of population size and duplicate values. Duplicate values increase selective pressure resulting in convergence to less effective solutions.

Specifically, we showed that the way in which new solutions are constructed can have a significant impart when generalizations are constrained. The size of the population also plays an important role. We have shown that smaller populations tend to converge more quickly. Using our new crossover operator, this does not effect the solution quality on the benchmark function we used. Furthermore, disallowing duplicates from the population resulted in faster convergence to and more effective solutions.

Future work will explore other potential benchmark problems. Hopefully, our crossover operator will behave similarly on problems with different underlying distributions. This will allow our empirical work to be more meaningful, robust and convincing. Until new benchmark problems are identified, our work, and other empirical work in this area, remains extremely preliminary.

Iyengar used a population of 5,000. We believe that this results in a much slower convergence time than necessary. Duplicating his results, and comparing these with smaller populations will help shed some light on this issue.

6. REFERENCES

[1] R. J. Bayardo and R. Agrawal. Data privacy through optimal k-anonymization. In *Proceedings of the International Conference on Data Engineering*, pages 217–228, Washington, DC, USA, 2005.

[2] L. Booker. *Improving Search in Genetic Algorithms*. Pitman, London, and Morgan Kaufman: Los Altos., 1987.

[3] A. Hundepool and L. Willenborg. Mu and Tao Argus: Software for statistical disclosure control. In *In Proceedings of Third International Seminar on Statistical Confidentiality*, 1996.

[4] V. S. Iyengar. Transforming data to satisfy privacy constraints. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pages 279 – 288, 2002.

[5] A. Meyerson and R. Williams. On the complexity of optimal k-anonymity. In *In Proceedings of the Symposium on the Principles of Database Systems*, pages 223–228, 2004.

[6] L. Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *International Journal on Uncertainty, Fuzziness, and Knowledge-based Systems*, 10:571–588, 2002.

[7] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal on Uncertainty, Fuzziness, and Knowledge-based Systems*, 10:557–570, 2002.

[8] D. Whitley. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *Proceedings of the International Conference on Genetic Algorithms*, San Mateo, CA, 1989. Morgan Kaufman.