Mendelian error detection in complex pedigrees using weighted constraint satisfaction techniques

Marti Sanchez, Simon de Givry, Thomas Schiex

Mendelian error detection in complex pedigrees using weighted constraint satisfaction techniques

Marti Sanchez
msanchez@toulouse.inra.fr
Simon de Givry
degivry@toulouse.inra.fr
Thomas Schiex
tschiex@toulouse.inra.fr
INRA - UBIA
Toulouse, France

October 12, 2007

Abstract

With the arrival of high throughput genotyping techniques, the detection of likely genotyping errors is becoming an increasingly important problem. In this paper we are interested in errors that violate Mendelian laws. The problem of deciding if a Mendelian error exists in a pedigree is NP-complete [1]. Existing tools dedicated to this problem may offer different level of services: detect simple inconsistencies using local reasoning, prove inconsistency, detect the source of error, propose an optimal correction for the error. All assume that there is at most one error. In this paper we show that the problem of error detection, of determining the minimum number of errors needed to explain the data (with a possible error detection) and error correction can all be modeled using soft constraint networks. Therefore, these problems provide attractive benchmarks for weighted constraint network (WCN) solvers. Because of their sheer size, these problems drove us into the development of a new WCN solver `toulbar2` which solves very large pedigree problems with thousands of animals, including many loops and several errors.

Biological background and motivations

Chromosomes carry the genetic information of an individual. A position that carries some specific information on a chromosome is called a *locus* (which typically identifies the position of a gene). The specific information contained at a locus is called the *allele* carried at the locus. Except for the sex chromosomes, diploid individuals carry chromosomes in pair and therefore a single locus carries a pair of alleles. Each allele originates from one of the parents of the individual considered. This pair of alleles at this locus define the *genotype* of the individual at this locus. It is said to be *homozygous* if both alleles are the same. Genotypes are not always completely observable and the indirect observation of a genotype (its expression) is termed the *phenotype*. A phenotype can be considered as a set of possible genotypes for the individual. These genotypes are said to be compatible with the phenotype.

A pedigree is defined by a set of related individuals together with associated phenotypes for some locus. Every individual is either a founder (no parents in the pedigree) or not. In this latter case, the parents of the individual are identified inside the pedigree. Because a pedigree is built from a combination of complex experimental processes it may involve

experimental and human errors. The errors can be classified as parental errors or typing errors. A parental error means that the very structure of the pedigree is incorrect: one parent of one individual is not actually the parent indicated in the pedigree. We assume that parental information is correct. A phenotype error means simply that the phenotype in the pedigree is incompatible with the true (unknown) genotype. Phenotype errors are called Mendelian errors when they make a pedigree inconsistent with Mendelian law of inheritance which states that the pair of alleles of every individual is composed of one paternal and one maternal allele. The fact that at least one Mendelian error exists can be effectively proved by showing that every possible combination of compatible genotypes for all the individual violates this law at least once. Since the number of these combinations grows exponentially with the number of individuals, only tiny pedigrees can be checked by enumeration. The problem of checking pedigree consistency is actually shown to be NP-complete in [1]. Other errors are called non Mendelian errors[1].

The detection and correction of errors is crucial before the data can be exploited for the construction of the genetic map of a new organism (genetic mapping), the localization of genes (loci) related to diseases, or other quantitative traits. Because of its NP-completeness, most existing tools only offer a limited polynomial time checking procedure. The different tools we are aware of that try to tackle this problem are either incomplete, restricted by strong assumptions (such as unique error), or incapable of dealing with large problems. This is specifically important for animal pedigrees which may contain thousands of animals, many errors and many loops (a marriage between two individuals which have a parental relation) and thus with a large tree-width.

In this paper, we introduce soft constraint network models for the problem of checking consistency, the problem of determining the minimum number of errors needed to explain the data and the problem of proposing an optimal correction to an error. These problems offer attractive benchmarks for (weighted) constraint satisfaction. In Section 3, we describe the algorithms used to solve these problems. We report extensive results using the weighted constraint network solver `toulbar2` and other solvers in Section 4.

1 Modeling the problems

A constraint network $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ [7] is defined by a set of n variables $\mathcal{X} = \{x_1, \ldots, x_n\}$, a set of matching domains $\mathcal{D} = \{D_1, \ldots, D_n\}$ with maximum size equal to d and a set of e constraints $\mathcal{C}$. Every variable $x_i \in \mathcal{X}$ takes its value in the associated domain D_i. A constraint $c_S \in \mathcal{C}$ is defined as a relation on a set of variables $S \subset \mathcal{X}$ which defines authorized combination of values for variables in S. Alternatively, a constraint may be seen as the characteristic function of this set of authorized tuples. It maps authorized tuples to the boolean *true* (or `1`) and other tuples to *false* (or `0`). The set S is called the scope of the constraint and $|S|$ is the arity of the constraint. For arities of one, two or three, the constraint is respectively said to be unary, binary, or ternary. A constraint may be defined by a predicate that decides if a combination is authorized or not, or simply as a set of combination of values which are authorized. For example, if $D_1 = \{1,2,3\}$ and $D_2 = \{2,3\}$, two possible equivalent definitions for a constraint on x_1 and x_2 would be $x_1 + 1 > x_2$ or $\{(2,2),(3,2),(3,3)\}$.

The central problem of constraint networks is to give a value to each variable in such a way that no constraint is violated (only authorized combinations are used). Such a variable assignment is called a solution of the network. The problem of finding such a solution is called the CONSTRAINT SATISFACTION PROBLEM (CSP). Proving the existence of a solution for an arbitrary network is an NP-complete problem.

Often, tuples are not just completely authorized or forbidden but may be associated with a cost. Several ways to model such problems have been proposed among which the

[1] Non Mendelian errors may be identified only in a probabilistic way using several locus simultaneously and a probabilistic model of recombination and errors [9].

most famous are the semi-ring and the valued constraint networks frameworks [3]. In this paper we consider weighted constraint networks (WCN) where constraints map tuples to non negative integers. For every constraint $c_S \in \mathcal{C}$, and every variable assignment A, $c_S(A[S]) \in N$ represents the cost of the constraint for the given assignment where $A[S]$ is the projection of A on the constraint scope S. The aim is then to find an assignment A of all variables such that the sum of all tuple costs $\sum_{c_S \in \mathcal{C}} c_S(A[S])$ is minimum. This is called the Weighted Constraint Satisfaction Problem (WCSP), obviously NP-hard. Several algorithms for tackling this problem, all based on the maintenance of local consistency properties [26] have been recently proposed [13, 5, 17, 4, 15, 12, 6]. They are presented in Section 3.

1.1 Genotyped pedigree and constraint networks

Consider a pedigree defined by a set I of individuals. For a given individual $i \in I$, we note $pa(i)$ the set of parents of i. Either $pa(i) \neq \varnothing$ (non founder) or $pa(i) = \varnothing$ (founder). At the locus considered, the set of possible alleles is $\{1,...,m\}$. Therefore, each individual carries a genotype defined as an unordered pair of alleles (one allele from each parent, both alleles can be identical). The set of all possible genotypes is denoted by G and has cardinality $\frac{m(m+1)}{2}$. For a given genotype $g \in G$, the two corresponding alleles are denoted by g^l and g^r and the genotype is also denoted as $g^l|g^r$. By convention, $g^l \leq g^r$ in order to break symmetries between equivalent genotypes (e.g. $1|2$ and $2|1$). The experimental data is made of phenotypes. For each individual in the set of observed individuals $I' \subset I$, its observed phenotype restricts the set of possible genotypes to those which are compatible with the observed phenotype. This set is denoted by $G(i)$ (very often $G(i)$ is a singleton, observation is complete).

A corresponding constraint network encoding this information uses one variable per individual *i.e.* $\mathcal{X} = I$. The domain of every variable $i \in \mathcal{X}$ is simply defined as the set of all possible genotypes G. If an individual i has an observed phenotype, a unary constraint that involves the variable i and authorizes just the genotypes in $G(i)$ is added to the network. Finally, to encode Mendelian law, and for every non founder individual $i \in \mathcal{X}$, a single ternary constraint involving i and the two parents of i, $pa(i) = \{j,k\}$ is added. This constraint only authorizes triples (g_i, g_j, g_k) of genotypes that verify Mendelian inheritance *i.e.* such that one allele of g_i appears in g_j and the other appears in g_k. Equivalently:

$$(g_i^l \in g_j \wedge g_i^r \in g_k) \vee (g_i^l \in g_k \wedge g_i^r \in g_j)$$

For a pedigree with n individuals among which there are f founders, with m possible alleles, we obtain a final CN with n variables, a maximum domain size of $\frac{m(m+1)}{2}$ and $n-f$ ternary constraints. Existing problems may have more than 10,000 individuals with several alleles (the typical number of alleles varies from two to a dozen).

A solution to such a constraint network defines a genotype for each individual that respects Mendelian law (ternary constraints) and experimental data (domains) and the consistency of this constraint network is therefore obviously equivalent to the consistency of the original pedigree. As such, pedigree consistency checking offers a direct problem for constraint networks. In practice, solving this problem is not enough (i) if the problem is consistent, one should simplify the problem for further probabilistic processing by removing all values (genotypes) which do not participate in any solution, this specific problem is known as "genotype elimination", (ii) if the problem is inconsistent, errors have to be located and corrected.

Example 1 *A small example is given in Fig. 1. There are $n = 12$ individuals and $m = 3$ distinct alleles. Each box corresponds to a male individual, and each oval to a female. The arcs describe parental relations. For instance, individuals 1 and 2 have three children 3,4, and 5. The founders are individuals 1,2,6, and 7 ($f = 4$). The possible genotypes*

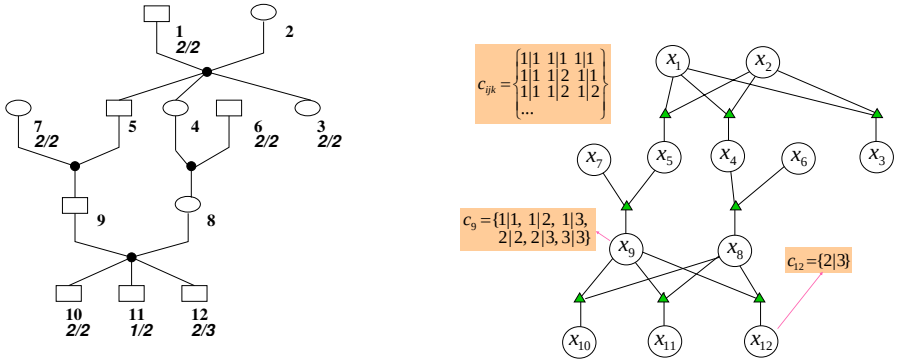

Figure 1: Left: A pedigree taken from [23]. Right: its corresponding CSP.

are $G = \{1|1, 1|2, 1|3, 2|2, 2|3, 3|3\}$. *There are 7 individuals (1,3,6,7,10,11, and 12) with an observed phenotype (a single genotype). The corresponding CSP has 12 variables, with maximum domain size of 6, and 8 ternary constraints. This problem can be proven inconsistent, that is, there is no combination of pair of alleles that we can assign to the individuals of unobserved phenotype that respects all mendelian laws.*

2 Error detection

From an inconsistent pedigree, the first problem is to identify errors. In our knowledge, this problem is not perfectly addressed by any existing program which either identifies a not necessarily minimum cardinality set of individuals that only restore a form of local consistency or makes the assumption that a single error occurs. The first approach may detect too many errors that moreover may not suffice to remove global inconsistency and the second approach is limited to small data-sets (even high quality automated genotyping may generate several errors on large data-sets).

A typing error for an individual $i \in \mathcal{X}$ means that the domain of variable i has been wrongly reduced: the true (unknown) value of i has been removed. To model the possibility of such errors, genotypes in G which are incompatible with the observed phenotype $G(i)$ should not be completely forbidden. Instead, a soft constraint forbids them with a cost of 1 (since using such a value represents one typing error). We thereby obtain a weighted constraint network with the same variables as before, the same hard ternary constraints for Mendelian laws and soft unary constraints for modeling genotyping information. Domains are all equal to G.

If we consider an assignment of all variables as indicating the real genotype of all individuals, it is clear that the sum of all the costs induced by all unary constraints on this assignment precisely gives the number of errors made during typing. Finding an assignment with a minimum number of errors follows the traditional parsimony principle or and is consistent with a low probability of independent errors (quite reasonable here) or Occam's razor principle. This defines the Parsimony problem. One solution of the corresponding WCSP with a minimum cost therefore defines a possible diagnostic (variables assigned with a value forbidden by $G(i)$ represent errors). These networks have the same size as the previous networks with the difference that all variables now have the maximum domain size $|G|$. The fundamental difference lies in the shift from satisfaction to optimization. The fact that only *unary* soft constraints arise here is not a simplification in itself w.r.t. the general WCSP since every n-ary weighted constraint network can be simply translated in an equivalent dual network with only unary soft constraints and hard binary constraints [14].

Example 2 *Fig. 2 shows the WCSP associated to the previous pedigree example of Fig. 1. The problem still has 12 variables, with domain size of 6. It has 8 hard ternary constraints*

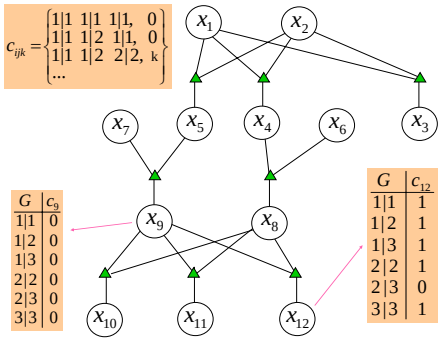

Figure 2: The corresponding WCSP of Fig. 1.

and 7 soft non-null unary constraints, one for every observed individual. The minimum number of typing errors is one. There are 66 optimal solutions of cost one, which can occur in any of the typed individuals except individual 10. One optimal solution is $\{(1,2|2)$, $(2,1|2)$, $(3,2|2)$, $(4,1|2)$, $(5,2|2)$, $(6,2|2)$, $(7,2|2)$, $(8,1|2)$, $(9,2|2)$, $(10,2|2)$, $(11,1|2)$, $(12,2|2)\}$ *such that the* erroneous *typing* $2|3$ *of individual 12 has been changed to* $2|2$.

2.1 Error correction

When errors are detected, one would like to optimally correct them. The simple parsimony criterion is usually not sufficient to distinguish alternative values. More information needs to be taken into account. Since errors and Mendelian inheritance are typical stochastic processes, a probabilistic model is attractive. A Bayesian network is a network of variables related by conditional probability tables (CPT) forming a directed acyclic graph. It allows to concisely describe a probability distribution on stochastic variables. To model errors, a usual approach is to distinguish the observation O and the truth T. A CPT $P(O|T)$ relates the two variables and models the probability of error.

Following this, we consider the following model for error correction: we first have a set of n variables T_i each representing the true (unknown) genotype of individual i. The domain is G. For every observed phenotype, an extra observed variable O_i is introduced. It is related to the corresponding true genotype by the CPT $P_i^{error}(O_i|T_i)$. In our case, we assume that there is a constant α probability of error: the probability of observing the true genotype is $1-\alpha$ and the remaining probability mass is equally distributed among remaining values.

For the individual i and its parents $pa(i)$, a CPT $P_i^{mendel}(T_i|pa(i))$ representing Mendelian inheritance connects T_i and its corresponding parent variables. Given that each parent has two alleles, there are four possible combinations for the children and all combinations are equiprobable (probability $\frac{1}{4}$). However, since all parental alleles are not always different, the genotype of a child has probability $\frac{1}{4}$ times the number of combinations of the parental alleles that produce this genotype.

Finally, prior probabilities $P^{founder}(i)$ for each genotype must be given for every founder i. These probabilities are obtained by directly estimating the frequency of every allele in the genotyped population. For a genotype, its probability is obtained by multiplying each allele frequency by the number of its possible combinations (a genotype $a|b$ can be obtained in two ways by selecting an a from a father and a b from the mother or the converse). If both alleles are equal there is only one way to achieve the genotype. The probability of a complete assignment $P(O,T)$ (all true and observed values) is then defined as the product of the three collections of probabilities (P^{error}, P^{mendel} and $P^{founder}$). Note that equivalently, its log-probability is equal to the sum of the logarithms of all these probabilities.

The evidence given by the observed phenotypes $G(i)$ is taken into account by reducing the domains of the O_i variables to $G(i)$. One should then look for an assignment of the variables $T_i, i \in I'$ which has a maximum a posteriori probability (MAP). The MAP probability of such an assignment is defined as the sum of the probabilities of all complete assignments extending it and maximizing it defines an $\mathsf{NP}^{\mathsf{PP}}$-complete problem [24], for which there exists no exact method that can tackle large problems. The PEDCHECK solver [22, 23] tries to solve this problem using the extra assumption of a unique already identified error. This is not applicable in large data-sets. Another very strong assumption (known as the Viterbi assumption) considers that the distribution is entirely concentrated in its maximum and reduces MAP to the so-called Maximum Probability Explanation problem (MPE) which simply aims at finding a complete assignment of maximum probability. Using logarithms as mentioned above, this problem directly reduces to a WCSP problem where each CPT is transformed in an additive cost function. This allows to solve MPE using WCN dedicated tools. introduced in `toulbar2`.

3 Soft constraint processing

In this section, we describe the algorithms we used to solve general weighted constraint networks. For simplicity reasons and adequacy to the pedigree problem, in the next section we restrict our presentation to the case of ternary or smaller arity constraints. In particular, we show how to extend existing soft binary constraint processing techniques to the case of ternary constraints. The technical difficulty resides in the simultaneous enforcement of two important soft local consistency properties (AC and DAC which are defined below) in polynomial time. In the following, we show how to enforce DAC in one direction of a ternary constraint without breaking AC in the two other directions. This allows to close an open question from [4] (Section 4) "whether a form of directional k-consistency can be established in polynomial time for $k > 2$". The answer is yes for ternary constraints and we believe it can be generalized to any bounded constraint arity.

3.1 Notations

Following [17], the valuation structure of Weighted CSP (WCSP) is, $S(k) = ([0..k], \oplus, \geq)$ where $k > 0$ is a natural number; $\oplus$ is defined as $a \oplus b = \min\{k, a+b\}$; $\geq$ is the standard order among naturals. Observe that in $S(k)$, we have minimum cost $\bot = 0$ and maximum cost $\top = k$. It is useful to define the *subtraction* $\ominus$ of costs. Let $a, b \in [0..k]$ be two costs such that $a \geq b$,

$$a \ominus b = \begin{cases} a-b & : \quad a \neq k \\ k & : \quad a = k \end{cases}$$

A ternary *weighted constraint satisfaction problem* (WCSP) is a tuple $P = (S(k), \mathcal{X}, \mathcal{D}, \mathcal{C})$. $S(k)$ is the valuation structure. $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a set of variables to which we will often refer to just by their index. Each variable $x_i \in \mathcal{X}$ has a finite domain $D_i \in \mathcal{D}$ of values that can be assigned to it. (i,a) denotes the assignment of value $a \in D_i$ to variable x_i. $\mathcal{C}$ is a set of unary, binary, and ternary weighted constraints (namely, cost functions) over the valuation structure $S(k)$. A unary weighted constraint C_i is a cost function $C_i(a \in D_i) \rightarrow [0..k]$. A binary constraint C_{ij} is a cost function $C_{ij}(a \in D_i, b \in D_j) \rightarrow [0..k]$. A ternary constraint C_{ijl} is a cost function $C_{ijl}(a \in D_i, b \in D_j, c \in D_l) \rightarrow [0..k]$. We assume the existence of a unary constraint C_i for every variable, binary constraints (C_{ij},C_{il}, and C_{jl}) for every ternary constraint C_{ijl}, and a *zero*-arity constraint (i.e. a constant), noted $C_\varnothing$ (if no such constraint is defined, we can always define *dummy* ones: $\forall a \in D_i, C_i(a) = \bot$, $\forall a \in D_i, b \in D_j, C_{ij}(a,b) = \bot$, $\forall a \in D_i, b \in D_j, c \in D_l, C_{ijl}(a,b,c) = \bot$, $C_\varnothing = \bot$).

When a constraint C assigns cost $\top$, it means that C forbids the corresponding assignment, otherwise it is permitted by C with the corresponding cost. The *cost* of an assignment

$X = (x_1, \ldots, x_n)$, noted $\mathcal{V}(X)$, is the sum over all the problem cost functions (here x_i refers to the value $a \in D_i$ assigned to variable i),

$$\mathcal{V}(X) = \sum_{C_{ij} \in \mathcal{C}} C_{ij}(x_i, x_j) \oplus \sum_{C_{ijl} \in \mathcal{C}} C_{ijl}(x_i, x_j, x_l) \oplus \sum_{C_i \in \mathcal{C}} C_i(x_i) \oplus C_\varnothing$$

An assignment X is *consistent* if $\mathcal{V}(X) < \top$. The usual task of interest is to *find a consistent assignment with minimum cost*, which is NP-hard. Observe that WCSP with $k = 1$ reduces to classical CSP.

3.2 Soft local consistency extended to ternary constraints

In this section, we present several local consistency properties, previously defined for binary constraints [26, 13, 5, 15] and extended to the case of ternary constraints in order to deal with our pedigree problem.

Two WCSPs defined over the same variables are said to be *equivalent* if they define the same cost distribution on complete assignments. Local consistency properties are widely used to transform problems into equivalent simpler ones. When enforcing a local consistency property at every node of the search, implicit costs can be deduced and so the search space can be hopefully reduced and variable values pruned earlier (a non trivial lower bound is given by $C_\varnothing$). The deduced unary costs and the current variable domains can also guide the search, as they can be used by the variable and value ordering heuristics (explained in Section 3.3).

The simplest form of local consistency we used is node consistency (NC): $\forall x_i \in \mathcal{X}, (\exists a \in D_i / C_i(a) = \bot) \wedge (\forall a \in D_i / C_\varnothing \oplus C_i(a) < \top)$ [13]. NC is enforced by procedures ProjectUnary and PruneVar (Algorithm1) which perform the projection of a unary constraint towards $C_\varnothing$ and prune unfeasible values, respectively. Note that the propagation queues Q, S and R are explained in algorithm 3 description.

To extend the notion of soft (directional) arc consistency to ternary cost functions, we extend the classic notion of support in the WCSP framework. Given a binary constraint C_{ij}, $b \in D_j$ is a *simple support* for $a \in D_i$ if $C_{ij}(a,b) = \bot$. Similarly, for directional arc consistency, $b \in D_j$ is a *full support* for $a \in D_i$ if $C_{ij}(a,b) \oplus C_j(b) = \bot$.

For a ternary cost function C_{ijk}, we say that the pair of values $(b \in D_j, c \in D_k)$ is a simple support for $a \in D_i$ if $C_{ijk}(a,b,c) = \bot$. Similarly, we say that the pair of values $(b \in D_j, c \in D_k)$ is a full support for $a \in D_i$ if $C_{ijk}(a,b,c) \oplus C_{ij}(a,b) \oplus C_{ik}(a,c) \oplus C_{jk}(b,c) \oplus C_j(b) \oplus C_k(c) = \bot$.

A WCSP is arc consistent (AC) if every variable is NC and every value of its domain has a simple support in every constraint [26, 13]. Given a static variable ordering, a WCSP is directional arc consistent (DAC) if every value of every variable x_i has a full support in every constraint C_{ij} such that $j > i$ and in every constraint C_{ijk} such that $j > i \wedge k > i$ [5]. A WCSP is full directional arc consistent (FDAC) if it is both AC and DAC [5, 17].

Procedures Project2To1 (Algorithm 1) and FindSupports2 (Algorithm 2) enforce simple supports in D_j for every value in D_i. The same process can be done for ternary constraints, by using Project3To1 (Algorithm 1) and FindSupports3 (Algorithm 2).

In order to enforce full supports in D_j for D_i values (procedure FindFullSupports2 in Algorithm 2) we need to extend unary costs from $C_j(\cdot)$ towards $C_{ij}(\cdot,\cdot)$ (procedure Extend1To2 in Algorithm 1). Then, binary costs are projected towards $C_i(\cdot)$ by Project2To1.

The previous reasoning has to be carefully adapted to ternary constraints. The idea is to extend unary and binary costs involved in the scope of ternary constraint C_{ijk} in such a way that a maximum projection is achievable on variable i without losing simple supports for variables j and k. Procedure FindFullSupports3(i, j, k) (Algorithm 2) enforces full supports for each value of variable i by computing the maximum possible projection ($P_i[a]$) for every $a \in D_i$ w.r.t. available unary, binary, and ternary costs. Accordingly to

each $P_i[a], a \in D_i$, it also computes the maximum cost extensions for unary and binary constraints such that each extension has to be done in order to be able to project $P_i[a]$ without introducing negative costs in the constraint network. Moreover, each extension is minimum in the sense that a weaker extension would result in negative costs. Assuming the constraint network is already AC, this condition guarantees that for each unary cost extension ($E_j[b]$ or $E_k[c]$), there is at least a value $a \in D_i$ such that the resulting binary cost ($C_{ij}(a,b) \oplus E_j[b] \ominus E_{ij}[a,b]$ or $C_{ik}(a,c) \oplus E_k[c] \ominus E_{ik}[a,c]$) is equal to zero at the end of the procedure, preserving simple supports at the binary level. The proof is similar to the proof of Theorem 2 in [17]. The same is true for each binary cost extension ($E_{ij}[a,b]$, $E_{ik}[a,c]$ or $E_{jk}[b,c]$): there exists a value in the third variable ($c \in D_k$, $b \in D_j$ or $a \in D_i$ respectively) such that $C_{ijk}(a,b,c) \oplus E_{ij}[a,b] \oplus E_{ik}[a,c] \oplus E_{jk}[b,c] = 0$, thus preserving simple supports for variables j and k at the ternary level. The order of cost extension operations (which are performed by procedures Extend1To2 and Extend2To3 in Algorithm 1) is built according to the DAC variable ordering in order to move costs from the highest variables to the lowest first. The resulting ordered cost flow is summarized in Figure 3. An example of enforcing full supports is given below.

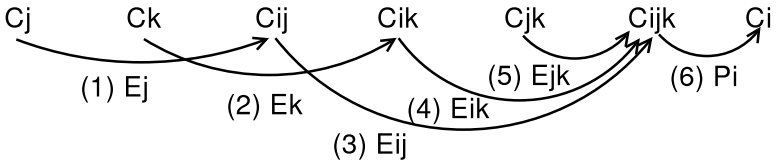

Figure 3: Order of cost operations for FindFullSupports3(i,j,k) with $i < j < k$: cost extensions ($E_.$) from unary to binary, binary to ternary constraint C_{ijk}, and cost projection (P_i) from ternary to unary constraint C_i.

Example 3 *Consider the problem depicted in Figure4.a. It has four variables i,j,k,l with two values (a,b) in their domain. Unary costs are depicted within small circles. Binary (red continuous line) or ternary (green broken line for C_{ijl} and blue dotted line for C_{ikl}) costs are represented by edges or hyper-edges connecting the corresponding values. The label of each edge (hyper-edge) is the corresponding cost. If two or three values are not connected, the cost between them is 0. In this problem the optimal cost is 1 and it is reached by the assignment (a,a,a,a). The next figures (4.b,.c,.d) show the effect of FindFullSupports3 which enforces full supports for each value in the domain of variable i w.r.t. ternary constraints C_{ijl} and C_{ikl}. FindFullSupports3(i,j,l) computes the following projection/extension costs:*

$$P_i[a] = E_l[b] = E_{ij}[a,a] = E_{il}[a,b] = 1$$

all other projection/extension costs being null. The result is shown in Figure 4.d.

FindFullSupports3(i,k,l) computes the following costs:

$$P_i[b] = E_{il}[b,b] = E_{ik}[b,a] = 1$$

all other costs being null. The result is shown in Figure 4.f.

In the case of two ternary constraints sharing two variables as shown in Figure 4, a unary cost ($C_l(b)$ in the example) can be used by both ternary constraints due to the extension of unary costs to binary constraints when enforcing full supports for ternary constraints. This is not the case for complete 3-consistency as defined in [4] where unary costs are directly extended to ternary constraints. Moreover k-consistency [4] does not move unary costs from one variable to another as it is done by directional arc consistencies.

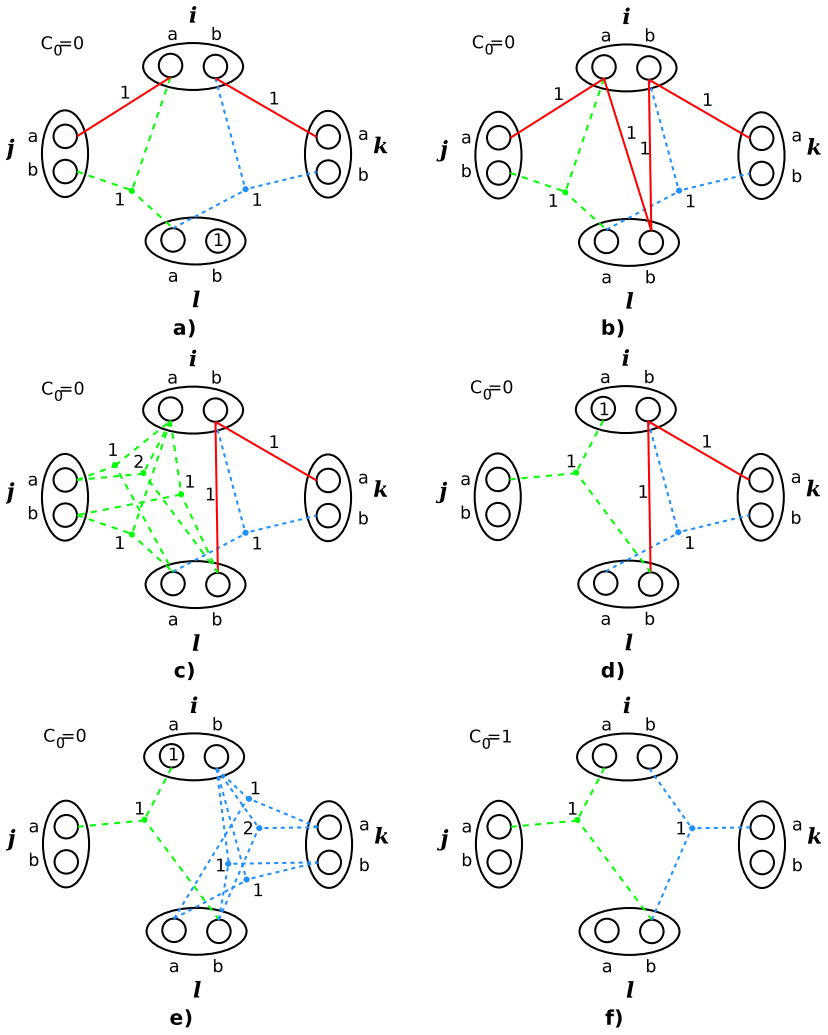

Figure 4: Six equivalent WCSP problems ($\top = 4$): (a) original problem with two ternary costs $C_{ijl}(a,b,a) = C_{ikl}(b,b,a) = 1$, two binary costs $C_{ij}(a,a) = C_{ik}(b,a) = 1$, and a unary cost $C_l(b) = 1$; it is AC but not DAC ($i < j < k$) (b) extending a cost of one from unary constraint C_l to binary constraint C_{il} ($E_l[b] = 1$) (c) extending a cost of one from binary constraints C_{ij} and C_{il} to ternary constraint C_{ijl} ($E_{ij}[a,a] = E_{il}[a,b] = 1$) (d) projecting a cost of one from C_{ijl} to $C_i(a)$ ($P_i[a] = 1$) (e) extending a cost of one from binary constraints C_{il} and C_{ik} to ternary constraint C_{ikl} ($E_{il}[b,b] = E_{ik}[b,a] = 1$) (f) finally, made FDAC after projecting a cost of one from C_{ikl} to $C_i(b)$ ($P_i[b] = 1$) then projecting a cost of 1 to $C_\emptyset$.

The strongest form of local consistency we use is existential directional arc consistency (EDAC) [15]. A WCSP is existential arc consistent (EAC) if every variable x_i has at least one value $a \in D_i$ such that $C_i(a) = \bot$ and a has a full support in every constraint. A WCSP is EDAC if it is both FDAC and EAC.

EAC enforcement is done by finding at least one *fully supported* value per variable i.e. which is fully supported in all directions[2]. If there is no such value for a given variable,

[2]Distinctly from EAC for binary constraints, EAC for ternary constraints does not guarantee that there exists an assignment of all the neighborhood variables of a given variable i such that the constraints involving i are all satisfied due to possible ternary constraints C_{ijl} and C_{ikl} sharing a second variable in common.

Algorithm 1: Algorithms to propagate costs.

Procedure PruneVar(i)
  foreach $a \in D_i$ **do**
    if $(C_\varnothing \oplus C_i(a) \geq \top)$ **then**
      $D_i := D_i - \{a\}$;
1     $Q := Q \cup \{i\}$;

Procedure ProjectUnary(i)
  $\alpha := \min_{a \in D_i}\{C_i(a)\}$;
  $C_\varnothing := C_\varnothing \oplus \alpha$;
  foreach $a \in D_i$ **do** $C_i(a) := C_i(a) \ominus \alpha$;

Procedure Project2To1(i, a, j, α)
  if $(\alpha > \bot \wedge C_i(a) = \bot)$ **then**
2     $R := R \cup \{i\}$;
3     $S := S \cup \{i\}$;
  $C_i(a) := C_i(a) \oplus \alpha$;
  foreach $b \in D_j$ **do** $C_{ij}(a,b) := C_{ij}(a,b) \ominus \alpha$;

Procedure Extend1To2(i, a, j, α)
  foreach $b \in D_j$ **do** $C_{ij}(a,b) := C_{ij}(a,b) \oplus \alpha$;
  $C_i(a) := C_i(a) \ominus \alpha$;

Procedure Project3To1(i, a, j, k, α)
  if $(\alpha > \bot \wedge C_i(a) = \bot)$ **then**
4     $R := R \cup \{i\}$;
5     $S := S \cup \{i\}$;
  $C_i(a) := C_i(a) \oplus \alpha$;
  foreach $b \in D_j, c \in D_k$ **do** $C_{ijk}(a,b,c) := C_{ijk}(a,b,c) \ominus \alpha$;

Procedure Extend2To3(i, a, j, b, k, α)
  foreach $c \in D_k$ **do** $C_{ijk}(a,b,c) := C_{ijk}(a,b,c) \oplus \alpha$;
  $C_{ij}(a,b) := C_{ij}(a,b) \ominus \alpha$;

then projecting all the constraints towards this variable will increase the lower bound, resulting in at least one *fully supported* value. Notice that EDAC, even for binary constraints only, is incomparable with k-consistency [4] because EDAC can consider all the neighborhood (which can contain more than k variables) of a variable in order to increase $C_\emptyset$. By definition, for each triplet of variables involved in a ternary constraint, EDAC is locally optimal in the sense that it finds the best lower bound. However, EDAC is weaker than OSAC [6], a recently defined optimal arc consistency property based on a linear programming formulation with rational costs.

Observe that in the CSP case AC, FDAC, and EDAC instantiate to classical arc consistency generalized to binary and ternary constraints where the scope of any binary constraint is not included in the scope of a ternary constraint (if it is not the case, then the binary constraint can be merged in the ternary constraint).

In our pedigree problem, EDAC is able to deduce a non trivial lower bound ($C_\emptyset = 1$) for any inconsistent nuclear family satisfying one of the following condition, as expressed in [22]: the alleles of a child and a parent are incompatible; the child is compatible with each parent separately but not when both parents are consider simultaneously; there are more than four alleles in a sibship; there are more than three alleles in a sibship with a homozygous child; there are more than two alleles in a sibship with two different homozygotes among the sibs.

Algorithm 2: Algorithms to enforce supports.

Procedure FindSupports2(i, j)
  foreach $a \in D_i$ **do**
    $\alpha := \min_{b \in D_j} \{C_{ij}(a,b)\}$;
    Project2To1(i, a, j, α) ;
  ProjectUnary(i) ;

Procedure FindSupports3(i, j, k)
  foreach $a \in D_i$ **do**
    $\alpha := \min_{b \in D_j, c \in D_k} \{C_{ijk}(a,b,c)\}$;
    Project3To1(i, a, j, k, α) ;
  ProjectUnary(i) ;

Procedure FindFullSupports2(i, j)
  foreach $a \in D_i$ **do**
    $P_i[a] := \min_{b \in D_j} \{C_{ij}(a,b) \oplus C_j(b)\}$;
  foreach $b \in D_j$ **do**
    $E_j[b] := \max_{a \in D_i} \{P_i[a] \ominus C_{ij}(a,b)\}$;
  foreach $b \in D_j$ **do** Extend1To2($j, b, i, E_j[b]$) ;
  foreach $a \in D_i$ **do** Project2To1($i, a, j, P_i[a]$) ;
  ProjectUnary(i) ;

Procedure FindFullSupports3(i, j, k)
  foreach $a \in D_i$ **do**
    $P_i[a] := \min_{b \in D_j, c \in D_k} \{C_{ijk}(a,b,c) \oplus C_{ij}(a,b) \oplus C_{ik}(a,c) \oplus C_{jk}(b,c) \oplus C_j(b) \oplus C_k(c)\}$;
  foreach $b \in D_j$ **do**
    $E_j[b] := \max_{a \in D_i, c \in D_k} \{P_i[a] \ominus C_{ijk}(a,b,c) \ominus C_{ij}(a,b) \ominus C_{ik}(a,c) \ominus C_{jk}(b,c) \ominus C_k(c)\}$;
  foreach $c \in D_k$ **do**
    $E_k[c] := \max_{a \in D_i, b \in D_j} \{P_i[a] \ominus C_{ijk}(a,b,c) \ominus C_{ij}(a,b) \ominus C_{ik}(a,c) \ominus C_{jk}(b,c) \ominus E_j[b]\}$;
  foreach $a \in D_i, b \in D_j$ **do**
    $E_{ij}[a,b] := \max_{c \in D_k} \{P_i[a] \ominus C_{ijk}(a,b,c) \ominus C_{ik}(a,c) \ominus C_{jk}(b,c) \ominus E_k[c]\}$;
  foreach $a \in D_i, c \in D_k$ **do**
    $E_{ik}[a,c] := \max_{b \in D_j} \{P_i[a] \ominus C_{ijk}(a,b,c) \ominus C_{jk}(b,c) \ominus E_{ij}[a,b]\}$;
  foreach $b \in D_j, c \in D_k$ **do**
    $E_{jk}[b,c] := \max_{a \in D_i} \{P_i[a] \ominus C_{ijk}(a,b,c) \ominus E_{ij}[a,b] \ominus E_{ik}[a,c]\}\}$;
  foreach $b \in D_j$ **do** Extend1To2($j, b, i, E_j[b]$) ;
  foreach $c \in D_k$ **do** Extend1To2($k, c, i, E_k[c]$) ;
  foreach $a \in D_i, b \in D_j$ **do** Extend2To3($i, a, j, b, k, E_{ij}[a,b]$) ;
  foreach $a \in D_i, c \in D_k$ **do** Extend2To3($i, a, k, c, j, E_{ik}[a,c]$) ;
  foreach $b \in D_j, c \in D_k$ **do** Extend2To3($j, b, k, c, i, E_{jk}[b,c]$) ;
  foreach $a \in D_i$ **do** Project3To1($i, a, j, k, P_i[a]$) ;
  ProjectUnary(i) ;

Algorithm 3 description We present an algorithm for the enforcement of EDAC in the case of binary and ternary WCSPs, based on previous work in [17, 15]. EDAC (Algorithm 3) transforms an arbitrary problem into an *equivalent* one verifying the EDAC local property. It uses three propagation queues, Q, R and P. If $l \in Q$, it means that some value in D_l has been pruned (line 1), neighbors of l may have lost their simple support and must be revised. If $l \in R$, it means that some value in D_l has increased its unary cost from $\perp$ (lines 2 and 4), neighbors of l may have lost their full support and must be revised. If $i \in P$, it means that some value in D_j (j neighbor of i, including i itself at line 7) has increased its unary cost from $\perp$ (lines 3, 5 and 12), i may have lost the full support of its fully supported

Algorithm 3: Enforcing EDAC, initially, $Q = R = S = \mathcal{X}$.

Procedure EDAC

6 **while** $(Q \neq \varnothing \vee R \neq \varnothing \vee S \neq \varnothing)$ **do**

7 $P := S \cup \{j \mid i \in S, C_{ij} \in \mathcal{C}\} \cup \{j,k \mid i \in S, C_{ijk} \in \mathcal{C}\}$;

8 **while** $(P \neq \varnothing)$ **do**

$i := \mathsf{pop}(P)$;

9 $\alpha := \min_{a \in D_i}\{C_i(a) \bigoplus_{C_{ij} \in \mathcal{C} \text{ s.t. } C_{ijk} \notin \mathcal{C}} \min_{b \in D_j}\{C_{ij}(a,b) \oplus C_j(b)\}$
$\bigoplus_{C_{ijk} \in \mathcal{C}} \min_{b \in D_j, c \in D_k}\{C_{ijk}(a,b,c) \oplus C_{ij}(a,b) \oplus C_{ik}(a,c) \oplus C_{jk}(b,c) \oplus C_j(b) \oplus C_k(c)\}\}$

if $(\alpha > \bot)$ **then**

10 **foreach** $C_{ij} \in \mathcal{C}$ **do** FindFullSupports2(i,j) ;

11 **foreach** $C_{ijk} \in \mathcal{C}$ **do** FindFullSupports3(i,j,k) ;

12 $P := P \cup \{j \mid C_{ij} \in \mathcal{C}\} \cup \{j,k \mid C_{ijk} \in \mathcal{C}\}$;

$S := \varnothing$;

13 **while** $(R \neq \varnothing)$ **do**

$l := \mathsf{popMax}(R)$;

14 **foreach** $C_{il} \in \mathcal{C}$ s.t. $i < l$ **do** FindFullSupports2(i,l) ;

15 **foreach** $C_{ijl} \in \mathcal{C}$ s.t. $i < j \wedge i < l$ **do** FindFullSupports3(i,j,l) ;

16 **while** $(Q \neq \varnothing)$ **do**

$l := \mathsf{pop}(Q)$;

17 **foreach** $C_{il} \in \mathcal{C}$ **do** FindSupports2(i,l) ;

18 **foreach** $C_{ijl} \in \mathcal{C}$ **do**

FindSupports3(i,j,l) ;

FindSupports3(j,i,l) ;

19 **foreach** $i \in \mathcal{X}$ **do** PruneVar(i) ;

value and must be revised. Besides, there is an auxiliary queue S that is used to efficiently build P.

The algorithm is formed by a main loop, with four inner loops. The **while** loops at lines 8, 13 and 16 respectively enforce EAC, DAC, and AC; the line 19 enforces NC. Each time some costs are projected by the enforcement of a local property, another property may be broken. The variables for which the local property may be broken are stored in a queue for revision.

The EAC property is checked at line 9. If $\alpha = \bot$ then variable i has already a fully supported value in D_i, otherwise this value is obtained by enforcing a full support for every value in D_i in all the binary and ternary constraints the scope of which contains i. After these projections (when $\alpha > \bot$), the neighbors of i may have lost their full support due to new non-zero unary costs in i, they are stored in P (line 12) for EAC revision and i is also stored in R (lines 2 and 4) for DAC revision. DAC is enforced by finding a full support in one direction to ensure termination: each constraint projects its costs only on the smallest variable in its scope w.r.t. the DAC variable ordering. AC is enforced by finding a simple support in all directions. DAC and AC may insert variables in S (lines 3 and 5) for future EAC revision. Only NC can break AC and insert a variable in Q (line 1).

For simplicity reasons, the case of inconsistent problems where $C_\varnothing$ reaches $\top$ is not described.

Theorem 1 *The complexity of ternary EDAC is time $O(ed^3 max\{nd, \top\})$ and space $O(ed^2)$, where n is the number of variables, d is the maximum domain size, e is the number of constraints and $\top$ is the maximum cost.*

Proof 1 *Regarding space, we use the data-structure suggested in [13] to bring the space complexity of Algorithms 1 to $O(ed)$. For Extend2To3, this implies to explicitly memorize binary information, resulting in an overall space complexity of $O(ed^2)$.*

The time complexity of algorithms FindSupports2 and FindFullSupports2 is $O(d^2)$, while the complexity of FindSupports3 and FindFullSupports3 is $O(d^3)$.

*Let us now focus on the **while** loop complexities in EDAC. Before computing their complexities, we will show that each loop terminates and then we will compute the* cumulated *time complexity of their content by accumulating the time spent in each inner loop (lines 8, 13, 16 and 19) with the main loop (line 6).*

The loop at line 8 enforces EAC. It always terminates, although it may reinsert in P a variable several times at line 12, because a variable is reinserted in P only if EAC is violated which can occur at most $O(\top)$ times (each time EAC is violated, $C_\varnothing$ increases). The cumulated time complexity of lines 10, 11 and 12 is $O(ed^3\top)$ because every binary and ternary constraint is observed at most $O(\top)$ times.

*The main loop at line 6 is repeated if and only if a value has been removed by the **for** loop at line 19, breaking AC and possibly DAC and EAC at the next iteration (due to AC enforcement at line 16), or some unary costs have been moved by EAC enforcement at line 8, conducting to DAC revision at line 13 also moving unary costs and breaking EAC again. So, the main loop always terminates and iterates at most $O(\max\{nd, \top\})$ times. It follows that the cumulated time complexity of line 9 is $O(ed^3 \max\{nd, \top\})$.*

The loop at line 13 enforces DAC. Notice that FindFullSupports3 at line 15 cannot reinsert in R a previously popped variable, due to the way we select the highest variable w.r.t. DAC variable ordering in the priority queue R (popMax), resulting in a finite loop at line 13. The cumulated time complexity of lines 14 and 15 is $O(ed^2 \max\{nd, \top\})$ and $O(ed^3 \max\{nd, \top\})$ respectively.

The loop at line 16 which enforces AC terminates because each variable is examined only once. The cumulated time complexity of lines 17 and 18 is $O(ed^3)$ and $O(ed^4)$ respectively because each variable can be reinserted in Q by node consistency at most d times.

*The **for** at line 19 is time $O(nd)$ (which is less than $O(ed)$, as the graph is supposed to be connected). Compiling the different results, the overall complexity is $O(ed^3 max\{nd, \top\})$.*

3.3 Depth-first branch and bound

In order to find an optimal solution and prove its optimality, a classical depth-first branch and bound algorithm is applied. An initial upper bound ($\top$) is given by the number of genotyping data plus one for the parsimony pedigree problem. For MPE, we multiply for each individual the minimum probabilities different from zero of P^{error}, P^{mendel} and $P^{founder}$ (see Section 2.1) and take the negated logarithm (to get additive positive costs): $\top = -\log(\frac{1}{4^{n-nf}d^{nf}}\left(\frac{\alpha}{d-1}\right)^{no})$, with $d = \frac{m(m+1)}{2}$, the number of possible genotypes, n, the number of individuals, nf, the number of founders, no, the number of genotyping data and α, the probability of error. Each time a better solution is found, its cost becomes the new upper bound.

We maintain EDAC during search, producing a lower bound in $C_\varnothing$. The DAC variable ordering corresponds to the pedigree file order, which is usually a temporal order. If $C_\varnothing \geq \top$ then, the algorithm backtracks. We use dynamic variable and value ordering heuristics. By default, we choose the first unassigned variable having the minimum ratio of current domain size divided by future degree in the remaining constraint network. Ties are broken by choosing the variable with the highest unary cost in its domain, and if all equal, DAC variable ordering is taken. This heuristic is named dom/deg in the rest of the paper.

Following [18], we add a basic form of conflict back-jumping by always choosing the last variable in conflict (i.e. its assignment leads to an empty domain or $C_\varnothing \geq \top$). If a variable is in conflict with a set of variables previously assigned in the current assignment, then by repeatedly choosing the same conflicting variable instead of the variable preferred by dom/deg, will have the effect to backtrack up to the point where the last variable in the set has been assigned. This combined heuristic is named $conflict$ in the experiments.

The value ordering heuristic chooses first the fully supported value found by EAC. We use a binary branching scheme: the chosen variable is assigned to its fully supported value or this value is removed from its domain.

Finally, we apply a limited form of variable elimination during the search as proposed in [16]. If a variable is connected to at most two other unassigned variables, then we eliminate the variable by projecting its constraints to the two variables, possibly creating a new binary constraint between them. This elimination process is iterated until all variables are connected to three or more variables. Like in variable elimination algorithms, when a solution is found, the correct values of the eliminated variables can be retrieved by following the variable elimination ordering in the reverse order and choosing at each step a value which minimizes the cost of the projected constraints. In pedigrees without loops (i.e. marriage of two individuals having a common ancestor), this limited form of variable elimination, named *varelim* in the experiments, completely solves the problem.

Moreover, we apply in preprocessing, before the construction of the constraint network, a second kind of variable elimination based on the semantic of the pedigree problem. As a matter of fact, any individual which is not genotyped and has no genotyped descendant in the pedigree will have no effect on the genotyped individuals and can be discarded.

4 Experimental evaluation

The experimental section has two different aims. First, we want to compare the accuracy of error detection for the different models introduced: Parsimony, MPE, and MAP.

Then, we want to compare the efficiency of different solvers and evaluate the contribution of different technical points in this efficiency. We will therefore compare the efficiency of resolution of the different problems introduced using complete algorithms.

4.1 Solvers considered

Each problem can be solved by different solvers. The most complex MAP problem is a mixed optimization/integration problem that can be only solved by dedicated Bayes net solvers. We have chosen Samiam (see http://reasoning.cs.ucla.edu/samiam) because it is one of the most efficient and robust solver available according to the last BN solving competition. In the version 2.2.1 used (last stable version available on the web), MAP is solved by the *Shenoy-Shafer* inference algorithm whose computational cost is related to the tree-width of the instance tackled. Thus, it can only be applied to relatively small instances.

The MPE problem is a pure optimization problem which requires however to be able to deal with very large costs such as those produced by logarithms of probabilities (see Section 3.3). These problems can be addressed again by Samiam but also by `toulbar2` which has been extended to use very large integer costs. The problem can only be solved on small or mid-size instances.

Finally, the simplest Parsimony problem can be directly tackled by `toulbar2` but also by the previous version toolbar and by pseudo-boolean solvers (WCSP being easily cast into pseudo boolean SAT [11]). The version of toolbar used, called `toolbar/BTD`, integrates a specific tree-decomposition based branch and bound (version 2.2, see [12]) that should perform well on pedigree problems which have usually a tree-width much smaller than the number of variables. It also uses only binary EDAC and thus will show the interest of higher order consistencies. The pseudo-boolean solvers considered are MiniSat+ (version 1.0 based on `MiniSat` version 1.13, [8]) and Pueblo(version 1.5, [27]), among the most efficient solvers[3]. The Parsimony problem can be solved on very large instances.

Because the pedigree analysis problem is not a new problem, one must also acknowledge the existence of different solvers for the real problem. However, none of these tools

[3] See the Pseudo Boolean Evaluation 2006 results at http://www.cril.univ-artois.fr/PB06/.

will be considered in the analysis because they either make very strong assumptions incompatible with the pedigree size considered (PedCheck [22] assumes that there is only one error), may be incomplete solvers (CheckFam [25] can prove inconsistency but produces only local corrections on nuclear families that may not always restore consistency while GenCheck [2] provides corrections that optimizeneither parsimony nor likelihood) or have very limited efficiency compared to the solvers considered here (GMCheck [28] tackles the MPE problem but is totally dominated by SamIam).

4.2 Pedigree considered

Two types of pedigree have been used to perform the evaluation: random pedigree and real pedigree.

4.2.1 Random pedigree

The random pedigree have been generated using a pedigree generator designed by geneticists at INRA [30]. It is controlled by three main parameters: nf the number of founder individuals, *nmale* the number of males among nf and *ngen* the number of simulated generations. We used the generator essentially as a black box, but noticed that the *nmale* and *ngen* parameters together influence the connectivity of the individuals and thus the tree-width of the final generated pedigree instance[4]. The total number of individuals is equal to $ngen \times (nf - nmale) + nf$. Once the pedigree is generated, we randomly erase the genotypes of some individuals with a given probability and introduce errors in some individuals with a given probability. The original correct genotypes are recorded in order to be able to evaluate the accuracy of error correction. We used a genotyping error probability $\alpha = 5\%$ (see Section 2.1). Equifrequent allele frequencies $P^{founder}(i)$ are used for founders.

We have generated four different collections of instances of increasing size and tree-width using parameters as described in Table 1. Each collection contains 50 instances. $pedclass_{A,B}$, and $_C$ are generated with increasing number of individuals. $pedclass_D$ further increases the number of males in the number of founders. All instances and the generator are available for download at http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/SoftCSP (*Benchmarks* section) in *simplified* `LINKAGE` ('.pre') and `toolbar/toulbar2` ('.wcsp') formats.

	nf	*ngen*	*nmale*	individuals	treewidth	errors
$pedclass_A$	4...44	3	2	[10,170]	[2.73,4.82]	[0.97,3.3]
$pedclass_B$	8...68	5	4	[28,388]	[4.54,14.92]	[0.68,19.6]
$pedclass_C$	40...200	5	20	[140,1100]	[16.84,59.38]	[5.96,44.22]
$pedclass_D$	200	6	4...100	[1000,1376]	[19.64,128.16]	[51.54,81.94]

Table 1: Four different classes of sets of pedigree instances generated by different parameterizations of the simulator.

For random pedigree, all experiments have been performed on a 3 GHz Intel Xeon with 2 GB of RAM.

[4][21] showed that increasing the ratio between the number of non root nodes (C) and the number of root nodes (V) causes an approximately linear increase in the upper bound on the tree-width of randomly-generated Bayesian networks. The simulated pedigree instances are parent-regular (2 parents) and child irregular multipartite Bayesian networks corresponding to class B in [21]. But the parent selection process differs from [21]: for each generation, the parents are randomly selected in the previous generations and each time a male is selected, it is mated with approximately $\frac{nf-nmale}{nmale}$ females. In our case, we found that the *nmale* parameter was a better indicator than the C/V ratio for controlling tree-width and hardness for inference and optimization tasks.

4.2.2 Real pedigree instances

Table 2 shows the different characteristics of the real pedigree instances used in our experiments.

	ind	*vars*	*genotyped*	alleles	*nf*	*ngen*	treewidth ub
eye	36	36	28	6	11	4	2
cancer	49	48	37	8	18	5	2
parkinson	37	34	13	4	7	7	5
$berrichon_{1nc}$	129516	9947	2448	4	8821	17	262
$berrichon_{1}$	129516	10017	2483	4	8786	17	330
$berrichon_{2nc}$	27255	19337	10215	4	4719	19	-
$berrichon_{2}$	27255	19562	10215	4	2381	19	-
$langlade_1$	1355	1209	711	9	298	13	84
$langlade_2$	1355	1223	715	7	298	13	82
$langlade_3$	1355	1258	787	5	298	13	85
$langlade_4$	1355	1186	672	8	298	13	83
$moissac_1$	283	260	183	2	81	5	6
$moissac_2$	283	244	167	7	81	5	6
$moissac_3$	283	225	151	3	81	5	6
$moissac_4$	283	256	179	2	81	5	6
$moissac_5$	283	237	161	8	81	5	6
$moissac_6$	283	201	131	11	81	5	5

Table 2: Real pedigree instances. Columns: name of the instance, number of individuals, number of variables, number of genotyped individuals, number of alleles, number of founders, number of generations, and treewidth upper bound of the instance.

The first three instances are human genotyped pedigrees (genetic studies of eye, cancer, and Parkinson diseases) as reported in [22, 23].

The following two groups (*berrichon* and *langlade*) are pedigree instances coming from sheep animals provided by the CTIG (*Centre de Traitement de l'Information Génétique*) in France, which gathers and treats all the genetic information coming from animals in farms. The files correspond to all genotypings done for the enhancement program of genetic resistance to the *scrapie* disease [29]. Scrapie is a fatal, degenerative disease affecting the central nervous system of sheep and goats. The program is founded by the French Ministry of Agriculture. The instances are named by the specific sheep species (*Langlade* and *Berrichon du Cher*) which are actually spread between 29 sheep flocks in France where genetic selection programs are performed since the 60's.

The final group of *moissac* instances are goat pedigrees collected at the *Moissac Goat Experimental Station* (Lozère, France) with genotyping data of micro-satellite markers to analyze genetic variability in milking speed of dairy goats.

For real pedigrees, all experiments have been performed on a 3 GHz Intel Xeon 64-bit with 16 GB of RAM.

4.3 Evaluation of the error prediction accuracy

To compare the error prediction accuracy provided by the MAP, MPE, and Parsimony, we had to limit ourselves to relatively small instances ($pedclass_A$) that could be solved to optimality by Samiam. The MPE problem has been solved by using `toulbar2` and Samiam. Finally, Parsimony was solved by using `toulbar2` only.

A usual approach to evaluate the accuracy of prediction programs is to compute the so-called sensitivity and specificity of the predicted features. Two features were evaluated: the prediction of the individuals (denoted *ind*) containing an error in the pedigree and, more fine grained, the prediction of the correct genotype (denoted by *geno*).

For a given feature (*ind* or *geno*), the sensitivity of the prediction is the percentage of features that should be detected and which are actually correctly predicted. Similarly, specificity is percentage of predicted features which are correct. For a perfect prediction, both equal 100%.

Fig. 5 reports sensitivities and specificities for the three problems. The individual specificity is not shown because it is close to 100% for all methods: when an individual is detected as erroneous, it is actually erroneous.

MAP gives results which are very similar to MPE. The main advantage of MAP is its 10% higher genotype specificity, meaning that is more robust in predicting the corrections of genotypes, as expected. However, the CPU time used for the different problems are very different, as shown on the right of Fig. 5 using a log-scale: MAP is typically 3 orders of magnitude more costly (despite small instances with limited treewidth). The main conclusion of this analysis is that despite its superiority, MAP is too expensive and cannot be solved at this time on not too small instances. MPE gives very similar results while Parsimony is interesting for just restoring consistency. Note also that `toulbar2` outperforms Samiam on the MPE problem.

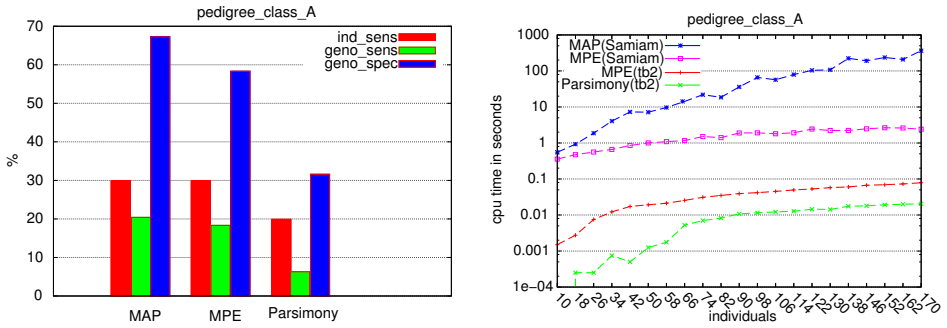

Figure 5: Left: Histograms of the sensitivities and specificities for MAP, MPE, and Parsimony. For each problem, we report the individual sensibility, the genotype sensibility and the genotype specificity. Right: CPU time is compared for the same problems with Samiam and `toulbar2` (tb2).

We further compared Parsimony and MPE on larger data sets using `toulbar2`. This is reported in Fig. 6 using the $pedclass_D$ dataset and a CPU-time limit of 300 seconds. MPE has nearly a 10% better individual sensitivity and a 15% better genotype sensitivity and specificity on the larger problems. $pedclass_D$ was chosen to because it contains instances of variable treewidth. Indeed, we observe that the CPU-time needed to solve the instances is highly sensitive to the treewidth for both MPE and Parsimony. For tree-widths above 50, `toulbar2` encountered some hard MPE instances it could not solve in the time limit. Note that the increase in treewidth is obtained by increasing the number of males in the founders in the generator[5]. However, the generator behavior tends to simultaneously lower the overall number of total generated individuals which explains why individuals eventually decrease along the x-axis.

4.4 Efficiency and features evaluation

Since our aim is to solve very large real size instances, we conclude the evaluation by comparing time efficiency of different solvers on the simplest Parsimony problem. Indeed, on the largest real instances defined by the sheep pedigree, MPE remained unsolvable

[5]Notice that the treewidth is anti-monotone in the number of individuals. This is because the increase in the treewidth is achieved (paramterizing the simulator) by increasing the number of males in a population. Increasing the number of males has the side effect of decreasing the number of individuals.

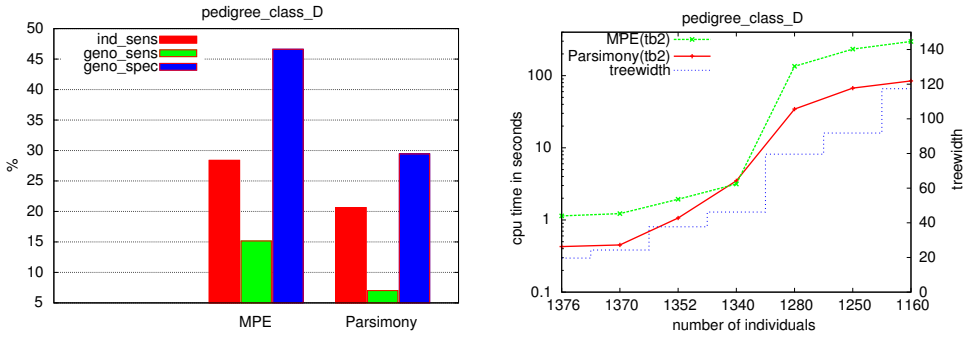

Figure 6: Left: Histograms compare the sensitivities and specificities of MPE and Parsimony. Right: `toulbar2` CPU-time for both problems.

in less than 10 hours. Despite its lower accuracy, Parsimony still provides the essential service of consistency restoration and this with minimum loss of data, a criterion that may look very attractive in practice to biologists.

4.4.1 Random pedigree

Fig. 7 shows a CPU time comparison for the selected solvers: `toulbar2`, `toolbar/BTD`, and the pseudo-boolean solvers `MiniSAT+` and `Pueblo`. As the problem size increases, `toulbar2` has the best performance.

Pseudo-boolean and SAT solvers have the extra ability of learning. In the Parsimony problems, all Mendelian constraints are hard and this may be exploited directly by clause learning. However, this does not apparently compensates for the probably much weaker lower bound produced by pseudo-boolean/SAT propagation.

The fact that `toulbar2` outperforms `toolbar/BTD`, which explicitly exploits tree-decompositions may be explained by the fact that `toolbar/BTD` only exploits binary EDAC (waiting until all but 2 variables are assigned before propagating), thus showing the interest of generalized FDAC and EDAC.

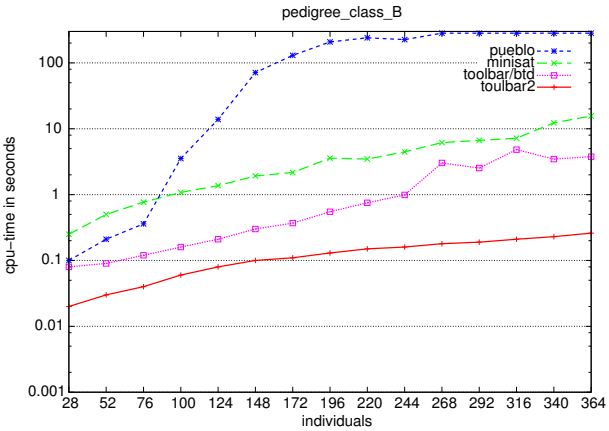

Figure 7: Solving time comparison of methods `toulbar2`, `toolbar/BTD`, `MiniSat+`, and `Pueblo`.

4.4.2 Results on real pedigree instances

In Table 3 we present the results of solving real pedigree instances with `toulbar2`. All pedigrees can be solved in reasonable time, despite the very large size of the underlying problems and the important number of errors detected in some sheep pedigrees.

	`toulbar2`		
	errors	time	nodes
eye	1	0.02	0
cancer	1	0.21	0
Parkinson	0	0	6
$berrichon_{1nc}$	2	4.73	8805
$berrichon_1$	23	5.81	8384
$berrichon_{2nc}$	41	5.89	6170
$berrichon_2$	106	17.23	15445
$langlade_1$	38	12.28	391
$langlade_2$	89	60.56	17857
$langlade_3$	39	14.19	6731
$langlade_4$	43	59.7	3520
$moissac_1$	0	0	5
$moissac_2$	0	0.51	6
$moissac_3$	0	0	4
$moissac_4$	0	0	5
$moissac_5$	0	1.02	5
$moissac_6$	0	5.64	6

Table 3: Solving real pedigree instances. Columns: name of the instance, optimal number of errors found, CPU time in seconds, and number of visited nodes.

4.4.3 `toulbar2` features evaluation

Because `toulbar2` contains a variety of technical ingredients, we wanted to know the importance of the different mechanisms that have been activated in the previous evaluations such as variable elimination during search (VarElim), dynamic variable ordering based on conflicts (conflict), EDAC based lower bound vs FDAC based lower bound (EDAC vs FDAC), and binary branching vs. n-ary branching.

For binary branching, which is known to be theoretically better than n-ary branching in terms of associated proof systems, we always observed a better behavior for binary branching in practice too. For a finer analysis, Fig. 8 reports the CPU-time of `toulbar2` on the $pedclass_C$ with different combinations of the above options. The use of EDAC propagation shows to be very effective in comparison to FDAC. As the size of instances increase EDAC is an order of magnitude better. The combination of the conflict heuristic and variable elimination has the best performance and corresponds to the combination used in all the previous experiments. So, no single feature explains the good results of `toulbar2`.

5 Conclusion

In this paper, we have presented a direct application of the Weighted CSP framework to a difficult problem which occurs very frequently in genetics. Cleaning genotyping data is a prerequisite before doing any further genetic analysis. In particular, we are interested in detecting Mendelian errors and providing an optimal correction. Compared to existing tools dedicated to this problem [22, 25, 2, 28], the novelty of our approach is to provide an optimal correction based on parsimony or maximum likelihood criterion for large loopy pedigree data as they are common in animal breeding.

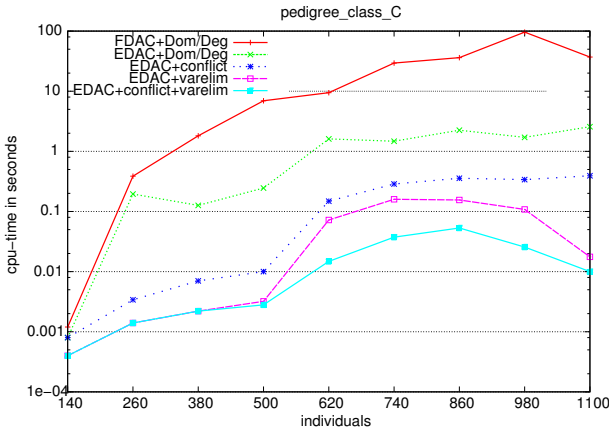

Figure 8: Solving time comparison of `toulbar2` different options.

For the parsimony problem, we were able to solve real pedigrees with up to 120,000 individuals in a few seconds. These problems were modeled as soft constraint networks with up to 9,500 variables and 17,000 constraints (*berrichon*$_2$ instance, after `toulbar2` preprocessing and *varelim*). Solving such a large network is possible thanks to the powerful lower bounds provided by soft local consistencies, in particular EDAC extended to ternary constraints.

This application lead us to the development of new algorithms, described in Section 3, for non-binary constraints. Although our presentation was restricted to ternary constraints, we believe it can be directly generalized to *n*-ary constraints, by considering all the intermediate arity levels (from 1 to *n*) and extending costs, in a minimal way, from one level to the next one by following a cost operation order similar to the order shown in Figure 3.

For MAP and MPE, we have shown on simulated data that MPE is a good approximation of MAP and is orders of magnitude faster to solve than MAP. However, on large real pedigrees, MPE could not be solved by `toulbar2`. Other techniques, such as structural learning [12] or using a specific dominance rule described in [10] (*Allele Recoding*), may be useful to break this barrier.

In the future, we will explore more complex probabilistic models in order to detect non Mendelian errors [9]. It implies working on multi-locus models, where other interesting biological questions have been recently investigated by the AI community [19, 20].

Acknowledgements This work was supported by a grant from the french "Agence Nationale pour la Recherche" (contract STAL-DEC-OPT). We are also very grateful to Zulma Vitezica and Isabelle Palhiere for the valuable discussions we had on the Mendelian error detection problem and for providing us with a simulator and several real pedigree instances. Thanks also to Eduardo Manfredi for convincing us to put efforts on a dedicated software, `MendelSoft`[6] based on `toulbar2`.

References

[1] Aceto, L., Hansen, J. A., Ingólfsdóttir, A., Johnsen, J., and Knudsen, J. The complexity of checking consistency of pedigree information and related problems. *Journal of Computer Science Technology* (2004), 19(1):42–59.

[6] http://www.inra.fr/mia/T/MendelSoft/.

[2] Bennewitz, J., Reinsch, N., and Kalm, E. GENCHECK: A program for consistency checking and derivation of genotypes at co-dominant and dominant loci. *Journal of Animal Breeding and Genetics* (2002), 119(5):350–360.

[3] Bistarelli, S., Fargier, H., Montanari, U., Rossi, F., Schiex, T., and Verfaillie, G. Semiring-based CSPs and valued CSPs: Frameworks, properties and comparison. *Constraints* (1999), 4:199–240.

[4] Cooper, M. High-order Consistency in Valued Constraint Satisfaction. *Constraints* (2005), 10(3):283–305.

[5] Cooper, M. Reduction operations in fuzzy or valued constraint satisfaction. *Fuzzy Sets and Systems* (2003), 134(3):311–342.

[6] Cooper, M., de Givry, S., and Schiex, T. Optimal soft arc consistency. In *Proc. of IJCAI-2007*:68–73, Hyderabad, India.

[7] Dechter, R. *Constraint Processing*. Morgan Kaufmann Publishers, 2003.

[8] Eén, N., and Sörensson, N. Translating Pseudo-Boolean Constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation* (2006), 2:61–96.

[9] Ehm, M. G., Cottingham Jr., R. W., and Kimmel, M. Error Detection in Genetic Linkage Data Using Likelihood Based Methods. *American Journal of Human Genetics* (1996), 58(1):225–234.

[10] Fishelson, M., Dovgolevsky, N., Geiger, D. Maximum Likelihood Haplotyping for General Pedigrees. *Human Heredity* (2005), 59:41–60.

[11] de Givry, S., Larrosa, J., Meseguer, P., and Schiex, T. Solving Max-SAT as weighted CSP. In *Proc. of CP-2003*:363–376, Cork, Ireland.

[12] de Givry, S., Schiex, T., and Verfaillie, G. Exploiting tree decomposition and soft local consistency in weighted CSP. In *Proc. of AAAI-2006*, Boston, MA.

[13] Larrosa, J. On Arc and Node Consistency in weighted CSP. In *Proc. of AAAI-2002*:48–53, Edmondton.

[14] Larrosa, J., and Dechter, R. On the dual representation of non-binary semiring-based CSPs. In *CP'2000 Workshop on Soft Constraints*, Singapore.

[15] Larrosa, J., de Givry, S., Heras, F., and Zytnicki, M. Existential arc consistency: getting closer to full arc consistency in weighted CSPs. In *Proc. of IJCAI-2005*:84–89, Edinburgh, Scotland.

[16] Larrosa, J., Morancho, E., and Niso, D. On the practical applicability of Bucket Elimination: Still-life as a case study. *Journal of Artificial Intelligence Research* (2005), 23 :421–440.

[17] Larrosa, J., and Schiex, T. In the quest of the best form of local consistency for weighted CSP. In *Proc. of IJCAI-2003*:239–244, Acapulco, Mexico.

[18] Lecoutre, C., Sais, L., Tabary, S., and Vidal, V. Last Conflict based Reasoning. In *Proc. of ECAI-2006*:133–137, Trento, Italy.

[19] Lynce, I., and Marques Silva, J.P. Efficient Haplotype Inference with Boolean Satisfiability. In *Proc. of AAAI-2006*, Boston, MA.

[20] Marinescu, R., and Dechter, R. Memory Intensive Branch-and-Bound Search for Graphical Models. In *Proc. of AAAI-2006*, Boston, MA.

[21] Mengshoel, O., Roth, D., and Wilkins, D. Controlled generation of hard and easy Bayesian networks: Impact on maximal clique size in tree clustering. *Artificial Intelligence* (2006), 170(16-17):1137–1174.

[22] O'Connell, J. R., and Weeks, D. E. PedCheck: a program for identification of genotype incompatibilities in linkage analysis. *American Journal of Human Genetics* (1998), 63(1):259–66.

[23] O'Connell, J. R., and Weeks, D. E. An optimal algorithm for automatic genotype elimination. *American Journal of Human Genetics* (1999), 65(6):1733–40.

[24] Park, J. D., and Darwiche, A. Complexity results and approximation strategies for MAP explanations. *Journal of Artificial Intelligence Research* (2004), 21:101–133.

[25] Saito, M., Saito, A., and Kamatani, N. Web-based detection of genotype errors in pedigree data. *Journal of human genetics* (2002), 47(7):377–379.

[26] Schiex, T. Arc consistency for soft constraints. In *Proc. of CP-2000*:411–424, Singapore.

[27] Sheini, H., and Sakallah, K. Pueblo: A Hybrid Pseudo-Boolean SAT Solver. *Journal on Satisfiability, Boolean Modeling and Computation* (2006), 2:61–96.

[28] Thomas, A. GMCheck: Bayesian error checking for pedigree genotypes and phenotypes. *Bioinformatics* (2005), 21(14):3187–3188.

[29] Vitezica, Z., Elsen, J.-M., Rupp, R., and Díaz, C. Using genotype probabilities in survival analysis: a scrapie case. *Genetics Selection Evolution* (2005), 37(4):403–415.

[30] Vitezica, Z., Mongeau, M., Manfredi, E., and Elsen, J.-M. Selecting Loop Breakers in General Pedigrees. *Human Heredity* (2004), 57:1–9.