CONGRESSUS NUMERANTIUM 237 (2026) 51–77

Congressus Numerantium
www.combinatorialpress.com/cn

Combinatorial Press

Abstract questionnaires and FS-decision digraphs

Jiaye Chen, Suzan Kadri, Mateja Šajna* and Ioana Schiopu-Kratina

ABSTRACT

A questionnaire is a sequence of multiple choice questions aiming to collect data on a population. We define an *abstract questionnaire* as an ordered pair $(N, \mathcal{M})$, where N is a positive integer and $\mathcal{M} = (m_0, m_1, \ldots, m_{N-1})$ is an N-tuple of positive integers, with m_i, for $i \in \mathbb{Z}_N$, as the number of possible answers to question i. An abstract questionnaire may be endowed with a skip-list (which tells us which questions to skip based on the sequence of answers to the earlier questions) and a flag-set (which tells us which sequences of answers are of special interest). An FS-decision tree is a decision tree of an abstract questionnaire that also incorporates the information contained in the skip-list and flag-set. The main objective of this paper is to represent the abstract questionnaire using a directed graph, which we call an FS-decision digraph, that contains the full information of an FS-decision tree, but is in general much more concise. We present an algorithm for constructing a fully reduced FS-decision digraph, and develop the theory that supports it. In addition, we show how to generate all possible orderings of the questions in an abstract questionnaire that respect a given precedence relation.

Keywords: survey, questionnaire, abstract questionnaire, decision tree, FS-decision tree, FS-decision digraph

2020 Mathematics Subject Classification: 05C85, 05C20.

1. Introduction

A questionnaire, from the perspective of this paper, is a sequence of multiple choice questions aiming to collect data on a population. A well-designed questionnaire maximizes the response rate while minimizing the response burden, resulting in the collection of good quality data with minimum bias. The process of designing an effective questionnaire

* Corresponding author.

Received 29 Sep 2025; Revised 20 May 2026; Accepted 01 Jun 2026; Published Online 12 June 2026.

DOI: 10.61091/cn237-04

© 2026 The Author(s). Published by Combinatorial Press. This is an open access article under the CC BY license (https://creativecommons.org/licenses/by/4.0/).

typically relies on its content (that is, the actual questions and possible responses). In this paper, however, the focus is on *abstract questionnaires*; that is, we shall not be concerned with the content. Moreover, our main focus will be not the design of the questionnaire itself, but rather on how to represent it by a graph in a way that is as simple and concise as possible, yet retains full information on the questionnaire.

The idea to represent a questionnaire by a graph is not new. In the literature, we find two basic approaches to this task: flow charts (see, for example, [1, 2, 5, 6, 7, 8, 9, 10]) and decision trees (see [4]). A flow chart (called survey chart in [8]) is a directed graph whose vertices are essentially the questions (each question corresponding to a single vertex, possibly with some additional vertices representing additional controls), and whose arcs (directed edges) represent responses that lead from one question to another. Each arc is labeled with the corresponding response, and is sometimes additionally assigned other parameters (for example, probability of the response), depending on the purpose of the graph model. In a decision tree, however, each question corresponds to a level, and thus may be represented by many vertices, each corresponding to a different sequence of responses to the previous questions. The edges from a given vertex to its children in the tree represent the possible responses to the question. The ordering of the children at each vertex uniquely identifies the responses, so no additional labels on the edges are necessary.

In this paper, we start by modelling a questionnaire with a decision tree, and then condense it into a so-called decision digraph, which retains full information contained in the decision tree. Decision digraphs are a novel way to represent a questionnaire, and can be considered a compromise between decision trees and flow charts: they are potentially much smaller than decision trees, but a lot simpler than flow charts since they do not require any labels on the arcs. Additionally, we endow an abstract questionnaire with a flag-set, which tells us which sequences of answers are of special interest, and a skip-list, which tells us which questions are to be skipped based on the sequence of answers to the earlier questions. As far as we can tell, flagging has not previously been considered in the literature, while skipping has been treated only with the knowledge of the content of the questionnaire (see [3]). When flagging and skipping are taken into account, we talk of FS-decision trees and FS-decision digraphs. We point out that in both of these representations, the order of the questions is fixed; that is, the order in which the questions are asked does not depend on the sequence of responses to the previous questions. An algorithm for generating all possible orderings of the questions in an abstract questionnaire that satisfies a given precedence relation is discussed in Section 3.

This paper is organized as follows. In Section 2 we give the prerequisites on graphs and relations. After a brief discussion of the ordering of questions in Section 3, we move on to the main topic, representing the flow of a questionnaire by a graph, presented in Section 4. In Subsection 4.1, we introduce flag-sets, skip-lists, and FS-decision trees, and present an algorithm for constructing an FS-decision tree of an abstract questionnaire. Vertex equivalence in an FS-decision tree, and FS-decision digraphs are discussed in Subsection 4.2; here, we also present an algorithm that constructs an FS-decision digraph, and prove a theorem that shows that the resulting digraph is fully reduced (that is, as small as possible). In the last subsection we talk about how to generate a skip-list and

a compatible flag-set that can serve as inputs for our algorithm from a more intuitively constructed pre-skip-list and pre-flag-set. Finally, in Section 5 we give an example of a simple concrete questionnaire to illustrate the contributions of Sections 3 and 4.

2. Prerequisites

An *abstract questionnaire* is an ordered pair $(N, \mathcal{M})$, where N is a positive integer, and $\mathcal{M} = (m_0, m_1, \ldots, m_{N-1})$ is an N-tuple of positive integers. An abstract questionnaire $(N, \mathcal{M})$ models a questionnaire with N questions (labelled $0, 1, \ldots, N-1$) such that question i has m_i possible answers. We denote the set of possible answers to question i by A_i; usually, we assume that $A_i = \mathbb{Z}_{m_i}$. An abstract questionnaire $(N, \mathcal{M})$ is called *binary* if $\mathcal{M} = (2, 2, \ldots, 2)$; that is, if every question has exactly two answers.

2.1. Graphs, trees, and digraphs

A *graph* G is an ordered triple (V, E, ψ), where V is a non-empty finite set, E is a finite set disjoint from V, and ψ is a function assigning to each element of E an unordered pair of elements of V. Sets V and E are the *vertex set* and *edge set*, respectively, of the graph G, and ψ is its *incidence function*. The elements of V and E are called the *vertices* and *edges* of G, respectively. Vertices u and v in G are said to be *adjacent* or *neighbours* if $\psi(e) = \{u, v\}$ for some $e \in E$.

In the definition of a graph, the incidence function may be omitted and edges may be identified with unordered pairs of vertices if there is no ambiguity. We then write $e = \{u, v\}$, or shortly $e = uv$, instead of $\psi(e) = \{u, v\}$.

A graph $G' = (V', E', \psi')$ is a *subgraph* of a graph $G = (V, E, \psi)$ if $V' \subseteq V$, $E' \subseteq E$, and ψ' is the restriction of ψ to E'. If $G' = (V', E', \psi')$ is a subgraph of $G = (V, E, \psi)$ such that $E' = \{e \in E : \psi(e) = \{u, v\}, u, v \in V'\}$, then G' is the *subgraph of G induced by V'*, and we write $G' = G[V']$.

A (v_0, v_k)-*path* (of *length* k) in a graph $G = (V, E)$ is a sequence $v_0 e_1 v_1 e_2 v_2 \ldots v_{k-1} e_k v_k$, where $v_0, \ldots, v_k$ are distinct vertices of G; $e_1, \ldots, e_k$ are edges of G, and $e_i = v_{i-1} v_i$ for all $i = 1, \ldots, k$.

A *cycle* (of *length* k) in a graph $G = (V, E)$ is a sequence $v_0 e_1 v_1 e_2 v_2 \ldots v_{k-1} e_k v_k$, where $v_0, \ldots, v_{k-1}$ are distinct vertices of G while $v_k = v_0$; $e_1, \ldots, e_k$ are distinct edges of G, and $e_i = v_{i-1} v_i$ for all $i = 1, \ldots, k$.

A graph $G = (V, E)$ is said to be *connected* if for all $u, v \in V$, there exists a (u, v)-path in G. The *distance* $dist(u, v)$ between vertices u and v in a connected graph G is the length of a shortest (u, v)-path in G.

A connected graph with no cycles is called a *tree*. A *rooted tree* is a tree with a distinguished vertex called the *root*. If T is a rooted tree with root r, and v is any vertex of T, then the unique (r, v)-path in T defines a sense of direction (away from the root). The vertices of T on this unique (r, v)-path, excluding v, are called the ancestors of v, and the neighbours of v not on this path are called the *children* of v. A vertex is a *descendant* of v if v is its ancestor. A rooted tree is said to be *ordered* if the set of children at each vertex is ordered. If T is a rooted tree and $v \in V(T)$, then the *subtree of T rooted at*

v is the subgraph of T induced by the subset of vertices that contains v and all of its descendants.

A *directed graph* (shortly *digraph*) D is an ordered triple (V, A, ψ), where V is a non-empty finite set, A is a finite set disjoint from V, and ψ is a function assigning to each element of A an ordered pair of elements of V. Sets V and A are the *vertex set* and *arc set*, respectively, of the digraph D, and ψ is its *incidence function*. The elements of V and A are called the *vertices* and *arcs* of D, respectively. Again, the incidence function may be omitted if there is no ambiguity, and we write $a = (u, v)$ instead of $\psi(a) = (u, v)$. If $u, v \in V$ are such that $(u, v) \in A$, then u is an *in-neighbour* of v, and v is an *out-neighbour* of u. Distinct arcs $a_1, a_2 \in A$ are said to be *parallel* if $\psi(a_1) = \psi(a_2)$.

A *subdigraph* and *vertex-set induced subdigraph* of a digraph are defined analogously to graphs.

A *directed cycle* of length k in a digraph $D = (V, A)$ is a sequence $v_0 a_1 v_1 a_2 v_2 \dots v_{k-1} a_k v_k$, where $v_0, \dots, v_{k-1}$ are distinct vertices of D while $v_k = v_0$; $a_1, \dots, a_k$ are arcs of D, and $a_i = (v_{i-1}, v_i)$ for all $i = 1, \dots, k$. A directed cycle of length one corresponds to an arc of the form (v, v), which is called a *directed loop*.

2.2. Relations, transitive closure, partial orders, and total orders

A *binary relation* on a finite set S is any subset of $S \times S$; that is, a set of ordered pairs of elements from S. A binary relation R on a set S is said to be *reflexive* if $(a, a) \in R$ for all $a \in S$; *irreflexive* if $(a, a) \notin R$ for all $a \in S$; *antisymmetric* if $(a, b), (b, a) \in R$ implies $a = b$, for all $a, b \in S$; and transitive if $(a, b), (b, c) \in R$ implies $(a, c) \in R$, for all $a, b, c \in S$. Elements $a, b \in S$ are said to be *comparable* with respect to R if $(a, b) \in R$ or $(b, a) \in R$.

If R and R' are two binary relations on a set S, and $R \subseteq R'$, then R' is said to *extend* R.

The *transitive closure* (*reflexive closure*, resp.) of a binary relation R on a set S is a minimal binary relation on the set S that is transitive (reflexive, resp.) and extends R.

A binary relation that is reflexive, antisymmetric, and transitive is called a *partial order*. A *minimal element* in a partial order R on the set S is an element $m \in S$ such that $(x, m) \in R$ implies $x = m$, for all $x \in S$.

If R is a partial order on a set S, and every pair of elements $a, b \in S$ are comparable with respect to R, then R is called a *total order*. It is well-known that every partial order has a total order that extends it.

A binary relation R on a set S is equivalent to the digraph (S, R), which has no parallel arcs; conversely, a digraph with no parallel arcs can be viewed as a binary relation. Hence the terms defined in Section 2.1 for digraphs (for example, directed cycle) apply to binary relations as well.

3. Ordering questions in a questionnaire

In this section, we describe a procedure for constructing all possible orderings of the questions in a questionnaire. Such orderings will be fixed; that is, independent from

the responder and independent from the responses to any previous questions. We will, however, assume that these orderings respect a given precedence relation. In other words, our input will be a binary relation R on the set of questions $\mathbb{Z}_N$ such that $(a, b) \in R$ if question a must be asked before question b. Note that we may assume that R is irreflexive. Our procedure is based on two well-known algorithms: the Roy-Warshall Algorithm for constructing the transitive closure of a binary relation, and Topological Sorting for constructing a total order that extends a given partial order.

The following observation will allow us to determine whether the input precedence relation is admissible (that is, extendible to a total order) or not.

Lemma 3.1. *Let R be an irreflexive binary relation on a set S, and R^* its transitive closure. Then the following are equivalent.*

(i) *R has no directed cycles.*

(ii) *R^* has no directed loops.*

(iii) *R^* has no directed cycles.*

(iv) *There exists a total order extending R.*

Proof. (i) $\Rightarrow$ (ii): This is obvious.

(ii) $\Rightarrow$ (iii): This is obvious.

(iii) $\Rightarrow$ (iv): Assume R^* has no directed cycles, and let $\overline{R^*}$ be the reflexive closure of R^*. Then $\overline{R^*}$ is reflexive by definition, and transitive since R^* is transitive. We claim $\overline{R^*}$ is antisymmetric. Take any $a, b \in S$ such that $(a, b), (b, a) \in \overline{R^*}$. If $a \neq b$, then $(a, b), (b, a) \in R^*$, and by transitivity, we have $(a, a) \in R^*$ — a contradiction since R^* has no directed cycles. Hence the implication $(a, b), (b, a) \in \overline{R^*} \Rightarrow a = b$ holds for all $a, b \in S$.

We conclude that $\overline{R^*}$ is a partial order, and hence there exists a total order T extending $\overline{R^*}$. Since $\overline{R^*}$ extends R, we have that T is a total order extending R.

(iv) $\Rightarrow$ (i): If T is a total order extending R, then T has no directed cycles of length at least 2, and hence R has no directed cycles of length at least 2. Since R is irreflexive, it also has no directed cycles of length 1. □

Note that in all of our algorithms, (ordered) lists are denoted using square brackets, and concatenation of lists is denoted with a plus sign; that is, $[x_1, \ldots, x_n] + [y_1, \ldots, y_m] = [x_1, \ldots, x_n, y_1, \ldots, y_m]$.

Algorithm 3.2. Constructing all possible orderings of the questions in a questionnaire

procedure Orderings(N, R)

 # Input: N is the number of questions, R is an irreflexive binary relation (set of ordered pairs) on $\mathbb{Z}_N$.

 # Output: a list $\mathcal{T}$ of all possible total orders on $\mathbb{Z}_N$ that extend R.

    R^* := transitive closure of R *# use the Roy-Warshall Algorithm*

    if R^* has a directed loop **then return** None

    else *# construct the list $\mathcal{T}$ of all total orders*

        R := reflexive closure of R^*

        $S := \mathbb{Z}_N$ *# the underlying set*

$L := [\,]$ # *current total order*
$\mathcal{T} := [\,]$
TotalOrder$(S, R, L, \mathcal{T})$
return $\mathcal{T}$

procedure TotalOrder$(S, R, L, \mathcal{T})$
Input: S is the underlying set, R is a partial order (set of ordered pairs) on S, L is the current total order we are constructing, $\mathcal{T}$ is the list of total orders constructed so far.
Output: the list $\mathcal{T}$ of all possible total orders on S that extend R.
if $S = \emptyset$ **then** $\mathcal{T} = \mathcal{T} + [L]$ # *total order L is complete; add it to list $\mathcal{T}$*
else
$M :=$ the set of minimal elements of the partial order (S, R)
for $m \in M$ **do**
$L' := L + [m]$
$S' := S - \{m\}$
$R' := R \cap (S' \times S')$
TotalOrder$(S', R', L', \mathcal{T})$

4. Representing the flow of a questionnaire by a graph

In this section, we aim to represent the flow of a questionnaire by a graph. We shall start with the well-known decision tree representation, then introduce the FS-decision tree, that is, a more compact version of the decision tree that also encodes information on special answer sequences, and finally describe an FS-decision digraph, a much more compact graph that nevertheless contains the full information on the questionnaire. Throughout this section, we shall assume that the order of the questions has been fixed; that is, question 0 is asked first, then question 1, and so on until question $N-1$. An additional question N, with no answers, will represent the end.

Throughout this section, we shall assume that we have an abstract questionnaire $\mathcal{Q} = (N, \mathcal{M})$, and that $A_i = \mathbb{Z}_{m_i}$ the set of possible answers to question i. In addition, for all $i \in \mathbb{Z}_N$, we define $A_i^* = A_i \cup \{*\}$. The symbol $*$ will represent all possible answers, and hence may correspond to a skipped question.

Definition 4.1. Let k and ℓ be integers, $0 \le k \le \ell \le N-1$. A (k, ℓ)-*answer string* for an abstract questionnaire $\mathcal{Q} = (N, \mathcal{M})$ is an element of $A_k^* \times A_{k+1}^* \times \ldots \times A_\ell^*$; that is, a string of the form $a_k a_{k+1} \ldots a_\ell$, where $a_i \in A_i^*$ for all $i = k, k+1, \ldots, \ell$.

An *answer string* for $\mathcal{Q}$ is either the empty string, denoted ϵ, or a (k, ℓ)-answer string for $0 \le k \le \ell \le N-1$.

The concatenation of strings a and b will be denoted ab. If a is a (k, ℓ)-answer string and b is an $(\ell+1, \ell')$-answer string, for some $0 \le k \le \ell < \ell' \le N-1$, then ab is a (k, ℓ')-answer string.

We now formally define a decision tree for an abstract questionnaire $\mathcal{Q} = (N, \mathcal{M})$, as

well as an assignment of questions and answer strings to its vertices.

Definition 4.2. A *decision tree* T for the abstract questionnaire $\mathcal{Q} = (N, \mathcal{M})$ is an ordered rooted tree with $V(T) = \mathbb{Z}_n$, where $n = 1 + \sum_{i=0}^{N-1} m_0 m_1 \dots m_i$. We define the edge set of T, together with the *question assignment* $\kappa : V(T) \to \mathbb{Z}_{N+1}$ and *answer string assignment* $\alpha : V(T) \to \{\epsilon\} \cup \bigcup_{i=0}^{N-1} A_0 \times A_1 \times \ldots \times A_i$ recursively as follows.

(i) Vertex 0 is the root of T, and $\kappa(0) = 0$ and $\alpha(0) = \epsilon$.

(ii) A vertex u with $\kappa(u) = q$ has m_q children if $q < N$, and has no children if $q = N$.

(iii) If u is a vertex with $\kappa(u) = q$ and c is the j-th child of vertex u (for $j \in \mathbb{Z}_{m_q}$), then $\kappa(c) = \kappa(u) + 1$ and $\alpha(c) = \alpha(u)j$.

Observe that a vertex u with $\kappa(u) = q$ is at distance q from the root, and its answer string $\alpha(u)$ is a $(0, q-1)$-answer string.

The decision tree of an abstract questionnaire is easily constructed using a Breadth-First-Search algorithm; see Algorithm 4.10 with $F = \emptyset$ and $\mathcal{S} = (\emptyset, \dots, \emptyset)$.

4.1. FS-decision trees

In this section, we shall upgrade our decision tree for an abstract questionnaire in two ways. First, we are going to skip all unnecessary vertices; these are vertices corresponding to answer strings that we do not want to be included in the questionnaire. Second, we are going to flag all those vertices whose answer strings are of special interest (possibly contradictory, but should not be excluded). The latter feature, of course, has no effect on the structure of the tree. We call the resulting tree an FS-decision tree of the abstract questionnaire.

To that end, we make two additional assumptions on the input. First, we assume that we have, for each question $q \geq 1$, a set S_q of $(0, q-1)$-answer strings. The answer strings in S_q represent sequences of answers to questions 0 to $q-1$ for which question q should be skipped. The list $(S_0, S_1, \dots, S_N)$ will be called a skip-list (see Definition 4.4 below). Second, we assume that we are given a set of answer strings that are of special interest (contradictory or in any other way significant so that we wish to keep track of them); we call this set a flag-set (see Definition 4.3 below). We will then flag every vertex of the tree whose answer string is of special interest. The restrictions imposed on a skip-list in Definition 4.4 will ensure that an FS-decision tree has a unique skip-list, and the additional assumption of compatibility that we impose on a flag-set later on (see Definition 4.8) will guarantee uniqueness of the flag-set. Both of these properties will consequently improve the efficiency of our algorithm for constructing an FS-decision digraph in Section 4.2.

We formally define a flag-set and a skip-list as follows.

Definition 4.3. A *flag-set* for the abstract questionnaire $\mathcal{Q} = (N, \mathcal{M})$ is a set F of $(0, N-1)$-answer strings. A $(0, k)$-answer string $a_0 \dots a_k$ of $\mathcal{Q}$ is said to be *flagged* with respect to the flag-set F if for some i, $0 \leq i \leq k$, the $(0, N-1)$-answer string $a_0 \dots a_i * \dots *$ is in F.

Definition 4.4. A *skip-list* for the abstract questionnaire $\mathcal{Q} = (N, \mathcal{M})$ is an $(N+1)$-tuple $\mathcal{S} = (S_0, S_1, \ldots, S_N)$ such that the following hold.

(i) $S_0 = \emptyset = S_N$.

(ii) For each q, $1 \leq q \leq N$, we have that S_q is a set of $(0, q-1)$-answer strings.

(iii) For each q, $1 \leq q \leq N$, and each $(0, q-1)$-answer string a:

• if $a \in S_q$, then for all $j \in A_{m_q}$, we have that $aj \notin S_{q+1}$ and for all $b \in A^*_{m_{q+1}} \times \ldots \times A^*_{m_k}$, $q+1 \leq k \leq N-1$, the $(0,k)$-answer string ajb is not in S_{k+1}; and

• if $a \notin S_q$, then $a* \notin S_{q+1}$ and for all $b \in A^*_{m_{q+1}} \times \ldots \times A^*_{m_k}$, $q+1 \leq k \leq N-1$, the $(0,k)$-answer string $a * b$ is not in S_{k+1}.

Note that Requirement (iii) in Definition 4.4 will ensure that the set S_q contains precisely those $(0, q-1)$-answer strings that correspond to the skipped vertices of the FS-decision tree (see Definition 4.5 below).

We are now ready to define an FS-decision tree.

Definition 4.5. An *FS-decision tree* T for the abstract questionnaire $\mathcal{Q} = (N, \mathcal{M})$ with a flag-set F and skip-list $\mathcal{S} = (S_0, \ldots, S_N)$ is an ordered rooted tree with $V(T) = \mathbb{Z}_n$, for some integer $n \leq 1 + \sum_{i=0}^{N-1} m_0 m_1 \ldots m_i$. We define the edge set of T, together with the

(i) subset U of *skipped vertices*,

(ii) *question assignment* $\kappa : V(T) \to \mathbb{Z}_{N+1}$ and

(iii) *answer string assignment* $\alpha : V(T) \to \{\epsilon\} \cup \bigcup_{i=0}^{N-1} A_0^* \times A_1^* \times \ldots \times A_i^*$

recursively as follows.

(i) Vertex 0 is the root, $\kappa(0) = 0$ and $\alpha(0) = \epsilon$.

(ii) If u is a vertex with $\kappa(u) = q \leq N$, then $u \in U$ if and only if $\alpha(u) \in S_q$. (iii) If u is a vertex with $\kappa(u) = q < N$, then

• if $u \in U$, then u has exactly one child, say c, and $\kappa(c) = q+1$ and $\alpha(c) = \alpha(u)*$;

• if $u \notin U$, then u has exactly m_q children, and for each $j \in A_q$, the j-th child, say c, satisfies $\kappa(c) = q+1$ and $\alpha(c) = \alpha(u)j$.

(iii) A vertex u with $\kappa(u) = N$ has no children.

In addition, we define the *flag function* $\varphi : V(T) \to \mathbb{Z}_2$ as follows: for all $v \in V(T)$, we have $\varphi(v) = 1$ (and we say that v is *flagged* with respect to F) if and only if $\alpha(v)$ is flagged with respect to F.

From the above definition, it is clear that the skip-list of an abstract questionnaire $\mathcal{Q} = (N, \mathcal{M})$ uniquely defines its FS-decision tree.

In Lemma 4.7 below, we show the converse to this statement: namely, that each FS-decision tree arises from a unique skip-list. In the proof, we shall use the following lemma.

Lemma 4.6. *Let $\mathcal{Q} = (N, \mathcal{M})$ be an abstract questionnaire with a skip-list $\mathcal{S} = (S_0, \ldots, S_N)$, and T its FS-decision tree with answer string assignment α. For $0 \leq k \leq N-1$, let a be a $(0,k)$-answer string such that $a \in S_{k+1}$. Then there exists $u \in V(T)$ such that $\alpha(u) = a$.*

Proof. Let $a = a_0 \ldots a_k$, and let i, for $0 \leq i \leq k$, be the largest index such that there exists $v \in V(T)$ with $\alpha(v) = a_0 \ldots a_i$. Suppose $i < k$. Then there is no vertex in T with

answer string $a_0 \dots a_i a_{i+1}$. Suppose first that $a_{i+1} \neq *$. Then, by Definition 4.5, we know that $v \in U$ and $\alpha(v) \in S_{i+1}$, and by requirement (iii) of Definition 4.4, we have that $a = \alpha(v)a_{i+1} \dots a_k \notin S_{k+1}$, which is a contradiction. Hence $a_{i+1} = *$, and since there is no vertex in T with answer string $a_0 \dots a_i a_{i+1}$, Definition 4.5 tells us that $v \notin U$. Hence $\alpha(v) \notin S_{i+1}$, and it follows that $a = \alpha(v) * a_{i+2} \dots a_k \notin S_{k+1}$, again a contradiction.

We conclude that $i = k$, and hence there exists $u \in V(T)$ such that $\alpha(u) = a$. □

Lemma 4.7. *Let $\mathcal{Q} = (N, \mathcal{M})$ be an abstract questionnaire with a skip-list $\mathcal{S} = (S_0, \dots, S_N)$, and T its FS-decision tree. If T is also an FS-decision tree for $\mathcal{Q}$ with skip-list $\mathcal{S}' = (S'_0, \dots, S'_N)$, then $\mathcal{S} = \mathcal{S}'$.*

Proof. Let U, κ, and α be the set of skipped vertices, question assignment, and answer string assignment, respectively, for the FS-decision tree T. Observe that, by Definition 4.5, a vertex $v \in V(T)$ is in U if and only if $\alpha(v) \in S_{\kappa(v)+1}$, and also if and only if $\alpha(v) \in S'_{\kappa(v)+1}$.

Clearly, $S_0 = S'_0$ and $S_N = S'_N$ by definition. We show that $S_{k+1} = S'_{k+1}$ for all $k = 0, \dots, N-2$. Take any $k \in \{0, \dots, N-2\}$, and suppose $S'_{k+1} \neq S_{k+1}$. Then, without loss of generality, there exists a $(0,k)$-answer string $a = a_0 \dots a_k$ such that $a \in S'_{k+1} - S_{k+1}$.

Since $a \in S'_{k+1}$, by Lemma 4.6, there is a vertex $u \in V(T)$ such that $\alpha(u) = a$. However, by the first observation of this proof, we know $u \in U$, and hence $a \in S_{k+1}$, a contradiction.

We conclude that $S'_{k+1} = S_{k+1}$. □

From Definition 4.5, it is also clear that the flag function of an S-decision tree is uniquely determined by the flag-set. For the converse to hold, an additional condition on the flag-set — namely, compatibility with the skip-list, to be defined below — is required.

Definition 4.8. Let F be a flag-set of an abstract questionnaire $\mathcal{Q} = (N, \mathcal{M})$ with skip-list $\mathcal{S} = (S_0, \dots, S_N)$. Then F is said to be *compatible with* $\mathcal{S}$ if for each $f \in F$ there exists $\ell \in \{0, \dots, N-1\}$ such that the following hold:

(i) $f = f_0 \dots f_\ell * \dots *$;

(ii) for all $i \in \{0, \dots, \ell\}$, we have

$$f_0 \dots f_{i-1} \in S_i \iff f_i = *;$$

(iii) if $f_\ell \neq *$, then for some $j \in A_\ell$, the $(0, N-1)$-answer string $f_0 \dots f_{\ell-1} j * \dots *$ is not in F; and

(iv) for all $i \in \{0, \dots, \ell - 1\}$, we have that the $(0, N-1)$-answer string $f_0 \dots f_i * \dots *$ is not in F.

Note that for $i = 0$, Statement (ii) says that $\epsilon \in S_0 \iff f_0 = *$, in other words, that $f_0 \neq *$.

The following lemma will explain the significance of the parameter ℓ in Definition 4.8.

Lemma 4.9. *Let F be a flag-set and $\mathcal{S} = (S_0, \dots, S_N)$ a skip-list of an abstract questionnaire $\mathcal{Q} = (N, \mathcal{M})$, and assume that F is compatible with $\mathcal{S}$. Take any $(0, N-1)$-answer*

string $f_0 \dots f_\ell * \dots * \in F$, where ℓ satisfies the conditions in Definition 4.8. Then the following hold.

(a) *For all $i \in \{0, \dots, \ell\}$, there exists $v \in V(T)$ such that $\alpha(v) = f_0 \dots f_i$.*

(b) *Let $v \in V(T)$ be such that $\alpha(v) = f_0 \dots f_\ell$. Then v is flagged, every descendant of v is flagged, and every ancestor of v is not flagged.*

Proof. (a) We prove this statement by induction on i. Since $\epsilon \notin S_0$, by Definition 4.5, the root vertex has m_0 children, with answer strings $0, \dots, m_0 - 1$. Furthermore, by Definition 4.8, we know that $f_0 \neq *$. Hence indeed, there is a vertex with answer string f_0.

Suppose the claim holds for some i, $0 \leq i < \ell$. Let $v \in V(T)$ be such that $\alpha(v) = f_0 \dots f_i$. By Definition 4.5, if $f_0 \dots f_i \in S_{i+1}$, then v has a child with answer string $f_0 \dots f_i *$, and by Definition 4.8, we know that $f_{i+1} = *$. If, however, $f_0 \dots f_i \notin S_{i+1}$, then by Definition 4.5, vertex v has a child with answer string $f_0 \dots f_i j$ for all $j \in A_{i+1}$, and by Definition 4.8, we know that $f_{i+1} \neq *$. In both cases, we conclude that v has a child with answer string $f_0 \dots f_i f_{i+1}$.

The claim then follows by induction.

(b) Note that v exists by Statement (a). Let u be any ancestor of v. Then $\alpha(u) = f_0 \dots f_i$ for some $i < \ell$, and by Definition 4.8, we know that the $(0, N-1)$-answer string $f_0 \dots f_j * \dots *$ is not in F for all $j < \ell$. Hence by Definition 4.3, vertex u is not flagged.

Let u be a vertex of T that is either v itself or a descendant of v. Then $\alpha(u) = f_0 \dots f_\ell f_{\ell+1} \dots f_k$ for some k, $\ell \leq k \leq N-1$, and $f_{\ell+1} \dots f_k \in A_{\ell+1}^* \times \dots \times A_k^*$. Then by Definition 4.3, since $f_0 \dots f_\ell * \dots * \in F$, we know that u is flagged. □

We are now ready to outline our algorithm for constructing an FS-decision tree of a an abstract questionnaire.

Algorithm 4.10. Constructing an FS-decision tree of an abstract questionnaire

procedure FS-tree$(N, \mathcal{M}, F, \mathcal{S})$

 # Input: abstract questionnaire $(N, \mathcal{M})$ with flag-set F and skip-list $\mathcal{S} = (S_0, \dots, S_N)$.

 # Output: FS-decision tree T for $(N, \mathcal{M})$ with the subset of skipped vertices U, question assignment κ, answer string assignment α, and flag function φ.

 # The vertices of T are labelled $0, 1, 2, \dots$.

 # $Out(u)$ is the list of children of vertex u.

```
    U := ∅
    κ(0) := 0
    α(0) := ϵ
    φ(0) := 0
    L := [0]        # BFS queue of unprocessed vertices
    c := 0          # last vertex label used
    while L ≠ [ ] do
        u := first vertex in L
        remove u from L
        q := κ(u)
```

```
        Out(u) := [ ]
        if q < N then
            if u ∈ U then A = {∗}
            else A = Z_{m_q}
            for all a_q ∈ A do
                c := c + 1
                κ(c) = q + 1
                α(c) = α(u)a_q
                Out(u) :=Out(u) + [c]
                if α(c) ∈ S_{κ(c)} then U := U ∪ {c}        # c is a skipped vertex
                if flagged(α(c), F) = 1 then φ(c) := 1
                else φ(c) := 0
                L := L + [c]
    return c, Out, U, κ, α, φ        # V(T) = {0, 1, ..., c}
```

```
procedure flagged(a_0 ... a_k, F)
# Input: a (0, k)-answer string a_0 ... a_k, flag-set F.
# Output: 1 if a_0 ... a_k is flagged; 0 otherwise.
    Ans:= 0
    for i := 0 to k
        if the (0, N − 1)-answer string a_0 ... a_i ∗ ... ∗ ∈ F
        then
            Ans:= 1
            break
    return Ans
```

4.2. FS-decision digraphs

It may happen — and very frequently it does — that one subtree of an FS-decision tree looks just like another subtree, except for the initial substring of all corresponding answer strings. In this case, it is advantageous to merge these two subtrees and thus reduce the size of the graph. There are two things to be careful about. First, we do not wish to lose any information; to that end, we replace the answer string of each vertex with the set of answer strings of all vertices it represents. Second, the resulting graph will no longer be a tree. However, if we imagine our FS-decision tree as a digraph with all edges directed away from the root, then our merged graph retains this sense of direction away from the root; even though it may not be acyclic, it does not have any directed cycles. (In fact, every cycle decomposes into two paths of the same length directed in opposite ways.) This sense of direction is, of course, the flow of the questionnaire.

4.2.1. Vertex equivalence and FS-decision digraphs. The following definition will make the idea of similar subtrees precise.

Definition 4.11. Let $\mathcal{Q} = (N, \mathcal{M})$ be an abstract questionnaire with flag-set F and skip-

list $\mathcal{S}$, and let T be the FS-decision tree for $\mathcal{Q}$, with question assignment κ and answer string assignment α. Let v and w be two vertices of T, and T_v and T_w the subtrees of T rooted at v and w, respectively. Furthermore, assume $\kappa(v) = \kappa(w) = k$.

Then vertices v and w are said to be *equivalent* in T if there exists a bijection $\Phi : V(T_v) \to V(T_w)$ such that for each $u \in V(T_v)$,

(a) if $\alpha(u) = \alpha(v)x$, where $x = \epsilon$ or x is a $(k, \kappa(u) - 1)$-answer string, then $\alpha(\Phi(u)) = \alpha(w)x$; and

(b) $\alpha(u)$ is flagged $\iff \alpha(\Phi(u))$ is flagged.

The following properties of the mapping Φ from Definition 4.11 are easy to see.

Lemma 4.12. *Let $\mathcal{Q} = (N, \mathcal{M})$ be an abstract questionnaire with flag-set F and skip-list $\mathcal{S}$, and let T be the FS-decision tree for $\mathcal{Q}$, with the set of skipped vertices U, question assignment κ, and answer string assignment α. Let v and w be two vertices of T, and T_v and T_w the subtrees of T rooted at v and w, respectively. Furthermore, let $\kappa(v) = \kappa(w) = k$, and let $\Phi : V(T_v) \to V(T_w)$ be a bijection satisfying Property (a) from Definition 4.11. Then:*

(i) $\Phi(v) = w$;

(ii) *for all $u \in V(T_v)$, we have $\kappa(\Phi(u)) = \kappa(u)$;*

(iii) *Φ is unique;*

(iv) *Φ is an isomorphism from T_v to T_w; and*

(v) *for all $u \in V(T_v)$, if $m_{\kappa(u)} \geq 2$, then $u \in U$ if and only if $\Phi(u) \in U$.*

Proof. (i) Since T has a unique vertex with answer string $\alpha(w)$, this statement follows from Property (a) in Definition 4.11, with $x = \epsilon$.

(ii) Let $u \in V(T_v)$. Since, by assumption, $\kappa(v) = \kappa(w)$, by Property (a), the answer strings of u and $\Phi(u)$ are of the same length. Hence $\kappa(u) = \kappa(\Phi(u))$.

(iii) Since no two distinct vertices in an FS-decision tree have the same answer string, this statement follows directly from Property (a).

(iv) By (i) and Property (a), the bijection Φ maps v to w, and for any $u \in V(T_v)$, it maps the children of vertex u to the children of vertex $\Phi(u)$. Hence Φ is an isomorphism from T_v to T_w.

(v) Take any $u \in V(T_v)$, let $\ell = \kappa(u)$, and assume $m_\ell \geq 2$. If $u \in U$, then $\alpha(u) \in S_\ell$ and $\ell \leq N - 1$. By Definition 4.5, vertex u has exactly one child, and since Φ is an isomorphism from T_v to T_w that maps the root of T_v to the root of T_w, it also maps the children of u to the children of $\Phi(u)$. It follows that $\Phi(u)$ has exactly one child, and since $\kappa(\Phi(u)) = \ell$ and $m_\ell \geq 2$, it follows that $\Phi(u) \in U$.

Since Φ^{-1} is an isomorphism from T_w to T_v that satisfies Property (a) from Definition 4.11, the converse follows by symmetry. □

In other words, vertices v and w of an FS-decision tree are equivalent if there exists an isomorphism from T_v to T_w that preserves the flagging property as well as preserves each answer string minus the initial substring $\alpha(v)$.

A digraph resulting from merging some of the pairs of equivalent vertices will be called

an FS-decision digraph of the abstract questionnaire; see Definition 4.16 below. Any FS-decision digraph is an ordered levelled digraph, in the following sense.

Definition 4.13. An *ordered levelled digraph* is a digraph $D = (V, A)$ together with
(a) an ordering of the set of out-neighbours of each vertex, and
(b) a *level function* $\kappa : V \to \mathbb{N}$ assigning to each vertex v its level $\kappa(v)$ so that
(i) there exists at least one vertex v such that $\kappa(v) = 0$, and
(ii) if $(v, u) \in A$, then $\kappa(u) = \kappa(v) + 1$.

Note that an ordered rooted tree can be viewed as an ordered levelled digraph with κ being the distance from the root, all arcs being directed away from the root, and each set of out-neighbours ordered as the corresponding set of children in the tree. The analogue of a subtree rooted at a vertex v in a rooted tree is the subdigraph rooted at v, defined below. It is easy to see that, as stated in the subsequent lemma, a subdigraph rooted at v of an ordered levelled digraph is itself an ordered levelled digraph.

Definition 4.14. *Let $D = (V, A)$ be an ordered levelled digraph with level function κ, and $v \in V$. The* subdigraph of D rooted at v *is the induced digraph $D_v = D[V']$ where $V' = \{u \in V :$ there exists a directed $(v, u) -$ path in $D\}$.*

Lemma 4.15. *Let $D = (V, A)$ be an ordered levelled digraph with level function κ, and $v \in V$. Using the same ordering of the set of out-neighbours at each vertex, the subdigraph of D rooted at v is an ordered levelled digraph with level function $\kappa_v = \kappa - \kappa(v)$.*

We are now ready to define an FS-decision digraph of an abstract questionnaire. Note that the arc set of an ordered levelled digraph, as well as the ordering of the out-neighbours of each vertex, is fully defined by giving an (ordered) list of out-neighbours, to be denoted $\mathrm{Out}(v)$, for each vertex v. Note that $\mathrm{Out}(v)$ may contain repeated elements.

Definition 4.16. Let $\mathcal{Q} = (N, \mathcal{M})$ be an abstract questionnaire with flag-set F and skip-list $\mathcal{S}$, and let T be the FS-decision tree for $\mathcal{Q}$, with set of skipped vertices U, question assignment κ, answer string assignment α, and flag function φ.

The *FS-decision digraph D_T corresponding to T* is the orientation of T with all arcs directed away from the root. More precisely, D_T is an ordered levelled digraph with level function κ, and the ordering of the set of out-neighbours of each vertex inherited from T. The *answer-string assignment* $\mathcal{A}$ for D_T is defined as $\mathcal{A}(v) = \{\alpha(v)\}$ for all $v \in V(T)$. The set of skipped vertices and flag function are inherited from T.

An *FS-decision digraph* for $\mathcal{Q}$ is defined recursively as follows: it is either the FS-decision digraph D_T corresponding to T, or is obtained from any FS-decision digraph D via the operation merge(v, w), where v and w are vertices of D equivalent in T. The operation merge(v, w) is defined as follows:
(a) delete the subdigraph D_w of D rooted at w;
(b) for each in-neighbour p of w, replace each occurrence of w in $\mathrm{Out}(p)$ with v;
(c) for each $u \in V(D_v)$, if z is the corresponding vertex in D_w (that is, $z = \Phi(u)$

where $\Phi : T_v \to T_w$ is the unique isomorphism satisfying Definition 4.11), then adjoin the elements of $\mathcal{A}(z)$ to $\mathcal{A}(u)$;

(e) delete all $z \in V(D_w)$ from U;

(f) restrict κ and φ to $V(D) - V(D_w)$.

An FS-decision digraph for $\mathcal{Q}$ is said to be *fully reduced* if it contains no pair of distinct vertices that are equivalent in T.

In other words, D_T is the FS-decision digraph obtained from T in the obvious way, that is, by orienting each edge away from the root. Any other FS-decision digraph is obtained from D_T via a sequence of operations whereby the subdigraphs rooted at two equivalent vertices are merged while preserving all information encoded by the FS-decision tree. Recall that the list of out-neighbours of a vertex in an FS-decision digraph may contain repeated elements; in other words, an FS-decision digraph may contain parallel arcs.

Figure 1 shows, for an example of a binary questionnaire $\mathcal{Q}$, its FS-decision tree T and the FS-decision digraph corresponding to T (both represented by the tree on top), an FS-decision digraph for $\mathcal{Q}$ that is not fully reduced, and the fully reduced FS-decision digraph for $\mathcal{Q}$.

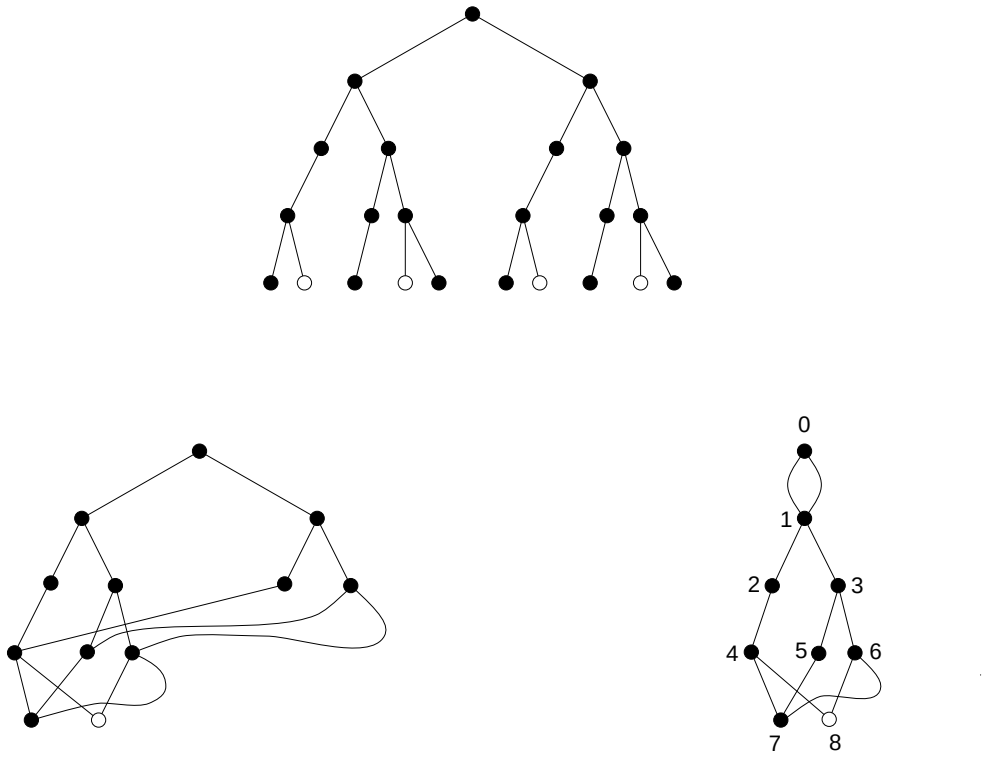

Fig. 1. The FS-decision tree T (top) for a binary questionnaire $\mathcal{Q}$. If we imagine all edges directed downwards, this figure also represents the FS-decision digraph D_T corresponding to T. Bottom left: an FS-decision digraph D' for $\mathcal{Q}$ that is not fully reduced. Bottom right: the fully reduced FS-decision digraph D'' for $\mathcal{Q}$. All edges are directed downwards, out-neighbours are ordered from left to right, and flagged and unflagged vertices are coloured white and black, respectively. Vertices in D'' are labelled in the order created by Algorithm 4.19.

Furthermore, Figures 2, 3, and 4 give a list of FS-decision trees for all binary question-

naires with at most 3 questions, and their fully reduced FS-decision digraphs.

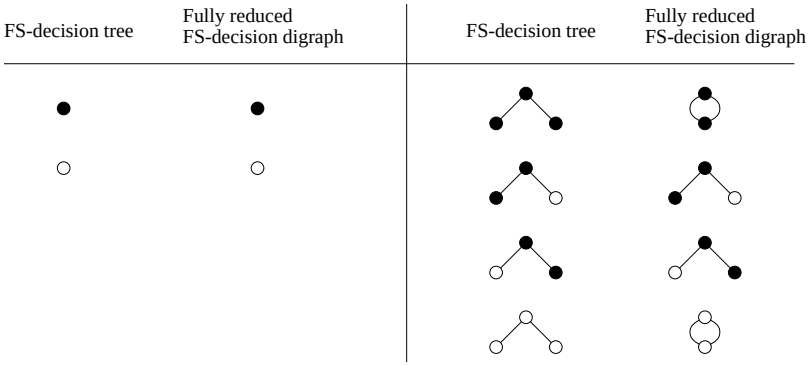

Fig. 2. FS-decision trees for binary questionnaires with at most two questions, and the corresponding fully reduced FS-decision digraphs. All edges are directed downwards, out-neighbours are ordered from left to right, and flagged and unflagged vertices are coloured white and black, respectively.

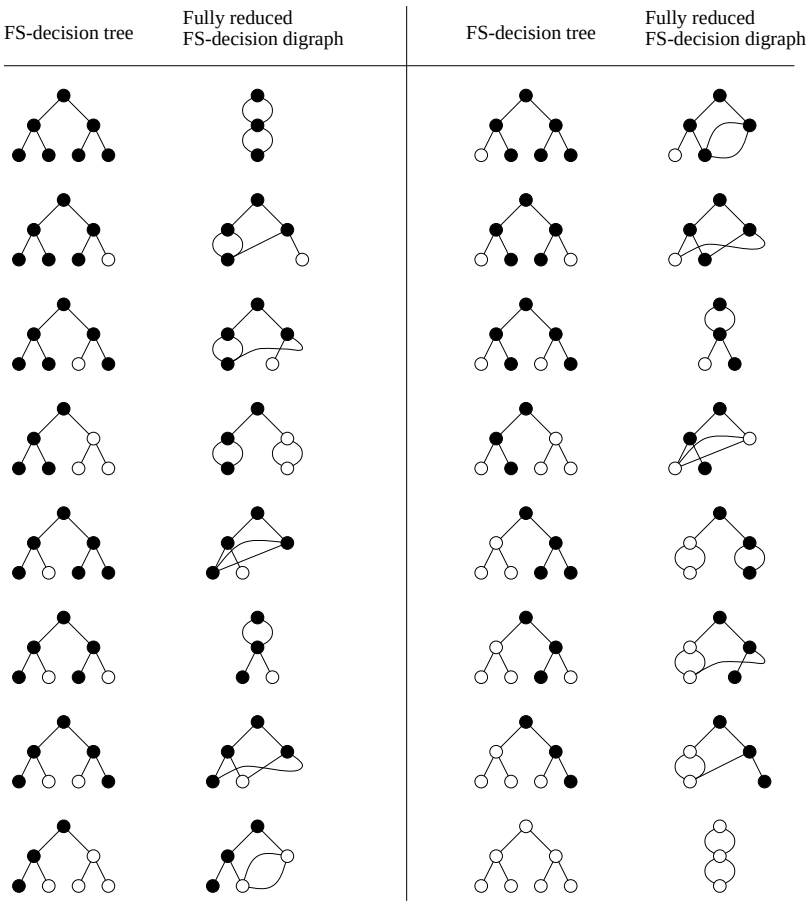

Fig. 3. FS-decision trees for binary questionnaires with three questions, and the corresponding fully reduced FS-decision digraphs. All edges are directed downwards, out-neighbours are ordered from left to right, and flagged and unflagged vertices are coloured white and black, respectively. Continued in Figure 4.

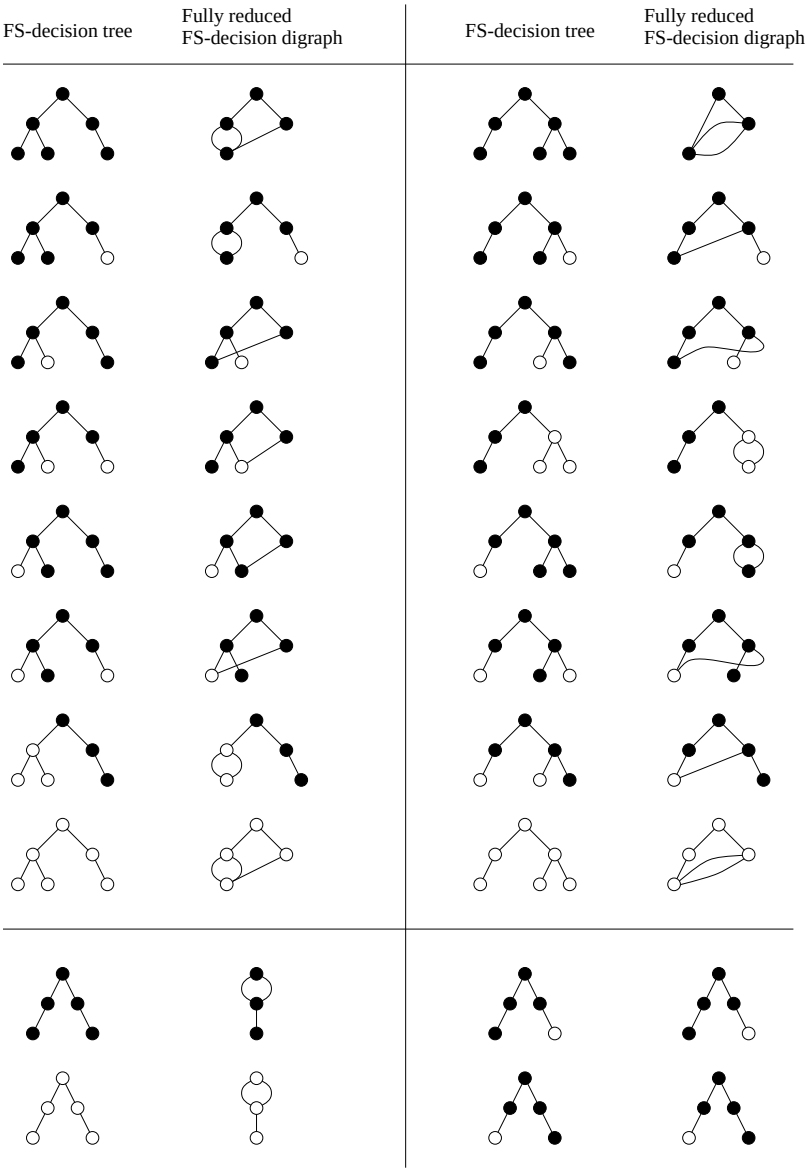

Fig. 4. FS-decision trees for binary questionnaires with three questions, and the corresponding fully reduced FS-decision digraphs. All edges are directed downwards, out-neighbours are ordered from left to right, and flagged and unflagged vertices are coloured white and black, respectively. Continued from Figure 3.

The obvious way to construct an FS-decision digraph would be as follows. First, construct the FS-decision tree, then establish equivalence between all (relevant) pairs of vertices. Finally, perform a sequence of merges until no more pairs of equivalent vertices remain. In order to save time and space, however, instead of constructing the FS-decision tree first, we will directly construct the FS-decision digraph that would result from merging all pairs of equivalent vertices; that is, we will be able to recognize such pairs of vertices without first constructing the subtrees rooted at them. We will then prove (Theorem 4.18) that this procedure results in the fully reduced FS-decision digraph.

4.2.2. Recognizing vertex equivalence. We shall now discuss how to determine whether two vertices in an FS-decision tree are equivalent. We would like to do this as efficiently as possible; that is, without prior construction of the subtrees rooted at the two vertices. For that purpose, we introduce the concepts of a local flag-set and local skip-list.

Definition 4.17. Let $k \in \mathbb{N}$, and let $a = a_0 \dots a_k$ be a $(0,k)$-answer string of an abstract questionnaire $(N, \mathcal{M})$ with flag-set F and skip-list $\mathcal{S} = (S_0, \dots, S_N)$.

(i) The *local flag-set at* a is a set F^a of $(k+1, N-1)$-answer strings defined as follows: if for some $i \leq k$, the $(0, N-1)$-answer string $a_0 \dots a_i * \dots * \in F$, then $F^a = \{* \dots *\}$; otherwise,

$$F^a = \{b \in A_{k+1}^* \times \dots \times A_{N-1}^* : ab \in F\}.$$

(ii) The *local skip-list at* a is a list $\mathcal{S}^a = (S_{k+1}^a, \dots, S_N^a)$ such that for each $i = k+1, \dots, N$,

$$S_i^a = \{b \in A_{k+1}^* \times \dots \times A_{i-1}^* : ab \in S_i\}.$$

Note that if $a \in S_{k+1}$, then $S_{k+1}^a = \{\epsilon\}$; otherwise, $S_{k+1}^a = \emptyset$.

In Theorem 4.18 below, which will be crucial for constructing FS-decision digraphs, we show that two vertices in an FS-decision tree are equivalent if and only if their local flag-sets, as well as their local skip-lists, are identical. However, for this to hold, the flag-set of the abstract questionnaire must be compatible with its skip-list, in the sense of Definition 4.8.

Theorem 4.18. *Let $\mathcal{Q} = (N, \mathcal{M})$ be an abstract questionnaire with a flag-set F and skip-list $\mathcal{S} = (S_0, \dots, S_N)$. Assume that $m_q \geq 2$ for all $q = 0, \dots, N-1$, and that F is compatible with $\mathcal{S}$. Let T be the FS-decision tree for $\mathcal{Q}$ with question assignment κ and answer string assignment α, and let $v, w \in V(T)$ be such that $\kappa(v) = \kappa(w) = k+1$. Furthermore, let $\alpha(v) = a = a_0 \dots a_k$ and $\alpha(w) = b = b_0 \dots b_k$.*

Then v and w are equivalent in T if and only if $F^a = F^b$ and $\mathcal{S}^a = \mathcal{S}^b$.

Proof. ($\Rightarrow$) Assume v and w are equivalent in T. Thus, by Lemma 4.12, there exists an isomorphism $\Phi : T_v \to T_w$ satisfying Properties (a) and (b) in Definition 4.11.

First, we show that $F^a = F^b$. Suppose first that $F^a = \{* \dots *\}$. Then, by Definition 4.17, for some $i \leq k$, the $(0, N-1)$-answer string $a_0 \dots a_i * \dots * \in F$. Hence v is flagged, and since $w = \Phi(v)$, so is w. That means that for some $j \leq k$, the $(0, N-1)$-answer string $b_0 \dots b_j * \dots * \in F$, and hence $F^b = \{* \dots *\} = F^a$.

Hence assume that $F^a \neq \{* \dots *\}$. This means that there is no $i \leq k$ such that the $(0, N-1)$-answer string $a_0 \dots a_i * \dots * \in F$, and consequently, the answer string $a_0 \dots a_k$ is not flagged. As $a_0 \dots a_k = \alpha(v)$ and $w = \Phi(v)$, we know that w and $\alpha(w)$ are not flagged. Hence there is no $i \leq k$ such that the $(0, N-1)$-answer string $b_0 \dots b_i * \dots * \in F$.

Take any $a_{k+1} \dots a_{N-1} \in F^a$. Then $a_0 \dots a_k a_{k+1} \dots a_{N-1} \in F$. Let $\ell \in \{0, \dots N-1\}$ be such that $a_0 \dots a_{N-1} = a_0 \dots a_\ell * \dots *$ and ℓ satisfies the conditions in Definition 4.8. Note that, by the above observation, $\ell > k$. By Lemma 4.9, there exists a vertex $u \in V(T)$ such that $\alpha(u) = a_0 \dots a_\ell$. Clearly $u \in V(T_v)$ and u is flagged. Let $z = \Phi(u)$.

Then $\alpha(z) = b_0 \dots b_k a_{k+1} \dots a_\ell$ and z is flagged. Hence for some $j \leq \ell$, the $(0, N-1)$-answer string $b_0 \dots b_k a_{k+1} \dots a_j * \dots * \in F$. Note that by the conclusion of the previous paragraph, we indeed have $j \geq k+1$. By Lemma 4.9, there exists a vertex $z' \in V(T)$ such that $\alpha(z') = b_0 \dots b_k a_{k+1} \dots a_j$, and necessarily $z' \in V(T_w)$. Let $u' = \Phi^{-1}(z')$. Then $\alpha(u') = a_0 \dots a_k a_{k+1} \dots a_j$, and since z' is flagged, so is u'. But then the $(0, N-1)$-answer string $a_0 \dots a_k a_{k+1} \dots a_i * \dots * \in F$ for some $k+1 \leq i \leq j$, and by Definition 4.8(iv), we must have $i \geq \ell$. Since $j \leq \ell$, we can see that $i = j = \ell$.

It follows that the $(0, N-1)$-answer string $b_0 \dots b_k a_{k+1} \dots a_\ell * \dots * \in F$, and hence the $(k+1, N-1)$-answer string $a_{k+1} \dots a_\ell * \dots * \in F^b$. Hence $a_{k+1} \dots a_{N-1} \in F^b$. By symmetry, we conclude that $F^a = F^b$.

Next, we show that $\mathcal{S}^a = \mathcal{S}^b$. If $S^a_{k+1} = \emptyset$, then $a \notin S_{k+1}$ and hence $v \notin U$. By Lemma 4.12(v) we have that $w \notin U$, and hence $b \notin S_{k+1}$ and $S^b_{k+1} = \emptyset$. By symmetry, we conclude that $S^a_{k+1} = S^b_{k+1}$. Now let $i \in \{k+1, \dots, N-1\}$, and take any $a_{k+1} \dots a_{i-1} \in S^a_i$. Then $a_0 \dots a_k a_{k+1} \dots a_{i-1} \in S_i$, and by Lemma 4.6, there exists a vertex $u \in V(T)$ such that $\alpha(u) = a_0 \dots a_k a_{k+1} \dots a_{i-1}$. Note that $u \in V(T_v)$. Let $z = \Phi(u)$. Then $\alpha(z) = b_0 \dots b_k a_{k+1} \dots a_{i-1}$. Since $\alpha(u) \in S_i$, we know that $u \in U$ and u has a single child. Hence z has a single child, and since $m_i \geq 2$, we have $z \in U$. It follows that $\alpha(z) \in S_i$, and hence $a_{k+1} \dots a_{i-1} \in S^b_i$. Using symmetry, we obtain $S^a_i = S^b_i$, and thus conclude that $\mathcal{S}^a = \mathcal{S}^b$.

($\Leftarrow$) Assume $F^a = F^b$ and $\mathcal{S}^a = \mathcal{S}^b$. Define a function

$$\Theta : \{a\} \cup \left(\{a\} \times \bigcup_{i=k+1}^{N-1} A^*_{k+1} \times \dots \times A^*_i \right) \quad \to \quad \{b\} \cup \left(\{b\} \times \bigcup_{i=k+1}^{N-1} A^*_{k+1} \times \dots \times A^*_i \right),$$

as follows: $\Theta(a) = b$ and

$$\Theta(a a_{k+1} \dots a_i) = b a_{k+1} \dots a_i \quad \text{for all } a_{k+1} \dots a_i \in A^*_{k+1} \times \dots \times A^*_i, \quad k+1 \leq i \leq N-1.$$

We show that Θ induces a bijection from $V(T_v)$ to $V(T_w)$.

Take any $u \in V(T_v)$. If $u = v$, then $\alpha(u) = a$ and $\alpha(w) = b = \Theta(a) = \Theta(\alpha(u))$ by the definition of Θ. Otherwise, $\alpha(u) = a a_{k+1} \dots a_\ell$ for some $a_{k+1} \dots a_\ell \in A^*_{k+1} \times \dots \times A^*_\ell$, $k+1 \leq \ell \leq N-1$. Let $i \leq \ell$ be the largest index such that there exists a vertex $z' \in V(T)$ with $\alpha(z') = b a_{k+1} \dots a_i$. Clearly, $i \geq k$, and suppose that $i < \ell$. Let $u', u'' \in V(T_v)$ be such that $\alpha(u') = a a_{k+1} \dots a_i$ and $\alpha(u'') = a a_{k+1} \dots a_i a_{i+1}$. Note that $a_{i+1} = *$ if and only if $u' \in U$, that is, if and only if $\alpha(u') \in S_{i+1}$, that is, if and only if $a_{k+1} \dots a_i \in S^a_{i+1}$. However, since there is no vertex z'' with $\alpha(z'') = b a_{k+1} \dots a_i a_{i+1}$, we have that $a_{i+1} = *$ if and only if $z' \notin U$, that is, if and only if $\alpha(z') \notin S_{i+1}$, that is, if and only if $a_{k+1} \dots a_i \notin S^b_{i+1}$. Since $S^a_{i+1} = S^b_{i+1}$, we have a contradiction. Hence $i = \ell$, which means that there exists a vertex $z \in V(T)$ such that $\alpha(z) = \Theta(\alpha(u))$. Note that necessarily $z \in V(T_w)$.

By symmetry, we conclude that Θ induces a bijection $\Phi : V(T_v) \to V(T_w)$ that satisfies Requirement (a) from Definition 4.11. It remains to show that every $u \in V(T_v)$ is flagged if and only if $\Phi(u)$ is flagged.

Take any $u \in V(T_v)$, and assume u is flagged. First suppose $u = v$. Then there is an index i, $0 \leq i \leq k$, such that the $(0, N-1)$-answer string $a_0 \dots a_i * \dots * \in F$. It follows

that $F^a = \{*\ldots*\}$, and hence $F^b = \{*\ldots*\}$. Consequently, for some j, $0 \leq j \leq k$, we have that the $(0, N-1)$-answer string $b_0 \ldots b_j * \ldots * \in F$, which implies that $\Phi(u)$ is flagged.

Hence assume $u \neq v$, and so $\alpha(u) = aa_{k+1} \ldots a_\ell$ for some $a_{k+1} \ldots a_\ell \in A_{k+1}^* \times \ldots \times A_\ell^*$, $k+1 \leq \ell \leq N-1$. Since u is flagged, for some i, $0 \leq i \leq \ell$, we have that the $(0, N-1)$-answer string $a_0 \ldots a_i * \ldots * \in F$. If $i \leq k$, then it follows that $F^a = \{*\ldots*\}$, and hence $F^b = \{*\ldots*\}$. Consequently, for some $j \leq k$, the $(0, N-1)$-answer string $b_0 \ldots b_j * \ldots * \in F$, and it follows that $\Phi(u)$ is flagged.

Hence assume $i \geq k+1$. Since the $(0, N-1)$-answer string $a_0 \ldots a_i * \ldots * \in F$, we have that the $(k+1, N-1)$-answer string $a_{k+1} \ldots a_i * \ldots *$ is in F^a, and so also in F^b. Therefore $ba_{k+1} \ldots a_i * \ldots * \in F$, which implies that $\Phi(u)$ is flagged.

By symmetry, we obtain that Requirement (b) in Definition 4.11 holds for Φ as well. Therefore, vertices v and w are equivalent in T. □

4.2.3. Constructing a fully reduced FS-decision digraph. Theorem 4.18 will now be used to construct a fully reduced FS-decision digraph without prior construction of the FS-decision tree, thereby saving time and space.

This FS-decision digraph will be constructed in a BFS order, analogously to the construction of the FS-decision tree T in Algorithm 4.10. When a vertex u is processed, its out-neighbours (children in T) are examined in order. If a potential child c is equivalent to an earlier vertex, say w, then c is absorbed into w, the answer string for c is adjoined to the set of answer strings for w, and w is appended to the list of out-neighbours of vertex u. Otherwise, c is created as a new vertex and is placed in the queue to be processed later.

This algorithm is detailed below as Algorithm 4.19. Note that procedure flagged can be found in Algorithm 4.10.

Algorithm 4.19. Constructing a fully reduced FS-decision digraph of an abstract questionnaire

procedure FS-digraph$(N, \mathcal{M}, F, \mathcal{S})$

 # Input: abstract questionnaire $(N, \mathcal{M})$ with flag-set F, compatible with the skip-list $\mathcal{S}$.

 # Output: a fully reduced FS-decision digraph D for $(N, \mathcal{M})$ with the subset of skipped vertices U, question assignment κ, answer string assignment $\mathcal{A}$, and flag function φ.

 # The vertices of D are labelled $0, 1, 2, \ldots$.

 # $Out(u)$ is the (ordered) list of out-neighbours of vertex u.

$\kappa(0) := 0$

$\mathcal{A}(0) := \{\epsilon\}$

$\varphi(0) := 0$

$L := [0]$ *# BFS queue of unprocessed vertices*

$c := 0$ *# last vertex label used*

for all $k \in \mathbb{Z}_{N+1}$ **do** first$(k) := -1$ *# the label of the first vertex with question k*

first$(0) := 0$

while $L \neq [\,]$ **do**
$u :=$ first vertex in L
remove u from L
$q := \kappa(u)$
$\mathrm{Out}(u) := [\,]$
if $q < N$ **then**
if $u \in U$ **then** $A = \{*\}$
else $A = \mathbb{Z}_{m_q}$
for all $a_q \in A$ **do**
$c := c + 1$ # *create a potential new vertex*
new:= True
$\kappa(c) = q + 1$
$\mathcal{A}(c) := \{ba_q : b \in \mathcal{A}(u)\}$ # *the set of answer strings for vertex* c
if $\mathrm{first}(q+1) \neq -1$
then
if $\exists w \in \{\mathrm{first}(q+1), \ldots, c-1\}$ such that $\mathrm{equiv}(c, w) = 1$
then
merge vertex c *with vertex* w
new:= False
$\mathcal{A}(w) := \mathcal{A}(w) \cup \mathcal{A}(c)$
$c := c - 1$
$\mathrm{Out}(u) :=$Out$(u) + [w]$
else $\mathrm{first}(q+1) := c$
if new **then**
$a :=$ the first element of $\mathcal{A}(c)$
if $a \in S_{q+1}$ **then** $U := U \cup \{c\}$ # *c is a skipped vertex*
if $\mathrm{flagged}(a, F) = 1$ **then** $\varphi(c) := 1$
else $\varphi(c) := 0$
$\mathrm{Out}(u) :=$Out$(u) + [c]$
$L := L + [c]$
return c, Out, U, κ, $\mathcal{A}$, φ

procedure $\mathrm{equiv}(v, w)$
Input: vertices v *and* w *of an FS-decision tree* T.
The abstract questionnaire $(N, \mathcal{M})$, *its skip-list* $\mathcal{S}$, *and a compatible flag-set* F *are considered global variables.*
Output: 1 if v *and* w *are equivalent in* T, *and 0 otherwise.*
Ans:= 0
if $\kappa(v) = \kappa(w)$
then
$k = \kappa(v) - 1$
$a := \alpha(v)$

$b := \alpha(w)$
$t :=$ the $(k+1, N-1)$-answer string $* \ldots *$
if flagged$(a, F) = 1$ **then** $F^a := \{t\}$
else $F^a := \{d \in A_{k+1}^* \times \ldots \times A_{N-1}^* : ad \in F\}$
if flagged$(b, F) = 1$ **then** $F^b := \{t\}$
else $F^b := \{d \in A_{k+1}^* \times \ldots \times A_{N-1}^* : bd \in F\}$
if $F^a = F^b$
then
  if $a \in S_{k+1}$ **then** $S_{k+1}^a := \{\epsilon\}$
  else $S_{k+1}^a := \{\ \}$
  if $b \in S_{k+1}$ **then** $S_{k+1}^b := \{\epsilon\}$
  else $S_{k+1}^b := \{\ \}$
  $i := k+1$
  while $S_i^a = S_i^b$ **and** $i < N-1$ **do**
    $i := i+1$
    $S_i^a := \{d \in A_{k+1}^* \times \ldots \times A_{i-1}^* : ad \in S_i\}$
    $S_i^b := \{d \in A_{k+1}^* \times \ldots \times A_{i-1}^* : bd \in S_i\}$
  if $S_i^a = S_i^b$ **then** Ans:= 1
return Ans

4.3. Generating a skip-list and a compatible flag-set

In this section, we explain how a questionnaire designer can generate a skip-list and a compatible flag-set for a given questionnaire, starting from an intuitively constructed "pre-skip-list" and "pre-flag-set". First, we need a concept of a generalized answer string (Definition 4.21 below).

As before, we assume that we have an abstract questionnaire $\mathcal{Q} = (N, \mathcal{M})$, and that $A_i = \mathbb{Z}_{m_i}$ is the set of possible answers to question i. In addition, for all $i \in \mathbb{Z}_N$, we define $A_i^\diamond = A_i \cup \{\diamond\}$. We think of the symbol $\diamond$ as representing "anything". (Note that $\diamond$ has a similar meaning as $*$, but without the restrictions imposed by Definitions 4.4 and 4.8.)

Definition 4.20. Let k and ℓ be integers, $0 \le k \le \ell \le N-1$. A *generalized* (k, ℓ)*-answer string* for an abstract questionnaire $\mathcal{Q} = (N, \mathcal{M})$ is an element of $A_k^\diamond \times A_{k+1}^\diamond \times \ldots \times A_\ell^\diamond$; that is, a string of the form $a_k a_{k+1} \ldots a_\ell$, where $a_i \in A_i^\diamond$ for all $i = k, k+1, \ldots, \ell$.

A generalized (k, ℓ)-answer string $a_k a_{k+1} \ldots a_{i-1} \diamond a_{i+1} \ldots a_\ell$ can be thought of as representing all (k, ℓ)-answer strings of the form $a_k a_{k+1} \ldots a_{i-1} x a_{i+1} \ldots a_\ell$, for $x \in A_i$.

We next explain what we mean by a pre-skip-list.

Definition 4.21. A *pre-skip-list* for the abstract questionnaire $\mathcal{Q} = (N, \mathcal{M})$ is an $(N+1)$-tuple $\mathcal{P} = (P_0, P_1, \ldots, P_N)$ such that the following hold.

(i) $P_0 = \emptyset = P_N$.

(ii) For each q, $0 \le q \le N$, we have that P_q is a set of generalized $(0, q-1)$-answer strings.

For each q, we think of the set P_q as containing all $(0, q-1)$-answer strings that necessitate the skipping of question q. However, to simplify the work of the questionnaire designer, these answer strings can be provided in the form of generalized answer strings, whereby only the relevant answers are specified, and all irrelevant answers are replaced by the symbol $\diamond$. For example, if we have an abstract questionnaire $\mathcal{Q} = (N, \mathcal{M})$ with $N = 5$ and $\mathcal{M} = (2, 2, 3, 2, 3)$, and we wish question 4 to be skipped when the answers to questions 0 and 2 are 1 and 0, respectively, then we set $P_4 = \{1\diamond 0\diamond\}$, which represents the set of answer strings $\{1a_1 0a_3 : a_i \in A_i, i = 1, 3\}$.

We next describe a procedure that converts a pre-skip-list to a skip-list that provides equivalent information but satisfies the requirements of Definition 4.4, so it can be used in our main algorithms. Intuitively speaking, for each $q = 0, \ldots, N$, each generalized answer string in S_q must be expanded into a corresponding set of answer strings that also satisfies Condition (iii) in Definition 4.4; that is, this set must contain only those corresponding answer strings that represent actual vertices of the FS-decision tree.

Algorithm 4.22. Constructing a skip-list of an abstract questionnaire from a pre-skip-list

procedure skip-list$(N, \mathcal{M}, \mathcal{P})$
Input: abstract questionnaire $(N, \mathcal{M})$ *with a pre-skip-list* $\mathcal{P}$.
Output: skip-list $\mathcal{S}$.
    $S_0 := \emptyset$
    $S_N := \emptyset$
    $S_1 := P_1$ # *we may assume* $\diamond \notin P_1$
    for $q := 2$ **to** $N - 1$
        $S_q := \emptyset$
        for all $p_0 \ldots p_{q-1} \in P_q$ **do**
            if $p_0 = \diamond$ **then** $S := A_0$
            else $S := \{p_0\}$ # S *contains answer strings arising from the generalized answer string* $p_0 \ldots p_{q-1}$*; at step* i*, it contains* $(0, i-1)$*-answer strings*
            for $i := 1$ **to** $q - 1$ **do**
                if $p_i = \diamond$ **then**
                    $S' := \{ts_i : t \in S - S_i, s_i \in A_i\} \cup \{t* : t \in S \cap S_i\}$
                else $S' := \{tp_i : t \in S - S_i\}$
                $S := S'$
            $S_q := S_q \cup S$
    return $\mathcal{S} = (S_0, \ldots, S_N)$

Next, we turn our attention to flag-sets. First we need the concept of a pre-flag-set, which we define as follows.

Definition 4.23. A *pre-flag-set* for the abstract questionnaire $\mathcal{Q} = (N, \mathcal{M})$ is a set of generalized $(0, N-1)$-answer strings, usually denoted by P.

We think of a pre-flag-set P as containing every $(0, N-1)$-answer string that is contradictory or of special interest to the questionnaire designer. Again, for simplicity, the answer strings in P are given in the form of generalized answer strings. To illustrate, consider again the example where $\mathcal{Q} = (N, \mathcal{M})$ with $N = 5$ and $\mathcal{M} = (2, 2, 3, 2, 3)$. Say that, if chosen together in a response, the answers 1 and 2 to questions 0 and 2, respectively, and the answers 0 and 1 to questions 3 and 4, respectively, are of special interest. Then our pre-flag-set is $P = \{1\diamond 2\diamond\diamond, \diamond\diamond\diamond 01\}$, and it represents the set of answer strings $\{1a_1 2a_3 a_4 : a_i \in A_i, i = 1, 3, 4\} \cup \{a_0 a_1 a_2 01 : a_i \in A_i, i = 0, 1, 2\}$.

Given a skip-list $\mathcal{S}$ and pre-flag-set P, Algorithm 4.24 below creates a flag-set F that is compatible with $\mathcal{S}$, and therefore can be used in our main algorithms. The idea is to expand each generalized answer string in P into a set of answer strings containing equivalent information while also satisfying the conditions of compatibility in Definition 4.8.

Algorithm 4.24. Constructing a compatible flag-set of an abstract questionnaire from a pre-flag-set

procedure flag-set$(N, \mathcal{M}, \mathcal{S}, P)$
Input: abstract questionnaire $(N, \mathcal{M})$ *with skip-list* $\mathcal{S}$ *and pre-flag-set* P.
Output: flag-set F.
    $F := \emptyset$
    for all $p_0 \dots p_{N-1} \in P$ **do**
        if $p_0 = \diamond$ **then** $G := A_0$
        else $G := \{p_0\}$ # *G contains answer strings arising from the generalized answer string $p_0 \dots p_{N-1}$; at step i, it contains $(0, i-1)$-answer strings*
        for all $g \in G$ **do**
            if is-contained$(0, g, F)$ **then** $G := G - \{g\}$
        for $i := 1$ **to** $N - 1$ **do**
            if $p_i \dots p_{N-1} = \diamond \dots \diamond$
            then
                $G' := \{g * \dots * : g \in G\}$ # *G' is a set of $(0, N-1)$-answer strings*
                for all $g \in G'$ **do**
                    if is-contained$(N-1, g, F)$ **then** $G' := G' - \{g\}$
                $G := G'$
                break # *exit the for loop*
            else
                if $p_i = \diamond$ **then**
                    $G' := \{ga_i : g \in G - S_i, a_i \in A_i\} \cup \{g* : g \in G \cap S_i\}$
                else $G' := \{gp_i : g \in G - S_i\}$
                for all $g \in G'$ **do**
                    if is-contained(i, g, F) **then** $G' := G' - \{g\}$
                $G := G'$
        for all $g \in G$ **do**
            for all $f \in F$ **do**

if replaces(g, f) **then** $F := F - \{f\}$
$F := F \cup G$
return F

procedure is-contained(k, g, F)
Input: $(0, k)$*-answer string* g *and set* F *of* $(0, N-1)$*-answer strings.*
Output: True if and only if the $(0, N-1)$*-answer string* $g*\ldots*$ *is in* F*.*
for $i := k+1$ **to** $N-1$ **do** $g := g*$
if $g \in F$ **then return** True
else return False

procedure replaces(g, f)
Input: $(0, N-1)$*-answer strings* $g = g_0 \ldots g_{N-1}$ *and* $f = f_0 \ldots f_{N-1}$*.*
Output: True iff $g = g_0 \ldots g_k * \ldots *$ *and* $f = g_0 \ldots g_\ell * \ldots *$ *with* $k < \ell$*.*
$i := 0$
while $g_i = f_i$ and $i < N$ **do** $i := i+1$
if $i < N$ and $g_i \ldots g_{N-1} = * \ldots *$ **then return** True
else return False

5. Example

In this section, we give an example of a simple concrete questionnaire, and use it to illustrate the concepts and algorithms presented in this paper.

Consider the following short questionnaire directed at university students taking a certain course.

0. For your program, this course is
   0) compulsory
   1) not required
1. Which of the following best describes your reason for taking this course?
   0) It seemed interesting
   1) I wanted to challenge myself
   2) I wanted to boost my GPA
2. What percentage of classes did you attend?
   0) 0-25%
   1) 26-50%
   2) 51-75%
   3) 76-100%
3. I think the professor conveys the subject matter effectively.
   0) agree
   1) disagree
4. This course is challenging.
   0) agree
   1) disagree

Note that the corresponding abstract questionnaire has $N = 5$ and $\mathcal{M} = (2, 3, 4, 2, 2)$.

First, we shall illustrate the results of Section 3. Suppose that we insist that questions 1 and 2 be asked after question 0, and that questions 4 and 3 be asked after questions 1 and 2, respectively. This gives us the precedence relation $R = \{(0, 1), (0, 2), (1, 4), (2, 3)\}$. Note that R is an irreflexive binary relation with no directed cycles. By Lemma 3.1, there exists a total order extending R, and we may apply Algorithm 3.2. The resulting output is

$$\mathcal{T} = [\ [0, 1, 2, 3, 4], [0, 1, 2, 4, 3], [0, 1, 4, 2, 3], [0, 2, 1, 3, 4], [0, 2, 1, 4, 3], [0, 2, 3, 1, 4]\ ];$$

that is, $\mathcal{T}$ is a list of all total orders on $\mathbb{Z}_5$ that extend the relation R. Note that for the remainder of this section, we will be using the default question order $[0, 1, 2, 3, 4]$.

Next, we would like to represent the flow of our questionnaire with a graph. To illustrate the work of Section 4, suppose that we would like to impose the following restrictions on our questionnaire: if a respondent answers with 0 to question 0, they should skip question 1 (no need to ask why they took the course since it was compulsory for them), and if they answer with 0 to question 2, they should skip question 3 (since they attended very few classes, their opinion on the professor is not very relevant). Thus our pre-skip-list is $\mathcal{P} = (P_0, P_1, \ldots, P_5)$, where $P_1 = \{0\}$, $P_3 = \{\diamond\diamond 0\}$, and $P_0 = P_2 = P_4 = P_5 = \emptyset$. Applying Algorithm 4.22 gives us the skip list $\mathcal{S} = (S_0, S_1, \ldots, S_5)$, where $S_1 = \{0\}$, $S_3 = \{0 * 0, 100, 110, 120\}$, and $S_0 = S_2 = S_4 = S_5 = \emptyset$.

We would also like to flag certain answer strings. It is questionable for someone to take the course out of interest, then not attend any classes, so we add $\diamond 00\diamond\diamond$ to the pre-flag-set P. We would also like to know more about why someone who wanted to challenge themselves with this course did not find the course challenging; hence, we let $\diamond 1\diamond\diamond 1 \in P$. Lastly, we would like to keep track of those who are required to take the course yet who have attended very few classes, so we add $0\diamond 0\diamond\diamond$ to the pre-flag-set as well. Thus, we have a pre-flag-set $P = \{\diamond 00\diamond\diamond, \diamond 1\diamond\diamond 1, 0\diamond 0\diamond\diamond\}$. To create a flag-set F that is compatible with the skip-list $\mathcal{S}$, we expand each of the answer strings as outlined in Algorithm 4.24. We thus obtain

$$F = \{100 * *, 110 * 1, 11101, 11111, 11201, 11211, 11301, 11311, 0 * 0 * *\}.$$

Finally, we apply Algorithm 4.10 to obtain the FS-decision tree for $\mathcal{Q}$, and Algorithm 4.19 to obtain the fully reduced FS-decision digraph for $\mathcal{Q}$ (see Figure 5).

6. Conclusion

In this paper, we introduced FS-decision trees and, more importantly, FS-decision digraphs as new models for abstract questionnaires endowed with the additional information in the form of a skip-list and a flag-set. We presented algorithms for constructing both models, as well as for generating suitable input data (a skip-list and a compatible flag-set) from the more intuitive pre-skip-list and pre-flag-set. We also described how to construct all possible orderings of the questions based solely on an abstract precedence relation. Our hope is that our models will help questionnaire designers visualize and automatize their work.

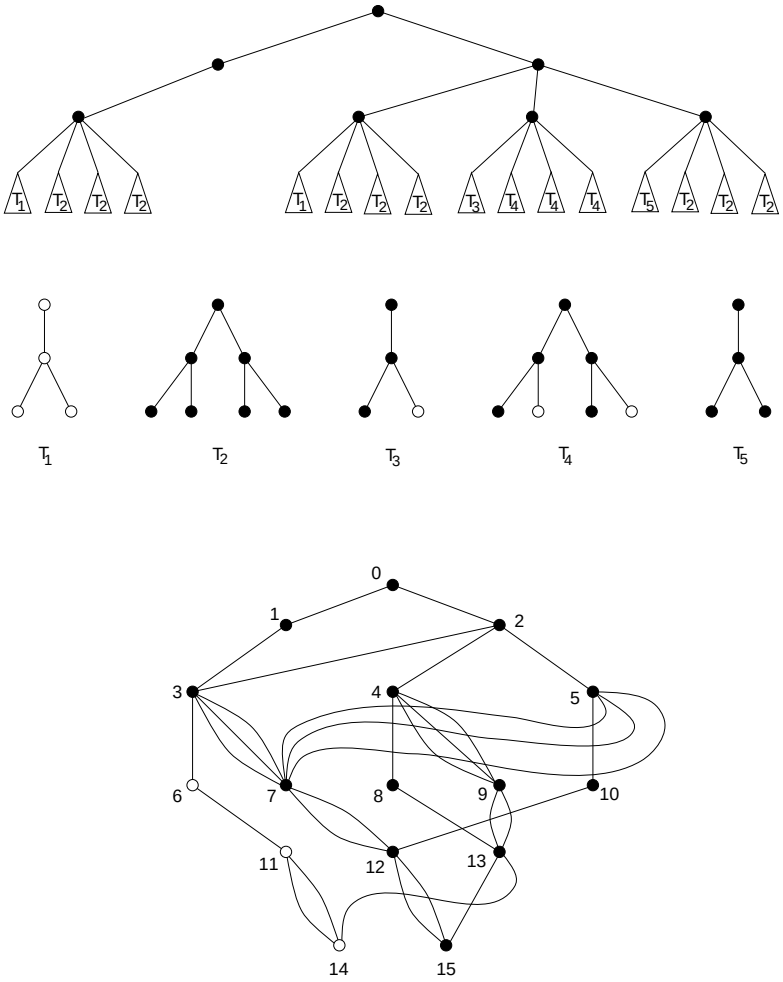

Fig. 5. The FS-decision tree (top) and the fully reduced FS-decision digraph (bottom) for the example in Section 5. All edges are directed downwards, out-neighbours are ordered from left to right, and flagged and unflagged vertices are coloured white and black, respectively. Vertices in the FS-decision digraph are labelled in the order created by Algorithm 4.19.

References

[1] J. Bethlehem and A. Hundepool. TADEQ: a tool for the documentation and analysis of electronic questionnaires. *Journal of Official Statistics*, 20:233–264, 2004.

[2] S. Elliott. The application of graph theory to the development and testing of survey instruments. *Survey Methodology*, 38:11–21, 2012.

[3] G. Feeney and S. Feeney. On the logical structure of census and survey questionnaires. *Genus*, 75:Article 19, 2019. https://doi.org/10.1186/s41118-019-0065-y.

[4] L. Fenn. Decision Trees and Surveys. Technical report, Hunter College, CUNY, 2015. http://larryfenn.com/decisiontrees.pdf.

[5] T. B. Jabine. Flow charts: a tool for developing and understanding survey questionnaires. *Journal of Official Statistics*, 1:189–207, 1985.

[6] P. P. Parkhomenko. Questionnaires and organizational hierarchies. *Automation and Remote Control*, 71:124–134, 2010. https://doi.org/10.1134/S0005117910060135.

[7] C. Picard. *Th'eorie des Questionnaires*, number 20 in Les Grands Probl'emes des Sciences. Gauthier-Villars, Paris, 1965.

[8] I. Şchiopu-Kratina, C. M. Zamfirescu, K. Tr'epanier, and L. Marques. Survey questionnaires and graphs. *Electronic Journal of Statistics*, 9:2202–2254, 2015. https://doi.org/10.1214/15-EJS1067.

[9] K. Stark and S. Zinn. Using Mathematical Graphs for Questionnaire Testing in Large-Scale Surveys. Technical report 1135, DIW Berlin, 2021. https://www.diw.de/documents/publikationen/73/diw_01.c.818369.de/diw_sp1135.pdf.

[10] G. Xing and Q. Sun. Greedy topological sorting on questionnaire directed acyclic graph and its application in patient surveys. In *Proceedings of the 6th ACM Conference on Bioinformatics, Computational Biology and Health Informatics*, pages 549–550, 2015. https://doi.org/10.1145/2808719.2811459.

Jiaye Chen
Department of Mathematics and Statistics, University of Ottawa
150 Louis-Pasteur Private, Ottawa, ON, K1N 6N5, Canada
E-mail jevchen1994@gmail.com

Suzan Kadri
Department of Mathematics and Statistics, University of Ottawa
150 Louis-Pasteur Private, Ottawa, ON, K1N 6N5, Canada
E-mail skadr037@uottawa.ca

Mateja Šajna
Department of Mathematics and Statistics, University of Ottawa
150 Louis-Pasteur Private, Ottawa, ON, K1N 6N5, Canada
E-mail msajna@uottawa.ca

Ioana Schiopu-Kratina
University of Ottawa, 150 Louis-Pasteur Private, Ottawa, ON, K1N 6N5, Canada
E-mail ioanakratina@gmail.com