Vol. 24 no. 5 2008, pages 696–703
doi:10.1093/bioinformatics/btn001

Systems biology

Interpool: interpreting smart-pooling results

Nicolas Thierry-Mieg* and Gilles Bailly†
TIMC-IMAG, CNRS UMR5525, Faculte de Medecine, 38706 La Tronche Cedex, France

Received on July 6, 2007; revised on October 17, 2007; accepted on January 1, 2008
Advance Access publication January 9, 2008
Associate Editor: Alfonso Valencia

ABSTRACT

Motivation: In high-throughput projects aiming to identify rare positives using a binary assay, smart-pooling constitutes an appealing strategy liable of significantly reducing the number of tests while correcting for experimental noise. In order to perform simulations for choosing an appropriate set of pools, and later to interpret the experimental results, the pool outcomes must be 'decoded'. The intuitive aim is clearly to identify the positives that gave rise to an observation, whether real or simulated. However, this goal is not well-formalized and has been the focus of very few studies.

Results: We first provide a clear combinatorial formalization of the 'decoding problem'. We then present *interpool*, an exact algorithm to solve this problem. An efficient implementation is freely available. Its usefulness is illustrated in the context of yeast-two-hybrid interactome mapping with the Shifted Transversal Design.

Availability: The implementation, licensed under the GNU GPL, can be downloaded from http://www-timc.imag.fr/Nicolas.Thierry-Mieg/

Contact: nicolas.thierry-mieg@imag.fr

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

1 INTRODUCTION

Many high-throughput projects rely on a basic yes-or-no test, and aim to identify a few rare positives in a large collection of 'objects' (clones, proteins, peptides, etc.). The main issue is obviously to obtain information as efficiently as possible, but another major difficulty stems from the fact that biological assays can be somewhat noisy: false positives and false negatives can and do occasionally occur. However, often the basic assay can be applied to a pool of objects, yielding a positive readout if the pool contains at least one positive. When this is the case, smart-pooling constitutes an appealing approach that can significantly reduce the number of tests while providing the power to correct the experimental errors (Barillot *et al*., 1991; Bruno *et al*., 1995; Jin *et al*., 2007; Thierry-Mieg, 2006a; Vermeirssen *et al*., 2007).

The strategy consists in assaying well-chosen pools, such that each object is present in several pools (i.e. the pools are redundant). Pools are designed so that all positive objects can usually be identified from the pattern of positive pools, and when this is not the case, only a few candidates need to be retested. In addition, the pools' redundancy means that each object is tested several times: this provides a potential increase in both sensitivity and specificity.

The first difficulty is to choose a 'good' set of pools. This is the focus of the so-called pooling problem, or combinatorial group testing problem: given bounds on the numbers of positives and errors, this problem consists in designing a set of pools that guarantees the correction of all errors and the identification of all positives. Several mathematical constructions satisfying this so-called 'guarantee requirement' have been described (e.g. Thierry-Mieg, 2006b and references therein).

Once the design has been chosen and the pools are constructed, they can be assayed in every 'condition' of interest, giving rise to an observation for each condition. For example in a yeast-two-hybrid experiment, the preys can be smart-pooled and then screened against individual baits, as in Jin *et al*. (2007). Each bait can be seen as a condition, and the resulting observation takes the form of a score for each pool (often simply 'negative' or 'positive'). The goal is then to identify the positive objects (preys) that produced this observation. Whether this is achievable or impossible depends on the pooling design, and also on the numbers of positives and errors that actually occurred. But even when it is theoretically feasible, doing it is not algorithmically trivial: this is referred to as the decoding problem, although it has never been formally defined as far as we know.

Proving that a pooling design satisfies the guarantee requirement, is sometimes achieved by exhibiting a provably correct decoding algorithm (e.g. Thierry-Mieg, 2006b). However, such algorithms rely on the given bounds on numbers of positives and errors. In real experiments, these bounds are unknown but can be very large. Attempting to estimate and use the true bounds when selecting the set of pools would lead to inefficient designs, with many pools that would be useless except perhaps in extremely rare instances. In practice, if one accepts that in a small fraction of instances the experiment will fail to identify every positive, designs with many fewer pools can be used. A drawback is that the previous decoding algorithms cannot be used: more general algorithms that do not assume knowledge of the maximum numbers of positives and errors are needed.

Such general algorithms are also crucial for selecting a pooling design well-adapted to the experimental context. Indeed, the numbers of positives and errors will vary in each tested condition, and can usually only be roughly estimated beforehand. Large numbers of simulations should be performed, using

*To whom correspondence should be addressed.
†Present address: LIG, University of Grenoble 1, BP 253, 38041 Grenoble Cedex 9, France.

various reasonable values for the expected rates of positives and errors, in order to find the right compromise between minimizing the number of pools that have to be built and screened, and maximizing the sensitivity and specificity in the conditions of interest. Consequently, the algorithms must be efficient.

A decoding algorithm called the Markov Chain Pool Decoder (MCPD) has been proposed (Knill *et al.*, 1996), and an implementation is available. Given an observation, it estimates the posterior probability that each object is positive. The formal problem addressed by MCPD is related to the decoding problem implicit in combinatorial group testing and to its generalization presented in Section 2: there is a link between the likelihood function of (Knill *et al.*, 1996) and the distance minimized in the combinatorial problem, although determining the exact relationship would require further studies. However, because it relies on a Markov Chain Monte Carlo method, MCPD's accuracy depends on the number of steps and there is no easy way to know when convergence has been attained. In addition, although it is fast enough to be used for decoding real observations, its speed becomes a limiting factor when performing large numbers of simulations. Finally, MCPD is a stochastic algorithm, while we wished to study the decoding problem from the deterministic point of view, maintaining a direct link with the classic decoding problem implicit in combinatorial group testing.

In this article, we present a new exact algorithm for interpreting smart-pooling results: *interpool*. We first provide a clear formalization of the decoding problem. In Section 3, we give the theoretical basis of our method. We finally present the algorithm, and illustrate its use in the context of an interactome mapping project.

2 FORMALIZING THE DECODING PROBLEM

NOTATION. *Let $\mathcal{V}$ be a set of Boolean variables. We call pool a subset of $\mathcal{V}$. A pooling design, noted $\mathcal{D}$, is a set of pools. In the whole article, we will consider that $\mathcal{V}$ and $\mathcal{D}$ are given once and for all.*

When performing an experiment, each pool produces a signal which is a priori continuous, although it should hopefully be highly contrasted for high-thoughput yes-or-no assays. This signal is then interpreted by the user or image-analysis software to obtain a discrete outcome for each pool, typically chosen among the two values 'positive' and 'negative'. However, in general it could be interesting to use more than two values, because higher confidence can be placed into the strongest and weakest signals than in the intermediate ones. In this context, assaying a set of pools in a given condition yields an observation, defined as follows.

NOTATION. *Let Ω be the set of values representing the possible discrete outcomes of a pool.*

For example, one could allow four outcomes of increasing strength for each pool with $\Omega = \{\text{NONE}, \text{FAINT}, \text{WEAK}, \text{STRONG}\}$.

DEFINITION 1. *An observation is a mapping between $\mathcal{D}$ and Ω: to each pool, it associates its discretized outcome. Equivalently, an observation can be seen as a vector in $\Omega^{|\mathcal{D}|}$.*

Given an observation, the goal is to identify the positive variables. An implicit hypothesis is that although the assays may yield more or less continuous signals, the variables themselves are truly Boolean. For example, in the context of yeast-two-hybrid—and particularly in a high-throughput setting—we must assume that either a pair of proteins do have the potential to interact physically, or they do not. It follows that conceptually, each pool's outcome should ideally also be Boolean: in the absence of noise, a pool should be positive if and only if it contains a positive variable. This leads to the following definitions.

DEFINITION 2. *An interpretation is a mapping between $\mathcal{D}$ and $\{0, 1\}$, or equivalently it is a vector in $\{0, 1\}^{|\mathcal{D}|}$: to each pool, it associates a unique value in $\{0, 1\}$ meaning that the pool is respectively negative or positive in this interpretation.*

DEFINITION 3. *An interpretation ι is consistent if there exists a mapping between $\mathcal{V}$ and $\{0, 1\}$ such that the value of each pool, defined as the disjunction (logical OR) of the pool's variables' values in this mapping, is equal to its value in the interpretation ι.*

NOTATION. *Let $\mathcal{C}$ be the set of consistent interpretations.*

$\mathcal{C}$ is totally determined by $\mathcal{V}$ and $\mathcal{D}$, although it is typically huge. For example, in a modest interactome project where we wish to detect up to five positives among 1000 preys, there must be at least $\binom{1000}{5} \approx 2^{50}$ different consistent interpretations. Hence $\mathcal{C}$ cannot be computed. However, the following simple algorithm allows to test whether an interpretation is consistent, and simultaneously decode it if it is:

(1) variables that appear in a negative pool are negative;
(2) the remaining variables are positive if they are the only non-negative variables in at least one positive pool, and 'ambiguous' otherwise.
(3) Finally, if every positive pool contains at least one non-negative variable, the interpretation is consistent.

Ambiguous variables are those for which there is no clear evidence one way or the other. These need to be retested in confirmatory assays if completeness is sought. However, with a well-chosen pooling design, they should only occur when the number of positives is much larger than expected.

In order to decode an observation, the only component that is still missing is a relationship between observations and interpretations. First, a mapping between Ω and $\{0, 1\}$ has to be specified: one must decide whether each outcome of Ω is to be preferentially interpreted as positive or negative. When $|\Omega| = 2$, there are only two possible outcomes: we can choose to call them NEG and POS, i.e. $\Omega = \{\text{NEG}, \text{POS}\}$, which can be naturally mapped to $\{0, 1\}$. In the general case where $|\Omega| > 2$, it is less trivial yet the mapping is usually implicit to the choice of the values and cutoffs (with regards to the underlying continuous signal) that define Ω.

NOTATION. *We will note $\mathcal{N}$ and $\mathcal{P}$ the subsets of Ω that map to 0 and 1, respectively.*

Given such a mapping between Ω and $\{0, 1\}$, any observation can be associated to the unique interpretation induced by this mapping. This is called the observation's canonical interpretation. In the absence of experimental errors, the canonical interpretation of any observation is necessarily consistent. Indeed, the mapping that associates each variable to its true value clearly works. But in general, an observation's canonical interpretation can be inconsistent. In fact, if $\mathcal{D}$ is well-chosen, this should always be the case when the observation contains errors. Therefore, we need to specify the relationship between an observation and an arbitrary interpretation.

To this end, let us assume given a 'distance' δ between each element of Ω and each of 0 and 1. This distance $\delta : \Omega \times \{0,1\} \rightarrow I\!N$ must satisfy the following rules:

- the distance between $\omega \in \Omega$ and its canonical mapping is 0;
- the distance between $\omega \in \Omega$ and the element of $\{0,1\}$ that it does not map to, noted δ_ω, is a strictly positive integer.

By extension, this also defines the distance between an observation and an interpretation, simply by summing the distances over all pools as follows.

Definition 4. *The distance between an observation* o *and an interpretation* ɪ *is defined as:*

$$\delta(o, \text{I}) \triangleq \sum_{p \in \mathcal{D}} \delta(o(p), \text{I}(p)) .$$

Note that δ is fully defined by $\mathcal{N}$, $\mathcal{P}$ and the value of δ_ω for each $\omega \in \Omega$.

Definition 5. *We can now formally define the decoding problem as we see it. Consider given a set of possible assay outcomes* Ω *and a distance* δ. *For any observation* o, *the goal is to identify the set* $\mathcal{S}(o)$ *of consistent interpretations at minimal distance to* o: $\mathcal{S}(o) \triangleq \{\imath \in \mathcal{C} \mid \delta(o, \imath) = \Delta(o)\}$, *where* $\Delta(o) \triangleq min_{\imath \in \mathcal{C}}\{\delta(o, \imath)\}$, *i.e.* $\Delta(o)$ *is the distance between* o *and its nearest consistent interpretation*(*s*).

This general framework is illustrated in the following three increasingly flexible instances.

First, consider the case $\Omega = \{\text{NEG}, \text{POS}\}$. We can define δ by $\delta(\text{NEG}, 0) = \delta(\text{POS}, 1) = 0$, and $\delta(\text{NEG}, 1) = \delta(\text{POS}, 0) = 1$. Equivalently and perhaps more intuitively, this same distance could be defined by $\mathcal{N} = \{\text{NEG}\}$, $\mathcal{P} = \{\text{POS}\}$, and $\delta_{\text{NEG}} = \delta_{\text{POS}} = 1$ [recalling that in this case $\delta_{\text{NEG}} = \delta(\text{NEG}, 1)$ and $\delta_{\text{POS}} = \delta(\text{POS}, 0)$]. The distance between an observation and an interpretation is then the standard Hamming distance. This first instance of the decoding problem is precisely the one that is implicit in the context of the combinatorial group testing problem, where the error model is simply defined by the total number of errors. Note that although in this case we have a true metric distance in the topological sense, this is not required in our framework, and is not true in the following instances.

A limitation of the above is that it makes no distinction between false positives and false negatives. However, in many assays the error-rates can be very different for these two types of errors. This can be taken into account, still using $\Omega = \{\text{NEG}, \text{POS}\}$, by defining the distance δ such that $\delta_{\text{NEG}} \neq \delta_{\text{POS}} : \delta_{\text{NEG}}$ and δ_{POS} can be seen as the respective 'costs' of false negative and false positive outcomes. Different choices for δ_{NEG} and δ_{POS} can lead to different solutions $\mathcal{S}(o)$.

For example, in a project where sensitivity is a major goal or where confirmatory retests can be quickly and cheaply performed, one could choose $\delta_{\text{NEG}} = 1$ and $\delta_{\text{POS}} = 2$. This would lead to interpretations in $\mathcal{S}(o)$ with potentially more positive pools, resulting in a larger set of putative positives than that obtained using the Hamming distance. Some of the additional putative positives might not be true positives and would therefore be eliminated when they failed to retest, but others could be genuine true positives that had an exceptionally high number of false negative assays, causing them to be missed with the Hamming distance.

This model still has one shortcoming: it assumes that the assays are truly binary and allows only two outcomes, positive or negative. Any information that may have been available with regards to the strength of the signal is lost and cannot be used in the decoding. For some assays this can be significant. For example, in yeast-two-hybrid interactome mapping, the assay readout for true positives can vary widely in intensity, ranging from a strong unmistakable signal to a weakish one that could easily be a false positive.

This situation can be taken into account by using more than two discrete outcomes. For example, consider Ω and δ defined by: $\Omega = \{\text{NONE, FAINT, WEAK, STRONG}\}$, $\mathcal{N} = \{\text{NONE, FAINT}\}$, $\mathcal{P} = \{\text{WEAK, STRONG}\}$, $\delta_{\text{NONE}} = 2$, $\delta_{\text{FAINT}} = 1$, $\delta_{\text{WEAK}} = 2$, and $\delta_{\text{STRONG}} = 4$. This model allows four discrete outcomes. NONE is the typical negative signal. The FAINT signal is also a priori negative, but can easily be considered as a false negative ($\delta_{\text{FAINT}} = 1$). WEAK represents a moderate positive signal, and STRONG is reserved for very clear signals that are unlikely to be false positives ($\delta_{\text{STRONG}} = 4$).

As shown, Ω and δ can be used to specify a wide variety of models for the experimental errors.

In the beginning of this section, we gave a simple algorithm for testing whether an interpretation is consistent and identifying the putative positives if it is. Assuming $\mathcal{S}(o)$ is known, this algorithm can be applied to every interpretation of $\mathcal{S}(o)$. When $|\mathcal{S}(o)| > 1$, several strategies can be employed for merging the decoding results of the different nearest consistent interpretations (e.g. intersection, union, majority,...), but it is a matter of policy: in any case identification of $\mathcal{S}(o)$ is required, and clearly constitutes the computational bottleneck.

One could imagine using the following naïve algorithm:

(1) $d = 0$.

(2) Test the consistency of every interpretation at distance d.

(3) If no interpretation was consistent, increment d and go to (2).

However, the number of interpretations at a given distance d increases exponentially with d. In addition, if the set of pools is well chosen, the distance to the nearest consistent interpretation should by and large be proportional to the number of observation errors; otherwise, the number of pools should typically be increased to avoid 'mis-taggings', i.e. erroneous decoding results. Therefore the naïve algorithm cannot be used with realistic error-rates.

One tempting idea would be to use integer linear programming (ILP) methods. Indeed, it is straightforward to express the decoding problem as an ILP problem, which can then be solved using general software such as GLPK or lp_solve. We successfully applied this approach on small toy examples (100 variables). However, in our hands the ILP solvers' performance degrades rapidly when attempting to scale up to realistic problem sizes: in the specific context of the decoding problem, the ILP approach does not appear more powerful than the naïve algorithm, leaving us in need of a better solution.

3 METHODS

In this section, we present the concepts that underlie the *interpool* algorithm. The notions of 'conflicting variables' and 'conflicting pools' are introduced. We then define the 'score' of a set of conflicting negative pools, and the 'closure' of a set of variables. Finally, we show that the decoding problem can be solved by finding the set of maximal-scoring closures of conflicting variables.

For clarity, all definitions and theorems are presented in the simplest instance of the decoding problem, i.e. where $\Omega=\{\text{POS}, \text{NEG}\}$ and $\delta_{\text{POS}}=\delta_{\text{NEG}}=1$. They can easily be generalized to arbitrary Ω and δ, and all proofs still hold. Proofs are provided in the supplementary materials.

3.1 Notations

$\forall v\in\mathcal{V}$, we note $\pi(v)\triangleq\{p\in\mathcal{D}\mid v\in p\}$ the set of pools that contain v. Given an interpretation I, we note $N(\text{I})$ and $P(\text{I})$ the set of negative and positive pools (in I), respectively. Similarly, we note $N(o)$ and $P(o)$ the sets of negative and positive pools in an observation o. Using these notations, the distance between an observation o and an interpretation I is:

$$\delta(o,\text{I})=\left|N(o)\cap P(\text{I})\right|+\left|P(o)\cap N(\text{I})\right|.$$

The first term represents the distance induced by the pools observed negative but interpreted positive (i.e. interpreted as false negatives), and the second term accounts for the distance induced by the pools interpreted as false positives.

3.2 Conflicting variables, conflicting pools

Given an observation o, the partitioning of Ω into $\mathcal{N}$ and $\mathcal{P}$ obviously splits the set of pools into two categories: negative and positive, noted $N(o)$ and $P(o)$ as stated above. However, a more thorough analysis reveals that each category can further be partitioned into two sub-classes, which we call 'conflicting' and 'non-conflicting'. These classes, and the underlying notion of 'conflicting variables', can be described as follows.

- A positive pool is non-conflicting in o, if it contains at least one variable that appears only in positive pools. Otherwise, it is conflicting.
- A variable is conflicting in o if it appears in at least one conflicting positive pool.
- A negative pool is non-conflicting in o, if it does not contain any conflicting variables. Otherwise, it is conflicting.

Let us now define these classes formally.

DEFINITION 6. *Conflicting pools and variables. Let o be an observation. The classes of non-conflicting and conflicting positive pools* (*respectively negative pools*) *in o, noted $P_{\bar{c}}(o)$ and $P_c(o)$* [*respectively $N_{\bar{c}}(o)$ and $N_c(o)$*], *are:*

$$P_{\bar{c}}(o)\triangleq\left\{p\in P(o)\mid\exists v\in p,\ \pi(v)\subset P(o)\right\},$$
$$P_c(o)\triangleq\left\{p\in P(o)\mid\forall v\in p,\ \pi(v)\cap N(o)\neq\emptyset\right\},$$
$$N_{\bar{c}}(o)\triangleq\left\{p\in N(o)\mid\forall v\in p,\ \pi(v)\cap P(o)\subset P_{\bar{c}}(o)\right\},$$
$$N_c(o)\triangleq\left\{p\in N(o)\mid\exists v\in p,\ \pi(v)\cap P_c(o)\neq\emptyset\right\}.$$

The set of conflicting variables $\mathcal{V}_c$ is:

$$V_c\triangleq\left\{v\in V\mid\exists p\in P_c(o),\ v\in p\right\}.$$

These definitions can be extended naturally to interpretations. For any interpretation I, clearly $P_{\bar{c}}(\text{I})$, $P_c(\text{I})$, $N_{\bar{c}}(\text{I})$ and $N_c(\text{I})$ form a partition of the set of pools. It is easy to see that:

$$\begin{aligned}P_c(\text{I})=\emptyset\ &\Leftrightarrow\ P_{\bar{c}}(\text{I})=P(\text{I})\ \Leftrightarrow\ N_c(\text{I})=\emptyset\\&\Leftrightarrow\ N_{\bar{c}}(\text{I})=N(\text{I})\Leftrightarrow\ \text{I}\in\mathcal{C}.\end{aligned}$$

In particular, an interpretation is consistent if and only if it has no conflicting positive pools. It follows that given an observation o, $\mathcal{S}(o)$ is the set of interpretations I such that $P_c(\text{I})=\emptyset$ and $\delta(o,\text{I})=\Delta(o)$.

The following Lemma shows that any pool that is non-conflicting in o keeps its (canonical) value in any nearest consistent interpretation.

LEMMA 1. *Let o be an observation and $\text{I}\in\mathcal{S}(o)$, then:*

$$P_{\bar{c}}(o)\subset P(\text{I}),\text{ and }N_{\bar{c}}(o)\subset N(\text{I}).$$

This leads us to propose the following strategy: given an observation o, start off with its canonical interpretation, and empty $P_c(o)$ as 'cheaply' as possible, by changing the values of conflicting pools. Changing conflicting positive pools to negative can obviously contribute to this goal, but changing conflicting negative pools to positive may also do so. Indeed, it can result in conflicting positive pools becoming non-conflicting, as explicited below.

3.3 Score of a set of negative conflicting pools

DEFINITION 7. *Consistent interpretation associated to a subset of $N_c(o)$. Let o be an observation. For every $\text{N}\subset N_c(o)$, the interpretation associated to N, noted $\iota(\text{N})$, is defined by:*

$$P(\iota(\text{N}))=\{p\in P(o)\cup\text{N}\mid\exists v\in p,\pi(v)\subset P(o)\cup\text{N}\}.$$

One way to understand this is the following.
Consider the interpretation I' defined by $P(\text{I}')=P(o)\cup\text{N}$. By definition, $P_{\bar{c}}(\text{I}')=\left\{p\in P(\text{I}')\mid\exists v\in p,\ \ \pi(v)\subset P(\text{I}')\right\}$. This is precisely $P(\iota(\text{N}))$. Therefore, $\iota(\text{N})$ is the interpretation where, starting from I', all conflicting positive pools [i.e. $P_c(\text{I}')$] are changed to negative. It is easy to verify that $P(\iota(\text{N}))=P_{\bar{c}}(\iota(\text{N}))$, hence $\iota(\text{N})\in\mathcal{C}$: the interpretation associated to any $\text{N}\subset N_c(o)$ is consistent. The following Lemma shows that, somewhat reciprocally, every nearest consistent interpretation is associated to some subset of $N_c(o)$.

LEMMA 2. *Let o be an observation.*
$\forall\ \text{I}\in\mathcal{S}(o)$, $N(o)\cap P(\text{I})\subset N_c(o)$ and $\text{I}=\iota(N(o)\cap P(\text{I}))$.

DEFINITION 8. *Score of a subset of $N_c(o)$.*
Let o be an observation. $\forall\ \text{N}\subset N_c(o)$, we define the score of N as:

$$\sigma(\text{N})\triangleq\left|P_c(o)\cap P(\iota(\text{N}))\right|-|\text{N}|.$$

For example, $\iota(\emptyset)$ is such that $P(\iota(\emptyset))=P_{\bar{c}}(o)$: it is the interpretation where, starting from o's canonical interpretation, every positive conflicting pool is changed to negative. Clearly, $\iota(\emptyset)$ is consistent, and $\sigma(\emptyset)=0$.

Generally, the score of N can be interpreted as follows. Changing all the pools in N to positive has a cost: it induces a distance $|\text{N}|$. However, this leads to some conflicting positive pools becoming non-conflicting, namely $P_c(o)\cap P(\iota(\text{N}))$: these pools remain positive in $\iota(\text{N})$, whereas they must be changed to negative in $\iota(\emptyset)$. $\sigma(\text{N})$ is therefore the difference between what is gained and what is paid, when considering $\iota(\text{N})$ as a possible solution [i.e. as an element of $\mathcal{S}(o)$] and $\iota(\emptyset)$ as the reference point.

The distance between o and $\iota(\emptyset)$ is $\delta(o,\iota(\emptyset))=|P_c(o)|$. In the case of an arbitrary N, the relationship between $\sigma(\text{N})$ and $\delta(o,\iota(\text{N}))$ is explicited in Lemma 3, leading immediately to Lemma 4.

Lemma 3. *Let o be an observation.*

$$\forall \mathrm{N} \subset N_c(o),\ \sigma(\mathrm{N}) + \delta(o, \iota(\mathrm{N})) \le |P_c(o)|,$$

with equality if and only if $\mathrm{N} \subset P(\iota(\mathrm{N}))$.

Lemma 4. *Let o be an observation.*

$$\forall \mathrm{N} \subset N_c(o),\ \sigma(\mathrm{N}) \le |P_c(o)| - \Delta(o),$$

with equality if and only if $\iota(\mathrm{N}) \in \mathcal{S}(o)$ and $\mathrm{N} \subset P(\iota(\mathrm{N}))$.

In conjunction with Lemma 2, we obtain:

$$\mathcal{S}(o) = \{\, \iota(\mathrm{N}) \mid \mathrm{N} \subset N_c(o)\ ,\ \sigma(\mathrm{N}) \text{ maximal} \,\}.$$

$\mathcal{S}(o)$ can therefore be identified by finding the highest-scoring subsets of $N_c(o)$. In addition, Lemma 4 shows that this search can be restricted to the subsets $\mathrm{N} \subset N_c(o)$ that satisfy $\mathrm{N} \subset P(\iota(\mathrm{N}))$. In fact, this condition simply states that all pools of N must be positive in $\iota(\mathrm{N})$. Given Definition 7 and the ensuing remark, this seems natural. Indeed, if $p_1 \in \mathrm{N}$ is such that $p_1 \in N(\iota(\mathrm{N}))$, then one can consider $\mathrm{N}' = \mathrm{N} \setminus \{p_1\}$, and obviously $\iota(\mathrm{N}') = \iota(\mathrm{N})$: p_1 becomes positive in I' defined by $P(\mathrm{I}') = P(o) \cup \mathrm{N}$, but this is useless since p_1 gets changed back to negative in $\iota(\mathrm{N})$.

In the following section, we see that this condition can be expressed simply in terms of closures of variables.

3.4 Closure of a set of conflicting variables

Definition 9. *Closure of a set of conflicting variables.*
Let V_1 be a set of conflicting variables. The closure of V_1 in a subset of $\mathcal{D}$ is the set of pools of the subset that contain some variable of V_1. In particular, the closures of V_1 in $P_c(o)$ and in $N_c(o)$, denoted $\pi_p(V_1)$ and $\pi_n(V_1)$, are:

$$\pi_p(V_1) \triangleq \bigcup_{v \in V_1} \pi(v) \cap P_c(o) \qquad \text{and}$$

$$\pi_n(V_1) \triangleq \bigcup_{v \in V_1} \pi(v) \cap N_c(o).$$

Lemma 5. *Let o be an observation, and $\mathrm{N} \subset N_c(o)$ such that $\sigma(\mathrm{N})$ is maximal. Then:*

$$\exists V_1 \subset V_c,\ \mathrm{N} = \pi_n(V_1).$$

Combined with the previous section's conclusion, we finally obtain the following reformulation of the decoding problem.

Theorem 1. *Let* o *be an observation.*

$$\mathcal{S}(o) = \{\, \iota(\pi_n(V_1)) \mid V_1 \subset V_c\ ,\ \sigma(\pi_n(V_1)) \text{ maximal} \,\}.$$

The decoding problem can therefore be solved by finding the set(s) of conflicting variables whose closure(s) in $N_c(o)$ has/have the highest score. This presentation of the problem has the advantage that for any set of conflicting variables V_1, the score of its closure can be easily calculated: $\sigma(\pi_n(V_1)) = |\pi_p(V_1)| - |\pi_n(V_1)|$, provided that $V_1 = \{v \in V_c \mid \pi_n(\{v\}) \subset \pi_n(V_1)\}$. This condition simply states that V_1 should include as many conflicting variables as possible, as long as they do not change the set $\pi_n(V_1)$. If the condition is not satisfied, $|\pi_p(V_1)| - |\pi_n(V_1)|$ underestimates $\sigma(\pi_n(V_1))$; the correct score is calculated by considering V_2, the largest superset of V_1 such that $\pi_n(V_2) = \pi_n(V_1)$.

4 ALGORITHM

In this section, we describe the *interpool* algorithm for solving the decoding problem described in Definition 5. This algorithm relies on Theorem 1: it identifies all maximal-scoring closures (in $N_c(o)$) of sets of conflicting variables.

As in Section 3, for the sake of clarity results are presented in the simplest instance of the decoding problem. Their analogs remain valid in the general framework and have been implemented, but they require unwieldy notations. For example, $|P_c(o)|$ becomes $\Sigma_{\omega \in \mathcal{P}}\ \delta_\omega \cdot |\{p \in P_c(o) \mid o(p) = \omega\}|$.

4.1 The search space

The goal is to find all sets of conflicting variables $V \subset \mathcal{V}_c$ such that $\sigma(\pi_n(V))$ is maximal. The algorithm follows a branch-and-bound strategy. The general idea is to build V by adding conflicting variables one at a time. The search space is a tree where each node represents a set of conflicting variables: the root is the empty set, and the children of a node V_1 represent the sets $V_1 \cup \{v\}$, where v is not in V_1 and not in any elder brother of V_1 or of any of its ancestors.

We use the term unit closure to represent the closure $\pi_n(\{v\})$ of a single conflicting variable v. Before beginning the search, the values of $\pi_n(\{v\}), \pi_p(\{v\}), |\pi_n(\{v\})|$ and $\sigma(\pi_n(\{v\}))$ are pre-calculated for every $v \in \mathcal{V}_c$. These quadruplets, which represent unit closures and their associated information, are referred to as unit structures.

The search tree is explored following a depth-first strategy, and the variables are initially sorted by decreasing score (of the corresponding unit structure). This leads to the quick identification of some high-scoring—though not necessarily optimal—closures, which prove valuable for pruning as discussed in Section 4.2. Internal nodes are considered as virtual observations, derived from the real observation but where some choices have already been made: namely, that the selected conflicting variables are actually positive, and that any conflicting negative pool that contains them is a false negative. At each step leafwards, the unit structures are updated by 'substracting' the selected variable's unit structure from them. This process is performed by the function substractFromUnits, and detailed in Section 4.3. As a result, internal nodes along with their 'local' unit structures can be dealt with—and pruned where appropriate—in the same way as the root node, with a small adjustment to take into account the selected closures. For example, the score of each child node can be trivially calculated (or underestimated as discussed following Theorem 1, but this is easily corrected) by simply adding the corresponding unit structure's local score to the current score. In addition, the search in each sub-tree is performed by decreasing order of local score instead of the less relevant initial score. Finally, although the search space remains huge, the following propositions can be used to prune large sub-trees, resulting in an efficient exact algorithm.

4.2 Pruning criterion

Propositions 1 and 2 can be combined to provide an upper bound for the score of any remaining descendant of the current node. When this upper bound is smaller than the current best score, the corresponding sub-trees can be ignored and pruned. These propositions are effective even early in the search, because high-scoring closures are found fast due to the search strategy, as discussed in Section 4.1.

Notation. *In this sub-section, we use the following notations to represent instrinsic characteristics of the pooling design $\mathcal{D}$:*

- *k is the pooling design's maximal redundancy (maximum number of pools that can contain any single variable).*
- *Γ is the maximal co-occurence of variables in $\mathcal{D}$, i.e. the maximum number of pools that can contain any two variables.*

Proposition 1. *Let $q \in I\!N^*$, and $\{\mathrm{N}_i\}_{i=1,\ldots,q}$ be the q highest-scoring unit closures. Consider a set of unit closures $\{\mathrm{N}'_i\}_{i=1,\ldots,q}$ corresponding to conflicting variables $\{v'_i\}_{i=1,\ldots,q}$ satisfying the following condition: $\forall v \in V_c \setminus \{v'_i\}_{i=1,\ldots,q},\ \pi_n\{v\} \not\subset \bigcup_{i=1}^{q} \mathrm{N}'_i$. Then:*

$$\sigma\left(\bigcup_{i=1}^{q} \mathrm{N}'_i\right) \leq \sum_{i=1}^{q} \Big(\sigma(\mathrm{N}_i) + \min\big((i-1)\Gamma, k-1\big)\Big).$$

Proposition 2. *Let $q \in I\!N^*$, and $\{\mathrm{N}_i\}_{i=1,\ldots,q}$ be the q smallest unit closures sorted by increasing size. Then for any set of $q' \geq q$ unit closures $\{\mathrm{N}'_i\}_{i=1,\ldots,q'}$:*

$$\sigma\left(\bigcup_{i=1}^{q'} \mathrm{N}'_i\right) \leq |P_c(o)| - \sum_{i=1}^{q} \Big(|\mathrm{N}_i| - \min\big((q-i)\Gamma, |\mathrm{N}_i|\big)\Big).$$

Note that the upper bound in proposition 1 tends to increase with q, while that in Proposition 2 can only decrease. They can be used conjointly to obtain a powerful pruning criterion. Indeed, noting α the current best score (adjusted by substracting the current node's score), one can apply the following algorithm:

(1) Find the smallest q such that

$$\sum_{i=1}^{q} \Big(\sigma(\mathrm{N}_i) + \min\big((i-1)\Gamma, k-1\big)\Big) \geq \alpha,$$

where $\{\mathrm{N}_i\}_{i=1,\ldots,q}$ are the q highest-scoring unit closures.

(2) Consider now the q smallest unit closures $\{\mathrm{N}'_i\}_{i=1,\ldots,q}$ sorted by increasing size. If

$$\sum_{i=1}^{q} \Big(|\mathrm{N}'_i| - \min\big((q-i)\Gamma, |\mathrm{N}'_i|\big)\Big) > |P_c(o)| - \alpha,$$

every remaining descendant of the current node can be pruned.

This algorithm's correctness, which results from the two propositions, is proved in the supplementary materials. From the point of view of the search tree, the algorithm anticipates several moves in advance: the first step bounds the score of any node less than q generations leafwards, while the second step does so for nodes at least q generations away.

4.3 The *interpool* algorithm

A rough description of the core of *interpool* is the recursive algorithm presented in Figure 1.

Before the initial call to findBest, all conflicting pools and variables are identified. For each conflicting variable, its unit structure is determined: this is used to initialize *units*. The

```
findBest($units$, $best$, $previous$) {
  sort $units$ by decreasing score;
  sort $units$ by increasing size;
  for each $u \in units$ (in order of decreasing score) {
    if (pruning algorithm succeeds)
      break;
    $units$ = $units \setminus u$, $current$ = $previous \cup u$;
    $newUnits$ = substractFromUnits($units$, $u$, $current$);
    updateBest($best$, $current$);
    findBest($newUnits$, $best$, $current$); }}
```

Fig. 1. The core of the *interpool* algorithm, described in pseudo-C. *units* is the list of unit structures, *best* stores the highest-scoring structures found up to now, and *previous* holds the structure examined in this node's father.

algorithm is intially called with the empty structure (defined by $\pi_n(\emptyset) = \emptyset$, $\pi_p(\emptyset) = \emptyset$ and $\sigma(\emptyset) = 0$) serving as both *previous* and *best*. Indeed, it constitutes a possible solution corresponding to the interpretation where every positive conflicting pool becomes negative.

Upon being called, the unit structures *units* are sorted by decreasing score as well as by increasing size. In addition to guiding the search, these orderings are used within the pruning algorithm as described in Section 4.2. Each unit structure is then selected successively, by order of decreasing score. If the algorithm derived from Propositions 1 and 2 authorizes pruning, the function returns immediately: this skips the currently selected unit structure and its descendance, but also every unit structure that remained to be selected in *units*. Otherwise, the current unit structure u is removed from *units* and merged with the running structure *previous* to obtain *current* (i.e. the closures are unioned and the scores and sizes are added). It is also 'substracted' from each unit structure of *units* to obtain *newUnits*. More precisely, substractFromUnits substracts each component (π_n and π_p) of u from the remaining unit structures, and updates their scores and sizes accordingly. In addition, it discards unit structures if:

(1) the resulting unit structure no longer has any conflicting positive pools, i.e. the variable isn't conflicting anymore;

(2) the resulting unit structure no longer has any conflicting negative pools, in which case its former conflicting positive pools are added to π_p of *current* beforehand.

The latter step results in the score of *current* being exact, rather than under-estimated as discussed following Theorem 1. This score is then compared to the current best score by updateBest, and *best* is updated if required. Finally, the recursive call occurs. When the initial call returns, *best* holds the highest-scoring structures.

Determining the average complexity of interpool is hard, as is often the case for combinatorial optimization algorithms. Indeed, the search tree is dynamically built and pruned as the search progresses, and this process cannot be easily modeled. In addition, as discussed in Section 6, the performance of interpool does not depend so much on the total number of

Table 1. Simulation results: STD(940;13;13), false positive rate 10%

Positives	FNR%	TPs missed	Retests	Simulations	Time
2	10	0	2.26	10 000	1 min
2	20	0	2.26	10 000	1 min
2	30	1.2%	2.27	10 000	4 min
3	10	0	3.57	10 000	4 min
3	20	0.4%	3.58	10 000	33 min
3	30	3.4%	3.60	10 000	2 h
4	10	0	5.06	10 000	32 min
4	20	1.0%	5.11	10 000	10 h 39 min
4	30	6.2%	5.26	7500	2 days 11 h
5	10	0.1%	6.71	10 000	4 h
5	20	1.7%	6.94	1000	12 h 47 min
5	30	12.9%	7.88	300	3 days 10 h

Positives: number of simulated positives. FNR: false negative rate of the individual assays. TPs missed: fraction of true positives that are not recovered. Retests: number of variables decoded as positive or ambiguous, that must be retested (this includes the recovered true positives). TPs missed and Retests are the upper bounds of the 95% confidence interval for the mean. Simulations: number of simulations performed in the run. Time: total real time taken by the run, on a 2.13 GHz Intel Core 2 Duo GNU/linux system.

variables and pools. Instead, it depends mainly on how 'comfortable' the pooling design is, with respect to the number of positives and errors. This is difficult to quantify, hence the choice of the input size to use in a complexity analysis is unclear. However, Table 1, with run-times for a variety of realistic instances, provides a useful indication of how interpool performs.

5 IMPLEMENTATION

The *interpool* algorithm has been implemented in C, and is freely available under the terms of the GNU General Public Licence (GPL). It builds cleanly and has been tested on several hardware and software combinations, including various GNU/linux i386 and x86_64 setups, SunOS 5.9 and Cygwin. It can be downloaded from our web page (http://www-timc.imag.fr/Nicolas.Thierry-Mieg/).

The implementation allows up to four discrete outcomes, two of which are interpreted as positive and two as negative: this corresponds to the instance $\Omega = \{\text{NONE, FAINT, WEAK, STRONG}\}$, $\mathcal{N} = \{\text{NONE, FAINT}\}$, and $\mathcal{P} = \{\text{WEAK, STRONG}\}$ presented in Section 2. The distance δ is set by the user. Additional discrete outcomes could be allowed with a little work, although we have never felt the need in our applications: changing the costs in δ provides sufficient flexibility.

The code has been heavily optimised for speed. For example, the data structures have been chosen so that all bottleneck calculations are implemented as bitwise operations. This provides high efficiency, and additionally allows to take full advantage of modern 64-bit architectures. Indeed, on an Intel Core 2 Duo CPU the *interpool* implementation is almost twice as fast when compiled in 64-bit mode as it is in 32-bit mode (produced by compiling with the -m32 switch to gcc). Another useful trick is that when entering findBest, the unit structures *units* are not completely sorted. Instead, we only identify and sort a small number of top-ranking structures, and later extend the sorted lists if necessary. In practice, extending is relatively rarely needed, due to pruning fairly early in the loop.

In addition to the *interpool* algorithm proper, the package includes several independant implementations of simpler decoding algorithms that are used for cross-validation, as well as a tool for performing simulations. One simulation consists in four steps: (1) randomly pick t variables: the simulated positives; (2) generate the corresponding observation, add noise; (3) interpret the observation, using *interpool*, to obtain a decoded value for each variable; (4) compare the variables' decoded values with their real ones (from step 1) to identify any mis-taggings. The simulator's usefulness is illustrated in Section 6.

6 RESULTS AND DISCUSSION

The *interpool* algorithm is essential both before and after the actual experimental work. First, because it is very efficient, it allows to perform large numbers of simulations for choosing the pooling design most appropriate for a given experimental setup. For example, with the Shifted Transversal Design (STD; Thierry-Mieg, 2006b), the goal is to select values for parameters q and k such that the design achieves the desired compromise between number of pools and decoding power—given that this is always a trade-off, since identifying more positives and/or correcting more errors requires a greater number of pools. Second, once the pools have been selected, built and screened, it allows to decode the experimental observations.

Although this latter task is paramount, the former is computationally much more intensive. Indeed, given the sizes of the spaces that are being explored, large numbers of simulations must be performed in order to obtain satisfactory coverage. We illustrate this process in the context of a pilot project we are involved with in collaboration with Marc Vidal and co-workers (Dana Farber Cancer Institute, Boston), where smart-pooling with STD is applied to yeast-two-hybrid interactome mapping.

In this project, we wish to screen some baits against 940 prey proteins. For each bait, we generally expect at most 3 positive preys. Rough estimates of the expected error-rates are in the 5–10% range for false positives and up to 25% for false negatives.

We first performed a series of 'easy' runs of simulations, with wide ranges of values for the STD parameters q and k, and the two least demanding experimental conditions (2 or 3 positives, 5% false positives and 10% false negatives). Each run comprised 10 000 simulations, and we ruled out any design that showed signs of weakness, i.e. failed to systematically identify every simulated positive, or gave rise to more than 0.1 ambiguous variables on average (out of the 940 variables). We also terminated runs that took longer than 2 min to complete the 10 000 simulations, and excluded the corresponding designs. This is justified by our observation that although *interpool* is very fast when the conditions (number of positives and errors) can be comfortably dealt with by the pooling design, its performance degrades when the design can barely or imperfectly cope with the conditions. Therefore, slow runs in the easy conditions are an indication that the design will not be powerful enough in more difficult ones. This was confirmed by performing small numbers of simulations with increased error rates. This first stage led to the pre-selection of five STD designs.

In a second step, each candidate design was examined in more detail: the number of positives varied between 1 and 5, while the error-rates were set at up to 15% false positives and 30% false negatives. In this way, the behaviour of each design in the case of highly connected baits and/or unexpectedly high error-rates could be studied. The main measure of performance is the fraction of true positives that are not recovered: it represents the false negative rate of the smart-pooling method. Another interesting measure is the number of preys that are decoded as positive or ambiguous, i.e. the retest burden, assuming the strategy is to retest all the candidates individually.

For example, Table 1 shows results obtained with the design STD(940;13;13), using a false positive rate of 10%. This design performs well in most settings, although it begins to miss a non-negligible fraction of positives when there are three or more positives and a 30% false negative rate. Yet even in the hardest setting, the smart-pooling false negative rate is only 12.9%: much less than that of the individual pairwise screen (30%), which requires 940 tests (instead of 169 for this STD design). Notice that in all settings, the retest burden is at most a few more than the number of true positives: most candidates will be genuine positives. In this phase, conditions leading to slow runs were still studied, although the number of simulations was decreased when necessary. This results in the measured means for the two performance criteria being less precise; but since we report the upper bounds of the 95% confidence intervals, which are correspondingly larger, the reported numbers remain valid over-estimates of the true means and can be compared to the other conditions.

As a side note, this data confirms that the guarantee requirement is indeed overkill for practical purposes. For example, identification of three positives with STD(940;13;13) is only guaranteed when there are at most three errors of each type, but the error rates used here when $t = 3$ correspond to 13 false positives and 3, 7 and 10 false negatives (for 10, 20 and 30% respectively). Clearly, the first two conditions are dealt with very well despite 10 excess false positives and up to four extra false negatives. This shows how important it is to perform simulations in order to choose a design.

After comparing similar datasets obtained with the other candidate designs, this one was selected as the best compromise between robustness and size—it has 169 pools, hence fits into two 96-well plates. Based on these results, a pilot experiment was performed in collaboration with Marc Vidal and co-workers, where 100 baits were screened against 940 preys smart-pooled according to STD(940;13;13). This experiment will be reported elsewhere.

7 CONCLUSION

In this article, we provide a clear formalization of the decoding problem and present a deterministic algorithm to solve it. This algorithm is exact, i.e. it always finds the optimal solution, yet proves very efficient. An open-source implementation is freely available. It can be used to perform simulations in order to choose appropriate sets of pools for a given application before carrying out assays. Subsequently, it allows to interpret the experimental results, correcting for false positives and false negatives and identifying the positives.

ACKNOWLEDGEMENTS

We thank Jean Thierry-Mieg, Laurent Trilling, Jean-Louis Roch and Michael Blum for stimulating discussions. This work was funded by a BQR'2003 grant from the Institut National Polytechnique de Grenoble (INPG) to NT.

Conflict of Interest: none declared.

REFERENCES

Barillot,E. *et al.* (1991) Theoretical analysis of library screening using a N-dimensional pooling strategy. *Nucleic Acids Res.*, **19**, 6241–6247.

Bruno,W.J. *et al.* (1995) Efficient pooling designs for library screening. *Genomics*, **26**, 21–30.

Jin,F. *et al.* (2006) A pooling-deconvolution strategy for biological network elucidation. *Nat. Methods*, **3**, 183–189.

Jin,F. *et al.* (2007) A yeast two-hybrid smart-pool-array system for protein-interaction mapping. *Nat. Methods*, **4**, 405–407.

Knill,E. *et al.* (1996) Interpretation of pooling experiments using the Markov chain Monte Carlo method. *J. Comput. Biol.*, **3**, 395–406.

Thierry-Mieg,N. (2006a) Pooling in systems biology becomes smart. *Nat. Methods*, **3**, 161–162.

Thierry-Mieg,N. (2006b) A new pooling strategy for high-throughput screening: the Shifted Transversal Design. *BMC Bioinformatics*, 7:28.

Vermeirssen,V. *et al.* (2007) Matrix and Steiner-triple-system smart pooling assays for high-performance transcription regulatory network mapping. *Nat. Methods*, **4**, 659–664.