

# On Gray-Coded Binary Representation for Supporting a (Repair-by-Interpolation) Genetic Operator for Constrained Optimization Problems

Steven Orla Kimbrough

David Harlan Wood

**Abstract**—A new genetic operator is proposed in the context of Genetic Algorithms that are applied to constrained optimization problems with binary decision variables. A solution is said to be feasible if it satisfies the constraints, and infeasible otherwise. Provided the two inputs of the operator are of differing feasibility, the operator outputs a infeasible/feasible pair that differ by only a single bit. This is valuable because any optimal solution is within a single bit of transitioning between infeasibility and feasibility, unless the constraints are irrelevant. The operator is implemented by binary search along a path connecting the two inputs. The path is a portion of a randomly selected Gray code (an ordered list of all binary strings having the property that adjacent strings differ by a single bit).

## I. INTRODUCTION

When problems depend on yes-or-no (binary) decision variables, Genetic Algorithms (GAs) seem an especially natural approach. In fact, GAs are often applied to optimization problems with binary variables, even the challenging problems that have constraints on the variables. One challenge peculiar to GAs is that the genetic operators—mutation, crossover and others—typically do not respect the constraints. Applying these operators can generate candidate solutions that violate the constraints. (A violating solution is said to be *infeasible*; otherwise a solution is *feasible*.)

This raises the still-vexing question of how to handle infeasible solutions generated by a GA operating on a constrained optimization problem (COP). The problem has long been recognized (see [16], [17] and [19] for early surveys) and remains an open one, on which much progress has nonetheless been made. (See [18, chapter 9] and [5] for recent surveys. See [9], [12], [10], [22] for examples of recent innovative approaches.) This issue—of handling infeasible solutions generated by a GA in processing a COP—naturally raises a related problem: how to explore the boundary between the feasible and infeasible regions of a COP. It is this *boundary exploration problem* that we emphasize in our ongoing research [9], [10], [12], [13].

What makes the boundary exploration salient for any heuristic approach to COPs is the fact that typically, optimal and near-optimal solutions will lie on or near the boundary of the feasible region. Note: The regions do not need to be convex or even connected; the feasible region might look like

the holes in a Swiss cheese. In speaking of “the boundary between the feasible and infeasible regions” we are not assuming that there is only one connected boundary, that it is continuous, etc.

Notable prior work by Schoenauer and Michalewicz [24], [25] has recognized the importance of boundary exploration, and their work has presented techniques by which GAs may explore the boundary of the feasible region, and has achieved excellent results in certain cases. As noted by these authors and in the review by Coello, however, these particular approaches are not fully general [5]:

The main drawback of this approach is that the operators designed are either highly dependent on the chosen parameterization..., or more complex calculations are required to perform crossover and mutation. Also, many problems have disjoint feasible regions and the use of operators of this sort would not be of much help in those cases since they would explore only one of those feasible regions.

In any event it is well worth exploring other approaches by which GAs may probe the boundary in a COP. The so-called feasible-infeasible two-population GA (FI2PopGA) is interesting in this regard [9], [10], [12]. In the FI2PopGA, feasible solutions are processed in the feasible population and infeasible solutions in the infeasible population. The effect of selection is that both populations tend to be driven towards the boundary between the feasible and infeasible regions, the result being a population-based exploration of the boundary, both from the feasible and the infeasible sides.

Exploring the boundary between the feasible and infeasible regions requires first finding solutions that are on or near the boundary. How might these solutions be produced? Can they be produced in a way that itself explores the boundary, so that the solutions are not all concentrated in one spot?

We propose a genetic operator, new to us, which invokes the constraints on the binary variables, but does not depend on any other problem-dependent information. The operator maps one infeasible/feasible pair of solutions into another such pair. Significantly, the two offspring differ only by a single flipped bit, irrespective of the parents. This is important because we are seeking feasible optimal and near-optimal solutions, and they must have this property: changing some single bit makes them infeasible. (Provided that the constraints are relevant, which we assume. In the case of knapsack problems, and many other problems, every optimal solution is one bit away from infeasibility.)

Steven Orla Kimbrough is a Professor of Operations and Information Management, University of Pennsylvania, Philadelphia, PA, 19104, USA (email: kimbrough@wharton.upenn.edu). David Harlan Wood is a Research Professor of Computer and Information Sciences, University of Delaware, Newark, DE 19714 USA (email: wood@cis.udel.edu).

The proposed operator uses an ordered list of all possible bitstrings of length  $n$ . Each parent is a different one of the possible bitstrings, and each occupies a particular position on the list. We are particularly interested in lists of bitstrings with the property that each bitstring differs from its neighbors as little as possible, usually by a single bit. This is the defining property of Gray codes [14], [23]. Our operator starts with two *arbitrary* bitstrings (“the parents”), one feasible and one infeasible. The operator produces a feasible/infeasible *neighboring* pair of bitstring children by doing binary search between the two parents on a list of Gray-coded bitstrings. This binary search uses at most  $n$  feasibility tests, where  $n$  is the length of the bitstrings. Of course, one may restrict search to only the  $k$  bits that differ in the two parents. This restriction to differing bits is used in alternative forms of crossover that emphasize exploitation, e.g., [26].

The next section presents basic information on constrained optimization. After that we provide a more detailed construction of the proposed operator. We then briefly interpret the proposed operator to be an analog of real-variable interpolation. This is followed by two sections suggesting possible applications of the proposed operator to GAs. Particular attention is focused on aspects related to exploration and exploitation. The overall emphasis is that there are intriguing possibilities, but little is yet known about this proposed operator, and much additional research is called for.

## II. CONSTRAINED OPTIMIZATION

By way of background, optimization problems may usefully be distinguished as either constrained or unconstrained. Our focus in this paper is on constrained optimization problems,<sup>1</sup> which have the following general form:

$$\max z = d(\vec{x}) \quad (1)$$

subject to

$$f_i(\vec{x}) \leq a_i, \quad i = 1, 2, \dots, n_f \quad (2)$$

$$g_j(\vec{x}) \geq b_j, \quad j = 1, 2, \dots, n_g \quad (3)$$

$$h_k(\vec{x}) = c_k, \quad k = 1, 2, \dots, n_h \quad (4)$$

$$x_l \in S, \quad l = 1, 2, \dots, n_l. \quad (5)$$

$d(\vec{x})$  in expression (1) is called the *objective function* for the problem. Its value,  $z$ , is what we seek to maximize (or minimize) by finding values of, or settings for, the *decision variables*, the  $x_l$ s, that yield the highest (or lowest if minimizing) value for  $z$  among the settings that satisfy the *constraints*, namely the expressions (2)–(5). Such a setting of values for the decision variables is said to be optimal.

Any particular choice of the values for the decision variables is called a *solution* to the problem, regardless of whether it is optimal or whether it satisfies the constraints. A solution that satisfies all of the constraints is said to be *feasible*, otherwise it is *infeasible*. Optimal solutions must be feasible, but need not be unique.

<sup>1</sup>The distinction is perhaps not absolute, since there are cases in which constraints may be eliminated by an alternative encoding of the problem.

The constraints, as we have just noted, serve to classify solutions as either feasible or infeasible. The *right-hand side* (RHS) values of the inequality constraints, the  $a_i$ s,  $b_k$ s and the  $c_j$ s, are said to define *boundaries* between the feasible and infeasible regions for the problem. A given solution,  $\vec{x}$ , is said to be *near to the boundary* (for a particular constraint) if the *left-hand side* of the constraint is close (pragmatically defined for the problem at hand) to the right-hand side. The solution is said to be *on* the boundary if the left-hand side equals the right-hand side. More generally, we say that a solution is on or near the boundary of the feasible region if it is on or near the boundary of at least one constraint. In typical constrained optimization problems encountered in practice, the optimal solutions, as well as the good (near optimal) solutions, are on or near the boundary.

## III. IMPLEMENTATION OF THE PROPOSED OPERATOR

**Definitions.** By *numerical binary order* for the set of bitstrings of fixed length, we mean ordering them according to their numerical interpretation coded in the standard, base 2 binary fashion. By *Gray code order* for the set of bitstrings of fixed length, we mean an order given by a particular Gray code. We use Gray codes with the property that adjacent bitstrings differ by a single bit. (Knuth sometimes allows differences of more than one bit [14].)

There are a huge, and unknown, number of Gray codes. The Gray code usually used in evolutionary computation is the standard binary reflected Gray code [14]. However, to avoid biases that depend on the location of a variable within in a bitstring, the proposed operator would need to use so-called *balanced* Gray codes [2]. In a balanced Gray code, the number of bit changes is maximally uniform among the bit positions. Other types of Gray codes [14] with additional or alternative properties might also be useful.

When the proposed operator calls for a randomly selected uniform Gray code, one option is to merely offset the indexing of one particular Gray code by a pseudorandomly generated bitstring, as is done for an unbalanced Gray code in [23].

We use these key functions:

- 1) *GrayToBin* returns a binary representation of the index of a bitstring in a Gray code.
- 2) *BinToGray* returns the bitstring in a Gray code, given a binary binary representation of its index.
- 3) *binAverage* returns a natural binary representation that is the average to two given binary representations

In outline the procedure is as follows. We work throughout assuming two populations of solutions: a population of feasible solutions and a population of infeasible solutions. A solution, in our discussions, is just a string of binary digits; its interpretation is set by the GA’s fitness evaluation function.

- 1) Pick a feasible solution and assign it to the variable *feasible*.
- 2) Pick an infeasible solution and assign it to the variable *infeasible*.
- 3)  $\text{binFeasible} \leftarrow \text{grayToBin}(\text{feasible})$

- 4)  $\text{binInfeasible} \leftarrow \text{grayToBin}(\text{infeasible})$
- 5)  $\text{binTest} \leftarrow \text{binAverage}(\text{binFeasible}, \text{binInfeasible})$
- 6) If  $\text{binTest} = \text{binFeasible}$  or  $\text{binTest} = \text{binInfeasible}$ , halt and accept feasible and infeasible as adjacent; otherwise continue.
- 7)  $\text{grayTest} \leftarrow \text{binToGray}(\text{binTest})$
- 8) Determine whether  $\text{grayTest}$  is feasible.
- 9) If  $\text{grayTest}$  is feasible, set:  $\text{feasible} \leftarrow \text{grayTest}$   
If  $\text{grayTest}$  is infeasible, set:  $\text{infeasible} \leftarrow \text{grayTest}$
- 10) Continue at step 3.

#### IV. THE PROPOSED OPERATOR INTERPOLATES TO THE INFEASIBLE/FEASIBLE BOUNDARY

On one hand the proposed operator is a genetic operator in the sense that two parent bitstrings produce two offspring bitstrings that compromise between the bitstrings of the parents. On the other hand, we can consider the operator to “randomly interpolate to the infeasible/feasible boundary.” We now discuss the analogy between the proposed operator and the operator of interpolation.

If we wanted to find the infeasible/feasible boundary in a problem where the variables were real (not binary), we would numerically interpolate variables. That is, we would do a binary search along a straight line connecting the endpoints defined by the two parents. For binary variables, we replace selecting a straight line with selecting with an ordered list of discrete bitstrings, usually with a segment of a Gray-coded list. The analogy wavers because there are many Gray codes for discrete variables, but only one linear segment defined by the parents in the real variable case. The analogy recovers if it is broadened. In the case of continuous variables, one could search for the infeasible/feasible boundary along *arbitrary* continuous nonlinear curves linking the parents. In fact, this is likely to be a useful exploration producing essentially random samples on the boundary. Randomly selected Gray codes do this in the case of binary variables.

*Remark 1:* Interpolation between two bit strings, one feasible and one infeasible, results in a feasible/infeasible pair of bitstrings whose indices are adjacent. We observe that if the bitstrings are indexed using a Gray code where neighbors differ by one bit, then the resulting feasible/infeasible pair of bitstrings also differs by only one bit.

*Remark 2:* The interpolation between an original feasible solution and an original infeasible solution using any ordered list of bitstrings is guaranteed to produce a final feasible/infeasible pair. Either one of the final pair of solutions may, of course, happen to be identical with one of the original solutions. (For example, some problems have only one feasible solution.) The paramount fact is: given a feasible bitstring and an infeasible bitstring *any* bitstring between them will either be feasible or infeasible, which determines a *closer* feasible/infeasible pair.

*Remark 3:* The interpolation process produces a feasible/infeasible pair, regardless of the shape of the feasible region. In particular, the feasible region need not be convex or even connected; it may contain infeasible regions and indeed be completely arbitrary. Additionally, the constrained optimization problem need not be linear; interpolation allows any functions whatsoever to characterize the objective and the constraints.

*Remark 4:* Our method applies to constrained optimization problems having binary decision variables. This class is large, interesting, challenging, and significant for applications. Even the restricted class of problems with only binary variables is itself large, interesting, challenging, and significant for applications. Among these are knapsack problems, such as multidimensional knapsack problems, multiple knapsack problems, multiple-choice knapsack problems, and quadratic variants of all of these types [15], [4], [8], [7]. The class also includes generalized assignment problems [3], [27], [28]. Relevant problem sets can be found at [1], [6], [21].

#### V. CONTRASTING THE OPERATOR TO MORE USUAL ONES

The proposed operator can be used in conjunction with more usual genetic operators. However, is it distinguished from the following operators in the ways listed below.

- Repair. We have a general method for repairing infeasible solutions. This method does not require additional problem-dependent information. The proposed operator uses only constraint information, which is given in the definition of the problem.
- Mutation. Randomly selecting a new Gray code for each application of the proposed operator is a type of mutation. This generally affects multiple bits, which is more change than is usual for conventional mutation. Yet, unlike conventional mutation, the results are guaranteed to be within one bit of feasibility.
- Crossover. The proposed operator uses not just limited, undirected random choice(s) of bitstring portions to be exchanged, but rather makes multiple decisions based on constraint-directed binary search. Again, the offspring are an infeasible/feasible pair differing by a single bit.

#### VI. PRODUCTION OF FEASIBLE SOLUTIONS

In many constrained optimization problems it is difficult to find any feasible solutions at all. The proposed operator may be particularly helpful in this regard. One can interpolate between any infeasible/feasible pair. This process yields an intermediate pair of infeasible/feasible offspring — in particular, another feasible solution (although it might happen to be identical with the original feasible solution). Each repeated interpolation of a given pair generally results in different offspring. This is true even if the parents are themselves already on the boundary. The non-unique results of interpolation are due to using a different randomly selected Gray code each time. Although non-unique results are the rule, the exception is that the feasible offspring can

be identical to the feasible parent. In particular, the problem might happen to have only one feasible solution.

## VII. EXPLORING THE INFEASIBLE/FEASIBLE BOUNDARY

- 1) GAs typically produce a large number of samples from the decision space of a constrained optimization problem. We observe that using the proposed operator generates samples of solution space near both sides of the infeasible/feasible boundary. These samples may be valuable for post-evaluation questions concerning relaxation and tightening of constraints [13]. For example, in the relaxation problem we are interested in infeasible solutions near the present boundary, some of which might become feasible were the boundary constraints relaxed.
- 2) Cross breeding among two pairs of siblings would be one technique of exploring *along* the infeasible/feasible boundary. Even incest would *not* be cloning, since a different Gray code would be used for each application of the proposed operator.
- 3) A Gray code is a *circular* list of bitstrings of a particular length. We could take the shorter sublist starting with one parent and ending with the other. Even if parents are close to each other in the Gray code order, the proposed operator has an aspect of exploration. Since a Gray code is a circular list, exploration could be enhanced by using the *other*, longer of the two list segments connecting the two parents.
- 4) Given a feasible solution, its complement, generated by flipping all of its bits, is far away (at the maximum possible Hamming distance), and is very likely to be infeasible. Applying the proposed operator to two such solutions would be likely to be explorative.
- 5) Repeatedly applying the proposed operator to any parent and to one feasible parent (however trivial, e.g. all zeros) might be used for initializing populations, since we wish to explore near the infeasible/feasible boundary. The results would be distinct because each application of the proposed operator uses a different Gray code.

## VIII. CROSSBREEDING INFEASIBLE AND FEASIBLE POPULATIONS

We are intrigued by prior research that has advocated exploring the boundary region by evolving two populations, one of infeasible solutions and one of feasible solutions (FI2PopGA, cited above, is a special case). Selection for breeding within the feasible population involves, as usual, fitness being equated with good objective function values. Selection for breeding within the infeasible population involves fitness being equated with low constraint violations. The key idea is that breeding two individuals in a population can result in (genetically similar) individuals migrating to the other population. Ultimately, we have two genetically similar populations, one minimizing constraint violations and the other maximizing objective functions values. This genetic

similarity of the two populations is our notion of convergence to one or more optima.

Rather than leaving migration between the two populations to chance, interpolation can serve as a form of crossbreeding. A selected infeasible/feasible pair of parents gives rise to an infeasible/feasible pair of offspring as elaborated earlier. On one hand, the infeasible individual of this pair could be allowed to replace its infeasible parent because it has minimal constraint violation. Minimal in the sense that flipping a single bit would convert it into a feasible individual (its sibling). On the other hand, the feasible sibling could be used replace its feasible parent if it has a sufficiently high objective function value. Many variations are possible and need to be investigated, for example:

- 1) Select the feasible parents because their objective function values are promising. And/or select the infeasible parents by a small count (or other measure) of constraint violations. Or select parents by their genetic similarity (small Hamming distance).
- 2) The proposed operator could be applied to a restricted subset of bits, for example to only those positions that differ in the two parents as in [26]. This could be exploitive if used to explore the the immediate neighborhood of a solution having a promising objective function value.

The proposed operator is superficially similar to optima linking [20], which address a more difficult matter. Optima linking, roughly speaking, searches a Gray code type path between two relative optima looking for new relative optima. This task is not amenable to binary search since (1) the path is generated by a greedy search using repeated fitness evaluations, and (2) objective function values are continuous, which is very unlike feasibility which is true or false.

## IX. COMPUTATIONAL EXPLORATIONS

In this section we report example computational results on a simple type of problem. We do this mainly for purposes of illustration and to raise issues for future research.

### A. An Illustrative Problem

In expanding upon the observations above we will, for the sake of illustration, provide computational results for a single constraint knapsack problem. This will help to demonstrate concepts clearly. More extensive evaluation of concepts and algorithms must await future investigation.

The following class of problems is used for our illustrations. Knapsack problems with a single constraint are a special case of constrained optimization problems. In words, the problem is to select various objects that will fit into a given ‘knapsack’ so as to maximize their total value, subject to a constraint on their total ‘weight.’ The problem has the following form:  $\max z = \sum_{i=0}^n p_i x_i$  subject to the constraint  $\sum_{i=0}^n w_i x_i \leq c$  by selecting  $x_i \in \{0, 1\}$ ,  $i = 0, 1, 2, \dots, n$ . In several of our examples, we use Knap101. It has 50 decision variables; specifics are available from the authors. We also use what we call the KnapRandomX families of

problems. These are randomly generated knapsack problems of the above form, having  $X$  decision variables. Our method for generating random knapsack problems follows, but generalizes, the standard method given in [15, page 67]. For all of the experimental results reported here, we used the system clock to initialize the random number generator.

Throughout, we use our version of the FI2PopGA. Unless otherwise noted, the feasible and infeasible populations were limited to 150 solutions each. The mutation rate was 0.025 per bit and the (single point) crossover rate was 0.4. We use tournament selection within a population to obtain parents. For both populations we use elite-2 selection: each new generation contains the best two solutions from the previous generation. For initialization, we use the “unbiased” method described in [11]: at initialization of each solution a new random float  $\sim U(0, 1)$  is drawn to set the probability any given bit in the solution will be set to 1.

### B. First Experiments: Repairing Initially Infeasible Solutions

In each of these experiments we randomly generated 300 solutions and grouped them by feasibility. We then repaired each infeasible solution. A random *feasible* solution was chosen and the feasible-infeasible pair was interpolated to the boundary. The resulting feasible interpolant (= interpolated solution) was placed in the *feasible interpolant* pool of solutions for comparison with the pool of *originally feasible* solutions. In consequence we obtained two pools of feasible solutions. Each experiment consisted of a number of runs and for each run four statistics were kept: (1) AvgOrig, the average objective value of the originally feasible solutions, (2) MaxOrig, the maximum objective value of the originally feasible solutions, (3) AvgIntp, the average objective value of the feasible interpolants, and (4) MaxIntp, the maximum value of the feasible interpolants. As summary statistics we report for each experiment: (1) AvgAvgOrig, the average of the averages of the objective values of the originally feasible solutions, (2) AvgMaxOrig, the average of the maxima of the objective values of the originally feasible solutions, (3) MaxMaxOrig, the maximum of the maxima of the objective values of the originally feasible solutions, (4) AvgAvgIntp, the average of the average objective values of the feasible interpolants, (5) AvgMaxIntp, the average of the maximum values of the feasible interpolants, and (6) MaxMaxIntp, the maximum of the maximum values of the feasible interpolants.

In experiment 1 we performed 1000 runs on Knap101.<sup>2</sup> We used 100 runs in experiments 2–5. In experiments 1–3 the number of initially feasible solutions was approximately equal to the number of infeasible solutions. In experiment 4 we created problems with a smaller feasible region, so that only about 1 in 6 randomly-created solutions will be feasible. And in experiment 5 we increased of the feasible region so that about 5 of 6 randomly-created solutions being feasible. The summary results in Figure 1 display a clear, and we think remarkable, pattern: AvgAvgIntp > AvgAvgOrig,

Interpolating infeasible solutions with randomly-chosen feasible solutions:

- 1) Knap101, 1000 runs. Initially feasible:infeasible  $\approx$  1:1 (e.g., 150, 150), but with a slight bias towards having more infeasible than feasible solutions.
- 2) KnapRandom100, 100 runs, vvv = 50, rrr = 36, rhsDeflator = 0.18. Initially feasible:infeasible  $\approx$  1:1 (e.g., 150, 150).
- 3) KnapRandom400, 100 runs, vvv = 50, rrr = 36, rhsDeflator = 0.18. Initially feasible:infeasible  $\approx$  1:1 (e.g., 150, 150).
- 4) KnapRandom100, 100 runs, vvv = 50, rrr = 36, rhsDeflator = 0.14. Initially feasible:infeasible  $\approx$  1:5 (e.g., 50, 250).
- 5) KnapRandom100, 100 runs, vvv = 50, rrr = 36, rhsDeflator = 0.22. Initially feasible:infeasible  $\approx$  5:1 (e.g., 250, 50).

Interpolating feasible solutions with randomly-chosen infeasible solutions:

- 6) Knap101, 100 runs. Initially feasible:infeasible  $\approx$  1:1 (e.g., 150, 150), but with a slight bias towards having more infeasible than feasible solutions.
- 7) KnapRandom100, 100 runs, vvv = 50, rrr = 36, rhsDeflator = 0.18. Initially feasible:infeasible  $\approx$  1:1 (e.g., 150, 150).

Interpolating feasible solutions with ideal solution of all 1s:

- 8) Knap101, 100 runs. Initially feasible:infeasible  $\approx$  1:1 (e.g., 150, 150), but with a slight bias towards having more infeasible than feasible solutions.

Interpolating infeasible solutions with ‘ideal’ solution of all 0s:

- 9) Knap101, 100 runs. Initially feasible:infeasible  $\approx$  1:1 (e.g., 150, 150), but with a slight bias towards having more infeasible than feasible solutions.

\* \* \*

- 10) KnapRandom200, 4 runs. Initially feasible:infeasible  $\approx$  1:3 (e.g., 50, 200). vvv = 100, rrr = 68, and rhsDeflator = 0.18.
- 11) Knap101, 4 runs, 400 generations.
- 12) KnapRandom200, 4 runs. vvv=100, rrr=68, rhsDeflator = 0.08. maxFeasiblePop = maxInfeasiblePop = 100. Number of generations is 200. Initially feasible:infeasible  $\approx$  1:7 (e.g., 25:175 or less).

Fig. 4. List of Experiments

<sup>2</sup>All experiment numbers refer to items in Figure 4.

Experiment	AvgAvgOrig	AvgMaxOrig	MaxMaxOrig	AvgAvgIntp	AvgMaxIntp	MaxMaxIntp
1	330.4594482	635.69538	901.936	457.8797974	771.946516	960.744
2	368.5132581	612.4411306	732.1630616	439.3148596	649.9477374	808.3502723
3	1672.045458	2174.013643	2427.176228	1833.404816	2253.418637	2613.169806
4	303.3196018	486.0084794	616.8792745	344.485491	545.2255911	685.2194071
5	422.2505364	728.8195957	842.2539274	542.6197975	734.9440063	909.9526299

Fig. 1. Summary of results for repairing initially infeasible solutions by interpolating against randomly-chosen feasible solutions

Experiment	AvgAvgOrig	AvgMaxOrig	MaxMaxOrig	AvgAvgIntp	AvgMaxIntp	MaxMaxIntp
6	330.4371184	635.40656	830.804	456.7755909	758.98987	879.141
7	371.5201013	623.7840595	779.4256039	445.9818806	658.9039887	876.0144186

Fig. 2. Summary of results for interpolating initially feasible solutions against randomly-chosen infeasible solutions

Experiment	AvgAvgOrig	AvgMaxOrig	MaxMaxOrig	AvgAvgIntp	AvgMaxIntp	MaxMaxIntp
8	330.9452238	633.325	768.989	438.8640487	741.42532	862.331
9	329.1461737	629.13159	760.592	486.169911	781.27642	927.559

Fig. 3. Summary of results for interpolating to ideal solutions. 8: feasible solutions interpolated to all 1s solution, Knap101. 9: infeasible solutions interpolated to all 0s solution, Knap101. \*Orig refers to the initial feasible population.

$\text{AvgMaxIntp} > \text{AvgMaxOrig}$ , and  $\text{MaxMaxIntp} > \text{MaxMaxOrig}$ . In short, throughout the summary statistics  $\text{Intp} > \text{Orig}$ ; on balance the solutions obtained by interpolation from the infeasible solutions have higher objective function values (higher fitnesses in the GA), than the feasible solutions that were randomly created.

We note that while the feasible solutions discovered in this process are often very good, they are far from optimal. For example, the optimal objective function value for the problem in experiment 1 is 1119.984. A good lower bound estimate (using the bang-per-buck heuristic) for the problem in experiment 3 is 4373.; for experiment 5 it's 1334.7.

### C. Second Experiments: Improving Feasible Solutions

Figure 2 summarizes the results of our second group of experiments, experiments 6 and 7. In these experiments we interpolate initially feasible solutions and retain the resulting feasible interpolants. Again we have the pattern:  $\text{AvgAvgIntp} > \text{AvgAvgOrig}$ ,  $\text{AvgMaxIntp} > \text{AvgMaxOrig}$ , and  $\text{MaxMaxIntp} > \text{MaxMaxOrig}$ . In short, throughout the summary statistics in Figure 2  $\text{Intp} > \text{Orig}$ ; on balance the solutions obtained by interpolation from the initial feasible solutions have higher objective function values (higher fitnesses in the GA), than the feasible solutions that were randomly created.

### D. Third Experiments: Interpolating Against Ideals

Assuming its parameters  $p_i$ s and  $w_i$ s are  $> 0$  and that it is non-degenerate, the single-constraint knapsack has the useful properties that (a) the solution consisting of all 0s is feasible and the worst possible solution in terms of the objective function, and (b) the solution consisting of all 1s is infeasible and the best possible solution in terms of the objective function. We shall call the all 1s solution the *ideal optimal* solution and the all 0s solution the *ideal pessimal* solution. Figure 3 presents summary results from interpolating initially feasible solutions with the ideal optimal solution (experiment

8) and the initially infeasible solutions with the ideally pessimal solution (figure 9). Again we have the pattern:  $\text{AvgAvgIntp} > \text{AvgAvgOrig}$ ,  $\text{AvgMaxIntp} > \text{AvgMaxOrig}$ , and  $\text{MaxMaxIntp} > \text{MaxMaxOrig}$ . Note further that on balance the interpolants produced from originally *infeasible* solutions have higher objective function values than those produced from feasible solutions.

Experiment 10, see Figure 4, consisted of four runs of larger randomly picked knapsack models. The basic pattern we have observed showed up again: feasible interpolants, of whatever origin, score better than randomly-generated feasible solutions.

### E. Fourth Experiments: After Evolution

In experiment 11 we ran the Knap101 model for 400 generations, which is usually more than sufficient for it to find the optimal solution. We then interpolated the feasible population against random members of the infeasible population and against the ideal optimal solution. The interpolants were uniformly worse. In fact, the best (in terms of objective function value) of the interpolants was always in the lower 20% of the feasible population produced by the GA (so long as the interpolant was different than the original feasible solution). The story is the same on the infeasible side. Starting with infeasible solutions from generation 400, interpolating them to randomly chosen feasible solutions from the same generation or to the ideal pessimal solution produces similarly weak feasible interpolants. Optimal solutions cannot, of course, be improved. In knapsack problems optimal solutions will always be on the boundary: the constraint will be violated if anything is added to the knapsack. It is interesting that optimal and near-optimal solutions get interpolated away to suboptimal boundary points, indicating that there are many feasible boundary points.

Experiment 12 examined a larger knapsack (200 decision variables) with a tighter constraint. Only about 1 in 8 or

9 randomly generated solutions were feasible. After 200 generations, the sizes of the feasible populations ranged from 5 to 11. Again, the best feasible solutions were not beaten by interpolating feasible solutions with either randomly chosen solutions from the infeasible of that generation or the ideal optimal solution, but the interpolants did score well and did not duplicate the initially feasible solutions. For example, and typically, in the fourth run we have the results shown in Figure 5. We note that none of the runs in experiment 12 found an optimal solution to its problem. The best solutions found had objective values there were 85–90% of the optimal value.

Most interestingly, in this run (which we find is quite typical) the infeasible population numbered the maximal 100. When these 100 infeasible solutions are interpolated to randomly chosen feasible solutions among the 8 in the feasible population, here are the objective function values of the 100 resulting feasible solutions:

1936.003116			
2072.49661	2116.19405	2036.835572	2125.946244
1385.121247	2063.987422	1409.095454	2116.19405
2130.967787	2116.19405	2263.191409	2035.127962
1573.354134	2064.883457	1939.961534	1936.003116
1975.466384	1400.818535	1960.111642	1833.648248
2263.191409	2023.263264	1850.374017	2061.273095
1936.003116	2315.398036	2108.936309	2116.19405
1774.67235	1936.003116	1969.476822	1942.559182
1999.221382	2263.191409	1994.40392	1786.891265
1928.474713	2072.336159	1936.003116	2118.522289
953.7401375	2037.470537	2017.356767	1975.466384
1936.003116	1936.003116	2116.19405	2130.967787
2269.152024	2006.120408	1991.881893	1832.511014
1861.316226	1975.466384	2263.191409	2108.245197
2180.384362	2041.786931	2263.191409	1820.017177
2315.398036	1607.692244	2011.830358	2263.191409
1963.895414	2011.830358	1959.444116	2134.322301
1958.004054	2315.398036	2269.152024	2320.220241
2157.942916	2263.191409	2064.793809	2075.469891
2131.277531	2045.906973	2116.19405	2315.398036
1862.996277	1963.469811	2116.19405	2188.357803
1988.134225	2012.004854	2263.191409	1950.725671
2180.384362	2011.830358	2134.322301	2263.191409
2263.191409	2263.191409	2315.398036	1991.881893
2006.120408	2073.168302	2263.191409	

Notice that there are very few duplications and there are quite a few that beat the weakest members of the feasible population produced by the GA. (Compare with the left-most column of Figure 5.)

## X. DISCUSSION AND CONCLUSION

We have proposed and made preliminary investigation of an interpolation algorithm for GAs with binary variables. This algorithm has the attractive property that if initiated with (feasible, infeasible) pairs, it returns a feasible/infeasible pair such that the two solutions differ by one bit and hence are on or one bit away from the boundary between the feasible and infeasible regions of the problem. The algorithm has attractive computational properties in that at most  $n$

feasibility tests are needed to evaluate intermediate interpolants for obtaining a feasible/infeasible pair. Further, we have reported exploratory computations indicating that this interpolation algorithm can find good new and even better solutions from an initial population, randomly generated. Further, the algorithm does this while interpolating from either feasible or infeasible. If the GA is run to maturity, our preliminary computations indicate that interpolation is not particularly productive for these examples. However, for more difficult problems interpolation may be useful even after considerable effort by the GA. All this is, of course, entirely exploratory and quite provisional, although it does agree with findings we do not report here.

It remains to be seen whether, or better, under what conditions, investing in finding interpolants yields a larger return than other available alternatives, such as just running the GA longer or with a larger population. How that will come out will, of course, very much depend on the specifics of the problem. Our sense is that single-constraint knapsacks are not particularly good candidates, if only because good solutions are found comparatively easily. (For the record, we have tried some fairly obvious ways, simpler than those indicated in §8, of incorporating interpolation into the GA for these knapsack problems and cannot report being able to make it pay.)

We conclude with two comments bearing on the ultimate usefulness of interpolating feasible/infeasible pairs. First, it is often the case in practice that finding any feasible solution or more than just a few is very difficult. Interpolation offers the prospect, as we have seen realized, of finding very many new feasible solutions, even for a reasonably mature run of a GA. We illustrated this at the end of the previous section. Second, our interpolants are all on or next to (“at”) the boundary of the feasible region. Having so many solutions at the boundary may constitute excess exploitation. Moving feasible interpolants away from the boundary, towards the interior (say by moving them away from their infeasible siblings) is an exploratory move that merits investigation.

In conclusion, we can see several reasons why, in principle, it should be interesting and useful to undertake interpolation to find feasible/infeasible pairs on the boundary of the feasible region of a constrained optimization problem.

- 1) In very many constrained optimization problems of practical import simply finding any feasible solutions is a major impediment (see [9] for a case study). Interpolation offers the prospect of reliably generating many new feasible solutions, given just one feasible solution. Our computational experience to date is very encouraging in this regard.
- 2) Probing the boundary of the feasible region is generally recognized as an important tool in constrained optimization. Reliably producing feasible/infeasible pairs differing by one bit is, we believe, a most promising development in this regard.
- 3) Interpolation may produce, or may produce more quickly, new feasible solutions that are comparatively

Member of feasible population		Feasible interpolant	
Objective value	LHS value	Objective Value	LHS value
2346.797719	770.1437964	2315.398036	774.5137867
2346.797719	770.1437964	2320.220241	773.3859092
2147.593732	760.9960162	1928.884528	743.9253863
2077.306655	730.510768	2108.245197	774.9858825
2064.883457	710.9321947	2157.942916	776.3604129
2031.042331	713.9834285	2075.469891	746.8758016
2019.609426	755.7276373	1936.003116	739.8352388
2005.576649	659.5797118	2007.721724	775.9250049

Fig. 5. Example from experiment 12. The feasible population after 200 generations and the feasible results of interpolating the feasible solutions to randomly chosen solutions from the infeasible population.

attractive. Our experience to date with single constraint knapsack problems is not encouraging in this regard, but this is hardly dispositive. More challenging problems need to be investigated as do supplemental heuristics, such as employing local search to move solutions off the boundary slightly, where they may be evolutionarily blocked by a jagged frontier. Interpolation may be usefully preparatory to local search.

#### REFERENCES

- [1] J. E. Beasley. OR-Library. World Wide Web, Accessed 2007-03-30. <http://people.brunel.ac.uk/~mastjjb/jeb/info.html>.
- [2] G. S. Bhat and C. D. Savage. Balanced Gray codes. *Electronic Journal of Combinatorics*, 3(1):R25, 1996.
- [3] P. C. Chu and J. E. Beasley. A genetic algorithm for the generalized assignment problem. *Computers and Operations Research*, 24(1):17–23, 1997.
- [4] P. C. Chu and J. E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4(1):63–86, June 1998.
- [5] C. Coello. Theoretical and numerical constraint handling techniques used with evolutionary algorithms: A survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, 191(11–12):1245–1287, January 2002. [citeseer.ist.psu.edu/coello02theoretical.html](http://citeseer.ist.psu.edu/coello02theoretical.html).
- [6] GAMS World. Global world. Pages on the World Wide Web, <http://www.gamsworld.eu/>, accessed 2007-03-30.
- [7] A. Hiley and B. A. Julstrom. The quadratic multiple knapsack problem and three heuristic approaches to it. In *Proceedings of the Genetic and Evolutionary Computing Conference (GECCO '06)*, pages 547–552, Seattle, Washington, USA, July 8–12, 2006. Available in the ACM digital library.
- [8] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, Berlin, Germany, 2004.
- [9] S. O. Kimbrough, M. Lu, and S. M. Safavi. Exploring a financial product model with a two-population genetic algorithm. In *Proceedings of the 2004 Congress on Evolutionary Computation*, pages 855–862, Piscataway, NJ, June 19–23, 2004. IEEE Neural Network Society, IEEE Service Center. ISBN: 0-7803-8515-2.
- [10] S. O. Kimbrough, M. Lu, and D. H. Wood. Exploring the evolutionary details of a feasible-infeasible two-population GA. In X. Yao et al., editors, *Parallel Problem Solving from Nature – PPSN VIII*, volume 3242 of *LNCS: Lecture Notes in Computer Science*, pages 292–301. Springer-Verlag, Berlin, Germany, 18–22 September 2004.
- [11] S. O. Kimbrough, M. Lu, D. H. Wood, and D. J. Wu. Exploring a two-market genetic algorithm. In W. B. Langdon, E. Cantú-Paz, and et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002)*, pages 415–21, San Francisco, CA, 2002. Morgan Kaufmann.
- [12] S. O. Kimbrough, M. Lu, D. H. Wood, and D. J. Wu. Exploring a two-population genetic algorithm. In E. Cantú-Paz et al., editors, *Genetic and Evolutionary Computation (GECCO 2003)*, LNCS 2723, pages 1148–1159, Berlin, Germany, 2003. Springer.
- [13] S. O. Kimbrough and D. H. Wood. On how solution populations can guide revision of model parameters. Late breaking papers, GECCO 2006, July 2006. [lbp133.pdf](http://lbp133.pdf) at GECCO 2006. Available at <http://opim-sky.wharton.upenn.edu/~sok/sokpapers/2007/lbp133.pdf>.
- [14] D. E. Knuth. *The Art of Computer Programming*, volume 4, fascicle 2. Addison-Wesley, Upper Saddle River, NJ, 2005.
- [15] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, New York, NY, 1990.
- [16] Z. Michalewicz. Do not kill unfeasible individuals. In Dabrowski, Michalewicz, and Ras, editors, *Proceedings of the Fourth Intelligent Information Systems Workshop (IIS'95)*, pages 110–123, Augustow, Poland, 5–9 June 1995.
- [17] Z. Michalewicz. A survey of constraint handling techniques in evolutionary computation methods. In J. R. McDonnell, R. G. Reynolds, and D. B. Fogel, editors, *Proc. of the 4th Annual Conf. on Evolutionary Programming*, pages 135–155, Cambridge, MA, 1995. MIT Press.
- [18] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer, Berlin, Germany, 2000.
- [19] Z. Michalewicz and M. Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32, 1996.
- [20] S. B. Rana and L. D. Whitley. Bit representations with a twist. In T. Bäck, editor, *ICGA*, pages 188–195. Morgan Kaufmann, 1997.
- [21] C. A. I. Repository. SAC94 Suite: Collection of multiple knapsack problems. World Wide Web, Accessed 2007-03-30. <http://www-2.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/genetic/ga/test/sac/>.
- [22] R. L. Riche and F. Guyon. Dual evolutionary optimization. In *Artificial Evolution*, pages 281–294, 2001.
- [23] J. Rowe, D. Whitley, L. Barbulescu, and J.-P. Watson. Properties of Gray and binary representations. *Evol. Comput.*, 12(1):47–76, 2004.
- [24] M. Schoenauer and Z. Michalewicz. Evolutionary computation at the edge of feasibility. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Proceedings of the 4th Parallel Problem Solving from Nature, Lecture Notes in Computer Science, Vol. 1141*, pages 245–254, Berlin, Germany, 22–27 September 1996. Springer-Verlag.
- [25] M. Schoenauer and Z. Michalewicz. Sphere operators and their applicability for constrained parameter optimization problems. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Evolutionary Programming VII: Proceedings of the Seventh Annual Conference on Evolutionary Programming, Lecture Notes in Computer Science, Vol. 1447*, pages 241–250, San Diego, CA, March 1998. Springer-Verlag.
- [26] J.-P. Watson, C. Ross, V. Eisele, J. Denton, J. Bins, C. Guerra, L. D. Whitley, and A. E. Howe. The traveling salesrep problem, edge assembly crossover, and 2-opt. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *PPSN*, volume 1498 of *Lecture Notes in Computer Science*, pages 823–834. Springer, 1998.
- [27] J. M. Wilson. A genetic algorithm for the generalised assignment problem. *Journal of the Operational Research Society*, 48(8):804–809, August 1997.
- [28] M. Yagiura, T. Ibaraki, and F. Glover. An ejection chain approach for the generalized assignment problem. *Inform. Journal on Computing*, 16(2):133–151, 2004.