Point Representation for Local Optimization

Towards Multi-Dimensional Gray Codes

Shumeet Baluja
Google Research
Google, Inc.
Mountain View, CA. USA

Michele Covell
Google Research
Google, Inc.
Mountain View, CA. USA

***Abstract*— In the context of stochastic search, once regions of high performance are found, having the property that small changes in the candidate solution correspond to searching nearby neighborhoods provides the ability to perform effective local optimization. To achieve this, Gray Codes are often employed for encoding ordinal points or discretized real numbers. In this paper, we present a method to label similar and/or close points within arbitrary graphs with small Hamming distances. The resultant point labels can be viewed as an approximate high-dimensional variant of Gray Codes. The labeling procedure is useful for any task in which the solution requires the search algorithm to select a small subset of items out of many. A large number of empirical results using these encodings with a combination of genetic algorithms and hill-climbing are presented.**

Keywords—Gray Code; Graph Labeling; Local Search; Genetic Algorithms; Stochastic Search

I. INTRODUCTION

Many optimization problems include the task of picking a small subset of items out of many – for example, picking a set of nodes from a graph that form a vertex cover or selecting a set of physical items that meet a set of constraints (such as total size) while maximizing another objective (such as value). Generate-test-and-revise approaches such as evolutionary algorithms, genetic algorithms (GA), simulated annealing and hill-climbing type heuristics are often used to address these problems. Though a variety of encodings can be used to represent candidate solutions, one of the most common is a fixed-length binary string.

In typical select-p-of-N points tasks, the simplest solution is to assign a random, and unique, binary string (of length $\lceil \log_2 N \rceil$) to each of the N points. The search algorithm then encodes the full solution in a binary string of length $p * \lceil \log_2 N \rceil$. Each substring is mapped to a point to be selected (in graph problems, each point may encode a vertex; in knapsack problems, each point may encode a physical item, etc.). Typical search algorithms progress by repeatedly stochastically modifying the candidate solutions and testing the result. A severe drawback of randomly assigning labels to vertices/items is that a small change in a single bit may radically change the solution. A single bit flip may change the resultant substring to encode a point/item that is unrelated to the point prior to the flip. Conversely, another problem is that making small moves (to nearby vertices or to similar objects) may require a large number of changes to the candidate solution string, thereby making it difficult to search local neighborhoods once a region of high performance is found.

The two primary objectives of this paper are to (1) present a method for deriving neighborhood preserving binary labels to effectively represent "close" points and (2) to demonstrate how it can be effectively used within genetic algorithms [25]. By preserving neighborhoods, making local moves in the search space is possible. A wide class of problems, including selecting nodes in graphs (Fig. 1), are amenable to these labeling techniques.

In this past, neighborhood-preserving codes have largely been explored in the context of optimization of an objective function in which the parameters are discretized encodings of real numbers [1]. For optimization, a standard binary representation of numbers can be problematic – consider representing 127 in binary: 01111111 and 128 in binary: 10000000; though 127 & 128 follow each other, their Hamming distance is 8. For a search algorithm to move from one value to the next, the number of modifications required to the bit-string is large. These 'Hamming cliffs' are commonly addressed through encoding the parameters with Gray Codes [2][3]. With Gray Codes, consecutive items have a Hamming distance of 1. 127 & 128 are represented as 01000000 & 11000000 – a single bit flip moves to nearby values. This is

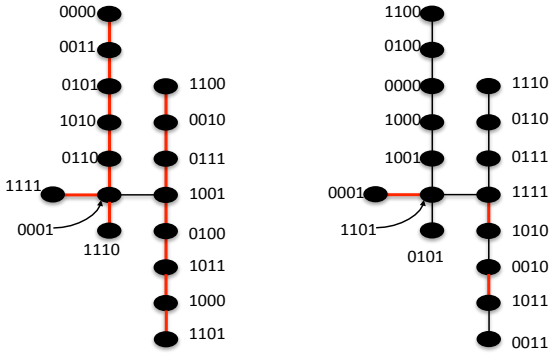


Fig 1. 16 vertex graph with 15 edges. Left: nodes labeled randomly, summed Hamming distance between connected nodes is 40. Right: better node labeling, summed Hamming distance is 18. *Edges with hamming distance > 1 are marked with thick red lines.*

particularly important in the latter stages of search when search has landed in a basin of high performance [4][5] and local optimizations are required for improvement.

Gray code has been extensively studied in the evolutionary algorithm literature over several decades [6][7][8][9][10]. The neighborhood-preserving encodings presented in this paper have applicability beyond ordinal number representation. For example, within a graph, if the binary labels are assigned to nodes randomly, Hamming cliffs along the edges may be prevalent, Fig 1 (left). In Section II, we present our algorithm to remove the Hamming cliffs in arbitrary graphs, Fig. 1 (right). The derived encodings share an important property with Gray-code: small changes in the candidate solution allow search to explore local neighborhoods. Although the label assignment procedure does not guarantee single-bit Hamming distances between close items, it significantly lowers the Hamming distances found by random assignment.

Numerous applications exist which could benefit from this label assignment, including selecting prototype samples for machine learning [11], point selection for object localization [12], and a multitude of classic NP hard problems [13][14]. In Section III, we demonstrate how the neighborhood preserving codes are used within a GA to address a number of such tasks.

II. Graph Propagation for Point Labeling

The basis of our approach to finding neighborhood preserving codes is to "propagate" vertex labels along the edges of a graph. For many classes of problems, the underlying graph will be obvious; in others, a graph will need to be inferred. Examples of both types of problems will be given in the next section. For simplicity, in this section, we are given an unlabeled graph that is either sparsely or densely connected and may contain either weighted or unweighted connections. The goal is to label vertices with unique binary labels of length $T = \lceil \log_2 n \rceil$ with minimal Hamming distances for connected vertices.

The algorithm is initialized by randomly assigning unique labels of length T to each of the vertices. Over a number of iterations, the algorithm will iteratively improve the labels to reduce the Hamming distance between connected vertices. In each iteration, each vertex's label is propagated to each of its neighbors. Each vertex accumulates the labels from all of its neighbors, combines the accumulation with its own label, and propagates the accumulated tally to its neighbors in the next step after a normalization step. See Fig 2.

Over successive iterations, each vertex's label will 'reach' all of the nodes in the graph. To ensure that a vertex's influence is more pronounced upon close neighbors, the edge weights are scaled to a positive weight < 1.0. This serves as an exponential decay over connection hops for the influence of a vertex on its neighbors (in all the experiments, the max weight was 0.9). This algorithm is a variant of the *Adsorption* label propagation approach [15], in which YouTube videos were propagated along the inferred co-view graph to find novel video recommendations. In [21], we first explored its use in optimization.

```
Initialize:
  Randomly label each vertex with unique bit-string
  length T.

Propagation Step:
  For each node, n:
       For each node connected to n, m:
          At Node m – record n's label.

Accumulation Step:
  For each node, n:
       For each bit position 1-T, t:
         Initialize Evidence bit positions to label_n[t]
         with weight α
       For each recorded labels at node-n, R:
          Evidence_n [t] = Evidence_n [t] + R[t] .
       Scale Evidence vector to 1.0 max (by normalizing
       by weight of incoming connections and α)

Label Reassignment Step:
  Set P = all possible labels of length T (|P| = 2^T)
       For each node, n:
          For each label in P, p:
                Calculate Distance Evidence_n, p
  Reassign all labels, (one label per vertex) minimizing
  the overall distance between Evidence and assignment
  across all nodes.

Repeat from Propagation Step
```

Fig 2. Graph Propagation for Label Assignment, α set to 0.1. Shown for a uniformly weighted graph. A straightforward extension is possible for non-uniform edge weights [21].

Once the propagation has sufficiently converged, each vertex will have a summary of the labels from all of its neighbors in the form a single vector specifying the distribution of 0's and 1's in each bit position for itself and its neighbors. With these summary statistics, new labels are assigned to each node to replace the initially random labels. The new labels are chosen to minimize the summed distance between the summary statistics in each node and the new node's label. Once the new labels are chosen, the full algorithm is repeated. This continues for a set number of iterations or until the labels no longer change in this reassignment step. At the end of this procedure, each node will have a label similar to its neighbor's label.

Originally, Adsorption was designed for classification tasks that may have sparse data, but in which an underlying "relatedness" graph exists [15][16]. The variant described here, with a vector of binary, unique, labels, was a modification of Adsorption created to provide 'forgiving' hash labels to clustering tasks – in which the goal is to associate nearby clusters (*i.e.,* vector quantization) with minimal distance bit-strings. Adsorption, and this variant, lend themselves to simple iterative computation (similar to PageRank [17]) and are efficient to implement in the MapReduce framework [18].

A. Propagation Variations & Practical Heuristics

The label reassignment step can be expensive (described below). Because the propagation and accumulation steps

work incrementally, it is not necessary to run the reassignment step after each iteration. Instead, the propagation and accumulation can be repeated without invoking label reassignment until either (a) the graph converges or (b) a set number of iterations is reached. This dramatically reduces the computational expense.

A variety of matching procedures can be used for the reassignment step. A simple greedy method will not guarantee an optimal assignment. A procedure popularized in 1957, termed the "Hungarian Method" or the "Munkres-Assignment Algorithm," provides a method for finding the optimal assignment [19][20]. Given an *N*x*N* matrix in which the element in the *i*-th row and the *j*-column represents the non-negative cost of assigning label *i* to point *j*, the Hungarian Method finds the minimal cost assignment. It is often used to assign jobs to workers with minimum cost. Although beyond the scope of this paper to provide details (see [19][20]), the Hungarian Method can be implemented in $O(N^3)$.

Another heuristic that has repeatedly shown to improve the final performance in terms of reducing the summed Hamming distance across edges is to modify the distance calculation measurement employed in the reassignment step. Recall that the distance to be minimized is the summed distance across all nodes *N* between $Evidence_n$ and all binary labels of size *T*. A simple heuristic to promote smoothness across connecting vertices is to weight the *Evidence* at each node with the *Evidence* from all the neighboring connections (note that this differs from simply a propagation step in that the node's assigned label does *not* contribute directly in this step, only the *Evidence* accumulated at the node and its neighbors). In this case, $Evidence_n$ is based on $Evidence_{neighbors\text{-}of\text{-}n}$ (in *Adsorption* parlance, this is equivalent to a step in which a node's self-"injection" labels are not used). This yields dramatic improvements in the result of the labeling procedure.

Fig 3. shows the results for 100 randomly initialized runs for (A) a 256-node "line-graph" where the nodes are arranged in a line and each node is connected to each of its two neighbors (B) a sparsely connected random graph of 256 nodes. The results shown are the summed Hamming distance between connected nodes. For reference, a random labeling of graph-A (510 connections) had a summed Hamming distance of **2049** (close to expected: 510*(8/2)), a randomly labeled graph-B (768 connections): **3076** (~768 * (8/2)).

From Fig. 3, it is first apparent that there is no guarantee of optimal assignments (*i.e.* where all connected nodes have a Hamming distance of 1), since different runs end with different summed Hamming distances. Second, the line-graph was chosen because using a standard Gray-code to label sequential nodes would yield a known optimal assignment. Had a standard Gray-code been found, the summed Hamming distance would be 510. Note that the average summed Hamming distance found was not optimal: 771 for the Hungarian Matching, but it was far better than random (2049). Third, the multi-node distance heuristic worked with both

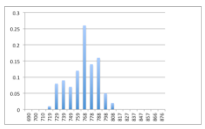
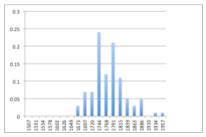
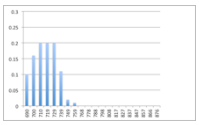
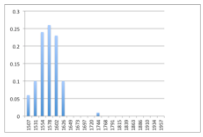
	256 Node Line Graph (510 Connections)		256 Node Random Graph (768 Connections)
Hungarian Matching	avg=771 		avg=1788 
Hungarian Matching & Multi-Node Distance	avg=721 		avg=1587 

Fig 3. The summed hamming distance across connected nodes of two graphs (1) line-graph, left and (2) random graph, right. Distribution of results across 100 trials using just Hungarian and Hungarian with modified node distance. Distributions closer to the left are better (lower summed Hamming distance) (x-axis is constant for each graph type). Average score over 100 trials also given.

graphs, the results were statistically significant ($p<0.001$). All further experiments will employ this distance calculation.

To provide insight into the algorithm progress, we examine four sample runs for a graph with 8192 nodes (Fig. 4). In each run, the graph connectivity was varied: 24,576 edges (3 per node), 32,768 (4 per node) 40,960 (5 per node) and 81,920 (10 per node) respectively. Each graph was constructed so that an optimal assignment existed. Up to 100 label reassignment steps are shown in Fig. 4. The Y-axis of the graph is the ratio of the summed Hamming distance across all edges in the graph / the Hamming distance of the optimal assignment (=number of edges). With connectivity of 10 and 5, the algorithm reaches a ratio of 1.0 (optimal assignment). With C=4 the ratio is 1.07 with Connectivity 3, the ratio is 2.7. In this example, the more constraints provided in the graph, the more rapid the convergence and the better the solution found. The least connected graph neither converged as rapidly nor performed as well as when more edges were provided.

III. Neighborhood Preserving Encodings for Search

To this point, we have presented methods to reduce the summed Hamming distance between connected nodes in a graph. The goal of this section is to demonstrate that the neighborhood preserving point labeling schemes are general and the resultant codes can be easily and effectively used within a standard Genetic Algorithms (GA).

In a recent previous study [21], we demonstrated the use of neighborhood preserving encodings in the context of simple bit-flip stochastic hill-climbing (also see [26]). One of the most salient findings of that study was that the encodings performed best in the latter stages of search – during local optimization. However, during the early stages of search, when diverse exploration was necessary, the use of the encodings actually *hindered* performance. The reason was

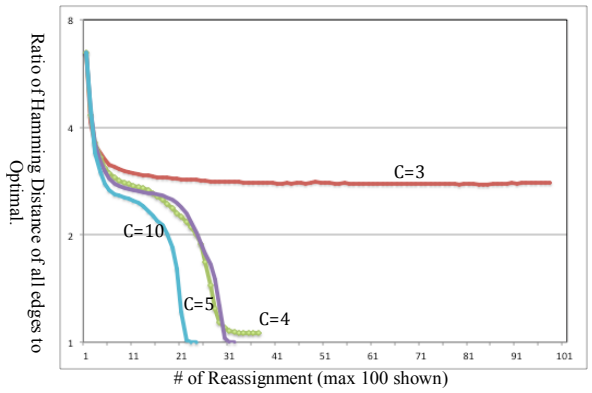


Fig 4. Sample runs with 8192 node graphs. 4 Levels of connectivity are explored. Shown is the Ratio of hamming distances to minimum possible Hamming distance over the first 100 label reassignments. All runs start with ratio of ~6.5 (this is expected as the assignments are random and the label length is 13 (2^{13}=8192)). With Connectivity=10 and Connectivity=5, the ratio is driven to 1.0 (optimal assignment). With C=4, ratio is 1.07, with Connectivity=3, ratio is 2.7.

the nature of the hillclimber tested – it operated with single bit-flips. With single bit flips, the probability of large moves is, by design, reduced using these encodings. If these encodings were used only in the later portions of search, once a basin of good performance was found and small moves were required for local optimization, these reduced Hamming distance encodings far outperformed random encodings. This finding led to a simple heuristic: use random encodings for the initial portion of search, and switch to reduced-Hamming distance encodings once progress slowed [21].

Analogously, in a number of previously published studies, it was found that GAs very quickly find regions of high performance, but incorporating hill-climbing (HC) methods often yields better results than simply continuing the GA for more iterations [13][22][23][24][27]. Though numerous methods for incorporating HC into GAs have been explored, the simplest is in two consecutive stages: Stage-1: run a simple, generational, GA to find a region of high performance. Stage-2: starting with the best result found by the GA, use HC for local optimization. This approach will be used here.

The Stage-1 generational GA was run with the following parameters: Population Size: 50, Crossover Type: Two Point (performed equivalently to uniform), Mutation Rate: 1%-10% (multiple were tried) and Elitist Selection (the best solution from one generation is passed to the next unperturbed). The GA is run until a local optima is found (1000 consecutive evaluations without improvement). The GA's best solution initializes the stochastic next-ascent hill-climbing (HC) step.

The Stage-2 HC algorithm begins with the GA's best solution string. It randomly perturbs a bit in the solution string, evaluates the new string, keeps the perturbation if the evaluation has improved or stayed the same, and discards the perturbation if the evaluation has worsened with the perturbation. Like the GA, HC is continued until 1,000 consecutive evaluations without improvement. The best result found from the HC step is returned as the final answer.

Our primary goal is to determine, in a combined scheme of GA+HC, whether there is a need for using the neighborhood preserving mappings or whether this encoding's benefit is mitigated when two search algorithms with complementary strengths (global and local optimization) are combined. [1] Additionally, we examine whether the Hungarian-based labeling is useful with a GA itself. We examine four variants:

1. **GA(R1)+HC(R1):** The bitstring to vertex mapping for the GA is chosen randomly. The same mapping is then used with the hill-climbing algorithm. This is the baseline.
2. **GA(R1)+HC(R2):** The bitstring to vertex mapping for the GA is chosen randomly. A newly chosen, but still random, mapping is used with the hill-climbing algorithm. This variant determines whether simply using a new encoding for the hill-climbing stage provides the same benefit as the neighborhood-preserving mappings.
3. **GA(R1)+HC(Hungarian):** The bitstring to vertex mapping for the GA is chosen randomly. However, the hill-climbing mapping is chosen through the Hungarian variant of the graph propagation algorithm. This test is included to determine whether a reduced hamming distance encoding is more important for local optimization (Stage 2) than for the initial stages of search.
4. **GA(Hungarian) + HC (Hungarian):** The Hungarian mapping is used for both the GA and HC.

In addition to the results presented here, we completed two sets of experiments not shown due to space restrictions. First, we experimented with a GA without an HC step – using both random and Hungarian encodings. The results without an HC step faired consistently worse than those where an HC step was employed. Second, we experimented with greedy matching (a non-optimal reassignment step shown in Fig 2.) instead of Hungarian matching. In most cases, there was little difference between Hungarian and greedy matching. In the majority of runs where there was a significant ($p<0.001$) difference, the Hungarian matching performed better. This is similar to the results found when using only HC [21].

We begin our empirical examination with the *Long-Shortest Path* problem. Given a sparsely connected graph of N vertices, the goal is to select K points that have the longest

[1] In this paper, the HC and GA algorithms operate on binary strings. Nonetheless, it is possible to construct variants with operators that move directly in graph space without an intermediate encoding. Numerous studies (see references) have been conducted examining the efficacy of binary encodings, particularly in the context of GAs; we will not enter that debate here. Rather, if the problem to be addressed is of the select-*p*-of-*N* points variety, our goal is to improve the performance of any algorithm that employs a binary encoding for exploring the search space.

shortest-paths between the points in K. We would like to maximize the sum of (K*K) distances. For the experiments, graphs are randomly generated with N=1024 nodes with varying levels of connectivity, with the task of selecting K=20 nodes with the furthest distance from each other. The solution string is represented with 200 bits (20*10); each contiguous substring of 10 bits represents a selected vertex.

Before introducing the Stage 2-HC, we first examine the performance of the GA with random vs. Hungarian-assignments of labels. With 480 trials tested with 1024-node graphs created with random 10-node connectivity, 215 trials were better with random labels, 265 with Hungarian, and there was no statistical difference in the quality of the best results found. With graphs that were created with varying numbers of clumps of clustered connectivity (instead of uniformly random connectivity), of the 480 trials, 225 were better with random labels, 255 with Hungarian, again with no statistical difference in the best results found. For this problem, when a GA is used in isolation, we did not see much difference in performance between using random encodings and neighborhood preserving encodings.

Why is there so little difference in performance? Unlike HC, which explores the search space through single bit flips, the crossover and mutation operators can be disruptive to the candidate solutions. Therefore, the intelligent labeling does not have the same detrimental impact of reducing exploration in the early stages of search as it did with single-bit-flip-HC, as reported in [21]. However, this only explains why the results with Hungarian labeling were not *worse*. Why were the results not *better* using Hungarian labeling? While GAs excel at rapidly finding regions of high performance, they are often less effective in the latter stages of search, which require smaller moves for local optimizations [7][8]. One reason for this is that the finite population of points from which the GA searches may converge too quickly (rendering crossover ineffective), thereby making mutation the primary search operator. However, mutation in a population of points is inefficient as compared with other, simpler, methods. As will be demonstrated in the next few sections, when a Stage 2, local-optimization hill-climbing step is introduced, the outcomes with neighborhood preserving mappings overwhelmingly outperform other encodings.

A. Test Problem 1: Long-shortest Paths

Throughout the experimental sections, a large number of empirical results will be presented. Because the problems are randomly generated, giving the raw evaluation numbers yields little insight. Instead, we present comparative results. For each comparison, we give the number of times algorithm A performed better than algorithm B and vice-versa. Ties are not attributed to either algorithm. We also tell whether the average performance, in terms of best evaluation found, was statistically different from each other (> 99% confidence using a paired two-tail t-test).

In Table 1 and Table 2, we compare the performances of 4 GA+HC variants as the size of the graphs (N) and the number points to be selected (K) are varied. Row 1, (GA(R1)+HC(R2) Vs. GA(R1)+HC(R1), tests whether the fact that the vertex encoding between the two stages changes improved performance by itself – note that the vertex encoding was switched to another *randomly* selected encoding (R2) with no neighborhood preservation properties. Previous experiments with a single search algorithm [9][21] showed that simply changing the representation to another, even random, representation helped escape local optima. Here, however, we witness a less consistent improvement. Likely, because changing the search algorithm (from GA to HC) changed the primary search operator, that served the same role for escaping local optima as changing the encoding. The encoding change had only a secondary effect.

Next, we test whether adding neighborhood preserving properties to the encoding can improve the results. The effects of using the Hungarian encoding with Stage 2-HC are shown in Row 2. The GA still uses the random encoding for these tests. The majority of the cases improved significantly over using a random encoding with HC. Row 3 shows that this improvement remains even if the same Hungarian coding was used in both the Stage-1 GA and Stage-2 HC. This indicates that the *change* in encoding was not the driver of the improved performance, it was the encoding itself.

Finally, we compare the results of using the neighborhood preserving codes throughout the entire search, and only in Stage-2. Row 4 of both tables shows no statistical difference between using a random or Hungarian encoding for the GA stage when the Hungarian encoding is used in Stage 2 for local optimization. This corroborates the findings in [21] – the effects of neighborhood preserving codes are most pronounced in local optimization.

Table 1. Longest Shortest-Paths, N=256,512,1024 Vertices. C=Connectivity 3. K = 20. Random Graph

		N=256	N=512	N=1024
1	GA(R1)+HC(R2) Vs. GA(R1)+HC(R1)	51/68 No	59/61 No	73/47 No
2	GA(R1)+HC(Hung) Vs. GA(R1)+HC(R2)	77/42 Yes	79/40 Yes	78/42 Yes
3	GA(Hung)+HC(Hung) Vs. GA(R1)+HC(R2)	81/35 Yes	70/50 Yes	78/41 Yes
4	GA(Hung)+HC(Hung) Vs. GA(R1)+HC(Hung)	66/54 No	52/66 No	65/53 No

Table 2. Longest Shortest-Paths, N=1024 Vertices. C=3. Random Graph

		K=5	K=10	K=20	K=100
1	GA(R1)+HC(R2) Vs. GA(R1)+HC(R1)	44/52 No	69/49 No	73/47 No	90/30 Yes
2	GA(R1)+HC(Hung) Vs. GA(R1)+HC(R2)	76/30 Yes	76/40 Yes	78/42 Yes	79/41 Yes
3	GA(Hung)+HC(Hung) Vs. GA(R1)+HC(R2)	77/36 Yes	76/43 Yes	78/41 Yes	62/58 No
4	GA(Hung)+HC(Hung) Vs. GA(R1)+HC(Hung)	58/48 No	65/52 No	65/53 No	61/59 No

We next experiment with increasing the graph's connectivity (Table 3). This is important to determine whether using the label propagation techniques can be applied to highly overconstrained problems. From Table 3, we again see that introducing a second, random, mapping rarely improves performance (Row 1). Importantly, using the Hungarian-derived-mapping usually outperforms just using random mappings (Rows 2 & 3). The results shown in Row 4 are similar to those seen earlier; in Stage 1-GA, the encoding was less important than the encoding used for the HC; the results in Row 4 are not statistically different.

Table 3. Longest Shortest-Paths, N=1024. C= 10. Random Graphs.

		K=5	K=10	K=20	K=100
1	GA(R1)+HC(R2) Vs. GA(R1)+HC(R1)	16/8 No	26/38 No	54/50 No	58/62 No
2	GA(R1)+HC(Hung) Vs. GA(R1)+HC(R2)	36/16 Yes	98/2 Yes	68/34 Yes	66/54 No
3	GA(Hung)+HC(Hung) Vs. GA(R1)+HC(R2)	50/28 Yes	86/20 Yes	78/34 Yes	58/62 No
4	GA(Hung)+HC(Hung) Vs. GA(R1)+HC(Hung)	38/34 No	26/32 No	56/44 No	54/62 No

For the remainder of the experiments, we omit presenting the results for changing between two random encodings: GA(R1)+HC(R2) vs. GA(R1)+HC(R1); most cases were not significantly different, and those that were, favored using a different random encoding for the GA and HC steps. Therefore, all future comparisons will be made to the better of the two, GA(R1)+HC(R2). We retain the harder of the two baselines to compare with the neighborhood preserving codes.

The comparisons that remain are those that focus on the effects of using the Hungarian encoding for HC (Row 2), and using the Hungarian encoding for both (Row 3). These two columns provide the evidence to determine whether the neighborhood preserving codes for local optimization outperform random encodings. For completeness, Row 4 is kept, though there is rarely any significant difference in performance in the random vs. Hungarian encodings used for Stage 1-GA as long Stage 2-HC uses the Hungarian encoding.

B. Test Problem #2: Shortest Distance to Selected Vertices

In this problem, the goal is to select K vertices from an N node graph such that the summed distance of all the vertices in N to a member of K is minimized. If the graph is planar, this is a grossly-simplified problem of cell-phone tower layout. Like the previous problem, this problem is parameterized with K,N, and C (the connectivity of the graph).

In Table 4, experiments are conducted varying the number of nodes in the graph; we always see a large improvement in using the Hungarian-derived mappings over random labels.

The results in Table 5 indicate that even as the number of points to be selected increases, the same performance trends hold across random and cluster graphs. Local optimization consistently benefits from neighborhood preserving mappings – regardless of whether the Hungarian mappings are used for the GA portion. As before, when the neighborhood preserving codes are used for Stage-2 HC, there is little performance difference in the encoding used for the GA (Row 3, both tables).

Table 4. Effects of Graph Size. N=256,512,1024 Vertices. C=Connectivity 3. K = 20. Random Graphs.

		N=256	N=512	N=1024
1	GA(R1)+HC(Hung) Vs. GA(R1)+HC(R2)	105/10 Yes	110/8 Yes	92/23 Yes
2	GA(Hung)+HC(Hung) Vs. GA(R1)+HC (R2)	105/11 Yes	111/8 Yes	85/34 Yes
3	GA(Hung)+HC(Hung) Vs. GA(R1)+HC(Hung)	56/59 No	58/60 No	62/56 No

Table 5. Effects of varying K. N=1024 Vertices. Cluster Graph, Random Graph. All connectivities.

		K = 5	K = 10	K = 20	K = 100
1	GA(R1)+HC(Hung) Vs. GA(R1)+HC(R2)	429/85 Yes	492/102 Yes	497/97 Yes	424/161 No
2	GA(Hung)+HC(Hung) Vs GA(R1)+HC (R2)	404/124 Yes	463/134 Yes	477/118 Yes	421/166 Yes
3	GA(Hung)+HC(Hung) Vs. GA(R1)+HC(Hung)	238/240 No	293/300 No	307/287 No	309/265 No

C. Test Problem #3 – Multi-Dimensional Knapsack

In this problem, there are *N* objects with *D* traits (weight, volume, etc.) and an associated Value (such as $). The object is to find the set of objects that can fit into a knapsack with maximal value, subject to the constraint that the summed value of any of the traits not exceed the knapsack's capacity for that trait. The problems are generated randomly; each of the *D*-traits and the value are drawn uniformly from 1-1000.

Unlike the two graph problems explored in the previous sections, there is no explicit graph to propagate the labels from which to create the neighborhood-preserving mappings. Instead, we must first synthesize an appropriate graph. We represent each object as a node in the graph. To determine which objects are connected to each other (the edges in the graph), we concatenate each object's traits and value into a single vector and connect the object to the *S* other most similar objects, as measured by the l_2-norm difference between the object's vectors. In these experiments, it should be noted that the edges are not necessarily symmetric. In the completed graph, each node is connected to at least *S* other nodes. With this, we can proceed with the same propagation mechanisms used in previous experiments – with the labels derived from propagation in this graph. The solution string was encoded as a bit-string of selected items. If items were represented more than once in the bit-string, they were still added only once. Items were added until the knapsack exceeded its volume in

any dimension; then the total value of the items in the knapsack was returned as the value of the bit-string.

In Table 6 and Table 7, we examine the effects of increasing the problem difficulty. Three sizes of the knapsack are explored: ranging from $1/5^{th}$ of the expected total in any dimension to $1/20^{th}$ of the expected total in any dimension – *i.e.* if there are 256 nodes, the total in any dimension is expected to be 256*(1000/2)=128,000, therefore for the hardest case of $1/20^{th}$ the expected total, the knapsack is limited to a total 6,400 in any dimension. In Table 6, each item is has D=9 traits, in Table 7, D=1 trait. The larger the number of traits, the more unlikely it is to find high value items that are small across all traits. We expect a more uniform performance across algorithms as D increases.

Table 6. Multi-Dimensional Knapsack. N=1024, 512, & 256 Vertices. Graph modeled 10 Nearest Neighbors, D = 9 Traits

		Loose Constraint $1/5^{th}$ total	Medium Constraint $1/10^{th}$	Tight Constraint $1/20^{th}$
1	GA(R1)+HC(Hung) Vs. GA(R1)+HC(R2)	234/126 Yes	267/93 Yes	266/94 Yes
2	GA(Hung)+HC(Hung) Vs GA(R1)+HC (R2)	238/122 Yes	268/92 Yes	267/93 Yes
3	GA(Hung)+HC(Hung) Vs. GA(R1)+HC(Hung)	178/182 No	177/182 No	183/177 No

Table 7. Multi-Dimensional Knapsack. 1024, 512, & 256 Vertices. Graph modeled 10 Nearest Neighbors, D = 1 Trait

		Loose Constraint $1/5^{th}$ total	Medium Constraint $1/10^{th}$	Tight Constraint $1/20^{th}$
1	GA(R1)+HC(Hung) Vs. GA(R1)+HC(R2)	331/29 Yes	196/164 No	239/121 Yes
2	GA(Hung)+HC(Hung) Vs GA(R1)+HC (R2)	340/20 Yes	210/150 Yes	236/124 Yes
3	GA(Hung)+HC(Hung) Vs. GA(R1)+HC(Hung)	199/161 No	210/150 Yes	187/173 No

As in the previous problems, the introduction of the Hungarian encoding in the HC portion of the optimization provided significant benefits in every case in terms of wins. The Hungarian encoding was beneficial in almost every case in terms of average results found, too – in all cases for (D=9) – Table 6, and in all but one case in the problem sets shown in Table 7. As before, as seen in Row 3, for the GA, using the Hungarian assignments vs. Random did not have a significant impact on performance in the majority of the cases as long as the Hungarian Encoding was used for Stage 2-HC.

D. Test Problem #4: Multi-Dimensional Partitioning

In this problem, the goal is to divide a set of numbers G into two subsets G_1 and G_2 such that the sum of G_1 is as close to the sum of G_2 as possible. We add a twist to the problem by having a set of vectors (of length D) in G instead of numbers. The goal is to divide G into G_1 and G_2 where the l_1-norm difference between the sums of vectors G_1 and G_2 is minimized. Similar to the knapsack problem explored in Section 3.4, there is no explicit graph given. Instead, we construct the graph like the knapsack synthetic graphs – by finding the most similar, S=10, vectors. The results for the GA+HC combinations are shown in Table 8 and Table 9.

For both cases, with 10 Dimensions (Table 8) and 2 Dimensions (Table 9), the results always favor using the Hungarian-labeling for HC. As we have seen in many problems, the difference between using the Hungarian labeling for the GA and not was not significantly better as long as the the Stage-2 HC also used the Hungarian labeling.

Table 8. Results – Partitioning with GA+HC. D= 10 Dimensions. S = 10 (10 similar vertices are connected to each vertex)

		N=256	N=512	N=1024
1	GA(R1)+HC(Hung) Vs. GA(R1)+HC(R2)	95/25 Yes	87/33 Yes	101/19 Yes
2	GA(Hung)+HC(Hung) Vs GA(R1)+HC (R2)	90/30 Yes	90/30 Yes	99/21 Yes
3	GA(Hung)+HC(Hung) Vs. GA(R1)+HC(Hung)	56/62 No	63/56 No	65/55 No

Table 9. Results – Partitioning with GA+HC. D=2 Dimensions. S=10 (10 similar vertices are connected to each vertex)

		N=256	N=512	N=1024
1	GA(R1)+HC(Hung) Vs. GA(R1)+HC(R2)	110/9 Yes	114/6 Yes	118/2 Yes
2	GA(Hung)+HC(Hung) Vs GA(R1)+HC (R2)	117/3 Yes	116/3 Yes	119/1 Yes
3	GA(Hung)+HC(Hung) Vs. GA(R1)+HC(Hung)	54/38 No	60/51 No	52/57 No

Recall that for this problem (and the knapsack problem explored in the previous section), we had to synthesize a graph on which to propagate the labels. We modeled the S=10 most similar vectors for the connections between the vertices. For a final experiment, we see how sensitive the results are to this parameter. If there are too few connections in the graph, we expect there to be less information to exploit in labeling and therefore a reduction in improvement seen. (Conversely, too many constrains and the propagation may degrade due to a muddied signal, fewer relevant vs. irrelevant connections, etc.) We reduce the number of similar vectors that are modeled in the graph by 80%, to S=2. We repeat the experiments from Table 8, setting D=10. The results are shown in Table 10.

Like the results shown with representing S=10 similar vectors in the graph, the same trends hold with the Hungarian labeling as before, it provides a benefit in almost every case, though not as strongly in these tests. The percentage of wins of the Hungarian over random labeling reduces in many cases. This is what we expected; by severely reducing the number of modeled connections, we have removed a significant amount

of information implicitly provided to the algorithm through the labeling. This also reaffirms an underlying assumption of this study – the graph from which the point labels are derived contains problem specific information that can be effectively used in the search algorithms through the neighborhood preserving encodings. The more/cleaner the information in the graph, the more beneficial it will be.

Table 10. Results – Partitioning with GA+HC. D= 10 Dimensions. S=2. (2 similar vertices are connected to each vertex)

		N=256	N=512	N=1024
1	GA(R1)+HC(Hung) Vs. GA(R1)+HC(R2)	54/26 Yes	41/39 No	51/29 Yes
2	GA(Hung)+HC(Hung) Vs GA(R1)+HC (R2)	62/18 Yes	52/28 Yes	52/28 Yes
3	GA(Hung)+HC(Hung) Vs. GA(R1)+HC(Hung)	48/32 No	42/38 No	38/42 No

IV. Conclusions & Future Work

There are two major contributions of this paper. The first is a technique to encode binary-string labels on nodes in arbitrarily dense graphs that reduces the labels' Hamming distance between connected nodes. This technique is based on graph propagation of labels commonly found in machine learning literature [15][16].

The second contribution is a large empirical demonstration of the effectiveness of these labels within stochastic search algorithms for exploring local neighborhoods once regions of high performance are found. Additionally, when employing a GA for the early stages of search, there was no significant difference in using these vs. random encodings, as long as the neighborhood encodings were used for Stage-2 local optimization. This contrasts earlier studies [21], which showed that when HC with single-bit-flips was used for Stage-1 exploration, the neighborhood encodings were too restrictive. GAs did not suffer from this drawback. In summary, through an extensive empirical comparison encompassing both problems that had explicit graphs and those for which the graph had to be constructed, the results pointed to the efficacy of using the reduced-Hamming distance labels for local optimization.

Looking forward, there are three directions for research. First, the solutions to the problems presented in this paper were encoded as binary-strings. However, there is no inherent limitation of the propagation algorithm to a binary alphabet. Higher cardinality alphabets can be accommodated and have already been explored outside the optimization domain [15].

Second, we have noted that in graphs where label assignments exist such that connected nodes can have a Hamming distance of 1, the more connections that are present in the graph, the more likely it is that the labeling algorithm approaches the optimal solution (Fig 4), especially when the heuristic of modifying the distance calculation, as described in Section II.A, is employed. This warrants further study to understand the ideal conditions for this labeling procedure.

Third, the propagation algorithm works equally well with directed and undirected graphs and with non-uniformly weighted edges. These extensions were not comprehensively explored in this paper. Nonetheless, many problems in which a graph is inferred based on similarity of points (such as the Knapsack and Partitioning problems) are particularly well suited to these variants. This is left for future study.

References

[1] De Jong, K. A. (1975). *Analysis of the behavior of a class of genetic adaptive systems.*

[2] Gray, F. (1953). U.S. Patent No. 2,632,058. Washington, DC: U.S.P.T.O.

[3] Savage, C. (1997). A survey of combinatorial Gray codes. *SIAM Review*, 39(4), 605-629.

[4] Renders, J. M., & Flasse, S. P. (1996). Hybrid methods using genetic algorithms for global optimization. *IEEE-SMC-B 26*(2), 243-258.

[5] Lourenço, H., Martin, O., & Stützle, T. (2003). Iterated local search. *Handbook of metaheuristics*, 320-353.

[6] Caruana, R. A. & Scharffer, J.D. (1988). Representation and hidden bias: Gray vs. binary coding for genetic algorithms. In *ICML-5*.

[7] Harada, K, Ikeda, K., Kobayashi, S., 2006. Hybridization of genetic algorithm and local search in multiobjective function optimization: recommendation of GA then LS. *Proc. of the 8th GECCO*, 667-674

[8] Reeves, C. (1994) Genetic Algorithms and Neighbourhood Search, *Evolutionary Computing Lecture Notes Volume* 865, 1994, pp 115-130

[9] Rowe, J., Whitley, D., Barbulescu, L, Watson, J.P., 2004, Properties of Gray and Binary Representations, *Evo. Computation* 12(1): 47-76.

[10] Mathias, K. E., & Whitley, L. D. (1994). Transforming the search space with gray coding. *Proc. IEEE Conf. on Evolutionary Comp*. 513-518

[11] Skalak, D. B. (1994). Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *ICML-11* (pp. 293-301).

[12] Baluja, S., & Simon, D. (1998). Evo.-based methods for selecting point data for object local.: app. to comp.-assisted surgery. *App. Int.* 8 7-19.

[13] Duvivier, D., Preux, P., & Talbi, E. (1996). Climbing up NP-hard hills. *Parallel Problem Solving from Nature—PPSN IV*, 574-583.

[14] De Jong, K. A., & Spears, W. M. (1989). Using genetic algorithms to solve NP-complete problems. In *Proc. ICGA-3* 124-132.

[15] Baluja, S., Seth, R., Sivakumar, D., Jing, Y., Yagnik, J., Kumar, S., & Aly, M. (2008). Video suggestion and discovery for youtube: taking random walks through the view graph. *17th WWW* 895-904.

[16] Zhu, X., & Ghahramani, Z. (2002). *Learning from labeled and unlabeled data with label propagation*. CMU-CALD-02-107.

[17] Page, L., Brin, S., Motwani, R., & Winograd, T. (1999). The PageRank citation ranking: bringing order to the web. SIDL-WP-1999-0120.

[18] Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *CACM*, *51*(1), 107-113.

[19] Munkres, J. (1957). Algorithms for the assignment and transportation problems, *J. Soc. for Ind. & Applied Mathematics*, *5*(1), 32-38.

[20] Kuhn, H. W. (2006). The Hungarian method for the assignment problem. *Naval research logistics quarterly*, *2*(1-2), 83-97.

[21] Baluja, S. & Covell, M. (2013) Neighborhood Preserving Codes for Assigning Point Labels: Applications to Stochastic Search. To Appear in: *4th Workshop on Comp. Opt., Modelling and Sim.* (COMS2013)

[22] Wattenberg, M., & Juels, A. (1996, June). Stochastic hill-climbing as a baseline method for evaluating genetic algorithms. *NIPS-8, p 430.*

[23] Chakraborty, U. K., & Janikow, C. Z. (2003). An analysis of Gray versus binary encoding in genetic search. *Informat. Sciences*, 156(3), 253-269.

[24] García-Martínez C, Lozano M. (2008) Local search based on genetic algorithms. *Advances in metaheuristics for hard optimization*.199–221.

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

[26] Rothlauf, F., 2003, "On the Locality of Representations," *GECCO-2003*

[27] El-Mihoub, T.A., Hopgood, A.A., Nolle, L. Battersby, A., 2006, Hybrid Genetic Algorithms: A Review. *Eng Letters* 13, p 124-137.