Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 18 (2013) 956 – 965

Procedia Computer Science

2013 International Conference on Computational Science

Neighborhood Preserving Codes for Assigning Point Labels: Applications to Stochastic Search

Shumeet Baluja, Michele Covell

Google Inc. 1600 Amphitheatre Parkway, MountainView, CA. USA

Abstract

Selecting a good representation of a solution-space is vital to solving any search and optimization problem. In particular, 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, it is common for stochastic search algorithms, such as stochastic hillclimbing, evolutionary algorithms (including genetic algorithms), and simulated annealing, to employ Gray Codes for encoding ordinal points or discretized real numbers. In this paper, we present a novel method to label similar and/or close points within arbitrary graphs with small Hamming distances. The resultant point labels can be seen as an approximate high-dimensional variant of Gray Codes with standard Gray Codes as a subset of the labels found here. The labeling procedure is applicable to any task in which the solution requires the search algorithm to select a small subset of items out of many. Such tasks include vertex selection in graphs, knapsack-constrained item selection, bin packing, prototype selection for machine learning, and numerous scheduling problems, to name a few.

Keywords: Search Representation; Gray Code; Graph Propagation, Adsorption, Graph Labeling, Genetic Algorithms

1. 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, 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 solution and testing the result. The difficulty with random assignment of labels to vertices/items is that making 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

doi:10.1016/j.procs.2013.05.261

to the point prior to the flip. Conversely, another problem is that making small moves (*i.e.*, 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.

Consider a typical optimization example in which an objective function's parameters are discretized encodings of real numbers [1]. A standard binary representation can be problematic, for example representing 127 in binary: 01111111 and 128 in binary: 10000000; though 127 & 128 follow each other, their Hamming distance is 8. For the 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 particularly important in the latter stages of search when the search algorithm has landed in a basin of high performance [4][5] and local optimizations are required for further improvement.

The importance of Gray code has been extensively studied in the evolutionary algorithm and GA literature over several decades [6][7][8][9][10]. The neighborhood-preserving encodings presented in this paper 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. The most obvious class of problems suited for this encoding is selecting nodes from a graph subject to some criteria, see Fig.1 (left). If the binary labels are assigned to the nodes randomly, Hamming cliffs along the edges may be prevalent, Fig.1 (middle). Next, we present an algorithm to remove these Hamming cliffs, Fig. 1 (right).

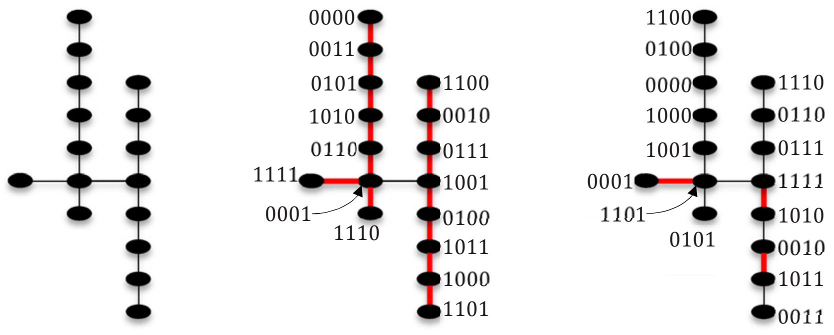


Fig. 1. (left) unlabeled 16 vertex graph with 15 edges. (middle) nodes labeled randomly, summed Hamming distance between connected nodes is 40. (right) better node labeling, summed Hamming distance is 18. *Edges with hamming distance > 1 are marked with thick red lines.*

2. Neighborhood-preserving Point Labeling through Graph Propagation

The basis of our approach 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 will assume that 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, see Fig. 1. As a secondary consideration, if the edges are weighted, effort should be concentrated on small Hamming distances for connections with the larger weights.

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 set to 0.9). This algorithm is a variant of the *Adsorption* label propagation approach [11], in which YouTube videos were propagated along the inferred co-view graph.

```
Initialize:
        Randomly label each vertex with a unique bit-string of length T.
Propagation Step:
        For each node, n:
                For each node connected to n, m:
                        At Node m – record n's label.
                        (if weighted connections, record W(n,m))
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] .
                                (if weighted connections, weight by W(n,m))
                        Scale evidence vector to 1.0 max (by normalizing by weight)
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.

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.

Adsorption was originally designed for classification tasks that may have sparse data, but in which an underlying "relatedness" graph exists [11][14]. Adsorption, and this variant, lend themselves to simple iterative computation (similar to PageRank [12]) and are efficient to implement in MapReduce [13].

2.1. Graph Propagation Variations and Practical Considerations

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 every 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. The greedy method, which does not guarantee an optimal assignment, is $O(N^2\log_2(N^2))$. Simply sort the N^2 distances between evidence_n and label_m, and assign the next pair that has an available label and an unassigned node in the order determined by the sort.

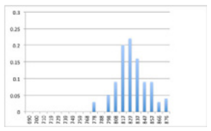
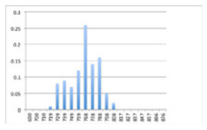
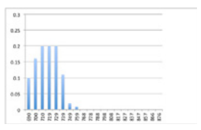
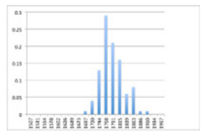
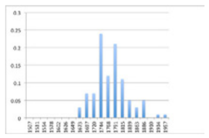
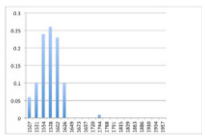
This simple greedy method does not guarantee an optimal assignment. A procedure popularized in 1957, termed the "Hungarian Method" or the "Munkres-Assignment Algorithm," provides a method for finding the optimal assignment [15][16]. Given an NxN matrix in which the element in the i-th row and the j-column represents the non-negative cost of assigning label i to point j, the Hungarian Method finds the minimal cost assignment. The algorithm is often used to assign jobs to workers with minimum cost. Although beyond the scope of this paper to provide details (see [15][16]), the algorithm can be implemented in $O(N^3)$.

We demonstrate the differences in the end assignments with the greedy vs. Hungarian method by labeling two graphs. 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 conns.): 3076 (~768 * (8/2)).

Fig 3. Distribution of results using Greedy vs .Hungarian Matching (Top) A 256 Node Graph connected as a line. (Bottom Row) a 256 node sparsely connected graph.

Distributions closer to the left indicate better performance (lower summed Hamming distance)

x-axis is the same across all graphs in a row.

	Greedy Matching	Hungarian Matching	Hungarian & Multi-Node Distance
256 Node Line Graph (510 conns)	avg = 835 	avg=771 	avg=721 
256 Node Random Graph (768 conns)	avg=1800 	avg=1788 	avg=1587 

From Fig. 3, first it is apparent that there is no guarantee of optimal assignments (where connected nodes have a Hamming distance of 1) even if an optimal assignment exists. Second, the line-graph was chosen for experimentation because using a standard Gray-code to label sequential nodes would yield an optimal assignment. Had a 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 far better than random (2049). Third, there is a significant difference ($p<0.001$) between the greedy and Hungarian-Matching based approaches for the line graph. Though the average Hamming distance using Hungarian vs. Greedy is also lower for the randomly connected graph, the difference is not significant.

A second heuristic 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). Here, $Evidence_n$ is based only on $Evidence_{neighbors\text{-}of\text{-}n}$. This yields dramatic improvements in the result of the labeling procedure, as shown in Fig. 3 (right). The differences in performance when employing neighbor-distances compared to not employing them is significantly better ($p < 0.001$). All further experiments will use this distance calculation.

3. Experimental Results – Hillclimbing

To this point, we have presented methods to reduce the summed Hamming distance between connected nodes in a graph. In this section, we examine their use in stochastic hillclimbing. Hillclimbing (HC) is one of the simplest stochastic search techniques; see Fig. 4. Although simple, many problems tackled with more complex search algorithms are equally well tackled with this approach [17]. The HC used here works on a binary alphabet and the operator is a single, randomly chosen, bit flip. It is a greedy, *next-ascent*, algorithm: as soon as an improvement is found through random perturbation, the new solution is accepted. When a perturbation leads to a solution that has an *equal* evaluation to the current best, it is also accepted; this allows the algorithm to explore plateaus (regions of equal value) in the evaluation space.

The mappings from bit-string to vertices (*i.e.* the labels found by the algorithms described in the previous section) is represented by the mapping, M. The search algorithm, HC, stays the same, regardless of the encoding used. Only the evaluation function interprets the binary solution string, to map the binary string to vertices, which are then evaluated for how well they meet the problem specific objective function.

Fig 4. Next-Ascent, Single Perturbation Hillclimbing (HC) for Maximization Problems encoded with a binary alphabet. Z, the maximum number of perturbations tested without an improvement, is set to 1000.

```
Initialize:
        Generate Binary Candidate Solution, C.
        Evaluate C, Using Point Mapping, M ➔ V
        MovesWithoutImprovement = 0

Repeat:
        Flip a single, randomly chosen, bit in C ➔ C'.
        Evaluate C', using bit-string to point Mapping, M ➔ V'
        If (V' < V):
                Undo perturbation.
                MovesWithoutImprovement ++
        Else if (V' == V):
                MovesWithoutImprovement ++
        Else:
                C = C' ; V = V'; MovesWithoutImprovement = 0

        ## Test End Condition Met.
        If (MovesWithoutImprovement == Z):
                Break Loop – Return C, V. Quit.
```

An important aspect of this algorithm is tracking when progress has stopped. The variable, *Z*, specifies the number of perturbations allowed without finding an improvement in the evaluation. If this number is exceeded, it is assumed that the algorithm has reached a (local/global)-optima and the algorithm is terminated.

3.1. The Interplay of Encoding and Search

Before naively using the vertex encodings derived in the previous section, it is important to understand the interplay between HC and the encodings used. Recall that the HC algorithm uses *single bit flips* to explore the search space. With a random binary encoding for the vertices (Fig. 1-middle), a bit flip has the potential to radically change the selected vertex. However, using the reduced-Hamming encodings, a bit flip has a higher probability of only making a small change (in terms of moves along edges in the graph). For exploration, this has benefits and drawbacks. In the early stages of search, when wide exploration is crucial, it is possible that using a reduced-Hamming encoding that constrains movement in the graph *may effectively limit the exploration of HC*. This leads to a revision of the HC algorithm described above. We perform HC in two stages. In the first stage, termed HC(Random), we use a random mapping from bit-strings to vertices. When a local-optima is reached (i.e. ***Z*** moves are made without improvement) we decode the binary solution string to the nodes it represents, and then re-encode those nodes with the reduced-Hamming-encoding, and restart the optimization procedure HC (Hungarian). Unlike the random initialization of HC(Random), HC(Hungarian) is initialized with the best solution from HC (Random). This allows the rapid exploration afforded by a random encoding of vertices, followed by the focused exploration afforded through the use of reduced Hamming labels.

We would be remiss in not pointing out an alternative to the HC algorithm presented above. In this study, the HC algorithm operates on binary strings. However, one can easily construct variants of HC that operate directly on graphs (i.e., make moves directly in graph space). 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 through which to explore the search space.

We demonstrate the efficacy of this concretely in the problem of finding the ***Long-Shortest-Paths.*** Given a sparsely connected graph of N vertices, the goal is to select K points that have the longest 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. We compare 5 search variants:

1. **HC (R1,Greedy):** Run HC with a random mapping of bit-strings to vertices (R1). The best result from this is used to initialize stage 2, where HC is run with the mapping of bit-strings to vertices found by the Greedy approach.
2. **HC (R1,Hung):** Similar to above, except in stage 2, the mapping found by the Hungarian approach is used.
3. **HC (R1,R2):** Similar to above, except that in the second stage, a second random encoding is used. This tests whether the intelligent mapping is important or whether any new mapping can be employed.
4. **HC (R1,R1):** Baseline. Simply run the second HC with the *same* random encoding as the first run. This determines whether just the extra iterations were enough to yield improved results or whether the new encoding was needed.
5. **HC (Hung,Hung):** Hungarian mapping for both Stage 1 and Stage 2. Does the Hungarian mapping work well in the early stages of search or just for local optimization (once a region of high performance is already found)?

Throughout this section, a large number of empirical results will be presented. Because the problems are randomly generated, giving the final, 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, vice-versa, and whether the average performance, in terms of best evaluation found, were statistically different from each other (> 99% confidence using a paired two-tail t-test).

In Table 1, we begin with examining the question: is using a reduced-Hamming distance encoding beneficial in the early stages of search? In the first result row, we see that when compared to only using a single random assignment, HC (R1,R1), using the Hungarian assignment for both stages, HC(Hung,Hung), performs competitively. However, the results are quite different when Stage 1 used a random encoding and the neighborhood-preserving encodings were used only in Stage 2: HC(R1,Hung) & HC (R1,Greedy). Though both use random encodings for the initial phases of search, they both *outperformed* HC(Hung,Hung). As posited earlier, in the beginning of search, more exploration occurs with a random encoding. When neighborhood-preserving labels are used in Stage 2, when local optimizations are necessary, the benefits of neighborhood-preserving encodings are fully realized. Note, however, that these results are a function of the search algorithm, HC, and the single-bit flip operator that is employed. Other search algorithms, such as genetic algorithms, may not have this performance characteristic (Section 4).

Table 1. Long-shortest-Paths N=1024 Vertices. Testing the effects of reduced Hamming distance mappings in Stage 1 of HC search. Left: Random Graphs. Right: Cluster Graphs.

Baseline	**RANDOM GRAPHS** (480 total runs)			**CLUSTER GRAPHS** (480 total runs)		
	Baseline wins	HC (Hung,Hung) wins	Signif (p <0.01)	Baseline wins	HC (Hung,Hung) wins	Signif(p < 0.01)
HC(R1,R1)	155	325	NO	227	257	NO
HC(R1,R2)	208	272	YES	369	110	YES
HC(R1,Greedy)	283	197	YES	398	81	YES
HC(R1,Hung)	287	193	YES	394	85	YES

We also verify these results in a second setting. In the right table (labeled Cluster Graphs), we generate graphs with “clusters” of connections – randomly chosen nodes are grouped into clusters that are densely connected. Only sparse connections exist between clusters. All of the experiments are repeated with these graphs. The results further magnify the trend seen earlier – using the neighborhood-preserving labeling only in Stage 2 performs better than using it in both stages with HC. This trend was also noted in all of the other problems tested. To preserve space, we will no longer present this variant in the results. Instead, the neighborhood-preserving encodings will be used only in Stage 2, where local optimization is the objective.

3.2. Test Problem 1: Long-shortest Paths

Here we extend the experiments with the problem described in the previous section. The problem instances are parameterized with 3 variables, N: the number of vertices in the graph, C: the number of directed connections emanating from each node, and K: the number of points that need to be selected. For each problem instance created, 20 independent HC runs are performed with a random initialization (a uniformly random chosen bit-string). The HC algorithm is run until 1000 evaluations are evaluated without improvement (***Z*** parameter). The best solution found in Stage 1 is used to initialize the Stage 2 of HC.

In the next two tables, we compare the performances of 4 HC variants as the size of the graphs (N) and also the number points to be selected (K) are varied. Each cell in the table contains a triplet (#Wins-Algorithm-A / #Wins-Algorithm B, and whether the average end result found was statistically different to 99% confidence). Note that ties are not reported as a win for either algorithm. Column 1 shows that between the two stages of HC, replacing one random mapping with another outperforms keeping the same mapping. A new, randomly chosen vertex encoding helps escape local maxima. This is noteworthy, as this requires no external processing or intelligent mapping, and yet consistently improved results. Similar findings have been noted previously [9].

In Column 2, we compare HC (R1,Greedy) vs. HC (R1,R2); the mapping found with the Greedy matching tends to outperform in large graphs – in both Table 2 and Table 3. As seen in Column 5, the same is true when HC (R1,Hung) is compared with SP(R1,R2). When comparing Greedy vs. Hungarian, in Column 6, no clear trends are discernible. It is interesting to note that in a few trials, the Greedy encoding significantly outperformed the Hungarian encoding (marked with *). The Long Shortest Paths is the only problem, out of the 4 tested, for which we witnessed this. For completeness, we also present HC (R1,Hung) vs. HC (R1,R1); in almost every trial examined, in both this problem and in others, HC(R1, Hung) outperformed HC(R1,R1).

Table 2. Long Shortest-Paths, C=Connectivity 3. K = 20. Random Graph

	HC(R1,R2) vs. HC(R1,R1)	**HC (R1, Greedy) vs. HC (R1,R2)**	**HC (R1, Hung) Vs. HC (R1,R1)**	**HC (R1,Hung) Vs. HC (R1,R2)**	**HC(R1,Hung)Vs. HC(R1, Greedy)**
N=256	120/0 Yes	73/43 Yes	120/0 Yes	90/30 Yes	81/37 Yes
N=512	118/1 Yes	64/56 No	120/0 Yes	71/49 No	61/56 No
N=1024	118/1 Yes	81/38 Yes	119/1 Yes	82/36 Yes	51/69 Yes

Table 3. Long Shortest-Paths, N=1024 Vertices. C=Connectivity 3. Random Graph

	HC(R1,R2) vs. HC(R1,R1)	**HC (R1, Greedy) vs. HC (R1,R2)**	**HC (R1, Hung) Vs. HC (R1,R1)**	**HC (R1,Hung) Vs. HC (R1,R2)**	**HC(R1,Hung)Vs. HC(R1, Greedy)**
K=5	79/2 Yes	64/33 Yes	100/0 Yes	68/30 Yes	55/48 No
K=10	104/0 Yes	78/37 Yes	116/1 Yes	70/46 Yes	54/57 No
K=20	118/1 Yes	81/38 Yes	119/1 Yes	82/36 Yes	51/69 Yes *
K=100	120/0 Yes	75/45 Yes	120/0 Yes	87/33 Yes	64/56 No

Numerous additional experiments increasing the graph's connectivity were conducted. These revealed whether the labeling techniques were effective in highly constrained graphs in which the number of connections limited the number of good assignments. With both randomly graphs and graphs with clusters (see Section 3.1), we found (1) introducing a second random mapping usually improves performance, and (2) similar to the results presented here, using the Hungarian-mapping outperformed using random mappings.

3.3. 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.

For clarity, for the remainder of this section, we eliminate two columns from the result table. The first is HC(R1,R2) vs. HC (R1,R1); changing representations consistently provided better results in the vast majority of the problems explored. From this point on, we only compare our algorithms with the better of the two baselines, HC(R1,R2). Second, we eliminate the comparison HC (R1, Hung) vs. HC (R1,R1); HC (R1, Hung) always outperformed the standard technique and was always significantly ($p < 0.01$) better. It is should be noted that this is in itself a significant result and the basis of this paper – ***using the Hungarian-based mappings for local optimization always outperformed search using the standard, random, encodings.***

The result trends are even cleaner than those in Long-Shortest-Distance problems. The reduced Hamming distance encodings far outperform random encodings. Similar to the experiments in the previous section, when comparing Hungarian vs. Greedy encodings, however, there was no clear winner.

Table 4. (LEFT) Effects of Graph Size. C=Connectivity 3. K = 20, Random Graph
(RIGHT) Effects of K, N=1024 Vertices. Cluster Graph & Random Graph. All connectivities.

	HC (R1, Gr) vs.HC (R1,R2)	**HC (R1,Hung) Vs.HC (R1,R2)**	**HC(R1,Hung) Vs. HC(R1, Gr)**
N=256	94/15 Yes	98/13 Yes	62/45 No
N=512	87/29 Yes	100/18 Yes	81/37 Yes
N=1024	82/38 Yes	84/36 Yes	55/63 No

	HC(R1, Gr) vs. HC(R1,R2)	**HC (R1,Hung) Vs. HC(R1,R2)**	**HC(R1,Hung) Vs. HC(R1, Gr)**
K= 5	47/95 Yes	450/94 Yes	234/225 No
K= 10	463/126 Yes	492/98 Yes	318/261 Yes
K= 20	479/119 Yes	478/117 Yes	296/295 No
K=100	406/154 Yes	429/153 Yes	324/244 Yes

Numerous additional experiments were conducted to examine the effects of changing the connectivity of the Randomly Generated graphs and the number of clusters in the Cluster-Graphs. The results were consistent with those shown in the above two tables.

3.4. 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 objective 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 Sections 3.2 and 3.3, 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, nor are they weighted by similarity; these variations are open for future exploration. Once this graph is constructed, 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 5, varying levels of problem difficulty are examined. The total the knapsack can hold in any dimension determines how difficult the problem is. Three sizes of the knapsack are explored: ranging from 1/5th of the expected total in any dimension to 1/20th of the expected total– *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 1/20th the expected total, the knapsack is limited to a total 6400 in any dimension. In Table 5, (LEFT) each item has D=9 traits, (RIGHT) 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 5. Multi-Dimensional Knapsack N=1024, 512, & 256 Vertices. Three level of problem difficulty explored. Synthesized Graph modeled 10 Nearest Neighbors. (LEFT) D = 9 Traits (RIGHT) D = 1 Trait.

	HC (R1, Gr) Vs. HC(R1,R2)	HC(R1,Hung) Vs. HC (R1,R2)	HC(R1,Hung) Vs. HC(R1, Gr)	HC (R1, Gr) Vs. HC(R1,R2)	HC(R1,Hung) Vs.HC (R1,R2)	HC(R1,Hung) Vs. HC(R1, Gr)
Loose 1/5th	268/84 Yes	283/72 Yes	212/145 No	349/11 Yes	356/4 Yes	260/100 Yes
Medium 1/10th	289/65 Yes	294/62 Yes	208/150 Yes	294/66 Yes	300/60 Yes	198/162 Yes
Tight 1/20th	249/101 Yes	292/66 Yes	207/152 Yes	285/75 Yes	303/57 Yes	207/153 No

Using the neighborhood-preserving mappings, both Greedy and Hungarian, consistently improved the results. As can be seen by the number of wins, the effect was magnified when items only had a single (D=1) trait. With D=1, it was possible to select items with a better ratio of value/trait. In comparing the Hungarian vs. greedy labeling, there was a statistical difference in many trials, favoring the Hungarian labeling.

3.5. 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 multidimensional knapsack problem explored in Section 3.4, there is no explicit graph given in the problem. Instead, we construct the graph like the knapsack synthetic graphs – by finding the most similar, S=10, vectors. The results are shown in Table 6.

Table 6. Multi Dimensional Partitioning (LEFT) D=10 Dimensions, (RIGHT) D = 2 Dimensions.

	HC (R1, Gr) vs. HC (R1,R2)	HC (R1,Hung) Vs.HC (R1,R2)	HC(R1,Hung) Vs. HC(R1, Gr)	HC (R1, Gr) vs. HC (R1,R2)	HC (R1,Hung) Vs.HC (R1,R2)	HC(R1,Hung) Vs. HC(R1, Gr)
256 Vert	85/15 Yes	95/12 Yes	67/40 Yes	99/7 Yes	101/7 Yes	55/47 No
512 Vert	104/11 Yes	108/9 Yes	64/55 No	100/8 Yes	101/9 Yes	55/45 No
1024 Vert	107/4 Yes	111/4 Yes	65/50 No	103/3 Yes	105/3 Yes	57/43 Yes

Similar to the results shown with the Knapsack problems, incorporating either Hungarian or Greedy labeling greatly improves performance over random labeling. For this problem, across both settings of D, the performance between the Greedy and Hungarian labeling was not statistically different in most cases. Additionally, experiments with D=5 were conducted (not shown); the results were consistent with Table 6 – a large performance gain for the neighborhood-preserving labelings.

4. Genetic Algorithms – Empirical Result Summary

The entire set of experiments was replicated using a standard genetic algorithm [1]. In the pervious section, we used HC both for finding regions of good performance (Stage 1) and local optimization (Stage 2) once the regions were found. In a number of previous studies, it has been determined that GAs very quickly find regions of high performance, but incorporating hillclimbing methods often yields better results than simply continuing the GA for more iterations [7][8][18][19][20]. To make our experiments parallel to those presented here, we ran the GA until a local optima was found. Then, the best result found by the GA was used to initialize a HC run in a manner similar to the Stage 2 HC described in Section 3.

Due to space restrictions, we only summarize the experiments here [21]. We first explored whether, like with HC, the neighborhood-preserving encodings limit exploration in the early stages of search. We determined that, unlike HC, which explored the search space through single bit flips, the crossover and mutation operators of a GA were disruptive enough to the candidate solutions that the encoding did not restrict search; the GA was not constrained to move with only bit flips. Second, we explored whether these new encodings improved the performance of the GA when no supplementary second stage HC was used. GAs, though efficient at finding regions of high performance, are often less effective at "last-step" local optimizations due to population convergence among other reasons. However, it is precisely in these local optimization stages that these encodings are designed to do well. Using these encodings vs. not resulted in mixed performance.

The third and most important set of experiments was whether when a stage-2 local optimization step was employed, would the neighborhood-preserving encodings help, or would the addition of the stage-2 step, by itself, achieve equivalent results? Numerous variations were explored, including using the same encoding for the GA and Stage-2 HC, different random encodings, the Hungarian encoding for both, or the Hungarian encoding only for the Stage-2 HC. The results overwhelmingly pointed to the following: the best performing procedure used a stage 2 HC step with the neighborhood-preserving mappings – regardless of the encoding the GA used for the initial search. This solidifies the results in the previous sections: the neighborhood-preserving mappings improve the performance of local optimization across a variety of search techniques.

5. Conclusions and 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 [11][14]. The second is a large empirical demonstration of how these labels can be effectively used in stochastic search to explore local neighborhoods once regions of high performance are found. 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.

There are three immediate directions for research. (1) Further understanding the benefits of Hungarian vs. Greedy approaches: Was the performance difference limited because of the encodings or the search algorithms? (2) There is no limitation of the propagation algorithm to a binary alphabet. Higher cardinality alphabets should be tested. (3) The labeling algorithm can handle undirected and directed graphs and non-uniformly weighted edges. Using these features opens the possibility of addressing an even wider range of problems.

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. Patent and Trademark Office.

[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 GAs 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. 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. *Proceedings of the 8th GECCO*, 667-674

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

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

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

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

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

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

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

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

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

[17] Wattenberg, M., & Juels, A. (1996, June). Stochastic hillclimbing as a baseline method for evaluating genetic algorithms. *NIPS-8*

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

[19] García-Martínez C, Lozano M. Local search based on GAs. (2008) *Adv. in metaheuristics for hard optimization*. 199–221.

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

[21] Baluja, S., & Covell, M. Point Representation for Local Optimization, *in progress.*