BIOINFORMATICS ORIGINAL PAPER Vol. 26 no. 16 2010, pages 1932–1937
doi:10.1093/bioinformatics/btq318

Genome analysis Advance Access publication July 2, 2010

Rapid match-searching for gene silencing assessment

Mark E.T. Horn[1,*] and Peter M. Waterhouse[2]

[1]Division of Mathematics, Informatics and Statistics, CSIRO, Locked Bag 17, North Ryde NSW 1670 and [2]School of Biological Sciences, University of Sydney, NSW 2006, Australia

Associate Editor: Ivo Hofacker

ABSTRACT

Motivation: Gene silencing, also called RNA interference, requires reliable assessment of silencer impacts. A critical task is to find matches between silencer oligomers and sites in the genome, in accordance with one-to-many matching rules (G–U matching, with provision for mismatches). Fast search algorithms are required to support silencer impact assessments in procedures for designing effective silencer sequences.

Results: The article presents a matching algorithm and data structures specialized for matching searches, including a kernel procedure that addresses a Boolean version of the database task called the skyline search. Besides exact matches, the algorithm is extended to allow for the location-specific mismatches applicable in plants. Computational tests show that the algorithm is significantly faster than suffix-tree alternatives.

Availability: Source code, executable, data and test results are freely available at ftp://ftp.csiro.au/Horn/RapidMatch

Contact: mark.horn@csiro.au

Supplementary information: Supplementary data are available at *Bioinformatics* online.

Received on March 4, 2010; revised on June 8, 2010; accepted on June 10, 2010

1 INTRODUCTION

Gene silencing involves mechanisms to suppress the expression of particular genes. These mechanisms, also called RNA interference (RNAi), contribute to the development of organisms and to protection against viral attacks; they also have important experimental and therapeutic applications. The silencing process studied in this article commences with the formation of double-stranded RNA (dsRNA) molecules, either as part of an organism's normal functions or via a synthetic pathway. The dsRNAs are processed into single-stranded RNA oligomers called microRNA (miRNA) or small interfering RNA (siRNA), depending on the origin of the dsRNA: miRNA is formed from irregular pseudo-hairpin miRNA-precursor molecules, whereas siRNA is formed from long double-stranded or hairpin RNA. The RNA oligomers then are inserted into RISC protein complexes, which they guide to matching sites on messenger RNA (mRNA) molecules encoded by the genome. When a match is found between a guiding oligomer and an mRNA, the RISC cleaves the mRNA, causing its destruction. In plants (the main focus of the present research), the oligomers directly involved in the silencing process are predominantly 21 bases in length, in both the miRNA and siRNA pathways (Eamens *et al*., 2008; Fusaro *et al*., 2006; Ossowski *et al*., 2008).

In synthetic RNAi applications, the RNA oligomers are used to suppress the expression of one or more target genes. An important task here is to formulate sequences for the oligomers (and in practice their precursors), so that they will have substantial impact on the targets, while minimizing their impact on non-target genes (i.e. cross-silencing). One strand of research in this respect concerns the identification of sites in mRNA where silencing is likely to occur (Reynolds *et al*., 2004; Santoyo *et al*., 2005; Yamada and Morishita, 2005). In addition, methods have been developed for assessing the silencing impact expected from any given candidate silencer, based on an identification of all matching sites in the mRNA (Rajewsky and Socci, 2004; Rehmsmeier *et al*., 2004). These methods use classical energy-minimizing RNA-folding techniques and are evidently highly reliable, but they are extremely demanding in terms of computational time.

Fast performance in identifying matching sites is important because the search for an effective design may require an assessment of large numbers of candidates. This is so especially in the case of long hairpin silencers, where the complexity of the design task is augmented by the spawning of multiple overlapping oligomers from a single hairpin RNA molecule.

A matching relationship involves a pairing of the reverse-complement of a silencer candidate with an mRNA location, with allowance for G–U variants and some tolerance for mismatches. A widely used technique for finding matches commences by taking the reverse-complement of the candidate silencer, for which a direct match is then sought in the mRNA, which allows the use of suffix-tree algorithms (Gusfield, 1997). The suffix-tree approach has been extended in recent years in the GUUGle software, which implements a direct search for complementary matches, including built-in provision for G–U matching (Gerlach and Giegerich, 2006). Suffix-tree algorithms have been adapted also to handle a wide range of mismatches, notably in the flexible pattern-matching capabilities provided by the STAN software (Nicolas *et al*., 2005).

This article proposes an alternative to the suffix-tree approach. The genome is represented as an in-memory map of 21-mers, and the matching rules are incorporated in an algorithm that seeks matches within this map. The aim has been to develop a fast search procedure with modest memory requirements, for use especially in optimization procedures for constructing effective hairpin silencers.

2 SYSTEM AND METHODS

In the remainder of this article, we refer to a silencing oligomer simply as a *silencer* and the mRNA sites which it matches as the silencer's *objects*. The impact made by a given silencer can be estimated by a search for

*To whom correspondence should be addressed.

 © The Author 2010. Published by Oxford University Press. All rights reserved. For Permissions, please email: journals.permissions@oxfordjournals.org

Downloaded from bioinformatics.oxfordjournals.org at Masaryk University on February 21, 2011

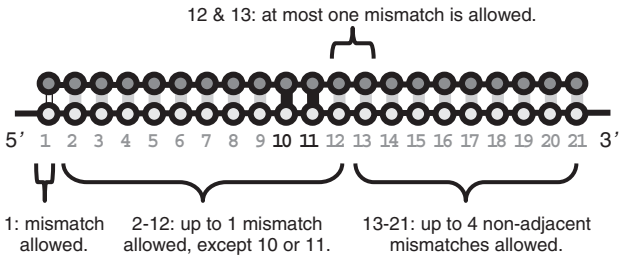

Fig. 1. Locational matching rules.

matching objects, followed by a quantitative assessment of silencing efficacy aggregated over all the silencer–object pairs. The latter assessment—which lies beyond the scope of the present article—involves estimation of the free energy of pairwise RNA foldings (the silencer–object pairs are linked to form hairpins and analyzed using self-folding techniques described by Zuker *et al.*, 1999); in the case of siRNA, an estimation is required also of strand-selection probability for the silencer.

Matching between the respective bases of a given silencer–object pair is determined by the canonical chemical bondings between RNA bases. The potential base-wise matches are C–G and G–C (three H-bonds), A–U and U–A (two H-bonds), together with the wobble pairs G–U and U–G (one H-bond). A silencer makes an *exact match* with an object if all the silencer's bases, taken in reverse order, match the corresponding bases in the object. For technical purposes, it is convenient to consider also a *one-off* rule allowing a single mismatched base pair.

Extensive recent research in plant biology (see Ossowski *et al.*, 2008; Schwab *et al.*, 2006) has shown that the possibility of silencing in reality is determined by the locations and configurations within the silencer–object pair of any mismatches, and has confirmed the effectiveness of silencers designed in the light of these findings. The findings are encapsulated in the 'locational' matching rules set out below and illustrated in Figure 1.

- Location 1: a mismatch is tolerated.
- Locations 2–12 (except 10 or 11): a single mismatch is tolerated.
- Locations 13–21: up to four mismatches can occur in this region, provided none are adjacent to each other or to a mismatch at location 12.

3 ALGORITHMS

3.1 Modeling of bases and sequences

Let each RNA base p be represented as a pair of Boolean variables (p', p'') with the following values in each case.

$$A = (0,0)$$
$$C = (1,0)$$
$$G = (0,1)$$
$$U = (1,1).$$

It follows from the base-wise matching rules (see Section 2 above) that a silencer-base p matches an object-base q if and only if both of the following conditions are true:

$$\sim p' = q', \text{ and} \tag{3.1}$$
$$\sim p'' \leq q''. \tag{3.2}$$

These conditions are verified in Table 1. Conditions (3.1) and (3.2) apply similarly to 21-mers, in that a silencer m matches an object o if and only if both conditions are true for each pair of bases (i, 22-i) in the two 21-mers. A 21-mer m can be recorded as a pair of 21-bit

Table 1. Encoding of base-wise matching rules

Silencer base	Object base	Condition (3.1)			Condition (3.2)		
		P'	q'	$\sim p' = q'$	p''	q''	$\sim p'' \leq q''$
A	A	0	0	0	0	0	
A	C	0	1	1	0	0	0
A	G	0	0	0	0	1	
A	U	0	1	1	0	1	1
C	A	1	0	1	0	0	0
C	C	1	1	0	0	0	
C	G	1	0	1	0	1	1
C	U	1	1	0	0	1	
G	A	0	0	0	1	0	
G	C	0	1	1	1	0	1
G	G	0	0	0	1	1	
G	U	0	1	1	1	1	1
U	A	1	0	1	1	0	1
U	C	1	1	0	1	0	
U	G	1	0	1	1	1	1
U	U	1	1	0	1	1	

Boolean arrays (m', m''), such that the parts (p', p'') of each base $m(i)$ are recorded in the corresponding $m'(i)$ and $m''(i)$. Then, for a silencer m and object o, the matching conditions can be given as:

$$\sim m'(i) = o'(22-i) \qquad \forall i : 1 \leq i \leq 21 \tag{3.3}$$
$$\sim m''(i) \vee o''(22-i) = o''(22-i) \qquad \forall i : 1 \leq i \leq 21 \tag{3.4}$$

These conditions can be tested efficiently on a modern computer using built-in arithmetic operations, with the arrays m' and m'' representing each 21-mer implemented as native machine words.

3.2 Search procedures

Let m be a silencer and let A be the set of all objects to be searched, normally comprising an entire genome. To evaluate silencer impacts, a procedure is required for finding all matches of m in A, including duplicates (i.e. multiple sites in A matched by m). As indicated in Section 2, this search is not limited to exact matches, but must include locational variants.

Such variants can be found by a tailored application of an exact matching procedure, as illustrated by the simpler case of one-off matching. Let $N_1(x)$ be the set of all words that differ from a word x in one position. To find all one-off matches of a silencer m in the object-sequence (or set) A, one approach is to carry out an interleaved search for any objects $o \in$ A such that $o \in N_1(m)$ (Gerlach and Giegerich, 2006). An equivalent approach (adopted in the present research) is to search with each member of $N_1(m)$ in turn, looking for an exact match of any of these in A (in fact, a suitably defined subset of $N_1(m)$ suffices for this purpose).

This approach to variant searching is illustrated in procedure *find-one-offs*, which finds all one-off matches of m in A by seeking the exact matches of the single-base variants of m. The following description of the procedure is simplified in that a search is actually required for only about half the variants of each base.

Downloaded from bioinformatics.oxfordjournals.org at Masaryk University on February 21, 2011

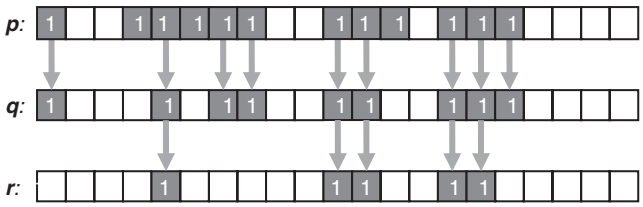

Fig. 2. Dominance with $p \geq q$ and $q \geq r \Rightarrow p \geq r$.

Procedure find-one-offs

(1) Define *one-off*s = $\emptyset$.
(2) For each base $m(i)$, $i \in (1,21)$:
  Set $m* = m$.
  For each variant $v \in$ (A,C,G,U) $\neq m(i)$:
    Set $m*(i) = v$.
    Find all exact matches of $m*$with objects
    in A, and record each pair ($m*$, $o*$) in *one-offs*.

The locational matching rules are handled in a similar way, through enumeration and testing of all variants satisfying those rules. The key to any search for locational matches is thus an exact-matching procedure. For this purpose, a specialized map of A is constructed prior to any searching. To build this map, A is defined as a list of word-pairs (o', o'') as indicated in Section 3.1, and the list is sorted by ascending values of o'. A is thus formed into clusters of objects, such that all objects with common values of o' are adjacent. The clusters are referred to here as *complement-subsets* S(A), and the members of a complement-subset $\varphi \in$ S(A) are defined as Boolean arrays $\{o'(\varphi), o''(\varphi)\}$, where by definition $o'(\varphi)$ is common to all members of φ.

Given this ordering, procedure *find-exact-matches* carries out the search for all exact matches of a silencer m with objects in A:

Procedure find-exact-matches

(1) Form a 21-mer $\underline{m}$ with reverse sequence to m.
(2) For Condition (3.3): perform a binary search amongst the complement-subsets of A to find a complement-subset φ with $o'(\varphi) = \sim\underline{m}'$.
(3) For condition (3.4): if a subset φ was found in Step 2, search for all object(s) $\{o\}$ in φ for which $\sim\underline{m}'' \vee o'' = o''$.

Step 3 can be specified as a search for dominance among Boolean arrays. Given two Boolean arrays p and q, the relation $p \geq q$ (p dominates q) is true if $p(i) = 1$ in every location i where $q(i) = 1$. Now consider Boolean arrays p, q, and r, where $p \geq q$ and $q \geq r$. In each location i with $r(i) = 1$, $q \geq r$ implies that $q(i) = 1$, and hence $p \geq q$ implies also that $p(i) = 1$; therefore $p \geq r$. The dominance relation is thus transitive, as illustrated in Figure 2; it is also evidently reflexive and antisymmetric; hence with the dominance relation, $\{p, q, r \ldots\}$ constitutes a partially ordered set.

Referring now to the Boolean arrays for silencers and objects ($\underline{m}''$ and o'', respectively) as m and o, a match of m with an $o \in \varphi$ requires $\sim m \vee o = o$, which is equivalent to $o \geq \sim m$. The search task in Step 3 is thus equivalent to seeking every o ($o \in \varphi$) that dominates $\sim m$. This can be done via a simple linear scan, visiting every element of φ, and the implementation of *find-exact-matches* tested in the present research uses such a scan. Thus *find-exact-matches* has complexity $O(\log_2|S(A)| + |\varphi|)$: in *Arabidopsis thaliana* |S(A)| = 2 096 609 (see Section 4), and usually $|\varphi| << 100$. The occasionally large size of φ does, however, pose the question of whether a faster search method can be devised for Step 3, where the task is a special case of a database operation called skyline query.

4 IMPLEMENTATION AND PERFORMANCE

The algorithms described above are implemented in software called oligomers matching objects (OMOs), which has been tested with the transcript of *Arabidopsis thaliana*, obtained from the web site of the Arabidopsis information resource (TAIR) (TAIR6_cdna_20051108 in FASTA format, from ftp://ftp.arabidopsis.org/home/tair/Genes/). OMO is written in C++, and all the tests were carried out on a Windows platform with an Intel Core™2 Duo CPU and 1.95 GB of RAM.

As indicated in Section 3.2, a prerequisite for the matching algorithm is to generate a map of complement-subsets. This is done once for each genome in a pre-processing module that reads the transcript, generates the map, and writes it in a binary form to a file (size 561, 668 kb) for subsequent input to OMO. For *Arabidopsis*, the pre-processing step requires around 168 s of CPU time, and subsequent input of the binary file by OMO requires around 6 s.

The *Arabidopsis* transcript comprises 46 447 255 bases which make up 45 756 240 distinct 21-mers within gene-sequences, and in the preprocessing step the 21-mers are apportioned among 2 096 609 complement-subsets. Each subset φ comprises the common $\varphi(o')$, together with an array of its component 21-mers. In this array, each 21-mer m has implicitly the common value $m' = \varphi(o')$; the explicit attributes of m (packed into two 32-bit words) are m'', the number of instances of m in its gene $G(m)$, and an index to $G(m)$. For the main tests conducted here the consequent total memory requirement is ~428 000 kb, which presents no practical difficulty on a standard 16-bit Windows PC (the more compact internal representation makes total memory smaller than the binary input file).

The OMO tests involved searching for matches between randomly generated silencers and 21-mers of the genome. Searches were conducted for the three match types defined in Section 2 (we emphasize that only the locational rules have biological significance). Three main series of tests were carried out, each with a different set of 100 000 silencers. The silencer sets are defined as follows.

- SM-R: each silencer m is obtained as a pair (m', m'') of numbers according to the encoding described in Section 3.1, both numbers being randomly distributed between 0 and $2^{21} - 1$.
- SM-M: silencers are generated so as to match 21-mers of the genome. They are obtained as the reverse-complements of 21-mers selected at random from all silencers in the genome.
- SM-T: silencers are generated so as to match 21-mers of the largest complement-subset of the genome. The silencers are obtained as the reverse-complements of 21-mers selected at random from that subset.

In each case, a separate search was made for every silencer, with the *find-exact-matches* procedure as the kernel for all variant matches (see Section 3.2). Separate tests with smaller silencer-sets (with 100, 1000 and 10 000 silencers) confirmed that search time per silencer was essentially independent of the number of silencers tested in a given run (see Supplementary Material).

Downloaded from bioinformatics.oxfordjournals.org at Masaryk University on February 21, 2011

Table 2. Test results for OMO, 100 000 silencers

Silencer set	Match type	Silencers matched	Total matches	CPU per silencer (s)
SM-R	exact	9031	18 552	1.090E-06
SM-R	one-off	61 075	527 933	2.063E-05
SM-R	locational	99 072	16 953 612	4.511E-04
SM-M	exact	100 000	233 736	1.870E-06
SM-M	one-off	100 000	1 233 144	3.125E-05
SM-M	locational	100 000	21 584 039	8.975E-04
SM-T	exact	100 000	24 468 095	3.550E-04
SM-T	one-off	100 000	153 533 332	8.498E-03
SM-T	locational	100 000	704 152 342	1.543E-01

Table 2 summarizes the results. The third column shows the number of silencers (out of the 100 000 tested in each case) for which at least one match was found in the genome using the nominated match type. For the SM-M and SM-T silencer-sets these figures are 100 000 for all match-types because the silencers in these cases were defined deliberately to find matches in the genome. The fourth column shows the total number of sites matched in the genome. For example, with SM-M there were 100 000 silencers and 233 736 sites exactly matched, indicating that on average there were ∼2.34 instances in the genome of the reverse-complement of each silencer. The fifth column shows the average CPU time to find all matches for a silencer, averaged over all 100 000 silencers in each test. These times cover all search operations carried out for each silencer, excluding input operations and the generation of the silencers.

The results for SM-R and SM-M give a precise view of the way in which the partial matching conditions expand the number of matches found, and hence the complexity of the search task (see Column 4). A notable point here is the substantial difference between SM-R and SM-M, with SM-M making relatively fewer partial matches than SM-R (e.g. a ratio of exact to locational matches of 1 : 914 for SM-R versus 1 : 92 for SM-M), even though the actual number of matches for SM-M are consistently higher. A possible explanation is that for many of its 21-mers the genome includes one or more exact duplicates (more than if the genome were formed at random); but with fewer variant forms than might be expected if the genome were constructed at random.

The SM-T silencers were defined to match 21-mers in the largest subset, with the aim of assessing the impact on performance of within-subset searching (Step 3 of *find-exact-matches*). The results for SM-T show that this impact is quite strong, with a 326-fold increase in CPU time compared with SM-R for exact matches. This is still much faster than linear search within the genome as a whole (see below), and there is no reason to think that genomes with predominantly large subsets exist in nature.

To establish a performance baseline for the above results, further tests were carried out using a linear search procedure instead of *find-exact-matches*. Given a silencer m, this procedure tests m for an exact match with each 21-mer of the genome in turn, using the same implementation of the matching test as that used elsewhere (see Section 3.1). CPU time with this baseline procedure was 4.75 s/silencer, averaged over 100 SM-R silencers. This is 4 357 798 times greater than the corresponding result with *find-exact-matches*, and confirms the efficiency of the latter procedure.

Table 3. GUUgle: average CPU times (seconds/silencer) for exact matching

Silencer set	10 silencers	100 silencers	1000 silencers	10 000 silencers	100 000 silencers
SM-R	6.716E-01	9.654E-02	1.447E-02	2.945E-03	7.256E-04
SM-M	7.358E-01	9.732E-02	1.495E-02	3.159E-03	7.112E-04
SM-Ra	2.593E-01	5.531E-02	1.034E-02	2.533E-03	6.844E-04
SM-Ma	2.406E-01	4.780E-02	9.998E-03	2.664E-03	6.617E-04

Additional tests were run to obtain comparable performance figures with GUUGle (Gerlach and Giegerich, 2006). In these tests, GUUGle (from http://bibiserv.techfak.uni-bielefeld.de/guugle/) was applied to searches for the SM-R and SM-M silencers as described above. Because GUUGle builds a suffix tree for its target sequences (here the whole of the *Arabidopsis* transcript) incrementally, its performance depends on the number of query-strings tested. For this reason, the GUUGle tests were run with different numbers of silencers; also, to exclude the time required for initial setup and input, adjusted CPU figures (SM-Ra and SM-Ma) were obtained in each case by deducting the search time for a single silencer from total CPU time.

The results (see Table 3) show that OMO outperforms GUUGle by factors of 628 : 1 and 354 : 1, respectively in exact matching of the SM-R and SM-M 100 000 silencer-sets, and by even more for smaller numbers of silencers. GUUGle on the other hand has memory usage of around 250 000 kb, which is <60% of the memory used by OMO.

Similar comparisons have not been possible with STAN (Nicolas *et al*., 2005), which is not available for stand-alone testing. However, the STAN Manual (Valin *et al*., 2007) includes performance data, including comparisons with other pattern-oriented software called PatMatch, PatScan and Genlang. For exact literal matches of a 24-mer in *Arabidopsis*, the search times with STAN and PatMatch were 1.17 and 1.01 s, respectively, considerably faster than for PatScan and Genlang. STAN was also tested with several more complex patterns, none of which, however, allow direct comparison with the results given above for locational matches. We conclude that OMO has a clear performance advantage over these codes.

5 DISCUSSION

This article has presented techniques to enable rapid prediction of the locations where a given silencer is likely to have impact on a given genome. The techniques as implemented in the OMO software have been shown to be faster than GUUGle by a factor of at least 350 and, therefore, will be valuable where performance is critical (e.g. in optimization procedures). OMO is thus well-suited to applications—notably optimization—requiring large numbers of searches. For *ad hoc* queries, the overhead associated with initial input of the binary file could be eliminated through a persistent-memory implementation (e.g. providing OMO as a web service).

Other notable aspects of the research include the development of a specialized coding scheme for direct testing of matches (including G–U matches), the elucidation of reductions for testing of mismatches, and the identification of a fundamental search task—a Boolean variant of skyline search—which apparently has not been addressed previously.

Downloaded from bioinformatics.oxfordjournals.org at Masaryk University on February 21, 2011

The main disadvantage of the approach presented here is its inflexibility compared with alternatives that allow arbitrary silencer lengths (GUUGle and STAN) or declarative statements of matching rules (STAN). In contrast with such alternatives, low-level changes would be required to model alternative silencing processes (e.g. alternative matching rules, or silencer oligomers of length other than 21; see Fusaro *et al*., 2006, Ossowski *et al.*, 2008). Even so, the changes required in these cases would preserve the performance advantages demonstrated in this article, and would not necessarily involve any fundamental technical difficulty.

ACKNOWLEDGEMENTS

We thank Andreas Ernst and Bob Anderssen for their encouragement, and for valuable advice. We also thank the anonymous reviewers for their useful comments on earlier versions of the paper.

Conflict of Interest: none declared.

REFERENCES

Eamens,A. *et al*. (2008) RNA silencing in plants: yesterday, today, and tomorrow. *Plant Physiol.*, **147**, 456–468.

Fusaro,A.F. *et al.* (2006) RNA interference-inducing hairpin RNAs in plants act through the viral defence pathway. *EMBO Rep.*, **7**, 1168–1175.

Gerlach,W. and Giegerich,R. (2006) GUUGle: a utility for fast exact matching under RNA complementary rules including G-U base pairing. *Bioinformatics*, **22**, 762–764.

Godfrey,P. *et al.* (2006) Algorithms and analyses for maximal vector computation. *VLDB J.*, **16**, 5–28.

Gusfield,D. (1997) Algorithms on strings, trees and sequences: computer science and computational biology. Cambridge University Press, Cambridge.

Nicolas,J. *et al.* (2005) Suffix-tree analyzer (STAN): looking for nucleotidic and periodic patterns in chromosomes. *Bioinformatics*, **21**, 4408–4410.

Ossowski,S. *et al*. (2008) Gene silencing in plants using artificial microRNAs and other small RNAs. *Plant J.*, **53**, 674–690.

Papadias,D. *et al*. (2005) Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, **30**, 41–82.

Rajewsky,N. and Socci,N.D. (2004) Computational identification of microRNA targets. *Dev. Biol.*, **267**, 529–535.

Rehmsmeier,M. *et al*. (2004) Fast and effective prediction of microRNA/target duplexes. *RNA*, **10**, 1507–1517.

Reynolds,A. *et al*. (2004) Rational siRNA design for RNA interference. *Nat. Biotechnol.*, **22**, 326–330.

Santoyo,J. *et al*. (2005) Highly specific and accurate selection of siRNAs for high-throughput functional assays. *Bioinformatics*, **21**, 1376–1382.

Schwab,R. *et al.* (2006) Highly specific gene silencing by artificial microRNAs in Arabidopsis. *Plant Cell*, **18**, 1121–1133.

Valin,A.-S. *et al*. (2007) STAN manual. IRISA/INRIA. Available at http://genoweb1.irisa.fr/Serveur-GPO/outils/patternMatching/STAN/ (last accessed date 2 July, 2010).

Yamada,T. and Morishita,S. (2005) Accelerated off-target search algorithm for siRNA. *Bioinformatics*, **21**, 1316–1324.

Zuker,M., Mathews,D.H. and Turner,D.H. (1999) Algorithms and thermodynamics for RNA secondary structure prediction: a practical guide. In Barciszewski,J. and Clark,B.F.C. (eds) *RNA Biochemistry and Biotechnology*. Kluwer Academic Publishers, Dordrecht.

APPENDIX A: VARIANT SEARCH REDUCTIONS

This Appendix shows how the canonical base-wise matching rules (including G–U pairings) allow a significant reduction in the extent of search required with variant (e.g. locational) matching rules. For example, searching for the one-off matches of a given silencer would require in principle that match-searches be carried out for each of the three variant-bases at each of the silencer's 21 locations, that is, for a total of 63 one-off variants of the silencer as a whole. We show here how this number can be reduced substantially. Consider first the object-bases matched by base G, and by the one-offs (A, C, U) of G (Table A1).

Table A1. Exact and one-off matches of base G in silencer

Exact matches	One-off matches
G with C or U	A with U C with G U with G or A

Table A2. One-off reductions

Silencer base (matching object bases)	One-offs of silencer base (matching object bases)
A(U)	~~C(G)~~, G(U,C), U(G,A)
C(G)	~~A(U)~~, G(U,C), U(G,A)
G(C,U)	~~A(U)~~, ~~C(G)~~, U(G,A)
U(A,G)	~~A(U)~~, ~~C(G)~~, G(U,C)

Both G and its one-off A match U, and one-offs C and U both match G. Hence, for a base G in the silencer the one-off tests for A and C can be omitted without compromising the opportunity to find these matches. Table A2 shows the full set of such reductions.

Thus, the testing of one-off base-wise variants can be restricted as follows.

- A or C in silencer: test only one-offs G and U (omit C).
- G in silencer: test only one-off U (omit A and C).
- U in silencer: test only one-off G (omit A and C).

APPENDIX B: SIZES OF COMPLEMENT-SUBSETS

Subset sizes in practice: Figure B1 shows the distribution of sizes of complement-subsets in the *Arabidopsis* transcript. Three series are shown, defined as follows.

- *Subsets fully reduced*: excluding duplicates and internally dominated objects.
- *Subsets partly reduced*: excluding duplicates.
- *Subsets raw*: no exclusions.

The maximum subset sizes according to these definitions are 1406, 2889 and 8140, respectively. The frequency distribution is skewed heavily to smaller sizes; for instance, for the fully reduced subsets the most frequent size is 10 (129 553 instances), and the largest three sizes are 662, 728 and 1406, with one subset each.

Upper bounds: The maximal complement-subset φ^{**} may be defined as the distinct possible values of a Boolean array of length 21, and the effective maximal set φ^{*} as the largest subset of φ^{**} that contains no objects dominated by others in φ^{*}. Clearly $|\varphi^{**}|=2^{21}$ is an upper bound on subset size in general, while $|\varphi^{*}|$ is the maximum number of dominant objects in a subset.

Downloaded from bioinformatics.oxfordjournals.org at Masaryk University on February 21, 2011

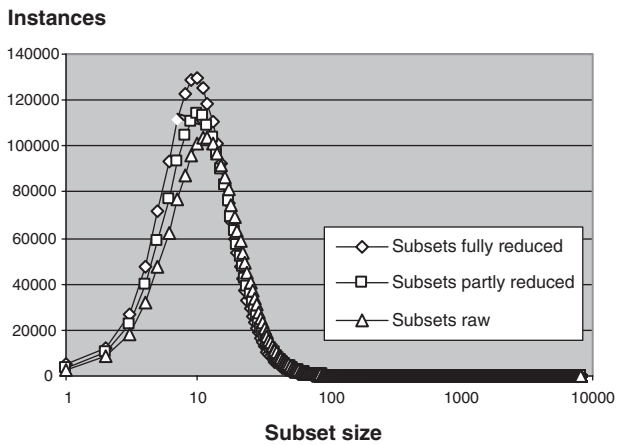

Fig. B1. Complement-subsets in *Arabidopsis*.

What is the maximum possible size of $|\varphi^*|$? To answer this question we begin by defining the cardinality of an object (a Boolean array of length 21) as the number of its elements that are set to 1. Now consider the maximal set φ_k of distinct instances of an object with cardinality k ($k \leq 21$). Because the members of φ_k are distinct and have equal cardinality, no mutual dominance can occur. The size of φ_k is 21 things taken k at a time,

$$|\varphi_k| = \binom{21}{k} = \frac{21!}{k!(21-k)!}$$

The question now arises, does the maximal set φ^* comprise objects with a single cardinality (i.e. $\exists k : \varphi^* = \varphi_k$), or is it a mixture of cardinalities? If the latter is the case, consider the largest cardinality *kmax* of any object in φ^*. If $|\varphi^*|$ could be increased by removing any object of cardinality *kmax*, $|\varphi^*|$ could be further increased by removing other objects of the same cardinality, until all such objects were removed. Hence, $\varphi \subseteq \varphi^*$. But any object with cardinality less than *kmax* would be dominated by some object in φ_{kmax}; therefore, φ^* is not a mixture, and in particular, $\varphi^* = \varphi_k$ for a single cardinality k. The cardinality that maximizes $|\varphi_k|$ is evidently $k* = 21/2 = 10$ or 11, yielding $|\varphi_{k*}| = |\varphi^*| = 352\,716$.

APPENDIX C: SEARCH IN COMPLEMENT-SUBSETS

The search for non-dominated Boolean arrays in the *find-exact-matches* procedure (Section 3.2) is a special case of the database operation called *skyline query*. A skyline is a set of tuples in a given database, of which each tuple is maximal with respect to at least one of a given set of attributes (Papadias *et al*., 2005). The task of retrieving such a set is also called the maximal vector problem (Godfrey *et al*., 2006).

Despite these formal affinities, the present search task merits specialized treatment on account of its distinctive features. In particular, the criteria are all Boolean-valued, and the size of the set φ to be searched is in most cases very small, making a linear scan very attractive. Some instances of φ, however, comprise hundreds and even thousands of objects (Appendix B). An interesting question then is whether a faster-than-linear search is possible.

Some alternative procedures are outlined below. The task in each case is to find any Boolean arrays o ($o \in \varphi$) that dominate a given $\sim m$. The procedures involve pre-calculation of summary quantities, and specialized orderings of the complement-subset φ. In computational experiments with *Arabidopsis*, no consistent performance improvement has been achieved over linear search.

Bit-count ordering: Let $nbits(p)$ denote the number of non-zero elements in a Boolean array p. It is evident that $p \geq q \Rightarrow nbits(p) \geq nbits(q)$. One can make use of this fact by applying in advance a descending bit-count ordering to φ, that is, by ordering the elements of φ by the numbers of one-valued bits in them (e.g. 21-bit objects then 20-bit objects and so on). Then a search for instances of domination of $\sim m$ by members of φ can cease when it reaches an $o \in \varphi$ with $nbits(o) < nbits(\sim m)$.

Dominance ordering: The dominant objects in φ may be defined as those that are not dominated by other members of φ, while the dom-set s of a dominant object o_s comprises o_s together with any other objects which it dominates. A descending bit-count ordering (see above) may be applied both to dominant objects and their dom-sets. A procedure to find all matches of m in a set φ' of dom-sets ordered in this way is given below.

Procedure find-domset-matches

(1) Define initially $V = \emptyset$.
(2) For each dom-set $s \in \varphi'$':
If $nbits(o_s) < nbits(\sim m)$, stop.
If $o_s \geq m$, do the following.
Add o_s to V, then perform a linear search of the other members of s, adding any of these that dominate $\sim m$ to V. This search can terminate when an object is reached with fewer bits than $nbits(\sim m)$.

Dominance reduction: Where the aim is merely to determine whether a match exists, the above search strategy can be simplified by removing in advance all non-dominant objects from φ'. This reduces the effective size of φ and makes the search again simply linear, without compromising the outcome. Note that if an $\sim m$ is dominated by a dominant object o, there is no guarantee that $\sim m$ is dominated by all members of o's dom-set. Consequently a dominance-reduction method (e.g. removing dominated objects but retaining the number of objects per set) cannot reliably yield the number of matches made by m.

Summary objects: A further device to reduce the scope of search involves defining in advance a 'summary object' o_φ as the Boolean OR of all the members of φ. Then a search for a match of m in φ may commence with a test for $o_\varphi \geq \sim m$: failure of this test indicates that there can be no $o* \geq \sim m$ for any $o* \in \varphi$. Unfortunately where φ is large, $nbits(o_\varphi) \approx 21$, and for this reason the test by itself is unlikely to improve performance. However, a composite application could be more effective. For example, with dominance ordering the dom-sets of φ may be partitioned in groups with a common number of bits in each group, and a summary object $o_{\varphi(k)}$ may be predefined for each such group. Then when searching for a match of m, if $o_{\varphi(k)} \geq \sim m$ the group can be ignored.

Downloaded from bioinformatics.oxfordjournals.org at Masaryk University on February 21, 2011