An LP-Rounding Algorithm for Degenerate Primer Design*

Yu-Ting Huang and Marek Chrobak

Department of Computer Science, University of California, Riverside

Abstract. In applications where a collection of similar sequences needs to be amplified through PCR, *degenerate primers* can be used to improve the efficiency and accuracy of amplification. Conceptually, a degenerate primer is a sequence in which some bases are *ambiguous*, in the sense that they can bind to more than one nucleotide. These ambiguous bases allow degenerate primers to bind to multiple target sequences. When designing degenerate primers, it is essential to find a good balance between high coverage (the number of amplified target sequences) and low degeneracy. In this paper, we propose a new heuristic, called RRD2P, for computing a pair of forward and reverse primers with near-optimal coverage, under the specified degeneracy threshold. The fundamental idea of our algorithm is to represent computing optimal primers as an integer linear program, solve its fractional relaxation, and then apply randomized rounding to compute an integral solution. We tested Algorithm RRD2P on three biological data sets, and our experiments confirmed that it produces primer pairs with good coverage, comparing favorably with a similar tool called HYDEN.

1 Introduction

Polymerase Chain Reaction (PCR) is an amplification technique widely used in molecular biology to generate multiple copies of a desired region of a given DNA sequence. In a PCR process, two small pieces of synthetic DNA sequences called *primers*, typically of length 15-30 bases, are required to identify the boundary of amplification. This pair of primers, referred to as *forward* and *reverse primers*, are obtained from the 5' end of the target sequences and their opposite strand, respectively. Each primer hybridizes to the 3' end of another strand and starts to amplify toward the 5' end.

In applications where a collection of similar sequences needs to be amplified through PCR, *degenerate primers* can be used to improve the efficiency and accuracy of amplification. Degenerate primers [1] can be thought of, conceptually, as having *ambiguous bases* at certain positions, that is bases that represent several different nucleotides. This enables degenerate primers to bind to several different sequences at once, thus allowing amplification of multiple sequences in a single PCR experiment. Degenerate primers are represented as strings formed

* Research supported by NSF grants CCF-1217314 and NIH 1R01AI078885.

D. Brown and B. Morgenstern (Eds.): WABI 2014, LNBI 8701, pp. 107–121, 2014.

from IUPAC codes, where each code represents multiple possible alternatives for each position in a primer sequence (see Table 1).

The *degeneracy* $\deg(p)$ of a primer p is the number of distinct non-degenerate primers that it represents. For example, the degeneracy of primer $p = \texttt{ACMCM}$ is 4, because it represents the following four non-degenerate primers: `ACACA`, `ACACC`, `ACCCA`, and `ACCCC`.

Table 1. IUPAC nucleotide code table for ambiguous bases

IUPAC nucleotide code	M	R	W	S	Y	K	V	H	D	B	N
represented bases	A	A	A				A	A	A		A
	C			C	C		C	C		C	C
		G		G		G	G		G	G	G
			T		T	T		T	T	T	T

Figure 1 shows a simple example of a pair of primers binding to a target DNA sequence. The forward primer `TRTAWTGATY` matches the substring `TGGACTGATT` of the target sequence in all but two positions, illustrating that, in practice, binding can occur even in the presence of a small number of mismatched bases. The reverse primer `AGAAAAGTCM` matches the target sequence (or, more precisely, its reverse complement) perfectly. This primer pair can produce copies of the region `ACCGATGACT` of the target sequence, as well as its reverse complement.

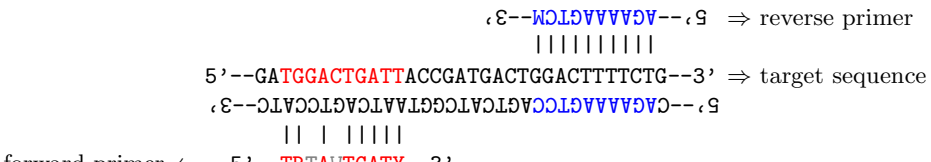

Fig. 1. A symbolic illustration of a pair of primers binding to a DNA sequence

Quite obviously, primers with higher degeneracy can cover more target sequences, but in practice high degeneracy can also negatively impact the quality and quantity of amplification. This is because, in reality, degenerate primers are just appropriate mixtures of regular primers, and including too many primers in the mixture could lead to problems such as mis-priming, where unrelated sequences may be amplified, or primer cross-hybridization, where primers may hybridize to each other. Thus, when designing degenerate primers, it is essential to find a good balance between high coverage and low degeneracy.

PCR experiments involving degenerate primers are useful in studying the composition of microbial communities that typically include many different but similar organisms (see, for example, [2,3]). This variant of PCR is sometimes

referred to as *Multiplex PCR (MP-PCR)* [4], although in the literature the term MP-PCR is also used in the context of applications where non-similar sequences are amplified, in which case using degenerate primers may not be beneficial. Designing (non-degenerate) primers for MP-PCR applications also leads to interesting algorithmic problems – see, for example, [5] and references therein.

For the purpose of designing primers we can assume that our target sequences are single-strain DNA sequences. Thus from now on target sequences will be represented by strings of symbols `A`, `C`, `T`, and `G`.

We say that a (degenerate) primer p *covers* a target sequence s if at least one of the non-degenerate primers represented by p occurs in s as a substring. In practice, a primer can often hybridize to the target sequence even if it only approximately matches the sequence. Formally, we will say that p *covers* s *with at most* m *mismatches* if there exists a sub-string s' of s of length $|p|$ such some non-degenerate primer represented by p matches s' on at least $|p| - m$ positions. We refer to m as *mismatch allowance*.

Following the approach in [6], we model the task as an optimization problem that can be formulated as follows: *given a collection of target sequences, a desired primer length, and bounds on the degeneracy and mismatch allowance, we want to find a pair of degenerate primers that meet these criteria and maximize the number of covered target sequences*. We add, however, that, as discussed later in Section 2, there are other alternative approaches that emphasize other aspects of primer design, for example biological properties of primers.

As in [6], using heuristics taking advantage of properties of DNA sequences, the above task can be reduced to the following problem, which, though conceptually simpler, still captures the core difficulty of degenerate primer design:

Problem: $\mathbf{MCDPD}^{mis}$.

Instance: A set of n target strings $A = \{a^1, a^2, \ldots, a^n\}$ over alphabet Σ, each of length k, integers d (degeneracy threshold) and m (mismatch allowance);

Objective: Find a degenerate primer p of length k and degeneracy at most d that covers the maximum number of strings in A with up to m mismatches.

This reduction involves computing the left and right primers separately, as well as using local alignment of target sequences to extract target strings that have the same length as the desired primer. There may be many collections of such target strings (see Section 4), and only those likely to produce good primer candidates need to be considered. Once we solve the instance of $\mathbf{MCDPD}^{mis}$ for each collection, obtaining a number of forward and reverse primer candidates, we select the final primer pair that optimizes the joint coverage, either through exhaustive search or using heuristic approaches.

The main contribution of this paper is a new algorithm for $\mathbf{MCDPD}^{mis}$, called $\mathrm{SRR}_{\mathrm{dna}}$, based on LP-rounding. We show that $\mathbf{MCDPD}^{mis}$ can be formulated as an integer linear program. (This linear program actually solves a slightly modified version of $\mathbf{MCDPD}^{mis}$ – see Section 3 for details.) Algorithm $\mathrm{SRR}_{\mathrm{dna}}$ computes the optimal fractional solution of this linear program, and then uses

an appropriate randomized rounding strategy to convert this fractional solution into an integral one, which represents a degenerate primer.

Using the framework outlined above, we then use Algorithm $\mathrm{SRR}_{\mathrm{dna}}$ to design a new heuristic algorithm, called RRD2P, that computes pairs of primers for a given collection of target DNA sequences. We implemented Algorithm RRD2P and tested it on three biological data sets. The results were compared to those generated by the existing state-of-the-art tool HYDEN, developed by Linhart and Shamir [7]. Our experiments show that Algorithm RRD2P is able to find primer pairs with better coverage than HYDEN.

2 Related Work

The problem of designing high-quality primers for PCR experiments has been extensively studied and has a vast literature. Much less is known about designing *degenerate primers*. The work most relevant to ours is by Linhart and Shamir [7], who introduced the $\mathbf{MCDPD}^{mis}$ model, proved that the problem is $\mathbb{NP}$-hard, and gave some efficient approximation algorithms.

In their paper [7], the ideas behind their approximation algorithms were incorporated into a heuristic algorithm HYDEN for designing degenerate primers. HYDEN uses an efficient heuristic approach to design degenerate primers [7] with good coverage. It constructs primers of specified length and with specified degeneracy threshold. HYDEN consists of three phases. It first uses a non-gap local alignment algorithm to find best-conserved regions among target sequences. These regions are called *alignments*. The degree to which an alignment A is conserved is measured by its entropy score:

$$H_A = -\sum_{j=1}^{k} \sum_{\sigma \in \Sigma} \frac{D_A(\sigma,j)}{n} \cdot \log_2 \frac{D_A(\sigma,j)}{n},$$

where k is the length of A, n is the number of target sequences, and $D_A(\sigma, j)$ is the number of sequences in A that have symbol σ at the j_{th} position. Matrix $D_A()$ is called the *column distribution matrix*.

Then, HYDEN designs degenerate primers for these regions using two heuristic algorithms called CONTRACTION and EXPANSION. Finally, it chooses a certain number of best primer candidates, from which it computes a pair of primers with good coverage using a hill-climbing heuristic.

HYDEN has a number of parameters that users can specify, including the desired primer length, the preferred binding regions, the degeneracy threshold, and the mismatch allowance. HYDEN has been tested in a real biological experiment with 127 human olfactory receptor (OR) genes, showing that it produces fairly good primer pairs [7].

Linhart and Shamir [7] also introduced another variant of degenerate primer design problem, called **MDDPD**, where the objective is to find a degenerate primer that covers all given target sequences and has minimum degeneracy. This problem is also $\mathbb{NP}$-hard. An extension of this model where multiple primers are sought was studied by Souvenir *et al.* [8]. See also [9,10] for more related work.

We briefly mention two other software packages for designing degenerate primers. (See the full paper for a more comprehensive survey.) PrimerHunter [11,12] is a software tool that accepts both target and non-target sequences on input, to ensure that the selected primers can efficiently amplify target sequences but avoid the amplification of non-target sequences. This feature allows PrimerHunter to distinguish closely related subtypes. PrimerHunter allows users to set biological parameters. However, it does not provide the feature to directly control the primer degeneracy. Instead, it uses a degeneracy mask that specifies the positions at which fully degenerate nucleotides are allowed.

iCODEHOP is a web application which designs degenerate primers at the amino acid level [13,14]. This means that during the primer design process, it will reverse-translate the amino acid sequences to DNA, using a user-specified codon usage table. iCODEHOP does not explicitly attempt to optimize the coverage.

3 Randomized Rounding

We now present our randomized rounding approach to solving the $\mathbf{MCDPD}^{mis}$ problem defined in the introduction. Recall that in this problem we are given a collection A of strings over an alphabet Σ, each of the same length k, a degeneracy threshold d, and a mismatch allowance m, and the objective is to compute a degenerate primer p of length k and degeneracy at most d, that covers the maximum number of strings in A with at most m mismatches.

An optimal primer p covers at least one target string $a^i \in A$ with at most m mismatches. In other words, p can be obtained from a^i by (i) changing at most m bases in a^i to different bases, and (ii) changing some bases in a^i to ambiguous bases that match the original bases, without exceeding the degeneracy limit d. Let $Tmpl_m(A)$ denote the set of all strings of length k that can be obtained from some target string $a^i \in A$ by operation (i), namely changing up to m bases in a^i. By trying all strings in $Tmpl_m(A)$, we can reduce $\mathbf{MCDPD}^{mis}$ to its variant where p is required to cover a given template string (without mismatches). Formally, this new optimization problem is:

Problem: $\mathbf{MCDPD}^{mis}_{tmpl}$.

Instance: A set of n strings $A = \{a^1, a^2, \ldots, a^n\}$, each of length k, a template string $\hat{p}$, and integers d (degeneracy threshold) and m (mismatch allowance);

Objective: Find a degenerate primer p of length k, with $\deg(p) \leq d$ that covers $\hat{p}$ and covers the maximum number of sequences in A with mismatch allowance m.

We remark that our algorithm for $\mathbf{MCDPD}^{mis}$ will not actually try all possible templates from $Tmpl_m(A)$ – there are simply too many of these, if m is large. Instead, we randomly sample templates from $Tmpl_m(A)$ and apply the algorithm for $\mathbf{MCDPD}^{mis}_{tmpl}$ only to those sampled templates. The number of samples affects the running time and accuracy (see Section 5).

We present our algorithm for $\mathbf{MCDPD}^{mis}_{tmpl}$ in two steps. In Section 3.1 that follows, we explain the fundamental idea of our approach, by presenting the linear program and our randomized rounding algorithm for the case of binary strings, where $\Sigma = \{\mathtt{0}, \mathtt{1}\}$. The extension to DNA strings is somewhat complicated due to the presence of several ambiguous bases. We present our linear program formulation and the algorithm for DNA strings in Section 3.2.

3.1 Randomized Rounding for Binary Strings

In this section we focus on the case of the binary alphabet $\Sigma = \{\mathtt{0}, \mathtt{1}\}$. For this alphabet we only have one ambiguous base, denoted by $\mathtt{N}$, which can represent either $\mathtt{0}$ or $\mathtt{1}$. We first demonstrate the integer linear program representation of $\mathbf{MCDPD}^{mis}_{tmpl}$ for the binary alphabet and then we give a randomized rounding algorithm for this case, called $\mathrm{SRR}_{\mathrm{bin}}$. The idea of $\mathrm{SRR}_{\mathrm{bin}}$ is to compute an optimal fractional solution of this linear program and then round it to a feasible integral solution.

Let $\hat{p} = \hat{p}_1\hat{p}_2\cdots\hat{p}_k$ be the template string from the given instance of $\mathbf{MCDPD}^{mis}_{tmpl}$. It is convenient to think of the objective of $\mathbf{MCDPD}^{mis}_{tmpl}$ as converting $\hat{p}$ into a degenerate primer p by changing up to $\log d$ symbols in $\hat{p}$ to $\mathtt{N}$. For each target string $a^i = a^i_1 a^i_2 \cdots a^i_k$, we use a binary variable x^i to indicate if a^i is covered by p. For each position j, a binary variable n_j is used to indicate whether $\hat{p}_j$ will be changed to $\mathtt{N}$. To take mismatch allowance into consideration, we also use binary variables μ^i_j, which indicate if we allow a mismatch between p and a^i on position j, that is, whether or not $a^i_j \not\sqsubseteq p_j$.

With the above variables, the objective is to maximize the sum of all x^i. Next, we need to specify the constraints. One constraint involves the mismatch allowance m; for a string a^i, the number of mismatches $\sum_j \mu^i_j$ should not exceed m. Next, we have the bound on the degeneracy. In the binary case, the degeneracy of p can be written as $\deg(p) = \prod_j 2^{n_j}$, and we require that $\deg(p) \le d$. To convert this inequality into a linear constraint, we take the logarithms of both sides. The last group of constraints are the covering constraints. For each j, if p covers a^i and $\hat{p}_j \neq a^i_j$, then either $p_j = \mathtt{N}$ or p_j contributes to the number of mismatches. This can be expressed by inequalities $x^i \le n_j + \mu^i_j$, for all i, j such that $a^i_j \neq \hat{p}_j$. Then the complete linear program is:

$$\begin{array}{rll} \text{maximize} & \sum_i x^i & \\ \text{subject to} & \sum_j \mu^i_j \le m & \forall i \\ & \sum_j n_j \le \log_2 d & \\ & x^i \le n_j + \mu^i_j, & \forall i, j : a^i_j \neq \hat{p}_j \\ & x^i, n_j, \mu^i_j \in \{0,1\} & \forall i, j \end{array} \tag{1}$$

The pseudo-code of our Algorithm $\mathrm{SRR}_{\mathrm{bin}}$ is given below in Pseudocode 1. The algorithm starts with $p = \hat{p}$ and gradually changes some symbols in p to $\mathtt{N}$,

solving a linear program at each step. At each iteration, the size of the linear program can be reduced by discarding strings that are too different from the current p, and by ignoring strings that are already matched by p. More precisely, any a^i which differs from the current p on more than $m + \log_2 d$ positions cannot be covered by any degenerate primer obtained from p, so this a^i can be discarded. On the other hand, if a^i differs from p on at most m positions then it will always be covered, in which case we can set $x^i = 1$ and we can also remove it from A. This pruning process in Algorithm $\mathrm{SRR}_{\mathrm{bin}}$ is implemented by function FilterOut.

Pseudocode 1. Algorithm $\mathrm{SRR}_{\mathrm{bin}}(\hat{p}, A, d, m)$

```
1: p ← p̂
2: while deg(p) < d do
3:     FilterOut(p, A, d, m)
4:     if A = ∅ then break                          ▷ updates A
5:     LP ← GenLinProgram(p, A, d, m)
6:     FracSol ← SolveLinProgram(LP)
7:     RandRounding_bin(p, FracSol, d)              ▷ updates p and d
8: return p
```

If no sequences are left in A then we are done; we can output p. Otherwise, we construct the linear program for the remaining strings. This linear program is essentially the same as the one above, with $\hat{p}$ replaced by p, and with the constraint $x^i \leq n_j + \mu_j^i$ included only if $p_j \neq \mathtt{N}$. Additional constraints are added to take into account the rounded positions in p, namely we add the constraint $n_j = 1$ for all p_j already replaced by N.

We then consider the relaxation of the above integer program, where all integral constraints $x^i, n_j, \mu_j^i \in \{0, 1\}$ are replaced by $x^i, n_j, \mu_j^i \in [0, 1]$, that is, all variables are allowed to take fractional values. After solving this relaxation, we call Procedure $\mathrm{RandRounding}_{bin}$, which chooses one fractional variable n_j, with probability proportional to its value, and rounds it up to 1. (It is sufficient to round only the n_j variables, since all other variables are uniquely determined from the n_j's.) To do so, let J be the set of all j for which $n_j \neq 1$ and $\pi = \sum_{j \in J} n_j$. The interval $[0, \pi]$ can be split into consecutive $|J|$ intervals, with the interval corresponding to $j \in J$ having length n_j. Thus we can randomly (uniformly) choose a value c from $[0, \pi]$, and if c is in the interval corresponding to $j \in J$ then we round n_j to 1.

If the degeneracy of p is still below the threshold, Algorithm $\mathrm{SRR}_{\mathrm{bin}}$ executes the next iteration: it correspondingly adjusts the constraints of the linear program, which produces a new linear program, and so on. The process stops when the degeneracy allowance is exhausted.

3.2 Randomized Rounding for DNA Sequences Data

We now present our randomized rounding scheme for $\mathbf{MCDPD}^{mis}_{tmpl}$ when the input consists of DNA sequences.

We start with the description of the integer linear program for $\mathbf{MCDPD}^{mis}_{tmpl}$ with $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$. Degenerate primers for DNA sequences, in addition to four nucleotide symbols A, C, G and T, can use eleven symbols corresponding to ambiguous positions, described by their IUPAC codes M, R, W, S, Y, K, V, H, D, B, and N. The interpretation of these codes was given in Table 1 in Section 1. Let Λ denote the set of these fifteen symbols. We think of each $\lambda \in \Lambda$ as representing a subset of Σ, and we write $|\lambda|$ for the cardinality of this subset. For example, we have $|\mathtt{C}| = 1$, $|\mathtt{H}| = 3$ and $|\mathtt{N}| = 4$.

The complete linear program is given below. As for binary sequences, x^i indicates whether the i-th target sequence a^i is covered. Then the objective of the linear program is to maximize the primer coverage, that is $\sum_i x^i$.

$$\begin{aligned}
&\text{maximize} \quad \textstyle\sum_i x^i \\
m_j + r_j + w_j + s_j + y_j + k_j + v_j + h_j + d_j + b_j + n_j \le 1 \qquad & \forall j \\
\textstyle\sum_j \mu^i_j \le m \qquad & \forall i \\
\textstyle\sum_j \left[(m_j + r_j + w_j + s_j + y_j + k_j) + \log 3 \cdot (v_j + h_j + d_j + b_j) + 2 \cdot n_j\right] \le \log d & \\
x^i \le m_j + v_j + h_j + n_j + \mu^i_j \qquad & \forall i,j : (\hat{p}_j = \mathtt{A}, a_{ij} = \mathtt{C}) \vee (\hat{p}_j = \mathtt{C}, a_{ij} = \mathtt{A}) \\
x^i \le r_j + v_j + d_j + n_j + \mu^i_j \qquad & \forall i,j : (\hat{p}_j = \mathtt{A}, a_{ij} = \mathtt{G}) \vee (\hat{p}_j = \mathtt{G}, a_{ij} = \mathtt{A}) \\
x^i \le w_j + h_j + d_j + n_j + \mu^i_j \qquad & \forall i,j : (\hat{p}_j = \mathtt{A}, a_{ij} = \mathtt{T}) \vee (\hat{p}_j = \mathtt{T}, a_{ij} = \mathtt{A}) \\
x^i \le s_j + v_j + b_j + n_j + \mu^i_j \qquad & \forall i,j : (\hat{p}_j = \mathtt{C}, a_{ij} = \mathtt{G}) \vee (\hat{p}_j = \mathtt{G}, a_{ij} = \mathtt{C}) \\
x^i \le y_j + h_j + b_j + n_j + \mu^i_j \qquad & \forall i,j : (\hat{p}_j = \mathtt{C}, a_{ij} = \mathtt{T}) \vee (\hat{p}_j = \mathtt{T}, a_{ij} = \mathtt{C}) \\
x^i \le k_j + d_j + b_j + n_j + \mu^i_j \qquad & \forall i,j : (\hat{p}_j = \mathtt{G}, a_{ij} = \mathtt{T}) \vee (\hat{p}_j = \mathtt{T}, a_{ij} = \mathtt{G}) \\
x^i, m_j, r_j, w_j, s_j, y_j, k_j, v_j, h_j, d_j, b_j, n_j, \mu^i_j \in \{0,1\} \qquad & \forall i,j
\end{aligned}$$

To specify the constraints, we now have eleven variables representing the presence of ambiguous bases in the degenerate primer, namely m_j, r_j, w_j, s_j, y_j, k_j, v_j, h_j, d_j, b_j, and n_j, denoted using letters corresponding to the ambiguous symbols. Specifically, for each position j and for each symbol $\lambda \in \Lambda$, the corresponding variable λ_j indicates whether $\hat{p}_j$ is changed to this symbol in the computed degenerate primer p. For example, r_j represents the absence or presence of R in position j. For each j, at most one of these variables can be 1, which can be represented by the constraint that their sum is at most 1.

Variables μ^i_j indicate a mismatch between p and a^i on position j. Then the bound on the number of mismatches can be written as $\sum_j \mu^i_j \le m$, for each i. The bound on the degeneracy of the primer p can be written as

$$\deg(p) = \textstyle\prod_j 2^{(m_j + r_j + w_j + s_j + y_j + k_j)} \times 3^{(v_j + h_j + d_j + b_j)} \times 4^{n_j} \;\le\; d,$$

which after taking logarithms of both sides gives us another linear constraint.

In order for a^i to be covered (that is, when $x^i = 1$), for each position j for which $a^i_j \neq \hat{p}_j$, we must either have a mismatch at position j or we need $a^i_j \subseteq p_j$. Expressing this with linear constraints can be done by considering cases corresponding to different values of $\hat{p}_j$ and a^i_j. For example, when $\hat{p}_j = \texttt{A}$ and $a^i_j = \texttt{C}$ (or vice versa), then either we have a mismatch at position j (that is, $\mu^i_j = 1$) or p_j must be one of ambiguous symbols that match A and C (that is M, V, H, or N). This can be expressed by the constraint $x^i \leq m_j + v_j + h_j + n_j + \mu^i_j$. We will have one such case for any two different choices of $\hat{p}_j$ and a^i_j, giving us six groups of such constraints.

We then extend our randomized rounding approach from the previous section to this new linear program. From the linear program, we can see that the integral solution can be determined from the values of all variables λ_j, for $\lambda \in \Lambda$. In the fractional solution, a higher value of λ_j indicates that p_j is more likely to be the ambiguous symbol λ. We thus determine ambiguous bases in p one at a time by rounding the corresponding variables.

As for binary strings, Algorithm SRR_{dna} will start with $p = \hat{p}$ and gradually change some bases in p to ambiguous bases, solving a linear program at each step. At each iteration we first call function FilterOut that filters out target sequences that are either too different from the template $\hat{p}$, so that they cannot be matched, or too similar, in which case they are guaranteed to be matched. The pseudocode of Algorithm SRR_{dna} is the same as in Pseudocode 1 except that the procedure RandRounding$_{bin}$ is replaced by the corresponding procedure RandRounding$_{dna}$ for DNA strings.

If no sequences are left in A then we output p and halt. Otherwise, we construct a linear program for the remaining sequences. This linear program is a slight modification of the one above, with $\hat{p}$ replaced by p. Each base p_j that was rounded to an ambiguous symbol is essentially removed from consideration and will not be changed in the future. Specifically, the constraints on x^i associated with this position j will be dropped from the linear program (because these constraints apply only to positions where $p_j \in \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}\}$). For each position j that was already rounded, we appropriately modify the corresponding variables. If $p_j = \lambda$, for some $\lambda \in \Lambda - \Sigma$, then the corresponding variable λ_j is set to 1 and all other variables λ'_j are set to 0. If $a^i_j \in p_j$, that is, a^i_j is already matched, then we set $\mu^i_j = 0$, and if $a^i_j \notin p_j$ then we set $\mu^i_j = 1$, which effectively reduces the mismatch allowance for a^i in the remaining linear program.

Next, Algorithm SRR_{dna} solves the fractional relaxation of such constructed integer program, obtaining a fractional solution *FracSol*. Finally, the algorithm calls function RandRounding$_{dna}$ that will round one fractional variable λ_j to 1. (This represents setting p_j to λ.) To choose j and the symbol λ for p_j, we randomly choose a fractional variable λ_j proportionally to their values among undetermined positions. This is done similarly as in the binary case, by summing up fractional values corresponding to different symbols and positions, and choosing uniformly a random number c between 0 and this sum. This c determines which variable should be rounded up to 1.

3.3 Experimental Approximation Ratio

To examine the quality of primers generated by algorithm $\mathrm{SRR}_{\mathrm{dna}}$, we compared the coverage of these primers to the optimal coverage. In our experiments we used the human OR gene [7] data set consisting of 50 sequences, each of length approximately 1Kbps. For this dataset we computed 15 alignments (regions) of length 25 with highest entropy scores (representing sequence similarity, see Section 2). Thus each obtained alignment consists of 50 target strings of length 25. Then, for each of these alignments A, we use each target string in A as a template to run $\mathrm{SRR}_{\mathrm{dna}}$, which gives us 50 candidate primers, from which we choose the best one. We then compared this selected primer to a primer computed with Cplex using a similar process, namely computing an optimal integral solution for each template and choosing the best solution.

Table 2. Algorithm $\mathrm{SRR}_{\mathrm{dna}}$ versus the integral solution obtained with Cplex. The numbers represent coverage values for the fifteen alignments.

	$d = 10000$, $m = 0$													$d = 625$, $m = 2$											
A_i	1	2	3	4	5	6	7	8	9	10	11	12	13	1	2	3	4	5	6	7	8	9	10	11	12	13
Opt	26	24	24	24	26	26	24	24	24	26	24	24	24	43	42	42	42	43	43	42	42	42	43	42	42	42
SRR	26	24	23	23	26	26	23	24	23	26	24	23	23	42	40	42	42	43	43	40	41	42	43	42	42	40

This experiment was repeated for two different settings for m (the mismatch allowance) and d (the degeneracy threshold), namely for $(m, d) = (0, 1000), (2, 625)$. The results are shown in Table 2. As can be seen from this table, Algorithm $\mathrm{SRR}_{\mathrm{dna}}$ computes degenerate primers that are very close, and often equal, to the values obtained from the integer program. Note that for $m = 0$ the value obtained with the integer program represents the true optimal solution for the instance of $\mathbf{MCDPD}^{mis}$, because we try all target strings as templates. For $m = 2$, to compute the optimal solution we would have to try all template strings in $Tmpl_2(A_h)$, which is not feasible; thus the values in the first row are only close approximations to the optimum.

The linear programs we construct are very sparse. This is because for any given a^i and position j, the corresponding constraint on x^i is generated only when p and a^i differ on position j (see Section 3.2), and our data sets are very conserved. Thus, for sufficently small data sets one could simply use integral solutions from Cplex instead of rounding the fractional solution. For example, the initial linear programs in the above instances had typically around 150 constraints, and computing each integral solution took only about 5 times longer than for the fractional solution (roughly, $0.7s$ versus $0.15s$). For larger datasets, however, computing the optimal integral solution becomes quickly infeasible.

4 RRD2P – Complete Primer Design Algorithm

To assess the effectiveness of our randomized rounding approach, we have extended Algorithm $\mathrm{SRR}_{\mathrm{dna}}$ to a complete primer design algorithm, called RRD2P, and

we tested it experimentally on real data sets. In this section we describe Algorithm RRD2P; the experimental evaluation is given in the next section.

Algorithm RRD2P (see Pseudocode 2) has two parameters: S_{fvd} and S_{rev}, which are, respectively, two sets of target sequences, one for forward and the other for reverse primers. They are provided by the user and represent desired binding regions for the two primers. The algorithm finds candidates for forward primers and reverse primers separately. Then, from among these candidates, it iterates over all primer pairs to choose primer pairs with the best joint coverage.

Pseudocode 2. Algorithm RRD2P($S_{fvd}, S_{rev}, k, d, m$)

1: $PrimerList_{fvd} \leftarrow$ DesignPrimers(S_{fvd}, k, d, m)
2: $PrimerList_{rev} \leftarrow$ DesignPrimers(S_{rev}, k, d, m)
3: ChooseBestPairs($PrimerList_{fvd}, PrimerList_{rev}$) ▷ Find best primer pairs (f, r)

For both types of primers, we call Algorithm DesignPrimers (Pseudocode 3), that consists of two parts. In the first part, the algorithm identifies conserved regions within target sequences (Line 1). As before, these regions are also called alignments, and they are denoted A_h. In the second part we design primers for these regions (Lines 2-7).

Pseudocode 3. Algorithm DesignPrimers($S = \{s^1, s^2, \cdots, s^n\}, k, d, m$)

1: $A_1, A_2, \cdots A_N \leftarrow$ FindAlignments(S, k)
2: **for all** alignments A_h, $h = 1, \cdots N$ **do**
3: $PL_h \leftarrow \emptyset$
4: $T_h \leftarrow$ set of templates ▷ see explanation in text
5: **for all** $\hat{p} \in T_h$ **do**
6: $p \leftarrow \mathrm{SRR}_{\mathrm{dna}}(\hat{p}, A_h, d, m)$
7: Add p to PL_h
8: $PrimerList \leftarrow PL_1 \cup PL_2 \cdots \cup PL_N$
9: **return** *PrimerList* (sorted according to coverage)

Finding alignments. Algorithm FindAlignments for locating conserved regions (Pseudocode 4) follows the strategy from [7]. It enumerates over all sub-strings of length k of the target sequences. For each k-mer, K, we align it against every target sequence s^i without gaps, to find the best match a^i of length k, i.e, a^i has the smallest Hamming distance with K. The resulting set $A = \{a^1, a^2, \cdots, a^n\}$ of the n best matches, one for each target string, is a conserved region (alignment). Intuitively, more conserved alignments are preferred, since they are more likely to generate low-degeneracy primers. In order to identify how well-conserved an alignment A is, the entropy score is applied.

Pseudocode 4. Algorithm FINDALIGNMENTS($S = \{s^1, s^2 \cdots, s^n\}, k$)

```
1: AlignmentList ← ∅
2: for all k-mers, K, in S do
3:     A ← ∅
4:     for all s^i ∈ S do
5:         a^i ← substring of s^i that is the best match for K
6:         Add a^i to A
7:     Add A to AlignmentList
8: return AlignmentList (sorted according to entropy)
```

Computing primers. In the second part (Lines 2-7), the algorithm considers all alignments A_h computed by Algorithm FINDALIGNMENTS. For each A_h, we use the list T_h of template strings (see below), and for each $\hat{p} \in T_h$ we call $\mathrm{SRR}_{\mathrm{dna}}(\hat{p}, A_h, d, m)$ to compute a primer p that is added to the list of primers PL_h. All lists PL_h are then combined into the final list of candidate primers.

It remains to explain how to choose the set T_h of templates. If the set $Tmpl_m(A_h)$ of all candidate templates is small then one can take T_h to be the whole set $Tmpl_m(A_h)$. (For instance, when $m = 0$ then $Tmpl_0(A_h) = A_h$.) In general, we take T_h to be a random sample of r strings from $Tmpl_m(A_h)$, where the value of r is a parameter of the program, which can be used to optimize the tradeoff between the accuracy and the running time. Each $\hat{p} \in T_h$ is constructed as follows: (i) choose uniformly a random $a^i \in A_h$, (ii) choose uniformly a set of exactly m random positions in a^i, and (iii) for each chosen position j in a^i, set a^i_j to a randomly chosen base, where this base is selected with probability proportional to its frequency in position j in all sequences from A_h.

5 Experiments

We tested Algorithm RRD2P on three biological data sets, and we compared our results to those from Algorithm HYDEN [7].
1. The first data set is a set of 50 sequences of human olfactory receptor (OR) gene [7], of length around 1Kbps, provided along with the HYDEN program.
2. The second data set is from the NCBI flu database [15], from which we chose Human flu sequences of lengths 900-1000 bps (dated from November 2013). This set contains 229 flu sequences.
3. The third one contains 160 fungal ITS genes of various lengths, obtained from NCBI-INSD [16]. Sequence lengths vary from 400 to 2000 bps.

We run Algorithm RRD2P with the following parameters: (i) Primer length = 25. (ii) Primer degeneracy threshold (forward, reverse) : (625,3750), (1250,7500), (1875, 11250), (2500, 15000), (3750, 22500), (5000,30000), (7500,45000), (10000,60000). Note that the degeneracy values increase roughly exponentially, which corresponds to a linear increase in the number of ambiguous bases.The degeneracy of the reverse primer is six times larger than that of the forward primer (the default in HYDEN). (iii) Forward primer binding range : $0 \sim 300$,

reverse primer binding range : $-1 \sim -350$. (iv) Mismatch allowance : $m = 0, 1, 2$ (m represents the mismatch allowance for each primer separately). (v) Number of alignments: $N = 50$. (vi) Number of template samples: $r = 5$.

We compare our algorithm to HYDEN in terms of the coverage of computed primers. To make this comparison meaningful, we designed our algorithm to have similar input parameters, which allows us to run HYDEN with the same settings. For the purpose of these experiments, we use the best primer pair from the list computed by Algorithm RRD2P.

The results are shown in Figures 2, 3 and 4, respectively. The x-axis represents the degeneracy of the forward primer; the degeneracy of the reverse primer is six times larger. The y-axis is the coverage of the computed primer pair. The results show that RRD2P is capable to find better degenerate primers than HYDEN, for different choices of parameters.

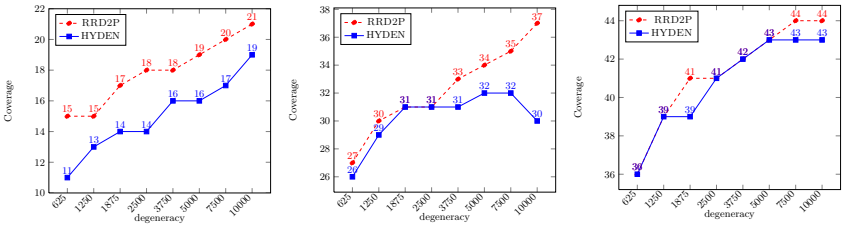

Fig. 2. Comparison of RRD2P and HYDEN on human OR genes for $m = 0$ (left), $m = 1$ (center) and $m = 2$ (right)

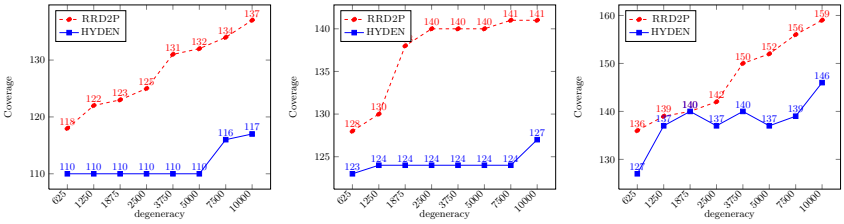

Fig. 3. Comparison of RRD2P and HYDEN on flu sequences for $m = 0$ (left), $m = 1$ (center) and $m = 2$ (right)

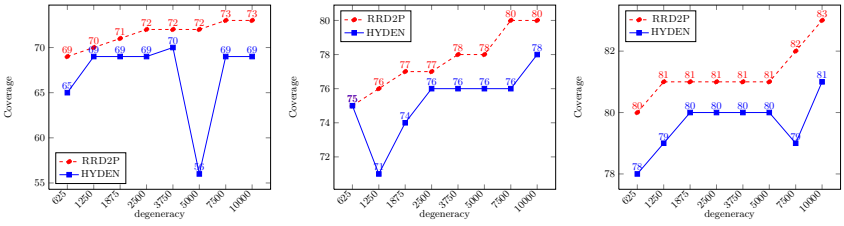

Fig. 4. Comparison of RRD2P and HYDEN on fungal sequences for $m = 0$ (left), $m = 1$ (center) and $m = 2$ (right)

Running time. The running time of Algorithm RRD2P is dominated by the module running Cplex to solve the linear program, and it depends, roughly linearly, on the number of times the LP solver is run. The above experiments were performed for $r = 5$. For the third dataset above and $m = 0$, the running times of Algorithm RRD2P varied from $110s$ for $d = 625$ to $164s$ for $d = 10000$ (on Windows 8 2.4 GHz CPU, 8.0 G memory). The respective run times of HYDEN were lower, between $25s$ and $28s$. The run time of Algorithm RRD2P can be adjusted by using smaller values of r. For example, for $r = 1, 2$, RRD2P is actually faster than HYDEN for small to moderate degeneracy values, and the loss of accuracy is not significant.

6 Discussion

We studied the problem of computing a pair of degenerate forward and reverse primers that maximizes the number of covered target sequences, assuming upper bounds on the primer degeneracy and the number of mismatches. We proposed an algorithm for this problem, called RRD2P, based on representing the problem as an integer linear program, solving its fractional relaxation, and then rounding the optimal fractional solution to integral values. We tested Algorithm RRD2P on three biological datasets. Our algorithm usually finds solutions that are near optimal or optimal, and it produces primer pairs with higher coverage than Algorithm HYDEN from [7], regardless of the parameters.

Our work focussed on optimizing the coverage of the sequence data by degenerate primers. Algorithm RRD2P does not consider biological parameters that affect the quality of the primers in laboratory PCR, including the melting temperature of the primers, GC content, secondary structure and other. In the future, we are planning to integrate Algorithm RRD2P into our software tool, called PRISE2 [17], that can be used to interactively design PCR primers based both on coverage and on a variety of biological parameters.

The integrality gap of the linear programs in Section 3.1 can be shown to be $\Omega(n(m + \log d)/k)$, where n is the number of target sequences, k is their length, d is the degeneracy bound and m is the mismatch allowance. An example with this integrality gap consists of n target binary sequences of length k such that each two differ from each other on more than $m + \log_2 d$ positions. When we choose any target sequence as template, the optimal coverage can only be 1. However, there is a fractional solution with value $n(m + \log d)/k$, obtained by setting $n_j = (\log d)/k$ and $\mu_j^i = m/k$, for all i, j.

Nevertheless, as we show, for real DNA datasets the solutions produced by rounding the fractional solution are very close to the optimum. Providing some analytical results that explain this phenomenon would be of considerable interest, both from the theoretical and practical standpoint, and will be a focus of our future work. This work would involve developing formal models for "conserved sequences" (or adapting existing ones) and establishing integrality gap results, both lower and upper bounds, for such datasets.

Acknowledgements. We would like to thank anonymous reviewers for pointing some deficiencies in the earlier version of the paper and for insightful comments.

References

1. Kwok, S., Chang, S., Sninsky, J., Wang, A.: A guide to the design and use of mismatched and degenerate primers. PCR Methods and Applications 47, S39–S47 (1994)
2. Hunt, D.E., Klepac-Ceraj, V., Acinas, S.G., Gautier, C., Bertilsson, S., Polz, M.F.: Evaluation of 23s rRNA PCR primers for use in phylogenetic studies of bacterial diversity. Applied Environmental Microbiology 72, 2221–2225 (2006)
3. Ihrmark, K., Bödeker, I.T., Cruz-Martinez, K., Friberg, H., Kubartova, A., Schenck, J., Strid, Y., Stenlid, J., Brandström-Durling, M., Clemmensen, K.E., Lindahl, B.D.: New primers to amplify the fungal its2 region–evaluation by 454-sequencing of artificial and natural communities. FEMS Microbiology Ecology 82, 666–677 (2012)
4. Chamberlain, J.S., Gibbs, R.A., Rainer, J.E., Nguyen, P.N., Casey, C.T.: Deletion screening of the duchenne muscular dystrophy locus via multiplex dna amplification. Nucleic Acid Research 16, 11141–11156 (1988)
5. Konwar, K.M., Mandoiu, I.I., Russell, A.C., Shvartsman, A.A.: Improved algorithms for multiplex PCR primer set selection with amplification length constraints. In: Proc. 3rd Asia-Pacific Bioinformatics Conference, pp. 41–50 (2005)
6. Linhart, C., Shamir, R.: The degenerate primer design problem: theory and applications. Journal of Computational Biology 12(4), 431–456 (2005)
7. Linhart, C., Shamir, R.: The degenerate primer design problem. Bioinformatics 180, S172–S180 (2002)
8. Souvenir, R., Buhler, J.P., Stormo, G., Zhang, W.: Selecting degenerate multiplex PCR primers. In: Benson, G., Page, R.D.M. (eds.) WABI 2003. LNCS (LNBI), vol. 2812, pp. 512–526. Springer, Heidelberg (2003)
9. Balla, S., Rajasekaran, S.: An efficient algorithm for minimum degeneracy primer selection. IEEE Transactions on NanoBioscience 6, 12–17 (2007)
10. Sharma, D., Balla, S., Rajasekaran, S., DiGirolamo, N.: Degenerate primer selection algorithms. Computational Intelligence in Bioinformatics and Computational Biology, 155–162 (2009)
11. Duitama, J., Kumar, D.M., Hemphill, E., Khan, M., Mandoiu, I.I., Nelson, C.E.: Primerhunter: a primer design tool for PCR-based virus subtype identification. Nucleic Acids Research 37, 2483–2492 (2009)
12. `http://dna.engr.uconn.edu/software/PrimerHunter/primerhunter.php`
13. Boyce, R., Chilana, P., Rose, T.M.: iCODEHOP: a new interactive program for designing COnsensus-DEgenerate Hybrid Oligonucleotide Primers from multiply aligned protein sequences. Nucleid Acids Research 37, 222–228 (2009)
14. `http://blocks.fhcrc.org/codehop.html`
15. `http://www.ncbi.nlm.nih.gov/genomes/FLU/FLU.html`
16. `http://www.ncbi.nlm.nih.gov/genbank/collab/`
17. Huang, Y.T., Yang, J.I., Chrobak, M., Borneman, J.: Prise2: Software for designing sequence-selective PCR primers and probes (2013) (in preparation)