Decision Policy Design as Pareto-Minimization of Infeasible Lower Bounds

Ulrich Junker*, Alexis Tsoukiàs†

Abstract

A decision policy chooses an outcome dependent on given input parameters. Policies can adequately be represented by production rules, which are at the heart of modern business rule management systems. Classic ways of policy design are rule authoring by experts or learning from data. In this paper, we show that policies can also be derived from a model consisting of constraints and preferences. We can design a policy that respects the given preferences by solving a particular combinatorial Pareto-optimization problem. We consider a combined parameter and decision space and introduce a rule for each Pareto-minimal infeasible lower bound in this space. The approach gives interesting insights in the relationships between combinatorial optimization under preferences and rule-based decision making.

Key words: Pareto-Optimization, Decision Rules

1 Introduction

A decision policy defines which outcome is chosen dependent on given input. For example, consider an automated pricing system that has to assign discounts depending on the customer profile and the shopping cart. Discounts may significantly depend on the input parameters and production rules are a convenient way to represent complex policies that make different decisions depending on which conditions are met by the parameters [3]. As production rules constitute a very natural representation of the decision making policies, they can directly be acquired from the experts. Another method for rule acquisition is learning from historical data and the whole breadth of data mining techniques can

*ILOG, 1681, route des Dolines, 06560 Valbonne, France {ujunker}@ilog.fr

†LAMSADE-CNRS, Université Paris Dauphine, 75775 Paris Cedex 16, France {tsoukias}@lamsade.dauphine.fr

be applied for this purpose. However, the rule-based representation of policies also has drawbacks. There is no guarantee that a set of rules is consistent and complete, meaning that there may be cases where multiple decisions are made or no decision is made at all. However, even when a rule-set is consistent and complete, there is no guarantee that it represents a policy which is a rational choice function [1] and which makes the decisions in agreement with a given preference order.

These problems become even more apparent if the decision space is combinatorial. For example, we want to design a recommender system for configuring computers. To keep things simple, we suppose that the computer consists of a PC and an external disk. The recommender system will ask the user for budget limitations and the minimal speed and will return the disk space provided by the configuration. Even in this simple setting, there may be numerous combinations of the PC type and the disk type and the ideal combination will strongly vary depending on the budget and the required speed. It may not be easy to establish a rule set that is consistent, complete, and rational as explained above. But now suppose that such a desired rule-set is given, but the product catalog changes slightly. If a new product is added to the catalog, it may lead to better configurations under certain parameter configurations, but not under others. The simple change may affect several rules and a tedious rule updating process is necessary for this purpose [2].

Those changes are easy to incorporate into a model which describes the possible configurations in terms of variables, constraints, and preferences. For example, there will be constraints that model the product catalogs and others that describe the costs, speed, and memory of a given component. Those catalogs can directly be imported from data-bases and mapped to constraints. A constraint-based configurator [6] can directly determine a best configuration depending on the user input. In this case, the decision is determined by solving a combinatorial optimization problem for each user request. Whereas this process implements a policy in a desired way, the policy is not represented explicitly and cannot be inspected and refined by experts.

Decision-making policies can thus be represented explicitly by rules or implemented by an optimizer. Both approaches have desirable properties and are complementary. In spite of this, it is difficult to conciliate both approaches and to move from combinatorial optimization to rules or vice versa. In this paper, we show that work in multi-criteria decision analysis and multi-objective optimization provides the necessary concepts and methods to provide an interesting link between combinatorial optimization and rules.

We first define policies and their representation in form of rules. We then discuss policy design for combinatorial decision making problems. We map this problem to a Pareto-minimization problem over a space of infeasible lower bounds. Duality results about Pareto-optimization allow us to deduce the minimal bounds from a standard Pareto-frontier, which can be computed by existing methods.

2 Policies and Rules

We consider repetitive decision making problems as they occur in (web) services for business automation and in online recommender systems. In those problems, a decision has to be made in function of parameter values, which capture the characteristics of a particular request. For example, a limit on the budget is one typical parameter occurring in online configuration services for cars and computers. Minimal speed may be another parameter in those systems. The parameters determine which decisions are possible in the problem under consideration. We thus obtain a *decision space* $\mathcal{X}$ which contains all the possible decisions and a *parameter space* $\mathcal{P}$ which describes the possible combinations of parameter values. The decision space is restricted by the parameter values, which is modelled by a mapping $X : \mathcal{P} \to 2^{\mathcal{X}}$ from the parameter space to the powerset of the decision space. $X(p)$ describes the set of decisions that are feasible under the parameter value p.

A decision policy is choosing a decision from $X(p)$ depending on the parameter values p. We can also describe a policy by a function $\pi : \mathcal{P} \to \mathcal{X}$ that maps a parameter value p to a decision $\pi(p)$ from $X(p)$. Hence, we define policies as in Markov decision processes, but use them in a simpler setting. Usually, the parameters have the character of bounds or limits, meaning that less options are obtained if the parameter values are replaced by stricter values. For example, smaller budget usually means that less options are available. We model this by a (partial) preorder $\succsim^p$ on the parameter space $\mathcal{P}$, i.e. a transitive and reflexive relation, and require that X is monotonic under this order:

$$p_1 \succsim^p p_2 \quad \textit{implies} \quad X(p_1) \subseteq X(p_2) \tag{1}$$

We may also require that policies make optimal choices under a given weak preference order $\succsim^x$ on the decision space, which is again a preorder. The strict part of this preorder is the strict partial order $\succ^x$ that satisfies $x_1 \succ^x x_2$ iff $x_1 \succsim^x x_2$ holds, but not $x_2 \succsim^x x_1$. A policy makes an optimal (or best) choice for parameter value p iff there is no decision x^* in the restricted set $X(p)$ that dominates the chosen decision $\pi(p)$ w.r.t. the preference order (i.e. there is no $x^* \in X(p)$ s.t. $x^* \succ^x \pi(p)$). A policy is *rational* iff it makes optimal choices for all parameter values under the same preference order $\succsim^x$. If the preference order $\succsim^x$ is a total order, then there is a single best decision in each option set. Hence, there is a unique rational policy in this case.

As policies describe which decisions are made under which parameter values, they are of highest importance for the provider of a decision-making service. In order to understand and control the service, the provider must be able to analyze the policies and to adapt them if needed. An explicit representation of policies is therefore desirable. Production rules provide a compact representation of complex policies as they specify which decision will be made under which conditions.

A simple example is that of choosing a customer category x such as Silver, Gold, and Platinum in a customer fidelity program. The set of options depends on the amount p of

Table 1: Catalogs for components a and b

y_a	$y_{1,a}$	$y_{2,a}$	$y_{3,a}$
a_1	10	20	30
a_2	20	30	10

y_b	$y_{1,b}$	$y_{2,b}$	$y_{3,b}$
b_1	10	10	30
b_2	20	30	10

Table 2: Updated catalogs

y_a	$y_{1,a}$	$y_{2,a}$	$y_{3,a}$
a_1	10	20	30
a_2	20	30	10
a_3	20	20	10

y_b	$y_{1,b}$	$y_{2,b}$	$y_{3,b}$
b_1	10	10	30
b_2	20	30	10
b_3	10	30	20

goods that have been bought by the customer. If the customer buys for less than $\$500$, then only the category Silver is possible. If he buys goods for an amount between $\$500$ and $\$1000$, then Silver and Gold are possible. Finally, if the customer buys for more than $\$1000$, then all options are possible. Furthermore, suppose that Platinum is preferred to Gold, which is preferred to Silver. Then there is a single rational policy, which can be described by the following rules:

$$\begin{aligned}&\text{if } p \leq 500 \text{ then } x := \text{Silver}.\\ &\text{if } p > 500 \wedge p \leq 1000 \text{ then } x := \text{Gold}.\\ &\text{if } p > 1000 \text{ then } x := \text{Platinum}.\end{aligned}$$

Given the amount bought by a customer, these rules can then be applied to choose the category. For example, if the customer has bought for $\$600$, the chosen category will be Gold. The policy makes this choice directly and does not need to determine the set of feasible options and its optimal elements. Indeed, the policy neither describes the mapping X, nor the preference order $\succsim^x$. In simple problems, it is even possible to write down those policies without specifying the set of feasible options for each case and without specifying a preference order.

3 Policies and Combinatorial Optimization

In more complex problems, policies cannot be determined so easily. If data about historical decisions are available, then learning and classification methods can be used to induce a decision policy. For example, the dominance-based rough set approach in [4]

determines a policy that respects a total preference order on the decision space. In other problems, the set of feasible options may be too large to be described explicitly. A good example is product configuration where multiple components have to be chosen such that given customer requirements are satisfied and technical compatibility constraints are respected. In this case, the set of feasible options is implicitly described in form of variables and constraints. As a policy determines the best option depending on the parameter values, finding an explicit representation of this policy is a non-trivial task. In this paper, we adapt the framework of [4] to this policy design problem.

We describe the set of feasible options in terms of a constraint satisfaction problem. We consider m variables with finite and non-empty domains $D_1, \ldots, D_m$. We suppose that $n < m$ and that $\mathcal{P} := D_1 \times \ldots \times D_n$ is the parameter space and that $\mathcal{X} := D_{n+1}$ is the decision space. All the other domains $D_{n+2}, \ldots, D_m$ are the domains of auxiliary variables. The overall problem space is the Cartesian product $\mathcal{D} := D_1 \times \ldots \times D_m$ of all these domains. We also suppose that there is a total order $\geq_i$ on each parameter domain D_i (for $i = 1, \ldots, n$). The preorder on the parameter space is defined by a weak Pareto-ordering $\succsim^p$ on this space: $(p_1^*, \ldots, p_n^*) \succsim^p (p_1, \ldots, p_n)$ holds iff $p_i^* \geq_i p_i$ for all $i = 1, \ldots, n$. Furthermore, we suppose that there is a total order $\geq^x$ on the decision space $\mathcal{X}$. The worst element in this decision space is denoted by $x^\perp$. Given a tuple $y := (y_1, \ldots, y_m)$ from $\mathcal{D}$, we write y_p for the parameter values $(y_1, \ldots, y_n)$ and y_x for the decision y_{n+1}.

The problem space is restricted by a set of constraints. Each constraint has a scope $i_1, \ldots, i_k$ consisting of indexes from $1, \ldots, m$ and a relation R which is a subset of $D_{i_1} \times \ldots \times D_{i_k}$. A tuple y from $\mathcal{D}$ satisfies this constraint iff $(y_{i_1}, \ldots, y_{i_k})$ is an element of the relation R. A constraint respects the preorder $\succsim^p$ on the parameter space iff the following property holds for all tuples $y^*, y \in \mathcal{D}$: if $(y_1^*, \ldots, y_n^*) \succsim^p (y_1, \ldots, y_n)$, $y_j^* = y_j$ for $j = n+1, \ldots, m$, and y^* satisfies the constraint, then y satisfies the constraint.

In the sequel, we consider a set of constraints C that respect the preorder $\succsim^p$. The set of solutions $Sol(C)$ is the set of all tuples from $\mathcal{D}$ that satisfy all constraints in C.

The parameter value $p \in \mathcal{P}$ reduces the set of solutions of C to those elements $y \in Sol(C)$ that support p, i.e. that satisfy the property $y_p = p$. The set of feasible options $X(p)$ consists of all elements x in the decision space $\mathcal{X}$ that are supported by an element y of this reduced solution space meaning that the property $y_x = x$ holds. Furthermore, the worst element $x^\perp$ is an element of $X(p)$:

$$X(p) := \{x^\perp\} \cup \{x \in \mathcal{X} \mid \text{there is } y \in Sol(C) \text{ s.t. } y_p = p \text{ and } y_x = x\} \tag{2}$$

It can easily be shown that the sets $X(p)$ of feasible options satisfy the monotonicity property (1).

A rational policy will choose the best element of $X(p)$ under the order $\geq^x$. For a given $p \in \mathcal{P}$, this problem corresponds to a classic single-objective combinatorial optimization

problem. If this problem has no solution for a parameter value p, then the worst decision $x^{\perp}$ will be chosen. Hence, there is a unique policy which determines an optimal decision for each combination of parameter values p. In this paper, we study the problem of finding an explicit representation of this policy.

As an example, we consider a simple product configuration problem. A customer wants to buy a product consisting of two components a and b. Each component i has a product type y_i and three positive integer attributes $y_{1,i}, y_{2,i}, y_{3,i}$ such as price, power consumption, disk space capacity. Component a may have type a_1 or a_2 and component b may have type b_1 or b_2. These product types and their attribute values are specified by the two product catalogs in table 1. The product catalog for component i defines a constraint on the variables $y_i, y_{i,1}, y_{i,2}, y_{i,3}$ such that the table entries are the admissible tuples that may be assigned to the variables of this constraint. The customer does not want any combination of those product types, but requires that the sum of the $y_{1,j}$'s exceeds a parameter p_1 and the sum of the $y_{2,j}$'s exceeds a parameter p_2.

$$y_{1,a} + y_{1,b} \geq p_1$$
$$y_{2,a} + y_{2,b} \geq p_2$$

Furthermore, the customer wants to maximize the sum x of the $y_{3,j}$'s.

$$y_{3,a} + y_{3,b} = x$$

Each variable has a positive integer domain which is limited by a suitably chosen upper bound. Greater parameter values lead to stricter problems meaning that the orders $\geq_i$ correspond to the order $\geq$ of the integers. Similarly, the order $\geq^x$ on the decision space corresponds to this order since greater values are preferred.

This example is kept small for didactic reasons. Real configuration problems in the car and computer industries will have more components, additional constraints such as compatibility constraints between the component types, and much larger product catalogs. However, the number of parameters and decision variables will usually be small.

We give a rational policy for this example. It may happen that there is no configuration that meets the customer request. In this case, the policy chooses the worst value $x^{\perp}$ in the decision space, which is 0 in this example. In this case, the request will be rejected. Furthermore, the resulting policy does not include the values of the auxiliary variables as they do not impact the choice behaviour of the system. As the preference order $\geq^x$ is not expressed on the auxiliary variables, two solutions y and y' that differ in the values of the auxiliary variables, but have the same decision $y_x = y'_x$, are indifferent under the order $\geq^x$. Any of these solution can be kept as a support for this decision and displayed to the user. We do not detail this step in this paper. We thus obtain a policy that determines a

decision depending on the parameters:

$$\begin{array}{l}
\text{if } p_1 \geq 0 \wedge p_1 \leq 20 \wedge p_2 \geq 0 \wedge p_2 \leq 30 \text{ then } x := 60. \\
\text{if } p_1 \leq 30 \wedge p_2 \geq 31 \wedge p_2 \leq 50 \text{ then } x := 40. \\
\text{if } p_1 \leq 40 \wedge p_2 \geq 51 \wedge p_2 \leq 60 \text{ then } x := 20. \\
\text{if } p_1 \geq 0 \wedge p_2 \geq 61 \text{ then x := 0.} \\
\text{if } p_1 \geq 21 \wedge p_1 \leq 30 \wedge p_2 \leq 50 \text{ then } x := 40. \\
\text{if } p_1 \geq 31 \wedge p_1 \leq 40 \wedge p_2 \leq 60 \text{ then } x := 20. \\
\text{if } p_1 \geq 41 \wedge p_2 \geq 0 \text{ then x := 0.}
\end{array} \tag{3}$$

As explained above, experts can easily analyze and refine such a policy. Hence, the rules are sufficient as long as no question about the rationality of the policy is asked. But now suppose that two new component types a_3 and b_3 are introduced as shown in table 2. Is the policy still rational?

4 Policy Design by Exhaustive Optimization

This question is not trivial as the set of options is specified implicitly in form of a constraint satisfaction problem. A naive approach consists in exploring the parameter space completely and to solve a combinatorial optimization problem for each value v in the parameter space.

We can formulate this problem as that of finding a best lower bound for the decision with respect to the order $\geq^x$. The value l is a lower bound for x under parameter values v iff there is a solution $y \in Sol(C)$ that satisfies $y_p = v$ and $y_x \geq^x l$. Alternatively, we can also consider strict upper bounds and determine a worst element among those bounds. A value u is a strict upper bound for the decision under parameter values v iff there is no solution $y \in Sol(C)$ that satisfies the conditions $y_p = v$ and $y_x \geq^x u$. Now let l^* be the best lower bound and u^* the worst strict upper bound with respect to the order $\geq^x$

As u^* is a strict upper bound, the following property holds for all solutions $y \in Sol(C)$:

$$y_p = v \Rightarrow y_x <^x u^* \tag{4}$$

This rule will reduce the set of possible decisions to the set $\{x \in \mathcal{X} \mid x \leq^x l^*\}$ if $y_p = v$ holds. Hence, l^* is the optimal decision in this case and it is feasible. We can therefore incorporate the following rule in our policy:

If $p = v$ then $x := l^*$

Repeating this process for the other parameter values will produce the unique rational policy.

Although the combinatorial problem specified by constraints and preferences has a unique rational policy, there are different ways to represent it by rules. The naive approach generates a rule for each element of the parameter space. This is costly in space and also in time since each rule requires to solve a combinatorial optimization problem.

5 Policy Design by Pareto-Optimization

In this section, we will show how to compute compact representations of policies for problems where the set of feasible options satisfies the monotonicity property (1). In this case, we can replace the value assignments to parameters by bounds on the parameters and eliminate redundant rules.

First, we reconsider rules of the form $y_p = v \Rightarrow y_x <^x u^*$. The monotonicity assumption (1) means that we cannot obtain better decisions when replacing v by any value v' that is stricter than v. Hence, the following property holds for all those values and all solutions y of C:

$$y_p = v' \Rightarrow y_x <^x u^* \tag{5}$$

Hence, we obtain an equivalent rule if we replace the condition $y_p = v$ by the relaxed condition $y_p \succsim^p v$:

$$y_p \succsim^p v \Rightarrow y_x <^x u^* \tag{6}$$

These rules are in a format proposed in [4]. If y_p is at least as good as v, then y_x is strictly worse than u^*. They can be encoded in production rule systems by using the best lower bound l^*:

if $p \succsim^p v$ then $x := min(x, l^*)$

Those rules are confluent, meaning that there is a unique result, which complies to the rational policy. Hence, the first step has transformed the rules into a format which is easier to manipulate. When we reformulate the policy (3) in this format, the upper-bound conditions disappear:

$$\begin{array}{l}
\text{if } p_1 \geq 0 \wedge p_2 \geq 0 \text{ then } x := min(x, 60). \\
\text{if } p_1 \geq 0 \wedge p_2 \geq 31 \text{ then } x := min(x, 40). \\
\text{if } p_1 \geq 0 \wedge p_2 \geq 51 \text{ then } x := min(x, 20). \\
\text{if } p_1 \geq 0 \wedge p_2 \geq 61 \text{ then } x := min(x, 0). \\
\text{if } p_1 \geq 21 \wedge p_2 \geq 0 \text{ then } x := min(x, 40). \\
\text{if } p_1 \geq 31 \wedge p_2 \geq 0 \text{ then } x := min(x, 20). \\
\text{if } p_1 \geq 41 \wedge p_2 \geq 0 \text{ then } x := min(x, 0).
\end{array} \tag{7}$$

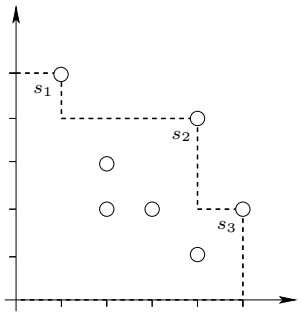

Figure 1: Pareto-frontier.

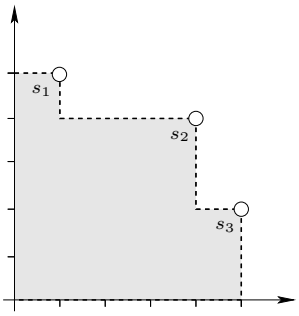

Figure 2: Dominated space.

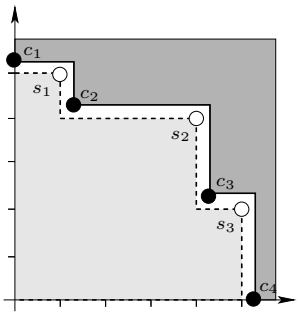

Figure 3: Dual frontiers.

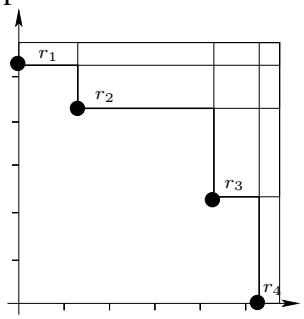

Figure 4: Rules.

Now consider two values v and v^* from the parameter space and two values u and u^* from the decision space. We consider the combined space $\mathcal{Z} := \mathcal{P} \times \mathcal{X}$ and define a weak Pareto-ordering $\succsim^z$ on this space by combining $\succsim^p$ and $\geq^x$. We define $vu \succsim^z v^*u^*$ iff $v \succsim^p v^*$ and $u \geq^x u^*$. Now consider two rules such that vu is as least as good as v^*u^* w.r..t the weak Pareto-order:

$$\begin{array}{c} y_p \succsim^p v \Rightarrow y_x <^x u \\ y_p \succsim^p v^* \Rightarrow y_x <^x u^* \end{array} \tag{8}$$

Since $v \succsim^p v^*$, the condition $y_p \succsim^p v$ of the first rule implies the condition $y_p \succsim^p v^*$ of the second rule. Moreover, since $u \geq^x u^*$, we also have $u^* \leq^x u$ and the conclusion $y_x <^x u^*$ of the second rule implies the conclusion $y_x <^x u$ of the first rule. Hence, the first rule is redundant and can be removed if it is different to the second one. When removing the redundant rules, we obtain a rule-set which is logically equivalent to the original one, but contains only rules for critical pairs (v^*, u^*). As a consequence, we obtain a more compact representation of the rational policy.

We can obtain those pairs as results of the following optimization problem. We consider the set of all candidate vectors v and u such that the following condition is satisfied

by all solutions y of C:

$$y_p \succsim^p v \Rightarrow y_x <^x u \tag{9}$$

and we determine those that are worst w.r.t. the Pareto-ordering for $\succsim^p$ and $\geq^x$. The candidate pairs are exactly those vectors v and u for which there is no solution y of C that satisfies the following condition:

$$y_p \succsim^p v \wedge y_x \geq^x u \tag{10}$$

As this condition is equivalent to $y_p y_x \succsim^z vu$, we can also say that we seek vectors w from $\mathcal{Z}$ that are not dominated by any solution w.r.t. to the combined Pareto-order $\succsim^z$ and which thus constitute infeasible lower bounds. Among all those infeasible lower bounds, we want to determine the worst elements w.r.t. the Pareto-ordering. Hence, we end up with the problem of doing a Pareto-minimization over the space of infeasible lower bounds.

6 Computing the Policy by a Dual Approach

In this section, we map the Pareto-minimization problem over the space of infeasible lower bounds to a Pareto-maximization problem over the solution space. This will allow us to solve the original policy design problem.

Given a vector y from $\mathcal{D}$, we write y_z for $y_p y_x$, i.e. for the tuple $(y_1, \ldots, y_n, y_{n+1})$. We consider the elements of $\mathcal{Z}$ that are supported by solutions y of $Sol(C)$:

$$S := \{w \in \mathcal{Z} \mid \text{there is } y \in Sol(C) \text{ s.t. } y_z = w\} \tag{11}$$

An element w of $\mathcal{Z}$ is a Pareto-maximal element of S iff there is no other element w^* of S that dominates w w.r.t. to the strict part $\succ^z$ of the preorder $\succsim^z$ (i.e. $w^* \succsim^z w$ holds, but not $w \succsim^z w^*$). We denote the set of Pareto-maximal elements of S by $Max(S, \succsim^z)$. Figure 1 shows the Pareto-maximal solutions for a two-dimensional space.

The Pareto-frontier of the constraint problem C can be computed by different methods. We briefly explain the method in [7]. This method does an outer branching which splits the Pareto-frontier in disjoint parts and which is thus different from the usual inner branching which splits the solution set of C in disjoint parts. In each branching step, we compute one Pareto-optimal solution by solving an ordinary lexicographical optimization problem. If w^* is the optimal value obtained from this step, then we consider all Pareto-optimal solutions that assign w^* to y_z in the left branch and all other solutions in the right branch. The left branch is already solved. In the right branch, we add the constraint $y_z \not\succsim^z w^*$ to the constraint set C and repeat the approach. The algorithm is summarized in figure 5. It enumerates the Pareto-optimal solutions in decreasing lexicographical order.

We now divide the space $\mathcal{Z}$ into two disjoint subspaces:

- The space of feasible lower bounds:

$$P := \{w \in \mathcal{Z} \mid \text{there is } y \in Sol(C) \text{ s.t. } y_z \succsim^z w\} \tag{12}$$

- The space of infeasible lower bounds:

$$N := \{w \in \mathcal{Z} \mid \text{there is no } y \in Sol(C) \text{ s.t. } y_z \succsim^z w\} \tag{13}$$

The space of feasible lower bounds contains all elements of $\mathcal{Z}$ that are weakly dominated by some Pareto-maximal element of Z. Hence, the set P can be characterized in terms of the Pareto-maximal elements of S:

$$P = \{w \in \mathcal{Z} \mid \text{there is } w^* \in Max(S, \succsim^z) \text{ s.t. } w^* \succsim^z w\} \tag{14}$$

Figure 2 illustrates this property. As a consequence, the set of Pareto-maximal elements of S and the set of Pareto-maximal elements of P coincide.

We now consider the dual problem, namely that of finding minimal elements in the space of infeasible bounds (see figure 3). This is the set of all elements of N which do not dominate any other element of N w.r.t. the strict part of the order $\succsim^z$. We denote this set by $Min(N, \succsim^z)$. It contains the Pareto-minimal infeasible lower bounds which are needed to generate the rules as explained in the last section. The set N contains exactly those elements of $\mathcal{Z}$ that weakly dominate such an infeasible lower bound:

$$N = \{w \in \mathcal{Z} \mid \text{there is } w^* \in Min(N, \succsim^z) \text{ s.t. } w \succsim^z w^*\} \tag{15}$$

Hence the space $\mathcal{Z}$ is partitioned into a part P that is weakly dominated by the Pareto-frontier w.r.t. the order $\succsim^z$ and a part N that is weakly dominated by the dual frontier w.r.t. the inverse order $\precsim^z$ as illustrated in Figure 3. Hence, there is a duality between the minimization of the infeasible lower bounds and the maximization of the feasible lower bounds under a preorder $\succsim$. For the particular case of Pareto-ordering, the duality result leads to the following equivalence, which holds for each element y of $\mathcal{D}$:

$$\bigvee_{w^* \in \mathrm{Min}(N, \succsim)} \bigwedge_i y_i \geq w_i^* \equiv \bigwedge_{w^* \in \mathrm{Max}(P, \succsim)} \bigvee_i y_i > w_i^* \tag{16}$$

A detailled study of duality notions can be found in the literature of multiobjective programming. For example, the duality result above has been stated in [5]. It can also be derived from propositions 4 and 5 in [8].

We now exploit this duality property to determine the set $Min(N, \succsim^z)$. We first determine N by using (14):

$$N = \{w \in \mathcal{Z} \mid w^* \not\succsim^z w \text{ for all } w^* \in Max(S, \succsim^z)\} \tag{17}$$

As the preorder $\succsim^z$ is a Pareto-ordering, we can describe this space in terms of constraints. For each element w^* of $Max(P, \succsim^z)$, we introduce a constraint with scope $1, \ldots, n, n+1$. The constraint is violated by the vectors that are weakly dominated by w^*. Hence, its relation contains all elements w of $\mathcal{Z}$ that satisfy the condition $w \not\precsim^z w^*$, which is equivalent to $\bigvee_i w_i > w_i^*$ as $\succsim^z$ is a Pareto-ordering. We denote the set of all those constraints by C_N. The solutions of C_N contain all vectors y from $\mathcal{D}$ that satisfy the condition $\bigwedge_{w^* \in Max(P,\succsim^z)} \bigvee_i y_i > w_i^*$. The set N is obtained by projecting these solutions to the indexes z, i.e. $N = \{y_z \mid y \in Sol(C_N)\}$.

We are now able to compute the Pareto-minimal solutions of the new problem C_N and to extract the dual frontier $Min(N, \succsim^z)$ from it. We use outer branching again for this step. We then introduce a rule for each element of this frontier in the format of [4]. The policy design algorithm in figure 6 performs these steps and computes the rational policy.

In the configuration example, there are five Pareto-optimal solutions after the update of the catalog, namely $(20, 30, 60)$, $(30, 50, 40)$, $(20, 50, 50)$, $(40, 60, 20)$, $(30, 60, 30)$. Consequently, each element y of N satisfies the following constraints:

$$\begin{array}{l} y_1 > 20 \vee y_2 > 30 \vee y_3 > 60 \\ y_1 > 30 \vee y_2 > 50 \vee y_3 > 40 \\ y_1 > 20 \vee y_2 > 50 \vee y_3 > 50 \\ y_1 > 40 \vee y_2 > 60 \vee y_3 > 20 \\ y_1 > 30 \vee y_2 > 60 \vee y_3 > 30 \end{array} \tag{18}$$

Pareto-minimization under these constraints yields the minimal infeasible lower bounds:

$$\begin{array}{c} Min(N, \precsim^z) = \{(0, 0, 61), (0, 31, 51), (0, 51, 31), \\ (0, 61, 0), (21, 0, 41), (31, 0, 21), (41, 0, 0)\} \end{array} \tag{19}$$

As a result, we obtain a new policy (note that y_1 corresponds to the parameter p_1, y_2 corresponds to the parameter p_2 and y_3 corresponds to the decision x):

$$\begin{array}{l} \text{if } p_1 \geq 0 \wedge p_2 \geq 0 \text{ then } x := min(x, 60). \\ \text{if } p_1 \geq 0 \wedge p_2 \geq 31 \text{ then } x := min(x, 50). \\ \text{if } p_1 \geq 0 \wedge p_2 \geq 51 \text{ then } x := min(x, 30). \\ \text{if } p_1 \geq 0 \wedge p_2 \geq 61 \text{ then } x := min(x, 0). \\ \text{if } p_1 \geq 21 \wedge p_2 \geq 0 \text{ then } x := min(x, 40). \\ \text{if } p_1 \geq 30 \wedge p_2 \geq 0 \text{ then } x := min(x, 20). \\ \text{if } p_1 \geq 41 \wedge p_2 \geq 0 \text{ then } x := min(x, 0). \end{array} \tag{20}$$

The second and third rules now assign a better value thanks to the new product types. The policy designer thus discovers meaningful changes of the policies, which are difficult to detect manually.

Algorithm Pareto-OB(z, $\succsim^z$, C)

1. $R := \emptyset$;
2. let $\succ^z$ be the strict part of $\succsim^z$;
3. let $>$ be a lexicographic order extending $\succ^z$;
4. **repeat**
5. **if** C has no solution **then** return R;
6. find a $>$-optimal solution y of C;
7. let w be y_z;
8. $R := R \cup \{w\}$;
9. $C := C \wedge \bigvee_i z_i >_i w_i$;

Figure 5: Computing Pareto-maximal solutions.

Algorithm PolicyDesigner(C, p, $\succsim^p$, x, $\geq^x$)

10. let $\succsim^z$ be the weak Pareto-order for $\succsim^p$ and $\geq^x$;
11. let L be the result of Pareto-OB(px, $\succsim^z$, C);
12. let C_N be $\bigwedge_{v \in L} \bigvee_i v_i > z_i$;
13. let U be the result of Pareto-OB(px, $\precsim^z$, C_N);
14. $R := \emptyset$;
15. **for each** $vw \in U$ **do**
16. add $y_p \succsim^p v \Rightarrow y_x <^x w$ to R;
17. return R;

Figure 6: Policy design.

7 Conclusion

Decision-making policies as used in business automation and recommender systems can explicitly be represented in forms of rules. Those rules can directly be acquired from the experts or learned from historical data. In this paper, we elaborated a third method of policy design, which consists in deriving the rules from a domain model which is described in terms of variables, constraints and preferences. Our approach consists in two step. Firstly, we determine Pareto-optimal solutions of the constraint problem, which represent the interesting trade-offs between the possible input and output of the system. Secondly, we convert the Pareto-frontier into a dual frontier by a logical transformation. Each point in the dual frontier creates a rule. Not only the resulting policy is consistent and complete, but also makes optimal choices under the given preference order. The approach is interesting for configuration problems and complex pricing problems where explicit representations of policies are desired, but frequent changes in product catalogs or

marketing strategies make it difficult to establish those rules. The paper provides a preliminary study of methods that derive those policies automatically. As the generated rule-sets may be large in size, future work is needed to study other rule formats or approximation techniques to obtain rule-sets of manageable size.

References

[1] Kenneth Arrow. Rational choice functions and orderings. *Economica*, 26:121–127, 1959.

[2] Virginia E. Barker, Dennis E. O'Connor, Judith Bachant, and Elliot Soloway. Expert systems for configuration at Digital: XCON and beyond. *Commununications of the ACM*, 32(3):298–318, 1989.

[3] Edward A. Feigenbaum. The art of artificial intelligence: Themes and case studies of knowledge engineering. In *IJCAI*, pages 1014–1029, 1977.

[4] S. Greco, B. Matarazzo, and R. Slowinski. Rough sets theory for multicriteria decision analysis. *European Journal of Operational Research*, 129:1–47, 2001.

[5] Thomas Hanne. On utilizing infeasibility in multiobjective evolutionary algorithms. In *MOPGP'06*, 2006.

[6] Ulrich Junker. Configuration. In *Handbook of Constraint Programming*, pages 837–873. Elsevier, 2006.

[7] Ulrich Junker. Outer branching: How to optimize under partial orders? In *ECAI-06 Workshop on Advances in Preference Handling*, pages 58–64, 2006.

[8] Ulrich Junker. Preference-based inconsistency proving: When the failure of the best is sufficient. In *ECAI*, pages 118–122, 2006.