Flexible Word Design and Graph Labeling*

Ming-Yang Kao, Manan Sanghi, and Robert Schweller

Department of Electrical Engineering and Computer Science
Northwestern University
Evanston, IL 60208, USA
{kao, manan, schwellerr}@cs.northwestern.edu

Abstract. Motivated by emerging applications for DNA code word design, we consider a generalization of the code word design problem in which an input graph is given which must be labeled with equal length binary strings of minimal length such that the Hamming distance is small between words of adjacent nodes and large between words of non-adjacent nodes. For general graphs we provide algorithms that bound the word length with respect to either the maximum degree of any vertex or the number of edges in either the input graph or its complement. We further provide multiple types of recursive, deterministic algorithms for trees and forests, and provide an improvement for forests that makes use of randomization.

1 Introduction

This work can be viewed either as a generalization of codeword design or a special restricted case of the more general graph labeling problem. The problem of graph labeling takes as input a graph and assigns a binary string to each vertex such that either adjacency or distance between two vertices can be quickly determined by simply comparing the two labels. The goal is then to make the labels as short as possible (see [14] for a survey). Early work in the field [8, 9] considered the graph labeling problem with the restriction that the adjacency between two nodes must be determined solely by Hamming distance. Specifically, the labels for any two adjacent nodes must be below a given threshold, while the nodes between non-adjacent nodes must be above it.

We return to this restricted type of graph labeling motivated by growing applications in DNA computing and DNA self-assembly which require the design of DNA codes that exhibit non-specific hybridization. A basic requirement for building useful DNA self-assembly systems and DNA computing systems is the design of sets of appropriate DNA strings (code words). Early applications have simply required building a set of n equal length code words such that there is no possibility of hybridization between the words or Watson Crick complement of words [1, 4, 6, 7, 18, 23]. Using hamming distance as an approximation to how well a word and the Watson Crick complement of a second word will bind,

* Supported in part by NSF Grant EIA-0112934.

T. Asano (Ed.): ISAAC 2006, LNCS 4288, pp. 48–60, 2006.

Table 1. Summary of our results

	Word Length	
	Lower Bound	Upper Bound
General Graphs (Matching Algorithm)	$\Omega(\gamma + n)$ Theorem 1	$O(\gamma\hat{D} + \hat{D}n)$ Theorem 3
General Graphs (StarDestroyer)		$O(\sqrt{\gamma^2\hat{m}n + \gamma\hat{m}n^2})$ Theorem 4
Forests (Randomized)	$\Omega(\gamma + \log n)$ Theorem 2	$O(\gamma D \log(\max\{f, \frac{n}{\gamma D}\}))$ Theorem 8

G : input graph (V, E)	m : number of edges in G
$\overline{G}$: complement of G	$\hat{m}$: smaller of the number of edges in G or $\overline{G}$
D : highest degree of any vertex in G	n : number of vertices in G
$\hat{D}$: smaller of the highest degree of any vertex in G or $\overline{G}$	f : maximum number of leaves in any tree in the input forest
γ : Hamming distance separation	

such a requirement can be achieved in part by designing a set of n words such that the Hamming distance between any pair in the set is large. There has been extensive work done in designing sets of words with this and other non-interaction constraints [5, 6, 10–13, 15–18, 21].

While the Hamming distance constraint is important for applications requiring that no pair of words in a code hybridize, new applications are emerging for which hybridization between different words in a code word set is desirable and necessary. That is, there is growing need for the efficient design of DNA codes such that the hybridization between any two words in the code is determined by an input matrix specifying which strands should bond and which should not. Aggarwal et al. [2, 3] have shown that a tile self assembly system that uses a set of glues that bind to one another according to a given input matrix, rather than only binding to themselves, greatly reduces the number of distinct tile types required to assemble certain shapes. Efficient algorithms for designing sets of DNA strands whose pairwise hybridization is determined by an input matrix may permit implementation of such tile efficient self-assembly systems.

Further, Tsaftaris et al. [19, 20] have recently proposed a technique for applying DNA computing to digital signal processing. Their scheme involves designing a set of equal length DNA strands, indexed from 1 to n, such that the melting temperature of the duplex formed by a word and the Watson Crick complement of another word is proportional to the difference of the indices of the words. Thus, this is again an example in which it is desirable to design a set of DNA words such that different words have varying levels of distinctness from one another.

Given an input graph G, we consider the problem of constructing a labeling such that the Hamming distance between labels of adjacent vertices is small, the Hamming distance between non-adjacent vertices is large, and there is a separation of at least γ between the small Hamming distance and the large Hamming distance. Breuer et al.[8, 9] first studied this problem for the special case of $\gamma = 1$ and achieved labels of size $O(Dn)$ for general graphs, where D is the degree of the node with the highest degree. By combining graph decompositions with codes similar in spirit to Hadamard codes from coding theory, we get a labeling of length $O(\gamma\hat{D} + \hat{D}n)$ where $\hat{D}$ is the smaller of the degree of the maximum degree node in G and its complement. We then explore more sophisticated graph decompositions to achieve new bounds that are a function of the number of edges in G. We also consider the class of trees and forests and provide various recursive algorithms that achieve poly-logarithmic length labels for degree bounded trees. Our forest algorithms also make use of probabilistic bounds from traditional word design to use randomization to reduce label length. Our results are summarized in Table 1.

Paper Layout: In Section 2, we introduce basic notation and tools and formulate the problem. In Section 3, we describe techniques for obtaining a restricted type of labeling for special graphs. In Section 4, we describe how to combine special graph labelings to obtain algorithms for labeling general graphs. In Section 5, we present recursive algorithms for labeling forests. In Section 6, we conclude with a discussion of future research directions. In the interest of space proofs and some algorithmic details are omitted in this version.

2 Preliminaries

Let $S = s_1 s_2 \dots s_\ell$ denote a length ℓ bit string with each $s_i \in \{0, 1\}$. For a bit s, the *complement* of s, denoted by s^c, is 0 if $s = 1$ and 1 if $s = 0$. For a bit string $S = s_1 s_2 \dots s_\ell$, the *complement* of S, denoted by S^c, is the string $s_1^c, s_2^c \dots s_\ell^c$. For two bit strings S and T, we denote the concatenation of S and T by $S \cdot T$.

For a graph $G = (V, E)$, a length ℓ labeling of G is a mapping $\sigma : V \to \{0, 1\}^\ell$. Let $\deg(G)$ denote the maximum degree of any vertex in G and let $\bar{G} = (V, \bar{E})$ denote the complement graph of G. A γ-*labeling* of a graph G is a labeling σ such that there exist integers α and β, $\beta - \alpha \geq \gamma$, such that for any $u, v \in V$ the Hamming distance $\mathrm{H}(\sigma(u), \sigma(v)) \leq \alpha$ if $(u, v) \in E$, and $\mathrm{H}(\sigma(u), \sigma(v)) \geq \beta$ if $(u, v) \notin E$. We are interested in designing γ-labelings for graphs which minimize the length of each label, $\mathrm{length}(\sigma)$.

Problem 1 (Flexible Word Design Problem).
Input: Graph G; integer γ
Output: A γ-labeling σ of G. Minimize $\ell = \mathrm{length}(\sigma)$.

Throughout this paper, for ease of exposition, we will assume the Hamming distance separator γ is a power of 2. For general graphs in Sections 3 and 4 we also assume the number of vertices n in the input graph is a power of 2. These assumptions can be trivially removed for these cases.

Theorem 1. *The required worst case label length for general n node graphs is $\Omega(\gamma + n)$.*

Theorem 2. *The required worst case label length for n node forests is $\Omega(\gamma + \log n)$.*

An important tool that we use repeatedly in our constructions is a variant of the Hadamard codes [22] from coding theory. The key property of this code is that it yields short words such that every pair of strings in the code has the exactly the same Hamming distance between them.

Hadamard Codes. We define two types of Hadamard codes using the Hadamard matrices. The size 2×2 Hadamard matrix is defined to be:

$$H_2 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

For n a power of 2 the size $n \times n$ Hadamard matrix H_n is recursively defined as:

$$H_n = \begin{bmatrix} H_{\frac{n}{2}} & H_{\frac{n}{2}} \\ H^c_{\frac{n}{2}} & H_{\frac{n}{2}} \end{bmatrix}$$

From the Hadamard matrices we define two codes. For n a power of 2 and γ a multiple of $\frac{n}{2}$, define the *balanced Hadamard* code $HR^B(n, \gamma)$ to be the set of words obtained by taking each row of H_n concatenated $\frac{2\gamma}{n}$ times. Similarly, define the *Hadamard* code $HR(n, \gamma)$ by taking each row of H_n, removing the last bit, and concatenating $\frac{2\gamma}{n}$ copies. The Hadamard and balanced Hadamard codes have the following properties.

Lemma 1. *Consider the codes $\mathrm{HR}(n, \gamma)$ and $\mathrm{HR}^B(n, \gamma)$ for n and γ powers of 2 and $2\gamma \geq n$. The following properties hold.*

1. *For any $S \in \mathrm{HR}(n, \gamma)$, $\mathrm{length}(S) = 2\gamma - \frac{2\gamma}{n}$.*
2. *For any non-equal $S_i, S_j \in \mathrm{HR}(n, \gamma)$ (or any $S_i, S_j \in \mathrm{HR}^B(n, \gamma)$), $H(S_i, S_j) = \gamma$.*
3. *For any non-equal $S_i, S_j \in \mathrm{HR}(n, \gamma)$, $H(S_i, S_j^c) = \gamma - \frac{2\gamma}{n}$.*
4. *For any $S \in \mathrm{HR}^B(n, \gamma)$, $\mathrm{length}(S) = 2\gamma$.*
5. *Let $F^B(n, \gamma) = \mathrm{HR}^B(n, \gamma) \setminus \{A_1, \ldots A_r\} \cup \{A_1^c, \ldots, A_r^c\}$ for an arbitrary subset $\{A_1, \ldots A_r\}$ of $\mathrm{HR}^B(n, \gamma)$. Then, properties 1 and 4 still hold for $F^B(n, \gamma)$.*
6. *The codes $\mathrm{HR}(n, \gamma)$ and $\mathrm{HR}^B(n, \gamma)$ can be computed in time $O(n \cdot \gamma)$.*

3 Exact Labelings for Special Graphs

In constructing a γ-labeling for general graphs, we make use of a more restrictive type of labeling called an *exact* labeling, as well as an *inverted* type of labeling. Such labelings can be combined for a collection of graphs to obtain labelings for

larger graphs. We consider two special types of graphs in this section, matchings and star graphs, and show how to obtain short exact labelings for each. These results are then applied in Section 4 by algorithms that decompose arbitrary graphs into these special subgraphs efficiently, produce exact labelings for the subgraphs, and then combine the labelings to get a labeling for the original graph.

Definition 1 (Exact Labeling). *A γ-labeling σ of a graph $G = (V, E)$ is said to be* exact *if there exist integers α and β, $\beta - \alpha \geq \gamma$, such that for any two nodes $u, v \in V$ it is the case that $\mathrm{H}(\sigma(u), \sigma(v)) = \alpha$ if $(u, v) \in E$, and $\mathrm{H}(\sigma(u), \sigma(v)) = \beta$ if $(u, v) \notin E$. A labeling that only satisfies $\mathrm{H}(\sigma(u), \sigma(v)) = \alpha$ if $(u, v) \in E$, but $\mathrm{H}(\sigma(u), \sigma(v)) \geq \beta$ if $(u, v) \notin E$ is called a lower exact labeling.*

Definition 2 (Inverse Exact Labeling). *A labeling σ of a graph $G = (V, E)$ is said to be an inverse exact labeling for value γ if there exist integers α and β, $\beta - \alpha \geq \gamma$, such that for any two nodes $u, v \in V$ it is the case that $\mathrm{H}(\sigma(u), \sigma(v)) = \alpha$ if $(u, v) \notin E$, and $\mathrm{H}(\sigma(u), \sigma(v)) = \beta$ if $(u, v) \in E$.*

Thus, the difference between an exact γ-labeling and a γ-labeling is that an exact labeling requires the Hamming distance between adjacent vertices to be exactly α, rather than at most α, and the distance between non-adjacent nodes to be exactly β, rather than at least β. An inverse exact labeling is like an exact labeling except that it yields a large Hamming distance between adjacent nodes, rather than a small Hamming distance.

We are interested in exact γ-labelings because the exact γ-labelings for a collection of graphs can be concatenated to obtain a γ-labeling for their union. We define an *edge decomposition* of graph $G = (V, E)$ into $G_1, \ldots, G_r$ where $G_i = (V_i, E_i)$ such that $V_i = V$ for all i and $E = \bigcup_i E_i$.

Lemma 2. *Consider a graph G with edge decomposition $G_1, \ldots G_r$. For each G_i let σ_i be a labeling of G_i with length $\mathrm{length}(\sigma_i) = \ell_i$. Consider the labeling $\sigma(v) = \sigma_1(v) \cdot \sigma_2(v) \cdots \sigma_r(v)$ defined by taking the concatenation of each of the labelings σ for each vertex in G. Then the following hold.*

1. *If each σ_i is an exact γ_i-labeling of G_i with thresholds α_i and β_i, then for $\gamma = \min\{\gamma_i\}$ the labeling $\sigma(v)$ is a γ-labeling of G with thresholds $\alpha = \sum \beta_i - \gamma$ and $\beta = \sum \beta_i$.*
2. *If each σ_i is an inverse exact γ_i-labeling of G_i with thresholds α_i and β_i, then for $\gamma = \min\{\gamma_i\}$ the labeling $\sigma(v)$ is a γ-labeling of the complement graph $\overline{G}$ with thresholds $\alpha = \sum_{i=1}^{r} \alpha_i$ and $\beta = \sum_{i=1}^{r} \alpha_i + \gamma$.*

We now discuss how to obtain exact and inverse exact labelings for special classes of graphs. For the classes of graphs we consider, it is surprising that we are able to achieve the same asymptotic label lengths for exact labelings as for inverse exact labelings. In Section 4 we discuss algorithms that decompose general graphs into these classes of graphs, obtain exact or inverse exact labelings, and then combine them to obtain a γ-labeling from Lemma 2.

3.1 Matchings

A graph $G = (V, E)$ is said to be a matching if each connected component contains at most two nodes. To obtain an exact labeling for a matching we use Algorithm 1 MatchingExact. To obtain an exact inverse matching, there exists an algorithm InverseMatchingExact (details omitted).

Algorithm 1. MatchingExact(G, γ)

1. Let $\gamma' = max(\gamma, \frac{n}{2})$. Generate HR$(n, \gamma')$.
2. Assign a distinct string from HR(n, γ') to each clique of G. That is, apply the labeling σ such that for each $v \in V$, $\sigma(v) \in$ HR(n, γ') and $\sigma(v) = \sigma(u)$ iff $(v, u) \in E$.
3. **Output** σ.

Lemma 3. *Algorithm 1 MatchingExact(G, γ) obtains an exact γ-labeling with $\alpha = 0$, $\beta = max(\gamma, \frac{n}{2})$, and length $O(\gamma + n)$, in run time $O(\gamma n + n^2)$.*

Lemma 4. *Algorithm InverseMatchingExact(G, γ) obtains an exact inverse labeling with $\alpha = \max(\gamma, \frac{n}{2})$, $\beta = 2 \cdot max(\gamma, \frac{n}{2})$, and length $O(\gamma + n)$, in run time $O(\gamma n + n^2)$.*

3.2 Star-Graphs

A graph is a *star graph* if there exists a vertex c such that all edges in the graph are incident to c. For such a graph, let A be the set of all vertices that are not adjacent to c and let B be the set of vertices that are adjacent to c. Algorithm 2 StarExact obtains an exact γ-labeling for a star graph G. (In fact, it achieves an exact 2γ-labeling). Figure 1 provides an example of the labeling assigned by StarExact. To obtain an exact inverse γ-labeling there exists an algorithm InverseStarExact (details omitted).

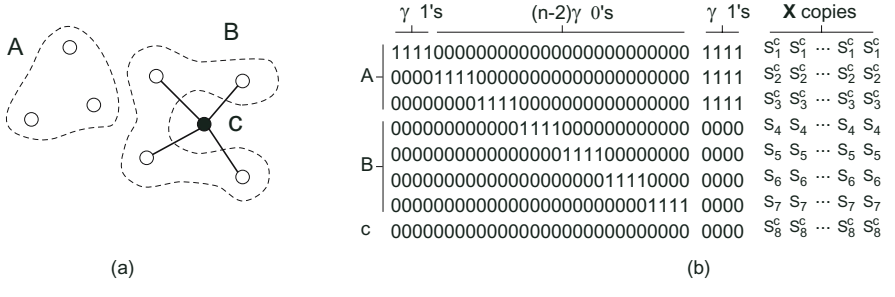

Fig. 1. (a) A star graph and (b) its corresponding exact labeling

Algorithm 2. StarExact(G, γ)

1. Let $\gamma' = max(\gamma, \frac{n}{2})$. Let $x = \min(\gamma, \frac{n}{2})$. Arbitrarily index the n vertices of V as $v_1, v_2, \ldots, v_n$ with $c = v_n$.
2. Set the first $(n-1)\gamma$ bits of $\sigma(c)$ to be 0's.
3. **For** each vertex $v_i \neq c$, set the first $(n-1)\gamma$ bits to be all 0's except for the i^{th} size γ word which is set to all 1's.
4. Append γ 1's to $\sigma(a)$ for each $a \in A$ and γ 0's to $\sigma(b)$ and $\sigma(c)$ for each $b \in B$.
5. **For** each $v_i \in A$ or $v_i = c$ append x copies of S_i^c to $\sigma(v_i)$ where S_i is the i^{th} string in $\mathrm{HR}(\gamma', n)$.
6. **For** each $v_i \in B$ append x copies of $S_i \in \mathrm{HR}(\gamma', n)$ to $\sigma(v_i)$.
7. **Output** σ.

Lemma 5. *Algorithm 2 StarExact(G, γ) obtains an exact γ-labeling for a Star graph G with $\alpha = \frac{\gamma n}{2}$, $\beta = 2\gamma + \frac{\gamma n}{2}$ and length $O(\gamma n)$, in run time $O(\gamma n^2)$.*

Lemma 6. *Algorithm InverseStarExact(G,γ) obtains an exact inverseγ-labeling for a Star graph G with $\alpha = \frac{\gamma n}{2}$, $\beta = 2\gamma + \frac{\gamma n}{2}$ and length $O(\gamma n)$, in run time $O(\gamma n^2)$.*

4 Labeling General Graphs

To obtain a γ-labeling for a general graph, we decompose either the graph or its complement into a collection of star and matching subgraphs. We then apply Lemmas 3 and 5 or Lemmas 4 and 6 to obtain exact or exact inverse labelings for these subgraphs, and then apply Lemma 2 to obtain a γ-labeling for the original graph. We first consider an algorithm that decomposes a general graph G into a collection of matchings.

4.1 Matching Decomposition

Lemma 7. *An edge decomposition of a graph $G = (V, E)$ into maximal matchings contains $\Theta(D)$ graphs where D is the maximum degree of any vertex in G.*

By breaking a given graph G into $\Theta(D)$ matchings and applying Lemmas 2 and 3, we have the algorithm MatchingDecomposition(G, γ) which yields a γ-labeling σ of G with length$(\sigma) = O(D{\cdot}(\gamma + n))$. For dense graphs whose vertices are all of high degree, MatchingDecomposition(G, γ) can be modified to decompose the complement graph $\overline{G}$ into maximal matchings and apply the routine InverseMatchingExact to obtain a length bound of $O(\overline{D} \cdot (\gamma + n))$ where $\overline{D}$ is the maximum degree of any vertex in $\overline{G}$. We thus get the following result.

Theorem 3. *For any graph G and γ, there exists a γ-labeling σ of G with* length$(\sigma) = O(\hat{D}\gamma + \hat{D}n)$ *that can be computed in time complexity $O(\gamma\hat{D}n + \hat{D}n^2)$ where $\hat{D}$ is the smaller between the degree of the maximum degree vertex in G and the maximum degree vertex in $\overline{G}$.*

4.2 Hybrid Decomposition (Star Destroyer)

The next algorithm for obtaining a γ-labeling adds the star decomposition to the basic matching algorithm. From Theorem 3, the matching algorithm may perform poorly even if there are just a few very high and very low degree vertices in the graph. The StarDestroyer(G, γ) algorithm thus repeatedly applies the star decomposition until all nodes have degree at most $\sqrt{m\gamma n}/\sqrt{\gamma + n}$, and then applies a final matching decomposition. With a few additional modifications we achieve the following.

Theorem 4. *For any graph G and γ, Algorithm StarDestroyer(G, γ) yields a γ-labeling σ of G with* $\text{length}(\sigma) = O(\sqrt{\gamma^2 \hat{m} n + \gamma \hat{m} n^2})$ *in time complexity* $O(\sqrt{\gamma^2 \hat{m} n^3 + \gamma \hat{m} n^4})$ *where* $\hat{m} = min\{|E|, |\bar{E}|\}$.

5 Trees and Forests

In this section we consider input graphs that are trees or forests and show that we are able to obtain substantially smaller labelings than what is possible for general graphs. For a collection of trees with a special type of γ-labeling, we show how to combine the collection into a single special γ-labeled tree. Thus, using recursive separators for trees we provide a recursive algorithm for tree labeling that achieves a length of $O(\gamma D \log n)$ where D is the largest degree node in the tree.

We then show how to improve this bound with a more sophisticated algorithm that assigns labels efficiently to paths as a base case, and recurses on the number of leaves in the tree rather than the number of nodes to achieve a length of $O(\gamma D \log(\max\{f, \frac{n}{\gamma D}\}))$ where f is the number of leaves in the tree. Note that this second bound is always at least as good as the first, and for trees with few leaves but high γ, is better. For example, consider the class of graphs consisting of $\log n'$ length $\frac{n'}{\log n'}$ paths, each connected on one end to a single node v. The number of nodes in this graph is $n = n' + 1$, the highest degree node has degree $D = \log n'$, and the number of leaves is $f = \log n'$. For $\gamma = \frac{n}{\log n'}$, the first bound yields $\ell = O(n \log n)$ while the second yields $\ell = O(n \log \log n)$.

5.1 Combining Trees

To make our recursive algorithms work, we need a way to take labelings from different trees and efficiently create a labeling for the tree resulting from combining the smaller trees into one. To do this, we will make use of a special type of γ-labeling.

Definition 3 (Lower Bounded Labeling). *A γ-labeling σ is said to be a lower bounded γ-labeling with respect to α_a, α_b, and β, $\alpha_a \leq \alpha_b < \beta$, $\beta - \alpha_b \geq \gamma$ if for any two nodes v and u, $\alpha_a \leq \text{H}(\sigma(v), \sigma(u)) \leq \alpha_b$ if v and u are adjacent, and $\text{H}(\sigma(v), \sigma(u)) \geq \beta$ if they are not adjacent.*

Given a collection of lower bounded labelings for trees, we can combine the labelings into a new lower bounded labeling with the same parameters according to Lemma 8. For the rest of this section, we will be dealing with a parameter D' which will be an upper bound on the maximum degree value of the input graph such that $D'+1$ is a power of 2 greater than 2.

Algorithm 3. CombineTrees$(T=(V,E), v, \{\sigma_i\}_{i=1}^{t})$

Input:

1. A degree t vertex v in tree T with $t \leq D'$.
2. An $\alpha_a = \gamma$, $\alpha_b = \frac{\gamma(D'-1)}{2}$, $\beta = \frac{\gamma(D'+1)}{2}$ lower bounded γ-labeling σ_i for each subtree of v.

Output: An $\alpha_a = \gamma$, $\alpha_b = \frac{\gamma(D'-1)}{2}$, $\beta = \frac{\gamma(D'+1)}{2}$ lower bounded γ-labeling of T.

1. **For** each labeling σ_i, append 0's such that length$(\sigma_i) = max_{i=1\ldots t}\{\text{length}(\sigma_i)\}$.
2. **For** each of the child trees $T_1, \ldots T_t$ of v, **do**
   (a) Let v_i be the vertex in T_i adjacent to v and let $v_{i,j}$ denote the value of the j^{th} character of $\sigma_i(v_i)$. For each $u \in T_i, u \neq v_i$, invert the j^{th} character of $\sigma_i(u)$ if $v_{i,j} = 1$.
   (b) Set $\sigma_i(v_i)$ to all 0's.
3. Let $\sigma(v)$ be $max_{i=1\ldots t}\{\text{length}(\sigma_i)\}$ 0's concatenated with $S^c_{t+1} \in \text{HR}(D'+1, \frac{\gamma(D'+1)}{2})$. Let $\sigma(u) = \sigma_i(u)$ for each $u \in T_i$.
4. **For** $i = 1$ to t
   (a) **For** each $u \in T_i$, $\sigma(u) \leftarrow \sigma(u) \cdot S_i$ for $S_i \in \text{HR}(D'+1, \frac{\gamma(D'+1)}{2})$.
5. **Output** σ.

Lemma 8 (Combining Trees). *Consider a vertex v in a tree T of degree t. Suppose for each of the t subtrees of v we have a corresponding length at most ℓ lower bounded γ-labeling σ_i with $\beta = \frac{\gamma(D'+1)}{2}$, $\alpha_b = \beta - \gamma$, and $\alpha_a = \gamma$ for some $D' \geq \max\{t, 2\}$, $D'+1$ a power of 2. Then, Algorithm 3 CombineTrees$(T, v, \{\sigma_i\}_{i=1}^{t})$ computes a lower bounded γ-labeling with the same α_a, α_b, and β values and length $\ell' \leq \ell + \gamma D'$.*

5.2 Node Based Recursion

Define a *node separator* for a graph to be a node such that its removal leaves the largest sized connected component with at most $\lceil \frac{n}{2} \rceil$ vertices. Given Lemma 8 and the well known fact that every tree has a node separator, we are able to label a tree by first finding a separator, then recursively labeling the separated subtrees using lower bounded labeling parameters $\alpha_a = \gamma$, $\alpha_b = \frac{\gamma(D'-1)}{2}$, and $\beta = \frac{\gamma(D'+1)}{2}$ for $D' = O(D)$. Since it is trivial to obtain a lower bounded labeling satisfying such α_a, α_b, and β for a constant sized base case tree, we can obtain a $O(\gamma D \log n)$ bound on length of labelings for trees.

We can then extend this to a t tree forest by creating t length $\frac{\gamma(D'+1)}{2} \log t$ length strings such that each pair of strings has Hamming distance at least

$\frac{\gamma(D'+1)}{2}$ and appending a distinct string to the nodes of each forest. This yields the following result.

Theorem 5 (Node Recursive Forest). *Given an integer γ and a forest F with maximum degree D, a γ-labeling of F with length $O(\gamma D \log n)$ can be constructed in time $O(n\gamma D \log^2 n)$.*

5.3 Leaf Based Recursion

Instead of performing recursion by halving the number of nodes in the graph, we can instead halve the number of leaves in the graph and use an efficient path labeling algorithm to solve the base case. We first describe the efficient path labeling scheme.

Path Labeling. As a base case for our recursive algorithm, according to Lemma 8, we want to be able to produce a short lower bounded γ-labeling for a path graph with $\beta = \frac{\gamma(D'+1)}{2}$, $\alpha_b = \frac{\gamma(D'-1)}{2}$, and $\alpha_a \geq \gamma$ for any given D'. When called from the tree algorithm, D' will be on the order of the maximum degree of any node in the input tree. The Path algorithm will achieve $\alpha_a = \frac{\beta}{2} \geq \gamma$ to satisfy the desired constraints. The reason for this choice of α_a is that it is a power of 2, which is necessary for our algorithm. The basic structure of the Path algorithm is that it uses recursion based on node separators and Lemma 8 until the path is sufficiently short. Then, a labeling based on the Hadamard code is used. Recursive Algorithm 4 Path achieves the following result.

Algorithm 4. Path($P = \langle v_1, \ldots v_n \rangle, \gamma, D'$)

1. **If** $n \leq 2\gamma(D'+1) - 1$ **then**
   (a) Compute $S_1, \ldots S_{\gamma(D'+1)} \in \mathrm{HR}(\gamma(D'+1), \frac{\gamma(D'+1)}{4})$.
   (b) **For** $i = 1$ to $\gamma(D'+1) - 1$ **do**
      i. $\sigma(v_{2i-1}) \leftarrow S_i.S_i$
      ii. $\sigma(v_{2i}) \leftarrow S_i.S_{i+1}$
   (c) $\sigma(v_{2\gamma(D'+1)-1}) \leftarrow S_{\gamma(D'+1)}.S_{\gamma(D'+1)}$
   (d) **Output** σ.
2. **Else**
   (a) Let $P_1 = \langle v_1, \ldots, v_{\frac{n}{2}-1} \rangle$, $P_2 = \langle v_{\frac{n}{2}+1} \ldots, v_n \rangle$.
   (b) $\sigma_1 \leftarrow$Path(P_1, γ, D'), $\sigma_2 \leftarrow$Path(P_2, γ, D').
   (c) **Output** CombineTrees($P, v_{\frac{n}{2}}, \{\sigma_1, \sigma_2\}$).

Lemma 9. *For $D' \geq 3$, D'+1 a power of 2, and path P, Algorithm 4 Path(P,γ,D') generates a lower bounded γ-labeling σ of P with $\alpha_a = \frac{\gamma(D'+1)}{4}$, $\alpha_b = \frac{\gamma(D'-1)}{2}$, $\beta = \frac{\gamma(D'+1)}{2}$, and length($\sigma$) $= O(\max\{\gamma D' \log(\frac{n}{\gamma D'}), \gamma D'\})$ in time $O(n \cdot (\max\{\gamma D' \log^2(\frac{n}{\gamma D'}), \gamma D'\}))$.*

Leaf Recursive Tree Algorithm. The leaf recursive tree algorithm recursively reduces the number of leaves in the tree until the input is a simple path, for which

Algorithm Path can be used. For a tree T with f leaves, a *leaf* separator is a node such that its removal reduces the largest number of leaves in any of the remaining connected components to at most $\lfloor \frac{f}{2} \rfloor + 1$. We start by observing that every tree must have a leaf separator.

Lemma 10. *Every tree has a leaf separator.*

Note that a leaf separator always reduces the number of leaves in a tree unless there are only 2 leaves, in which case the tree is a path which can be handled according to Lemma 9. Having removed a leaf separator and recursively solved for the sub trees, we can then apply Lemma 8 to finish the labeling. The details of the algorithm are given as follows. Here, the input parameter D' is the smallest integer such that $D' + 1$ is a power of 2 and D' is at least the degree of the highest degree node in the tree, or 3 in the case of an input path.

Algorithm 5. Tree$(T = (V, E), \gamma, D')$

1. **If** $\text{Deg}(T) \leq 2$, **then output** $\text{Path}(T, \gamma, D')$.
2. **Else**
   (a) Find a leaf separator v for T.
   (b) **For** each of the child trees $T_1, \ldots T_t$ of v, $\sigma_i \leftarrow \text{Tree}(T_i, \gamma, D')$.
   (c) **Output** $\text{CombineTrees}(T, v, \{\sigma_i\}_{i=1}^{t})$.

Theorem 6 (Trees). *Consider a tree T with f leaves and integer $D' = 2^j - 1 \geq \max\{\deg(T), 3\}$. Then, Algorithm 5 Tree$(T, \gamma, D')$ computes a length $O(\gamma D' \log(\max\{f, \frac{n}{\gamma D'}\}))$ γ-labeling of T in time complexity $O(n(\gamma D' \log^2(\max\{f, \frac{n}{\gamma D'}\})))$.*

To extend this result to a forest of trees $T_1 \cdots T_t$, we can use $\text{Tree}(T_i, \gamma, D')$ for each individual tree. We can then append a distinct string from a set of t strings to each tree such that the distance between any two strings is at least $\beta = \frac{\gamma(D'+1)}{2}$. Deterministically we can achieve such a set of strings trivially using additional length $O(\gamma D \log t)$ where $D = \deg(T)$. Alternately, we can use elements of $\text{HR}(t, \beta)$ for an additional length of $O(t + \gamma D)$. These approaches yield the the following theorem.

Theorem 7 (Leaf Recursive Forest). *There exists a deterministic algorithm that produces a length $O(\min\{t, \gamma D \log t\} + \gamma D \log(\max\{f, \frac{n}{\gamma D}\}))$ γ-labeling for an input forest F in time complexity $O(n(\min\{t, \gamma D \log t\} \gamma D \log^2(\max\{f, \frac{n}{\gamma D}\})))$, where $D = \deg(F)$, f is the largest number of leaves in any of the trees in F, and t is the number of trees in F.*

Alternately, we can use randomization to append shorter strings to each tree and avoid an increase in complexity. Kao et al.[15] showed that with high probability, a set of n uniformly generated random binary strings has Hamming distance at least x between any two words with high probability for words of length at least $10(x + \log n)$. Thus, we can produce a γ-labeling for a forest by first finding a

labeling for each tree, making the length of the labels equal, and finally picking a random string of length $10(\beta + \log n)$ for each tree and appending the string to each of the nodes in the tree. We thus get the following result.

Theorem 8 (Randomized Forest). *There exists a randomized algorithm that produces a length $O(\gamma D \log(\max\{f, \frac{n}{\gamma D}\}))$ γ-labeling for an input forest F with probability at least $1 - \frac{1}{n+2^\gamma}$, in time complexity $O(n \cdot (\gamma D \log(\max\{f, \frac{n}{\gamma D}\})))$, where $D = \deg(F)$, and f is the largest number of leaves in any of the trees in F.*

6 Future Directions

There are a number of potential research directions stemming from this work. A few of these are as follows. First, can our technique for labeling general graphs by decomposing the graph into exact labelings be extended? We considered two different types of decompositions, stars and matchings. Are there other types of decompositions that can yield better bounds? Second, our lower bounds are straightforward and stem primarily from lower bounds for labeling for adjacency in general, rather than our much more restricted problem. It is likely that much higher bounds exist for flexible word design. Third, an important class of graphs that permits short labels for general graph labeling is the class of planar graphs. It would be interesting to know whether or not a flexible word labeling that is sublinear in the number of vertices exists as well. Fourth, we have initiated the use of randomization in designing labels. Randomization is used extensively in the design of standard DNA code word sets, and it would be interesting to know if more sophisticated randomized algorithms can be applied to achieve better flexible word labelings. Finally, although not included in this draft, we have also considered generalizations of flexible word design to both distance labelings and weighted graphs. These generalizations present many open problems and may have direct applications to applying DNA computing to digital signal processing.

References

[1] L. M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, 1994.

[2] G. Aggarwal, Q. Cheng, M. H. Goldwasser, M.-Y. Kao, P. M. de Espanes, and R. T. Schweller. Complexities for generalized models of self-assembly. *SIAM Journal on Computing*, 34:1493–1515, 2005.

[3] G. Aggarwal, M. H. Goldwasser, M.-Y. Kao, and R. T. Schweller. Complexities for generalized models of self-assembly. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 880–889, 2004.

[4] A. Ben-Dor, R. Karp, B. Schwikowski, and Z. Yakhini. Universal DNA Tag Systems: A Combinatorial Design Scheme. In *Proceedings of the 4^{th} Annual International Conference on Computational Molecular Biology*, pages 65–75, 2000.

[5] A. Brenneman and A. E. Condon. Strand Design for Bio-Molecular Computation. *Theoretical Computer Science*, 287(1):39–58, 2001.

[6] S. Brenner. Methods for sorting polynucleotides using oligonucleotide tags, Feb. 1997. U.S. Patent Number 5,604,097.
[7] S. Brenner and R. A. Lerner. Encoded combinatorial chemistry. In *Proceedings of the National Academy of Sciences of the U.S.A.*, volume 89, pages 5381–5383, June 1992.
[8] M. Breuer. Coding vertexes of a graph. *IEEE transactions on Information Theory*, 8:148–153, 1966.
[9] M. Breuer and J. Folkman. An unexpected result on coding vertices of a graph. *Journal of Mathematical Analysis and Applications*, 20:583–600, 1967.
[10] R. Deaton, M. Garzon, R. C. Murphy, J. A. Rose, D. R. Franceschetti, and J. S. E. Stevens. Genetic search of reliable encodings for DNA-based computation. In *Proceedings of the 2nd International Meeting on DNA Based Computers*, 1996.
[11] A. G. Frutos, Q. Liu, A. J. Thiel, A. M. W. Sanner, A. E. Condon, L. M. Smith, and R. M. Corn. Demonstration of a word design strategy for DNA computing on surfaces. *Nucleic Acids Research*, 25(23):4748–4757, Dec. 1997.
[12] P. Gaborit and O. D. King. Linear constructions for DNA codes. *Theoretical Computer Science*, 334:99–113, 2005.
[13] M. Garzon, R. Deaton, P. Neathery, D. R. Franceschetti, and R. C. Murphy. A new metric for DNA computing. In *Proceedings of the 2nd Genetic Programming Conference*, pages 472–478. Morgan Kaufman, 1997.
[14] C. Gavoille and D. Peleg. Compact and localized distributed data structures. Technical Report RR-1261-01, Laboratoire Bordelais de Recherce en Informatique, 2001.
[15] M. Y. Kao, M. Sanghi, and R. Schweller. Randomized fast design of short dna words. In *Lecture Notes in Computer Science 3580: Proceedings of the 32nd International Colloquium on Automata, Languages, and Programming*, pages 1275–1286, 2005.
[16] O. D. King. Bounds for DNA Codes with Constant GC-content. *Electronic Journal of Combinatorics*, 10(1):#R33 13pp, 2003.
[17] A. Marathe, A. Condon, and R. M. Corn. On Combinatorial DNA Word Design. *Journal of Computational Biology*, 8(3):201–219, 2001.
[18] D. D. Shoemaker, D. A. Lashkari, D. Morris, M. Mittman, and R. W. Davis. Quantitative phenotypic analysis of yeast deletion mutants using a highly parallel molecular bar-coding strategy. *Nature Genetics*, 14(4):450–456, Dec. 1996.
[19] S. A. Tsaftaris. DNA Computing from a Signal Processing Viewpoint. *IEEE Signal Processing Magazine*, 21:100–106, September 2004.
[20] S. A. Tsaftaris. How can DNA-Computing be Applied in Digital Signal Processing? *IEEE Signal Processing Magazine*, 21:57–61, November 2004.
[21] D. C. Tulpan and H. H. Hoos. Hybrid Randomised Neighbourhoods Improve Stochastic Local Search for DNA Code Design. In Y. Xiang and B. Chaib-draa, editors, *Lecture Notes in Computer Science 2671: Proceedings of the 16th Conference of the Canadian Society for Computational Studies of Intelligence*, pages 418–433. Springer-Verlag, New York, NY, 2003.
[22] J. van Lint. *Introduction to Coding Theory.* Springer, third edition, 1998.
[23] E. Winfree, F. Liu, L. Wenzler, and N. Seeman. Design and self-assembly of two-dimensional DNA crystals. *Nature*, 394:539–544, August 1998.