Exact and Approximation Algorithms for DNA Tag Set Design*

Ion I. Măndoiu Dragoş Trincă

Abstract

In this paper we propose new solution methods for designing tag sets for use in universal DNA arrays. First, we give integer linear programming formulations for two previous formalizations of the tag set design problem. We show that these formulations can be solved to optimality for problem instances of moderate size by using general purpose optimization packages, and also give more scalable algorithms based on an approximation scheme for packing linear programs. Second, we note the benefits of periodic tags, and establish an interesting connection between the tag design problem and the problem of packing the maximum number of vertex-disjoint directed cycles in a given graph. We show that combining a simple greedy cycle packing algorithm with a previously proposed alphabetic tree search strategy yields an increase of over 40% in the number of tags compared to previous methods.

*A preliminary version of this manuscript has appeared in *Proc. 16th Annual Symposium on Combinatorial Pattern Matching*, pp. 383–393, 2005. Authors' address: Computer Science and Engineering Department, University of Connecticut, 371 Fairfield Rd., Unit 2155, Storrs, CT 06269-2155. E-mail: {ion.mandoiu,dragos.trinca}@uconn.edu.

Keywords: DNA tag arrays, SNP genotyping, cross-hybridization, graph algorithms, cycle packing.

1 Introduction

Recently developed *universal DNA tag arrays* (Brenner, 1997; Morris et al., 2002; Gerry et al., 1999) offer a flexible and cost-effective alternative to custom-designed DNA arrays for performing a wide range of genomic analyses. A universal tag array consists of a set of DNA strings called *tags*, designed such that each tag hybridizes strongly to its own *antitag* (Watson-Crick complement), but to no other antitag. A typical assay based on universal tag arrays performs Single Nucleotide Polymorphism (SNP) genotyping using the following steps (BenDor et al., 2004; Hirschhorn et al., 2000): (1) A set of *reporter oligonucleotide probes* is synthesized by ligating antitags to the $5'$ end of primers complementing the genomic sequence immediately preceding the SNP. (2) Reporter probes are hybridized in solution with the genomic DNA under study. (3) Hybridization of the primer part ($3'$ end) of a reporter probe is detected by a single-base extension reaction using the polymerase enzyme and dideoxynucleotides fluorescently labeled with 4 different dyes. (4) Reporter probes are separated from the template DNA and hybridized to the universal array. (5) Finally, fluorescence levels are used to determine the identity of the extending dideoxynucleotides (and hence the corresponding SNP genotype) for each primer.

Tag set design involves balancing two conflicting requirements: on one hand we would like a large number of tags to allow assaying a large number of biochemical reactions, on the other hand we would like the tags to maintain hybridization specificity under a wide range of experimental conditions. The antitag-to-tag hybridization specificity constraints have been previously formal-

ized in (Ben-Dor et al., 2000) using a hybridization model based on the nucleation complex theory and the classical 2-4 rule for melting temperature computation. Ben-Dor et al. have also proposed a near-optimal heuristic for this formalization of the problem. In this paper we give an integer linear programming (ILP) formulation for this problem and its variant in which tags are required to have equal length (Mandoiu et al., 2005) (Section 3), and propose faster heuristics based on an approximation scheme for packing linear programs due to (Garg & Konemann, 1998) (Section 4).

Previous works on tag set design (Ben-Dor et al., 2000; Mandoiu et al., 2005) have required that substrings that can act as nucleation complexes appear at most once within a selected tag. This substring non-repetition constraint simplifies analysis – e.g., it is the key property enabling the DeBruijn sequence based heuristic in (Ben-Dor et al., 2000)) – but is *not* required for correct tag functionality, which only requires for potential nucleation complexes not to appear simultaneously in two different tags. To our knowledge, no previous work has assessed the impact of this non-repetition constraint on tag set size. In Section 5 we give two algorithms for designing tag sets while allowing nucleation complex repetitions within a tag. The first one is a simple modification of the alphabetic tree search strategy in (Morris et al., 2002; Mandoiu et al., 2005). The second algorithm stems from the observation that periodic tags, particularly those with short period, "consume" the least number of minimal substrings that can form nucleation complexes, and thus should be given priority in tag selection. We then establish an interesting connection between the problem of finding the largest set of compatible periodic tags and the problem of packing the maximum number of vertex-disjoint directed cycles in a given graph. We prove that the latter problem is APX-hard even for regular directed graphs with in-degree and out-degree of 2, and propose a simple greedy cycle packing algorithm. Results in Section 6 show that combining the greedy cycle packing algorithm with the alphabetic tree search strategy yields an increase of over 40% in the

number of tags compared to previous methods.

2 Problem Formulations and Previous Work

A main objective of universal array designers is to maximize the number of tags, which directly determines the number of reactions that can be multiplexed using a single array. At the same time, tag sets must satisfy a number of *stability* and *non-interaction* constraints (Brenneman & Condon, 2002). The full set of constraints depends on factors such as the array manufacturing technology and the intended application. In this section we formalize the most important stability and non-interaction constraints using the hybridization model in (Ben-Dor et al., 2000).

Hybridization model. Hybridization affinity between two oligonucleotides is commonly characterized using the *melting temperature*, defined as the temperature at which exactly half of the duplexes are in hybridized state. However, accurate melting temperature estimation is computationally expensive, e.g., estimating the melting temperature between two non-complementary oligonucleotides using the near-neighbor model of SantaLucia (SantaLucia, 1998) is an NP-hard problem (Kaderali, 2001). A conservative hybridization model based on the observation that stable hybridization requires the formation of an initial *nucleation complex* between two perfectly complementary substrings of the two oligonucleotides was formalized by (Ben-Dor et al., 2000; BenDor et al., 2004). For nucleation complexes, hybridization affinity is modeled using the classical *2-4 rule* (Wallace et al., 1979), according to which the melting temperature of the duplex formed by an oligonucleotide with its complement is proportional to the sum between the number of *weak* bases (i.e., A and T) and twice the number of *strong* bases (i.e., G and C).

Following (Ben-Dor et al., 2000), we define the *weight* $w(x)$ of a DNA string $x = a_1a_2 \ldots a_k$ by $w(x) = \sum_{i=1}^{k} w(a_i)$, where $w(\mathsf{A}) = w(\mathsf{T}) = 1$ and $w(\mathsf{C}) = w(\mathsf{G}) = 2$. Throughout the paper we assume the following *c-token hybridization model*: hybridization between two oligonucleotides takes place only if one contains as substring the complement of a substring of weight c or more of the other, where c is a given constant. The *complement* of a string $x = a_1a_2 \ldots a_k$ over the DNA alphabet $\{\mathsf{A}, \mathsf{C}, \mathsf{T}, \mathsf{G}\}$ is defined as $\bar{x} = b_1b_2 \ldots b_k$, where b_i is the Watson-Crick complement of a_{k-i+1}.

Hybridization stability. Current industry designs require a predetermined tag length l, e.g., GenFlex universal tag arrays manufactured by Affymetrix use $l = 20$ (Affymetrix, Inc., 2001). The model proposed in (Ben-Dor et al., 2000) allows tags of unequal length and instead require a minimum tag weight of h, for a given constant h. In this paper we consider both types of stability constraints, and use the parameter $\alpha \in \{l, h\}$ to denote the specific model used for hybridization stability.

Pairwise non-interaction constraints. A basic constraint in this category is that every antitag must not hybridize to non-complementary tags (Ben-Dor et al., 2000). For a DNA string x and a set of tags $\mathcal{T}$, let $N_{\mathcal{T}}(x)$ denote the number of tags in $\mathcal{T}$ that contain x as a substring. Using the c-token hybridization model, this antitag-to-tag hybridization constraint is formalized as follows:

(C) For every feasible tag set $\mathcal{T}$, $N_{\mathcal{T}}(x) \leq 1$ for every DNA string x of weight c or more.

In many assays based on universal tag arrays it is also required to prevent antitag-to-antitag hybridization, since the formation of antitag-to-antitag duplexes or antitag hair-pin structures prevents reporter probes from performing their function in the solution-based hybridization steps (Brenne-

man & Condon, 2002; Mandoiu et al., 2005). The combined constraints on antitag hybridization are formalized as follows

($\bar{C}$**)** For every feasible tag set $\mathcal{T}$, $N_{\mathcal{T}}(x) + N_{\mathcal{T}}(\bar{x}) \leq 1$ for every DNA string x of weight c or more.

In the following we use the parameter $\beta \in \{C, \bar{C}\}$ to specify the type of pairwise non-interaction constraints.

Substring occurrences within a tag. Previous works on DNA tag set design (Ben-Dor et al., 2000; Mandoiu et al., 2005) have imposed the following c*-token uniqueness constraint* in addition to constraints (C) and $(\bar{C})$: a DNA string of weight c or more can appear as a substring of a feasible tag at most once. This uniqueness constraint simplifies analysis – e.g., it is the key property enabling the DeBruijn sequence based heuristics in (Ben-Dor et al., 2000)) – but is *not* required for ensuring correct assay functionality. In the following we will use the parameter $\gamma \in \{1, multiple\}$ to specify whether or not the c-token uniqueness constraint is enforced.

Problem formulation. For every $\alpha \in \{l, h\}$, $\beta \in \{C, \bar{C}\}$, and $\gamma \in \{1, multiple\}$, the *maximum tag set design problem with constraints* α, β, γ, denoted MTSDP($\alpha|\beta|\gamma$), is the following: given constants c and l/h, find a tag set of maximum cardinality satisfying constraints α, β, and γ.

Previous work on tag set design. The c-token model for oligonucleotide hybridization and the MTSDP($h|C|1$) problem are formalized in (Ben-Dor et al., 2000). Ben-Dor et al. also establish a constructive upper bound on the optimal number of tags for this formulation, and give a nearly optimal tag selection algorithm based on DeBruijn sequences. Similar upper bounds are established

for the MTSDP($l|C|1$) and MTSDP($*|\bar{C}|1$) problems in (Mandoiu et al., 2005), which also extends a simple alphabetic tree search strategy originally proposed in (Morris et al., 2002) to handle all MTSDP($*|*|1$) problem formulations. Tag selection with both tag-to-antitag and antitag-to-antitag non-interaction constraints is also considered in (Kaderali et al., 2003) using the near-neighbor hybridization model. The algorithm proposed in (Kaderali et al., 2003) generates random tags and checks compatibility with previously selected tags until enough tags are selected. This simple approach works well for applications where the number of required tags is relatively small such as SNP genotyping by multiplex single-base extension and flow cytometry using microsphere arrays, but does not provide any guarantees on the number of generated tags.

For a comprehensive survey of hybridization models, results on associated formulations for the tag set design problem, and further motivating applications in the area of DNA computing, we direct the reader to (Brenneman & Condon, 2002). Algorithms for the related tag assignment problem – in which the goal is to assign an antitag to each primer such that no unwanted (e.g., primer-to-tag) hybridization takes place – can be found in (BenDor et al., 2004; Kaderali et al., 2003; Mandoiu et al., 2005).

3 Integer Linear Programming Formulations for MTSDP($*|C|1$)

Before stating our integer linear program formulations, we introduce some additional notations.

Following (Ben-Dor et al., 2000), a DNA string x of weight c or more is called a *c-token* if all its proper suffixes have weight strictly less than c. Clearly, it suffices to enforce constraints (C) or

$(\bar{C})$ for all c-tokens x. Let N denote the number of c-tokens, and $\mathcal{C} = \{c_1, \ldots, c_N\}$ denote the set of all c-tokens. The results in (Ben-Dor et al., 2000) imply that $N = \Theta((1+\sqrt{3})^c)$. Note that the weight of a c-token can be either c or $c+1$, the latter case being possible only if the c-token starts with a strong base (G or C). We let $\mathcal{C}_0 \subseteq \mathcal{C}$ denote the set of c-tokens of weight $c+1$ that end with a weak base, i.e., c-tokens of the form S$<c-2>$W, where W (S) denotes a weak (strong) base, and $<c-2>$ denotes an arbitrary string of weight $c-2$. We also let $\mathcal{C}_2 \subseteq \mathcal{C}$ denote the set of c-tokens of weight c that end with a strong base, i.e., c-tokens of the form $<c-2>$S.

Clearly, there is at most one c-token ending at every letter of a tag. It is easy to see that each c-token $x \in \mathcal{C}_0$ contains a proper prefix which is itself a c-token, and therefore x cannot be the first c-token of a tag, i.e., cannot be the c-token with the leftmost ending. All other c-tokens can appear as first c-tokens. When a c-token in $C \setminus (\mathcal{C}_0 \cup \mathcal{C}_2)$ is the first in a tag, then it must be a prefix of the tag. On the other hand, tokens in $\mathcal{C}_2$ can be first both in tags that they prefix and in tags in which they are preceded by a weak base not covered by any c-token.

The ILP formulation for MTSDP($l|C|1$) uses an auxiliary directed graph $G = (V, E)$ with $V = \{s, t\} \cup \bigcup_{1 \le i \le N} V_i$, where $V_i = \{v_i^k \mid |c_i| \le k \le l\}$. G has a directed arc from v_i^k to v_j^{k+1} for every triple i, j, k with $|c_i| \le k \le l-1$ for which c_j can be obtained from c_i by appending a single nucleotide and removing the maximal prefix that still leaves a valid c-token. Finally, G has an arc from s to every $v \in V_{first}$, where $V_{first} = \{v_i^{|c_i|} \mid c_i \in \mathcal{C} \setminus \mathcal{C}_0\} \cup \{v_i^{|c_i|+1} \mid c_i \in \mathcal{C}_2\}$, and an arc from v_i^l to t for every $1 \le i \le N$. Notice that G has $O(lN)$ vertices. Furthermore, since s has outdegree less than $2N$ and every other vertex has outdegree at most 4, it follows that G has $O(lN)$ arcs.

We claim that, for $c \le l$, MTSDP($l|C|1$) can be reformulated as the problem of finding the maximum number of s-t paths in G that collectively visit at most one vertex v_i^k for every i. Indeed, let P be an s-t path and v_i^k be the vertex following s in P. If $k = |c_i|$, we associate to P the tag

obtained by concatenating c_i with the last letters of the c-tokens corresponding to the subsequently visited vertices, until reaching t. Otherwise, we must have $c_i \in \mathcal{C}_2$ and $k = |c_i| + 1$. In this case we associate to P the two tags obtained by concatenating either A or T with c_i and with the last letters of subsequently visited c-tokens. The claim follows by observing that at most one of the tags associated with each path can be used in a feasible solution.

Our ILP formulation can be viewed as a generalized version of the maximum integer flow problem in which unit capacity constraints are imposed on *sets of vertices* of G instead of individual vertices. The formulation uses 0/1 variables x_v and y_e for every vertex $v \in V \setminus \{s, t\}$, respectively arc $e \in E$. These variables are set to 1 if the corresponding vertex or arc is visited by an s-t path corresponding to a selected tag. Let $in(v)$ and $out(v)$ denote the set of arcs entering, respectively leaving vertex v. The integer program can then be written as follows:

$$\text{maximize} \quad \sum_{v \in V_{first}} x_v \tag{1}$$

subject to

$$x_v = \sum_{e \in in(v)} y_e = \sum_{e \in out(v)} y_e, \quad v \in V \setminus \{s, t\} \tag{2}$$

$$\sum_{v \in V_i} x_v \leq 1, \qquad 1 \leq i \leq N \tag{3}$$

$$x_v, y_e \in \{0, 1\}, \qquad v \in V \setminus \{s, t\}, e \in E \tag{4}$$

Constraints (2) ensure that variables y_e set to 1 correspond to a set of s-t paths, and that a variable x_v is set to 1 if and only if one of these paths passes through v. Antitag-to-tag hybridization constraints (C) and c-token uniqueness are enforced by (3). Finally, the objective (1) corresponds to maximizing the number of selected s-t paths, since every arc out of s goes to a vertex of V_{first}.

For a token $c_i = c_j a \in \mathcal{C}_0$, where $a \in \{\mathsf{A}, \mathsf{T}\}$, let $\hat{c_i} = c_j \bar{a}$. Since both c_i and $\hat{c_i}$ contain token c_j as a prefix, it follows that at most one of them can appear in $\mathcal{T}$. Therefore, the following valid

inequality can be added to the ILP formulation (1)–(4) to improve its integrality gap (i.e., the gap between the value of the optimum integer solution and that of the optimal fractional relaxation):

$$\sum_{v \in V_i \cup V_j} x_v \leq 1, \qquad c_i \in \mathcal{C}_0,\ c_j = \widehat{c}_i,\ i < j \tag{5}$$

The formulation of MTSDP($h|C|1$) has exactly the same objective and constraints for a slightly modified graph G. Let us define the *tail weight* of a c-token c_i, denoted $tail(c_i)$, as the weight of the last letter of c_i. Also, let $h_i = h$ if c_i has a tail weight of 1 and $h_i = h + 1$ if c_i has a tail weight of 2. We will require that every tag ending with token c_i has total weight of at most h_i – it is easy to see that this constraint is not affecting the size of the optimum tag set. The modified graph G has vertex set $V = \{s, t\} \cup \bigcup_{1 \leq i \leq N} V_i$, where $V_i = \{v_i^k \mid w(c_i) \leq k \leq h_i\}$. G contains a directed arc from v_i^k to $v_j^{k+tail(i)}$ for every triple i, j, k with $|c_i| \leq k \leq h_i - tail(c_i)$ for which c_j can be obtained from c_i by appending a single nucleotide and removing the maximal prefix that still leaves a valid c-token. Finally, G contains arcs from s to every $v \in V_{first}$, where V_{first} is now equal to $\{v_i^{w(c_i)} \mid c_i \in \mathcal{C} \setminus \mathcal{C}_0\} \cup \{v_i^{w(c_i)+1} \mid c_i \in \mathcal{C}_2\}$, plus arcs from every v_i^k to t for every $1 \leq i \leq N$ and $h_i - tail(c_i) < k \leq h_i$.

4 Approximation Algorithms for MTSDP($*|C|1$)

The ILP formulation for MTSDP($l|C|1$) (MTSDP($h|C|1$)) has $O(lN)$ (respectively $O(hN)$) variables and constraints, where $N = \Theta((1 + \sqrt{3})^c)$ is the number of c-tokens. For small values of c these formulations can be solved to optimality by general purpose optimization packages. However, as shown in Section 6, even state-of-the-art solvers such as CPLEX require a prohibitive amount of time for values of c greater than 8. In this section we describe a faster algorithm for computing near-optimal tag sets.

The algorithm is based on an equivalent ILP formulation of MTSDP($*|C|1$) using "path" instead of "arc" variables. Let $\mathcal{P}$ be the set of all s-t paths in the auxiliary graph G defined as in the previous section. Using a 0/1 variable x_p for every path $p \in \mathcal{P}$, MTSDP($*|C|1$) can be formulated as follows:

$$\text{maximize} \quad \sum_{p \in \mathcal{P}} x_p \tag{6}$$

subject to

$$\sum_{p \in \mathcal{P}} |p \cap V_i| x_p \leq 1, \qquad 1 \leq i \leq N \tag{7}$$

$$x_p \in \{0, 1\}, \qquad p \in \mathcal{P} \tag{8}$$

The fractional relaxation of ILP (6)-(8) is obtained by replacing integrality constraints (8) with

$$x_p \geq 0, \qquad p \in \mathcal{P} \tag{9}$$

The optimum solution of the fractional relaxation can be efficiently approximated within any desired accuracy using the algorithm in Figure 1, which is a specialization of the approximation algorithm for packing linear programs in (Garg & Konemann, 1998). Briefly, the algorithm starts by assigning a small weight $y_i = \delta$ to every set V_i. Then, the algorithm repeatedly computes minimum-weight s-t paths in G, where the weight of a node is given by the weight y_i of the corresponding set V_i. For every minimum-weight path p, the y_i's corresponding to visited sets V_i are multiplied by a factor of $(1 + \varepsilon \frac{|p \cap V_i|}{\max_i |p \cap V_i|})$. Finally, the algorithm stops when the weight of every s-t path is greater than or equal to 1.

Since the auxiliary graph $G = (V, E)$ is directed and acyclic, the minimum weight path can be computed in $O(|V| + |E|)$ time. Therefore, using the fact that G has $O(lN)$ vertices and arcs for MTSDP($l|C|1$), and $O(hN)$ vertices and arcs for MTSDP($h|C|1$), Theorem 3.1 of (Garg & Konemann, 1998) gives:

Theorem 1 *The algorithm in Figure 1 computes a* $(1-\varepsilon)^2$*-approximation to the fractional relaxation in* $O(lN^2\lceil\frac{1}{\varepsilon}\log_{1+\varepsilon}N\rceil)$ *time for MTSDP(*$l|C|1$*), and in* $O(hN^2\lceil\frac{1}{\varepsilon}\log_{1+\varepsilon}N\rceil)$ *time for MTSDP(*$h|C|1$*).*

The fractional solution computed by the Garg and Könemann algorithm is then used to construct a feasible set of tags using a simple method that has been shown in (Dragan et al., 2002) to work better in practice than classical randomized rounding (Raghavan & Thomson, 1987), particularly when starting from poor approximate solutions such as those obtained by running the algorithm in Figure 1 with a large value of ε. We simply save the list of s-t paths selected as minimum-weight paths by the Garg and Könemann algorithm (excluding minimum-weight paths that visit some set V_i more than once, since such paths do not correspond to valid tags) and then, traversing the list in reverse order, we sequentially pick tags that correspond to paths visiting only sets V_i not yet appearing in the already picked tags. Finally, we mark all c-tokens of picked tags as unavailable, and augment the set $\mathcal{T}$ of picked tags with the additional tags found by running the alphabetic tree search algorithm in (Mandoiu et al., 2005).

Note that by running the alphabetic tree search algorithm at the end, we guarantee that the set $\mathcal{T}$ of selected tags is *maximal*, i.e., there is no tag t such that $\mathcal{T}\cup\{t\}$ remains feasible. Hence, every tag of an optimal solution must share at least one c-token with tags in $\mathcal{T}$. Since every tag of $\mathcal{T}$ has at most $l-c/2+1$ c-tokens, it follows that the size of $\mathcal{T}$ is within a factor of $l-c/2+1$ of the size of an optimum MTSDP($l|C|1$) solution. Similarly, the approximation factor of the algorithm when applied to MTSDP($h|C|1$) is no more than $h-c/2+2$.

5 Algorithms for MTSDP($*|*|multiple$)

In the following we describe two algorithms for MTSDP($l|C|multiple$); both algorithms can be easily adjusted to handle the other MTSDP($*|*|multiple$) variants. The first algorithm (see Figure 2 for a detailed pseudocode) is similar to the alphabetic tree search algorithms proposed for MTSDP($l|C|1$) in (Mandoiu et al., 2005). The algorithm performs an alphabetical traversal of a 4-ary tree representing all 4^l possible tags, skipping over subtrees rooted at internal vertices that correspond to tag prefixes including unavailable c-tokens. The difference from the MTSDP($l|C|1$) algorithm in (Mandoiu et al., 2005) lies in the strategy used to mark c-tokens as unavailable. While the algorithm in (Mandoiu et al., 2005) marks a c-token C as unavailable as soon as it incorporates it in the current tag prefix (changing C's status back to "available" when forced to backtrack past C's tail), the algorithm in Figure 2 marks a c-token as unavailable only when a complete tag is found.

We call a tag t *periodic* if t is the length l prefix of an infinite string x^∞, where x is a DNA string with $|x| < |t|$. (Note that a periodic tag t is not necessarily the concatenation of an integer number of copies of its period x, as in the standard definition of string periodicity (Lothaire, 1983).)

The following lemma shows that tag set design algorithms can restrict the search to two simple classes of tags.

Lemma 1 *For every c and l, there exists an optimal tag set $\mathcal{T}$ in which every tag has the uniqueness property or is periodic. (Note that the two classes of tags are not disjoint, since there are periodic tags satisfying the uniqueness property.)*

Proof. Let $\mathcal{T}$ be an optimal tag set. Assume that $\mathcal{T}$ contains a tag t that does not have the uniqueness property, and let $c_{i_1}, \ldots, c_{i_k}$ be the sequence of c-tokens occurring in t, in left to right

order. Since t does not have the uniqueness property, there exist indices $1 \leq j < j' \leq i_k$ such that $c_{i_j} = c_{i_{j'}}$. Let t' be the periodic tag formed by taking the first l letters of the infinite string with c-token sequence $(c_{i_j}, \ldots, c_{i_{j'-1}})^\infty$. Since c-tokens $c_{i_j}, \ldots, c_{i_{j'-1}}$ do not appear in the tags of $\mathcal{T} \setminus \{t\}$, it follows that $(\mathcal{T} \setminus \{t\}) \cup \{t'\}$ is also optimal. Repeated application of this operation yields the lemma. □

Note that a periodic tag whose shortest period has length p contains as substrings exactly p c-tokens, while tags with the uniqueness property contain between $l - c + 1$ and $l - c/2 + 1$ c-tokens. Therefore, of the two classes of tags in Lemma 1, periodic tags (particularly those with short periods) make better use of the limited number of available c-tokens.

Each periodic tag corresponds to a directed cycle in the graph H_c which has $\mathcal{C}$ as its vertex set, and in which a token c_i is connected by an arc to token c_j iff c_i and c_j can appear consecutively in a tag, i.e., iff c_j is obtained from c_i by appending a single nucleotide and removing the maximal prefix that still leaves a valid c-token. Clearly, a vertex-disjoint packing of n cycles in H_c yields a feasible solution for MTSDP($l|C|multiple$) consisting of n tags, since we can extract at least one tag of length l from each cycle, and tags extracted from different cycles do not have common c-tokens. This motivates the following:

MAXIMUM VERTEX-DISJOINT DIRECTED CYCLE PACKING Problem: Given a directed graph G, find a maximum number of vertex-disjoint directed cycles in G.

The next theorem shows that MAXIMUM VERTEX-DISJOINT DIRECTED CYCLE PACKING in arbitrary graphs is unlikely to admit a polynomial approximation scheme. A stronger inapproximability results was recently established for arbitrary graphs by (Salavatipour & Verstraëte, 2005), who proved that there is no $O(\log^{1-\varepsilon} n)$-approximation for MAXIMUM VERTEX-DISJOINT DI-

RECTED CYCLE PACKING unless $NP \subseteq DTIME(2^{polylogn})$. On the positive side, Salavatipour and Verstraëte showed that MAXIMUM VERTEX-DISJOINT DIRECTED CYCLE PACKING can be approximated within a factor of $O(\sqrt{n})$ via linear programming techniques, matching the best approximation factor known for the arc-disjoint version of the problem (Krivelevich et al., 2005).

Theorem 2 MAXIMUM VERTEX-DISJOINT DIRECTED CYCLE PACKING *is APX-hard even for regular directed graphs with in-degree and out-degree of 2.*

Proof. We use a reduction from the MAX-2-SAT-3 problem, similar to the one in (Caprara et al., 2003). An instance ϕ of MAX-2-SAT-3 consists of a set $\{c_1, \ldots, c_m\}$ of disjunctive clauses over a ground set $\{x_1, \ldots, x_n\}$ of variables. Each clause consists of at most 2 literals (variables or negations of variable), and each variable appears in at most 3 clauses, counting both negated and non-negated occurrences. The objective is to find a truth assignment that satisfies as many of the clauses as possible. It is known that MAX-2-SAT-3 is APX-hard (Ausiello et al., 1999; Berman & Karpinski, 1999).

Let m_i denote the number of occurrences of variable x_i in a given instance of MAX-2-SAT-3. We construct, in polynomial time, a directed graph $G(\phi)$ as follows. For each variable x_i we add to G a directed cycle C_i of length $4m_i$, plus $2m_i$ additional vertices alternatively labeled by x_i and $\bar{x_i}$, used to close a directed cycle of length 3 with each arc of C_i, as in Figure 3(a). For each unary clause we pick a distinct vertex labeled by the *negation* of the respective literal and attach a loop to it. Finally, for each 2-literal clause we pick 2 vertices labeled by the negations of its literals, again without reusing labeled vertices between clauses, and use a new vertex to connect them via two length-2 cycles as in Figure 3(b). Note that, for every i, at least $2\sum_{i=1}^{n} m_i$ of the labeled vertices remain incident to a single cycle; we will refer to these as "free" labeled vertices.

We claim that every truth assignment that makes k clauses of ϕ true can be converted in polynomial time into a set of $k+2\sum_{i=1}^{n} m_i$ vertex disjoint cycles of $G(\phi)$, and vice-versa. Indeed, for a given truth assignment, select (1) the $2m_i$ length-3 cycles passing through nodes labeled by $\bar{x}_i$ for every variable x_i that is set to true, (2) the $2m_i$ length-3 cycles passing through nodes labeled by $\bar{x}_i$ for every variable x_i that is set to false, and (3) the loop or length-2 cycle passing through a labeled node corresponding to a *false* literal. It is easy to verify that these cycles are vertex-disjoint.

Conversely, let $\mathcal{C}$ be a set of $k+2\sum_{i=1}^{n} m_i$ vertex disjoint cycles of $G(\phi)$. If any of the cycles C_i is in $\mathcal{C}$, we replace it by the length-3 cycle passing through a free labeled vertex. Similarly, if any of the cycles in $\mathcal{C}$ visits two of the arcs of a 3-cycle (or one of the arcs of a 2-cycle), we replace it by the 3-cycle (respectively 2-cycle) itself. After this transformation we have a set of $k + 2\sum_{i=1}^{n} m_i$ vertex-disjoint loops, 2-cycles, and 3-cycles. We say that a set of cycles is *consistent* if only one of the labels x_i, $\bar{x}_i$ appear in $\mathcal{C}$ for every i. If $\mathcal{C}$ is consistent, we choose a truth assignment that makes all literals corresponding to labels in $\mathcal{C}$ true. It is easy to see that at least k of the cycles in $\mathcal{C}$ must be loops and 2-cycles, and clauses corresponding to these cycles are satisfied by the above truth assignment.

Otherwise, we make $\mathcal{C}$ consistent by repeating the following transformation. Let i be an index for which both x_i and $\bar{x}_i$ appear in $\mathcal{C}$. Without loss of generality, assume that x_i appears in only one clause of ϕ (recall that, together, x_i and $\bar{x}_i$ can appear in at most 3 clauses). It follows that there is a single loop or 2-cycle $C \in \mathcal{C}$ visiting a vertex labeled by $\bar{x}_i$ – all other vertices labeled by $\bar{x}_i$ are free. Since the x_i's and $\bar{x}_i$'s alternate around C_i, the cycles going through vertices labeled by $\bar{x}_i$ can be replaced by at least the same number of 3-cycles going through vertices labeled by x_i.

To complete the proof of the theorem, notice that the optimum number of satisfiable clauses, k_{opt}, is at least $m/2$, since we can repeatedly assign a variable such that at least half of the clauses

containing it are satisfied. Hence, $\sum_{i=1}^{n} m_i \leq 2m \leq 4k_{opt}$. If there exists a polynomial time algorithm with an approximation factor of $\frac{1}{1-\varepsilon}$ for MAXIMUM VERTEX-DISJOINT DIRECTED CYCLE PACKING, we can run it on $G(\phi)$ to get a set $\mathcal{C}$ of at least $k + 2\sum_{i=1}^{n} m_i \geq \frac{1}{1-\varepsilon}(k_{opt} + 2\sum_{i=1}^{n} m_i)$ vertex disjoint cycles, and then convert $\mathcal{C}$ as above into a truth assignment satisfying $k \geq \frac{1+8\varepsilon}{1-\varepsilon} k_{opt}$ clauses of ϕ. □

We use a simple greedy algorithm to solve MAXIMUM VERTEX-DISJOINT DIRECTED CYCLE PACKING for the graph H_c: we enumerate possible tag periods in pseudo-lexicographic order, and check for each period if all c-tokens are available for the resulting tag. We refer to this algorithm as the *greedy cycle packing algorithm*, since it is equivalent to packing cycles greedily in order of length.

6 Experimental results

Tables 1 and 2 give empirical results for the MTSDP($l|C|1$) and MTSDP($h|C|1$) problems respectively. We give the number of selected tags and runtimes for the following three algorithms:

- the tree search algorithm in (Mandoiu et al., 2005),
- the Garg-Könemann based algorithm described in Section 4 (denoted LP approx) ran with $\varepsilon = 0.5$, and
- the CPLEX 9.0 commercial solver applied to ILP (1)–(4).

All compared algorithms were run using a single CPU of a dual 2.8 GHz Dell PowerEdge 2600 Linux server with 4Gb of main memory. Missing LP/ILP entries did not complete in 10 hours.

Optimum tag sets are found by CPLEX for small values of c. However, even computing the optimum fractional relaxation of ILP (1)–(4) is impractical for c greater than 8. In contrast, the Garg-Könemann based algorithm is much more scalable than CPLEX, and generally produces better solutions than running the tree search algorithm alone, although we run it with a very large value for ε.

To help assessing the quality of the compared algorithms when the optimum solution is not available, we also include in Tables 1 and 2 the c-token count upper bound established for MTSDP($l|C|1$) in (Mandoiu et al., 2005) and the tail-weight upper bound established for MTSDP($h|C|1$) in (Ben-Dor et al., 2000), as well as the value of the fractional (LP) relaxation of ILP (1)–(4). For all cases where the optimum ILP solution could be computed, the difference between the optimal fractional and integer solution values is smaller than 1, indicating that the LP solution is a very tight upper bound. Furthermore, ILP results confirm the high quality of the upper bound established for MTSDP($h|C|1$) in (Ben-Dor et al., 2000); the upper bound established in (Mandoiu et al., 2005) for MTSDP($l|C|1$) appears to be somehow weaker.

Tables 3 and 4 give the results obtained for MTSDP($*|*|multiple$) by the alphabetic tree search algorithm in Figure 2 respectively by the greedy cycle packing algorithm (in our implementation, we impose an upper bound of 15 on the length of the cycles that we try to pack) followed by running the alphabetic tree search algorithm with the c-tokens occurring in the selected cycles already marked as unavailable. Performing cycle packing significantly improves the results compared to running the alphabetic tree search algorithm alone; as shown in the tables, most of the resulting tags are found in the cycle packing phase of the combined algorithm.

Across all instances, the combined algorithm increases the number of tags by at least 40% compared to the best available MTSDP($*|*|1$) algorithm –the improvement is much higher for

smaller values of c. Quite notably, although the number of tags is increased, the tag sets found by the combined algorithm use a *smaller* total number of c-tokens. Thus, these tag sets are less likely to cross-hybridize to the primers used in the reporter probes, enabling higher tag utilization rates during tag assignment (Hundewale et al., 2005).

7 Conclusions

In this paper we proposed new solution methods for designing optimal and near-optimal tag sets for universal DNA arrays. Most notably, we have shown that the use of periodic tags leads to over 40% more tags compared to best previous methods. Our algorithms use simple combinatorial ideas and greedy strategies that can be easily extended to handle more sophisticated hybridization models such as the near-neighbor model of (SantaLucia, 1998), and can incorporate additional practical design constraints, such as preventing the formation of hairpin secondary structures, or disallowing specific nucleotide sequences such as runs of 4 identical nucleotides (Morris et al., 2002).

In ongoing work we seek to extend our methods to emerging applications of universal tag arrays in microfluidics-based labs-on-a-chip, as well as DNA-mediated assembly of nanoscale devices such as carbon-nanotube-based field-effect transistors (Hazani et al., 2004). An interesting open problem is to find tight upper bounds and exact methods for the MTSDP($*|*|multiple$) formulations. Settling the approximation complexity of MAXIMUM VERTEX-DISJOINT DIRECTED CYCLE PACKING is another interesting problem.

Acknowledgments

This work has been supported in part by a Large Grant from the University of Connecticut's Research Foundation.

References

Affymetrix, Inc. (2001). Geneflex tag array technical note no. 1, available online at http://www.affymetrix.com/support/technical/technotes/genflex_technote.pdf.

Ausiello, G., Protasi, M., Marchetti-Spaccamela, A., Gambosi, G., Crescenzi, P., & Kann, V. (1999). *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Springer-Verlag New York, Inc.

Ben-Dor, A., Karp, R., Schwikowski, B., & Yakhini, Z. (2000). Universal DNA tag systems: a combinatorial design scheme. *Journal of Computational Biology*, *7*, 503–519.

BenDor, A., Hartman, T., Schwikowski, B., Sharan, R., & Yakhini, Z. (2004). Towards optimally multiplexed applications of universal arrays. *Journal of Computational Biology*, *11*, 477-493.

Berman, P., & Karpinski, M. (1999). On some tighter inapproximability results. *Proc. 26th Intl. Colloquium on Automata, Languages and Programming* (pp. 200–209).

Brenneman, A., & Condon, A. (2002). Strand design for biomolecular computation. *Theor. Comput. Sci.*, *287*, 39–58.

Brenner, S. (1997). Methods for sorting polynucleotides using oligonucleotide tags. *US Patent 5,604,097*.

Caprara, A., Panconesi, A., & Rizzi, R. (2003). Packing cycles in undirected graphs. *Journal of Algorithms, 48,* 239–256.

Dragan, F., Kahng, A., Mandoiu, I., Muddu, S., & Zelikovsky, A. (2002). Provably good global buffering by generalized multiterminal multicommodity flow approximation. *IEEE Transactions on Computer-Aided Design, 21,* 263–274.

Garg, N., & Konemann, J. (1998). Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *Proceedings of the 39th IEEE Annual Symposium on Foundations of Computer Science* (pp. 300–309).

Gerry, N., Witowski, N., Day, J., Hammer, R., Barany, G., & Barany, F. (1999). Universal DNA microarray method for multiplex detection of low abundance point mutations. *J. Mol. Biol., 292,* 251–262.

Hazani, M., Shvarts, D., Peled, D., Sidorov, V., & Naaman, R. (2004). Self-assembled carbon-nanotube-based field-effect transistors. *Applied Physics Letters, 85,* 5025–5027.

Hirschhorn, J., Sklar, P., Lindblad-Toh, K., Lim, Y.-M., Ruiz-Gutierrez, M., Bolk, S., Langhorst, B., Schaffner, S., Winchester, E., & Lander, E. (2000). SBE-TAGS: An array-based method for efficient single-nucleotide polymorphism genotyping. *PNAS, 97,* 12164–12169.

Hundewale, N., Mandoiu, I., Prajescu, C., & Zelikovsky, A. (2005). Integrated design flow for universal DNA tag arrays. *9th Annual International Conference on Research in Computational Molecular Biology (RECOMB) Poster Book* (pp. 141–142).

Kaderali, L. (2001). Selecting target specific probes for DNA arrays. Master thesis, Köln University.

Kaderali, L., Deshpande, A., Nolan, J.P., & White, P.S. (2003). Primer-design for multiplexed genotyping. *Nucleic Acids Res., 31,* 1796–1802.

Krivelevich, M., Nutov, Z., & Yuster, R. (2005). Approximation algorithms for cycle packing problems. *Proc. ACM-SIAM Annual Symposium on Discrete Algorithms* (pp. 556–561).

Lothaire, M. (1983). *Combinatorics on words*, vol. 17 of *Encylopedia of Mathematics and Its Applications*, xix+238. Addison-Wesley.

Morris, M., Shoemaker, D., Davis, R., & Mittmann, M. (2002). Selecting tag nucleic acids. *U.S. Patent 6,458,530 B1*.

Mandoiu, I., Prajescu, C., & Trinca, D. (2005). Improved tag set design and multiplexing algorithms for universal arrays. *LNCS Transactions on Computational Systems Biology, II*, 124–137.

Raghavan, P., & Thomson, C. (1987). Randomized rounding. *Combinatorica, 7*, 365–374.

Salavatipour, M., & Verstraëte, J. (2005). Disjoint cycles: Integrality gap, hardness, and approximation. *Proc. 11th Conference on Integer Programming and Combinatorial Optimization (IPCO), to appear* (pp. 51–65).

SantaLucia, J. (1998). A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics. *Proc. Natl. Acad. Sci. USA, 95*, 1460–1465.

Wallace, R., Shaffer, J., Murphy, R., Bonner, J., Hirose, T., & Itakura, K. (1979). Hybridization of synthetic oligodeoxyribonucleotides to phi chi 174 DNA: the effect of single base pair mismatch. *Nucleic Acids Res., 6*, 6353–6357.

Table 1: ILP results for MTSDP($l|C|1$), i.e., tag set design with specified tag length l, antitag-to-tag hybridization constraints, and a unique copy of each c-token allowed in a tag.

l	c	# Selected Tags			Upper Bounds		CPU Seconds			
		Tree search	LP approx	ILP	LP	c-token count	Tree search	LP approx	LP	ILP
10	4	7	7	8	8.57	9	0.00	0.27	0.13	0.71
	5	23	25	28	28.00	29	0.00	0.32	2.27	5.85
	6	67	79	85	85.60	96	0.00	0.57	11.40	98.25
	7	196	232	259	259.67	328	0.00	3.11	86.70	586.67
	8	655	793	853	853.33	1194	0.00	63.01	552.74	4321.66
	9	2359	2703	–	–	4896	0.01	841.64	–	–
	10	9072	10144	–	–	26752	0.04	11019.64	–	–
20	4	3	3	3	3.53	3	0.00	0.32	1.05	58.46
	5	9	9	10	10.50	11	0.00	0.40	13.72	381.33
	6	26	26	29	29.87	32	0.00	1.26	182.96	12448.61
	7	75	75	–	88.00	93	0.00	4.79	2675.68	–
	8	213	220	–	257.23	275	0.00	49.21	134525.81	–
	9	600	641	–	–	816	0.00	624.16	–	–
	10	1667	1854	–	–	2432	0.04	7717.13	–	–

Table 2: ILP results for MTSDP($h|C|1$), i.e., tag set design with specified minimum tag weight h, antitag-to-tag hybridization constraints, and a unique copy of each c-token allowed in a tag.

h	c	# Selected Tags			Upper Bounds		CPU Seconds			
		Tree search	LP approx	ILP	LP	tail-weight	Tree search	LP approx	LP	ILP
15	4	6	5	7	7.00	7	0.00	0.34	0.45	9.04
	5	18	18	21	21.09	21	0.00	0.38	5.66	117.62
	6	47	52	63	63.20	63	0.00	0.89	54.43	2665.39
	7	149	155	192	192.00	192	0.00	5.49	544.95	3644.85
	8	460	480	–	588.00	590	0.00	99.21	7153.87	–
	9	1197	1608	–	–	1842	0.00	1788.07	–	–
	10	3669	4947	–	–	5872	0.07	24223.92	–	–
28	4	3	3	3	3.30	3	0.00	0.46	1.88	132.78
	5	8	8	9	9.67	9	0.00	0.60	34.66	1137.21
	6	22	22	27	27.48	27	0.00	1.26	392.42	18987.09
	7	64	63	–	78.55	78	0.00	8.89	7711.41	–
	8	175	182	–	224.76	224	0.00	111.63	850642.82	–
	9	531	515	–	–	644	0.00	1606.85	–	–
	10	1428	1491	–	–	1854	0.02	26728.47	–	–

Table 3: Results for MTSDP($*|C|multiple$), i.e., tag set design with antitag-to-tag hybridization constraints and multiple copies of a c-token allowed in a tag.

l/h	c	One c-token copy		Multiple c-token copies				
		LP approx		Tree search		Cycle packing + Tree search		
		tags	c-tokens	tags	c-tokens	tags	c-tokens	% cyclic
$l = 20$	4	3	51	14	59	17	40	100.0
	5	9	146	31	165	40	140	100.0
	6	26	402	53	433	72	293	98.6
	7	75	1096	124	1179	178	928	99.4
	8	220	3014	281	3095	383	2411	97.1
	9	641	8322	711	8230	961	7102	96.9
	10	1854	22693	1835	21400	2344	19691	95.1
$h \geq 28$	4	3	58	14	61	17	40	100.0
	5	8	151	32	174	40	140	100.0
	6	22	391	44	432	72	300	98.6
	7	63	1083	118	1200	178	934	99.4
	8	182	2996	239	3037	379	2405	96.6
	9	515	8025	632	8622	943	6969	96.5
	10	1491	22183	1570	22145	2260	19270	94.1

Table 4: Results for MTSDP($*|\bar{C}|multiple$), i.e., tag set design with both antitag-to-tag and antitag-to-antitag hybridization constraints and multiple copies of a c-token allowed in a tag.

l/h	c	One c-token copy		Multiple c-token copies				
		Tree search		Tree search		Cycle packing + Tree search		
		tags	c-tokens	tags	c-tokens	tags	c-tokens	% cyclic
$l = 20$	4	1	17	10	35	10	25	100.0
	5	4	65	17	83	23	85	100.0
	6	13	200	30	241	41	171	97.6
	7	37	537	68	585	97	512	99.0
	8	107	1480	147	1619	202	1268	98.0
	9	300	3939	362	4124	512	3799	96.3
	10	844	10411	934	10869	1204	10089	95.8
$h \geq 28$	4	1	22	10	36	10	25	100.0
	5	4	74	17	84	23	85	100.0
	6	12	213	29	238	41	178	97.6
	7	32	559	64	586	97	518	99.0
	8	90	1489	135	1632	199	1238	98.0
	9	263	4158	329	4314	504	3760	95.8
	10	714	10837	809	11250	1163	9937	93.6

`Input:` $\varepsilon > 0$

`Output:` Feasible solution $(x_p)_{p \in \mathcal{P}}$ to the fractional relaxation of ILP (6)-(8)

For every $p \in \mathcal{P}$, $x_p \leftarrow 0$

$\delta \leftarrow (1+\varepsilon)((1+\varepsilon)N)^{-1/\varepsilon}$

For every $i \in \{1, \ldots, N\}$, $y_i \leftarrow \delta$

Find a minimum weight s-t path p in G, where $weight(v) = y_i$ for every $v \in V_i$, $i \in \{1, \ldots, N\}$

`While` $weight(p) < 1$ `do`

$M \leftarrow \max_i |p \cap V_i|$

$x_p \leftarrow x_p + \frac{1}{M}$

For every i, $y_i \leftarrow y_i(1 + \varepsilon \frac{|p \cap V_i|}{M})$

Find a minimum weight s-t path p in G, where $weight(v) = y_i$ for every $v \in V_i$,

$i \in \{1, \ldots, N\}$

`EndWhile`

For every $p \in \mathcal{P}$, $x_p \leftarrow x_p / \left(\log_{1+\varepsilon} \frac{1+\varepsilon}{\delta}\right)$

Figure 1: The Garg and Könemann algorithm.

```
Input: Positive integers c and l, c ≤ l
Output: Feasible MTSDP(l|C|multiple) solution 𝒯
---
Mark all c-tokens as available
For every i ∈ {1, 2, ..., l}, B_i ← A
𝒯 ← ∅; Finished ← 0; pos ← 1
While Finished = 0 do
    While the weight of B_1 B_2 ... B_pos < c do
        pos ← pos + 1
    EndWhile
    If the c-token ending B_1 B_2 ... B_pos is available then
        If pos = l then
            𝒯 ← 𝒯 ∪ {B_1 B_2 ... B_l}
            Mark all the c-tokens of B_1 B_2 ... B_l as unavailable
            pos ← [the position where the first c-token of B_1 B_2 ... B_l ends]
            I ← {i | 1 ≤ i ≤ pos, B_i ≠ G}
            If I = ∅ then
                Finished ← 1
            Else
                pos ← max{I}
                B_i ← A for all i ∈ {pos + 1, ..., l}
                B_pos ← nextbase(B_pos)
            EndIf
        Else
            pos ← pos + 1
        EndIf
    Else
        I ← {i | 1 ≤ i ≤ pos, B_i ≠ G}
        If I = ∅ then
            Finished ← 1
        Else
            pos ← max{I}
            B_i ← A for all i ∈ {pos + 1, ..., l}
            B_pos ← nextbase(B_pos)
        EndIf
    EndIf
EndWhile
```


Figure 2: The alphabetic tree search algorithm for MTSDP($l|C|multiple$). The $nextbase(\cdot)$ function is defined by $nextbase(\mathsf{A}) = \mathsf{T}$, $nextbase(\mathsf{T}) = \mathsf{C}$, and $nextbase(\mathsf{C}) = \mathsf{G}$.

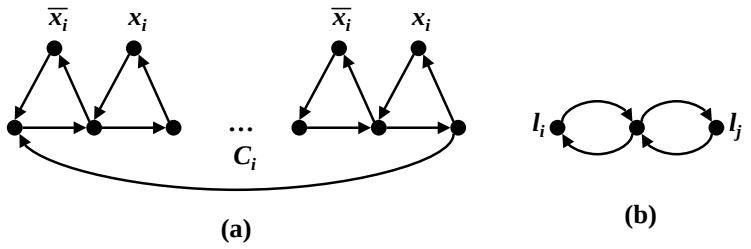

Figure 3: Vertices and arcs added to $G(\phi)$ for (a) variable x_i, and (b) clause $l_i \vee l_j$.