Extending choice assessments to choice functions: An algorithm for computing the natural extension

Arne Decadt*, Alexander Erreygers, Jasper De Bock*

Ghent University, Belgium

Abstract

We study how to infer new choices from prior choices using the framework of choice functions, a unifying mathematical framework for decision-making based on sets of preference orders. In particular, we define the natural (most conservative) extension of a given choice assessment to a coherent choice function—whenever possible—and use this natural extension to make new choices. We provide a practical algorithm for computing this natural extension and various ways to improve scalability. Finally, we test these algorithms for different types of choice assessments.

Keywords: Choice function, Natural extension, Algorithm.

1. Introduction

In classical probability theory, decisions are typically made by maximising expected utility. This leads to a single optimal decision, or a set of optimal decisions all of which are equivalent. In the theory of imprecise probabilities, where multiple probabilistic models are considered simultaneously, this decision rule can be generalised in multiple ways; Troffaes [1] provides a nice overview. A typical feature of the resulting decision rules is that they will not always yield a single optimal decision, as a decision that is optimal in one probability model may for example be suboptimal in another.

We here take this generalisation yet another step further by adopting the theory of choice functions: a mathematical framework for decision-making that incorporates several (imprecise) decision rules as special cases, including the classical approach of maximising expected utility [2, 3, 4]. An important feature of this framework of choice functions is that it allows one to impose axioms directly on the decisions that are represented by such a choice function [3, 4, 5]. We here adopt the coherence axioms that were put forward by De Bock and De Cooman [3]. We do not use these axioms directly though, but instead consider an alternative definition of coherence that is based on sets of preference orders and show that it is equivalent.

As we will explain and demonstrate in this contribution, we can use these coherent choice function to infer new choices from previous choices. In particular, for any given assessment of previous choices that is compatible with coherence, we will achieve this by introducing the so-called natural extension of this assessment: the unique most conservative coherent choice function that is compatible with the assessment.

We start in Section 2 with an introduction to choice functions and coherence. Section 3 then first defines choice assessments, their consistency and their natural extension, and then goes on to reformulate these concepts in terms of coherent sets of desirable options. Next, in Section 4, we show how this reduces the problems of checking the consistency of an assessment and computing its natural extension to something that we can solve practically and algorithmically. The running time of the algorithm depends rather heavily on the size of the assessment that is provided though. To reduce this running time, Sections 5 and 6 present several methods that can be used to replace an assessment by an equivalent object that contains the same

*Corresponding authors

Email addresses: arne.decadt@ugent.be (Arne Decadt), jasper.debock@ugent.be (Jasper De Bock)

information, but can be more efficiently used to check consistency and calculate the natural extension. In Section 7, we test our algorithms and examine how the size and imprecision of the assessment impact the time required to determine the natural extension. Section 8 concludes the paper and provides some suggestions for future work.

2. Choice functions

A choice function C is a function that, when applied to a set of options, may reject one or more—but not all—options from that set. The options that are not rejected are then said to be 'chosen'. Usually the options are actions that have a corresponding reward. This reward furthermore depends on the state of an unknown—uncertain—variable X that takes values in a set $\mathcal{X}$. We will assume that the rewards can be represented by real numbers, on a real-valued utility scale. In this setting, an option u is thus a function from states x in $\mathcal{X}$ to $\mathbb{R}$. We will denote the set of all possible options by $\mathscr{V} \subseteq \mathbb{R}^{\mathcal{X}}$ and require that this forms a real vector space with pointwise vector addition and scalar multiplication. Bounded options are sometimes also called gambles in the literature. Moreover, we endow $\mathscr{V}$ with the partial order $\leq$: for all $u, v \in \mathscr{V}$, $u \leq v$ if and only if $u(x) \leq v(x)$ for all $x \in \mathcal{X}$; $<$ is the corresponding strict version, so $u < v$ if $u \leq v$ and $u \neq v$.[1] We also let $2^{\mathscr{V}}$ denote the power set of $\mathscr{V}$. To make all of this more tangible, we consider the following toy problem as a running example.

Running Example 2.1. A farming company cultivates tomatoes and they have obtained a large order from a foreign client. However, due to government regulations they are not sure whether they can deliver this order. So the state space $\mathcal{X}$ is {order can be delivered, order cannot be delivered}. The company now has multiple options to distribute their workforce. They can fully prepare the order, partially prepare the order or not prepare the order at all. Since $\mathcal{X}$ only has two elements, we can identify the options with vectors in $\mathbb{R}^2$. We will let the first component of these vectors correspond to the reward if the order can be delivered. For example, the option of fully preparing the order could correspond to the vector $v_1 := (5, -3)$: if the order goes through, then the company receives a payment—or utility—of 5 for that order; however, if the order does not go through, the company "receives" a negative reward -3, reflecting the large amount of resources that they spent on an order that could not be delivered in the end. ◊

Finiteness of sets will be important throughout this paper, and we therefore introduce the symbol $\Subset$ to mean that a set is a non-empty finite subset. A first example is that we will restrict ourselves to choices from finite sets of options. That is, the domain of our choice functions will be $\mathscr{Q} := \{A \colon A \Subset \mathscr{V}\}$, the set of all finite subsets of $\mathscr{V}$ excluding the empty set. We also let $\mathscr{Q}_\emptyset := \mathscr{Q} \cup \{\emptyset\}$. Formally, a *choice function* is then any function $C \colon \mathscr{Q} \to \mathscr{Q}$ such that $C(A) \subseteq A$ for all $A \in \mathscr{Q}$. We will also consider the corresponding rejection function $R_C \colon \mathscr{Q} \to \mathscr{Q}_\emptyset \colon A \mapsto A \setminus C(A)$.

We will give the following interpretation to these choice functions. For every set $A \in \mathscr{Q}$ and option $u \in A$, we take $u \in C(A)$—u is 'chosen'—to mean that there is no other option in A that is preferred to u. Equivalently, $u \in R_C(A)$—u is rejected from A—if there is an option in A that is preferred to u.

Running Example 2.2. We will let the choice function C correspond to choices that the strategic advisor of the company makes or would make for a given set of options, where these choices can be multivalued whenever he does not single out a unique best option. Suppose for example that he has rejected v_3 and v_4 from a set $A_1 := \{v_1, v_2, v_3, v_4\}$, with $v_1 := (5, -3)$, $v_2 := (3, -2)$, $v_3 := (1, -1)$, and $v_4 := (-2, 1)$, but remains undecided about whether to choose v_1 or v_2. This corresponds to the statement $C(A_1) = \{v_1, v_2\}$, or equivalently, $R_C(A_1) = \{v_3, v_4\}$. ◊

De Bock and De Cooman [3] define coherent choice functions by imposing properties for rationality on the corresponding rejection functions. An example of such a property is their axiom R_4, which is analogous

[1] In principle, our results in Sections 2, 3, 5 and 6 (except Section 2.1) can also be adapted to work for any ordered vector space over the real numbers, i.e. a vector space $\mathcal{W}$ and a partial order $\lhd$ on $\mathcal{W}$ for which for any vectors $u, v, w \in \mathcal{W}$ and real number $r > 0$, $u \lhd v$ implies $u + w \lhd v + w$ and $ru \lhd rv$; we restrict ourselves to $(\mathscr{V}, <)$ for didactic purposes.

to Sen's Alpha [6, 7]: if some option is rejected from a set, then it will also be rejected from any of its supersets. We here opt to not define coherent choice functions in terms of such properties though, but to instead use an equivalent characterisation in terms of sets of preference orders.

In particular, these preference orders are taken to be strict partial vector orders on $\mathscr{V}$, typically denoted by $\prec$, that extend the original strict order $<$. This means that they should have the following properties [3, $\triangleright_0 - \triangleright_4$]: for all $u, v, w \in \mathscr{V}$ and $\lambda > 0$,

$\prec_0$. $u \not\prec u$, (irreflexivity)

$\prec_1$. if $u \prec v$ and $v \prec w$ then also $u \prec w$, (transitivity)

$\prec_2$. if $u \prec v$ then also $u + w \prec v + w$, (translation invariance)

$\prec_3$. if $u \prec v$ then also $\lambda u \prec \lambda v$, (scaling invariance)

$\prec_4$. if $u < v$ then $u \prec v$. (extends $<$)

We call orderings that satisfy Axioms $\prec_0$ to $\prec_4$ *(coherent) preference orders* and let $\mathbb{O}$ denote the set of all preference orders on $\mathscr{V}$. Note that thanks to Axiom $\prec_2$ preference orders are fully determined by their set of desirable options $G_\prec := \{u \in \mathscr{V} : 0 \prec u\}$, where desirable means preferred to the status quo 0. These sets of desirable options will be important later on and are useful because they reduce the 'pairs' of options necessary to describe a partial vector order to just 'single' options. We will call a set of desirable options $G \subseteq \mathscr{V}$ *coherent* if it satisfies the following properties: for all $u, v \in G$, $w \in \mathscr{V}$ and $\lambda > 0$,

d_0. $0 \notin G$, (irreflexivity)

d_1. $u + v \in G$, (additivity)

d_2. $\lambda u \in G$, (scaling invariance)

d_3. if $0 < w$ then $w \in G$. (extends $<$)

We furthermore use $\overline{\mathbf{G}}$ to denote the set of all coherent sets of desirable options. As our next result shows, if $\prec$ is a preference order, then $G_\prec$ is coherent. Furthermore, for any set of options $G \subseteq \mathscr{V}$, if we define the binary relation $\prec_G$, for all $u, v \in \mathscr{V}$ by

$$u \prec_G v \Leftrightarrow v - u \in G,$$

then the coherence of G implies that $\prec_G$ is a preference order. Working with coherent sets of desirable options, or with preference orders, is therefore equivalent.

Lemma 2.3. For any preference order $\prec$, $G_\prec$ is a coherent set of desirable options. Moreover, the map $\prec \mapsto G_\prec$ is a bijection between the set of preference orders and the set of coherent sets of desirable options, with inverse $G \mapsto \prec_G$.

Proof. First we prove that $G_\prec$ is coherent for any preference order $\prec$. Take any preference order $\prec$. Then by Axiom $\prec_0$ we have $0 \not\prec 0$ which implies Axiom d_0. If $0 \prec u$ and $0 \prec v$ by Axiom $\prec_2$ we have $0 \prec u \prec u + v$, so by Axiom $\prec_1$ we have $0 \prec u + v$, which implies Axiom d_1. Similarly, Axiom d_2 follows from Axiom $\prec_3$ and Axiom d_3 from Axiom $\prec_4$.

Next we prove that the map $\prec \mapsto G_\prec$ is a bijection. First we prove that this map is injective. Assume that we have two preference orders $\prec$ and $\prec'$ such that $G_\prec = G_{\prec'}$. Then for all $u, v \in \mathscr{V}$, we have $v - u \in G_\prec$ if and only if $v - u \in G_{\prec'}$. So, $0 \prec v - u$ if and only if $0 \prec' v - u$ and therefore, by Axiom $\prec_2$, $u \prec v$ if and only if $u \prec' v$ for all $u, v \in \mathscr{V}$, or equivalently, $\prec = \prec'$.

To prove that the map $\prec \mapsto G_\prec$ is surjective and has inverse $G \mapsto \prec_G$, assume that we have some coherent $G \subseteq \mathscr{V}$ and consider $\prec_G$. Then Axiom $\prec_2$ is satisfied trivially for $\prec_G$ and Axioms $\prec_0$, $\prec_3$ and $\prec_4$ follow immediately from respectively Axioms d_0, d_2 and d_3. For Axiom $\prec_1$, assume that $u \prec_G v$ and $v \prec_G w$. Then $v - u \in G$ and $w - v \in G$, so $w - u = (w - v) + (v - u) \in G$ by Axiom d_1, hence $u \prec_G w$. So we conclude that $\prec_G$ is a preference order. Furthermore, for all $u \in \mathscr{V}$, we have by definition that $u \in G$ if and only if $0 \prec_G u$, so by definition $G = G_{\prec_G}$, which shows that $G \mapsto \prec_G$ is indeed the inverse of $\prec \mapsto G_\prec$. □

For any preference order $\prec \in \mathbb{O}$ on $\mathscr{V}$, inspired by our interpretation for choice functions, we consider the corresponding choice function $C_\prec$ defined by

$$C_\prec(A) \coloneqq \{\, u \in A : (\forall a \in A) u \not\prec a \,\} \text{ for all } A \in \mathscr{Q}.$$

The corresponding rejection function $R_\prec$ [3, Equation (1)] is then given by

$$R_\prec(A) \coloneqq \{\, u \in A : (\exists a \in A) u \prec a \,\} \text{ for all } A \in \mathscr{Q}.$$

Crucially, however, the preference order $\prec$ need not be known. Instead, in its full generality, our definition will allow for the use of a set of preference orders, only one of which is the true order $\prec$. Any such set of preference orders $\mathcal{O} \subseteq \mathbb{O}$ corresponds to a function $C_\mathcal{O} \colon \mathscr{Q} \to \mathscr{Q}_\emptyset$ defined for all $A \in \mathscr{Q}$ by

$$C_\mathcal{O}(A) \coloneqq \{\, u \in A : (\exists \prec \in \mathcal{O})(\forall a \in A) u \not\prec a \,\} = \bigcup_{\prec \in \mathcal{O}} C_\prec(A); \tag{1}$$

it represents the choices—or rather, the rejections—that are compatible with each of the orders $\prec \in \mathcal{O}$. Whenever $C_\mathcal{O}$ is a choice function, the corresponding rejection function $R_\mathcal{O}$ is given by

$$R_\mathcal{O}(A) \coloneqq A \setminus C_\mathcal{O}(A) = \{\, u \in A : (\forall \prec \in \mathcal{O})(\exists a \in A) u \prec a \,\} \text{ for all } A \in \mathscr{Q}. \tag{2}$$

By definition, we already have that $C_\mathcal{O}(A) \subseteq A$ for all $A \in \mathscr{Q}$. For $C_\mathcal{O}$ to be a choice function—a map from $\mathscr{Q}$ to $\mathscr{Q}$—we also need that $C_\mathcal{O}(A) \neq \emptyset$ for all $A \in \mathscr{Q}$. This will be the case whenever $\mathcal{O}$ is not empty, as implied by the following proposition

Proposition 2.4. Fix some (possibly empty) set of preference orders $\mathcal{O} \subseteq \mathbb{O}$. Then for any $A \in \mathscr{Q}$, $C_\mathcal{O}(A) = \emptyset$ if and only if $\mathcal{O} = \emptyset$.

We will prove this using the following well-known lemma, which will come in handy in our proof of Lemma 3.5 further on as well.

Lemma 2.5 (variation of [8, I.3 Theorem 3])**.** Consider any strict preorder $\prec$, so an irreflexive ($\prec_0$) and transitive ($\prec_1$) binary relation, on a finite set A. Then for any element $a \in A$ there is an element $a^* \in A$, with $a \prec a^*$ or $a = a^*$, such that a^* is undominated in A, meaning that there is no $b \in A$ for which $a^* \prec b$.

Proof. Let the elements of A be $a_1, ..., a_n$ and assume without loss of generality that $a = a_1 =: m_1$. Now for every $k \in \{2, ..., n\}$ define m_k as a_ℓ for any $\ell \in \{1, ..., n\}$ for which $m_{k-1} \prec a_\ell$ if such ℓ exists and m_{k-1} otherwise. Let $a^* \coloneqq m_n$. Then $a = a^*$ or $a \prec a^*$, where the latter follows from transitivity. If $a = a^*$, then a is clearly undominated. If $a \prec a^*$, then a^* is undominated because if it were not, then there would be some $m_{n+1} \in A$ such that $m_1 \prec ... \prec m_{n+1}$. By the pigeonhole principle two of them would have to be equal, say $m_k = m_\ell$ with $k < \ell$, and by transitivity $m_k \prec m_\ell$, violating irreflexivity. □

Proof of Proposition 2.4 The reverse implication follows from Equation (1). We prove the direct implication by contraposition. Let $\mathcal{O} \neq \emptyset$ and fix any $\prec \in \mathcal{O}$. Take any $A \in \mathscr{Q}$ and $u \in A$. Then by Lemma 2.5 there is an undominated $u^* \in A$. But then $u^* \in C_\mathcal{O}(A) \neq \emptyset$ by definition. □

This allows us to define coherence for choice functions as follows.

Definition 2.6. We call a choice function $C \colon \mathscr{Q} \to \mathscr{Q}$ *coherent* if there is some non-empty set of preference orders $\mathcal{O} \subseteq \mathbb{O}$ such that $C = C_\mathcal{O}$.

Alternatively, as explained earlier, and as we formally prove in Appendix A, this definition of coherence can also be equivalently characterised using five axioms [3, $\mathrm{R}_0 - \mathrm{R}_4$] for rejection functions, but our characterisation in terms of preference orders is more convenient in the present setting.

We note also that the set of preference orders $\mathcal{O}$ in Definition 2.6 need not be unique. It follows from Equation (1) that the largest set of preference orders $\mathcal{O}$ for which a coherent choice function C is equal to $C_\mathcal{O}$ is

$$\mathcal{O}_C = \{\prec \in \mathbb{O} \colon (\forall A \in \mathscr{Q}) C_\prec(A) \subseteq C(A)\} = \bigcap_{A \in \mathscr{Q}} \{\prec \in \mathbb{O} \colon C_\prec(A) \subseteq C(A)\}.$$

2.1. Examples

Let us now try to get a bit more feel for the previous theory by looking at some well-known (sets of) orderings. We will show, for example, that due to the connection between preference orders and so-called coherent lower expectations, decision-making with maximality or E-admissibility [1], or by maximising expected utility, all fit in our framework. To do this, we will assume in this Section 2.1 that $\mathscr{V}$ is the subset of $\mathbb{R}^{\mathcal{X}}$ that contains all bounded options (that is, all gambles).

2.1.1. Lower expectations

A coherent lower expectation $\underline{\mathrm{E}}$ is a functional on $\mathscr{V}$ that satisfies the following properties: for all $f, g \in \mathscr{V}$ and $\lambda \geq 0$ [9, Section 2.2.1],[2]

1. $\underline{\mathrm{E}}(f) \geq \inf(f)$, (boundedness)
2. $\underline{\mathrm{E}}(f+g) \geq \underline{\mathrm{E}}(f) + \underline{\mathrm{E}}(g)$, (super-linearity)
3. $\underline{\mathrm{E}}(\lambda f) = \lambda \underline{\mathrm{E}}(f)$. (non-negative homogeneity)

They are relevant to our framework because, as discussed by Quaeghebeur [9, Section 1.6.3], every preference order $\prec$—or, equivalently, every corresponding coherent set of desirable options $G_{\prec} \coloneqq \{u \in \mathscr{V} \colon 0 \prec u\}$—determines a coherent lower expectation

$$\underline{\mathrm{E}}_{\prec} \colon \mathscr{V} \to \mathbb{R} \colon u \mapsto \underline{\mathrm{E}}_{\prec}(u) \coloneqq \sup\{\alpha \in \mathbb{R} \colon \alpha \prec u\} = \sup\{\alpha \in \mathbb{R} \colon u - \alpha \in G_{\prec}\},$$

where we identify any real number α with the option that takes on the constant value α. The lower expectation $\underline{\mathrm{E}}_{\prec}$ 'forgets' only the 'border structure' of the preference order, in the sense that the original preference order can be retrieved up to its boundary behaviour. A pseudoinverse that we consider here is

$$\prec_{\underline{\mathrm{E}}} \coloneqq \{(u, v) \colon u < v \text{ or } 0 < \underline{\mathrm{E}}(v - u)\},$$

as this is the most conservative preference order, i.e. the smallest one, that also extends $<$. All possible pseudoinverses are preference orders $\prec$ for which $\prec_{\underline{\mathrm{E}}} \subseteq \prec \subseteq \{(u, v) \colon 0 \leq \underline{\mathrm{E}}(v - u) \text{ and } u \neq v\}$; for each such pseudoinverse $\prec$, including $\prec_{\underline{\mathrm{E}}}$, we have that $\underline{\mathrm{E}}_{\prec} = \underline{\mathrm{E}}$.

In the special case that $\underline{\mathrm{E}}(u) = -\underline{\mathrm{E}}(-u)$ for all $u \in \mathscr{V}$, $\underline{\mathrm{E}}$ is a linear functional [9, Section 2.2.2] and is therefore called a linear expectation. Such a linear expectation is typically denoted simply as E. When $\mathcal{X}$ is finite, these linear expectations have a unique corresponding so-called probability mass function (pmf) $p \colon \mathcal{X} \to [0, 1]$ with $\sum_{x \in \mathcal{X}} p(x) = 1$ such that $\mathrm{E}(u) = \mathrm{E}_p(u) \coloneqq \sum_{x \in \mathcal{X}} p(x) u(x)$ for every option $u \in \mathscr{V}$. This makes such linear expectations particularly easy to specify and evaluate.

An important observation, that will prove convenient in Section 7, is that a general coherent lower expectation $\underline{\mathrm{E}}$ can also be interpreted as a lower envelope of a set of such linear expectations [9, Propositions 2.3]. In particular, if $\mathcal{X}$ is finite, then for every coherent lower expectation $\underline{\mathrm{E}}$ there is a closed convex set P of pmfs such that $\underline{\mathrm{E}}(u) = \min_{p \in P} \mathrm{E}_p(u)$ for every option $u \in \mathscr{V}$. In practice this set P will furthermore often have a finite number of extreme points $p_1, ..., p_m$, in which case we can evaluate $\underline{\mathrm{E}}$ efficiently because then $\underline{\mathrm{E}}(u) = \min_{j=1}^{m} \mathrm{E}_{p_j}(u)$ for every option $u \in \mathscr{V}$.

2.1.2. Decision-making based on lower expectations

Since preference orders are related to coherent lower expectations, it should not come as a surprise that the same is true for choice functions, or decision-making.

The classical way of decision-making is by maximising expected utility, as for example described in [1, Section 2], in which case we start from a single linear expectation E that determines everything. For any option set $A \in \mathscr{Q}$, we then map every option $u \in A$ to its expected utility $\mathrm{E}(u)$ and choose the option that maximises this expected utility, or in case of a tie, the set of options that maximise it and are undominated

[2] They are also called coherent lower previsions, and are then typically denoted by $\underline{\mathrm{P}}$.

with respect to $<$. Since E is a linear functional, we have that '$\mathrm{E}(u) < \mathrm{E}(v)$ or $u < v$' if and only if $u \prec_{\mathrm{E}} v$. So the chosen options are the options that are undominated by $\prec_{\mathrm{E}}$, and therefore the options chosen by $C_{\prec_{\mathrm{E}}}$.

A second important way of decision-making, which starts from a single lower expectation $\underline{\mathrm{E}}$, is by maximality, as for example described by Troffaes [1, Section 3.2]. When choosing with maximality, one compares options pairwise: an option v is preferred over an option u if v dominates u (so $u < v$) or if one would pay a positive amount of utility to swap u for v, in the sense that $\underline{\mathrm{E}}(v-u) > 0$. This is exactly what $\prec_{\underline{\mathrm{E}}}$ expresses. Maximality then chooses those options that are undominated with respect to this ordering $\prec_{\underline{\mathrm{E}}}$. So we see that $C_{\prec_{\underline{\mathrm{E}}}}$ corresponds to choosing with maximality for the lower expectation $\underline{\mathrm{E}}$.

A third important way of decision-making is E-admissibility, as for example described in [1, Section 3.4]. In that case, one starts with a set $\mathcal{E}$ of linear expectations on $\mathscr{V}$. The chosen options are the ones for which there is at least one $\mathrm{E} \in \mathcal{E}$ for which $C_{\prec_{\mathrm{E}}}$ chooses them. So if we consider the set of preference orders $\mathbb{O}(\mathcal{E}) := \{\prec_{\mathrm{E}} \colon \mathrm{E} \in \mathcal{E}\}$, then the corresponding choice function $C_{\mathbb{O}(\mathcal{E})}$ chooses with E-admissibility because, for any $A \in \mathscr{Q}$, we have that $C_{\mathbb{O}(\mathcal{E})}(A) = \bigcup_{\prec \in \mathbb{O}(\mathcal{E})} C_{\prec}(A) = \bigcup_{\mathrm{E} \in \mathcal{E}} C_{\prec_{\mathrm{E}}}(A)$.

Our framework however also captures a generalisation of the previous methods (although this generalisation is still not as general as our full framework). Consider a set $\mathcal{E}$ of lower expectations. Then we have a set $\mathbb{O}(\mathcal{E}) := \{\prec_{\underline{\mathrm{E}}} \colon \underline{\mathrm{E}} \in \mathcal{E}\}$ of preference orders that defines a choice function $C_{\mathcal{E}} := C_{\mathbb{O}(\mathcal{E})}$, with

$$\begin{aligned} C_{\mathcal{E}}(A) &= C_{\mathbb{O}(\mathcal{E})}(A) \\ &= \left\{u \in A \colon (\exists \prec_{\underline{\mathrm{E}}} \in \mathbb{O}(\mathcal{E}))(\forall v \in A) u \not\prec_{\underline{\mathrm{E}}} v\right\} \\ &= \{u \in A \colon (\exists \underline{\mathrm{E}} \in \mathcal{E})(\forall v \in A) \underline{\mathrm{E}}(v-u) \leq 0 \text{ and } u \not< v\} \end{aligned}$$

for every $A \in \mathscr{Q}$, where the last equality follows from the definition of $\prec_{\underline{\mathrm{E}}}$. If all expectations in $\mathcal{E}$ are linear then we choose with E-admissibility and if $\mathcal{E}$ is a singleton, then we choose with maximality. And if $\mathcal{E}$ contains a single linear expectation, we end up maximising expected utility.

3. Consistency and the natural extension of a choice assessment

Now that we are familiar with choice functions, we move on to the topic of this paper: how to extend a partial choice assessment to a coherent choice function. In this endeavour, we consider a decision-maker and assume that there is some coherent choice function C that represents her preferences. However, we (or she) may not fully know this function. Our partial information about C comes in the form of preferences regarding some—so not necessarily all—option sets. In particular, we assume that for some option sets $A \in \mathscr{Q}$, we know that the decision-maker rejects all options in $W \subseteq A$, meaning that $C(A) \subseteq A \setminus W$. An equivalent way of expressing this, which will be more convenient for our purposes, is to state that $C(V \cup W) \subseteq V$, with $V := A \setminus W$, or equivalently, that $W \subseteq R_C(V \cup W)$. We will represent such information by an *assessment*: a set $\mathcal{A} \subseteq \mathscr{Q} \times \mathscr{Q}_\emptyset$ of pairs (V, W) of disjoint option sets—so $V \cap W = \emptyset$—with the interpretation that, for all $(V, W) \in \mathcal{A}$, the options in W are definitely rejected from $V \cup W$. Note that we do not allow $V = \emptyset$ because this cannot represent partial information about a coherent choice function due to Definition 2.6 and Proposition 2.4. Also note that $W = \emptyset$ is uninformative, since it simply states that $C(V) \subseteq V$, but is nevertheless allowed. To make this idea more concrete, let us go back to the example.

Running Example 3.1. Suppose that the strategic advisor of the farming company has previously rejected the options v_3 and v_4 from the option set A_1, as in Running Example 2.2, and has chosen v_6 from $A_2 := \{v_5, v_6\}$, where $v_5 := (3, 1)$ and $v_6 := (-4, 8)$. This corresponds to the assessment

$$\mathcal{A} = \{(\{v_1, v_2\}, \{v_3, v_4\}), (\{v_6\}, \{v_5\})\}.$$

Suppose now that the company's strategic advisor has fallen ill and the company is faced with a new decision problem that amounts to choosing from the set $A_3 = \{(-3, 4), (0, 1), (4, -3)\}$. Since no such choice was made before, the conservative option is to make the completely uninformative statement $C(A_3) \subseteq A_3$. However, perhaps the company can make a more informative choice by taking into account the advisor's previous choices? ◊

3.1. Introducing consistency and natural extension

Given an assessment $\mathcal{A}$ that we would like to extend, a first important question is whether there is a coherent choice function C that agrees with it.

Definition 3.2. An assessment $\mathcal{A}$ is *consistent* if there is a coherent choice function C such that $C(V \cup W) \subseteq V$ for all $(V, W) \in \mathcal{A}$.

To characterise this notion of consistency, we recall that a choice function C is coherent whenever there is some non-empty set of preference orders such that $C = C_{\mathcal{O}}$, and observe that it follows from Equation (1) that $C_{\mathcal{O}}$ satisfies the conditions in Definition 3.2 if and only if

$$\mathcal{O} \subseteq \mathbb{O}(\mathcal{A}) := \{ \prec \in \mathbb{O} : (\forall (V, W) \in \mathcal{A})\, C_{\prec}(V \cup W) \subseteq V \}.$$

Hence, $\mathcal{A}$ is consistent if and only if $\mathbb{O}(\mathcal{A}) \neq \emptyset$.

If an assessment $\mathcal{A}$ is consistent and there is more than one choice function that agrees with it, then the question remains which one we should use. A careful decision-maker would only want to reject options if this is implied by the assessment. So she wants a most conservative agreeing coherent choice function: one that rejects the fewest number of options. Since larger sets of preference orders lead to more conservative choice functions, this most conservative agreeing choice function then clearly exists, and is then equal to $C_{\mathbb{O}(\mathcal{A})}$. For notational convenience, for any assessment $\mathcal{A}$, we denote this function by $C_{\mathcal{A}} := C_{\mathbb{O}(\mathcal{A})}$, regardless of whether $\mathcal{A}$ is consistent; the term natural extension though, we reserve for the consistent case.

Definition 3.3. Whenever an assessment $\mathcal{A}$ is consistent, we call $C_{\mathcal{A}}$ the *natural extension* of $\mathcal{A}$.

So we see that the consistency and the natural extension of an assessment $\mathcal{A}$ are both entirely characterised by $\mathbb{O}(\mathcal{A})$. Given the importance of this set, we will now investigate its structure in more detail.

3.2. Alternative characterisations of $\mathbb{O}(\mathcal{A})$

As a first step, we consider the following more practical expression for $\mathbb{O}(\mathcal{A})$.

Proposition 3.4. Consider an assessment $\mathcal{A}$. Then

$$\mathbb{O}(\mathcal{A}) = \{ \prec \in \mathbb{O} : (\forall (V, W) \in \mathcal{A})(\forall w \in W)(\exists v \in V) w \prec v \}.$$

To prove this, it suffices to apply the following lemma to every $(V, W) \in \mathcal{A}$.

Lemma 3.5. For any preference order $\prec$ and any disjoint $V \in \mathscr{Q}$ and $W \in \mathscr{Q}_{\emptyset}$ the following statements are equivalent:

$$C_{\prec}(V \cup W) \subseteq V \tag{3}$$

and

$$(\forall w \in W)(\exists v \in V) w \prec v. \tag{4}$$

Proof. Both statements are trivially true if $W = \emptyset$, so it suffices to consider the case $W \neq \emptyset$. First we prove that Equation (3) implies Equation (4). From Equation (3) and the fact that V and W are disjoint, it follows that $w \notin C_{\prec}(V \cup W)$ for all $w \in W$. This means by definition that

$$(\forall w \in W)(\exists a \in V \cup W) w \prec a. \tag{5}$$

We will now show that this implies Equation (4). Take any option $w \in W$. By Lemma 2.5 there is some option $w^* \in W$ that is undominated in W with respect to $\prec$ such that $w \prec w^*$ or $w = w^*$. Since $w^* \in W$, we know from Equation (5) that there is some $a^* \in V \cup W$ such that $w^* \prec a^*$. Since w^* is undominated in W with respect to $\prec$, it is impossible that $a^* \in W$, so it must be that $a^* \in V$. Thus, we have found some a^* in V such that $w \prec w^* \prec a^*$ or $w = w^* \prec a^*$ and therefore, in any case, $w \prec a^*$. As this holds for any option $w \in W$, this proves Equation (4).

Next we prove that Equation (4) implies Equation (3). Take any option $w \in W$. Since $V \subseteq V \cup W$, we have from Equation (4) and the definition of $C_{\prec}$ that $w \notin C_{\prec}(V \cup W)$. Since this holds for any $w \in W$, it follows that $C_{\prec}(V \cup W) \subseteq V$, and this is Equation (3). □

Proof of Proposition 3.4 This follows immediately from the definition of $\mathbb{O}(\mathcal{A})$ and Lemma 3.5. □

Taking a closer look at the expression in Proposition 3.4, and since we know from Axiom $\prec_2$ that $w \prec v$ is equivalent to $0 \prec v - w$, we see that

$$\mathbb{O}(\mathcal{A}) \coloneqq \{\prec \in \mathbb{O} \colon (\forall H \in \mathcal{H}_{\mathcal{A}})(\exists h \in H) 0 \prec h\},$$

with $\mathcal{H}_{\mathcal{A}} \coloneqq \{\{v - w \colon v \in V\} \colon (V, W) \in \mathcal{A}, w \in W\}$; we call this $\mathcal{H}_{\mathcal{A}}$ the *conjunctive generator.*[3] So we see that $\mathbb{O}(\mathcal{A})$ is a specific instance of a set of preference orders of the form

$$\mathbb{O}(\mathcal{H}) \coloneqq \{\prec \in \mathbb{O} \colon (\forall H \in \mathcal{H})(\exists h \in H) 0 \prec h\},$$

with $\mathcal{H} \subseteq \mathscr{Q}_{\emptyset}$ a set of option sets; in particular $\mathbb{O}(\mathcal{A}) = \mathbb{O}(\mathcal{H}_{\mathcal{A}})$. To make the connection set-theoretic, let us introduce for any option $h \in \mathscr{V}$ a corresponding set of preference orders $\mathbb{O}_h \coloneqq \{\prec \in \mathbb{O} \colon 0 \prec h\}$ and for any option set $H \in \mathscr{Q}_{\emptyset}$ the notation

$$\mathbb{O}[H] \coloneqq \{\prec \in \mathbb{O} \colon (\exists h \in H) 0 \prec h\} = \bigcup_{h \in H} \mathbb{O}_h, \tag{6}$$

this enables us to write

$$\mathbb{O}(\mathcal{H}) = \bigcap_{H \in \mathcal{H}} \bigcup_{h \in H} \mathbb{O}_h = \bigcap_{H \in \mathcal{H}} \mathbb{O}[H]. \tag{7}$$

We will now proceed to transform Equation (7) into a union of intersections. In particular, since every preference order in the set $\mathbb{O}(\mathcal{H})$ prefers at least one option h in each option set $H \in \mathcal{H}$ to zero, we will split the set $\mathbb{O}(\mathcal{H})$ in (possibly overlapping) subsets of preference orders, according to which options h in each H they prefer to zero.

To formalise this, for any $\mathcal{H} \subseteq \mathscr{Q}_{\emptyset}$, we let $\Phi(\mathcal{H})$ be the set of selection functions, so those maps $\phi \colon \mathcal{H} \to \mathscr{V}$ such that $\phi(H) \in H$ for every $H \in \mathcal{H}$. Then

$$\mathcal{G}(\mathcal{H}) \coloneqq \{\{\phi(H) \colon H \in \mathcal{H}\} \colon \phi \in \Phi(\mathcal{H})\} \tag{8}$$

is the set of all sets that can be obtained by selecting one option from each $H \in \mathcal{H}$.[4] The case where $\mathcal{H} = \{H_1, ..., H_m\}$ is finite might make this more intuitive, because then

$$\mathcal{G}(\mathcal{H}) = \{\{h_1, ..., h_m\} \colon (h_1, ..., h_m) \in \times_{k=1}^{m} H_k\}. \tag{9}$$

Since $\mathbb{O}(\mathcal{H})$ consists of all preference orders that prefer at least one $h \in H$ to zero for each $H \in \mathcal{H}$, we now find that

$$\mathbb{O}(\mathcal{H}) = \{\prec \in \mathbb{O} \colon (\exists G \in \mathcal{G}(\mathcal{H}))(\forall g \in G) 0 \prec g\} = \bigcup_{G \in \mathcal{G}(\mathcal{H})} \bigcap_{g \in G} \mathbb{O}_g = \bigcup_{G \in \mathcal{G}(\mathcal{H})} \mathbf{O}[G], \tag{10}$$

with, for every set of options $G \in 2^{\mathscr{V}}$,

$$\mathbf{O}[G] \coloneqq \bigcap_{g \in G} \mathbb{O}_g = \{\, \prec \in \mathbb{O} \,:\, (\forall g \in G)\ 0 \prec g \,\}. \tag{11}$$

So we see that $\mathbb{O}(\mathcal{H})$ is a particular instance of a set of preference orders of the form

$$\mathbf{O}(\mathcal{G}) \coloneqq \bigcup_{G \in \mathcal{G}} \mathbf{O}[G], \tag{12}$$

[3] This name is chosen because of similarities with the conjunctive normal form; see Equation (7). As for our choice of letter, since $\mathcal{G}$ is reserved for the disjunctive generator that will be introduced next, we opted for the next letter in the alphabet. Moreover, in West Flemish the letter 'h' is also called 'g upwards'.

[4] A noteworthy case is $\mathcal{G}(\emptyset) = \{\emptyset\}$. This follows from the fact that if there are no sets to select from, selecting one option from "each" of these sets yields the empty set. More formally, it follows from the fact that there is a single unique function that maps "every" element of $\emptyset$ to an element of $\mathscr{V}$. Another noteworthy case is that $\mathcal{G}(\{\emptyset\}) = \emptyset$.

with $\mathcal{G} \subseteq 2^{\mathscr{V}}$ a set of option sets. In particular, $\mathbb{O}(\mathcal{H}) = \mathbf{O}(\mathcal{G}(\mathcal{H}))$. We will refer to any such set $\mathcal{G}$ as a *disjunctive generator*, or simply a generator whenever it is clear from the context what type of generator we are referring to. We will call an option set G inside a generator $\mathcal{G}$ a *generator set*. Of particular importance is the generator $\mathcal{G}_{\mathcal{A}} \coloneqq \mathcal{G}(\mathcal{H}_{\mathcal{A}})$ because it characterises $\mathbb{O}(\mathcal{A})$.

Corollary 3.6. Consider an assessment $\mathcal{A} \subseteq \mathscr{Q} \times \mathscr{Q}_{\emptyset}$. Then $\mathbb{O}(\mathcal{A}) = \mathbf{O}(\mathcal{G}_{\mathcal{A}})$.

Proof. Follows immediately from the fact that $\mathbb{O}(\mathcal{A}) = \mathbb{O}(\mathcal{H}_{\mathcal{A}})$ and Equation (10). □

To illustrate these new concepts and their relation, let us look at what happens in our running example.

Running Example 3.7. The conjunctive generator is

$$\mathcal{H}_{\mathcal{A}} = \{\{v_1 - v_3, v_2 - v_3\}, \{v_1 - v_4, v_2 - v_4\}, \{v_6 - v_5\}\}.$$

To improve readability, let us define $h_1 \coloneqq v_1 - v_3 = (4, -2)$, $h_2 \coloneqq v_2 - v_3 = (2, -1)$, $h_3 \coloneqq v_1 - v_4 = (7, -4)$, $h_4 \coloneqq v_2 - v_4 = (5, -3)$ and $h_5 \coloneqq v_6 - v_5 = (-7, 7)$. Then $\mathcal{H}_{\mathcal{A}} = \{\{h_1, h_2\}, \{h_3, h_4\}, \{h_5\}\}$ and the corresponding (disjunctive) generator is

$$\mathcal{G}_{\mathcal{A}} = \mathcal{G}(\mathcal{H}_{\mathcal{A}}) = \{\{h_1, h_3, h_5\}, \{h_1, h_4, h_5\}, \{h_2, h_3, h_5\}, \{h_2, h_4, h_5\}\}. \qquad \lozenge$$

3.3. Consistency and natural extension for generators

For a given assessment $\mathcal{A}$, we have by Corollary 3.6 that $\mathbb{O}(\mathcal{A}) = \mathbf{O}(\mathcal{G}_{\mathcal{A}})$. However, as we will see in Sections 5 and 6, the same is often true for other, simpler generators $\mathcal{G}$. For that reason, rather than focus on $\mathcal{G}_{\mathcal{A}}$ in particular, we will consider arbitrary disjunctive generators $\mathcal{G} \subseteq 2^{\mathscr{V}}$, the sets $\mathbf{O}(\mathcal{G})$ of preference orders that are generated by them and the corresponding operators $C^{\mathcal{G}} \coloneqq C_{\mathbf{O}(\mathcal{G})}$. We start by defining consistency and natural extension for such generators.

Definition 3.8. A disjunctive generator $\mathcal{G} \subseteq 2^{\mathscr{V}}$ is called consistent if $\mathbf{O}(\mathcal{G}) \neq \emptyset$. Whenever it is consistent, we call the corresponding choice function $C^{\mathcal{G}}$ its natural extension.

Note that, for the particular case of $\mathcal{G}_{\mathcal{A}}$, the consistency and natural extension of $\mathcal{G}_{\mathcal{A}}$ is equivalent to that of $\mathcal{A}$: $\mathbb{O}(\mathcal{A}) \neq \emptyset \Leftrightarrow \mathbf{O}(\mathcal{G}_{\mathcal{A}}) \neq \emptyset$ and $C^{\mathcal{G}_{\mathcal{A}}} = C_{\mathbf{O}(\mathcal{G}_{\mathcal{A}})} = C_{\mathbb{O}(\mathcal{A})} = C_{\mathcal{A}}$.

Since $\mathbf{O}(\mathcal{G}) = \bigcup_{G \in \mathcal{G}} \mathbf{O}[G]$, we see that a generator $\mathcal{G}$ is consistent if and only if $\mathbf{O}[G] \neq \emptyset$ for at least one $G \in \mathcal{G}$. Alternatively, due to Proposition 2.4, $\mathcal{G}$ is consistent if and only if $0 \in C^{\mathcal{G}}(\{0\})$. For the operator $C^{\mathcal{G}}$ itself, on the other hand, we see from the definitions that

$$C^{\mathcal{G}}(A) = C_{\mathbf{O}(\mathcal{G})}(A) = \{u \in A \colon (\exists G \in \mathcal{G})(\exists \prec \in \mathbf{O}[G])(\forall a \in A) u \not\prec a\}, \tag{13}$$

for any $A \in \mathscr{Q}$. For these reasons, it will be useful to find a convenient way of checking, for any $G \subseteq \mathscr{V}$, whether $\mathbf{O}[G] \neq \emptyset$ and whether there is some $\prec \in \mathbf{O}[G]$ such that $u \not\prec a$ for all $a \in A$. As we will now show, both problems are closely related to well-known concepts from the theory of coherent sets of desirable options [5].

3.4. Consistency and natural extension for generator sets

Since we know from Lemma 2.3 that there is a one-to-one correspondence between preference orders and coherent sets of desirable options, studying preference orders $\prec \in \mathbf{O}[G]$ is equivalent to studying their corresponding coherent sets of desirable options $G_{\prec} \in \overline{\mathbf{G}}$. More explicitly, it follows from the definition of $\mathbf{O}[G]$ and $G_{\prec}$ that, for any preference order $\prec$,

$$\prec \in \mathbf{O}[G] \Leftrightarrow G \subseteq G_{\prec}. \tag{14}$$

So we see that working with $\mathbf{O}[G]$ is equivalent to working with $\overline{\mathbf{G}}_G \coloneqq \{\, D \in \overline{\mathbf{G}} : G \subseteq D \,\}$. Conveniently, this set of compatible coherent sets of desirable options is well studied, allowing us to borrow some results from the theory of coherent sets of desirable options.

First, an option set $G \subseteq \mathscr{V}$ is called *consistent* if $\overline{\mathbf{G}}_G \neq \emptyset$, meaning that it can be extended to a coherent set of desirable options. Van Camp et al. [5, Theorem 2,(i)⇔(ii)] provide the following alternative characterisation:

$$\overline{\mathbf{G}}_G \neq \emptyset \Leftrightarrow \text{posi}(G) \cap \{v \in \mathscr{V} : v \leq 0\} = \emptyset \tag{15}$$

where, for any $V \subseteq \mathscr{V}$,

$$\text{posi}(V) \coloneqq \left\{ \sum_{j=1}^{n} \lambda_j v_j \,:\, n \in \mathbb{N}, v_j \in V, \lambda_j > 0 \right\}.$$

If G is consistent, then by [5, Theorem 2,(v)] we furthermore have that among all the compatible $D \in \overline{\mathbf{G}}_G$, there is a least informative—smallest—coherent set of desirable options that contains G. It is called the natural extension of G and given by $\mathcal{N}(G) \coloneqq \text{posi}(G \cup \mathscr{V}_{>0})$, where $\mathscr{V}_{>0} \coloneqq \{u \in \mathscr{V} : 0 < u\}$. More formally: for any consistent G, we have that $\mathcal{N}(G) \in \overline{\mathbf{G}}_G$ and $\mathcal{N}(G) \subseteq D$ for all $D \in \overline{\mathbf{G}}_G$.

For non-consistent G, we also let $\mathcal{N}(G) \coloneqq \text{posi}(G \cup \mathscr{V}_{>0})$, but we then no longer call it the natural extension of G. This allows for the following alternative characterisation of consistency.

Lemma 3.9. A set of options $G \subseteq \mathscr{V}$ is consistent if and only if $0 \notin \mathcal{N}(G)$.

Proof. If G is consistent, then $\mathcal{N}(G) \in \overline{\mathbf{G}}_G$ is coherent by [5, Theorem 2,(v)], so it follows from Axiom d_0 that $0 \notin \mathcal{N}(G)$. So it remains to prove that $0 \in \mathcal{N}(G)$ for any G that is not consistent. If G is not consistent, then by Equation (15), we know that $\text{posi}(G) \cap \{v \in \mathscr{V} : v \leq 0\} \neq \emptyset$. Take any $u \in \text{posi}(G) \cap \{v \in \mathscr{V} : v \leq 0\}$. Then there are some $n \in \mathbb{N}$, $v_j \in G$ and $\lambda_j > 0$ such that $u = \sum_{j=1}^{n} \lambda_j v_j \leq 0$. If $u = 0$ then we are done because $0 = u = \sum_{j=1}^{n} \lambda_j v_j \in \text{posi}(G \cup \mathscr{V}_{>0}) = \mathcal{N}(G)$ by definition. If $u \neq 0$, then $p = -u \in \mathscr{V}_{>0}$ such that $0 = p + u = p + \sum_{j=1}^{n} \lambda_j v_j$. Since $p \in \mathscr{V}_{>0}$, we have that $0 \in \text{posi}(G \cup \mathscr{V}_{>0}) = \mathcal{N}(G)$. □

These results for coherent sets of desirable options can be immediately used in our context. First, due to Equation (14), G is consistent if and only if $\mathbf{O}[G] \neq \emptyset$. It therefore follows from Lemma 3.9 that

$$\mathbf{O}[G] \neq \emptyset \Leftrightarrow \overline{\mathbf{G}}_G \neq \emptyset \Leftrightarrow 0 \notin \mathcal{N}(G). \tag{16}$$

Secondly, we can also use the $\mathcal{N}$ operator to reformulate the condition that appears in Equation (13).

Lemma 3.10. Let $A \in \mathscr{Q}$ be an option set and consider an option $u \in A$ and a set of options $G \subseteq \mathscr{V}$. Then the following statements are equivalent

(i) there is a preference order $\prec \in \mathbf{O}[G]$ such that for all options $a \in A$ we have $u \not\prec a$;

(ii) there is some $D \in \overline{\mathbf{G}}_G$ such that $(A - u) \cap D = \emptyset$;

(iii) $(A - u) \cap \mathcal{N}(G) = \emptyset$.

Proof. If G is not consistent then all statements are trivially true by Equation (16) because $0 = u - u \in A - u$. So we only have to handle the case where G is consistent. We give a circular proof of the negation of all statements.

First we prove that the negation of statement (i) implies the negation of statement (ii). Assume that for all $\prec \in \mathbf{O}[G]$ there is some $a \in A$ such that $u \prec a$. Then for any $D \in \overline{\mathbf{G}}_G$, by Lemma 2.3, $\prec_D$ is a preference order such that $G_{\prec_D} = D$. It therefore follows from Equation (14) that $\prec_D \in \mathbf{O}[G]$. Therefore, there is some $a \in A$ such that $u \prec_D a$. Hence, $a - u \in D$, so that $(A - u) \cap D \neq \emptyset$.

Next we prove that the negation of statement (ii) implies the negation of statement (iii). Assume that for all $D \in \overline{\mathbf{G}}_G$ we have $(A - u) \cap D \neq \emptyset$. By [5, Theorem 2], since G is consistent, we have that $\mathcal{N}(G) \in \overline{\mathbf{G}}_G$, which immediately implies statement (iii).

Finally, we prove that the negation of statement (iii) implies the negation of statement (i). Assume that $(A - u) \cap \mathcal{N}(G) \neq \emptyset$. Take any $\prec \in \mathbf{O}[G]$. Since G is consistent, we have that $\mathcal{N}(G)$ is the smallest coherent set of desirable options in $\overline{\mathbf{G}}_G$ [5, Theorem 2,(v)]. Moreover, by Equation (14) we have that $G \subseteq G_{\prec}$ and, by Lemma 2.3 that $G_{\prec} \in \overline{\mathbf{G}}$, so $G_{\prec} \in \overline{\mathbf{G}}_G$. Whence, $\mathcal{N}(G) \subseteq G_{\prec}$. Therefore, since $(A - u) \cap \mathcal{N}(G) \neq \emptyset$, there is some $a \in A$ such that $a - u \in G_{\prec}$. By definition and Axiom $\prec_2$ this means that $a \prec u$. □

3.5. Connecting back to generators

From these results, we can now check the consistency of a generator in a straightforward way.

Lemma 3.11. A generator $\mathcal{G} \subseteq 2^{\mathscr{V}}$ is consistent if and only if there is some $G \in \mathcal{G}$ such that $0 \notin \mathcal{N}(G)$.

Proof. By definition, the consistency of $\mathcal{G}$ is equivalent to there being some $G \in \mathcal{G}$ such that $\mathbf{O}[G] \neq \emptyset$. By Lemma 3.9 this is equivalent to there being some $G \in \mathcal{G}$ such that $0 \notin \mathcal{N}(G)$. □

Similarly, the lemmas above also lead up to the following theorem that allows us to evaluate $C^{\mathcal{G}}$ for any finite option set in terms of checks on $\mathcal{N}(G)$ for the generator sets $G \in \mathcal{G}$.

Theorem 3.12. Consider any generator $\mathcal{G} \subseteq 2^{\mathscr{V}}$. For any option set $A \in \mathscr{Q}$ and option $u \in A$, $u \in C^{\mathcal{G}}(A)$ if and only if there is some $G \in \mathcal{G}$ such that $(A - u) \cap \mathcal{N}(G) = \emptyset$.

Proof. By Equation (13), $u \in C^{\mathcal{G}}(A)$ is equivalent to

$$(\exists G \in \mathcal{G})(\exists \prec \in \mathbf{O}[G])(\forall a \in A) u \not\prec a,$$

and this is by Lemma 3.10 equivalent to $(\exists G \in \mathcal{G})(A - u) \cap \mathcal{N}(G) = \emptyset$. □

Interestingly, this result does not only characterise the natural extension of consistent $\mathcal{G}$. Since we know from Section 3.3 that $\mathcal{G}$ is consistent if and only if $0 \in C^{\mathcal{G}}(\{0\})$, this result also includes Lemma 3.11 as a special case.

4. Practical methods for finite generator sets

We've just seen that both the problem of checking the consistency of an assessment and computing its natural extension reduce to checking for multiple options $v \in \mathscr{V}$ and option sets $G \in \mathcal{G}$ whether v belongs to $\mathcal{N}(G)$. This can be done in various ways, but in this section we will proceed to propose one way for the case where G is finite, and show how this leads to practical algorithms for checking the consistency and evaluating $C^{\mathcal{G}}$ for generators $\mathcal{G}$ that consist of a finite number of such finite generator sets G.

4.1. Checking if an option belongs to $\mathcal{N}(G)$ for finite G

If an assessment $\mathcal{A}$ is finite, which will often be the case in practice, then there are only a finite number of option sets in $\mathcal{H}_{\mathcal{A}}$ each containing a finite number of options. The disjunctive generator $\mathcal{G}_{\mathcal{A}}$ will then also consist of a finite number of option sets—at most $\prod_{H \in \mathcal{H}_{\mathcal{A}}} |H|$, as can be seen from Equation (9)—each of which contains only a finite number of options. The following proposition gives a more practical way of checking, for one such *finite* option set G, whether an option belongs to $\mathcal{N}(G)$. We use the convention that for all non-negative integers $n \geq 0$, $(\lambda_1, \ldots, \lambda_n) > 0$ means that $(\lambda_1, \ldots, \lambda_n)$ is an n-tuple of real numbers such that $\lambda_j \geq 0$ for all $j \in \{1, ..., n\}$ and $\lambda_j > 0$ for at least one $j \in \{1, ..., n\}$; in particular, for $n = 0$, no such $(\lambda_1, ..., \lambda_n) > 0$ exists.

Proposition 4.1. For any option set $G = \{g_1, ..., g_m\} \in \mathscr{Q}_{\emptyset}$ and option $v \in \mathscr{V}$, $v \in \mathcal{N}(G)$ if and only if at least one of the following two conditions holds:

(i) $0 < v$;

(ii) there is some $(\lambda_1, \ldots, \lambda_m) > 0$ such that $\sum_{j=1}^{m} \lambda_j g_j \leq v$.

Proof. First we prove the implication to the left. If (i) holds, we have that $v \in \mathscr{V}_{>0} \subseteq \text{posi}(G \cup \mathscr{V}_{>0}) = \mathcal{N}(G)$. If (ii) holds—which implies that $m > 0$—we consider two cases. If $\sum_{j=1}^{m} \lambda_j g_j = v$, then we are done by definition of the posi operator. Otherwise, let $p := v - \sum_{j=1}^{m} \lambda_j g_j \in \mathscr{V}_{>0}$ and $\lambda_{m+1} := 1$. Then $v = \sum_{j=1}^{m} \lambda_j g_j + \lambda_{m+1} p \in \text{posi}(G \cup \mathscr{V}_{>0}) = \mathcal{N}(G)$.

Next, we prove the implication to the right. Since $v \in \mathcal{N}(G)$, v is a finite positive linear combination of elements of $\{g_1, ..., g_m\} \cup \mathscr{V}_{>0}$. By summing terms with the same g_i together, and adding terms $\lambda_i g_i$ with

$\lambda_i = 0$ for those g_i that do not appear in this positive linear combination, we see that there is a natural number $\ell \geq 1$ and there are $(\lambda_1, \ldots, \lambda_{m+\ell}) > 0$ and $p_1, \ldots, p_\ell \in \mathscr{V}_{>0}$ such that $v = \sum_{j=1}^{m} \lambda_j g_j + \sum_{j=1}^{\ell} \lambda_{m+j} p_j$. If $\lambda_1 = \ldots = \lambda_m = 0$, we infer from this that $(\lambda_{m+1}, \ldots, \lambda_{m+\ell}) > 0$ and $0 < \sum_{j=1}^{\ell} \lambda_{m+j} p_j = v$, which is (i). In the other case we have that $(\lambda_1, \ldots, \lambda_m) > 0$ and, since $0 \leq \sum_{j=1}^{\ell} \lambda_{m+j} p_j$, also that $\sum_{j=1}^{m} \lambda_j g_j = v - \sum_{j=1}^{\ell} \lambda_{m+j} p_j \leq v$, which is (ii). □

For option sets that are singletons, this condition can be simplified even further.

Corollary 4.2. Consider two options $u, v \in \mathscr{V}$. Then $v \in \mathcal{N}(\{u\})$ if and only if there is some $\lambda \geq 0$ such that $\lambda u \leq v$.

Proof. First we prove the implication to the right. By Proposition 4.1, we have that either $0 < v$ or there is some $\lambda > 0$ such that $\lambda u \leq v$. If $0 < v$, then we can choose $\lambda = 0$ and in the other case we are done.

Next we prove the implication to the left. If $\lambda = 0$, then $0 < v$, which is sufficient by Proposition 4.1 (i). If $\lambda > 0$, then this is sufficient by Proposition 4.1 (ii). □

4.2. Consistency and natural extension in practice

With Proposition 4.1 at our disposal, the methods in Section 3.5 for checking consistency and evaluating $C^{\mathcal{G}}$ can now be simplified even further, at least if $\mathcal{G}$ consists of finite option sets.

For consistency, we then know from Lemma 3.11 that $\mathcal{G}$ is consistent if and only if there is some $G \in \mathcal{G}$ such that $0 \notin \mathcal{N}(G)$. Reformulating $0 \notin \mathcal{N}(G)$ with Proposition 4.1, we arrive at the following result.

Proposition 4.3. Consider a generator $\mathcal{G} \subseteq \mathscr{Q}_\emptyset$ consisting of finite option sets. Then $\mathcal{G}$ is consistent if and only if there is some $G = \{g_1, ..., g_m\} \in \mathcal{G}$ such that $\sum_{j=1}^{m} \lambda_j g_j \not\leqslant 0$ for all $(\lambda_1, ..., \lambda_m) > 0$.

Proof. Combining Lemma 3.11 and Proposition 4.1, we see that $\mathcal{G}$ is consistent if and only if there is some $G = \{g_1, ..., g_m\} \in \mathcal{G}$ for which Proposition 4.1 (i) and (ii) are both false for $v = 0$. We see immediately that (i) is always false because $0 \not< 0$, which leaves us with the requirement that (ii) should be false for $v = 0$. □

To illustrate this, we return to our running example.

Running Example 4.4. We will now go ahead and test if the strategic advisor was at least consistent (with coherence) in his choices. We know from Section 3.3 that $\mathcal{A}$ is consistent if and only if $\mathcal{G}_{\mathcal{A}}$ is. Therefore, by Proposition 4.3, we can demonstrate consistency by finding a generator set $G = \{g_1, ..., g_m\} \in \mathcal{G}_{\mathcal{A}}$ such that $\sum_{j=1}^{m} \lambda_j g_j \not\leqslant 0$ for every $(\lambda_1, ..., \lambda_m) > 0$. We will use the particular generator set $G = \{g_1, g_2, g_3\} = \{h_2, h_4, h_5\} = \{(2, -1), (5, -3), (-7, 7)\} \in \mathcal{G}_{\mathcal{A}}$. Assume *ex absurdo* that there is some $(\lambda_1, \lambda_2, \lambda_3) > 0$ such that $\lambda_1 g_1 + \lambda_2 g_2 + \lambda_3 g_3 \leq 0$. Notice that $2g_2 \leq 5g_1$, so if we let $\mu_1 := \frac{2}{5}\lambda_1 + \lambda_2$ and $\mu_2 := 7\lambda_3$ then

$$\sum_{j=1}^{3} \lambda_j g_j \geqslant \frac{2}{5}\lambda_1 g_2 + \lambda_2 g_2 + \lambda_3 g_3 = \mu_1 h_4 + \frac{1}{7}\mu_2 h_5 = (5\mu_1 - \mu_2, -3\mu_1 + \mu_2).$$

Since $\sum_{j=1}^{3} \lambda_j g_j \leq 0$, this implies that $5\mu_1 \leq \mu_2 \leq 3\mu_1$ and thus $\mu_1 \leq 0$ and $\mu_2 \leq 0$. This is impossible though because $(\lambda_1, \lambda_2, \lambda_3) > 0$ implies that $\mu_1 > 0$ or $\mu_2 > 0$. Hence, $\mathcal{A}$ is consistent. We conclude that the decisions of the strategic advisor were consistent with coherence, so we may use the natural extension to derive their consequences; we will do so in Running Example 4.6 further on. ◊

To determine $C^{\mathcal{G}}(A)$ for any option set $A \in \mathscr{Q}$, we can check for every individual $u \in A$ if $u \in C^{\mathcal{G}}(A)$. Due to Theorem 3.12, this requires us to check, for every $u \in A$, if there is some $G \in \mathcal{G}$ such that $(A - u) \cap \mathcal{N}(G) = \emptyset$, or equivalently, such that $v \notin \mathcal{N}(G)$ for all $v \in A - u$. If the generator sets in $\mathcal{G}$ are all finite, then we can reformulate this condition using Proposition 4.1.

Proposition 4.5. For any finite generator $\mathcal{G} \Subset \mathscr{Q}_\emptyset$, option set $A \in \mathscr{Q}$ and option $u \in A$, $u \in C^{\mathcal{G}}(A)$ if and only if

1. $v \not> 0$ for all $v \in A - u$, and

2. there is some $G = \{g_1, ..., g_m\} \in \mathcal{G}$ such that $\sum_{j=1}^m \lambda_j g_j \not\leq v$ for all $v \in A - u$ and $(\lambda_1, ..., \lambda_m) > 0$.

Proof. As explained in the text preceding this result, it follows from Theorem 3.12 that $u \in C^{\mathcal{G}}(A)$ if and only if there is some $G = \{g_1, ..., g_m\} \in \mathcal{G}$ such that $v \notin \mathcal{N}(G)$ for all $v \in A - u$. By Proposition 4.1, the condition $v \notin \mathcal{N}(G)$ is true if both (i) $v \not> 0$ and (ii) $\sum_{j=1}^m \lambda_j g_j \not\leq v$ for all $(\lambda_1, ..., \lambda_m) > 0$. Since (i) does not depend on G, this implies the stated result. □

We again illustrate this using our running example.

Running Example 4.6. We can now finally tackle the problem at the end of Running Example 3.1: choosing from the set $A_3 = \{(-3, 4), (0, 1), (4, -3)\}$ based on the expert's assessment. This comes down to computing $C_{\mathcal{A}}(A_3)$, or equivalently, as explained in Section 3.3, to computing $C^{\mathcal{G}_{\mathcal{A}}}(A_3)$. To see if $u = (4, -3)$ is chosen from A_3, as we know from Proposition 4.5, we need to check two things. First, whether $v \not> 0$ for all $v \in A_3 - u = \{(-7, 7), (-4, 4), (0, 0)\}$; this is clearly the case. Second, whether there is some set $G = \{g_1, ..., g_m\} \in \mathcal{G}_{\mathcal{A}}$ such that $\sum_{j=1}^m \lambda_j g_j \not\leq v$ for all $v \in A_3 - u$ and $(\lambda_1, ..., \lambda_m) > 0$. This is not the case because for every $G = \{g_1, g_2, g_3\} \in \mathcal{G}_{\mathcal{A}}$,[5] since $g_3 = h_5 = (-7, 7)$, we find for $(\lambda_1, \lambda_2, \lambda_3) = (0, 0, 1)$ and $v = (-7, 7)$ that $\sum_{j=1}^3 \lambda_j g_j = h_5 = (-7, 7) = v$. So $(4, -3)$ is not chosen from A_3. Checking if $(0, 1)$ is chosen is analogous. In this case $u = (0, 1)$ and $A_3 - u = \{(-3, 3), (0, 0), (4, -4)\}$, so a similar argument using $v = (-3, 3)$ and $(\lambda_1, \lambda_2, \lambda_3) = (0, 0, \frac{3}{7})$ leads us to conclude that $(0, 1)$ is not chosen either. Since $C_{\mathcal{A}}(A_3)$ must contain at least one option by the consistency of $\mathcal{A}$—see Running Example 4.4—and Proposition 2.4, it follows that $C_{\mathcal{A}}(A_3) = \{(-3, 4)\}$. So based on the advisor's earlier decisions, it follows that the company should choose $(-3, 4)$ from A_3. ◊

In this simple toy example, the assessment $\mathcal{A}$ was small and the conditions in Propositions 4.3 and 4.5 could be checked manually. In realistic problems, this may not be the case though. We therefore proceed to develop a more algorithmic approach.

4.3. An algorithmic approach

Fix some finite generator $\mathcal{G} \Subset \mathscr{Q}_\emptyset$. Then as we've seen in the previous section, for each $G = \{g_1, ..., g_m\} \in \mathcal{G}$, the step 'check if there is some tuple $(\lambda_1, ..., \lambda_m) > 0$ for which $\sum_{j=1}^m \lambda_j g_j \leq v$' is an essential element of our methods for checking the consistency of $\mathcal{G}$ and evaluating $C^{\mathcal{G}}$. For that reason, we introduce a boolean function IsFeasible: $\mathscr{Q}_\emptyset \times \mathscr{V} \to \{\text{true}, \text{false}\}$ for it. For every $\{g_1, ..., g_m\} \in \mathscr{Q}_\emptyset$ and $v \in \mathscr{V}$, it returns true if $\sum_{j=1}^m \lambda_j g_j \leq v$ for at least one $(\lambda_1, ..., \lambda_m) > 0$, and false otherwise.

Since we will need to evaluate IsFeasible repeatedly, we first look at how we can do this in practice. By definition, $(\lambda_1, \ldots, \lambda_m) > 0$ can be rewritten as $\lambda_j \geq 0$ for all $j \in \{1, \ldots, m\}$ and $\sum_{j=1}^m \lambda_j > 0$, which are all linear constraints. Since the condition $\sum_{j=1}^m \lambda_j g_j \leq v$ is linear as well, we have a linear feasibility problem to solve. However, strict inequalities such as $\sum_{j=1}^m \lambda_j > 0$ are problematic for software solvers for linear feasibility problems. A quick fix is to choose some very small $\epsilon > 0$ and impose the inequality $\sum_{j=1}^m \lambda_j \geq \epsilon$ instead, but since this is an approximation, it does not guarantee that the result is correct. A better solution is to use the following alternative characterisation that, by introducing an extra free variable, avoids the need for strict inequalities.[6]

Proposition 4.7. Consider any $v \in \mathscr{V}$ and any $G \coloneqq \{g_1, \ldots, g_m\} \in \mathscr{Q}_\emptyset$. Then IsFeasible$(G, v)$ = true if and only if there is some $(\mu_1, \ldots, \mu_{m+1}) \in \mathbb{R}^{m+1}$ such that $\sum_{k=1}^m \mu_k g_k \leq \mu_{m+1} v$, $\mu_k \geq 0$ for all $k \in \{1, \ldots, m\}$, $\mu_{m+1} \geq 1$ and $\sum_{k=1}^m \mu_k \geq 1$.

[5] We have already seen in Running Example 3.7 that in our example, $m = 3$ for all $G \in \mathcal{G}_{\mathcal{A}}$.

[6] This result is a special case of what Quaegebeur did for his analysis preceding the CONEstrip algorithm [11, Section 2], where $\mathcal{N}(G)$ is in his notation $\underline{\mathcal{R}}$.

Proof. Suppose IsFeasible(G, v) = true. Then there is some $(\lambda_1, ..., \lambda_m) > 0$ such that $\sum_{k=1}^{m} \lambda_k g_k \leq v$. Let

$$(\mu_1, ..., \mu_{m+1}) \coloneqq \begin{cases} (\lambda_1, \ldots, \lambda_m, 1) & \text{if } \sum_{j=1}^{m} \lambda_j \geq 1, \\ \left(\frac{\lambda_1}{\sum_{j=1}^{m} \lambda_j}, \ldots, \frac{\lambda_n}{\sum_{j=1}^{m} \lambda_j}, \frac{1}{\sum_{j=1}^{m} \lambda_j}\right) & \text{otherwise.} \end{cases}$$

Then $(\mu_1, ..., \mu_{m+1})$ satisfies the conditions in the statement.

Next we prove the implication to the left. So suppose that there is some $(\mu_1, ..., \mu_{m+1}) \in \mathbb{R}^{m+1}$ such that $\sum_{k=1}^{m} \mu_k g_k \leq \mu_{m+1} v$, $\mu_k \geq 0$ for all $k \in \{1, \ldots, m\}$, $\mu_{m+1} \geq 1$ and $\sum_{k=1}^{m} \mu_k \geq 1$. Let $(\lambda_1, ..., \lambda_n) \coloneqq (\frac{\mu_1}{\mu_{m+1}}, \ldots, \frac{\mu_m}{\mu_{m+1}})$. Then clearly $(\lambda_1, ..., \lambda_m) > 0$ and $\sum_{k=1}^{m} \lambda_k g_k \leq v$, so IsFeasible$(G, v)$ = true. □

Computing IsFeasible$(\{g_1, ..., g_n\}, v)$ is therefore a matter of solving the following linear feasibility problem:

$$\begin{array}{lll} \text{find} & \mu_1, \ldots, \mu_{n+1} \in \mathbb{R}, & \\ \text{subject to} & \mu_{n+1} v(x) - \sum_{k=1}^{n} \mu_k g_k(x) \geq 0 & \text{for all } x \in \mathcal{X}, \\ & \sum_{k=1}^{n} \mu_k \geq 1, \quad \mu_{n+1} \geq 1, & \\ & \mu_k \geq 0 & \text{for all } k \in \{1, \ldots, n\}. \end{array}$$

We will assume henceforth that you have a method for solving these problems. For finite $\mathcal{X}$, which we will consider in our experiments in Section 7, such problems can easily be solved by standard linear programming methods; see for example [12, 13].

Now, if the generator $\mathcal{G}$ itself is also finite, then we can use IsFeasible to automate Propositions 4.3 and 4.5. Consider any finite generator $\mathcal{G} \Subset \mathscr{Q}_\emptyset$. Then, by Proposition 4.3, we can check the consistency of $\mathcal{G}$ by iterating over all $G \in \mathcal{G}$ and checking whether IsFeasible$(G, 0)$ is false; if we find at least one such G, we conclude that $\mathcal{G}$ is consistent. This results in the pseudocode in Algorithm 1. Similarly, we can evaluate $C^{\mathcal{G}}$ using Proposition 4.5, as is done in Algorithm 2. The algorithm starts by iterating over all $v \in A - u$ and checking whether $v > 0$; if we find one such v we conclude that u is not chosen. Next, it checks the second condition of Proposition 4.5. So, it searches for some $G \in \mathcal{G}$ for which IsFeasible(G, v) is false for all $v \in A - u$. Once we find that IsFeasible(G, v) is true for some $v \in A - u$, then we do not have to check the remaining options in $A - u$ that we have not checked yet, and we can move on to the next G. If we have found a G for which IsFeasible(G, v) is false for all $v \in A - u$, then we conclude that u is chosen. If, in the end, we have checked all $G \in \mathcal{G}$, and we did not find such a G, then we conclude that u is not chosen.

In practice, we are however not interested in the consistency and natural extension of a generator but in that of an assessment $\mathcal{A}$. To that end, as explained in Section 3.3, it suffices to convert this assessment $\mathcal{A}$ to the conjunctive generator $\mathcal{H}_{\mathcal{A}}$ and then to the corresponding disjunctive generator $\mathcal{G}_{\mathcal{A}}$ and consider the consistency and natural extension of the latter. If $\mathcal{G}_{\mathcal{A}}$ consists of finitely many finite option sets —which it will, as we know from the beginning of Section 4.1, if $\mathcal{A}$ is finite—we can then apply Algorithms 1 and 2 to $\mathcal{G}_{\mathcal{A}}$ to check the consistency of $\mathcal{A}$ or evaluate its natural extension $C_{\mathcal{A}}$, respectively. To transform $\mathcal{A}$ to $\mathcal{H}_{\mathcal{A}}$ and $\mathcal{G}_{\mathcal{A}}$, we present two simple additional algorithms. Algorithm 3 converts an assessment $\mathcal{A}$ to the corresponding conjunctive generator $\mathcal{H}_{\mathcal{A}}$ as defined in Section 3.2. Algorithm 4 converts a conjunctive generator $\mathcal{H}$—such as $\mathcal{H}_{\mathcal{A}}$—to the corresponding disjunctive generator $\mathcal{G}(\mathcal{H})$, as given by Equation (9). For $\mathcal{H} = \emptyset$—which is the case for $\mathcal{H}_{\mathcal{A}}$ if $\mathcal{A} = \emptyset$, or may happen as a result of the simplifications in Section 5—we know from Footnote 4 that $\mathcal{G}(\mathcal{H}) = \{\emptyset\}$, which is why Algorithm 4 initialises $\mathcal{G}$ as such. We call both algorithms naive because, as we will see in Sections 5 and 6, both can be improved upon.

Algorithm 1 Check if a generator is consistent

Input: finite generator $\mathcal{G} \Subset \mathscr{Q}_\emptyset$
Output: true if $\mathcal{G}$ is consistent and false otherwise

```
for all G ∈ 𝒢 do
    if not IsFeasible(G, 0) then
        return true
return false
```

Algorithm 2 Decide if an option u is chosen from A by $C^{\mathcal{G}}$

Input: option set $A \in \mathscr{Q}$, option $u \in A$, finite generator $\mathcal{G} \Subset \mathscr{Q}_{\emptyset}$
Output: true if $u \in C^{\mathcal{G}}(A)$ and false otherwise

for all $v \in A - u$ **do**
  if $v > 0$ **then**
    return false
for all $G \in \mathcal{G}$ **do**
  Found-v $\leftarrow$ false
  for all $v \in A - u$ **do**
    if IsFeasible(G, v) **then**
      Found-v $\leftarrow$ true
      break
  if not Found-v **then**
    return true
return false

Algorithm 3 From assessment to conjunctive generator

Input: finite assessment $\mathcal{A} \Subset \mathscr{Q} \times \mathscr{Q}_{\emptyset}$
Output: finite conjunctive generator $\mathcal{H} \coloneqq \mathcal{H}_{\mathcal{A}} \Subset \mathscr{Q}$

function AssessmentToDisjunctiveNaive$(\mathcal{A})$
  $\mathcal{H} \leftarrow \emptyset$
  for all $(V, W) \in \mathcal{A}$ **do**
    for all $w \in W$ **do**
      $\mathcal{H} \leftarrow \mathcal{H} \cup \{V - w\}$
  return $\mathcal{H}$

Algorithm 4 From conjunctive to disjunctive generator

Input: finite conjunctive generator $\mathcal{H} \Subset \mathscr{Q}_{\emptyset}$
Output: finite disjunctive generator $\mathcal{G} \coloneqq \mathcal{G}(\mathcal{H}) \Subset \mathscr{Q}_{\emptyset}$

function ConjunctiveToDisjunctiveNaive$(\mathcal{H})$
  $\mathcal{G} \leftarrow \{\emptyset\}$
  for all $H \in \mathcal{H}$ **do**
    $\mathcal{G}^* \leftarrow \emptyset$
    for all $G \in \mathcal{G}$ **do**
      for all $h \in H$ **do**
        $\mathcal{G}^* \leftarrow \mathcal{G}^* \cup \{G \cup \{h\}\}$
    $\mathcal{G} \leftarrow \mathcal{G}^*$
  return $\mathcal{G}$

4.4. Practical considerations

If we want to implement Algorithms 1 to 4 in the programming language of your choice, there are some practical aspects to take into account.

A first such aspect is which data type to use to store generators in all the previous algorithms. One option, that is closest to the mathematical descriptions in our pseudocode, is to use the set data type, which is implemented in most common programming languages. However, we opt to use arrays; that is, we store option sets as arrays and sets of option sets as arrays of arrays. One reason is that arrays are usually optimised for iteration, which is what we do in Algorithms 1 and 2. Another advantage of arrays is that they use slightly less memory when no duplicates are present, because set-like structures save some additional information to maintain their hash table. The advantages of set-like structures, such as faster look-up and removal, on the other hand, are not of use in our algorithms. Moreover, the only difference is that for arrays it is possible that duplicates are present, and it can be seen from Propositions 4.3 and 4.5 that even when there are duplicate option sets, this would just lead us to check the same option set twice, which will not alter the result. Similarly, inside the option sets $G \in \mathcal{G}$ it will not matter to have the same option twice, as the λ's for the same options can be lumped together. Since it is often a reasonable assumption that duplicates will be very rare, this leads us to think that the advantages of arrays outweigh the disadvantages here.

A second particular aspect is that Algorithm 4 will lead to memory explosion for large conjunctive generators if we save the disjunctive generator in memory. This is because $|\mathcal{G}(\mathcal{H})| = |\times_{H\in\mathcal{H}} H|$ for a given conjunctive generator $\mathcal{H}$ (when using arrays). Therefore, it is in practice better to only save the conjunctive generator in memory and do the iteration over the disjunctive generator on the fly[7] when we need it in Algorithms 1 and 2.

We will take both considerations into account in our implementation when we will test our algorithms in Section 7. However, the exponential explosion can still manifest itself in the running time of Algorithms 1 and 2. To address this potential issue, in the next two sections, we will develop methods to reduce the size of a (conjunctive or disjunctive) generator, without altering the corresponding set of preference orders. We start with conjunctive generators.

5. Simplifying conjunctive generators

For any given conjunctive generator $\mathcal{H} \Subset \mathscr{Q}_\emptyset$—and for $\mathcal{H}_\mathcal{A}$ in particular—the consistency and natural extension of the corresponding generator $\mathcal{G}(\mathcal{H})$—and that of $\mathcal{G}_\mathcal{A} = \mathcal{G}(\mathcal{H}_\mathcal{A})$ in particular—are fully determined by the corresponding set of preferences $\mathbf{O}(\mathcal{G}(\mathcal{H})) = \mathbb{O}(\mathcal{H})$. Therefore, if we can simplify $\mathcal{H}$ without altering $\mathbb{O}(\mathcal{H})$, this will reduce the running time of Algorithms 1, 2 and 4 without altering the result. The aim of this section is to achieve such simplifications, either by removing option sets from $\mathcal{H}$ or by removing options inside the individual option sets in $\mathcal{H}$. At the end we also present an algorithm that implements these simplifications.

5.1. Removing option sets containing a positive option

First we show that we can sometimes remove a whole option set. In particular, for any conjunctive generator $\mathcal{H}$, we can remove all option sets $H \in \mathcal{H}$ for which $\mathbb{O}[H] = \mathbb{O}$, as doing so does not alter $\mathbb{O}(\mathcal{H}) = \bigcap_{H\in\mathcal{H}} \mathbb{O}[H]$. The following lemma gives a simple necessary and sufficient condition for this to happen.

Lemma 5.1. Consider an option set $H \in \mathscr{Q}$. Then $\mathbb{O}[H] = \mathbb{O}$ if and only if there is an option $h \in H$ such that $0 < h$.

Proof. First we prove the implication to the right. Suppose that $\mathbb{O}[H] = \mathbb{O}$, then $<\in \mathbb{O} = \mathbb{O}[H]$. So by definition there is some $h \in H$ such that $0 < h$.

Next we prove the implication to the left. Since $0 < h$ and any $\prec \in \mathbb{O}$ extends $<$ by Axiom $\prec_4$, we have $\mathbb{O}_h = \mathbb{O}$. Therefore, by Equation (7), we have that $\mathbb{O}[H] = \bigcup_{h\in H} \mathbb{O}_h = \mathbb{O}$. □

[7] In Julia and Python this is done by iterating over 'Itertools.product' of $\mathcal{H}$, for example.

5.2. Removing negative options

The next simplification is based on the idea that some options $u \in H$ are non-informative. In particular, if an option $u \in H$ is nonpositive ($u \leq 0$), then it cannot be preferred to zero by a preference order since preference orders extend $<$, so $\mathbb{O}_u = \emptyset$. Removing u from H will therefore not change $\mathbb{O}[H]$. This is made formal in the following lemma.

Lemma 5.2. Consider an option set $H \in \mathscr{Q}$ and an option $u \in H$. If $u \leq 0$, then $\mathbb{O}[H] = \mathbb{O}[H \setminus \{u\}]$.

Proof. We will first show that $\mathbb{O}_u = \emptyset$ if $u \leq 0$. Suppose *ex absurdo* that there is some $\prec \in \mathbb{O}_u$. Then by definition of $\mathbb{O}_u$, $0 \prec u$. If $u = 0$ this is already a contradiction with Axiom $\prec_0$. If $u < 0$ then by Axiom $\prec_4$ it follows that $u \prec 0 \prec u$. By Axiom $\prec_1$ then $u \prec u$, which is a contradiction with Axiom $\prec_0$.

Therefore, by Equation (7), we have that

$$\mathbb{O}[H] = \bigcup_{h \in H} \mathbb{O}_h = \bigcup_{h \in H \setminus \{u\}} \mathbb{O}_h = \mathbb{O}[H \setminus \{u\}].$$

□

It is possible that all options in H are nonpositive. We can then remove all of them, which leads to $H = \emptyset$ and $\mathbb{O}[H] = \bigcup_{h \in \emptyset} \mathbb{O}_h = \emptyset$. For the corresponding disjunctive generator $\mathcal{G}(\mathcal{H})$, this implies that

$$\mathbb{O}(\mathcal{G}(\mathcal{H})) = \mathbb{O}(\mathcal{H}) = \bigcap_{H \in \mathcal{H}} \mathbb{O}[H] = \emptyset, \tag{17}$$

so we can conclude that $\mathcal{G}(\mathcal{H})$ is not consistent.

We can extend the idea of Lemma 5.2 of removing negative options to removing options that are 'dominated' by another option in the same set; this is what we will look at next.

5.3. Removing dominated options

If we know that there are two options $u, v \in H$ such that $\mathbb{O}_u \subseteq \mathbb{O}_v$, then we can remove u from H because it does not contribute to the union in the definition of $\mathbb{O}[H]$. The following lemma gives a practical sufficient condition for this to happen.

Lemma 5.3. Consider an option set $H \in \mathscr{Q}$ and two distinct options $u, v \in H$. If $v \in \mathcal{N}(\{u\})$, then $\mathbb{O}[H] = \mathbb{O}[H \setminus \{u\}]$.

A direct proof using Corollary 4.2 is fairly straightforward. We instead opt to prove it using two more abstract lemmas though, because these will prove useful later on as well.

Lemma 5.4. Consider two sets of options $A, B \subseteq \mathscr{V}$. Then $A \subseteq \mathcal{N}(B)$ if and only if $\mathcal{N}(A) \subseteq \mathcal{N}(B)$.

Proof. First, assume that $A \subseteq \mathcal{N}(B)$. Take any $u \in \mathcal{N}(A)$. Then by definition of $\mathcal{N}(A)$ there are $n \in \mathbb{N}$ and $v_j \in A \cup \mathscr{V}_{>0}$ and $\lambda_j > 0$ such that $u = \sum_{j=1}^{n} \lambda_j v_j$. Since $A \subseteq \mathcal{N}(B)$, we have for every $v_j \in A$, by definition of the posi operator, that there are $n_j \in \mathbb{N}$, $w_{j,k} \in B \cup \mathscr{V}_{>0}$ and $\lambda_{j,k} > 0$ such that $v_j = \sum_{k=1}^{n_j} \lambda_{j,k} w_{j,k}$. The same is true for $v_j \in \mathscr{V}_{>0}$ if we choose $n_j = 1$ and $\lambda_{j,1} = 1$. Therefore, we have that

$$u = \sum_{j=1}^{n} \lambda_j \left(\sum_{k=1}^{n_j} \lambda_{j,k} w_{j,k} \right) = \sum_{j=1}^{n} \sum_{k=1}^{n_j} (\lambda_j \lambda_{j,k}) w_{j,k}$$

Since $w_{j,k} \in B \cup \mathscr{V}_{>0}$ and $(\lambda_j \lambda_{j,k}) > 0$, we have by definition of the posi operator that $u \in \mathcal{N}(B)$.

Next, assume that $\mathcal{N}(A) \subseteq \mathcal{N}(B)$. Take any $a \in A$. Using $n = 1$, $\lambda_1 = 1$ and $a \in A \cup \mathscr{V}_{>0}$ in the definition of $\mathcal{N}(A)$ we have that $a \in \mathcal{N}(A) \subseteq \mathcal{N}(B)$. Therefore, $A \subseteq \mathcal{N}(B)$. □

Lemma 5.5. Consider two option sets $G_1, G_2 \subseteq 2^{\mathscr{V}}$. If $\mathcal{N}(G_2) \subseteq \mathcal{N}(G_1)$, then $\mathbf{O}[G_1] \subseteq \mathbf{O}[G_2]$.

Proof. Take any $\prec \in \mathbf{O}[G_1]$. By Equation (14) we have that $G_1 \subseteq G_\prec$. Since $G_\prec$ is coherent because of Lemma 2.3, this implies that G_1 is consistent. By [5, Theorem 2] $\mathcal{N}(G_1)$ is therefore the smallest coherent set of desirable options that contains G_1. Since $G_1 \subseteq G_\prec$ and $G_\prec$ is coherent, this implies that $\mathcal{N}(G_1) \subseteq G_\prec$. On the other hand, since $\mathcal{N}(G_2) \subseteq \mathcal{N}(G_1)$ we know from Lemma 5.4 that $G_2 \subseteq \mathcal{N}(G_1)$. Hence, $G_2 \subseteq \mathcal{N}(G_1) \subseteq G_\prec$. Therefore, $\prec \in \mathbf{O}[G_2]$ by Equation (14). □

Proof of Lemma 5.3 By Lemmas 5.4 and 5.5 we have that $\mathbf{O}[\{u\}] \subseteq \mathbf{O}[\{v\}]$. By definition this is the same as $\mathbb{O}_u \subseteq \mathbb{O}_v$. Therefore, by Equation (7), we have that

$$\mathbb{O}[H] = \bigcup_{h \in H} \mathbb{O}_h = \bigcup_{h \in H \setminus \{u\}} \mathbb{O}_h = \mathbb{O}[H \setminus \{u\}],$$

as required. □

In the case of our running example, all the option sets in $\mathcal{H}_\mathcal{A}$ can even be reduced to singletons.

Running Example 5.6. In Running Example 3.7 we determined that $\mathcal{H}_\mathcal{A} = \{\{h_1, h_2\}, \{h_3, h_4\}, \{h_5\}\}$, where $h_1 = (4, -2)$, $h_2 = (2, -1)$, $h_3 = (7, -4)$, $h_4 = (5, -3)$ and $h_5 = (-7, 7)$. For $\{h_1, h_2\}$ we see that $2h_2 = h_1$ and in $\{h_3, h_4\}$ we see that $\frac{7}{5}h_4 \leq h_3$. So, by Lemma 5.3 and Corollary 4.2, we can simplify the conjunctive generator $\mathcal{H}_\mathcal{A}$ to $\mathcal{H} \coloneqq \{\{h_2\}, \{h_3\}, \{h_5\}\}$ without altering the corresponding set of preferences $\mathbb{O}(\mathcal{H}_\mathcal{A}) = \mathbb{O}(\mathcal{H})$. ◊

The generator in this example is small, making it easy to check which options can be removed. In general, however, it would be useful to automate this process, which is what we now set out to do.

Inspired by Lemma 5.3, we introduce the binary relation $\trianglelefteq$ on $\mathscr{V}$ defined by $u \trianglelefteq v \Leftrightarrow v \in \mathcal{N}(\{u\})$. Since we know from Lemma 5.4 that $u \trianglelefteq v$ is equivalent to $\mathcal{N}(\{v\}) \subseteq \mathcal{N}(\{u\})$ the relation $\trianglelefteq$ is a (non-strict) preorder since it is clearly reflexive and transitive. To describe $\trianglelefteq$, we use the function $\textsc{optionOrd}\colon \mathscr{V}^2 \to \{\text{true}, \text{false}\}$ that returns true for a given option pair $(u, v) \in \mathscr{V}^2$ whenever $u \trianglelefteq v$. Due to Proposition 4.1, $\textsc{optionOrd}(u, v)$ is true if $0 < v$ or is otherwise equal to $\textsc{IsFeasible}(\{u\}, v)$; this allows us to easily evaluate it in practice. Since we know from Lemma 5.3 that elements of H that are dominated according to $\trianglelefteq$ can be removed from H, the best we can do is replace H by a smallest subset that contains no such dominated elements.

Algorithm 5 is one way to construct such a minimal set. The algorithm works for any (non-strict) preorder, allowing us to use it for other purposes in Section 6 as well. To obtain a minimal set, it selects exactly one option from every equivalence class of options that are dominated by each other but by no other option. It works by starting with an empty set $S_{\max}$ and adding options to it, one by one, that are not dominated by any other option in $S_{\max}$. Every time we add such an option to $S_{\max}$, we also remove the other options from $S_{\max}$ that are dominated by this new option. Applied to a set S with n elements, Algorithm 5 requires at most $\frac{n(n-1)}{2}$ binary comparisons between elements of S.

In particular, starting from a conjunctive generator $\mathcal{H}$, we can replace it by the simplified conjunctive generator $\{\textsc{Max}(H, \textsc{optionOrd})\colon H \in \mathcal{H}\}$, where, for every option set $H \in \mathcal{H}$, $\textsc{Max}(H, \textsc{optionOrd})$ uses $\textsc{optionOrd}$ at most $\frac{|H|(|H|-1)}{2}$ times.

Due to Lemma 5.3, this will not alter the corresponding set of preferences.

Corollary 5.7. For any option set $H \in \mathscr{Q}_\emptyset$,

$$\mathbb{O}[H] = \mathbb{O}[\textsc{Max}(H, \textsc{optionOrd})].$$

Proof. Since Algorithm 5 only removes options that are dominated according to $\trianglelefteq$, this follows from repeated application of Lemma 5.3. □

We end this section on simplifying conjunctive generators by presenting an algorithm that uses these simplifications to simplify the conjunctive generator $\mathcal{H}_\mathcal{A}$ of an assessment $\mathcal{A}$. Combining the definition of $\mathcal{H}_\mathcal{A}$ and the simplifications in Lemmas 5.1 to 5.3, we get the procedure in Algorithm 6. It works by iterating

Algorithm 5 Find 'maximal' elements of a preordered set

Input: a preordered finite set S, a comparison function $f\colon S \times S \to \{\text{true}, \text{false}\}$ such that $f(s,t) = \text{true}$ whenever s is dominated by t in the preorder
Output: 'maximal' elements of S with only one of every equivalence class

```
function Max(S, f)
    S_max ← ∅
    for all s ∈ S do
        IsMaximal ← true
        for all t ∈ S_max do
            if f(s,t) then
                IsMaximal ← false
                break
        if IsMaximal then
            S_new ← {s}
            for all t ∈ S_max do
                if not f(t,s) then
                    S_new ← S_new ∪ {t}
            S_max ← S_new
    return S_max
```

over all pairs $(V, W) \in \mathcal{A}$ and all $w \in W$. For each such w it iterates over, the algorithm collects all $v - w \not\leq 0$, with $v \in V$, into an option set H—because if $v - w \leq 0$, we can remove it by Lemma 5.2—unless $v - w > 0$ for some $v \in V$, in which case the option set H can be removed by Lemma 5.1. After creating such an option set H we simplify it further by removing dominated options, using Algorithm 5, and add it to $\mathcal{H}$. It could be that we add $H = \emptyset$ at some point—if $v - w \leq 0$ for all $v \in V$—in which case, whatever the other option sets in $\mathcal{H}$ are, we know from Equation (17) that $\mathbb{O}(\mathcal{H}) = \emptyset$ and therefore that $\mathcal{A}$ is inconsistent. Since the other option sets do not matter, we then directly return the simplest conjunctive generator $\mathcal{H}$ for which $\mathbb{O}(\mathcal{H}) = \emptyset$, which is $\{\emptyset\}$.

Here too the use of arrays instead of sets is not a problem, because it will only make us check some things multiple times. It is even advisable, for the reasons explained in Section 4.4.

6. Simplifying disjunctive generators

Even if we simplify $\mathcal{H}$ using the methods in Section 5, transforming $\mathcal{H}$ into $\mathcal{G}(\mathcal{H})$ using Algorithm 4 can still lead to an exponential explosion in the size of this generator. To address this issue, we will now proceed to develop a method to simplify a disjunctive generator $\mathcal{G}$, and we will use it to simplify $\mathcal{G}(\mathcal{H})$ without ever explicitly constructing $\mathcal{G}(\mathcal{H})$ itself. If we do this without altering the corresponding set of preference orders $\mathbf{O}(\mathcal{G}(\mathcal{H}))$, then as before, this will reduce the running time of Algorithms 1 and 2 without altering the result. We will achieve this by removing entire generator sets that do not contribute any new information, or individual uninformative options inside generator sets.

6.1. Removing option sets from generators

Our first simplifications involve removing option sets $G \in \mathcal{G}$ that do not contribute any information, in the sense that $\mathbf{O}(\mathcal{G} \setminus \{G\}) = \mathbf{O}(\mathcal{G})$. The first such type of option sets are those that are inconsistent, or equivalently, due to Lemma 3.9, those for which $0 \in \mathcal{N}(G)$.

Lemma 6.1. Consider a set of option sets $\mathcal{G} \subseteq 2^{\mathscr{V}}$ and an option set $G \in \mathcal{G}$. If $0 \in \mathcal{N}(G)$, then $\mathbf{O}(\mathcal{G} \setminus \{G\}) = \mathbf{O}(\mathcal{G})$.

Proof. Follows straight from the definition $\mathbf{O}(\mathcal{G}) = \bigcup_{G \in \mathcal{G}} \mathbf{O}[G]$ and the fact that, due to Equation (16), $\mathbf{O}[G] = \emptyset$ when $0 \in \mathcal{N}(G)$. □

Algorithm 6 From assessment to conjunctive generator, with simplifications

Input: finite assessment $\mathcal{A} \Subset \mathscr{Q} \times \mathscr{Q}_\emptyset$
Output: $\mathcal{H} \Subset \mathscr{Q}_\emptyset$ such that $\mathbb{O}(\mathcal{H}) = \mathbb{O}(\mathcal{H}_\mathcal{A})$

```
function AssessmentToConjunctive(A)
    H ← ∅
    for all (V, W) ∈ A do
        for all w ∈ W do
            H ← ∅
            containsPositive ← false
            for all v ∈ V do
                if v − w > 0 then                                   ▷ Lemma 5.1
                    containsPositive ← true
                    break
                if v − w ≰ 0 then                                   ▷ Lemma 5.2
                    H ← H ∪ {v − w}
            if not containsPositive then
                if H = ∅ then
                    Warning: A is not consistent.                   ▷ Equation (17)
                    return {∅}
                H ← H ∪ {Max(H, optionOrd)}                         ▷ Corollary 5.7
    return H
```

The next condition is related to pairs of generator sets and is therefore a bit more intensive to check.

Lemma 6.2. Consider a set of option sets $\mathcal{G} \subseteq 2^\mathscr{V}$ and two distinct option sets $G_1, G_2 \in \mathcal{G}$. If $G_2 \subseteq \mathcal{N}(G_1)$, then $\mathbf{O}(\mathcal{G} \setminus \{G_1\}) = \mathbf{O}(\mathcal{G})$.

Proof. Since $G_2 \subseteq \mathcal{N}(G_1)$, we have $\mathbf{O}[G_1] \subseteq \mathbf{O}[G_2]$ by Lemmas 5.4 and 5.5. Because $\mathbf{O}(\mathcal{G}) = \bigcup_{G \in \mathcal{G}} \mathbf{O}[G]$, it follows that $\mathbf{O}(\mathcal{G}) = \mathbf{O}(\mathcal{G} \setminus \{G_1\})$. □

To show its relevance we will use an example with $\mathscr{V} = \mathbb{R}^3$.

Example 6.3. Consider the generator $\mathcal{G} = \{G_1, G_2\}$, where

$$G_1 = \{(2, 2, -1), (2, -1, 2), (-1, 2, 2)\},$$
$$G_2 = \{(5, -1, -1), (-1, 5, -1), (-1, -1, 5)\}.$$

It is fairly straightforward to verify that this generator cannot be simplified by Lemma 6.1. However, we do have that $G_1 \subseteq \text{posi}(G_2 \cup \mathscr{V}_{>0}) = \mathcal{N}(G_2)$; for example $(2, 2, -1) = \frac{1}{2}(5, -1, -1) + \frac{1}{2}(-1, 5, -1)$. It therefore follows from Lemma 6.2 that $\mathbf{O}(\{G_1, G_2\}) = \mathbf{O}(\{G_1\})$.

To automate this process we define the function G-Ord that returns true for a given pair of option sets $(G_1, G_2) \in 2^\mathscr{V} \times 2^\mathscr{V}$ whenever $G_2 \subseteq \mathcal{N}(G_1)$ and false otherwise. Since we know from Lemma 5.4 that this is equivalent to $\mathcal{N}(G_2) \subseteq \mathcal{N}(G_1)$, G-Ord defines a (non-strict) preorder $\preccurlyeq$ on $2^\mathscr{V}$—defined by $G_1 \preccurlyeq G_2 \Leftrightarrow$ G-Ord(G_1, G_2)—as it is clearly reflexive and transitive. This allows us to use Algorithm 5 once more, this time to select a minimal set Max$(\mathcal{G}, \text{G-Ord})$ of undominated elements from a generator $\mathcal{G}$, at least if $\mathcal{G}$ is finite and if we are able to evaluate G-Ord.[8]

Corollary 6.4. For any finite set of option sets $\mathcal{G} \Subset 2^\mathscr{V}$,

$$\mathbf{O}(\mathcal{G}) = \mathbf{O}(\text{Max}(\mathcal{G}, \text{G-Ord})).$$

[8]It should be noted that if we work with arrays, then duplicates can be present in the generator. In that case, only one of these duplicates will be kept because Algorithm 5 will retain exactly one option set from every equivalence class defined by $\preccurlyeq$.

Proof. This follows immediately by repeated application of Lemma 6.2. □

Evaluating G-ORD(G_1, G_2) for arbitrary $G_1, G_2 \in 2^{\mathscr{V}}$ is tricky, but if G_1 and G_2 are finite, then it can be evaluated as the conjunction of ISFEASIBLE(G_1, g) over all nonpositive options $g \in G_2 \setminus \mathscr{V}_{>0}$.

Lemma 6.5. Consider two finite option sets $G_1, G_2 \in \mathscr{Q}_\emptyset$. Then

$$\text{G-ORD}(G_1, G_2) \Leftrightarrow (\forall g \in G_2 \setminus \mathscr{V}_{>0}) \text{ ISFEASIBLE}(G_1, g).$$

Proof. G-ORD(G_1, G_2) is by definition true if and only if $G_2 \subseteq \mathcal{N}(G_1)$, or equivalently, if $g \in \mathcal{N}(G_1)$ for all $g \in G_2$. Due to Proposition 4.1 and the definition of ISFEASIBLE, this is equivalent to the condition that for every $g \in G_2$, we have that $g > 0$ or that ISFEASIBLE(G_1, g) is true. This is equivalent to the condition on the right. □

If an option set G_1 belongs to a generator $\mathcal{G}(\mathcal{H})$, where $\mathcal{H}$ is created using Algorithm 6, then there is no need to check if $g > 0$ because such g will have been removed based on Lemma 5.1.

6.2. Simplifying generator sets

In contrast to the previous section, we will now look at which options inside an individual option set of a generator do not contain any information. The following lemma provides a sufficient condition for two generator sets—one of which can be a subset of the other—to generate the same set of preference orders.

Lemma 6.6. For any two option sets $G, G' \in 2^{\mathscr{V}}$, if $\mathcal{N}(G) = \mathcal{N}(G')$, then also $\mathbf{O}[G] = \mathbf{O}[G']$.

Proof. This follows immediately from two applications of Lemma 5.5. □

Inspired by this result, we will try to simplify a generator set by replacing it with a smallest subset $G' \subseteq G$ for which $\mathcal{N}(G) = \mathcal{N}(G')$. That is, we want to come up with a function MINCONESUBSET: $\mathscr{Q}_\emptyset \to \mathscr{Q}_\emptyset$ that, for every $G \in \mathscr{Q}_\emptyset$, returns some subset $G' = \text{MINCONESUBSET}(G) \subseteq G$ for which $\mathcal{N}(G) = \mathcal{N}(G') \neq \mathcal{N}(G' \setminus \{u\})$ for all $u \in G'$. Due to Lemma 6.6, the resulting G' will then yield the same set of preferences.

Proposition 6.7. Consider an option set $G \in \mathscr{Q}_\emptyset$. Then

$$\mathbf{O}[G] = \mathbf{O}[\text{MINCONESUBSET}(G)].$$

Proof. This follows immediately from the properties that we require of MINCONESUBSET—i.e. $\mathcal{N}(G) = \mathcal{N}(\text{MINCONESUBSET}(G))$—and Lemma 6.6. □

A first way to obtain such a function is to use the following lemma to find such a G' iteratively.

Lemma 6.8. For any option set $G \in 2^{\mathscr{V}}$ and option $u \in G$, $u \in \mathcal{N}(G \setminus \{u\})$ if and only if $\mathcal{N}(G) = \mathcal{N}(G \setminus \{u\})$.

Proof. The implication to the left is immediate from $\mathcal{N}(G) = \text{posi}(G \cup \mathscr{V}_{>0})$, the definition of the posi operator with $n = 1$ and $u \in G$, so we prove the implication to the right.

On the one hand, since $\mathcal{N}(G \setminus \{u\}) \subseteq \mathcal{N}(G \setminus \{u\})$, Lemma 5.4 implies that $G \setminus \{u\} \subseteq \mathcal{N}(G \setminus \{u\})$. This together with $u \in \mathcal{N}(G \setminus \{u\})$ implies that $G \subseteq \mathcal{N}(G \setminus \{u\})$. Therefore, by Lemma 5.4, we have that $\mathcal{N}(G) \subseteq \mathcal{N}(G \setminus \{u\})$.

On the other hand, since $\mathcal{N}(G) \subseteq \mathcal{N}(G)$, Lemma 5.4 implies that $G \subseteq \mathcal{N}(G)$. Therefore, $G \setminus \{u\} \subseteq G \subseteq \mathcal{N}(G)$ and hence, by Lemma 5.4, $\mathcal{N}(G \setminus \{u\}) \subseteq \mathcal{N}(G)$. □

This can again be applied to our running example.

Running Example 6.9. Let us start from the conjunctive generator $\mathcal{H} = \{\{h_2\}, \{h_3\}, \{h_5\}\}$ that we found in Running Example 5.6 with $h_2 = (2, -1)$, $h_3 = (7, -4)$ and $h_5 = (-7, 7)$. The corresponding disjunctive generator is $\mathcal{G}(\mathcal{H}) = \{G\}$ with $G = \{h_2, h_3, h_5\}$. Since $4h_2 = (8, -4) > (7, -4) = h_3$, we know that $h_2 = \frac{1}{4}h_3 + \frac{1}{4}p$ for $p = (1, 0) \in \mathscr{V}_{>0}$, which implies that $h_2 \in \mathcal{N}(\{h_3, h_5\})$. We can therefore remove h_2 from the set G in this generator due to Lemmas 6.6 and 6.8, resulting in $G' = \{h_3, h_5\}$. This leads to a generator $\mathcal{G}' = \{G'\}$ that is equivalent to $\mathcal{G}(\mathcal{H})$, in the sense that $\mathbf{O}(\mathcal{G}') = \mathbf{O}(\mathcal{G}(\mathcal{H})) = \mathbb{O}(\mathcal{H})$. Since we already know from Running Example 5.6 that $\mathbb{O}(\mathcal{H}) = \mathbb{O}(\mathcal{H}_\mathcal{A}) = \mathbf{O}(\mathcal{G}_\mathcal{A})$, we find that instead of computing $C^{\mathcal{G}_\mathcal{A}} = C_{\mathbf{O}(\mathcal{G}_\mathcal{A})}$ to evaluate $C_\mathcal{A}$, as we did in Running Example 4.6, we can now instead evaluate $C^{\mathcal{G}'} = C_{\mathbf{O}(\mathcal{G}')}$. Since $\mathcal{G}'$ is smaller, we know from Algorithm 2 that this will be easier. ◊

To obtain a smallest subset G' of G such that $\mathcal{N}(G') = \mathcal{N}(G) \neq \mathcal{N}(G' \setminus \{u\})$ for all $u \in G'$, we can now keep on using Lemmas 6.6 and 6.8 to remove options from G until we cannot remove any more. This yields a method whose time complexity is polynomial in both the dimension of the state space and the number of options in G. To check the condition $u \in \mathcal{N}(G \setminus \{u\})$ in practice, due to Proposition 4.1 we can first check if $u > 0$—in which case it is always true, so u can be removed—and otherwise evaluate IsFeasible$(G \setminus \{u\}, u)$—removing u if it is true.

There are other ways to come up with a function MinConeSubset that has the required properties though, because it is related to a well-known problem in polyhedral theory: redundancy removal. To make the connection, we first need to introduce some definitions. For any set of options $V \subseteq \mathscr{V}$ the *conic hull* of V is defined as

$$\text{cone}(V) = \left\{ \sum_{j=1}^{n} \lambda_j v_j \,:\, n \in \mathbb{N}, v_j \in V, \lambda_j \geq 0 \right\},$$

and turns the set V into a (pointed) *cone*, which includes 0. This corresponds to [16, Eq. (2.6)] for finite V. An option $v \in V$ is called *redundant* if $\text{cone}(V \setminus \{v\}) = \text{cone}(V)$. Removing redundant options is therefore clearly a problem that is closely related to what is done in Lemma 6.8. If $V \in \mathscr{Q}_\emptyset$ is a finite set of options, then *redundancy removal* means finding a subset V^* of options so that V^* does not contain any redundant options and $\text{cone}(V) = \text{cone}(V^*)$, as described for example by Fukuda [16, Problem 7.2 and Section 7.4].

Let us now make the connection between redundancy removal and MinConeSubset more formal. It is clear that $\text{posi}(V) \cup \{0\} = \text{cone}(V)$ for any non-empty $V \in 2^\mathscr{V} \setminus \{\emptyset\}$. Therefore, we see that $\mathcal{N}(G) \cup \{0\} = \text{posi}(G \cup \mathscr{V}_{>0}) \cup \{0\} = \text{cone}(G \cup \mathscr{V}_{>0})$ is the conic hull of $G \cup \mathscr{V}_{>0}$. Hence, if $0 \notin \mathcal{N}(G)$—if it contained 0 then this G could be removed by Lemma 6.1—then $\mathcal{N}(G) = \text{cone}(G \cup \mathscr{V}_{>0}) \setminus \{0\}$. The following lemma uses this connection between the cone and the posi operator to establish a connection between redundancy removal and MinConeSubset.

Lemma 6.10. For any two option sets $G, G^* \in 2^\mathscr{V}$, if $0 \notin \mathcal{N}(G)$, $0 \notin G^*$ and $\text{cone}(G \cup \mathscr{V}_{>0}) = \text{cone}(G^*)$, then

(i) $\mathcal{N}(G) = \mathcal{N}(G^* \setminus \mathscr{V}_{>0})$;

(ii) for all $u \in G^* \setminus \mathscr{V}_{>0}$ such that $\text{cone}(G^*) \neq \text{cone}(G^* \setminus \{u\})$, we also have that $\mathcal{N}(G^* \setminus \mathscr{V}_{>0}) \neq \mathcal{N}((G^* \setminus \mathscr{V}_{>0}) \setminus \{u\})$.

We first prove the following intermediate result.

Lemma 6.11. For any two option sets $G, G^* \in 2^\mathscr{V}$, if $\mathcal{N}(G) = \text{posi}(G^*)$, then $\mathcal{N}(G) = \mathcal{N}(G^* \setminus \mathscr{V}_{>0})$.

Proof. Since $\mathcal{N}(G^* \setminus \mathscr{V}_{>0}) = \text{posi}((G^* \setminus \mathscr{V}_{>0}) \cup \mathscr{V}_{>0}) = \text{posi}(G^* \cup \mathscr{V}_{>0}) = \mathcal{N}(G^*)$, it suffices to prove that $\mathcal{N}(G) = \mathcal{N}(G^*)$.

First, we have from the definition of the posi operator that any option in $\text{posi}(G^*)$ must also be in $\text{posi}(G^* \cup \mathscr{V}_{>0})$, so $\mathcal{N}(G) = \text{posi}(G^*) \subseteq \text{posi}(G^* \cup \mathscr{V}_{>0}) = \mathcal{N}(G^*)$.

So it remains to prove that $\mathcal{N}(G^*) \subseteq \mathcal{N}(G)$. By Lemma 5.4, we know that this is equivalent to proving that $G^* \subseteq \mathcal{N}(G)$. But we have from the definition of the posi operator that $G^* \subseteq \text{posi}(G^*) = \mathcal{N}(G)$. □

Proof of Lemma 6.10 For the first part it suffices by Lemma 6.11 to prove that $\mathcal{N}(G) = \text{posi}(G^*)$. Since $0 \notin \mathcal{N}(G)$, we know from the main text preceding Lemma 6.10 that $\mathcal{N}(G) = \text{cone}(G \cup \mathscr{V}_{>0}) \setminus \{0\} = \text{cone}(G^*) \setminus \{0\}$. We will now prove that also $\text{posi}(G^*) = \text{cone}(G^*) \setminus \{0\}$. From the definitions of the posi and cone operators it can be seen that it is sufficient to prove that $0 \notin \text{posi}(G^*)$. Assume *ex absurdo* that $0 \in \text{posi}(G^*)$. Then there are $n \in \mathbb{N}$, $v_j \in G^*$, and $\lambda_j > 0$ such that $0 = \sum_{j=1}^n \lambda_j v_j$. It also follows from the definition of the cone operator and the assumptions in the statement that $G^* \subseteq \text{cone}(G^*) = \text{cone}(G \cup \mathscr{V}_{>0}) = \text{posi}(G \cup \mathscr{V}_{>0}) \cup \{0\}$, which, since $0 \notin G^*$, implies that $G^* \subseteq \text{posi}(G \cup \mathscr{V}_{>0})$. So for every $j \in \{1, ..., n\}$ there are $m_j \in \mathbb{N}$, $w_{j,i} \in G \cup \mathscr{V}_{>0}$, and $\mu_{j,i} > 0$ such that $v_j = \sum_{i=1}^{m_j} \mu_{j,i} w_{j,i}$. It therefore follows that $0 = \sum_{j=1}^n \lambda_j v_j = \sum_{j=1}^n \sum_{i=1}^{m_j} (\lambda_j \mu_{j,i}) w_{j,i} \in \text{posi}(G \cup \mathscr{V}_{>0}) = \mathcal{N}(G)$, which is a contradiction. Hence, $0 \notin \text{posi}(G^*)$ and therefore $\text{posi}(G^*) = \text{cone}(G^*) \setminus \{0\} = \mathcal{N}(G)$. It now follows from Lemma 6.11 that $\mathcal{N}(G) = \mathcal{N}(G^* \setminus \mathscr{V}_{>0})$.

We prove the second part by contraposition. Assume that $\mathcal{N}(G^* \setminus \mathscr{V}_{>0}) = \mathcal{N}((G^* \setminus \mathscr{V}_{>0}) \setminus \{u\})$ for some $u \in G^* \setminus \mathscr{V}_{>0}$. We'll show that then also $\text{cone}(G^*) = \text{cone}(G^* \setminus \{u\})$. For the set inclusion $\text{cone}(G^* \setminus \{u\}) \subseteq \text{cone}(G^*)$, take any $w \in \text{cone}(G^* \setminus \{u\})$, then clearly $w \in \text{cone}(G^*)$ by definition. We will now prove the other inclusion $\text{cone}(G^*) \subseteq \text{cone}(G^* \setminus \{u\})$ by first proving some intermediate things related to u.

Since $u \in G^* \setminus \mathscr{V}_{>0}$, it is clear from the definition of the posi operator that

$$\begin{aligned} u \in \text{posi}((G^* \setminus \mathscr{V}_{>0}) \cup \mathscr{V}_{>0}) = \mathcal{N}(G^* \setminus \mathscr{V}_{>0}) &= \mathcal{N}((G^* \setminus \mathscr{V}_{>0}) \setminus \{u\}) \\ &= \text{posi}(((G^* \setminus \mathscr{V}_{>0}) \setminus \{u\}) \cup \mathscr{V}_{>0}) \\ &= \text{posi}((G^* \setminus \{u\}) \cup \mathscr{V}_{>0}). \end{aligned}$$

Therefore, there are $n \in \mathbb{N}$, $v_j \in (G^* \setminus \{u\}) \cup \mathscr{V}_{>0}$ and $\lambda_j > 0$ such that $u = \sum_{j=1}^n \lambda_j v_j$. Without loss of generality, there is some $k \in \{0, ..., n\}$ such that $v_j \in G^* \setminus \{u\}$ for all $j \leq k$ and $v_j \in \mathscr{V}_{>0}$ for all $j > k$. Then we have that $u = \sum_{j=1}^k \lambda_j v_j + \sum_{j=k+1}^n \lambda_j v_j$. We have that $p \coloneqq \sum_{j=k+1}^n \lambda_j v_j$ is either equal to 0, when $k = n$, or belongs to $\mathscr{V}_{>0}$ by the scaling and translation property of $>$.

If $p \in \mathscr{V}_{>0}$, then p clearly belongs to $\text{posi}(G \cup \mathscr{V}_{>0}) = \mathcal{N}(G)$, by definition op the posi operator, and therefore also belongs to $\text{posi}(G^*)$ since we already proved in the first part that $\text{posi}(G^*) = \mathcal{N}(G)$. Since $p \in \text{posi}(G^*)$ and because we can rearrange and lump terms together, there are then $\ell \in \mathbb{N}$ with $\ell \geq 2$, $\nu_1 \geq 0$, $y_r \in G^* \setminus \{u\}$ and $\nu_r > 0$ for all $r \geq 2$, such that $p = \nu_1 u + \sum_{r=2}^\ell \nu_r y_r$, where ℓ is at least 2 since $u \notin \mathscr{V}_{>0}$. If $p = 0$, we simply let $p = \nu_1 u$ with $\nu_1 = 0$. We always have that $\nu_1 < 1$, trivially for $p = 0$ and, for $p \in \mathscr{V}_{>0}$, because otherwise $0 \in \text{posi}(G^*) = \mathcal{N}(G)$ because $0 = \sum_{j=1}^k \lambda_j v_j + (\nu_1 - 1)u + \sum_{r=2}^\ell \nu_r y_r$, which is not true by assumption. Therefore, in any case, there are $\ell \in \mathbb{N}$, $\nu_1 \geq 0$ with $\nu_1 < 1$, $y_r \in G^* \setminus \{u\}$ and $\nu_r > 0$ for all $r \geq 2$, such that $p = \nu_1 u + \sum_{r=2}^\ell \nu_r y_r$—if $\ell = 1$, the empty sum in this expression is 0. So, we have that $u = \sum_{j=1}^k \frac{\lambda_j}{1-\nu_1} v_j + \sum_{r=2}^\ell \frac{\nu_r}{1-\nu_1} y_r$ with $v_j, y_r \in G^* \setminus \{u\}$ for all $j \in \{1, ..., k\}$ and $r \in \{2, ..., \ell\}$.

Now, take any $w \in \text{cone}(G^*)$. Then, by definition, there are $m \in \mathbb{N}$, $z_i \in G^*$ and $\mu_i \geq 0$ such that $w = \sum_{i=1}^m \mu_i z_i$. Assume without loss of generality that $z_1 = u$ is the only appearance of u—as we can always reorder the terms in the sum, lump them together or add a term with a zero coefficient—then we have that $w = \sum_{j=2}^m \mu_j z_j + \mu_1 u = \sum_{j=2}^m \mu_j z_j + \sum_{j=1}^k \frac{\mu_1 \lambda_j}{1-\nu_1} v_j + \sum_{r=2}^\ell \frac{\mu_1 \nu_r}{1-\nu_1} y_r$. So, $w \in \text{cone}(G^* \setminus \{u\})$ by definition. □

Now if $G \cup \mathscr{V}_{>0}$ were finite, then this result would allow us to implement MinConeSubset using redundancy removal, applying it to reduce $G \cup \mathscr{V}_{>0}$ to the set G^* that appears in Lemma 6.10.

However, $G \cup \mathscr{V}_{>0}$ is always infinite because $\mathscr{V}_{>0}$ is. We resolve this by replacing $\mathscr{V}_{>0}$ with the set $\{\mathbb{I}_x \colon x \in \mathcal{X}\}$, where, for every $x \in \mathcal{X}$, the standard basis vector $\mathbb{I}_x \colon \mathcal{X} \to \mathbb{R}$ maps x to 1 and all other states in $\mathcal{X}$ to 0.

Lemma 6.12. Let the state space $\mathcal{X}$ be finite. For any option set $G \in 2^{\mathscr{V}}$, we have that $\text{cone}(G \cup \mathscr{V}_{>0}) = \text{cone}(G \cup \{\mathbb{I}_x \colon x \in \mathcal{X}\})$.

Proof. Take any $u \in \text{cone}(G \cup \mathscr{V}_{>0})$. Then by definition there are $n \in \mathbb{N}$, $v_j \in G \cup \mathscr{V}_{>0}$ and $\lambda_j \geq 0$ such that $u = \sum_{j=1}^n \lambda_j v_j$. Without loss of generality we can assume that there is some $m \in \{0, ..., n\}$ such that $v_j \in G$ for all $j \leq m$ and $v_j \in \mathscr{V}_{>0}$ for all $j > m$. Then we have that $u = \sum_{j=1}^m \lambda_j v_j + \sum_{j=m+1}^n \lambda_j v_j$. So, if we let $p \coloneqq \sum_{j=m+1}^n \lambda_j v_j$, then p maps every x to a nonnegative number $p(x)$ and $p = \sum_{x \in \mathcal{X}} p(x) \mathbb{I}_x$. Hence, $u = \sum_{j=1}^m \lambda_j v_j + \sum_{x \in \mathcal{X}} p(x) \mathbb{I}_x$ belongs to $\text{cone}(G \cup \{\mathbb{I}_x \colon x \in \mathcal{X}\})$.

Now take any $u \in \operatorname{cone}(G \cup \{\mathbb{I}_x \colon x \in \mathcal{X}\})$. Then by definition there are $n \in \mathbb{N}$, $v_j \in G \cup \{\mathbb{I}_x \colon x \in \mathcal{X}\}$ and $\lambda_j \geq 0$ such that $u = \sum_{j=1}^{n} \lambda_j v_j$. Since $v_j \in G \cup \{\mathbb{I}_x \colon x \in \mathcal{X}\} \subseteq G \cup \mathscr{V}_{>0}$, it follows that $u \in \operatorname{cone}(G \cup \mathscr{V}_{>0})$. □

For any $G \in \mathscr{Q}_\emptyset$ such that $0 \notin \mathcal{N}(G)$, we can use this result to implement $G' = \textsc{MinConeSubset}(G)$ as follows: we first apply redundancy removal to find a minimal subset G^* of $G \cup \{\mathbb{I}_x \colon x \in \mathcal{X}\}$ for which $\operatorname{cone}(G^*) = \operatorname{cone}(G \cup \{\mathbb{I}_x \colon x \in \mathcal{X}\})$ and then remove the positive options from G^* to obtain $G' = G^* \setminus \mathscr{V}_{>0}$. This is always possible because $G \cup \{\mathbb{I}_x \colon x \in \mathcal{X}\}$ is finite. To see why it works, we first observe that, due to Lemma 6.12, $\operatorname{cone}(G^*) = \operatorname{cone}(G \cup \mathscr{V}_{>0})$. Furthermore, since $0 \notin \mathcal{N}(G)$, we automatically have that $0 \notin G^*$ because $G^* \subseteq G \cup \{\mathbb{I}_x \colon x \in \mathcal{X}\}$ and $0 \notin G$ since $0 \notin \mathcal{N}(G)$. Using Lemma 6.10 (i), this already implies that $\mathcal{N}(G) = \mathcal{N}(G')$. Furthermore, since the minimality of G^* implies that $\operatorname{cone}(G^*) \neq \operatorname{cone}(G^* \setminus \{u\})$ for all $u \in G^*$, we know from Lemma 6.10 (ii) that $\mathcal{N}(G') \neq \mathcal{N}(G' \setminus \{u\})$ for all $u \in G'$. So we see that G' indeed satisfies the properties required of $\textsc{MinConeSubset}(G)$.

In practice, to find a minimal subset G^* of $G \cup \{\mathbb{I}_x \colon x \in \mathcal{X}\}$ such that $\operatorname{cone}(G^*) = \operatorname{cone}(G \cup \{\mathbb{I}_x \colon x \in \mathcal{X}\})$, we can for example use the function `removevredundancy` of the Julia library Polyhedra.jl [19]. Alternatively, such a minimal subset free of redundant options can also be found by calculating the dual representation, for example using the Quickhull [14] or the Double Description [15] methods, but the time complexity of these algorithms scale exponentially with the size of the state space; such methods are therefore usually only used for 2- and 3-dimensional vector spaces.

To summarise, it is possible to implement $\textsc{MinConeSubset}$ directly using $\textsc{IsFeasible}$, but it is also possible to use redundancy removal methods from polyhedral theory. In our experiments in Section 7, we will implement $\textsc{MinConeSubset}$ using `removevredundancy` from Polyhedra.jl, but everything works regardless of which approach is used.

6.3. Simplifying a generator

Having presented several techniques to simplify a disjunctive generator $\mathcal{G}$, Algorithm 7 now combines them into a single algorithm. For every $G \in \mathcal{G}$, we first check if $0 \in \mathcal{N}(G)$ because if this is the case, we know from Lemma 6.1 that we can remove it. As we know from Proposition 4.1, this happens if $\textsc{IsFeasible}(G, 0)$ is true. If this is not the case, we simplify G by replacing it with $\textsc{MinConeSubset}(G)$, as allowed by Proposition 6.7. Finally, we take the maximal elements of the generator with respect to G-$\textsc{Ord}$, which is allowed by Corollary 6.4.

Algorithm 7 Simplify a disjunctive generator

Input: finite disjunctive generator $\mathcal{G} \Subset \mathscr{Q}_\emptyset$.
Output: finite disjunctive generator $\mathcal{G}'$ for which $\mathbf{O}(\mathcal{G}') = \mathbf{O}(\mathcal{G})$

```
function Simplify(𝒢)
    𝒢' ← ∅
    for all G ∈ 𝒢 do
        if not IsFeasible(G, 0) then                    ▷ Lemma 6.1
            𝒢' ← 𝒢' ∪ {MinConeSubset(G)}                ▷ Proposition 6.7
    𝒢' ← Max(𝒢', G-Ord)                                 ▷ Corollary 6.4
    return 𝒢
```

6.4. From conjunctive generator to disjunctive generator: step by step

Algorithm 7 can be applied to any disjunctive generator, but in particular we will often be interested in applying it to a generator of the form $\mathcal{G}(\mathcal{H})$. The naive way to implement this would be to first create the generator $\mathcal{G}(\mathcal{H})$ and then use Algorithm 7 to simplify it.

This is however not efficient. Since the size of $\mathcal{G}(\mathcal{H})$ is exponential in the size of $\mathcal{H}$, the mere act of saving it in memory would already use a lot of resources (both time and memory). Fortunately though, there is no need to construct $\mathcal{G}(\mathcal{H})$ if all we want is to find a small $\mathcal{G}$ such that $\mathbf{O}(\mathcal{G}) = \mathbf{O}(\mathcal{G}(\mathcal{H}))$. Instead, we can construct such a simplified generator $\mathcal{G}$ iteratively from $\mathcal{H}$. Let us enumerate the option sets in $\mathcal{H}$ as

$H_1, ..., H_m$ and, for each $k = 0, ..., m$, let $\mathcal{H}_{1:k} = \{H_1, ..., H_k\}$. The idea is now to construct a generator $\mathcal{G}_k$ such that $\mathbf{O}(\mathcal{G}_k) = \mathbf{O}(\mathcal{G}(\mathcal{H}_{1:k}))$, for increasing values of $k = 1, ..., m$. For $k = m$, since $\mathcal{H}_{1:m} = \mathcal{H}$, we will then find the set $\mathcal{G} = \mathcal{G}_m$ that we are after.

We start from $\mathcal{G}_0 = \{\emptyset\}$ because $\mathbf{O}(\{\emptyset\}) = \mathbf{O}(\mathcal{G}(\emptyset)) = \mathbb{O}(\emptyset) = \mathbb{O}$ cf. Footnote 4. Next, for each $k \in \{1, ..., m\}$, we construct $\mathcal{G}_k$ from $\mathcal{G}_{k-1}$ and H_k, in such a way that $\mathbf{O}(\mathcal{G}_k) = \mathbf{O}(\mathcal{G}(\mathcal{H}_{1:k}))$. The following result shows that one way to achieve this is by letting $\mathcal{G}_k \coloneqq \{G \cup \{h\} \colon G \in \mathcal{G}_{k-1}, h \in H_k\}$.

Proposition 6.13. Let $\mathcal{H} \subseteq \mathscr{Q}_\emptyset$ be a conjunctive generator, $\mathcal{G}$ a generator such that $\mathbf{O}(\mathcal{G}) = \mathbf{O}(\mathcal{G}(\mathcal{H}))$ and consider any option set $H \in \mathscr{Q}_\emptyset$. Then

$$\mathbf{O}(\mathcal{G}(\mathcal{H} \cup \{H\})) = \mathbf{O}(\{G \cup \{h\} \colon G \in \mathcal{G}, h \in H\}).$$

Proof. For every $G \in \mathcal{G}$ and $h \in H$ we have that $\mathbf{O}[G \cup \{h\}] = \mathbf{O}[G] \cap \mathbb{O}_h$. Therefore, we have that

$$\begin{aligned}
\mathbf{O}(\{G \cup \{h\} \colon G \in \mathcal{G}, h \in H\}) &\overset{(12)}{=} \bigcup_{G \in \mathcal{G}} \bigcup_{h \in H} \mathbf{O}[G \cup \{h\}] = \bigcup_{G \in \mathcal{G}} \bigcup_{h \in H} (\mathbf{O}[G] \cap \mathbb{O}_h) \\
&= \bigcup_{G \in \mathcal{G}} \left(\mathbf{O}[G] \cap \left(\bigcup_{h \in H} \mathbb{O}_h \right) \right) \\
&= \left(\bigcup_{G \in \mathcal{G}} \mathbf{O}[G] \right) \cap \left(\bigcup_{h \in H} \mathbb{O}_h \right) \\
&\overset{(12),(6)}{=} \mathbf{O}(\mathcal{G}) \cap \mathbb{O}[H] = \mathbf{O}(\mathcal{G}(\mathcal{H})) \cap \mathbb{O}[H] \\
&\overset{(10),(12)}{=} \mathbb{O}(\mathcal{H}) \cap \mathbb{O}[H] \overset{(7)}{=} \bigcap_{H^* \in \mathcal{H} \cup \{H\}} \mathbb{O}[H^*] \\
&\overset{(7)}{=} \mathbb{O}(\mathcal{H} \cup \{H\}) \overset{(10),(12)}{=} \mathbf{O}(\mathcal{G}(\mathcal{H} \cup \{H\})).
\end{aligned}$$

□

However, this way of defining $\mathcal{G}_k$ would simply amount to using Algorithm 4, without any simplifications, yielding $\mathcal{G}(\mathcal{H})$. Fortunately, however, this iterative approach allows us to apply our simplifications to every intermediate $\mathcal{G}_k$. That is, for every k, we can let $\mathcal{G}_{k,\text{unsimp}} \coloneqq \{G \cup \{h\} \colon G \in \mathcal{G}_{k-1}, h \in H_k\}$ and then simplify it to obtain $\mathcal{G}_k = \textsc{Simplify}(\mathcal{G}_{k,\text{unsimp}})$. Due to Proposition 6.13 and Algorithm 7, we then still have that $\mathbf{O}(\mathcal{G}_k) = \mathbf{O}(\mathcal{G}_{k,\text{unsimp}}) = \mathbf{O}(\mathcal{G}(\mathcal{H}_{1:k}))$. This would however require us to loop twice over all elements of $\mathcal{G}_{k,\text{unsimp}}$: once to construct it and once to simplify it. We can avoid this by constructing $\mathcal{G}_k$ directly from $\mathcal{G}_{k-1}$ and H_k and doing the simplifications along the way: for every $G \cup \{h\}$ that we construct, we should not include it in $\mathcal{G}_{k,\text{unsimp}}$ if we will remove it anyway in the next step using Lemma 6.1, and if we include it, we might as well already replace it with $\textsc{MinConeSubset}(G \cup \{h\})$ if we are going to replace it by that anyway in the next loop in the simplification of $\mathcal{G}_{k,\text{unsimp}}$. We have implemented this in Algorithm 8.

7. Experiments

Having introduced a number of different algorithms, we now proceed to evaluate their performance. In particular, we will investigate when our algorithms can do the following efficiently: to start from a finite assessment $\mathcal{A} \subseteq \mathscr{Q} \times \mathscr{Q}_\emptyset$, construct a generator $\mathcal{G}$ such that $\mathbf{O}(\mathcal{G}) = \mathbf{O}(\mathcal{G}_\mathcal{A})$ and use this generator to evaluate the corresponding choice function $C^\mathcal{G} = C_\mathcal{A}$ for a given option set $A \in \mathscr{Q}$ containing options from which we want to choose. First, since we have provided multiple methods to process and simplify the information contained in an assessment, we will investigate to see whether these simplifications are actually useful or whether it is perhaps sometimes faster to skip these preprocessing steps and evaluate $C_\mathcal{A}$ directly. Second, since a lot of variables determine the speed of the algorithms—for both processing the assessments and making the decisions—we want to test how the performance changes based on the type of assessment. The variables that we will focus on are the size of the assessment and the amount of imprecision in the assessment.

Algorithm 8 From conjunctive to disjunctive generator, with simplifications

Input: finite conjunctive generator $\mathcal{H} \Subset \mathscr{Q}_\emptyset$
Output: finite disjunctive generator $\mathcal{G}$ for which $\mathbf{O}(\mathcal{G}) = \mathbf{O}(\mathcal{G}(\mathcal{H}))$

```
function ConjunctiveToDisjunctive(H)
    G ← {∅}
    for all H ∈ H do                                              ▷ Proposition 6.13
        G* ← ∅
        for all G ∈ G do
            for all h ∈ H do
                if not IsFeasible(G ∪ {h}, 0) then                ▷ Lemma 6.1
                    G* ← G* ∪ {MinConeSubset({G ∪ {h}})}
                                                                  ▷ Proposition 6.7
        G ← Max(G*, G-Ord)                                        ▷ Corollary 6.4
    return G
```

Since an assessment $\mathcal{A}$ that is not consistent gives rise to an uninteresting operator $C_{\mathcal{A}}$—in that case, as we know from Proposition 2.4, $C_{\mathcal{A}}(A) = \emptyset$ for all $A \in \mathscr{Q}$—we will focus on the case where $\mathcal{A}$ is consistent.

In our first two experiments, we will investigate how fast the number of option sets in the generator $\mathcal{G}$ grows as the size of the assessment increases, for different types of assessments. The first experiment focusses on the type of decision rule that is used to generate the assessments, whereas the second focusses on the amount of imprecision. The reason this is of interest is that for large $\mathcal{G}$, evaluating the natural extension $C^{\mathcal{G}}$ with Algorithm 2 will require many evaluations of IsFeasible, each of which requires solving a linear program. Finally, in our third set of experiments, we will measure the actual time it takes to preprocess and then evaluate a choice function.

7.1. Set-up of the experiments

To test the algorithms, we will start from consistent assessments. To get such consistent assessments, we will consider a coherent choice function C and a sequence of random option sets $A_1, ..., A_L \in \mathscr{Q}$, and use these to define the assessment as

$$\mathcal{A}(C, A_1, ..., A_L) \coloneqq \{(C(A_\ell), A_\ell \setminus C(A_\ell))\colon \ell \in \{1, ..., L\}\}.$$

We will also use the shorthand notation $\mathcal{A} \coloneqq \mathcal{A}(C, A_1, ..., A_L)$ if the choice function and option sets are clear from the context. To study the effect of the size of the assessment, we will often increase the size of the assessments one by one. To that end, for all $\ell \in \{1, ..., L\}$, we let

$$\mathcal{A}_{1:\ell} \coloneqq \mathcal{A}(C, A_1, ..., A_\ell) = \{(C(A_k), A_k \setminus C(A_k))\colon k \in \{1, ..., \ell\}\},$$

with then $\mathcal{A}_{1:L} = \mathcal{A}$. Since our assessments come in the form of partial information about a coherent choice function, they can trivially be extended to a coherent choice function and are therefore indeed consistent. The natural extension of such an assessment will then be a conservative approximation of the choice function C that is used to construct the assessment, where the approximation becomes less conservative—since it is based on more information—as ℓ increases.

The coherent choice functions that we will use to construct our assessments are instances of $C_{\mathcal{E}}$, as introduced in Section 2.1, where $\mathcal{E}$ consists of a finite number of lower expectations $\underline{\mathrm{E}}$ each of which is characterised by a finite number of extreme probability mass functions. If $p_1, ..., p_n$ are the extreme probability mass functions of one such $\underline{\mathrm{E}}$, then as we explained in Section 2.1, we can easily compute $\underline{\mathrm{E}}(u) = \min_{k \in \{1,...,n\}} \sum_{x \in \mathcal{X}} p_k(x)u(x)$ for every $u \in \mathscr{V}$. This implies that

$$C_{\mathcal{E}}(A) = \{u \in A\colon (\exists \underline{\mathrm{E}} \in \mathcal{E})(\forall v \in A)\underline{\mathrm{E}}(v - u) \leq 0 \text{ and } u \not\prec v\}$$

can be efficiently evaluated as well, for every $A \in \mathscr{Q}$, enabling us to create the aforementioned consistent assessments.

We will do this for four different types of $\mathcal{E}$'s:

1. the linear case, where $\mathcal{E} = \{\mathrm{E}\}$ contains one single linear expectation; we will also use the notation $C_{\mathrm{lin}} = C_{\{\mathrm{E}\}}$ and associate it with the colour lime ■;
2. maximality, where $\mathcal{E} = \{\underline{\mathrm{E}}\}$ contains one single lower expectation; we will also use the notation $C_{\max} = C_{\{\underline{\mathrm{E}}\}}$ and associate it with the colour magenta ■;
3. E-admissibility, where $\mathcal{E} = \{\mathrm{E}_1, \mathrm{E}_2, \mathrm{E}_3\}$ contains three linear expectations; we will also use the notation $C_{\mathrm{adm}} = C_{\{\mathrm{E}_1,\mathrm{E}_2,\mathrm{E}_3\}}$ and associate it with the colour blue ■;
4. another viable imprecise coherent choice function, where $\mathcal{E} = \{\underline{\mathrm{E}}_1, \underline{\mathrm{E}}_2, \underline{\mathrm{E}}_3\}$ contains three lower expectation; we will also use the notation $C_{\mathrm{imp}} = C_{\{\underline{\mathrm{E}}_1,\underline{\mathrm{E}}_2,\underline{\mathrm{E}}_3\}}$ and associate it with the colour indigo ■.

To generate the (lower) expectations that make up these $\mathcal{E}$'s, we use Algorithm 2 in [21], which works by randomly generating a finite number of probability mass function by means of Algorithm 1 in [21]. The parameters to be set for this algorithm are the state space (4-dimensional in our case) and the number of probability mass functions for each lower expectation (we use 4). We also take $L = 30$ and generate option sets $A_1, ..., A_L$ that contain a number of options between 2 and 8. The number of options in these option sets is drawn uniformly random from $\{2, 3, ..., 8\}$. The options themselves are drawn uniformly random from the unit cube $[0, 1]^4$. Note that this choice of domain is not restrictive, as any option set can be rescaled and shifted by the same vector to an option set with options inside the unit cube while still carrying the same information. This is because Axioms $\prec_2$ and $\prec_3$ gives us that $w \prec v$ if and only if $\lambda w + u \prec \lambda w + u$ for any $\prec \in \mathbb{O}$, $u, v, w \in \mathscr{V}$ and $\lambda > 0$. So if we use the notation $\lambda A + u \coloneqq \{\lambda a + u \colon a \in A\}$ for any $A \in \mathscr{Q}_\emptyset$, $u \in \mathscr{V}$ and $\lambda > 0$, then we have for a given assessment $\mathcal{A}$ and rescaled assessment $\mathcal{A}' = \{(\lambda V + u, \lambda W + u) \colon (V, W) \in \mathcal{A}\}$ that, for any $\prec \in \mathbb{O}$,

$$\begin{aligned}(\forall (V, W) \in \mathcal{A})(\forall w \in W)(\exists v \in V) w \prec v \\ \Leftrightarrow (\forall (V, W) \in \mathcal{A})(\forall w \in W)(\exists v \in V) \lambda w + u \prec \lambda v + u \\ \Leftrightarrow (\forall (V', W') \in \mathcal{A}')(\forall w' \in W')(\exists v' \in V') w' \prec v'.\end{aligned}$$

Therefore, by Proposition 3.4, $\mathbb{O}(\mathcal{A}) = \mathbb{O}(\mathcal{A}')$, implying that the consistency and natural extension of $\mathcal{A}$ is the same as that of $\mathcal{A}'$.

For each of the consistent assessments $\mathcal{A}_{1:\ell}$ that are constructed in this way, we will evaluate its natural extension with Algorithm 2, using 3 different methods to obtain a disjunctive generator $\mathcal{G}$:

1. the straightforward way of constructing the conjunctive generator $\mathcal{H}_\times$ from $\mathcal{A}_{1:\ell}$ using Algorithm 3 and then constructing $\mathcal{G}_\times \coloneqq \mathcal{G}(\mathcal{H}_\times)$ whenever necessary using Algorithm 4, without saving it in memory; the results are depicted by $^\times$ in our plots further on,
2. the intermediate method of constructing a conjunctive generator $\mathcal{H}_\triangle$ from $\mathcal{A}_{1:\ell}$ using Algorithm 6 and then constructing $\mathcal{G}_\triangle \coloneqq \mathcal{G}(\mathcal{H}_\triangle)$ whenever necessary using Algorithm 4, without saving it in memory; depicted by $^\triangle$,
3. the full simplification method of constructing a conjunctive generator $\mathcal{H}_\square = \mathcal{H}_\triangle$ from $\mathcal{A}_{1:\ell}$ using Algorithm 6 and then constructing $\mathcal{G}_\square$ using Algorithm 8 and saving it in memory; depicted by $\square$.

For the first two methods, we take the assessment $\mathcal{A}_{1:\ell}$ and construct the corresponding conjunctive generator $\mathcal{H}_\times$ and $\mathcal{H}_\triangle$. The size of the corresponding disjunctive generators, i.e. the number of option sets in these generators, can then be calculated using the formula $|\mathcal{G}| = \prod_{H \in \mathcal{H}} |H|$, at least if—as we do—we do not check for duplicates and store these sets as arrays. To evaluate the choice functions $C^{\mathcal{G}_\times}$ and $C^{\mathcal{G}_\triangle}$ using

$\{\mathrm{E}\}$ (lin.)	$\{\underline{\mathrm{E}}\}$ (max.)	$\{\mathrm{E}_1, \mathrm{E}_2, \mathrm{E}_3\}$ (E-adm.)	$\{\underline{\mathrm{E}}_1, \underline{\mathrm{E}}_2, \underline{\mathrm{E}}_3\}$ (imp.)
1.000	2.142	1.744	2.780

Table 1: The average size of $V_\ell = C_{\mathcal{E}}(A_\ell)$ in the pairs $(V_\ell, W_\ell) \in \mathcal{A}$, averaged over all pairs and all assessments for different $\mathcal{E}$.

Algorithm 2, we loop over the respective generators $\mathcal{G}_\times$ and $\mathcal{G}_\triangle$ without ever fully saving them in memory. As soon as an option set in the generator has been checked in Algorithm 2, we can forget about it. For the third method, we explicitly construct the disjunctive generator $\mathcal{G}_\square$ and save it in memory, before running Algorithm 2, because we need the full generator anyway in Algorithm 8.

As for the software we used: as programming language we used Julia, and for the linear programs we used the CPLEX solver.

7.2. Varying the 'real' model and size of the assessment

We start by studying the size of the constructed disjunctive generator as a function of the number ℓ of pairs in the assessment $\mathcal{A}_{1:\ell}$. The results are depicted in Figure 1. We consider 7 random assessments $\mathcal{A}_{1:L}$ and depict the average size of the generator over these assessments. As explained in Section 7.1, for the first two methods, the size of the generators can easily be obtained from the conjunctive generators using the formula $|\mathcal{G}| = \prod_{H \in \mathcal{H}} |H|$, whereas for the third method we explicitly construct the generator $\mathcal{G}_\square$ to determine its size.

In the linear case with a choice function C_{lin}, we found that every option set $C_{\mathrm{lin}}(A_k)$ in the assessment was a singleton and the conjunctive generators $\mathcal{H}_\times$ and $\mathcal{H}_\triangle$ were therefore just collections of singletons, resulting in generators $\mathcal{G}_\times$, $\mathcal{G}_\triangle$ and $\mathcal{G}_\square$ consisting of only a single option set. In the other cases—for $C_{\max}, C_{\mathrm{adm}}$ and C_{imp}—the option sets in $\mathcal{H}$ were not singletons any more. In these cases, we do however see comparable evolutions in all curves. As can be seen, for the first two methods (depicted by respectively $\times$ and $\triangle$) the size of $\mathcal{G}$ grows exponentially, but the reduction of each subsequent simplification is considerable.

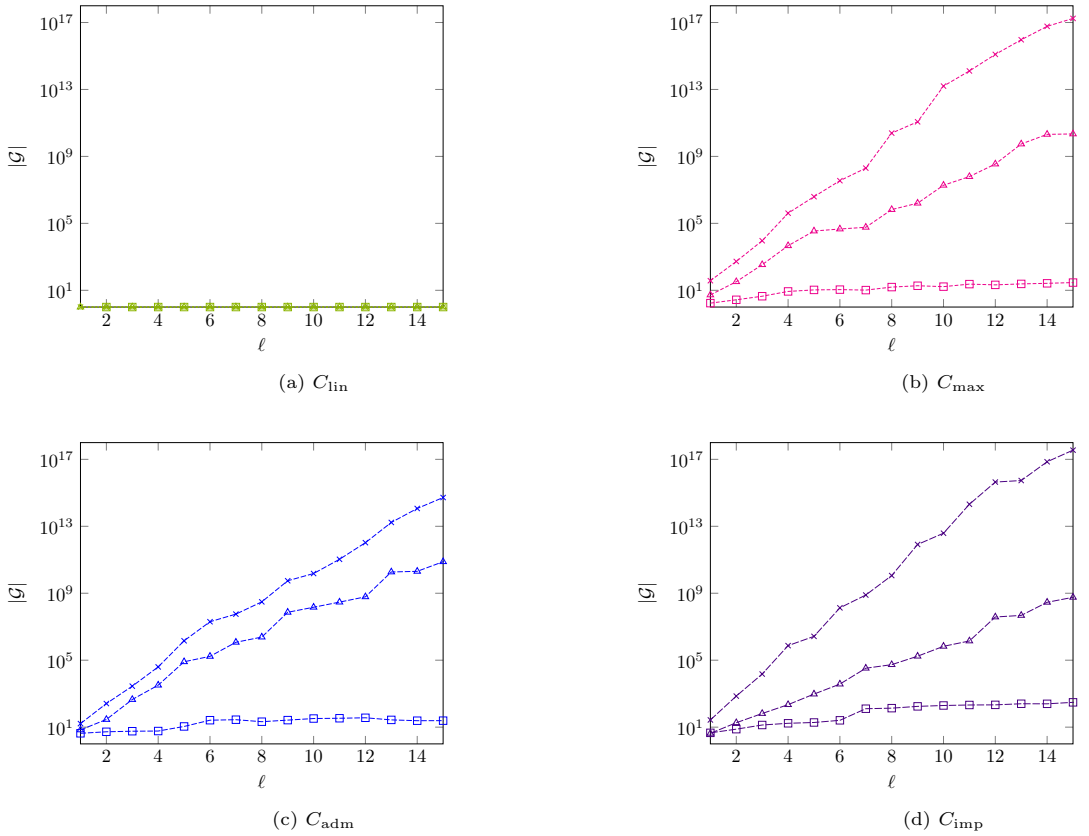
To compare the different $\mathcal{E}$'s we have Figure 2, where we split the graphs according to the simplification method instead. The slope for the first method, depicted by $\times$, seems to be slightly lower for C_{adm} than for $C_{\max}$ and C_{imp}. Looking at Table 1, the explanation for this is that V is smaller in that case on average. So in our experiments, more is rejected by E-admissibility with three randomly chosen mass functions than by maximality with a single lower expectation that is derived from four randomly chosen mass functions. It is however peculiar that the evolutions of $C_{\max}$ and C_{imp} are so close to each other. This seems to indicate that the lower number of rejections of C_{imp}—which means less option sets in the conjunctive generator—compensates for the higher number of options in each option set in the conjunctive generator.

Examining the simplifications of the third method, depicted by the squares ($\square$) in Figure 2, we see that the number of option sets in the simplified generator ($\mathcal{G}_\square$) is always relatively small. Figure 3 depicts the average size of $\mathcal{G}_\square$ over 7 experiments as a function of the length of the assessment for each of the four generating choice functions. In this figure we also extend the range to $\ell = 30$ pairs of option sets. There we see that for C_{imp} this seems to increase relatively rapidly while for C_{adm} and $C_{\max}$ it seems to stabilise. For C_{imp}, in the worst case of the 7 experiments $\mathcal{G}_\square$ contains a maximum of 45 714 option sets, which was for the full assessment of length 30, and in the best case it contains a maximum of 4 925 option sets, again for the full assessment, which is still a lot higher than for the other $\mathcal{E}$'s. For the other $\mathcal{E}$'s the size of $\mathcal{G}_\square$ stabilises and eventually even starts to decrease with increasing ℓ.

7.3. Varying the levels of imprecision

In the second experiment we look at ϵ-contaminations. These are lower expectations that are mixtures of a precise expectation with the so-called 'vacuous expectation', which we then use to choose with maximality. For any linear expectation E and an $\epsilon \in [0, 1]$, we can make an ϵ-contamination with the vacuous model as for example described in [9, Section 4.7.3], which leads to the corresponding lower expectation

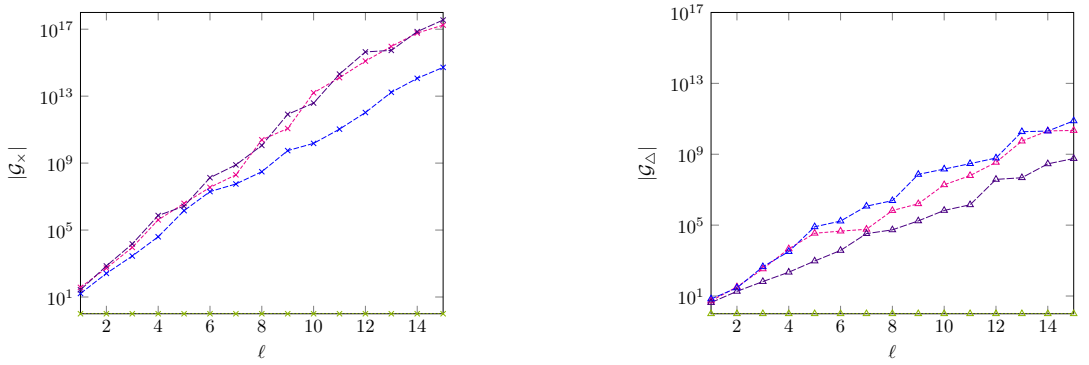
$$\underline{\mathrm{E}}^\epsilon \colon \mathscr{V} \to \mathbb{R} \colon u \mapsto (1 - \epsilon) E(u) + \epsilon \min u.$$



(a) C_{lin} (b) C_{max}

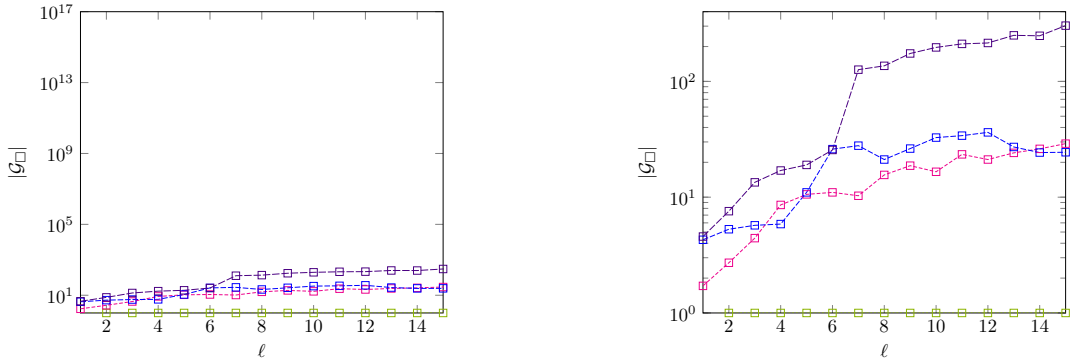
(c) C_{adm} (d) C_{imp}

Figure 1: Comparison of the number of option sets in the disjunctive generator of an assessment $\mathcal{A}_{1:\ell}$ of size ℓ (horizontal axis), each pair $(V, W) \in \mathcal{A}_{1:\ell}$ containing between two and eight options in total, using the first method ($\times$), the second method ($\triangle$) or the third method ($\square$). The results are the average of 7 individual experiments and plot with logarithmic vertical axes.



(a) first method

(b) second method



(c) third method

(d) third method (vertically zoomed)

Figure 2: Comparison of the number of option sets in the disjunctive generator of an assessment $\mathcal{A}_{1:\ell}$ of size ℓ (horizontal axis), each pair $(V, W) \in \mathcal{A}_{1:\ell}$ containing between two and eight options in total, and constructed using C_{lin} (-●-), C_{max} (-●-), C_{adm} (-●-) or C_{imp} (-●-). The results are the average of 7 individual experiments and plot with logarithmic vertical axes.

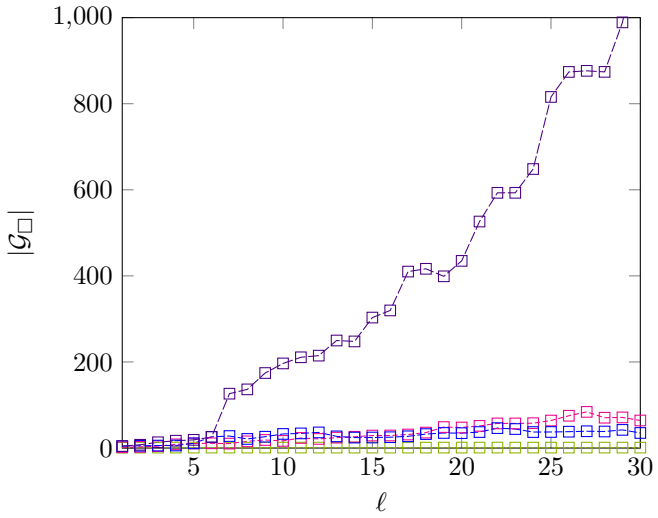


Figure 3: Comparison of the number of option sets in the simplified disjunctive generator $\mathcal{G}_\square$ of an assessment $\mathcal{A}_{1:\ell}$ of size ℓ (horizontal axis), each pair $(V, W) \in \mathcal{A}_{1:\ell}$ containing between two and eight options in total, and constructed using C_{lin} (-□-), C_{max} (-□-), C_{adm} (-□-) and C_{imp} (-□-). The results are the average of 7 individual experiments and shown in a non-logarithmic plot.

It is not difficult to see that the extreme probability mass functions of $\underline{\mathrm{E}}$ are $\{(1-\epsilon)p + \epsilon\mathbb{I}_x \colon x \in \mathcal{X}\}$, where $\mathbb{I}_x \colon \mathcal{X} \to \mathbb{R}$ is the standard basis vector that is 0 everywhere except in x where it is equal to 1. Since we choose with maximality, the corresponding choice function $C_{\max} = C_{\{\underline{\mathrm{E}}^\epsilon\}}$ is given by

$$\begin{aligned}
C_{\max}(A) &= C_{\{\underline{\mathrm{E}}^\epsilon\}}(A) \\
&= \{u \in A \colon (\forall v \in A)\ \underline{\mathrm{E}}^\epsilon(v-u) \leq 0 \text{ and } u \not< v\} \\
&= \{u \in A \colon (\forall v \in A-u)\ \underline{\mathrm{E}}^\epsilon(v) \leq 0 \text{ and } 0 \not< v\} \\
&= \{u \in A \colon (\forall v \in A-u)\ (1-\epsilon)E(v) + \epsilon \min v \leq 0 \text{ and } 0 \not< v\} \\
&= \{u \in A \colon (\forall v \in A-u)\ [(1-\epsilon)E(v) \leq -\epsilon \min v \text{ and } 0 \not< v]\}
\end{aligned}$$

for every $A \in \mathscr{Q}$ and $\epsilon \in [0,1]$. Intuitively ϵ can be seen as an indication of the amount of imprecision in the choice function. We see that for $\epsilon = 0$ we have a precise choice function C_{lin}, and for $\epsilon = 1$ we have the vacuous choice function[9] $C_<$ because then the first condition $0 \leq -\min v$ is implied by the second condition $0 \not< v$. In between this ϵ induces a continuous transformation between the two.

We are interested in the effect of imprecision, quantified by ϵ, on the size of the disjunctive generator. To investigate this, we repeat the following sub-experiment 100 times and take the average of the number of option sets in the resulting generators. First we generate one probability mass function p and an array of 10 option sets $A_1, ..., A_{10}$. Then we loop over ϵ from 0.03 to 0.99 in steps of 0.03. For each ϵ we created the assessment $\mathcal{A}(C_{\{\underline{\mathrm{E}}^\epsilon\}}, A_1, ..., A_{10})$. Next, we determine the size of the disjunctive generators $\mathcal{G}_\times, \mathcal{G}_\triangle$ and $\mathcal{G}_\square$ that are constructed by each of our three methods, respectively. For the first two methods we can still use $|\mathcal{G}| = \prod_{H \in \mathcal{H}} |H|$ to determine the size of the generator without actually constructing it in memory. For the third method we need to construct the disjunctive generator explicitly to determine its size.

In Figure 4, we have plotted the number of option sets in $\mathcal{G}_\times$, $\mathcal{G}_\triangle$ and $\mathcal{G}_\square$ on the vertical axis as a function of ϵ on the horizontal axis. The number of option sets in the generators starts low for all three methods, because in the precise case most options are rejected and therefore every $|H|$ in the product $|\mathcal{G}| = \prod_{H \in \mathcal{H}} |H|$ is small. But when we increase the imprecision, fewer and fewer options are rejected and the number of option sets in the generator $\mathcal{G}_\times$ starts to increase. It seems that adding imprecision makes things worse, but that this trend does not keep going. Rejecting fewer options also means that the number of option sets in $\mathcal{H}_\times$ decreases. At some point the low number of option sets in $\mathcal{H}_\times$ outweighs the fact that the number of options in these option sets increases, and then $|\mathcal{G}_\times|$ decreases. The generator $\mathcal{G}_\triangle$ starts out similar to $\mathcal{G}_\times$ but differs more from it as imprecision increases. The reason is that as imprecision increases, the rejections become less and less informative. For example, it becomes more likely that an option that is rejected, is rejected because of the ordering $<$ in which case this option set will be removed from the conjunctive generator $\mathcal{H}_\triangle$ by Lemma 5.1 in Algorithm 6 but not from $\mathcal{H}_\times$. The generator $\mathcal{G}_\square$ follows a similar trend as $\mathcal{G}_\triangle$, but it is smaller because there is more simplification, and it is also smoother.

7.4. Time benchmarking: creation vs. evaluation

In our final experiment, we focus on the actual time it takes to run the algorithms, rather than the size of the generators. We measured two things:

1. the time it takes to 'construct' a disjunctive generator for an assessment of a given size for each of the three methods; for the first two methods we mean with 'construct' that we find $\mathcal{H}_\times$ and $\mathcal{H}_\triangle$—since $\mathcal{G}_\times$ and $\mathcal{G}_\triangle$ are constructed on the fly without storing them in memory—while for the last method we mean that we find $\mathcal{G}_\square$,

2. the time to evaluate the natural extension of a given option set, again for a given size of assessment, using Algorithm 2 with either $\mathcal{G}_\times$, $\mathcal{G}_\triangle$ or $\mathcal{G}_\square$. For the first two methods this time also includes the time to construct the generator sets on the fly.

The timings are done using the BenchmarkTools Julia package. Everything else is the same as in Section 7.2.

[9]For a given option set $A \in \mathscr{Q}$, this choice function only rejects an option $u \in A$ if it is dominated by some other $v \in A$ in the sense that $u < v$.

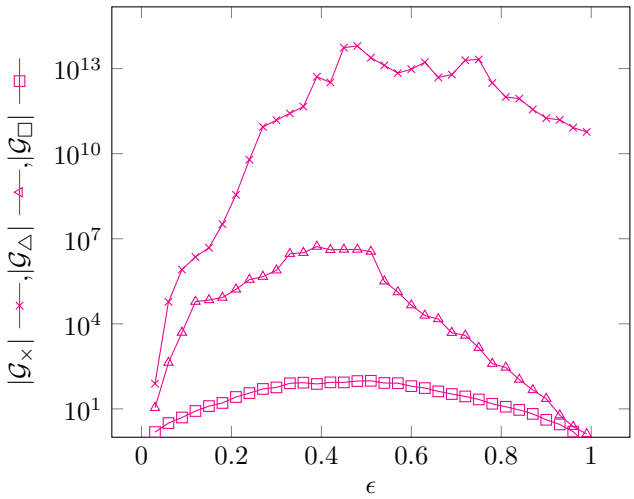
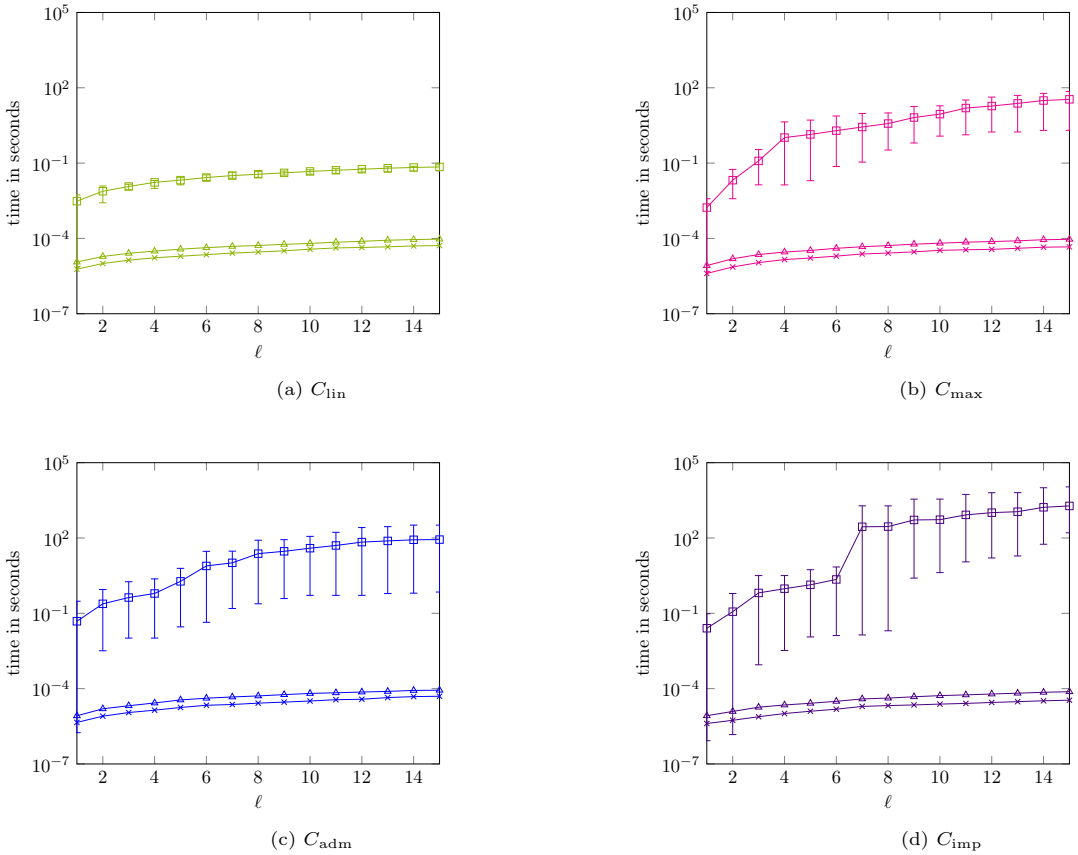


Figure 4: Total size of the disjunctive generator of an assessment with size 10 as a function of ϵ, the amount of contamination of an expectation with the vacuous model, using three different methods. The first method is indicated by $\times$, the second method is indicated by $\triangle$ and the third by $\square$. The results are the average of 100 individual experiments and plot with a logarithmic vertical axis.



(a) C_{lin}

(b) C_{max}

(c) C_{adm}

(d) C_{imp}

Figure 5: Time in seconds to 'construct' a disjunctive generator $\mathcal{G}$ as a function of the number ℓ of pairs (V, W) in the assessment $\mathcal{A}_{1:\ell}$ for the three methods: $\times$, $\triangle$ and $\square$ respectively. The indicated values are the average of 7 experiments and the error bars on the third method are the maximum and minimum value observed in the 7 experiments. The vertical axis is logarithmic.

In Figure 5, we plot the time in seconds to construct the generator $\mathcal{G}$ for a given assessment, using our three different methods. We consider the average over 7 different assessments. We see that the first method ($\times$) is always the fastest. This is because it corresponds to simple subtractions of vectors without additional checks. At the same time the second method seems to be very close to the first method. For the third method, it seems to take much longer to construct the generator than for the other two methods, for example for C_{imp} for an assessment containing 15 pairs of option sets, each containing between two and eight options per pair, it took on average 31 minutes and 36 seconds. This is because, compared to the second method, for the third method we have to construct $\mathcal{G}_\square$ from $\mathcal{H}_\triangle$ (in memory) while also doing the simplifications that make the difference between $\mathcal{G}_\triangle$ and $\mathcal{G}_\square$. For the third method, we also plotted error bars for the minimum and maximum time, but not for the others because they are already very close to each other and the error bars were also very small. From the error bars it can be seen that there is also a large spread in how long it takes to construct the generator $\mathcal{G}_\square$, except for C_{lin}. For C_{imp} a large jump can be seen in the number of option sets in the average of $\mathcal{G}_\square$, but this is due to a large jump in the maximum value, as the minimum value does not jump up until later.

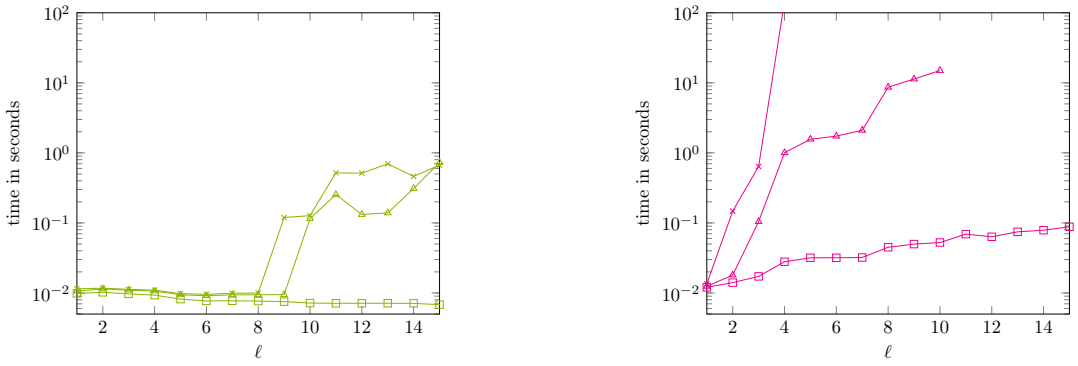
In Figure 6, we plot the time to choose from a single option set; that is, to evaluate the natural extension $C_{\mathcal{A}_{1:\ell}}$ for a single option set with Algorithm 2, using either $\mathcal{G}_\times$, $\mathcal{G}_\triangle$ or $\mathcal{G}_\square$. The points in the figure are an average over the same 7 assessments $\mathcal{A}_{1:L}$ that we have used in Section 7.2. Additionally, for each assessment $\mathcal{A}_{1:L}$, we also generated 7 random options sets to choose from, each containing between 2 and 8 options. So if the assessments are $\mathcal{A}^1_{1:L}, ..., \mathcal{A}^7_{1:L}$, we have option sets $B^1_1, ..., B^1_7$ corresponding to $\mathcal{A}^1_{1:L}$ up to $B^7_1, ..., B^7_7$ corresponding to $\mathcal{A}^7_{1:L}$. Every point on Figure 6 is the average of the time to calculate $C_{\mathcal{A}^r_{1:\ell}}(B^r_k)$ over $k \in \{1, ..., 7\}$ and $r \in \{1, ..., 7\}$, for every ℓ, using one of our three methods. The number of options in every option set was chosen uniformly random from $\{2, ..., 8\}$, and the options themselves are again generated uniformly random from $[0, 1]^4$. We see in Figure 6 that our simplifications indeed work, in the sense that more simplifications result in smaller choosing times. This is what we expected, as a smaller generator will lead to a faster loop over the generator in Algorithm 2.

Next, in Figure 7, we plot the sum of the time it takes to construct the generator and the time it takes to use this generator to evaluate the natural extension using Algorithm 2. For this plot the data points stop when the calculations take more than 50 minutes on average. This in fact already occurred in Figure 6, but is not visible there because we only plot up to 100 seconds there. For all methods, we see that it is initially faster to use the second method but that the third method eventually becomes faster for larger assessments. How fast this happens depends on the type of assessment, and we see that for C_{max} and C_{adm} it happens faster than for C_{lin} and C_{imp}. For C_{imp}, it may seem as if the third method is never better, but this is not the case because from $\ell = 13$ onwards, the results of the second method are not displayed because the calculations took too long; so for these higher values of ℓ, the third method is again better.

Figure 7 only considers the time for making one choice though. If we evaluate the natural extension for several option sets, for the same assessment, then there will come a point where the fact that the time investment $t_{\text{pre},\square}$ for the preprocessing of the third method is higher than to the time $t_{\text{pre},\triangle}$ for the preprocessing of the second method is compensated by the fact that the time $t_{\text{choose},\square}$ to evaluate the natural extension for the third method is lower than the time $t_{\text{choose},\triangle}$ to evaluate the natural extension for the second method. The number of evaluations n at which the total times for both methods break even is the smallest n such that $t_{\text{pre},\square} + nt_{\text{choose},\square} \leq t_{\text{pre},\triangle} + nt_{\text{choose},\triangle}$. If $t_{\text{choose},\triangle} > t_{\text{choose},\square}$—which is true in all our experiments—solving for n gives

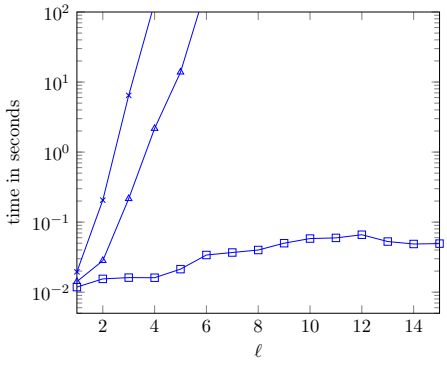
$$n \geq \frac{t_{\text{pre},\square} - t_{\text{pre},\triangle}}{t_{\text{choose},\triangle} - t_{\text{choose},\square}}.$$

We plot the right-hand side of this inequality in Figure 8 for the two most interesting cases: C_{lin} and C_{imp}. The main conclusion is that even for small assessments, the third method is still faster than the second method, provided that the number n of option sets that we need to evaluate is high enough. For C_{lin} we found the highest break-even point for $\ell = 9$, for which the third method is faster if we need to evaluate the natural extension for at least 21 option sets. For C_{imp}, we found that the highest value was for $\ell = 7$, for which we would need to evaluate at least 70 option sets before the third method becomes faster.

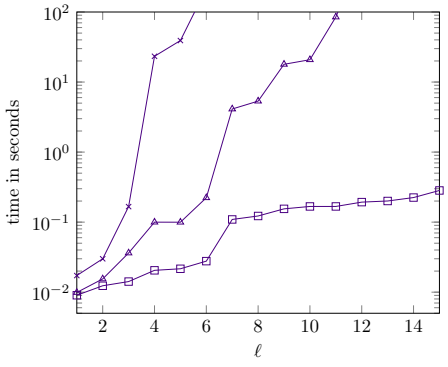


(a) C_{lin}

(b) C_{max}

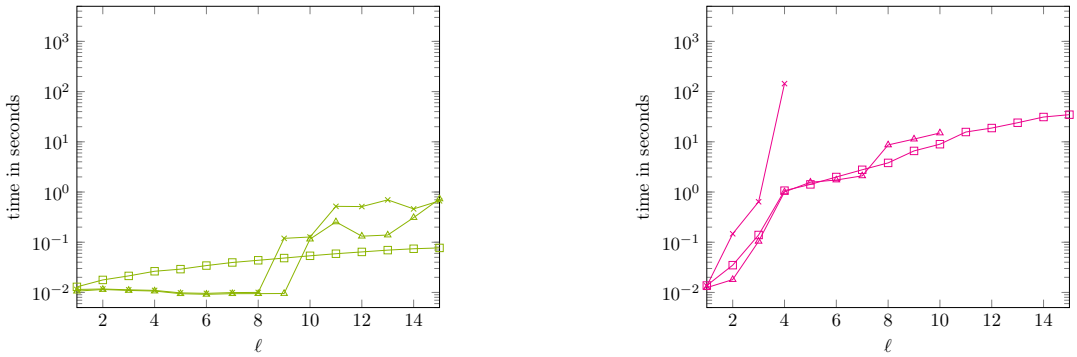


(c) C_{adm}



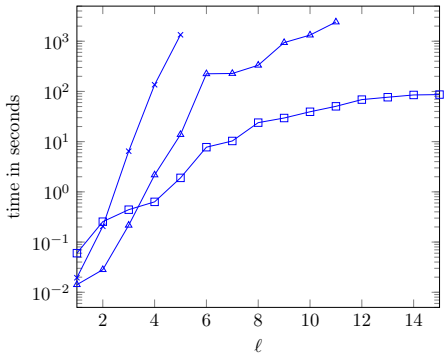
(d) C_{imp}

Figure 6: Time in seconds to evaluate the natural extension for a single option set using $\mathcal{G}_{\times}$, $\mathcal{G}_{\triangle}$ or $\mathcal{G}_{\square}$, as a function of the number ℓ of pairs (V, W) in the assessment. The vertical axis is logarithmic, and the experiments are averaged over 7 separate experiments and 7 option sets to choose from per experiment.

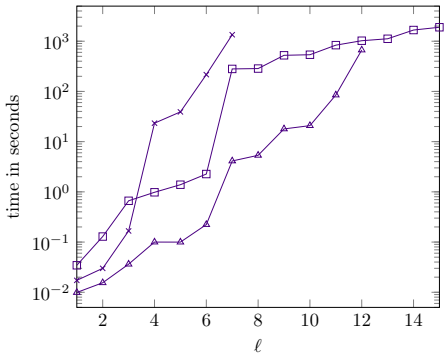


(a) C_{lin}

(b) C_{max}

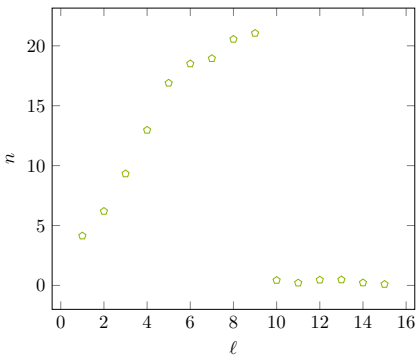


(c) C_{adm}

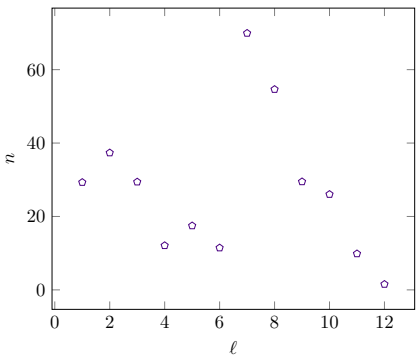


(d) C_{imp}

Figure 7: Total time in seconds to 'construct' a $\mathcal{G}_\times$, $\mathcal{G}_\triangle$ or $\mathcal{G}_\square$ and subsequently use it to evaluate the natural extension for a single option set, as a function of the number ℓ of pairs (V, W) in the assessment. The vertical axis is logarithmic, the experiments are averaged over 7 separate experiments (with for each experiment 7 option sets to choose from) and the lines for each method stop when the average was over 50 minutes.



(a) C_{lin}



(b) C_{imp}

Figure 8: The vertical axis depicts for how many option sets with (between 2 and 8 options each) you would have to evaluate the natural extension before it is better to use the third method rather than the second method for C_{lin} and C_{imp}. For C_{imp} the plot only goes up to $\ell = 12$ because for higher ℓ it took too long to evaluate the natural extension using the second method.

8. Conclusion and future work

The main conclusion of this work is that choice functions provide a principled viable framework for inferring new decisions from previous ones. The two key concepts that we introduced to achieve this were consistency and natural extension. The former allows one to check if there is at least one preference order that is compatible with an assessment, while the latter allows one to infer new choices based on the assessment by considering all compatible orders. From a practical point of view, our main contributions are algorithms that are able to execute these tasks. We have also developed effective methods to simplify the information in the assessment, while still retaining all decisive power. To demonstrate the performance when including these simplifications, we carried out experiments that measure the space and time efficiency under various scenarios. Our results show that our simplifications allow for larger assessments to be dealt with, and this for various levels of imprecision. Related to this, we saw that the third method—which implements all the simplifications—outperforms the other two methods in terms of efficiency for larger assessments. For smaller assessments, the second method is the most efficient, unless one has to evaluate the natural extension for multiple option sets, in which case the third method eventually becomes faster. A particularly intriguing observation was that for assessments that are generated with maximality or E-admissibility, the generator of the third method even seemed to stabilise, as seen in Figure 3. This is intriguing because it could mean that it is converging towards the 'real' model—the model that we used to make the assessment—through simple approximate models. It would be interesting to investigate further if this trend continues for even larger assessments, and explain why it happens.

Future work could also add onto Section 5 by trying to obtain a 'simplest' generator for any given assessment, or showing that our methods already achieves this. Also, how often and in what order one has to apply the simplifications can still be optimised. Approaches in between method 2 and 3 could for example be considered. Another direction could be to analyse how efficiency scales with other parameters, both theoretically and experimentally. This includes the size of the option sets in the assessment, the dimension of the vector space $\mathscr{V}$ and the size of the option set A for which we want to evaluate the natural extension. One could also consider alternative forms of assessments, such as bounds on probabilities, bounds on expectations and preference statements, and show how they can be made to fit in our choice functions framework. Finally, it would be nice to evaluate how our methods perform when applied to real-life decision problems.

Acknowledgements

This work was partially supported by Ghent University, through Jasper De Bock's BOF Starting Grant "Rational decision-making under uncertainty: a new paradigm based on choice functions", number 01N04819.

References

[1] M. C. M. Troffaes, Decision making under uncertainty using imprecise probabilities, International Journal of Approximate Reasoning 45 (2007) 17–29.

[2] De Bock, Jasper, Archimedean choice functions: an axiomatic foundation for imprecise decision making, in: Information Processing and Management of Uncertainty in Knowledge-Based Systems, volume 1238, Springer, 2020, pp. 195–209.

[3] J. De Bock, G. De Cooman, Interpreting, axiomatising and representing coherent choice functions in terms of desirability, in: Proceedings of the eleventh International Symposium on Imprecise Probabilities: Theories and Applications, PMLR, 2019, pp. 125–134.

[4] T. Seidenfeld, M. J. Schervish, J. B. Kadane, Coherent choice functions under uncertainty, Synthese 172 (2010) 157.

[5] A. Van Camp, E. Miranda, G. De Cooman, Natural extension of choice functions, in: Proceedings of the seventeenth International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, Springer, 2018, pp. 201–213.

[6] A. K. Sen, Choice functions and revealed preference, The Review of Economic Studies 38 (1971) 307–317.

[7] A. Sen, Social choice theory: A re-examination, Econometrica: journal of the Econometric Society (1977) 53–89.

[8] G. Birkhoff, Lattice theory, third ed., American Mathematical Society, 1940.

[9] T. Augustin, F. P. Coolen, G. De Cooman, M. C. Troffaes, Introduction to Imprecise Probabilities, John Wiley & Sons, 2014.

[10] A. Van Camp, G. De Cooman, E. Miranda, E. Quaeghebeur, Coherent choice functions, desirability and indifference, Fuzzy sets and systems 341 (2018) 1–36.

[11] E. Quaeghebeur, The CONEstrip algorithm, in: Synergies of Soft Computing and Statistics for Intelligent Data Analysis, Springer, 2013, pp. 45–54.

[12] J. Matoušek, B. Gärtner, Understanding and Using Linear Programming, Universitext, Springer, 2006.

[13] P. M. Vaidya, Speeding-up linear programming using fast matrix multiplication, in: 30th annual symposium on foundations of computer science, IEEE Computer Society, 1989, pp. 332–337.

[14] C. B. Barber, D. P. Dobkin, H. Huhdanpaa, The quickhull algorithm for convex hulls, ACM Trans. Math. Softw. 22 (1996) 469–483.

[15] K. Fukuda, A. Prodon, Double description method revisited, in: Combinatorics and Computer Science, Springer Berlin Heidelberg, Berlin, Heidelberg, 1996, pp. 91–111.

[16] K. Fukuda, Polyhedral computation, Department of Mathematics, Institute of Theoretical Computer Science ETH Zurich, 2020.

[17] K. Clarkson, More output-sensitive geometric algorithms, in: Proceedings of the 35th Annual Symposium on Foundations of Computer Science, 1994, pp. 695–702.

[18] T. Ottmann, S. Schuierer, S. Soundaralakshmi, Enumerating extreme points in higher dimensions, in: STACS, volume 95, 1995, pp. 562–570.

[19] B. Legat, Polyhedral computation, in: JuliaCon, 2023. URL: `https://pretalx.com/juliacon2023/talk/JP3SPX/`.

[20] B. Legat, Set programming: theory and computation, Ph.D. thesis, UCL-Université Catholique de Louvain, 2020.

[21] N. Nakharutai, M. C. M. Troffaes, C. C. Caiado, Improved linear programming methods for checking avoiding sure loss, International Journal of Approximate Reasoning 101 (2018) 293–310.

Appendix A. Equivalence of axioms

In this appendix we prove the equivalence of our characterisation of coherence and the axiomatic characterisation of De Bock and De Cooman [3]. In particular, with $\mathbb{R}^2_{>0} \coloneqq \{(\lambda, \nu) \in \mathbb{R}^2 \colon (\lambda, \nu) > 0\}$, we consider the following axioms for a rejection function $R \colon \mathscr{Q} \to \mathscr{Q}$: for all $A, B \in \mathscr{Q}$, $u \in A$ and $(\lambda_1, \lambda_2) \colon A \times B \to \mathbb{R}^2_{>0}$:

- R$_0$. $u \in R(A)$ if and only if $0 \in R(A - u)$;
- R$_1$. $R(A) \neq A$;
- R$_2$. if $u > 0$ then $0 \in R(\{0, u\})$;
- R$_3$. if $0 \in R(A \cup \{0\})$ and $0 \in R(B \cup \{0\})$ then
$$0 \in R(\{\lambda_1(v, w)v + \lambda_2(v, w)w \colon v \in A, w \in B\} \cup \{0\});$$
- R$_4$. if $A \subseteq B$ then $R(A) \subseteq R(B)$.

Our main result is then the following.

Theorem A.1. A choice function $C \colon \mathscr{Q} \to \mathscr{Q}$ is coherent if and only if its corresponding rejection function R_C satisfies R$_0$–R$_4$.

A subtle point is that these are not exactly the axioms used by De Bock and De Cooman [3], as they also add $\emptyset$ to the domain of their choice and rejection functions. We will however see that this is a mere technicality with no substantial effect.

To be able to use the results of De Bock and De Cooman [3] in our proof for Theorem A.1, we too will consider choice and rejection functions whose domain includes $\emptyset$ and call it them *extended choice and rejection functions*: a map $\hat{C} \colon \mathscr{Q}_\emptyset \to \mathscr{Q}_\emptyset$ is called an extended choice function if it satisfies $\hat{C}(A) \subseteq A$ for all $A \in \mathscr{Q}_\emptyset$. With any such extended choice function $\hat{C}$ we also associate a corresponding extended rejection function $\hat{R}_{\hat{C}} \colon \mathscr{Q}_\emptyset \to \mathscr{Q}_\emptyset$, defined by $\hat{R}_{\hat{C}}(A) = A \setminus \hat{C}(A)$ for all $A \in \mathscr{Q}_\emptyset$. Since we must have that $\hat{C}(\emptyset) \subseteq \emptyset$, we always have that $\hat{C}(\emptyset) = \emptyset$ and $\hat{R}(\emptyset) = \emptyset$, which makes this part of the domain uninteresting.

With any choice function $C \colon \mathscr{Q} \to \mathscr{Q}$ we associate the extended choice function

$$\overline{C} \colon \mathscr{Q}_\emptyset \to \mathscr{Q}_\emptyset \colon A \mapsto \begin{cases} C(A) & \text{if } A \neq \emptyset, \\ \emptyset & \text{if } A = \emptyset. \end{cases} \tag{A.1}$$

This link between extended choice functions and choice functions is bijective because $\hat{C}(\emptyset) = \emptyset$ for every extended choice function $\hat{C}$. For any extended rejection function $\hat{R} \colon \mathscr{Q}_\emptyset \to \mathscr{Q}_\emptyset$, De Bock and De Cooman [3] consider the following axioms: for all $A, B \in \mathscr{Q}_\emptyset$, $u \in A$ and $(\lambda_1, \lambda_2) \colon A \times B \to \mathbb{R}^2_{>0}$:

- $\hat{\text{R}}_0$. $u \in \hat{R}(A)$ if and only if $0 \in \hat{R}(A - u)$;
- $\hat{\text{R}}_1$. $\hat{R}(\emptyset) = \emptyset$ and if $A \neq \emptyset$ then $\hat{R}(A) \neq A$;
- $\hat{\text{R}}_2$. if $u > 0$ then $0 \in \hat{R}(\{0, u\})$;
- $\hat{\text{R}}_3$. if $0 \in \hat{R}(A \cup \{0\})$ and $0 \in \hat{R}(B \cup \{0\})$ then
$$0 \in \hat{R}(\{\lambda_1(v, w)v + \lambda_2(v, w)w \colon v \in A, w \in B\} \cup \{0\});$$
- $\hat{\text{R}}_4$. if $A \subseteq B$ then $\hat{R}(A) \subseteq \hat{R}(B)$.

These axioms are very similar to R$_0$–R$_4$; the only difference is that they are also imposed for $A = \emptyset$ and/or $B = \emptyset$, and that $\hat{\text{R}}_1$ adds the condition $\hat{R}(\emptyset) = \emptyset$. Due to this similarity, it should not come as a surprise that imposing R$_0$–R$_4$ on the rejection function R_C that corresponds to a choice function C is equivalent to imposing $\hat{\text{R}}_0$–$\hat{\text{R}}_4$ on the extended rejection function $\hat{R}_{\overline{C}}$ that corresponds to its extension $\overline{C}$.

Lemma A.2. For any choice function $C\colon \mathscr{Q} \to \mathscr{Q}$ and $k \in \{0, 1, ..., 4\}$ we have that R_C satisfies R_k if and only if $\hat{R}_{\overline{C}}$ satisfies $\hat{\mathrm{R}}_k$.

Proof. It follows immediately from the definitions that for any $A \in \mathscr{Q}$, $R_C(A) = \hat{R}_{\overline{C}}(A)$. Therefore, the implication to the left is immediate. So now we prove that when at least one of A and B are empty, the axioms still hold true. For $k = 0$, $\hat{\mathrm{R}}_0$ is trivially true for $A = \emptyset$ as there are no $u \in A$ then. For $k = 1$, $\hat{\mathrm{R}}_1$ is true because $\hat{R}_{\overline{C}}(\emptyset) = \emptyset$ by definition. $k = 2$ is unrelated to A and B, so trivially true. For $k = 3$, $\hat{\mathrm{R}}_3$ is true because if A and/or B are empty, then we have $0 \in \hat{R}_{\overline{C}}(\{0\})$ from the antecedent and this is also the implication. Finally, for $k = 4$, $\hat{\mathrm{R}}_4$ is true because the case where $A = \emptyset \subseteq B$, for any $B \in \mathscr{Q}_\emptyset$, implies that $\hat{R}_{\overline{C}}(A) = \emptyset \subseteq \hat{R}_{\overline{C}}(B)$ and the case where $B = \emptyset$ but $A \neq \emptyset$ is impossible. □

Similarly to how we associated a choice function $C_{\mathcal{O}}$ with any set of preference orders $\mathcal{O}$ in Equation (1), we can also associate an extended choice function $\hat{C}_{\mathcal{O}}$ with such a set $\mathcal{O}$, by letting

$$\hat{C}_{\mathcal{O}}(A) = \{u \in A\colon (\exists \prec \in \mathcal{O})(\forall a \in A) u \not\prec a\} \quad \text{for all} \quad A \in \mathscr{Q}_\emptyset. \tag{A.2}$$

We call an extended choice function coherent if and only if it is of this form, with $\mathcal{O}$ non-empty.

Definition A.3. An extended choice function $\hat{C}$ is coherent if and only if there is a non-empty set of preference orders $\mathcal{O} \subseteq \mathbb{O}$ such that $\hat{C} = \hat{C}_{\mathcal{O}}$.

Lemma A.4. A choice function C is coherent if and only if $\overline{C}$ is coherent.

Proof. The implication to the left is immediate as the definition of coherence is the same for both and C has a smaller domain than $\overline{C}$. For the implication to the right, assume that C is coherent. By definition of coherence, there is a set of preference orders $\mathcal{O}$ such that $C = C_{\mathcal{O}}$. Then for all $A \in \mathscr{Q}$ we have $\overline{C}(A) = C(A) = C_{\mathcal{O}}(A) = \hat{C}_{\mathcal{O}}(A)$. For $A = \emptyset$ we also have trivially that $\hat{C}_{\mathcal{O}}(\emptyset) = \emptyset = \overline{C}(\emptyset)$. □

Before we get to the proof of Theorem A.1, we now first use the results of De Bock and De Cooman [3] to establish a similar result for extended choice functions.

Proposition A.5. An extended choice function $\hat{C}\colon \mathscr{Q}_\emptyset \to \mathscr{Q}_\emptyset$ is coherent if and only if its corresponding extended rejection function $\hat{R}_{\hat{C}}$ satisfies $\hat{\mathrm{R}}_0$–$\hat{\mathrm{R}}_4$.

Our proof makes use of the following technical lemmata.

Lemma A.6. Consider any set of preference orders $\mathcal{O} \subseteq \mathbb{O}$. Then for all $A \in \mathscr{Q}_\emptyset$, $0 \in \hat{R}_{\hat{C}_{\mathcal{O}}}(A - u) \Leftrightarrow u \in \hat{R}_{\hat{C}_{\mathcal{O}}}(A)$.

Proof. $u \in \hat{R}_{\hat{C}_{\mathcal{O}}}(A)$ is by definition equivalent to $(\forall \prec \in \mathcal{O})(\exists a \in A) u \prec a$. This is by Axiom $\prec_2$ equivalent to $(\forall \prec \in \mathcal{O})(\exists v \in A - u) 0 \prec v$, which is equivalent to $0 \in \hat{R}_{\hat{C}_{\mathcal{O}}}(A - u)$. □

Lemma A.7. Consider an extended rejection function $\hat{R}\colon \mathscr{Q}_\emptyset \to \mathscr{Q}_\emptyset$ that satisfies $\hat{\mathrm{R}}_0$ and let $K := \{A \in \mathscr{Q}_\emptyset\colon 0 \in \hat{R}(A \cup \{0\})\}$. Then

$$(\forall u \in \mathscr{V})(\forall A \in \mathscr{Q}_\emptyset)\ u \in \hat{R}(A \cup \{u\}) \Leftrightarrow A - u \in K. \tag{A.3}$$

Proof. $A - u \in K$ is equivalent to $0 \in \hat{R}((A - u) \cup \{0\}) = \hat{R}((A \cup \{u\}) - u)$. By $\hat{\mathrm{R}}_0$ this is equivalent to $u \in \hat{R}(A \cup \{u\})$. □

Proof of Proposition A.5 First we prove the implication to the right. Consider any non-empty set $\mathcal{O} \subseteq \mathbb{O}$. We need to prove that $\hat{R}_{\hat{C}_{\mathcal{O}}}$ satisfies $\hat{\mathrm{R}}_0$ to $\hat{\mathrm{R}}_4$. Axiom $\hat{\mathrm{R}}_0$ follows immediately from Lemma A.6. Since $\hat{R}_{\hat{C}_{\mathcal{O}}}(\emptyset) = \emptyset$ follows from Definition (A.2) and, for any $A \in \mathscr{Q}$, we have $\hat{R}_{\hat{C}_{\mathcal{O}}}(A) = A \backslash C_{\mathcal{O}}(A)$, $\hat{\mathrm{R}}_1$ follows immediately from Proposition 2.4. $\hat{\mathrm{R}}_2$ follows immediately from Axiom $\prec_4$ and Definition (A.2). For $\hat{\mathrm{R}}_3$, take any $\prec \in \mathcal{O}$. From Definition (A.2) and Axiom $\prec_0$, we know that there are $a \in A$ such that $0 \prec a$ and $b \in B$

such that $0 \prec b$. Without loss of generality, assume that $\lambda_1(a,b) > 0$. By Axiom $\prec_3$, we have $0 \prec \lambda_1(a,b)a$ and either $\lambda_2(a,b)b = 0$ or $0 \prec \lambda_2(a,b)b$. In the former case we find that $0 \prec \lambda_1(a,b)a = \lambda_1(a,b)a + \lambda_2(a,b)b$. In the latter case, it follows from Axiom $\prec_2$ that $\lambda_1(a,b)a \prec \lambda_1(a,b)a + \lambda_2(a,b)b$ and therefore, since $0 \prec \lambda_1(a,b)a$, from Axiom $\prec_1$ that $0 \prec \lambda_1(a,b)a + \lambda_2(a,b)b$. Hence, in both cases, this implies that $0 \prec \lambda_1(a,b)a + \lambda_2(a,b)b$. Since $\lambda_1(a,b)a + \lambda_2(a,b)b \in \{\lambda_1(v,w)v + \lambda_2(v,w)w \colon v \in A, w \in B\} \cup \{0\}$ and $\prec \in \mathcal{O}$ was arbitrary, it therefore follows from Definition (A.2) that

$$0 \in \hat{R}_{\hat{C}_{\mathcal{O}}}(\{\lambda_1(v,w)v + \lambda_2(v,w)w \colon v \in A, w \in B\} \cup \{0\}).$$

For $\hat{\mathrm{R}}_4$, if $u \in \hat{R}_{\hat{C}_{\mathcal{O}}}(A)$, then for all $\prec \in \mathcal{O}$ there is some $a \in A \subseteq B$ such that $u \prec a$, whence also $u \in \hat{R}_{\hat{C}_{\mathcal{O}}}(B)$.

Next we prove the implication to the left making heavy use of the results in [3]. In Ref. [3], the main tool are so-called 'coherent sets of desirable option sets' $K \subseteq \mathscr{Q}_\emptyset$, where coherence is characterised through a number of axioms [3, Definition 2] that are not important for our purposes. Since $\hat{R}_{\hat{C}}$ satisfies $\hat{\mathrm{R}}_0$, we know from Lemma A.7 that $K \coloneqq \{A \in \mathscr{Q}_\emptyset \colon 0 \in \hat{R}_{\hat{C}}(A \cup \{0\})\}$ satisfies Equation (A.3). Since $\hat{R}_{\hat{C}}$ satisfies $\hat{\mathrm{R}}_0$–$\hat{\mathrm{R}}_4$, it therefore follows from [3, Proposition 4] that K is a 'coherent set of desirable option sets'. Therefore, it follows from [3, Theorem 9] that there is a non-empty set $\mathscr{D} \subseteq \overline{\mathbf{G}}$ of coherent sets of desirable options such that

$$K = \bigcap_{G \in \mathscr{D}} \{A \in \mathscr{Q}_\emptyset \colon A \cap G \neq \emptyset\}.$$

Let $\mathcal{O} \coloneqq \{\prec_G \colon G \in \mathscr{D}\}$ and recall from Lemma 2.3 that this is a set of preference orders, which is furthermore non-empty because $\mathscr{D}$ is. Observe also that, for all $G \in \mathscr{D}$,

$$\{A \in \mathscr{Q}_\emptyset \colon A \cap G \neq \emptyset\} = \{A \in \mathscr{Q}_\emptyset \colon (\exists a \in A) 0 \prec_G a\}$$

by definition of $\prec_G$. We then see that

$$\begin{aligned} K &= \bigcap_{G \in \mathscr{D}} \{A \in \mathscr{Q}_\emptyset \colon (\exists a \in A) 0 \prec_G a\} \\ &= \{A \in \mathscr{Q}_\emptyset \colon (\forall G \in \mathscr{D})(\exists a \in A) 0 \prec_G a\} \\ &= \{A \in \mathscr{Q}_\emptyset \colon (\forall \prec \in \mathcal{O})(\exists a \in A) 0 \prec a\} \\ &= \{A \in \mathscr{Q}_\emptyset \colon (\forall \prec \in \mathcal{O})(\exists a \in A \cup \{0\}) 0 \prec a\} \\ &= \{A \in \mathscr{Q}_\emptyset \colon 0 \in \hat{R}_{\hat{C}_{\mathcal{O}}}(A \cup \{0\})\}, \end{aligned}$$

where the second to last equality follows from Axiom $\prec_0$ and the last from Equation (A.2). By the definition of K, this implies that for any $A \in \mathscr{Q}_\emptyset$, $0 \in \hat{R}_{\hat{C}}(A \cup \{0\})$ if and only if $0 \in \hat{R}_{\hat{C}_{\mathcal{O}}}(A \cup \{0\})$. Now, by $\hat{\mathrm{R}}_0$, the previous observation and Lemma A.6 we have for any $A \in \mathscr{Q}_\emptyset$ and $u \in A$ that

$$\begin{aligned} u \in \hat{R}_{\hat{C}}(A) &\Leftrightarrow 0 \in \hat{R}_{\hat{C}}(A - u) \Leftrightarrow 0 \in \hat{R}_{\hat{C}}((A - u) \cup \{0\}) \\ &\Leftrightarrow 0 \in \hat{R}_{\hat{C}_{\mathcal{O}}}((A - u) \cup \{0\}) \Leftrightarrow 0 \in \hat{R}_{\hat{C}_{\mathcal{O}}}(A - u) \Leftrightarrow u \in \hat{R}_{\hat{C}_{\mathcal{O}}}(A), \end{aligned}$$

since $0 \in A - u = (A - u) \cup \{0\}$. Therefore, $\hat{R}_{\hat{C}} = \hat{R}_{\hat{C}_{\mathcal{O}}}$. □

Proof of Theorem A.1 By Lemma A.4, we have that coherence of C is equivalent to coherence of $\overline{C}$. By Proposition A.5, this is equivalent to $\hat{R}_{\overline{C}}$ satisfying $\hat{\mathrm{R}}_0$–$\hat{\mathrm{R}}_4$. Finally, by Lemma A.2, this is equivalent to R_C satisfying R_0–R_4. □