Flexible Word Design and Graph Labeling*

Ming-Yang Kao † Manan Sanghi ‡ Robert Schweller §

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. We also consider generalizations of this problem to weighted graphs and graphs with optional edges. Finally, we explore the extension from simple adjacency queries to more general distance queries and show how to obtain distance labelings for rings and paths by applying properties of hypercube traversal.

*Supported in part by NSF Grant EIA-0112934.

†Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60208, USA. Email: kao@cs.northwestern.edu.

‡Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60208, USA. Email: manan@cs.northwestern.edu.

§Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60208, USA. Email: schwellerr@cs.northwestern.edu.

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. 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, 19, 24]. Using hamming distance as an approximation to how well a word and the Watson Crick complement of a second word will bind, 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, 16–19, 22, 23].

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. [20, 21] 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 size of the highest degree node in the graph. 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 size 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

Table 1: Summary of our results for word length and time complexity for flexible word design.

	Word Length		Time Complexity
	Lower Bound	Upper Bound	
General Graphs (Matching Algorithm)	$\Omega(\gamma + n)$ Theorem 2	$O(\gamma\hat{D} + \hat{D}n)$ Theorem 14	$O(\gamma\hat{D}n + \hat{D}n^2)$ Theorem 14
General Graphs (StarDestroyer)		$O(\sqrt{\gamma\hat{m}^2n + \gamma\hat{m}n^2})$ Theorem 15	$O(\sqrt{\gamma\hat{m}^2n^3 + \gamma\hat{m}n^4})$ Theorem 15
Weighted Graphs (Multi-graph based)		$O\left(\min\left\{\begin{array}{l}\Delta\gamma + \Delta n,\\ \sqrt{\gamma W^2 n + \gamma W n^2}\end{array}\right\}\right)$ Theorem 31	$O\left(\min\left\{\begin{array}{l}\Delta\gamma n + \Delta n^2,\\ \sqrt{\gamma W^2 n^3 + \gamma W n^4}\end{array}\right\}\right)$ Theorem 31
Weighted Graphs (Weighted Matching)		$O\left(\min\left\{\begin{array}{l}\sum_i w_i d_i\gamma + \sum_i d_i n,\\ Dw_\rho\gamma + D\rho n\end{array}\right\}\right)$ Theorems 33 and 34	$O\left(\min\left\{\begin{array}{l}n\cdot(\sum_i w_i d_i\gamma + \sum_i d_i n),\\ n\cdot(Dw_\rho\gamma + D\rho n)\end{array}\right\}\right)$ Theorems 33 and 34
Weighted Graphs (Hybrid Algorithm)		$O(\sqrt[3]{W^2\gamma n(\gamma+n)(\pi\gamma+n)})$ Theorem 35	$O(n\cdot\sqrt[3]{W^2\gamma n(\gamma+n)(\pi\gamma+n)})$ Theorem 35
Don't care edges		$O\left(\min\left\{\begin{array}{l}\gamma\tilde{D} + \tilde{D}n,\\ \sqrt{\gamma\tilde{m}^2n + \gamma\tilde{m}n^2}\end{array}\right\}\right)$ Theorem 37	$O\left(\min\left\{\begin{array}{l}\gamma\tilde{D}n + \tilde{D}n^2,\\ n\cdot\sqrt{\gamma\tilde{m}^2n + \gamma\tilde{m}n^2}\end{array}\right\}\right)$ Theorem 37
Forests (Node Recursion)	$\Omega(\gamma + \log n)$ Theorem 3	$O(\gamma D\log n)$ Theorem 18	$O(n\gamma D\log^2 n)$ Theorem 18
Forests (Leaf Recursion)		$O\left(\begin{array}{l}min\{t, \gamma D\log t\} + \gamma D\log f\\ +\max\{\gamma D\log(\frac{n}{\gamma D}), 0\}\end{array}\right)$ Theorem 22	$O\left(\begin{array}{l}n\cdot min\{t, \gamma D\log t\} + n\cdot\gamma D\log^2 f\\ +n\cdot\max\{\gamma D\log^2(\frac{n}{\gamma D}), 0\}\end{array}\right)$ Theorem 22
Forests (Randomized)		$O(\gamma D\log f + \max\{\gamma D\log(\frac{n}{\gamma D}), 0\})$ Theorem 23	$O(n\gamma D\log^2 f + \max\{\gamma D\log^2(\frac{n}{\gamma D}), 0\})$ Theorem 23
Rings	Theorem 25	$O(\gamma\log n)$ Theorem 29	$O(n\gamma\log n)$ Theorem 29
Special Rings with Distance Queries	$\Omega(\gamma k + \log n)$	$O(\gamma k\log\frac{n}{k})$, where $k \leq \frac{n}{2} - 1$ Theorem 27	$O(n\gamma k\log\frac{n}{k})$ Theorem 27
Paths with Distance Queries	Theorem 25	$O(\gamma k + \gamma k\log\frac{n}{k})$ Theorem 28	$O(n\cdot(\gamma k + \gamma k\log\frac{n}{k}))$ Theorem 28

G : input graph (V, E)	D : highest degree of any vertex in G
$\overline{G}$: complement of G	$\hat{D}$: smaller of the highest degree of any vertex in G or $\overline{G}$
γ : Hamming distance separation	$\tilde{D}$: smaller of maximum near-degree and maximum far-degree
n : number of vertices in G	Δ : maximum weighted degree
m : number of edges in G	t : number of trees in the forest
$\hat{m}$: smaller of the number of edges in G or $\overline{G}$	f : maximum number of leaves in any tree in the input forest
$\tilde{m}$: smaller of the number of near edges and the number of far edges	k : range of distance queries
W : the sum of weights of all the edges	π : maximum weight of an edge
d_i : highest degree of any vertex with respect to weight w_i edges	ρ : number of distinct weights of edges

length.

We further consider some important generalizations to the basic flexible word design problem. First, we initiate the study of extending our problem from adjacency queries to distance queries. For this we apply basic properties of hypercube traversal to provide efficient labelings for rings and paths and discuss how this problem and class of graphs may have applications to DNA signal processing. We also consider a version of flexible word design for weighted graphs and provide multiple general case algorithms for the problem. Finally, we discuss graphs that have *don't care* edges which make no requirement for the Hamming distance between two nodes. 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 generalize our problem to distance labeling and provide solutions for rings and paths. In Section 7, we provide algorithms for solving the weighted version of flexible word design. In Section 8, we present results for a generalization to graphs with *don't care* edges. In Section 9, we conclude with a discussion of future research directions.

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, 4, 7, and 8 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. For the remaining sections, in particular Section 6 which deals with distance queries on rings and paths, we do not make this assumption as the extension to all n is non-trivial.

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

Proof. For graph labeling supporting adjacency queries, it is well known [15] that a class of $2^{\Omega(n^{1+\epsilon})}$ n-vertex graphs must contain a member that requires labels of size $\Omega(n^\epsilon)$. As there are $2^{\Omega(n^2)}$ n node graphs, we get a lower bound of $\Omega(n)$ on label length. Specific to our problem, we further observe that any graph other than the complete graph must have Hamming distance at least γ for some pair of nodes, which then requires length at least γ. Thus, there must exist some n node graph requiring label length at least $\max\{\gamma, n\} \geq \frac{1}{2}(\gamma + n) = \Omega(\gamma + n)$. □

Theorem 3. *The required worst case label length for n node forests is* $\Omega(\gamma + \log n)$.

Proof. Consider the n node graph for which there are no adjacent nodes. To satisfy $\gamma \geq 1$, each label must be distinct, yielding a lower bound of $\log n$. Further, each node must have a Hamming distance of at least γ, and thus must have length at least γ for a total length of at least $\max\{\gamma, \log n\} = \Omega(\gamma + \log n)$. □

An important tool that we will make use of repeatedly in our constructions is the following *equidistant* code which is similar to the Hadamard codes 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.

The Equidistant Code. To define the Equidistant code we first define a special type of string. Consider a binary string S of length ℓ and an integer γ' with ℓ a multiple of $2\gamma'$ such that the w^{th} size $2\gamma'$ word of S is either all 1's or all 0's. Then, define $f_{\gamma'}(S)$ to be the length ℓ binary string with w^{th} size $2\gamma'$ word being γ' 0's followed by γ' 1's if the w^{th} word of S is 0's, and γ' 1's followed by γ' 0's if the w^{th} word of S is 1's.

For a given n and γ, each powers of 2 with $2\gamma \geq n$, define the n element *equidistant code* $\mathrm{ED}(n, \gamma)$ recursively as follows.

1. Define $\mathrm{ED}(2^1, \gamma)$ to be the 2 word code with S_1 consisting of γ 0's and S_2 consisting of γ 1's.
2. For $i \geq 1$, define $\mathrm{ED}(2^{i+1}, \gamma)$ as follows. Let $\mathrm{ED}(2^i, \gamma) = \langle S_1, S_2, \ldots, S_{2^i} \rangle$. For $j = 1$ to 2^{i+1}, let $S'_j = S_j \cdot 00 \cdots 00$ ($\frac{\gamma}{2^i}$ 0's) if $j \leq 2^i$ and $S'_j = f_{\frac{\gamma}{2^i}}(S_{j-2^i}) \cdot 11 \cdots 11$ ($\frac{\gamma}{2^i}$ 1's) if $j > 2^i$. $\mathrm{ED}(2^{i+1}, \gamma)$ is then $\langle S'_1, \cdots S'_{2^{i+1}} \rangle$.

It is straightforward to argue inductively that $\mathrm{ED}(n, \gamma)$ is well defined for all powers of 2 with $2\gamma \geq n$. We will also use a slight variant of this code such that the length of each word is exactly twice γ.

Balanced Equidistant Code. For a given n and γ, each powers of 2 with $2\gamma \geq n$, define the n element *balanced* equidistant code $\mathrm{ED}^B(n, \gamma)$ by appending $\frac{2\gamma}{n}$ 0's to each $S_i \in \mathrm{ED}(n, \gamma)$.

Theorem 4. *Consider the codes* $\mathrm{ED}(n, \gamma)$ *and* $\mathrm{ED}^B(n, \gamma)$ *for* n *and* γ *powers of 2 and* $2\gamma \geq n$. *The following properties hold.*

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

Proof of 1. We can show this for $\mathrm{ED}(2^{i+1}, \gamma)$ by induction on i. For $i = 0$, $2\gamma - \frac{2\gamma}{n} = \gamma$, which is equal to the length by definition. For $i \geq 1$, the length ℓ for $\mathrm{ED}(2^{i+1}, \gamma)$, by definition, is equal to the length ℓ' for $\mathrm{ED}(2^i, \gamma)$ plus $\frac{\gamma}{2^i}$, which, by inductive hypothesis, is $2\gamma - \frac{2\gamma}{2^i} + \frac{\gamma}{2^i} = 2\gamma - \frac{2\gamma}{2^{i+1}}$.
Proof of 2. Consider the following cases.

Case 1: $i, j \leq 2^i$. In this case the result follows by inductive hypothesis.

Case 2: $i, j > 2^i$. Since $H(X, Y) = H(f_{\frac{\gamma}{2}}(X), f_{\frac{\gamma}{2}}(Y))$, the result follows by inductive hypothesis.

Case 3: $i \leq 2^i$, $j > 2^i$. Note that for two length ℓ strings X and Y, $H(X, f_{\frac{\gamma}{2}}(Y)) = \frac{\ell}{2}$. Thus, by part 1 of the theorem, $H(S_i, S_j) = (2\gamma - \frac{2\gamma}{2^i})/2 + \frac{\gamma}{2^i} = \gamma$. □

Proof of 3. This follows immediately from parts 2 and 1. □

Proof of 4. This follows from 1 and the definition of $\mathrm{ED}^b(n, \gamma)$. □

Proof of 5. This follows from 2 and 4. □

Proof of 6. It is straightforward to compute $\mathrm{ED}(n, \gamma)$ in $\Theta(\gamma n)$ time given $\mathrm{ED}(\frac{n}{2}, \gamma)$. Thus, such an algorithm achieves time $T(n) = \Theta(\gamma n) + T(\frac{n}{2})$ for a given n. The solution to this recurrence is $\Theta(\gamma n)$. □

3 Exact Labelings for Special Graphs

In constructing a γ-labeling for general graphs, we make use of a more restrictive type of satisfying label 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 5 (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 6 (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 as follows.

Definition 7 (Edge Decomposition). *Graphs $G_1, \ldots, G_r$ where $G_i = (V_i, E_i)$ are said to form an edge decomposition of graph $G = (V, E)$ if:*

1. *$V = V_i$ for all i.*
2. *$E = \bigcup_i E_i$.*

Theorem 8. *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 $\beta = \sum_{i=1}^{r} \alpha_i + \gamma$.*

Proof of 1. Consider an arbitrary pair of vertices (u, v). If $(u, v) \notin E$, then clearly $H(\sigma(u), \sigma(v)) = \sum_{i=1}^{r} \beta_i = \beta$. On the other hand, if $(u, v) \in E$, then for at least one j we know that $(u, v) \in E_j$. Thus, we get that

$$H(\sigma(u), \sigma(v)) \leq \sum_{i=1}^{r} \beta_i - \beta_j + \alpha_j.$$

And since $\beta_j - \alpha_j \geq \gamma_j \geq \gamma$, we get that

$$H(\sigma(u), \sigma(v)) \leq \sum_{i=1}^{r} \beta_i - \beta_j + \beta_j - \gamma = \sum_{i=1}^{r} \beta_i - \gamma = \alpha.$$

□

Proof of 2. Consider an arbitrary pair of vertices (u, v). If (u, v) is an edge in $\overline{G}$, then (u, v) is not an edge in G, and so $H(\sigma(u), \sigma(v)) = \sum_{i=1}^{r} \alpha_i = \alpha$. On the other hand, if $(u, v) \notin \overline{E}$, then $(u, v) \in E$, and so for at least one j we know that $(u, v) \in E_j$. Thus, we get that

$$H(\sigma(u), \sigma(v)) \geq \sum_{i=1}^{r} \alpha_i - \alpha_j + \beta_j \geq \sum_{i=1}^{r} \alpha_i + \gamma = \beta.$$

□

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 satisfying labeling from Theorem 8.

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, we use Algorithm 2 InverseMatchingExact. The performance of these algorithms is analyzed in Lemmas 9 and 10.

Algorithm 1 MatchingExact(G, γ)

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

Lemma 9. *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 $O(\gamma n + n^2)$.*

Proof. The exact γ-labeling and length properties follow from parts 2 and 1 respectively of Theorem 4. The run time is also clearly linear in the output size given part 6 of Theorem 4. □

Algorithm 2 InverseMatchingExact(G, γ)

1. Let $\gamma' = max(\gamma, \frac{n}{2})$. Generate $\text{ED}^B(n, \gamma')$.
2. Arbitrarily index the edges of E. For $i = 1$ to $|E|$, consider the i^{th} edge (u_i, v_i):
   (a) Set $\sigma(u_i) \leftarrow S_i$.
   (b) Set $\sigma(v_i) \leftarrow S_i^c$.
3. **For** each vertex $v \in V$ that is not adjacent to an edge in E, set $\sigma(v) \leftarrow S_i$ for some S_i in $\text{ED}^B(n, \gamma')$ that has not yet been assigned.
4. **Output** σ.

Lemma 10. *Algorithm 2 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* $O(\gamma n + n^2)$.

Proof. The bound on length follows from part 4 of Theorem 4, the exact γ-labeling from parts 2 and 4, and the run time from part 6. □

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 3 StarExact obtains an exact γ-labeling for a star graph G. (In fact, it achieves an exact 2γ-labeling. The length of the generated labeling could be reduced by half by tightening this but is left for ease of explanation.) Figure 1 provides an example of the labeling assigned by StarExact. Algorithm 4 InverseStarExact obtains an exact inverse γ-labeling for a star graph G.

Algorithm 3 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, …, 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 $\text{ED}(\gamma', n)$.
6. **For** each $v_i \in B$ append x copies of $S_i \in \text{ED}(\gamma', n)$ to $\sigma(v_i)$.
7. **Output** σ.

Lemma 11. *Algorithm 3 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* $O(\gamma n^2)$.

Proof. We will first show the stated word length is achieved, followed by the stated α and β values, and finally the stated run time complexity.

(Length) Step 1 of StarExact does not append any length to strings. Steps 2, 3, and 4 append a total of exactly $n\gamma$ bits to each of the n output strings. From Theorem 4, each S_i from $\text{ED}(n, \gamma')$ has length $2\gamma' - \frac{2\gamma'}{n}$. Thus, steps 5 and 6 together append exactly $x \cdot (2\gamma' - \frac{2\gamma'}{n}) < 2x\gamma' = 2 \cdot \max\{\frac{n}{2}, \gamma\} \cdot \min\{\frac{n}{2}, \gamma\} = \gamma n$. The total length is thus length$(\sigma) < 2n\gamma = O(n\gamma)$.

Algorithm 4 InverseStarExact(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, ..., v_n with $c = v_n$.
2. Set the first 2γ bits of $\sigma(c)$ to be 0's.
3. **For** each vertex $v_i \in B$, set the first 2γ bits of $\sigma(v_i)$ to be 1's.
4. **For** each vertex $v_i \in A$, set the first γ bits of $\sigma(v_i)$ to be 1's and the second γ bits of $\sigma(v_i)$ to be 0's.
5. **For** each $v_i \in B$ or $v_i = c$ append x copies of S_i^c to $\sigma(v_i)$ where S_i is the i^{th} string in ED(γ', n).
6. **For** each $v_i \in A$ append x copies of $S_i \in$ ED(γ', n) to $\sigma(v_i)$.
7. **Output** σ.

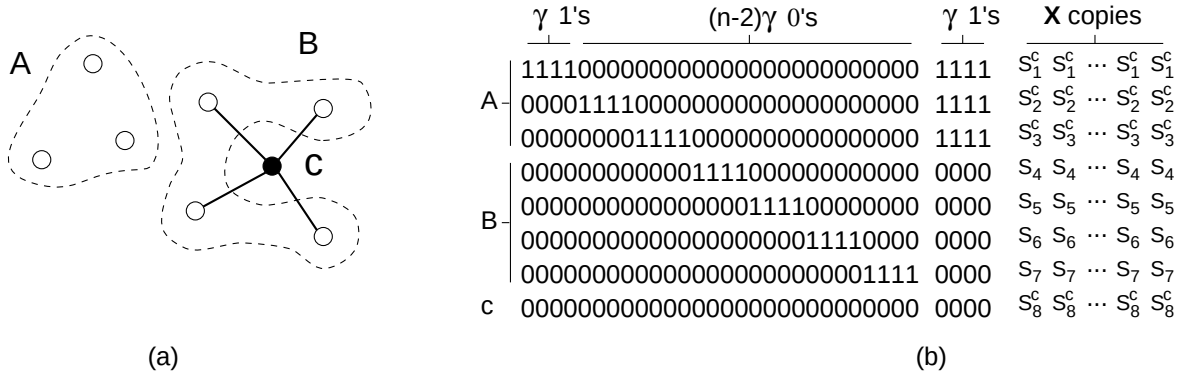

Figure 1: (a) A star graph and (b) its corresponding exact labeling.

(α and β) To show that the Hamming distance between each pair of strings satisfies the α and β requirement, consider two arbitrary vertices $v, u \in V$. There are five cases to consider.

Case 1: $v, u \in A$. From steps 3 and 4 a distance of 2γ is obtained. From step 5 an additional distance of exactly x times the Hamming distance between two distinct words from ED(n, γ') is added, which is exactly $x \cdot \gamma' = \frac{\gamma n}{2}$ from Theorem 4. This yields a total Hamming distance of $2\gamma + \frac{\gamma n}{2}$.

Case 2: $v, u \in B$. The analysis is the same as for case 1.

Case 3: $v \in A$, $u \in B$. Step 3 provides a Hamming distance of exactly 2γ. Step 4 adds an additional γ distance. By part 3 of Theorem 4 steps 5 and 6 add a Hamming distance of exactly $x \cdot (\gamma' - \frac{2\gamma'}{n})$. To finish the calculation, consider two subcases.

1. $\gamma \geq \frac{n}{2}$. In this case $x \cdot (\gamma' - \frac{2\gamma'}{n}) = \frac{n}{2}(\gamma - \frac{2\gamma}{n}) = \frac{\gamma n}{2} - \gamma$. The total Hamming distance is thus $H(u, v) = 2\gamma + \frac{\gamma n}{2}$.
2. $\gamma < \frac{n}{2}$. In this case $x \cdot (\gamma' - \frac{2\gamma'}{n}) = \gamma \cdot (\frac{n}{2} - 1) = \frac{\gamma n}{2} - \gamma$. The total Hamming distance is thus $H(u, v) = 2\gamma + \frac{\gamma n}{2}$.

Case 4: $v \in A$, $u = c$. Steps 2, 3, and 4 create a Hamming distance of exactly 2γ. By Theorem 4, step 5 adds another $x \cdot \gamma' = \frac{\gamma n}{2}$ Hamming distance for a total of $2\gamma + \frac{\gamma n}{2}$.

Case 5: $v \in B$, $u = c$. Steps 2, 3, and 4 create a Hamming distance of exactly γ. Steps 5 and

6 add an additional $x \cdot (\gamma' - \frac{2\gamma'}{n})$. As shown previously in cases 3.1 and 3.2, this value is always equal to $\frac{\gamma n}{2} - \gamma$, yielding a total Hamming distance of exactly $\frac{\gamma n}{2}$.

(Time Complexity) The time complexity of Steps 1, 2 and 4 are clearly dominated by that of step 3 which takes $\Theta(n^2\gamma)$. By Theorem 4 steps 5 and 6 each take $O(\gamma n)$ time to compute $\mathrm{ED}(n, \gamma)$. The rest of steps 5 and 6 are linear to the number of total bits appended for a total of $O(n \cdot x \cdot \gamma') = O(n^2\gamma)$. Finally, Step 7 is bounded by the size of the output $O(n^2\gamma)$ which is assured by the bound on the length of the labeling proven above. The total time complexity is thus $O(n^2\gamma)$. □

Lemma 12. *Algorithm 4 InverseStarExact(G, γ) obtains an exact inverse 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 $O(\gamma n^2)$.*

Proof. The run time complexity and word length analysis is essentially the same as for Lemma 11. To show the γ-labeling is achieved, consider two arbitrary vertices $v, u \in V$ and the following five cases.

Case 1: $v, u \in A$. $\sigma(u)$ and $\sigma(v)$ are identical except for the appendage from step 6 which yields, by Theorem 4 part 2, a Hamming distance of $x \cdot \gamma' = \frac{\gamma n}{2} = \alpha$.

Case 2: $v, u \in B$. By Theorem 4 parts 2 and 5, the Hamming distance is exactly $x \cdot \gamma' = \frac{\gamma n}{2} = \alpha$.

Case 3: $v \in A$, $u \in B$. From steps 3 and 4 a Hamming distance of exactly γ is obtained. By part 3 of Theorem 4 steps 5 and 6 add an additional distance of exactly $x(\gamma' - \frac{2\gamma'}{n}) = \frac{\gamma n}{2} - \gamma$, for a total distance of exactly $\frac{\gamma n}{2} = \alpha$.

Case 4: $v \in A$, $u = c$. The analysis is identical to that of Case 3.

Case 5: $v \in B$, $u = c$. Steps 2 and 3 create a Hamming distance of exactly 2γ. Step 5, by parts 2 and 5 of Theorem 4, add an additional $x\gamma' = \frac{\gamma n}{2}$ distance. The total is thus $2\gamma + \frac{\gamma n}{2} = \beta$. □

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 9 and 11 or Lemmas 10 and 12 to obtain exact or exact inverse labelings for these subgraphs, and then apply Theorem 8 to obtain a γ-labeling for the original graph. We first consider Algorithm 5 MatchingDecomposition that decomposes the general graph G into a collection of matchings.

4.1 Matching Decomposition

Lemma 13. *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.*

Proof. For any vertex, since each matching could contain only one of its incident edges, the number of matchings in a decompositions is $\Omega(D)$.

For the upper bound, we will prove that the maximum number of maximal matchings in the decomposition of a graph G is at most $2 \cdot D - 1$. Suppose we remove a maximal set of edges which form a matching in successive rounds of decomposition. We will argue that each edge $(x, y) \in E$ will be included in some decomposition within $2D - 1$ rounds. Since each round of decomposition computes a maximal matching M', $(x, y) \notin M'$ only if either x or y is an endpoint of an edge in M'. In the former case, one of the edges incident to x is included in M'. Since x is incident to at most D edges, this case can occur in at most $D - 1$ rounds. Similarly, the latter case can occur in at most $D - 1$ rounds. Therefore, the two cases can occur in at most $2 \cdot D - 2$ rounds. Consequently, in $2 \cdot D - 1$ rounds, (x, y) is included in some decomposition. □

Algorithm 5 MatchingDecomposition(G, γ) 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.

Algorithm 5 MatchingDecomposition$(G = (V, E), \gamma)$

1. Decompose G into maximal matchings $M_1, \ldots, M_r$.
2. **For** each matching M_i, generate an exact labeling $\sigma_i = \text{MatchingExact}(M_i, \gamma)$.
3. For every vertex v, $\sigma(v) = \sigma_1(v) \cdot \sigma_2(v) \cdots \sigma_r(v)$.
4. Output σ.

Theorem 14. *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 size of the maximum degree vertex in G and the maximum degree vertex in $\overline{G}$.*

Proof. Let D denote the size of the maximum degree node in G and D' denote the size of the maximum degree node in $\overline{G}$. To obtain such a labeling, apply Algorithm 5 MatchingDecomposition if $D \leq D'$. If $D > D'$, apply a modified version of Algorithm 5 MatchingDecomposition to the complement graph $\overline{G}$ in which step 2 assigns an inverse exact labeling to each matching according to Algorithm 2 InverseMatchingExact. Thus, by Theorem 8 a γ-labeling is obtained for G in either case. To bound the length, note that the labeling for each matching has length $O(\gamma + n)$ by Lemmas 9 and 10. Further, by Lemma 13 the total number of matchings is at most $\hat{D}$, yielding a total label length of $O(\hat{D}\gamma + \hat{D}n)$. The bound on run time follows from Lemmas 9 and 10. □

4.2 Hybrid Decomposition (Star Destroyer)

The next algorithm for obtaining a γ-labeling for a general graph adds the star decomposition to the basic matching algorithm. The intuition is that, from Theorem 14, the matching algorithm may perform poorly even if there are just a few very high and very low degree vertices in the graph. In that case, we can use the Star labeling to efficiently get rid of either the high or the low degree vertices, and then finish off with the matching algorithm.

Theorem 15. *For any graph G and γ, Algorithm 6 StarDestroyer(G, γ) yields a γ-labeling σ of G with* length$(\sigma) = O(\sqrt{\gamma \hat{m}^2 n + \gamma \hat{m} n^2})$ *in time complexity $O(\sqrt{\gamma \hat{m}^2 n^3 + \gamma \hat{m} n^4})$ where $\hat{m} = min\{|E|, |E^{-1}|\}$.*

Proof. It is straightforward to see that the run time is linear in the output size. To show that the output labeling σ is a γ-labeling, we observe that $\sigma(v)$ for each vertex is the concatenation of labelings from StarExact from step 2 and MatchingExact from step 3. From Lemmas 9, 11, and Theorem 8, this thus yields a γ-labeling of G.

To bound the length of the words, note that the while loop of step 2 can only execute at most $\frac{\hat{m}}{x}$ times. Thus, from Lemma 11, the length of each string $\sigma(v)$ after step 2 is at most $O(\frac{\hat{m}\gamma n}{x})$. After step 2, we know the degree of the graph is bounded by x, and thus know from Theorem 14 that the added length from step 3 is bounded by $O(x \cdot (\gamma + n))$. By choosing $x = \sqrt{\frac{\hat{m}\gamma n}{\hat{m}+n}}$ to balance these two terms a total length of length$(\sigma) = O(\frac{\hat{m}\gamma n}{x} + x \cdot (\gamma + n)) = O(\sqrt{\gamma \hat{m}^2 n + \gamma \hat{m} n^2})$ is achieved. □

Algorithm 6 StarDestroyer($G = (V, E), \gamma$)

1. **If** $|E| > |\overline{E}|$, **then** set INV$\leftarrow$ true and $G' \leftarrow (V, \overline{E})$. **Else**, set INV$\leftarrow$ false and $G' \leftarrow (V, E)$.
2. Let $m' = \min\{|E|, |E'|\}$. Set $x \leftarrow \sqrt{\frac{m'\gamma n}{m'+n}}$.
3. **While** $\exists v \in V$ such that $\deg(v) > x$ among edges in E' **Do**:
   (a) Let T denote the star graph consisting of vertex set V and the subset of E' consisting of all edges incident to v.
   (b) **If** INV= false, **then** compute the exact labeling $\sigma' = \text{StarExact}(T, \gamma)$.
   (c) **Else**, compute the exact inverse labeling $\sigma' = \text{InverseStarExact}(T, \gamma)$.
   (d) For each $u \in V$, set $\sigma(u) \leftarrow \sigma(u).\sigma'(u)$
   (e) Remove all edges incident to v from G'.
4. **If** INV= false, **then** set $\sigma' = \text{MatchingDecomposition}((V, E'), \gamma)$.
5. **Else** set $\sigma' = \text{InverseMatchingDecomposition}((V, E'), \gamma)$.
6. For each vertex $v \in V$, set $\sigma(v) = \sigma(v).\sigma'(v)$.
7. Output σ.

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 $O(\gamma D \log f + \max\{\gamma D \log(\frac{n}{\gamma D}), 0\})$ 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 combing the smaller trees into one. To do this, we will make use of a special type of γ-labeling.

Definition 16 (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, $\alpha_a \leq \mathrm{H}(\sigma(v), \sigma(u)) \leq \alpha_b$ if v and u are adjacent, and $\mathrm{H}(\sigma(v), \sigma(u)) \geq \beta$ 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 17. See Figure 2 for an illustration of Algorithm 7 CombineTrees. 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.

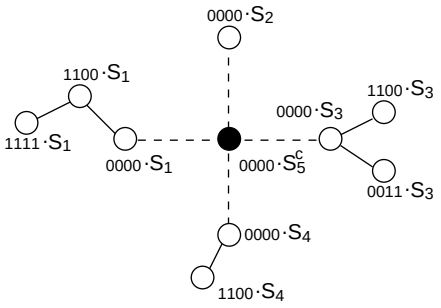

Figure 2: Given a lower bounded γ-labeling for four trees as shown, Algorithm CombineTrees first shifts the labeling so that the label for the root of each of the trees is all 0's. Then, $S_i \in \mathrm{ED}(D'+1, \frac{\gamma(D'+1)}{2})$ is appended to the label of each node in the i^{th} tree. To the central node is appended the complement of a final distinct string $S^c_{D'+1}$.

Lemma 17 (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 7 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 $\ell' \leq \ell + \gamma D'$.*

Proof. Let $T_1, \ldots T_t$ denote the t subtrees of v, σ denote the output labeling from CombineTrees(T, v, $\{\sigma_i\}_{i=1}^t$), and ℓ denote the largest label size from the t labelings σ_i. First note that the resultant length of σ is ℓ plus the length of the words in $\mathrm{ED}(D'+1, \frac{\gamma(D'+1)}{2}) = \gamma D'$ (Theorem 4), which yields a total length of $\ell' = \ell + \gamma D'$.

To show that CombineTrees preserves the lower bounded γ-labeling, consider two arbitrary vertices x and y in T. By a case analysis we will show that the Hamming distance between $\sigma(x)$ and $\sigma(y)$ always satisfies the specified lower bounded labeling. Without loss of generality, it is sufficient to consider the following 3 cases.

Case 1: x and y are both in the tree T_i for some i. In this case, the Hamming distance between $\sigma(x)$ and $\sigma(y)$ has not changed, and thus the preservation of the lower bounded labeling follows by assumption.

Case 2: x is in T_i, y is in T_j, $i \neq j$. As S_i is appended to x and S_j to y, the Hamming distance between $\sigma(x)$ and $\sigma(y)$ by Theorem 4 is at least β, which is consistent with the lower bounded labeling as x and y cannot be adjacent.

Case 3: x is in T_i, and $y = v$.

- Case 3.1: x is adjacent to $y = v$. In this case, $\sigma(x)$ and $\sigma(y)$ are equal (all 0's) except for the final appendage of S_i to $\sigma(x)$ and $S^c_{D'+1}$ to $\sigma(y)$. The Hamming distance is thus $\beta - \frac{2\beta}{D'+1} = \beta - \gamma$ by Theorem 4 which is equal to α_b, thus satisfying the lower bounded labeling.

- Case 3.2: x is not adjacent to $y = v$. By assumption, σ_i attains the specified lower bounded labeling. Thus, $\sigma_i(x)$ must have distance at least α_a from the root v_i of T_i. And as the

Algorithm 7 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 $S^c_{t+1} \in \text{ED}(D' + 1, \frac{\gamma(D'+1)}{2})$. 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 $S_i \in \text{ED}(D' + 1, \frac{\gamma(D'+1)}{2})$.
5. **Output** σ.

labeling σ_i is shifted in Step 2 of the algorithm so that the label of v_i is identical to that of $v = y$, the first part of $\sigma_i(x)$ with out the appended S_i must have Hamming distance at least α_a from $\sigma(y)$. Finally, the strings S_i and $S^c_{D'+1}$ add an additional distance of $\beta - \gamma$ (Theorem 4), for a total distance of at least $\alpha_a + \beta - \gamma = \beta$.

□

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 17 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 18 (Node Recursive Forest). *Given a forest F with maximum degree D, and an integer γ and a γ-labeling of F with length $O(\gamma D \log n)$ can be constructed in time $O(n\gamma D \log^2 n)$.*

Proof. Let D' be the smallest integer greater or equal to the maximum degree of any node in the input graph such that $(D' + 1)$ is a power of 2 greater than 2. Now, assuming at first that F is a tree, consider the simple recursive algorithm for computing a lower bounded γ-labeling with $\beta = \frac{\gamma(D'+1)}{2}$, $\alpha_b = \beta - \gamma$, and $\alpha_a = \gamma$. As a base case, it is simple to attain the desired lower

bounded labeling for a graph that is a path of only one or two nodes. For larger graphs, we simply find a node separator, which can be done in time $O(n)$, recursively solve for the subtrees of the separator, and combine using Lemma 17. By Lemma 17, the increase in length of the labeling for each combination of trees is $O(\gamma D')$, yielding a total length of $O(\gamma D' \log n) = O(\gamma D \log n)$.

To extend this to a t tree forest, we first apply the described algorithm to each tree of the forest, and then append a distinct string to each tree from a set of words such that each pair of distinct words has a Hamming distance of at least β. This can be done trivially by attaching length $O(\beta \log t) = O(\gamma D \log n)$ strings, which does not increase the length complexity or run time complexity beyond what is achieved for trees.

(Time Complexity) We bound the work done for each node as follows. Since each node has a label of length ℓ, for every run of the CombineTree algorithm the work done on a node in the input to the algorithm is $O(\ell)$. Since the size of the tree a node belongs to is at least doubled when the tree is combined by Algorithm CombineTree, the number of times any node participates in CombineTree is $O(\log n)$. Therefore the total work done for any node is $O(\ell \log n)$ which gives a total runtime of $O(n\ell \log n) = O(n\gamma D \log^2 n)$. □

5.3 Leaf Based Recursion

We now describe a more sophisticated recursive algorithm for labeling trees that makes use of recursion based on the number of leaves in the tree rather than the number of nodes. The key idea is that we can efficiently label paths using a combination of recursion for large paths and equidistant codes for shorter paths. Thus, we recursively break down the tree into trees with smaller numbers of leaves until we reach the base case of a path for which we employ our efficient path labeling algorithm.

Path Labeling. As a base case for our recursive algorithm, according to Lemma 17, 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 17 until the path is sufficiently short. Then, a labeling based on the equidistant code is used. Recursive Algorithm 8 Path achieves the following result.

Lemma 19. *For $D' \geq 3$, $D'+1$ a power of 2, and path P, Algorithm 8 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 $O(n \cdot (\max\{\gamma D' \log^2(\frac{n}{\gamma D'}), \gamma D'\}))$.*

Proof. For paths of length at most $2\gamma(D'+1) - 1$, Step 1 assigns each node a label of length $O(\gamma D)$ from Theorem 4. The Hamming distance between two adjacent nodes is the Hamming distance between two words in the equidistant code $\mathrm{ED}(\gamma(D'+1), \frac{\gamma(D'+1)}{4})$ which is $\frac{\gamma(D'+1)}{4}$. The Hamming distance between non adjacent nodes is twice the Hamming distance between two words in $\mathrm{ED}(\gamma(D'+1), \frac{\gamma(D'+1)}{4})$ which is $\frac{\gamma(D'+1)}{2}$. Therefore, the specified lower bounded γ-labeling is achieved for the base case.

For longer paths with n nodes, Algorithm 8 Path breaks the path into two paths of length $\frac{n}{2}$ and finds the labeling for each recursively. The labeling for the original path is then found using Algorithm 7 CombineTree which increases the label length by $O(\gamma D')$. The depth of recursion is

Algorithm 8 Path($P = \langle v_1, \ldots v_n \rangle, \gamma, D'$)

1. **If** $n \leq 2\gamma(D'+1) - 1$ **then**
   (a) Compute $S_1, \ldots S_{\gamma(D'+1)} \in \text{ED}(\gamma(D'+1), \frac{\gamma(D'+1)}{4})$.
   (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$(P, v_{\frac{n}{2}}, \{\sigma_1, \sigma_2\})$.

$O(\log \frac{n}{\gamma D'})$ and using Lemma 17 the increase in length of the labeling for each combination of trees is $O(\gamma D')$, yielding a total length of $O(\max\{\gamma D' \log(\frac{n}{\gamma D'}), \gamma D'\})$.

Analysis of time complexity is similar to that of Theorem 18. Each node has a label of length $O(\max\{\gamma D' \log(\frac{n}{\gamma D'}), \gamma D'\})$ and participates in CombineTree $O(\log(\frac{n}{\gamma D'})$ times. Therefore the total runtime is $O(n \cdot (\max\{\gamma D' \log^2(\frac{n}{\gamma D'}), \gamma D' \log(\frac{n}{\gamma D'}\})))$. □

5.3.1 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 20. *Every tree has a leaf separator.*

Proof. The proof is by construction. Consider an arbitrary tree T with f leaves. Removing any node u in T could lead to at most one connected component with more than $\lfloor \frac{f}{2} \rfloor + 1$ leaves. Let that component be called the *offending* component of u. For any node which is not a leaf separator with the offending component having g leaves and h nodes, we will find a node which is either a leaf separator or whose offending component has at most g leaves and $h - 1$ nodes. Since any component with at most $\lfloor \frac{f}{2} \rfloor + 2$ nodes could have at most $\lfloor \frac{f}{2} \rfloor + 1$ leaves, repeating this process will eventually find a leaf separator.

Consider an arbitrary node u in T. Let the adjacent nodes to u in T be $v_1, \ldots, v_r$. Consider the r trees $T_1, \ldots, T_r$ obtained by removing u from T where T_i is the tree containing v_i. If all these trees have at most $\lfloor \frac{f}{2} \rfloor + 1$ leaves we are done. Otherwise, note that only one of them can have more than $\lfloor \frac{f}{2} \rfloor + 1$ leaves. Without loss of generality, let T_1 be the tree with g leaves where $g > \lfloor \frac{f}{2} \rfloor + 1$ leaves.

Now consider removing v_1 from T. Of the trees obtained by this removal, note that the tree containing u will have at most $\lfloor \frac{f}{2} \rfloor + 1$ leaves and of the rest at most one could have greater than $\lfloor \frac{f}{2} \rfloor + 1$ leaves. Let that tree be T'. Note that number of leaves in T' is at most g and the number of nodes in T' is strictly less than that in T_1. □

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 19. Having removed a leaf separator and recursively solved for the sub trees, we can then apply Lemma 17 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 9 Tree$(T=(V,E),\gamma,D')$

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

Theorem 21 (Trees). *Consider a tree T with f leaves and integer $D'=2^j-1\geq\max\{\deg(T),3\}$. Then, Algorithm 9 Tree$(T,\gamma,D')$ computes a length $O(\gamma D'\log f+max\{\gamma D'\log(\frac{n}{\gamma D'}),0\})$ γ-labeling of T in time complexity $O(n\cdot(\gamma D'\log^2 f+\max\{\gamma D'\log^2(\frac{n}{\gamma D'}),0\}))$.*

Proof. This follows from Lemma 17, Lemma 19 and Lemma 20. □

To extend this result to a forest of trees $T_1\cdots T_t$, we can use Tree(T_i,γ,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 $\mathrm{ED}(t,\beta)$ for an additional length of $O(t+\gamma D)$. These approaches yield the the following theorem.

Theorem 22 (Leaf Recursive Forest). *There exists a deterministic algorithm that produces a length $O(\min\{t,\gamma D\log t\}+\gamma D\log f+\max\{\gamma D\log(\frac{n}{\gamma D}),0\})$ γ-labeling for an input forest F in time complexity $O(n\cdot(\min\{t,\gamma D\log t\}+\gamma D\log^2 f+\max\{\gamma D\log^2(\frac{n}{\gamma D}),0\}))$, 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.*

Proof. To achieve this for a given forest F, we can first label each tree in F using Theorem 21. We can then append strings of 0's to make all labels of equal length. This yields label length $O(\gamma D\log f+\max\{\gamma D\log(\frac{n}{\gamma D}),0\})$. Next, to obtain a γ-labeling for F, we can append a distinct string S_i from a list $\langle S_1,\ldots S_t\rangle$ to each of the t trees in F such that each distinct S_i and S_j has hamming distance at least γD. A satisfactory list of strings S_i of length $O(\gamma D\log t)$ can be obtained trivially. Alternately, a set of length $O(\gamma D+t)$ can be obtained by using the equidistant code according to Theorem 4. We thus get a total length bound of $O(\min\{t,\gamma D\log t\}+\gamma D\log f+\max\{\gamma D\log(\frac{n}{\gamma D}),0\})$. The time complexity bound follows from Theorems 21 and 4. □

Alternately, we can use randomization to append shorter strings to each tree and avoid an increase in complexity. Kao et al.[16] 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 23 (Randomized Forest). *There exists a randomized algorithm that produces a length* $O(\gamma D \log f + \max\{\gamma D \log(\frac{n}{\gamma D}), 0\})$ *γ-labeling for an input forest F with probability at least* $1 - \frac{1}{n+2^\gamma}$*, in time complexity* $O(n \cdot (\gamma D \log f + \max\{\gamma D \log(\frac{n}{\gamma D}), 0\}))$*, where* $D = \deg(F)$*, and f is the largest number of leaves in any of the trees in F.*

Proof. This is achieved in exactly the same fashion as in Theorem 22 except that the set $\mathcal{S} = \{S_1, \ldots, S_t\}$ can be constructed using randomized algorithms as described in [16] which yields words of length $O(\gamma D + \log t)$ in $O(t(\gamma D + \log t))$ time. Note that both the γD and $\log t$ terms of the added length are absorbed by the length complexity for building a single tree. □

6 Distance Labeling, Rings and Paths

In this section we initiate the study of distance labelings for graphs using Hamming distance by generalizing the flexible word design problem. We show how this problem can be solved efficiently for a certain class of ring (rings are paths such that the first and last vertices of the path are adjacent) by applying some basic properties of hypercube traversal. We then show how this can be extended to obtain efficient distance labelings for all paths as well as a γ-labeling for any ring. We also note that even distance labeling for simple graphs such as paths may already have use in the application of DNA computing to digital signal processing.

Definition 24 ((k, γ)-labeling). *A labeling σ of a graph G is said to be a (k, γ)-labeling if there exist non-negative integers $\alpha_1, \ldots, \alpha_k$, $\beta_1, \ldots, \beta_k$ such that $\beta_i - \alpha_i \geq \gamma$ and for each pair of nodes u, v in G, if the distance $d = dist(u, v) \leq k$, then $\beta_{d-1} \leq H(\sigma(u), \sigma(v)) \leq \alpha_d$, and if $d > k$, then $\beta_k \leq H(\sigma(u), \sigma(v))$. A (k, γ)-labeling is said to be lower exact if $H(\sigma(u), \sigma(v)) = \alpha_d$ for $d \leq k$.*

The goal of distance labeling is then to take as input a graph G and integers γ and k and output a (k, γ)-labeling of G. Note that for $k = 1$, this is the standard flexible word design problem. For the case of rings and paths, we have the following lower bound.

Theorem 25. *The required label length for (k, γ)-labelings for both the class of n-vertex rings and the class of n-vertex paths is* $\Omega(\gamma k + \log n)$.

Proof. For $n \geq 3$ each label must be distinct, thus yielding a bound of $\Omega(\log n)$. Further, given $k \leq \frac{n}{2}$ for rings and $k \leq n$ for paths there must be two nodes of distance at least k. The labels for these vertices must have Hamming distance, and thus length, of at least $\Omega(k\gamma)$ yielding a total bound of $\Omega(k\gamma + \log n)$. □

Our algorithms in this section make use of the following basic property concerning the traversal of nodes in a hypercube. Define a *Hamiltonian cycle* of a graph to be a path such that each vertex is visited exactly once, and the first and last vertex visited are adjacent.

Lemma 26. *For every degree d hypercube H^d and even integer $q = 2j$, there exists a size q subgraph H' of H^d such that H' has a Hamiltonian cycle p. Further, given d and q, such a path p can be computed in time $O(2^d)$.*

Proof. To show this, consider the algorithm HyperHamiltonian(H^d, q) to compute such of length q Hamiltonian path H' from a degree d hypercube H^d.

The correctness of HyperHamiltonian can be shown by induction on d. The base case of $d = 1$ is clear. Inductively assume that HyperHamiltonian is correct for $d - 1$. We then get by inductive hypothesis that a_i is adjacent to a_{i+1} and b_i is adjacent to b_{i+1} for $i = 1$ to $\frac{q}{2} - 1$. And by design $a_{\frac{q}{2}}$ is adjacent to $b_{\frac{q}{2}}$. The output is thus a Hamiltonian path of H^d.

Algorithm 10 HyperHamiltonian(H^d, q)

1. **If** $d = 1$, **output** any q node path.
2. **Else**
   (a) Divide H^d into the two subgraphs H_a^{d-1} and H_b^{d-1} that are each degree $d-1$ hypercubes.
   (b) Let $\langle a_1, \ldots, a_{d-1} \rangle \leftarrow$ HyperHamiltonian($H_a^{d-1}, \frac{q}{2}$) and b_i be the unique vertex in H_b^{d-1} that is adjacent to a_i.
   (c) **Output** $\langle a_1, \ldots, a_{\frac{q}{2}}, b_{\frac{q}{2}}, \ldots b_1 \rangle$.

To analyze the run time, let $T(d)$ denote the worst case run time of the algorithm for a degree d hypercube input. We see that $T(1) = c$ for a constant c, and for larger d, $T(d) \leq \Theta(2^d) + T(d-1)$. Solving this recurrence yields $T(d) = O(2^d)$. □

By thinking of the set of all length d binary strings as a graph such that strings are adjacent iff they have Hamming distance of 1, we have the degree d hypercube. Lemma 26 thus gives a way to produce an ordering for an even number of strings such that each consecutive pair has Hamming distance 1, and the first and last strings have Hamming distance 1. Our approach uses this to efficiently obtain a lower exact (k, γ)-labeling for rings that are of length a multiple of $2(k+1)$. With this result we can then obtain short (k, γ)-labelings for any length path or γ-labelings for any length ring.

Algorithm 11 SpecialRing($R = \langle v_1, \ldots v_n \rangle, k, \gamma$)

Input:

- Integer $k \leq \frac{n}{2} - 1$.
- An n vertex ring R with $n = (k+1)2w$ for integer $w \geq 1$.

Output: A lower exact (k, γ)-labeling of R with with $\alpha_i = i\gamma$.

1. Compute length $\lceil \log 2w \rceil$ strings $S_1, \ldots S_{2w}$ such that $H(S_1, S_{2w}) = H(S_i, S_{i+1}) = 1$ for $i = 1$ to $2w - 1$ (see Lemma 26).
2. **For** $i = 1$ to $k+1$, $S'_i \leftarrow S_1 \cdots S_1$ (i copies) $\cdot S_{2w} \cdots S_{2w} (k+1-i$ copies)
3. **For** $i = k+2$ to n, $S'_i \leftarrow S_{\lceil \frac{i}{k+1} \rceil} \cdot S_{\lceil \frac{i-1}{k+1} \rceil} \cdots S_{\lceil \frac{i-k}{k+1} \rceil}$
4. $\sigma(v_i) \leftarrow S'_i \cdot S'_i \cdot \cdots \cdot S'_i$ (γ copies) for $i = 1$ to n'.
5. **Output** σ.

Theorem 27. *For integers n, k, and γ such that $\frac{n}{k+1}$ is a positive even integer, Algorithm 11 SpecialRing(R, k, γ) produces a length $O(k\gamma \log \frac{n}{k})$ lower exact (k, γ)-labeling for the ring R with $\alpha_i = i\gamma$ in time $O(nk\gamma \log \frac{n}{k})$.*

Proof. The bound on label length and run time is straightforward. To show correctness, consider two arbitrary labels $S'_i \cdot S'_i \cdots S'_i$ and $S'_j \cdot S'_j \cdots S'_j$ with $\text{dist}(S'_i, S'_j) = r$, $j \geq i$. For simplicity, assume $i \geq k+2$.

To determine $H(S'_i, S'_j)$, consider the x^{th} size $\lceil \log 2w \rceil$ words $S'_{i,x}$ and $S'_{j,x}$ taken from S'_i and S'_j respectively. We know that

$$S'_{i,x} = S_{\lceil \frac{i-x}{k+1} \rceil} \quad \text{and} \quad S'_{j,x} = S_{\lceil \frac{i+r-x}{k+1} \rceil}.$$

Further, note that

$$\lceil \frac{i-x}{k+1} \rceil = \begin{cases} \lceil \frac{i-x}{k+1} \rceil & : \quad x < i \bmod (k+1) \\ \lceil \frac{i-x}{k+1} \rceil - 1 & : \quad x \geq i \bmod (k+1) \end{cases}$$

$$\lceil \frac{i+r-x}{k+1} \rceil = \begin{cases} \lceil \frac{i-x}{k+1} \rceil & : \quad x-r < i \bmod (k+1) \\ \lceil \frac{i-x}{k+1} \rceil - 1 & : \quad x-r \geq i \bmod (k+1) \end{cases}$$

Thus,

$$S'_{i,x} = S'_{j,x} \iff x < i \bmod (k+1) \vee x \geq i \bmod (k+1) + r.$$

Thus, the Hamming distance between the x^{th} words is non-zero for exactly r words. If $r \leq k$, each of these words has Hamming distance exactly 1, for a total of exactly Hamming distance r. For $r > k$, the Hamming distance is at least $k+1$. A similar argument applies if $i \leq k+1$. Thus, $H(S'_i \cdot S'_i \cdots S'_i, S'_j \cdot S'_j \cdots S'_j) = \gamma r$ for $r \leq k$ and $H(S'_i \cdot S'_i \cdots S'_i, S'_j \cdot S'_j \cdots S'_j) \geq \gamma(k+1)$ for $r > k$. □

With this theorem, we can immediately extend this result to a distance labeling for paths that does not limit the choice of n and k.

Theorem 28. *There exists an algorithm that computes a length $O(k\gamma + k\gamma \log \frac{n}{k})$ (k, γ)-labeling for an n node path in time $O(n \cdot (k\gamma + k\gamma \log \frac{n}{k}))$.*

Proof. To achieve this output the first n labels from SpecialRing(R', k, γ) where R' is a $4(k+1)\lceil \frac{n}{2(k+1)} \rceil$ node ring. This is sufficient since R' has a length that is at least twice that of n, implying the first and last nodes in the path have sufficiently large Hamming distance. □

Alternately, we can use the equidistant codes to insert 1, 2, or 3 extra labels into the list of labels from SpecialRing with $k = 1$ to obtain a γ-labeling for an arbitrary length n ring.

Algorithm 12 Ring$(R = \langle v_1, \ldots v_n \rangle, \gamma)$

1. Let $n' = 4\lfloor \frac{n}{4} \rfloor$, let $\rho = n - n'$, and let $\gamma' = \max\{\gamma, 2^{\lceil \log(2\rho+1) \rceil - 1}\}$. Compute SpecialRing$(R', 1, \gamma')$ for an n' node ring R'. Let the strings assigned by SpecialRing be $X_1, \ldots X_{n'}$.
2. Let $S_1, \ldots S_{2\rho+1}$ be distinct elements of $\mathrm{ED}^B(2^{\lceil \log(2\rho+1) \rceil}, \gamma')$.
3. **If** $\rho = 0$, **then output** $\sigma(v_i) = X_i$.
4. **Else**
   (a) **For** $i = 1$ to ρ **do**
      i. $\sigma(v_{3i-2}) \leftarrow X_{2i-1} \cdot X_{2i-1} \cdot S_{2i-1}$
      ii. $\sigma(v_{3i-1}) \leftarrow X_{2i-1} \cdot X_{2i} \cdot S_{2i}$
      iii. $\sigma(v_{3i}) \leftarrow X_{2i} \cdot X_{2i} \cdot S^c_{2i-1}$
   (b) **For** $i = 3\rho + 1$ to n, $\sigma(v_i) \leftarrow X_{i-\rho} \cdot X_{i-\rho} \cdot S_{2\rho+1}$
5. **Output** σ.

Theorem 29. *Algorithm 12 Ring(R, γ) computes a length $O(\gamma \log n)$ γ-labeling for the n node ring R in time $O(n\gamma \log n)$.*

Proof. The bounds for run time and word length follow from Theorem 27. To show correctness we show that the labeling is a γ-labeling with $\alpha = 3\gamma'$ and $\beta = 4\gamma'$ where γ' is as defined in the Ring algorithm.

Consider two arbitrary vertices u and w in R. First, if neither u nor w are one of the first 3ρ vertices indexed by the algorithm, then the desired Hamming distance between $\sigma(u)$ and $\sigma(w)$ is guaranteed by Theorem 27. Assuming otherwise, we consider the possible cases with respect to the indexing of the vertices in Ring. First, assume u and w are adjacent.

Case 1: $u = v_{3i-2}$, $w = v_{3i-1}$ or $u = v_{3i-1}$, $w = v_{3i}$. From Theorem 27 we have that $H(X_{2i}, X_{2i-1}) = \gamma'$. And from Theorem 4, $H(S_{2i-1}, S_{2i}) = H(S_{2i}, S^c_{2i-1}) = \gamma'$. Thus, for these two cases the Hamming distance between $\sigma(u)$ and $\sigma(w)$ is exactly $2\gamma' < \alpha$.

Case 2: $u = v_{3i}$, $w = v_{3i+1}$. $H(\sigma(v_{3i}), \sigma(v_{3i+1})) = 2H(X_{2i}, X_{2i+1}) + H(S^c_{2i-1}, S_{2i+1}) = 3\gamma' = \alpha$ by Theorems 27 and 4.

Now consider the scenario in which u and w are not adjacent. Since we have added at most only one extra label per each two labels, if $\text{dist}(u, w)$ is 3 or more, then we immediately get a Hamming distance of at least $\beta = 4\gamma'$ from Theorem 27. For $\text{dist}(u, w) = 2$, we have the following cases.

Case 1: $u = v_{3i-2}$, $w = v_{3i}$. $H(\sigma(v_{3i-2}), \sigma(v_{3i})) = 2H(X_{2i-1}, X_{2i}) + H(S_{2i-1}, S^c_{2i-1}) = 4\gamma' = \beta$ by Theorems 27 and 4.

Case 2: $u = v_{3i-1}$, $w = v_{3i+1}$. $H(\sigma(v_{3i-1}), \sigma(v_{3i+1})) = H(X_{2i-1}, X_{2i+1}) + H(X_{2i}, X_{2i+1}) + H(S_{2i}, S^c_{2i+1}) = 4\gamma' = \beta$ by Theorems 27 and 4.

Case 3: $u = v_{3i}$, $w = v_{3i+2}$. $H(\sigma(v_{3i-1}), \sigma(v_{3i+1})) = H(X_{2i}, X_{2i+1}) + H(X_{2i}, X_{2i+2}) + H(S^c_{2i-1}, S_{2i+2}) = 4\gamma' = \beta$ by Theorems 27 and 4.

□

Application. Tsaftaris et al. [20, 21] 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 given word and the Watson-Crick complement of a second word is proportional to the difference in the indices of the two words if the words are within some distance k of one another. Otherwise, if the difference of the indices of the words is greater than k, then the melting temperature needs to be large, but not necessarily proportional to the index difference. By taking hamming distance as an approximate measure of the melting temperature between a given string and the Watson-Crick complement of another, we can formulate this problem as finding a (k, γ)-labeling for an n node input path.

7 Edge Weighted Graphs

In this section we consider graphs with integer weights on edges. A weighted-γ-labeling for a weighted graph is a labeling such that for some β, the Hamming distance between nodes with an edge of weight w is $\beta - w\gamma$. Nodes which have no edge between them can be thought of as having edges of weight zero. Thus, the Hamming distance between the labels for such nodes is β. Note that the nodes with heavier edges have closer labels.

Problem 30 (Weighted Flexible Word Design Problem).
INPUT: *Weighted graph $G = (V, E)$; weight function $w : E \to \mathbb{N}$; integer γ*
OUTPUT: *A mapping σ: $V \to \{0,1\}^\ell$ such that for some integer β, $H(\sigma(v), \sigma(u)) = \beta - w(u, v)\cdot\gamma$ iff $(v, u) \in E$ and $H(\sigma(v), \sigma(u)) \geq \beta$ iff $(v, u) \notin E$. Minimize ℓ.*

Extending Exact Labeling for Un-weighted General Graphs. The algorithms presented in Section 4 for exact γ-labeling of un-weighted general graphs can be used to obtain a weighted-

γ-labeling for a weighted graph by replacing each edge of weight w with w un-weighted edges.

Let Δ be the maximum weighted degree and W be the sum of weights of all the edges in the weighted graph G.

Theorem 31. *For any weighted graph G and γ, there exists a weighted-γ-labeling σ of G with length$(\sigma) = O(\min\{\Delta\gamma + \Delta n, \sqrt{\gamma W^2 n + \gamma W n^2}\})$ which can be computed in time $O(\min\{n\cdot(\Delta\gamma + \Delta n), n\cdot\sqrt{\gamma W^2 n + \gamma W n^2}\})$.*

Proof. The proof is by construction. Let G' be the un-weighted multigraph obtained by replacing every edge with weight w in G with w un-weighted edges.

Algorithm MatchingDecomposition(G',γ) yields a weighted-γ-labeling σ of G with length$(\sigma) = O(\Delta\gamma + \Delta n)$ in time $O(n\cdot(\Delta\gamma + \Delta n))$. Algorithm StarDestroyer(G',γ) yields a weighted-γ-labeling σ of G with length$(\sigma) = O(\sqrt{\gamma W^2 n + \gamma W n^2}\ )$ in time $O(n\cdot\sqrt{\gamma W^2 n + \gamma W n^2})$. □

Weighted Matching Decomposition. A w-weighted matching is a matching where each edge has weight w. We can use the equidistant codes to get a weighted γ-labeling of length $O(Dw\gamma + Dn)$ for a w-weighted matching where D is the maximum un-weighted degree. Note that this is better than $O(\Delta\gamma + \Delta n)$ obtained by replacing each edge by w un-weighted edges and then using the Matching algorithm for general graphs.

Lemma 32. *For a weighted graph G, a weighted exact γ-labeling with $\beta = max(w\gamma, \frac{n}{2})$, and* length $= O(w\gamma + n)$ *can be constructed in time $O(w\gamma n + n^2)$.*

Proof. Since a γ-labeling for a w-weighted matching is the same as a $w\gamma$-labeling for an un-weighted matching, Algorithm MatchingExact$(G, w\gamma)$ outputs the required γ-labeling for a w-weighted matching G. □

We can exploit this idea to do weighted matching decompositions which yield shorter length labelings if the weighted graph has a small number of distinct weights. If the graph has ρ distinct weights $w_1, \ldots, w_\rho$, let d_i be the maximum un-weighted degree of the graph when restricted to edges of weight w_i. We can get a weighted-γ-labeling of length $O(\sum_i w_i d_i \gamma + \sum_i d_i n)$ using Algorithm 13 WeightedMatchingDecomposition-1(G, γ) (Theorem 33) and a weighted-γ-labeling of length $O(Dw_\rho\gamma + D\rho n)$ using Algorithm 14 WeightedMatchingDecomposition-2(G, γ) (Theorem 34).

Algorithm 13 WeightedMatchingDecomposition-1$(G = (V, E), \gamma)$

Let E_{w_i} be the subset of edges in E with weight w_i. Let $G_i = (V, E_{w_i})$. Let d_i be the maximum degree of a node in G_i.

1. **For** $i \leftarrow 1$ to ρ,
   (a) Decompose G_i into maximal matchings $M_1, \ldots, M_{k_i}$.
   (b) **For** each Matching M_j, Generate an exact labeling of M_j, σ^i_j = MatchingExact$(M_j, w_i\gamma)$.
   (c) **For** every vertex v, $\sigma^i(v) = \sigma^i_1(v).\sigma^i_2(v)\cdots\sigma^i_{k_i}(v)$.
2. **For** every vertex v, $\sigma(v) = \sigma^1(v).\sigma^2(v)\cdots\sigma^\rho(v)$.
3. **Output** σ.

Algorithm 14 WeightedMatchingDecomposition-2($G = (V, E), \gamma$)

Let the weights of the edges in G be $w_1, \ldots, w_\rho$ such that $w_1 \leq w_2 \leq \cdots \leq w_\rho$. Let E'_{w_i} be the subset of edges in E with weights at least w_i. Let $G'_i = (V, E'_{w_i})$. Let D be the maximum degree of a node in G. Let $w_0 = 0$.

1. **For** $i \leftarrow 1$ to ρ,
   (a) Decompose G'_i into maximal matchings $M_1, \ldots, M_{k_i}$.
   (b) **For** each Matching M_j, Generate an exact labeling of M_j, $\sigma^i_j = \text{MatchingExact}(M_j, (w_i - w_{i-1})\cdot\gamma)$.
   (c) **For** every vertex v, $\sigma^i(v) = \sigma^i_1(v).\sigma^i_2(v)\cdots\sigma^i_{k_i}(v)$.
2. **For** every vertex v, $\sigma(v) = \sigma^1(v).\sigma^2(v)\cdots\sigma^\rho(v)$.
3. **Output** σ.

Theorem 33. *For any graph G and γ, Algorithm 13 WeightedMatchingDecomposition-1(G, γ) yields a weighted-γ-labeling σ of G with* $\text{length}(\sigma) = O(\sum_i w_i d_i \gamma + \sum_i d_i n)$ *in time complexity* $O(n\cdot(\sum_i w_i d_i \gamma + \sum_i d_i n))$.

Proof. The bound on length follows immediately from Lemmas 32, 13 ($k_i = O(d_i)$), and Theorem 8. The bound on run time complexity follows from Lemmas 32 and 13. □

Theorem 34. *For any graph G and γ, Algorithm 14 WeightedMatchingDecomposition-2(G, γ) yields a weighted-γ-labeling σ of G with* $\text{length}(\sigma) = O(Dw_\rho\gamma + D\rho n)$ *in time complexity* $O(n\cdot(Dw_\rho\gamma + D\rho n))$.

Proof. An edge (u, v) with weight w_j occurs in matchings $G'_1, \ldots, G'_j$. For $i \leq j$, $\text{HD}(\sigma_i(u), \sigma_i(v)) = (w_i - w_{i-1})\gamma$. For $i > j$, $\text{HD}(\sigma_i(u), \sigma_i(v)) = 0$. Therefore, $\text{HD}(\sigma(u), \sigma(v)) = \sum_i \text{HD}(\sigma_i(u), \sigma_i(v)) = w_j\gamma$.

Using Lemma 9, the length of the labeling is $O(\sum_i (D(w_i - w_(i-1)\gamma + Dn)) = O(Dw_\rho\gamma + D\rho n)$ and the time complexity is $O(n\cdot(Dw_\rho\gamma + D\rho n))$. □

Hybrid Matching, Star and Onion Decomposition. An *onion* is a graph with just one edge of weight w. Using $\text{ED}(n, w\gamma)$, we get an exact weighted-γ-labeling of length $O(w\gamma + n)$ for an onion with edge weight w. The idea for the hybrid algorithm is to first generate labelings for all heavy edges. Then use the star algorithm for all the heavy degree vertices and use the matching algorithm for the remainder of the graph.

Algorithm 15 Weighted-StarExact($S = (V, E), \gamma$)

1. **While** $\exists e \in E$ **Do**:
   (a) $\sigma' = \text{StarExact}((V, E), \gamma)$.
   (b) **For** each $u \in V$, set $\sigma(u) \leftarrow \sigma(u) \cdot \sigma'(u)$.
   (c) Reduce the weight of every edge in E by one.
   (d) Remove any edges with zero weight from E.
2. **Output** σ.

Algorithm 16 WeightedHybrid$(G = (V, E), \gamma)$

Let π be the maximum weight of an edge in E and W the total weight of G.

1. Set $x \leftarrow \sqrt[3]{\frac{W(\pi\gamma+n)^2}{\gamma n(\gamma+n)}}$ and $y \leftarrow \sqrt[3]{\frac{W^2 n\gamma(\pi\gamma+n)}{(\gamma+n)^2}}$.
2. **While** $\exists e \in E$ such that weight$(e) > x$ **Do**:
   (a) Let $O = (V, e)$ denote the onion graph of weight weight(e). Compute exact labeling $\sigma' =$ MatchingExact$(O,$ weight$(e)\gamma)$.
   (b) **For** each $u \in V$, set $\sigma(u) \leftarrow \sigma(u).\sigma'(u)$.
   (c) Remove e from E.
3. **While** $\exists v \in V$ such that weighted-degree$(v) > y$ **Do**:
   (a) Let E' be the set of edges in E incident to v. Let E'' be a set of un-weighted edges where each edge in E' with weight w is replaced by w un-weighted edges. Let $S = (V, E'')$ be a star graph. Compute exact labeling $\sigma' =$ Weighted-StarExact(S, γ).
   (b) **For** each $u \in V$, set $\sigma(u) \leftarrow \sigma(u).\sigma'(u)$.
   (c) Remove E' from E.
4. Let E' be a set of un-weighted edges where each edge in E with weight w is replaced by w un-weighted edges. Set $\sigma' =$ MatchingDecomposition$((V, E'), \gamma)$.
5. **For** each vertex $v \in V$, set $\sigma(v) = \sigma(v).\sigma'(v)$.
6. **Output** σ.

Theorem 35. *For any weighted graph G and γ, Algorithm 16 WeightedHybrid(G, γ) yields a weighted-γ-labeling σ of G with* length$(\sigma) = O(\sqrt[3]{W^2\gamma n(\gamma+n)(\pi\gamma+n)})$ *in time complexity* $O(n\sqrt[3]{W^2\gamma n(\gamma+n)(\pi\gamma+n)})$.

Proof. It is straightforward to see that the run time is linear in the output size.

(Correctness) To show that the output labeling σ is a γ-labeling, note that Algorithm 16 WeightedHybrid(G, γ) decomposes the input graph G into onion-graphs, star graphs and matchings. Since the output labeling is the concatenation of the labeling for each of the decomposition, from Theorem 8 this yields a γ-labeling of G.

Next we bound the label length. The number of onion graphs is bounded by $\frac{W}{x}$ since there could be at most $\frac{W}{x}$ edges with weight at least x. Using Lemma 9, the length of the labeling for each onion decomposition is $O(\pi\gamma + n))$ since π is the maximum weight of an edge in G. So the length of the output labeling due to onion decompositions is $O(\frac{W}{x}(\pi\gamma+n))$. In the remaining graph the maximum weight of any edge is x. Since the number of nodes with weighted degree at least y could be at most $\frac{W}{y}$ and the length of the labeling for each of the star decompositions is $O(x\gamma n)$ using Lemma 11, the length of the output labeling due to star decompositions is $O(\frac{W}{y}(x\gamma n))$. Since the remaining graph has vertices of weighted degree at most y, using Lemma 9 the length of the labeling due to matching decompositions is $O(y(\gamma + n))$. Therefore, the total length of the output labeling is $O(\frac{W}{x}(\pi\gamma + n) + \frac{W}{y}(x\gamma n) + y(\gamma + n))$. Using $x = \sqrt[3]{\frac{W(\pi\gamma+n)^2}{\gamma n(\gamma+n)}}$ and $y = \sqrt[3]{\frac{W^2 n\gamma(\pi\gamma+n)}{(\gamma+n)^2}}$, we achieve an output labeling of length $O(n(W^2\gamma n\ (\gamma + n)\ (\pi\gamma + n))^{\frac{1}{3}}))$. □

8 Don't Care Edges

In this section we initiate an examination of a natural generalization of the flexible word design problem. The problem up till now has taken a graph as input such that each pair of vertices is required to have a large Hamming distance or a small Hamming distance. The generalization is to permit a third type of relation, one which makes no requirement on the Hamming distance of the pair of vertices. So the input graph can be seen as having two kinds of edges - *near* edges (N) and *far* edges (F). For nodes with a near edge, we require the Hamming distance between their labels to be small and for nodes with far edges, the Hamming distance between their labels should be large. For nodes that do not have an edge between them, we don't care about the Hamming distance between their labels.

Problem 36 (Flexible Word Design with Don't Care Edges).
INPUT*: Graph $G = (V, N, F)$; integer γ*
OUTPUT*: A mapping σ: $V \to \{0,1\}^\ell$ such that there exists integers β, α, $\beta - \alpha \geq \gamma$ such that $H(\sigma(v), \sigma(u)) \leq \alpha$ if $(u, v) \in N$ and $H(\sigma(v), \sigma(u)) \geq \beta$ if $(u, v) \in F$. Minimize ℓ.*

Our approach for this problem is to mark all the Don't Care edges as either far edges or near edges and hence reduce the problem to Flexible Word Design for un-weighted graphs. Corresponding to the two algorithms for un-weighted graphs, we get the following two algorithms for graphs with Don't Care edges.

Let D_N be the maximum degree of any vertex in $G_N = (V, N)$ and D_F be the maximum degree of any vertex in $G_F = (V, F)$. Let $\tilde{D} = \min\{D_N, D_F\}$ and $\tilde{m} = \min\{|N|, |F|\}$.

Theorem 37. *For any graph $G = (V, N, F)$ and γ, there exists a γ-labeling σ of G with length$(\sigma) = O(\min\{\sqrt{\gamma\tilde{m}^2 n + \gamma\tilde{m}n^2}, \gamma\tilde{D} + \tilde{D}n\})$ which can be computed in time $O(\min\{n\sqrt{\tilde{m}\cdot\gamma\cdot n\cdot(\tilde{m}+n)}, n(\gamma\tilde{D} + \tilde{D}n)\})$.*

Proof. The proof is by construction. The following two constructions achieve a labeling of lengths $O(\gamma\tilde{D} + \tilde{D}n)$ and $O(\sqrt{\gamma\tilde{m}^2 n + \gamma\tilde{m}n^2})$ in time $O(n(\gamma\tilde{D} + \tilde{D}n))$ and $O(n\sqrt{\tilde{m}\cdot\gamma\cdot n\cdot(\tilde{m}+n)})$ respectively.

1. If $D_N < D_F$, then let $\sigma =$ MatchingDecomposition$((V, N), \gamma)$, else let $\sigma =$ InverseMatching-Decomposition$((V, F), \gamma)$. From Theorem 14, the length of labeling σ is $O(\gamma\tilde{D}n + \tilde{D}n^2)$ and it can be computed in time $O(n(\gamma\tilde{D} + \tilde{D}n))$.
2. If $|N| < |F|$, then output StarDestroyer$((V, N), \gamma)$, else output StarDestroyer$((V, \overline{F}), \gamma)$. From Theorem 15, the length of labeling σ is $O(\sqrt{\gamma\tilde{m}^2 n + \gamma\tilde{m}n^2})$ and it can be computed in time $O(n\sqrt{\tilde{m}\cdot\gamma\cdot n\cdot(\tilde{m}+n)})$.

□

9 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. Finally, 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.

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] S. Kannan, N. Naor, and S. Rudich. Implicit representation of graphs. *SIAM Journal on Discrete Mathematics*, 5:596–603, 1992.

[16] 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.

[17] O. D. King. Bounds for DNA Codes with Constant GC-content. *Electronic Journal of Combinatorics*, 10(1):#R33 13pp, 2003.

[18] A. Marathe, A. Condon, and R. M. Corn. On Combinatorial DNA Word Design. *Journal of Computational Biology*, 8(3):201–219, 2001.

[19] 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.

[20] S. A. Tsaftaris. DNA Computing from a Signal Processing Viewpoint. *IEEE Signal Processing Magazine*, 21:100–106, September 2004.

[21] S. A. Tsaftaris. How can DNA-Computing be Applied in Digital Signal Processing? *IEEE Signal Processing Magazine*, 21:57–61, November 2004.

[22] 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.

[23] D. C. Tulpan, H. H. Hoos, and A. Condon. Stochastic Local Search Algorithms for DNA Word Design. In M. Hagiya and A. Ohuchi, editors, *Lecture Notes in Computer Science 2568: Proceedings of the 8th International Workshop on DNA-Based Computers*, pages 229–241. Springer-Verlag, New York, NY, 2003.

[24] E. Winfree, F. Liu, L. Wenzler, and N. Seeman. Design and self-assembly of two-dimensional DNA crystals. *Nature*, 394:539–544, August 1998.