How to make a best-seller: optimal product design problems

Ismael Rodríguez, Pablo Rabanal, Fernando Rubio[a,*]

[a]*Dept. Sistemas Informáticos y Computación. Facultad de Informática*
Universidad Complutense de Madrid, 28040 Madrid, Spain

Abstract

We formalize and analyze the computational complexity of three problems which are at the keystone of any marketing management process. Given the preferences of customers over the attribute values we may assign to our product (i.e. its possible features), the attribute values of products sold by our competitors, and which attribute values are available to each producer (due to e.g. technological limitations, legal issues, or availability of resources), we consider the following problems: (a) select the attributes of our product in such a way that the number of customers is maximized; (b) find out whether there is a feasible strategy guaranteeing that, at some point in the future before some deadline, we will reach a given average number of customers during some period of time; and (c) the same question as (b), though the number of steps before the deadline is restricted to be, at most, the number of attributes. We prove that these problems are Poly-APX-complete, EXPTIME-complete, and PSPACE-complete, respectively. After presenting these theoretical properties, heuristic methods based on genetic, swarm and minimax algorithms are proposed to suboptimally solve these problems. We report experimental results where these methods are applied to solve some artificially-designed problem instances, and next we present a case study, based on real data, where these algorithms are applied to a particular kind of product: We automatically design the *political platform* of a political party to maximize its numbers of votes in an election (problem (a)) and its number of supporters along time (problems (b) and (c)). The problem instances solved in this case study are constructed from publicly released polls on political tendencies in Spain.

Keywords: Marketing management, Product design, Computational complexity, Evolutionary computation, Heuristic methods, Non-cooperative Games

1. Introduction

Economics and Computer Science are significant sources of inspiration to solve the problems of each other. On the one hand, Computer Science has borrowed concepts

*Corresponding author

Email addresses: isrodrig@sip.ucm.es (Ismael Rodríguez), prabanal@fdi.ucm.es (Pablo Rabanal), fernando@sip.ucm.es (Fernando Rubio)

from Economics to face tasks such as delivering scarce computational resources (e.g. storage capacity, access to the CPU, etc) in distributed environments [1, 2, 3, 4] or even proposing new programming styles [5]. On the other hand, Computer Science has been used to face classical problems in Economics such as solving complex auctions [6, 7], implementing economic simulators [8], or even identifying probable limits of any economic society [9]. Moreover, in some cases the barrier between both disciplines fades away, such as in the development of efficient e-commerce systems [10]. Economic notions have even been used at the core of the definition of specification languages such as process algebra [11, 12].

In this paper we study the feasibility of automating one of the most important tasks faced in marketing management: deciding what product should be produced and sold [13]. In particular, which features should the product have to attract a sufficient number of customers? In order to answer this question, those responsible for deciding how the product or service should be must take into account their (formal or informal) knowledge about the preferences of customers, as well as the features of products sold by competitors. For instance, let us suppose that we lead a shampoo manufacturing company and we have to decide the features of our product. Should the gel be green or blue? Should it smell like coconut or vanilla? Should it be dense? Should it be cheap? What attributes should the product have? By means of market research and other preference elicitation techniques [14, 15], we may have some knowledge about the preferences of several types of customers. For instance, for some group of potential customers having an specific age, gender, and income, we may have some partial knowledge about their preferences, such as whether they prefer green over blue, or that their valuation of "vanilla" is "4 out of 5" and their valuation of "coconut" is "3 out of 5." Based on these preferences over individual attributes, each customer type (customer *profile*) will prefer some available products over others. We should also consider the features of the products sold by our competitors. Moreover, we should take into account our manufacturing limitations due to legal or technological reasons (e.g. our company does not know how to produce a vanilla smell), as well as the limitations of our competitors —at least, those we actually know. Given that information about customer profiles and competitors, in this paper we will study the following problems:

(a) Let us suppose that we design a new product fully from scratch. What product should we produce (i.e. how should we select its attributes) to maximize the number of customers or to reach a given target number of customers?

(b) Considering that the competitors will do nothing after we have selected a good product to sell is naive, so let us consider that producers play the following game. In each turn, a producer changes some attribute values of its product (e.g. replaces vanilla smell by coconut smell, etc), and next all customers select their favorite products. Any product change requires time or money, so we will consider that, in each turn, the corresponding producer can change only up to D attribute values of its product, for some known $D \in \mathbb{N}$. Is there a strategy we can follow such that, for any choices made by our competitors, we will always reach a given number of customers before a given deadline turn? Note that we could reach a good number of customers in some future turn, and yet it could

not be a good strategy for our company. For instance, at some turn we could get many customers by selling a product that is slightly better than other choices for a big number of heterogeneous customer profiles. However, right after that, other producers could change their products to aim only at specific customer profiles, offering very tailored products that are much better than ours for each specific customer profile. In this case, we would lose most of our customers in the next turn. Thus, despite our previous success in a single turn, this might not be a sustainable policy for our company. Rather than this, we will ask the following question: Can some strategy guarantee that, for any choices made by our competitors, the average number of our customers during the last t_p turns, for some given $t_p \in \mathbb{N}$, will reach a given target number at some future turn before a given deadline?

(c) In order to check if a simpler version of our problem is easier to handle, we also consider a version of problem (b) where the deadline is restricted: the number of turns until the deadline cannot be higher than the number of attributes of products.

In this paper we formally analyze the difficulty to computationally solve problems (a), (b), and (c), we apply heuristic algorithms to solve them, and we report their performance in some experiments. We prove that these problems are NP-complete, EXPTIME-complete, and PSPACE-complete, respectively, which proves that optimally solving them is intractable (i.e. they cannot be solved in polynomial time) under some standard assumptions (respectively: if the widely believed property $\mathtt{P} \neq \mathtt{NP}$ holds; without any additional consideration; and if the even more widely believed property $\mathtt{P} \neq \mathtt{PSPACE}$ holds). Though optimally solving these problems for very small instances may be feasible, it is clearly unfeasible for most of instances with reasonable sizes.[1] This intractability forces the use of heuristic solutions to suboptimally solve these problems. Unfortunately, not even the simplest problem, i.e. problem (a), can be well *approximated* in polynomial time in the worst case: we also prove that this problem is Poly-APX-complete, meaning that it cannot belong to any interesting approximation class such as FPTAS, PTAS, or APX (not even Log-APX) as long as $\mathtt{P} \neq \mathtt{NP}$.

Given the lack of problem-specific polynomial-time heuristics reaching some known constant performance in the worst case, implied by its Poly-APX-completeness, we heuristically solve problem (a) by means of generic-purpose heuristics, in particular a genetic algorithm [16] and particle swarm optimization [17]. Besides, a minimax algorithm is used to solve problems (b) and (c). Experiments where the performance

[1] For instance, let us suppose that we want to select the best combination of attributes for a car. These attributes include many details concerning the car itself (dozens of engine features, the form of the seats, the space between seats, the form and volume of the trunk, etc), the promotion campaign (the kind of customers the campaign is primarily oriented to, where advertisements are placed, etc), or even the availability to buy the car in installments (interest rate, conditions for applicability, etc), among others. Each of them consciously or unconsciously matters to a significant number of customers. Let us suppose that 100 attributes are considered. In a standard personal computer as of 2015, solving this problem by means of a typical straightforward brute-force algorithm taking e.g. 2^n steps (where n is the number of attributes) would take more time than the age of the universe.

of these algorithms is compared with the performance of some straightforward greedy solutions are presented. Results show that, despite the intractability of finding optimal solutions, these heuristic solutions are clearly better than straightforward strategies. In addition, we also present a case study where the product to be designed by our heuristic methods is the *political platform* of a party (i.e. its set of stances towards the political issues that matter voters). Problem instances concerned in the case study are created from real political tendencies collected by polls conducted in Spain. In this case, the goal of the heuristics methods is automatically designing the political platform of a party.

The rest of the paper is structured as follows. In the next section we discuss the differences between the problems considered in this paper and other related problems mentioned in the literature. Then, problems (a), (b), and (c) are formally defined in Section 3, where we also prove their computational intractability. In Section 4 we describe our heuristic algorithms to (suboptimally) solve them, and we report their performance in several experiments involving artificially-designed problem instances. Our case study, concerning the automatic creation of political platforms in a scenario based on real political polls, is presented in Section 5. The conclusions and future work plans are given in Section 6. For the sake of readability, all proofs of the theoretical results given in Section 3 are shown in the appendix of the paper.

2. Related Work

To the best of our knowledge, the problems of designing products to maximize the number of customers, either immediately (problem (a)) or in long/short term (problems (b) and (c), respectively), have never been studied before, not to mention their computational complexity. Next we mention some problems that are somehow related with them.

Market segmentation techniques [18] certainly have some relation to our problem. In market segmentation, the goal is to divide the whole set of customers into subsets defined by their characteristics, in such a way that the original heterogeneous set of customers is classified into smaller and (somehow) homogeneous sets. By doing so, the task of designing strategies for tailoring products for specific customer groups can be eased. In our case, part of the *input* of our problems is the set of customer profiles, i.e. the problem inputs are *already* segmented, so our goal is not to study how to perform such segmentation, but to use the given segmentation to solve another problem.

In the literature we can find several maximization problems that are related to ours. For instance, given a set of projects that can be carried out or not, the *combinatorial public projects* problem [19, 20] consists in selecting the subset of K projects that provides more utility to a collective of users, taking into account the individual preferences of these users. In particular, the *addition* of the utility of all users is maximized. Projects can be combined in any way as long as the number of projects stays below the given threshold (on the contrary, in our product design problems (a)-(c), exactly one value must be given to each attribute of the product, e.g. our shampoo must have exactly one color out of e.g. seven given possible colors). Note that the aim of the combinatorial public projects problem is to optimize the social welfare, whereas the goal of our problems is providing a product that is preferred over the products of our

competitors. That is, we only need our product to be *slightly better* than competitor products for many customers so that they choose it, rather than a product maximizing the addition of utility of all customers. In particular, we do not aim at maximizing the utility of those customers choosing our product (we only need to offer them a product being slightly better than other choices), and the utility of customers not choosing our product does not give us any profit at all, so increasing it is pointless —as long as they still prefer other products. Hence, the goal of our product design problems is not the social welfare.

The *facility location* problem [21] consists in deciding which facilities are opened, and how customers are assigned to open facilities, in such a way that the addition of costs due to opening these facilities and serving customers from them is minimized. A *service cost* is defined for each possible pair of facility and customer (alternatively, they can be defined simply in terms of geometrical distances between their corresponding points in the space). Here we will be interested in the facility location problem variant where different subsets of facilities can be controlled by different *competing* companies (see e.g. [22, 23]). In order to compare this problem with our product design problems, we could consider the products of our problems as possible *facility locations*: each location denotes a possible product we could produce, and the distance from a customer profile to a location is the preference of this profile towards the particular product denoted by that location. In our product design problems, we will numerically represent the preference of each customer profile towards each possible *attribute value* (e.g. in the shampoo example commented earlier, each customer profile would have a valuation towards each available color, each available density, etc). Hence, if the preference of each customer profile towards each potential *product* (a combination of attribute values) had to be expressed by a different number (or, as it is more customary in facility location problems, by an edge between them, where the edge cost is that number), then an explosion of these numbers (edges) would be produced: a pair would be required for each combination of customer and *possible product*, so the resulting model would be unfeasible. On the other hand, plain geometrical distances between points representing products and customers are far to provide enough expressivity to denote the preferences in our model.

Note that, in facility location problems, the difficulty emerges from the necessity to place several facilities to cover the (demand) space. On the contrary, in our product design problems, the difficulty emerges just from the combinatorial possibilities to design each product, even if a single product is sold by each company (in fact, this is the case considered in this paper). Some works have further developed the original facility location problem setting into much complex models, for instance letting decision makers build a set of improvements (e.g. a bigger facility, more parking spots, etc) that affect the attractiveness value of each location [24]. At each location, the attractiveness increment caused by each improvement equally affects all customers. On the contrary, in our product design problems, the core difficulty is the existence of *incompatible* preferences: some customers may like changing the shampoo color from blue to green, whereas other customers may *dislike* it. Each attribute value (blue, green) gets our product closer to some audience and farther from other audience, and each attribute (color, density, etc) can split and polarize the audience in a completely different way. Our product design problems aim at harmonizing these orthogonal attribute-

driven preference views into a single product preferred by many customers.

Other related works in the literature concern voting systems. As we will comment later, we chose *voting* and *elections* as the background of our case study in Section 5 because, for that domain, extensive stratified market research data (in particular, *political polls*) are publicly available (see Section 5). However, one could think that this particular domain (voting and elections) is illustrative of the kind of problems our general marketing problems can be related with. In the problem of *winner determination in voting protocols* (see e.g. [25]) the goal is *determining* the winner of an election according to the preferences of the voters, i.e. deciding who wins. Note that, on the contrary, in our problem the goal is deciding the strategy to maximize the number of customers (*votes*, in the context of the case study in Section 5). The winner determination, that is, given some preferences determining the winner, is certainly a *part* of our problem because our problem requires, in its intermediate steps, determining the products (or *parties*) selected by several customer profiles. However, in our problem this is not the *goal* but just a part of a bigger task (and, as we will see later, that part will be easy in our problem, as our setting does not focus on it). Our real goal is designing a product reaching many customers in a competitive environment, either in one step or interactively. In the context of the case study given in Section 5, by assuming that the products of problems (a)-(c) are political platforms, we get the problems of selecting the political platform of a candidate to maximize its number of votes in an election (in problem (a)) or its number of supporters during a period of time (in (b) and (c)). Although determining the winner will be easy in our setting, it is worth mentioning that this problem is much more complex in other scenarios (for instance, when dealing with proportional representation voting systems [26], the winner selection problem alone is NP-complete if the number of winners is not constant, that is, if the number of winners depends on the numbers of voters, see [27]).

Let us note that *winner determination* is also the term used in *combinatorial auctions* [28] to denote the problem of deciding how to assign some items among some bidders who previously bid for specific *bundles* of items, in such a way the auctioneer maximizes his profit. This problem is also different to our problem, since our aim is to *design* a product in such a way that it gets as many customers as possible, whereas the aim of winner determination is to partition a given set of (fix) items into bundles and deliver them in such a way that the money collected from all assigned bundles is maximized.

Let us also note that, in our problems (b) and (c), conditions change along the system evolution, so their game nature does not fit into any constant-state optimization problem like those commented before.

In Economics and Marketing domains, the task of making all decisions around the product, in order to reach more customers or maximize the profit, is known as *marketing mix*. Several marketing mix theories have been proposed to highlight the aspects product designers should focus on in this process: McCarthy's four Ps [29] (*product*, *price*, *promotion*, and *place*), Lauterborn's 4 Ps [30], and several variations on them (see e.g. [31, 32, 33]). All of these models are descriptive, that is, they do not formalize the problem in a pure mathematical form (e.g. as a maximization problem), but they focus on exhaustively identifying the factors that must be taken into account to properly put a product into the market. Note that our model will not treat in a

different way attribute values such as, for instance, "*blue*", "*cheap*", "*the ads will focus on a young audience*", or "*sell it in drug stores*" (which would belong, respectively, to product, price, promotion, and place categories according to McCarthy's four Ps). Our model will let us individually set the effect of each attribute value over the customer preferences, so explicitly splitting the attributes among all four Ps is unnecessary in the model.

In addition to these general descriptive models, some works have also focused on quantifying more specific aspects around the process of manufacturing products (see [34] for an interesting review on the topic). For instance, the theory of constraints [35] has been used to analyze (and try to eliminate) constraints that affect the production of goods at different levels. For example, it can be used to analyze bottlenecks when a product requires manufacturing several independent parts that take varying production times using common machines. However, as it is discussed in [36], the theory of constraints only obtains good results under the assumption that the constraints are relatively simple. Other quantitative approach that has been widely used is simulation models. They have been used, for instance, to improve the whole supply chain from supply sources to the customer [37] or to optimize the manufacturing scheduling: given limited manufacturing resources, optimize their allocation [38].

Game theory has also been considered in the context of interactions between companies. For instance, it has been used to analyze the interest of alliances between companies [39], the relations between manufacturers and retailers [40], and even how to allocate resources when a company is producing several products: each product is a *player* of the *game* and the aim is to optimize the overall profit of the company [41]. In our problems, the success of a producer generally implies the failure of another, which generally disables cooperative strategies. There could actually exist cooperative strategies in our problems if producers did not aim at maximizing their customers, but just at reaching some given numbers of customers —and no necessarily more. If the target numbers of customers to be reached by all producers were small enough, then there could be strategies letting many (or all) customers simultaneously accomplish their respective numbers. Note that setting some fix target number of customers for each producer is, in fact, a simplification of the actual goal of any real producer: getting as many customers as possible. Hence, the actual goals of producers are mutually exclusive in an intrinsic way.[2]

Several works have also dealt with recommendation systems (see e.g. [42]). Note that the problem of recommending a product (among a list of available problems) is very different to our problem. Assuming that we already know the preferences of the

[2]Note that a *collusion* where producers A and B collaborate against the rest of producers is equivalent to assuming that a single producer AB is allowed to sell *two* products. Even though each producer will sell just one product in our problems (a), (b), and (c), the definitions of these problems and their corresponding computational complexity results apply regardless of whether some producers collude or not. In problem (a), we want to design the best product to sell right now, regardless of whether the products currently sold by e.g. competitors A and B were collaboratively designed or not. In problems (b) and (c), we look for a strategy guaranteeing some number of average customers along time *in any case*, i.e. for any behavior of our competitors —regardless of whether A and B actually collude against us or not. Explicitly studying the effect of collusion in our problems is beyond the scope of this paper, though extending our problems to let producers sell more than one product is easy (this is briefly commented in the next section).

customer towards each possible attribute value, recommending the best product would be trivial under our model: we just have to find the product whose valuation is the highest according to these preferences. In fact, the real difficulty of recommender systems is eliciting the user's preferences by making as less questions as possible (or, even better, with none, if we can elicit her preferences just by silently observing her behavior). On the contrary, in our problems we assume that these preferences are already identified, that is, they are part of the input of our problem. Hence, both problems have a very different focus and goal.

Finally, the topic of product-mix flexibility [43] is also somehow related to our problems (b) and (c), where we iteratively modify our product while our competitors modify their owns. The problem of product-mix flexibility analyzes the ability of a company to keep reasonable profits over time even when the external conditions change. For instance, it considers how to adapt the management of resources, or how to modify the product-mix (i.e., how many products of each kind we should produce) when the demand of the products varies or when the price of the raw materials change. The practical models are restricted to deal with only a few possible products and external conditions, typically adapting the manufacturing system to produce (different amounts of) two different products at the same time [44, 45], or considering up to six different products in the most complex scenarios [46]. Note that our problems (b) and (c) are formally defined as competitive games for several producers, and they concentrate on deciding what new properties (attributes) should be assigned to a (single) product to react to the attributes of the products sold by our competitors, yielding a new product from a huge set of possible products (one for each valid combination of attribute values). Hence, our problems (b) and (c) concern iteratively changing the product to be sold, rather than adapting the manufacturing system itself.

3. Problem Definition and Properties

In this section we define the problems under consideration and prove their intractability. For the sake of readability, the proofs of all results are shown in the appendix.

Note that the definition and the difficulty of our three proposed problems is affected by the model assumed to represent the preference of customer profiles over products. We will consider a particularly simple model: for each customer profile, customers of that profile associate a valuation (a positive integer) to each attribute value, and customers choose the product whose addition of valuations due to all its attribute values is the highest. Note that, for some customers, some features could be negative rather than positive (e.g. some customers do not like peanuts), though a model allowing negative values can be converted into an equivalent model using only natural numbers (we just have to add some constant value to all valuations).

However, the simplicity of our model does not allow to represent other situations which typically appear in human preferences. For instance, some attribute values could increase their valuation if they come together (or if they do *not* come together), and this cannot be represented in our model. The motivation behind the simplicity of the preferences model we are considering is that proving the difficulty of *simpler* problems is more interesting, as any generalization of a difficult problem always has, at least, the

same difficulty as the particular problem. For instance, let us suppose that customer preferences were defined by means of more general functions which, given the attribute values of all available products, return the preferred product. Let us suppose that the only conditions imposed over these functions were these: (i) preferences are transitive (i.e. if we prefer A over B and B over C, then we prefer A over C); and (ii) the functions returning the selected product can be computed in some given polynomial time. In this case, the new versions of problems (a), (b), and (c) would be trivial generalizations of our original problems (note that these functions allow, in particular, representing the simple preferences model we are considering), so they would keep their NP- and Poly-APX-, EXPTIME-, and PSPACE-hardness, respectively.[3] For the same reason, other interesting generalizations trivially keep that complexity hardness. For instance, if we do not measure the success of a producer in terms of the number of achieved customers but in terms of the *money* earned due to the corresponding sales, then we get a generalization of our problems (note that, in the particular case that customers only select a product if its price is 1, all producers would be forced to set their prices to 1, so the money earned by sales would be equal to the number of cumulated customers). Similarly, new problem versions where the attribute values available for each producer may change along time (e.g. due to the technological development or the scarcity of raw materials), or where each producer can sell up to $q \in \mathbb{N}$ products rather than just one (so a producer could make each of his products cover a different kind of customers), also generalize our problems: respectively, attribute values could be kept constant, and we could set $q = 1$. Consequently, all of these variants keep, at least, the mentioned hardness. In order to illustrate this difficulty preservation, the hardness of some simple yet interesting problem variants is formally identified at the end of this section.

Next we formally define our (standard) problems. Given two sets A and B, 2^A denotes the powerset of A, A^* denotes the set of all finite sequences consisting of 0 or more elements from A, and AB (sometimes we will write $A \cdot B$) is the set of all sequences of two elements where the first element is taken from A and the second from B. Given n sequences $a_1, \ldots, a_n \in A^*$, their concatenation is represented by $concat_{1 \leq i \leq n}\, a_i \in A^*$.

Definition 1. Let us consider n product attributes, denoted by $A_1, \ldots, A_n$, where each A_h with $1 \leq h \leq n$ is a set $A_h = \{v_1, \ldots, v_{k_h}\}$ of possible attribute values. We assume that all sets A_h are pairwise disjoint. Let $P = A_1 \times \ldots \times A_n$ and $V = A_1 \cup \ldots \cup A_n$. We consider that a *product* p is a tuple containing a value for each attribute, that is, $p \in P$.

Let us assume that there exist d producers. The j-th producer can select the value of the h-th attribute of the product it sells from a given subset $A_h^j \subseteq A_h$ of values available for this particular producer. Let $P^j = A_1^j \times \ldots \times A_n^j$. The product sold by the j-th producer is a tuple $p_j = (p_1^j, \ldots, p_n^j) \in P^j \subseteq P$.

We also assume that there are m customer profiles, where the i-th customer profile is defined by its preference function $U_i : \underbrace{P \times \ldots \times P}_{d} \longrightarrow 2^{\{1,\ldots,d\}}$. Given the products

[3] In fact, they would remain NP-complete, EXPTIME-complete, and PSPACE-complete, respectively. The approximation class where problem (a) would be included in would depend on how preferences are defined (but it could not be better than Poly-APX).

sold by all d producers, it returns the set of indexes of the producers selling the products preferred by customers in the i-th customer profile. We assume that the number of customers of the i-th profile is $num_i \in \mathbb{N}$.

Let us consider that functions U_i are defined, in particular, as follows. For all customer profiles $1 \leq i \leq m$ and all attribute values $v \in V$, let $val_{i,v} \in \mathbb{N}$ be the *valuation* customers of the i-th profile give to value v. Then, we define

$$U_i(p_1,\ldots,p_d) = argmax\_set_{1\leq j\leq d}\left(\sum_{1\leq h\leq n} val_{i,p_h^j}\right)$$

where for all $1 \leq j \leq d$ we assume $p_j = (p_1^j,\ldots,p_n^j)$, and we consider that *argmax_set* returns the non-empty *set* of all indexes maximizing its target function. Therefore, $U_i(p_1,\ldots,p_d)$ returns the set of indexes of all producers whose products maximize the addition of (i-th profile) attribute value valuations.

The *weighted score* of the j-th producer, representing the number of customers achieved by producer j, is denoted by wsc_j, and is defined as

$$\sum\left\{ \frac{num_i}{|U_i(p_1,\ldots,p_d)|} \;\middle|\; 1 \leq i \leq m \wedge j \in U_i(p_1,\ldots,p_d) \right\}$$

□

Note that, if $U_i(p_1,\ldots,p_d)$ is not a singleton, then we are assuming that customers of customer profile i are equally divided among all producers preferred by this profile (whose set of indexes is denoted by $U_i(p_1,\ldots,p_d)$).

Definition 2. The *product design* problem, denoted as PD, is defined as follows: Given the attributes sets $A_1,\ldots,A_n$ and the subsets of attribute values $A_1^j,\ldots,A_n^j$ available for each producer $1 \leq j \leq d$, the $d-1$ products $p_2 \in P^2,\ldots,p_d \in P^d$ produced by all producers but producer 1, the valuations $val_{i,v}$ of all attribute values $v \in V$ given by all customer profiles $1 \leq i \leq m$, the numbers of customers num_i of each profile $1 \leq i \leq m$, and a number $K \in \mathbb{N}$ (denoting the number of customers producer 1 is required to achieve), is there a product $p_1 \in P^1$ such that $wsc_1 \geq K$?

The *product design optimization* problem, denoted as PDO, is defined as follows: Given the same input values as before except K, find the product $p_1 \in P^1$ maximizing wsc_1. □

Theorem 1. We have:

(i) PD ∈ NP-complete.

(ii) PDO ∈ Poly-APX-complete.

□

As a consequence of the previous result, if P ≠ NP then not only we cannot optimally solve PD in polynomial time, but also we cannot polynomially approximate PDO very well. Since PDO cannot belong to APX as long as P ≠ NP, we cannot build a

polynomial-time algorithm such that the ratio between the number of customers gained in the best solution and the number of customers gained in the solution given by the algorithm is *always* under some known constant threshold. Moreover, as PDO cannot belong to Log-APX as long as P $\neq$ NP, that cannot be done either even if we let that ratio grow logarithmically with the size of the problem, rather than being constant. On the contrary, since PDO $\in$ Poly-APX, this can be done indeed if we let that ratio grow polynomially with respect to the problem size. Actually, as it is shown in the proof of Theorem 1, given in the appendix, we can do it in such a way that the ratio grows linearly with the number of customer profiles. Although this is not necessarily the best algorithm making the performance ratio grow polynomially, algorithms guaranteeing some polynomially-growing performance ratio in the worst case do not seem to be a very competitive solution anyway. In this case, trying algorithms that do not guarantee any (constant or increasing) minimal performance ratio in the worst case, but may provide a good performance on *average* or in many *typical* instances, is a more suitable choice. This is the case of genetic algorithms [16] and particle swarm optimization [17], which will be used to face PDO in sections 4 and 5.

Next we define our game of product design.

Definition 3. The *product design* game, denoted by *PDG*, is defined as follows. Let us assume that $D, t_p, t_f, K \in \mathbb{N}$ with $D > 0$ and $t_p \leq t_f$ are game parameters. In the game, d producers play by turns in some arbitrary order. Each game configuration is defined as a tuple $(t, p_1, \ldots, p_d, g_1, \ldots, g_d)$, where the holder of the current turn is producer $t \in \{1, \ldots, d\}$, each $p_j \in P^j$ denotes the product sold by producer j at the current turn, and each $g_j = (g_j^1, \ldots, g_j^{t_p}) \in \mathbb{Q}^{t_p}$ is a tuple denoting the numbers of customers gathered by producer j in each of the last t_p game turns. The initial game configuration c_0 fulfills the conditions $t = 1$ and $g_1 = \ldots = g_d = \overline{0}$, where $\overline{0}$ denotes a tuple with t_p components where all are 0. In the turn of each producer j, this producer modifies the value of up to D attributes of its product (maybe 0), yielding a new product in P^j. After values are modified, all customer profiles select their favorite products among all available products by applying their functions U_i, each producer j records in g_j the number of customers which now choose its product (that is, the value of wsc_j according to the current configuration is stored in g_j^1, and all g_j^s are shifted to g_j^{s+1}), and the turn is passed to the next producer. A producer j *succeeds* if, for some turn $t_p \leq r \leq t_f$, its average number of customers along the last t_p previous turns is at least K, that is, if $\frac{\sum_{1 \leq s \leq t_p} g_j^s}{t_p} \geq K$ at turn r (thus, game parameter K denotes the average number of customers we require to *succeed*). Note that the game does not end when some producer succeeds, so other producers could also be successful later (i.e. there could be several "winners"). We assume that the initial product of each producer and its subsequent product modifications are public to all players, so it is a perfect information game.

Let c_1, c_2 be game configurations and $z \in 2^V$. We say that the move z makes the game evolve from c_1 to c_2 if, at configuration c_1, a move where the current turn holder adopts the values in z for its product (replacing the previous values of the corresponding attributes), and next all customers select their preferred products, leads to configuration c_2. □

Between two consecutive moves of a given producer j, the sequence of moves made by its $d-1$ opponents can be denoted by a sequence of moves $o = z_1 \dots z_{d-1}$, meaning that the opponent playing immediately after producer j adopts attribute values z_1, then the next opponent adopts z_2 in its turn, and so on. Let O denote the set of all of these sequences. Assuming the current turn holder is $w \in \{1,\dots,d\}$, the sequence of moves made by the opponent players playing immediately *before* producer j can also be denoted by a sequence $o' = z_1 \dots z_r$ where $r = ((j-1)+d-(w-1)) \bmod d$. Let O' denote the set of all of these sequences.

Definition 4. Let c be a configuration where the turn holder is producer $w \in \{1,\dots,d\}$. We consider that a *game strategy* for producer j from configuration c is denoted by a function $\texttt{strategy} : O' \cdot O^* \longrightarrow 2^V$ where O and O' are as before. We assume that $\texttt{strategy}(o_0\ o_1 \dots o_f) = z$ denotes that, if the first moves of the opponent players playing before producer j are those denoted by $o_0 \in O'$, the subsequent moves of all its $d-1$ opponents are those given by $o_1 \in O$, the next moves of these $d-1$ opponents are those of $o_2 \in O$, and so on up to $o_f \in O$, then the producer j chooses z as its next move after all of these moves. □

Note that, given the strategy of the previous definition, $\texttt{strategy}(o_0)$ denotes the move of producer j at the first configuration where it can move after c. Also note that, if we want to define the strategy of some producer j with some function `strategy`, then we do not need to include the previous moves of producer j itself in the sequence of moves denoting the input of function `strategy`, because they are determined indeed by the output of the same function for shorter sequences: $\texttt{strategy}(o_0)$ is the first move of producer j, $\texttt{strategy}(o_0 o_1)$ is its second move, $\texttt{strategy}(o_0 o_1 o_2)$ is the third, and so on.

Definition 5. Let c be a configuration where the turn holder is $w \in \{1,\dots,d\}$ and `strategy` be a game strategy for producer j from configuration c. Let us consider $o = o_0\ o_1 \dots o_f \in O' \cdot O^*$ where $o_0 \in O'$ and, for all $1 \le q \le f$, $o_q \in O$.

The *sequence of configurations* traversed by `strategy` from c through the opponent moves represented by sequence o, denoted by $\texttt{configs}(\texttt{strategy}, c, o)$, is the unique sequence of configurations traversed from c by iteratively performing all moves in o_0, next $\texttt{strategy}(o_0)$, next all moves in o_1, next $\texttt{strategy}(o_0\ o_1)$, next all moves in o_2, next $\texttt{strategy}(o_0\ o_1\ o_2)$, and so on, up to all moves in o_f.

We say that the game strategy `strategy` is *necessarily successful* for producer j from configuration b_0 if, for all $o \in O' \cdot O^*$ yielding t_f game turns (i.e. such that $\texttt{configs}(\texttt{strategy}, b_0, o) = b_0\ b_1 \dots b_{t_f}$ for some $b_1, \dots, b_{t_f}$), there exists $t_p \le r \le t_f$ such that producer j succeeds at b_r. □

Definition 6. The *succeed in the product design game* problem, denoted as `SPDG`, is the following problem: Given a configuration c (not necessarily initial), the game parameters $D, t_p, t_f, K \in \mathbb{N}$, the attributes sets $A_1, \dots, A_n$ and the subsets $A_1^j, \dots, A_n^j$ of values available for each producer $1 \le j \le d$, and the valuations $val_{i,v}$ of all values $v \in V$ for all customer profiles $1 \le i \le m$, is there a necessarily successful strategy for producer 1 from configuration c? □

Next we introduce the complexity of SPDG.

Theorem 2. SPDG $\in$ EXPTIME-complete. □

By the time hierarchy theorem [47], we conclude that SPDG will never be solvable in polynomial time, without the necessity to assume any additional consideration based on any unproved result (e.g. P $\neq$ NP, P $\neq$ PSPACE, etc). Besides, as we can deduce from the proof of Theorem 2, given in the appendix, SPDG would remain in EXPTIME-complete even if we constrained it so that only 2 players played the game and t_p were required to be 1 (i.e. if, in order to determine whether a producer is successful at the current turn, only the number of customers gathered at the *current* turn were considered).

Next we show that the proposed game problem keeps (a different kind of) intractability even if the number of turns is heavily constrained.

Theorem 3. Let the *linear-bounded succeed in the product design game* problem, denoted by lSPDG, be the same problem as SPDG, though only inputs where $t_f \in \mathbb{N}$ is not higher than n, where n is the number of attributes, are considered (for all other inputs, we consider that the answer is *no*). Then, lSPDG $\in$ PSPACE-complete. □

The proof of this result (see the appendix) includes, in its first lines, a quite straightforward method to compute lSPDG in polynomial space and exponential time. Again, this proof lets us infer that lSPDG would remain in PSPACE-complete even if we constrained it so that only 2 players played the game. Contrarily to SPDG, the possibility that lSPDG could ever be solved in polynomial time does exist, although it would require P = PSPACE (which would be even more surprising and unexpected than P = NP).

The intractability of SPDG, and the very likely intractability of lSPDG, motivate solving them by means of heuristic methods such as minimax algorithms, as we will do in sections 4 and 5.

In order to illustrate how easily PDO, SPDG, and lSPDG can be extended to denote other more complex scenarios, next we briefly introduce some samples of these extensions. On the one hand, we let designers explicitly ban some combinations of attribute values. We provide two possible ways for this: either some attribute values are determined by the remaining values (e.g. the weight and the battery consumption of a mobile phone are determined by the remaining attribute values of the phone, according to the skills of the manufacturer), or a function over all attribute values directly determines which combinations are feasible or not. On the other hand, the goal of reaching some numbers of customers in PDO, SPDG, and lSPDG can be replaced by the goal of collecting some *profit* from all sales. So, a producer could succeed by selling many cheap products or by selling a few expensive ones. In this variant, a function over the products (i.e. combinations of attribute values) returns the profit achieved by selling each unit of the corresponding product. Note that the designer of the problem instance can use one of the attributes to denote the price of the product, so she can define the profit function in such a way that it directly returns the price (one of the attributes) minus the production cost (determined by the remaining attribute values of the product).[4]

[4]Note that we do not need to explicitly include any "production cost function" as a part of the problem

Definition 7. Assuming the notation in Definition 1, let us consider several alternative ways to define sets P^j (i.e. the set of products producer j can produce):

(I) *(Same as in Definition 1)* $P^j = A_1^j \times \ldots \times A_n^j$.

(II) *(The first e attribute values determine the remaining ones)* Let $1 \leq e \leq n$. For all producers $1 \leq j \leq d$ and all $e+1 \leq w \leq n$, let $attrib_w : A_1^j \times \ldots \times A_e^j \longrightarrow A_w^j$. Then we consider

$$P^j = \left\{ (p_1^j, \ldots, p_n^j) \;\middle|\; \begin{array}{l} \forall\, 1 \leq y \leq e : p_y^j \in A_y^j \wedge \\ \forall\, e+1 \leq w \leq n : p_w^j = attrib_w(p_1^j, \ldots, p_e^j) \end{array} \right\}$$

(III) *(A function checks the feasibility of the whole combination of attribute values)* For all producers $1 \leq j \leq d$, let $feasible_j : A_1^j \times \ldots \times A_n^j \longrightarrow \{\texttt{true}, \texttt{false}\}$ denote which combinations of attribute values are feasible for producer j. Then we consider $P^j = \{(p_1^j, \ldots, p_n^j) \mid feasible_j(p_1^j, \ldots, p_n^j)\}$.

Besides, let us consider two ways to define the respective goals of PDO, SPDG, and lSPDG:

(i) *(Number of achieved customers)* PDO, SPDG, and lSPDG are defined as in the previous definitions.

(ii) *(Profits collected by all sold units)* For all producers $1 \leq j \leq d$, let function $profit_j : P^j \longrightarrow \mathbb{N}$ return, for each possible product producer j may build, the profit the producer gets for each sold unit. Assuming the notation in Definition 1, let the *weighted profit* of the j-th producer, denoted as wpr_j, be

$$\sum \left\{ \frac{num_i}{|U_i(p_1, \ldots, p_d)|} \cdot profit_j(p_j) \;\middle|\; 1 \leq i \leq m \;\wedge\; j \in U_i(p_1, \ldots, p_d) \right\}$$

Then, the goal of PDO is finding the product $p_1 \in P_1$ maximizing wpr_1. Similarly, the goal of SPDG and lSPDG is redefined by replacing, in Definition 3, the numbers of gathered customers by profits (that is, the numbers stored in tuples $g_j = (g_j^1, \ldots, g_j^{t_p}) \in \mathbb{Q}^{t_p}$ are not the numbers wsc_i of customers gathered in each turn, but the profits wpr_i gained in each turn).

□

We identify the hardness of the resulting problems in the next result. As it is shown in its proof (see the appendix), all of these variants trivially inherit the hardness of the corresponding original problems, as they generalize them.

Theorem 4. Let us suppose that functions $feasible_j$, $attrib_j$, and $profit_j$ can be computed in polynomial time. For any combination of (I)-(II)-(III) and (i)-(ii),

instance: the profit function is defined in terms of all attribute values (including those affecting the production cost), so the values it returns can already be dependant on them.

(a) the resulting variant of PDO is Poly-APX-hard.

(b) the resulting variant of SPDG is EXPTIME-complete.

(c) the resulting variant of lSPDG is PSPACE-complete.

□

4. Heuristic algorithms and experiments

In this section we present our heuristic algorithms to face PDO, SPDG, and lSPDG, and we report our experimental results. PDO will be solved by using a *genetic algorithm* (GA) and a *particle swarm optimization* algorithm (PSO), which can be naturally applied to a wide variety of NP-hard optimization problems. Besides, SPDG and lSPDG will be solved by applying a *minimax algorithm* with α-β pruning, which is a method typically used to heuristically face EXPTIME-hard and PSPACE-hard game problems.[5] Note that SPDG and lSPDG are the same problems in conceptual terms, as lSPDG is a version of SPDG where the number of turns t_f is constrained. Thus, hereafter we will only mention SPDG (the more general problem) regardless of the chosen value of t_f in each case.

The instances of both problems to be solved in our experiments were constructed as follows. We considered 100 attributes with 4 possible values each to construct PDO instances, whereas 10 attributes with 2 possible values were used for SPDG. In both cases, customer profiles were created as follows. The valuation each customer profile gives to each attribute value is always between 0 and 5. Initially, four customer profiles were randomly created. Next, for each of them we created two additional *variants*. Each variant initially takes the same valuations as the profile it comes from, and next 33% of the valuations are given new random values. Finally, four additional customer profiles were added completely at random. These 16 customer profiles try to simulate the form of preferences in a feasible market: most of customer profiles concentrate around a few areas of preference similarity, and there are a few other isolated customer types. In addition, 10 producers were considered in PDO instances. For 90% of the attributes, all their attribute values were available to be selected by any producer. However, for the remaining attributes, only one attribute value was available for all the producers, whereas 33% of the producers could have more than one available attribute value (with 50% probability for each possible value of the attribute). Regarding SPDG, only 2 producers were used, and all values of all attributes were available for both.

In PDO, the products initially produced by each of the 9 opponent producers were created according to the following greedy algorithm. For each producer, four customer profiles were randomly selected. Next, for each attribute and for each of its attribute values, we added the valuations of this attribute value in these four customer profiles.

[5] Note that several players could be successful in the same PDG game, as the requirements regarding how many customers must be gathered might not be mutually exclusive (in practice, they would not be so if the number of required customers were small). The existence of a necessarily successful strategy in PDG in up to t_f turns (i.e. the goal of SPDG and lSPDG) can be fully checked by running a minimax algorithm up to depth t_f. Hence, running it for shorter depths is an appropriate *heuristic* for these problems.

We identified the attribute value whose addition was the highest one, and this attribute value was the value assigned to the corresponding attribute of the producer.

Both the GA and the PSO algorithm were applied to find a good product for producer 1 in this scenario. The genetic algorithm was set up as follows: for each attribute, the mutation probability was 0.01, a uniform crossover was applied, and a roulette wheel selection was used. In the case of PSO, the parameters were directly taken from [48], using 0.05529 as inertia weight, 0.112175 as cognitive parameter and 1.749212 as social parameter. In the GA an initial population of 20 possible solutions (i.e. possible products for producer 1) was created, including the solution of the previous greedy algorithm, and next it was executed for $1,000$ turns. Thus, 20,020 fitness evaluations are performed, including the 20 evaluations of the initialization. In the case of PSO, according to [48] the number of particles was set to 140 (the solution of the greedy algorithm is again included in the first step), while the number of iterations was set to 142 so that the number of fitness evaluations in PSO was the same as for the GA (20,020, including 143 evaluations during the initialization of particles), which enables a fair comparison between both methods. Note that, in PDO, producer 1 has (by definition) the advantage of designing its product *after* opponents have fixed theirs. In order to assess to what extent the performance of the GA and the PSO algorithm are thanks to the greedy algorithm alone (as it aids them to compose their initial populations), we also conducted the following experiment: after setting the products of the 9 opponents as we said before, next we ran *only* the greedy algorithm for producer 1, and we compared the results of such greedy algorithm with that of the GA and PSO methods.

We also implemented variations of the GA and PSO methods. For both algorithms, we considered two modifications to their basic schemes. In the first one, we run a clustering algorithm, in particular simple K-means, to our customer profiles (we directly use its implementation provided by WEKA [49]). This algorithm lets us classify our customer profiles into clusters with similar preferences, i.e. with similar attribute value valuations. Next, for the centroid of each cluster (i.e. for the "artificial customer profile" representing the actual profiles grouped by the cluster), a product is composed by picking, for each attribute, the value with highest valuation for the centroid. That is, for each cluster, we create a product by selecting the preferred attribute values of the artificial customer profile representing the cluster. All of these cluster-tailored products (which do not target all profiles, but only those in its specific cluster) are added to the initial population of both GA and PSO, and the remaining individuals of their initial populations are generated as before.[6] In our examples, we set the number of clusters to be generated by WEKA to 6.[7]

The second modification concerns the fitness function. Note that PDO creates a quite abrupt fitness function space: a tiny difference in a single attribute could be the difference between gaining all customers of some profile, or not gaining any of them,

[6]Note that we are not using clustering to control the population of the GA or PSO during their execution (as in e.g. [50]), but just to make their initial population contain some interesting greedy solutions for the PDO instance to be solved.

[7]This number is intentionally different to the number of customer profiles we used in the generation of each random PDO instance to generate profile variants around them.

as all customers within a profile pick the best product according to the preferences of the profile (recall that, according to the problem definition, they are equally divided between several products only in the case of a perfect preference tie; otherwise, it is an all or nothing choice). Note that evolutionary and swarm methods such as GAs and PSO typically work better if, in general, similar solutions have similar fitness. Thus a fitness function with many big discontinuities, like that denoting the actual goal of PDO, makes the problem difficult.[8] In order to help to solve this problem, we have introduced the following modification. In the first turns of the algorithm, the fitness function assumes that customers are not gained only by the winner product (as PDO does indeed), but that that they are divided among all products proportionally to their valuations. That is, if the total valuation of product A for a customer profile is 15, and the total valuation of product B for the same profile is 10, then the fitness function assumes that 60% of customers in the profile select A, and 40% pick B. Along the execution of the GA and PSO, the bias of the fitness function towards the best product gradually increases: some turns later, if we had the same total valuations of A and B (15 and 10), e.g. 80% of these customers could pick A and only 20% would pick B. Finally, in the last iterations, the fitness function works according to the actual preferences as defined in PDO: with the same total valuations (15 and 10), all customers in that profile would pick A and none would select B. This way, the first steps give some incentive to *getting close* to gain some customer profile, whereas the last steps only give incentive to truly winning, in the standard all-or-nothing binary PDO way. This promotes exploring nearly-valid solutions at first, while focusing only on the real ones at the end.

We generated 20 PDO instances as proposed, and we solved them with each of the mentioned algorithms 20 times. The average quality of solutions found by each method is shown in Table 1. Let us note that, in addition to the greedy algorithm, we compare eight methods: four GA variants and four PSO variants, as for each of them we consider using or not the modified fitness function ("-F" versions), and applying or not the clustering preprocessing ("-C" versions). The results of the greedy algorithm are around 2-5 times worse than the results of the other eight methods, so it is clearly not a competitive method. In order to statistically analyze the differences amongst all the remaining eight methods, we have used STATService [51] to perform a statistical test of the mean results given in Table 1. In particular, we have used Friedman aligned ranks test [52] to check whether the hypothesis that all methods behave similarly (the null hypothesis) holds or not. This test constructs a global ranking where the values of all methods and problems are ranked together. In Friedman aligned ranks test, for each problem the difference of each method with respect to the average value for all methods is considered, and next all values of all problems are ranked together. Let us consider $\alpha = 0.05$, a standard significance level. From results given in Table 1, we calculate that the p-value for aligned Friedman is 0.01, which allows us to reject the null hypothesis with a high level of significance (as the p-value is much lower than 0.05). So, the test concludes that the eight methods are not considered equivalent. Since the null hypothesis is rejected, we can apply a post-hoc analysis to compare one of the methods (taken as control method) with the other seven. Ranks assigned by this test to these methods

[8] This particular difficulty is indeed consistent with the Poly-APX-hardness of PDO.

Table 1: Results for PDO by the greedy algorithm (Greedy), the genetic algorithm (GA), and the PSO algorithm (PSO).

	Greedy	GA	GA-F	GA-C	GA-F-C	PSO	PSO-F	PSO-C	PSO-F-C
Instance 1	8,12	28,76	27,94	29,77	28,90	27,26	28,58	28,09	28,79
Instance 2	11,14	33,22	34,40	35,22	35,29	37,42	37,77	36,70	36,74
Instance 3	12,28	32,90	34,23	32,89	33,22	37,90	37,21	38,17	38,70
Instance 4	6,18	29,11	27,77	30,65	30,95	26,01	27,11	26,73	26,25
Instance 5	14,02	34,23	34,07	34,14	35,90	42,38	42,04	40,89	41,61
Instance 6	9,61	31,51	30,32	33,07	32,68	33,25	31,57	30,85	32,88
Instance 7	8,29	37,04	35,08	39,12	39,99	37,22	36,52	41,16	41,06
Instance 8	12,88	35,71	35,45	34,06	33,14	32,23	33,53	32,97	34,87
Instance 9	10,68	29,55	31,24	36,08	33,24	31,49	32,75	31,44	32,40
Instance 10	7,02	36,15	36,38	37,64	37,55	35,34	35,55	35,79	36,70
Instance 11	7,30	27,37	26,67	30,42	30,59	28,86	28,56	29,65	30,15
Instance 12	16,23	33,13	33,40	33,61	34,27	33,40	31,80	35,17	32,92
Instance 13	11,43	43,12	46,08	42,74	42,29	42,76	43,24	44,37	43,20
Instance 14	8,63	29,33	30,13	29,84	29,22	30,28	30,51	27,91	30,15
Instance 15	11,37	28,30	27,94	28,29	30,43	30,48	28,18	29,65	29,52
Instance 16	8,32	26,12	25,78	25,97	28,00	28,49	28,97	28,41	27,34
Instance 17	9,02	32,36	32,05	33,31	32,15	31,97	29,72	31,82	30,09
Instance 18	5,75	27,35	27,51	28,25	28,91	25,79	26,31	25,08	24,73
Instance 19	14,86	33,20	32,22	34,07	35,61	32,93	35,28	35,17	33,04
Instance 20	9,15	38,48	38,73	39,08	40,08	41,40	39,49	42,23	40,18

Table 2: Aligned Friedman test ranking for PDO.

GA-F-C	PSO-C	PSO-F-C	GA-C	PSO	PSO-F	GA	GA-F
64.9	68.8	70.6	71.4	79	81.2	103.2	104.8

can be seen in Table 2 (smaller ranks denote better performance). As it can be seen, the best results are obtained by the genetic algorithm using both the clustering preprocessing and the alternative fitness function. Thus, we use it as the control method to be compared with the rest of methods. By using Holm's procedure, we conclude that there exists a statistically relevant difference between that method and both GA methods not using clustering preprocessing, while there does not exist a statistically relevant difference amongst the rest of methods. That is, using the alternative fitness function does not make a statistically significant difference, but using clusters is relevant for the GA. Moreover, in the case of PSO, neither using clusters nor the alternative fitness function make a relevant difference.

We conclude that the clustering step is more useful for the GA than for PSO. A possible reason is that the tendency of all PSO particles to partially follow the best solution makes it quickly forget most of initial products designed to satisfy specific clusters, as only one of these initial products (the one gaining more customers) influences the particles. On the other hand, the individuals in GA are less biased towards

Table 3: Results for SPDG by the minimax algorithm.

	Inst 1	Inst 2	Inst 3	Inst 4	Inst 5	Inst 6
Depth prod 1	4	4	6	6	8	8
Depth prod 2	1	2	1	2	1	2
Mean	421481	417100	442558	443779	470391	446865
Std dev	70941	70533	74762	54049	65411	94761
Cust mean	777068	794479	803910	829461	842712	824862
% Cust	54.24%	52.50%	55.05%	53.50%	55.82%	54.17%
% Advantage	18.53%	10.53%	22.47%	15.05%	26.35%	18.2%

the best individual (actually, a good second best individual could be almost as influential as the best one), so the GA can keep exploring variants of the (very different) products initially designed for the different profile clusters for longer. This maintains the diversity for longer, and the GA benefits from it.

Next we present our experimental results for SPDG. In this case, the products initially produced by both producers were randomly created. Besides, $D = 1$, $t_p = 10$, and $t_f = 10$. The strategy used by producer 1 consisted in running, for each game turn where it moves, a minimax algorithm with α-β pruning with some depth $d_1 \in \{4,6,8\}$, and playing the resulting move. For producer 2, we used the same method, though only depths $d_2 \in \{1,2\}$ were considered. For both producers, the heuristic scores generated at the leaves of the tree explored by the minimax algorithm were just the numbers of customers collected during all the turns traversed in the branch of the tree where the leaf is.

For SPDG, we randomly generated 20 instances as explained before, and each of them was solved for different depths for both players (d_1 and d_2, for player 1 and player 2, respectively). In Table 3, the results of the experiments with six different instances are presented. Rows show the depth of the generated minimax tree for producers 1 and 2, the number of customers gathered by player 1 after the execution of the minimax algorithm (Mean), the standard deviation (Std dev), the mean of the total number of customers in the experiment (Cust mean), the percentage of customers gathered by producer 1 (% Cust), and finally the percentage of advantage achieved by producer 1 over producer 2 (% Advantage).

Our experimental results illustrate that the proposed heuristics to solve the problems under consideration in this paper are significantly better than other more straightforward greedy/myopic heuristics. In particular, the genetic and the particle swarm optimization algorithms presented to solve PDO (aided by the greedy algorithm) clearly outperform the greedy algorithm alone, and some modifications like introducing the clustering preprocessing may help to improve the results (we conclude that it does indeed in the case of the GA). In the case of SPDG, the comparison between the more sophisticated strategy and the more straightforward one, assigned to producer 1 and 2, respectively, yields more similar results (see Table 3). Both players use a minimax algorithm (with α-β pruning), so the only difference is the depth of the search tree to be explored by each one. In this case the aim of the comparison is to detect whether

the results are significatively improved when a deeper search space is considered. As it can be expected, using deeper searches improves the obtained results. In fact, the larger the difference between the search spaces is, the larger the difference between the obtained results is. Note that the greatest difference is between depth 8 and 1, where the deeper search obtains 26.35% more customers, whereas the shortest difference is between depth 4 and 2, where the deeper search only obtains 10.53% more customers.

5. Case study

In this section we present a case study where the proposed problems and methods are applied to a realistic environment. After introducing the basic heuristic implementations in the previous section, handling a realistic case in this section will allow us to highlight some difficulties related to dealing with real data, such as the loss of relevant information due to having the statistical data split into a limited number of customer profiles (where each one is considered homogeneous by definition), the difficulty to empirically adjust the weight of those preference factors that are not directly reflected in the market research study under consideration, and the difficulty of handling a high number of attribute values, which critically affects the efficiency of the minimax algorithm to solve SPDG (as a high branching dramatically reduces the depth of the game trees we can explore in practice). The real environment considered in this section will force us to tackle issues not appearing when dealing with the artificial examples faced in the previous section.

Note that companies typically do not disclose their marketing research data to the public audience —at least, with enough detail to constitute a proper input to test our problems, as we need customer preferences to be stratified into a non-negligible number of population groups. However, there is a kind of statistics that actually provides suitable data to construct a case study without the necessity to access private company data. Statistics on *political preferences* are usually stratified into a significant number of groups and, more importantly, they can often be publicly accessed, as they are typically conducted by public institutions in many countries. In terms of the problems considered in this paper, dealing with politics is basically the same as with products: customers (voters) select their favorite product (political party) to buy (vote) by comparing the features of all available products (the policies of each party upon each significant issue, such as taxes, public investment, migration, security, territorial issues, etc), including those features which can only be present in some products (parties can attract or repel some voters just due to the particular history of the party), and next they pick their best choice. In a political context like this, PDO is the problem of designing the platform of a party from scratch in such a way that it maximizes its number of voters at a specific time (e.g. the election day), and SPDG is the problem of reaching a good average number of party supporters along time, taking into account that the platforms of parties can change along time (at some pace).

In our case study we will consider, in particular, the politics in Spain, and the data for our experiments will be collected from those publicly released by CIS (*Centro de Investigaciones Sociológicas*, Center of Sociological Research), a public institution which conducts polls in the country. From its website (http://www.cis.es), we retrieved statistical data regarding the opinion of Spaniards on many political issues,

including migration policies, public health care, level of self-government of the regions, policies to favor companies, policies towards families, etc. Due to the abundance of data with enough stratification existing in polls conducted in 2012, all data used in our experiments concerned that year. 69 questions, with a number of possible replies ranging between 2 and 11 each, were collected. In terms of the problem inputs to be solved by `PDO`, these questions are represented by 69 attributes with 2-11 possible values each. Consequently, in our model a *political platform* (the product each party *sells*) is a combination of stances towards each of these questions. These CIS polls were stratified according to several criteria (age, gender, income, education, region, etc), but not all criteria of stratification were included in all questions, so we decided to adopt in our data two stratification criteria that were applied indeed to all of these questions: age and level of education. Each of these two criteria is split into several categories (six age ranges and six levels of education), yielding a stratification with 36 categories. Since we know the replies (preferences) of the population within each of these 36 categories to each question, we could naturally transform each of these 36 categories into 36 customer profiles, each one representing the voters in that category (i.e. the voters of that combination of age and level of education). However, additional factors must be considered.

In order to construct the actual input model for our problems from these data, preferences collected from these polls must be converted into the numeric valuations each customer profile (group of population) gives to each attribute value (question reply) for each attribute (question). For each combination of age and level of education, we could just create a customer profile representing the preferences of the population in that specific group. Note that, in our model, customer profiles are considered homogeneous by definition: all members of a customer profile are represented by the same set of preferences (valuations of attribute values). There are several ways to extract the value valuations of each customer profile from the actual proportion of members within that group who preferred each choice in the polls. For instance, we could give the highest valuation to the statistical mode of the registered replies, or we could give it to its numerical average, after assigning a number to each reply. In either way, the resulting model would under-represent or just completely miss minority preferences. However, these minority preferences do affect markets in general, and electoral results in particular. For instance, in the considered CIS polls, there is no combination of age and level of education for which the preferred territorial organization is one that lets regions split and become independent states —although some voters from each combination of age and level of education, mainly concentrated on some geographical regions, *do* prefer that particular choice, yielding a non-negligible total number of voters preferring that choice if considered all together. A model with just 36 customer profiles would under-represent that choice, as *none* of the customer profiles would prefer this choice (so, that choice would be preferred by nobody in the resulting model). In order to cope with this problem, we split each of these 36 combinations into *several* customer profiles. Each actual customer profile is created by picking its preferences randomly, in such a way that the probability of each choice is proportional to the amount of poll participants from that combination of age and level of education who selected that choice. This way, if e.g. 10% of the poll respondents within some combination of age and level of education preferred letting regions become independent, then we expect that 10% of

the actual customer profiles being constructed from that combination will prefer that choice indeed. In our case study, from each combination of age and level of education, we followed that method to create a customer profile for every 20 poll respondents or fraction.[9]

After the preferred choices of each (final) customer profile are probabilistically selected in this way, the numerical valuations this customer profile gives to the available attribute values (i.e. question replies) are assigned as follows. For questions where replies represent different levels of support to a given idea (e.g. "I totally agree", "I tend to agree", "I neither agree nor disagree", etc), the choice selected for that particular customer profile is given the highest numeric valuation among all possible replies for the same question. The numeric valuations of the remaining replies are assigned by taking the valuation of the preferred choice and reducing it proportionally to the distance from the reply to that preferred choice. For instance, if "I tend to agree" is given a valuation of 5, then "I totally agree" and "I neither agree nor disagree" are given 3, "I tend to disagree" is given 1, and so on. Regarding questions where available replies do not represent different grades of support to a notion but incomparable choices, a positive valuation is assigned to the selected choice (e.g. 5), and all remaining choices are given a 0 valuation.

Our model also needs to represent the products sold by other producers, that is, the political platforms of all other parties the voters can vote. In `PDO`, competitors must be given some (fix) products to sell. For all 69 questions in our set of questions, CIS also provides a stratification of the question in terms of the party voted by the poll respondent in the last election, for the five most voted parties in Spain as of 2012 (PP, PSOE, IU/IC, UPyD and CiU). We used these data to automatically select the attribute values of each of these five parties as follows: For each attribute (question), we picked the reply (attribute value) corresponding to the statistical mode among all poll respondents who declared voting that party, and we assigned this value to the corresponding attribute of the party. Thus, we are implicitly assuming that the political platform of each party exactly corresponds to the combination of majority choices of its voters in the last election. Note that, alternatively, the political platforms could be defined by (manually) analyzing the actual platforms of the corresponding parties, and next converting them into attribute values. However, extracting attribute values by the majority method enables a more systematic, automatized, and replicable experiment.

Not all 69 questions have the same capability to affect the voting tendency of each potential voter, so different weights could be assigned to each question. In practice, we could create this effect just by making valuations of important questions be within wider ranges (e.g. valuations of important questions could be within 0-10, whereas less important questions could be within 0-5). Also, note that the preferences of voters towards available parties do not only depend on their current political platforms, but

[9]These final customer profiles miss any preference *correlations* existing inside each combination of age and level of education. Let us suppose that, in some combination, the members preferring reply A on question 1 usually prefer B on question 2. The set of customer profiles extracted from that combination of age and level of education will not show that correlation. Unfortunately, this problem is inherent to any way to aggregate information into a limited number of representative elements. In our case, our input data (CIS polls) already miss these correlations, so customer profiles extracted from them do so too.

also on their historical decisions in the past, the personal image of their candidates, the perception that voting a minority party might be useless because it will not affect future laws, etc.[10] This could be represented in our model by making each party have some unchangeable attribute value whose valuation quantifies the positive or negative perception of each customer profile towards the party (recall that our model lets setting attribute values that can accessed only by some producers; hence, if some party can select, for some attribute, only *its* own value, then the party is forced to have its particular value). Unfortunately, the data publicly released by CIS does not allow us to automatically assign the weight of each question on the selection of our favorite party, neither it allows us to quantify the particular image of each party beyond that directly depending on our opinion towards the political stances of its platform. In order to make this case study be as systematic and replicable as possible, we decided not to use manually-estimated question weights or party public images. Hence, all questions are considered equally important in our model, and all parties have the same public image.[11]

We apply the greedy algorithm, GA and PSO to solve the PDO instance representing this case study in the same way as we did to solve the instances considered in the previous section, but considering only the configurations of GA and PSO that obtained the best results in our previous experiments. That is, the genetic algorithm is executed for $1,000$ turns, the population is set to 20, and it uses both the clustering preprocessing and the alternative dynamic fitness function. The mutation probability is set to 0.01, and a uniform crossover is applied. Analogously, the PSO method uses again the parameters given in [48], and it applies clustering preprocessing. The average percentage of voters obtained by the GA after running 20 executions of the algorithm was 40.72%, PSO obtained an average of 37.13% voters, and the greedy algorithm obtained only 21.84%. That is, both GA and PSO obtain approximately two times the number of voters obtained by the greedy algorithm.

Regarding SPDG, the application of a minimax algorithm to the input instance of this case study faces a particular challenge: the number of attributes (69) and the number of attribute values (2-11 for each question) are so high that the number of children at each node in the minimax tree would be too big. Note that the size of this tree is proportional to the number of children at each node (i.e. the total number of attribute values) raised to the power of the depth of the tree, so trees of any non-negligible depth are unfeasible in this case. In order to cope with this problem, we just *partially* develop the minimax tree as follows. Rather than exploring all the attribute values that the corresponding player (party) could change at its current game turn, only a few attribute values, of a few attributes, will be explored (i.e. only their children subtrees will actually be developed). The actual attribute values to be explored will be chosen probabilistically,

[10] Similarly, if we were selling commercial products rather than political parties, the image of the producer brand might be as important as the product features.

[11] There are indirect methods we could use to infer the weights of questions and the party images from the data publicly released by CIS, such as e.g. studying the correlations between the replies given by poll respondents to each question and the parties they vote. However, since these methods would necessarily be speculative to some extent, we prefer to leave these issues to future research, and keep this case study in the scope of straightforward CIS data, avoiding the necessity to interpret data as much as possible.

Table 4: Case study results for SPDG by the minimax algorithm with 2 players.

Depths	2 vs 6	4 vs 6	6 vs 6	6 vs 4	6 vs 2
Mean	28.43%	30.76%	32.89%	33.39%	34.26%
% improvement	0%	8.2%	15.7%	17.4%	20.5%

in such a way that values of attributes with a higher impact on the voting tendencies will be picked more often. More specifically, those attribute values with more valuation *variance* among all customer profiles (i.e., those with a higher capability to polarize the preferences of voters) will be picked with higher probability, as changing them has a potentially higher effect on the state of a PDG game.[12]

Table 4 summarizes the results obtained in our experiments with SPDG. We used the aforementioned modification of the minimax algorithm to restrict the children of each node to 30. In each case, the game was played for 10 steps. We consider that the five political parties of our case study are present and can be voted by voters, but we assume that only two of them (those with more supporters according to the CIS polls, PP and PSOE) modify their political platforms in the game. Table 4 compares the results obtained by using different search depths for these two opponents, and it shows the average number of supporters along time of the first political party (PP). As it can be expected, bigger differences between the search depth of the first player and the search depth of the second one allow to significatively improve the percentage of supporters of the first player (the second row of Table 4 shows the voters increase of the first political party compared with its worse case of the table, the 2 vs 6 case).

Let us consider the previous 2-player experiments where the minimax was run up to depth 6. These minimax executions explored game evolutions where each player moved 3 times. Note that this experiment would not be feasible if all five parties played the game (i.e. if all could change their platforms during the game) and 30 possible moves of each party were explored at each turn as before. In this case, the number of minimax tree branches added just to make all players move once would be 30^5. Hence, the number of possible final configurations of the game after all players move three times would be $(30^5)^3$, which is roughly $1.5 \cdot 10^{22}$. The corresponding minimax tree could not be developed in a reasonable time, even if we are very optimistic with the effects of pruning.

In order to make all five political parties play the game in an experiment with SPDG, but simultaneously let the minimax reach a non-negligible number of anticipated turns

[12]Regarding our previous discussion about assigning weights to questions, note that a high variance among the replies we can give to some question does not imply that the question should be given a higher weight than the others. A question could partition voters into polarized groups, and yet not be very important to determine the voting tendencies (e.g. "who is your favorite singer?"). Hence, the variance in replies is not enough to construct a (well-motivated) set of different question weights. However, once we assume that all questions are given the same weight, the replies with higher valuation variance among all customer profiles have more probability to significantly affect the voting results if they change. Thus, a PDG player should pay more attention to them.

Table 5: Case study results for SPDG by the minimax algorithm with 5 players.

Depths	2 vs 6	2 vs 4	4 vs 4	4 vs 2	6 vs 2
Mean	19.32%	19.88%	22.02%	23.45%	23.83%
% improvement	0%	2.9%	13.9%	21.4%	23.3%

in a reasonable time, we perform the following modification of the standard minimax scheme. If we are using the minimax to decide the best move of party 1, then parties 2, 3, 4 and 5 are grouped all together as *"the rest of parties"*, and the levels of party 1 and those of "the rest or parties" strictly alternate in the minimax tree. In nodes where party 1 moves, 20 *interesting* moves (i.e. moves with high valuation variance, as before) are selected and explored by the minimax. On the contrary, in the nodes of "the rest of parties", 20 interesting moves for party 2 are picked, then we choose another 20 interesting moves for party 3, and so on for parties 4 and 5. Next, 20 random combinations of one interesting move for each of these four parties are composed. Then, these 20 combinations are explored as the possible moves of "the rest of parties" in the minimax node where it moves.

Despite the way the minimax groups parties together, each party within "the rest of parties" has its own voters, and all voters choose among all available five parties after each actual party moves (in fact, the successive playing order of all five parties is respected). The only difference is that the minimax algorithm designs the strategies of parties 2, 3, 4 and 5 all together, as if they were on the same side. Note that this strategy is implicitly assuming that parties 2, 3, 4 and 5 somehow collude to play against party 1. In real games, parties 2, 3, 4 and 5 could collude or not, so we are assuming the worst case for party 1. Note that the goal of SPDG is looking for a valid strategy regardless of the strategies of the other players. Hence, considering the worst scenario for the player whose move is decided is a sensible choice indeed.

After party 1 plays the move selected by that minimax execution, it is the turn of party 2, so its actual move has to be decided in the experiment. Now the minimax groups parties differently: on one side we have party 2, and on the other side we have parties 1, 3, 4, and 5 together. We do similarly when the actual moves of parties 3, 4, and 5 are decided later, in their respective turns of the game of the experiment. This way, each minimax execution gives more importance to the player whose move is being decided (in particular, its moves are more exhaustively explored) than to the rest of players, whose moves are aggregated all together by the minimax, as if there was a single opponent in strategic terms.

The experimental results of this heuristic method to handle five players in SPDG can be seen in Table 5. In each experiment, the game was played for 6 full rounds, i.e. until all five players moved 6 times each, so 30 minimax executions were required in each experiment (note that, in our previous experiment, each experiment involved 10 rounds with 2 players, that is, 20 minimax executions). As in the previous case, we tested five different depth configurations, ranging from a big disadvantage for *our* player (depth 2 vs depth 6) to a big advantage for *our* player (depth 6 vs depth 2). As

it can be expected, the average results show that deeper searches allow to improve the number of supporters of our party.

6. Conclusions

In this paper we have formally defined PDO, SPDG, and lSPDG, three problems which try to encapsulate the main goals of any marketing department: selecting a product that achieves a high number of customers immediately (in PDO), and following a sustainable product design strategy that provides a good rate of customers in the long or short term (in SPDG and lSPDG, respectively). We have proved that these problems are Poly-APX-complete, EXPTIME-complete, and PSPACE-complete, respectively. Their intractability motivates solving them by means of heuristic suboptimal methods rather than by optimal algorithms. In particular, the Poly-APX-completeness of the simplest problem (PDO) motivates solving it by using heuristics not focusing on guaranteeing some performance ratio in the worst case.

We develop experiments dealing with both artificially-designed inputs and inputs constructed from real data taken from the politics domain. Our experiments on artificially-designed inputs show that some simple heuristic algorithms (in particular, GA and PSO for PDO, and a minimax algorithm reaching some limited depth for SPDG) provide a better performance than more straightforward algorithms (respectively, a simple greedy algorithm, and a minimax algorithm reaching an even smaller depth). This was particularly significant for PDO, where both GA and PSO (supported by a greedy algorithm) usually reached 2-5 times more customers than the greedy strategy alone. A lower advantage of the more sophisticated strategy was observed in SPDG. For instance, producers executing the minimax algorithm up to depth 6-8 achieved only 15-18% more customers than opponent producers executing the minimax algorithm up to depth 2.

Our second experiment, constituting our case study, forced us to identify the main problems appearing when dealing with real data: (a) customer profiles could need to be (probabilistically) split into smaller profiles in order not to lose minority preferences; (b) real market research data could not explicitly provide enough information to assign weights to questions (attributes), as well as to quantify the public image of each producer; and (c) if the number of available attributes is high, then minimax algorithms should prioritize the most important attributes and attribute values, in order to enable executions reaching reasonable depths. Our experiments explicitly tackled issue (a) and (c). Concerning issue (b), equal attribute weights were used in our case study, and no producer images were introduced, in order to make our experiments stick to available raw data as much as possible, so we left the interpretation of available data in terms of weights and images to future work.

In addition to developing methods to extract attribute weights and producer images, our future work will consist in developing more sophisticated and tailored heuristic algorithms to deal with the problems presented here, as well as tackling some of the problem generalizations previously commented at the beginning of Section 3: letting problem instance designers denote more expressive customer preferences; letting available attribute values change along time; and letting producers simultaneously sell more than one product.

Acknowledgements

This work has been partially supported by projects TIN2012-39391-C04-04, TIN2015-67522-C3-3-R, and S2013/ICE-2731. We would like to thank Sonia Mardomingo and Manuel Moraga for their comments on the topic of this paper, Julia Rodríguez for her support, and the anonymous referees for their interesting suggestions.

References

[1] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, W. S. Stornetta, Spawn: A distributed computational economy, Software Engineering, IEEE Transactions on 18 (2) (1992) 103–117.

[2] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, A. Yu, Mariposa: a wide-area distributed database system, The VLDB Journal 5 (1) (1996) 48–63.

[3] R. Buyya, D. Abramson, S. Venugopal, The grid economy, Proceedings of the IEEE 93 (3) (2005) 698–714.

[4] X. León, T. A. Trinh, L. Navarro, Using economic regulation to prevent resource congestion in large-scale shared infrastructures, Future Generation Computer Systems 26 (4) (2010) 599–607.

[5] M. S. Miller, K. E. Drexler, Markets and computation: Agoric open systems, The ecology of computation 1.

[6] T. Sandholm, Algorithm for optimal winner determination in combinatorial auctions, Artificial intelligence 135 (1) (2002) 1–54.

[7] S. De Vries, R. V. Vohra, Combinatorial auctions: A survey, INFORMS Journal on computing 15 (3) (2003) 284–309.

[8] F. Luna, B. Stefansson, Economic Simulations in Swarm: Agent-Based Modelling and Object Oriented Programming, Vol. 14 of Advances in Computational Economics, Springer, 2000.

[9] P. Z. Maymin, Markets are efficient if and only if P=NP, Algorithmic Finance 1 (1) (2011) 1–11.

[10] M. Shaw, R. Blanning, T. Strader, A. Whinston, Handbook on electronic commerce, Springer, 2000.

[11] M. Núñez, I. Rodríguez, PAMR: A process algebra for the management of resources in concurrent systems, in: 21st IFIP WG 6.1 Int. Conf. on Formal Techniques for Networked and Distributed Systems, FORTE'01, Kluwer Academic Publishers, 2001, pp. 169–185.

[12] M. Núñez, I. Rodríguez, F. Rubio, Formal specification of multi-agent e-barter systems, Science of Computer Programming 57 (2) (2005) 187–216.

[13] P. Kotler, K. L. Keller, M. Brady, M. Goodman, T. Hansen, Marketing Management, Pearson, 2012.

[14] L. Chen, P. Pu, Survey of preference elicitation methods, Tech. rep., Swiss Federal Institute of Technology in Lausanne (EPFL) (2004).

[15] R. Aydoğan, P. Yolum, Learning opponents preferences for effective negotiation: an approach based on concept learning, Autonomous Agents and Multi-Agent Systems 24 (1) (2012) 104–140.

[16] M. Mitchell, An introduction to genetic algorithms, MIT press, 1998.

[17] J. Kennedy, Particle swarm optimization, in: Encyclopedia of machine learning, Springer, 2011, pp. 760–766.

[18] M. Wedel, W. A. Kamakura, Market segmentation: Conceptual and methodological foundations, Vol. 8 of International Series in Quantitative Marketing, Springer, 2000.

[19] C. Papadimitriou, M. Schapira, Y. Singer, On the hardness of being truthful, in: Foundations of Computer Science, 2008. FOCS'08, IEEE, 2008, pp. 250–259.

[20] D. Buchfuhrer, M. Schapira, Y. Singer, Computation and incentives in combinatorial public projects, in: Proceedings of the 11th ACM conference on Electronic commerce, ACM, 2010, pp. 33–42.

[21] R. Z. Farahani, M. Abedian, S. Sharahi, Dynamic facility location problem, Springer, 2009.

[22] T. Drezner, Z. Drezner, S. Salhi, Solving the multiple competitive facilities location problem, European Journal of Operational Research 142 (2002) 138–151.

[23] D. Saban, N. Stier-Moses, The competitive facility location problem in a duopoly: Connections to the 1-median problem, in: Proceedings of the 8th International Conference on Internet and Network Economics, WINE'12, Springer, 2012, pp. 539–545.

[24] R. Aboolian, O. Berman, D. Krass, Competitive facility location and design problem, European Journal of Operational Research 182 (2007) 40–62.

[25] N. Betzler, R. Bredereck, J. Chen, R. Niedermeier, Studies in computational aspects of voting, in: The Multivariate Algorithmic Revolution and Beyond, Springer, 2012, pp. 318–363.

[26] J. R. Chamberlin, P. N. Courant, Representative deliberations and representative decisions: Proportional representation and the borda rule, American Political Science Review 77 (03) (1983) 718–733.

[27] A. D. Procaccia, J. S. Rosenschein, A. Zohar, On the complexity of achieving proportional representation, Social Choice and Welfare 30 (3) (2008) 353–362.

[28] T. Sandholm, Algorithm for optimal winner determination in combinatorial auctions, Artificial intelligence 135 (1) (2002) 1–54.

[29] E. McCarthy, Basic marketing: A managerial approach, RD Irwin, 1978.

[30] B. Lauterborn, New marketing litany: four Ps passe: C-words take over, Advertising age 61 (41) (1990) 26.

[31] W. Van Waterschoot, C. Van den Bulte, The 4P classification of the marketing mix revisited, The Journal of Marketing (1992) 83–93.

[32] E. Constantinides, The marketing mix revisited: towards the 21st century marketing, Journal of marketing management 22 (3-4) (2006) 407–438.

[33] C. Goi, A review of marketing mix: 4Ps or more?, International Journal of Marketing Studies 1 (1) (2009) 2.

[34] M. Hallgren, J. Olhager, Quantification in manufacturing strategy: A methodology and illustration, International Journal of Production Economics 104 (1) (2006) 113–124.

[35] E. Goldratt, Theory of constraints, North River Croton-on-Hudson, 1990.

[36] R. Souren, H. Ahn, C. Schmitz, Optimal product mix decisions based on the theory of constraints? exposing rarely emphasized premises of throughput accounting, International Journal of Production Research 43 (2) (2005) 361–374.

[37] F. Persson, J. Olhager, Performance simulation of supply chain designs, International Journal of Production Economics 77 (3) (2002) 231–245.

[38] W. Shen, Distributed manufacturing scheduling using intelligent agents, Intelligent Systems, IEEE 17 (1) (2002) 88–94.

[39] P. Golden, M. Dollinger, Cooperative alliances and competitive strategies in small manufacturing firms, Entrepreneurship: Theory and Practice 17 (4) (1993) 43–57.

[40] Z. Huang, S. Li, Co-op advertising models in manufacturer–retailer supply chains: A game theory approach, European Journal of Operational Research 135 (3) (2001) 527–544.

[41] W. Yunzhi, H. Yaoguang, J. Yanhua, Z. Ruijun, A game theoretic approach to product-mix resource allocation, in: IEEE Conference on Industrial Electronics and Applications (ICIEA'10), IEEE, 2010, pp. 2142–2147.

[42] F. Ricci, L. Rokach, B. Shapira, Recommender Systems Handbook, Springer, 2011, Ch. Introduction to Recommender Systems Handbook, pp. 1–35.

[43] A. Sethi, S. Sethi, Flexibility in manufacturing: a survey, International journal of flexible manufacturing systems 2 (4) (1990) 289–328.

[44] S. Andreou, A capital budgeting model for product-mix flexibility, Journal of Manufacturing and Operations Management 3 (1) (1990) 5–23.

[45] A. Triantis, J. Hodder, Valuing flexibility as a complex option, The Journal of Finance 45 (2) (1990) 549–565.

[46] J. Bengtsson, J. Olhager, Valuation of product-mix flexibility using real options, International Journal of Production Economics 78 (1) (2002) 13–28.

[47] J. Hartmanis, R. Stearns, On the computational complexity of algorithms, Transactions of the American Mathematical Society 117 (1965) 285306.

[48] M. Pedersen, A. Chipperfield, Simplifying particle swarm optimization, Applied Soft Computing 10 (2) (2010) 618–628.

[49] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. Witten, The WEKA data mining software: an update, ACM SIGKDD explorations newsletter 11 (1) (2009) 10–18.

[50] S. Liu, M. Mernik, B. Bryant, A clustering entropy-driven approach for exploring and exploiting noisy functions, in: 2007 ACM Symposium on Applied Computing (SAC), ACM, 2007, pp. 738–742.

[51] J. Parejo, J. García, A. Ruiz-Cortés, J. Riquelme, STATService: Herramienta de análisis estadístico como soporte para la investigación con metaheurísticas, in: Actas del VIII Congreso Expañol sobre Metaheurísticas, Algoritmos Evolutivos y Bio-inspirados, 2012.

[52] J. Hodges, E. Lehmann, Ranks methods for combination of independent experiments in analysis of variance, Annals of Mathematical Statistics 33 (1962) 482–497.

[53] C. Bazgan, B. Escoffier, V. T. Paschos, Completeness in standard and differential approximation classes: Poly-(D)APX- and (D)PTAS-completeness, Theoretical Computer Science 339 (2-3) (2005) 272–292.

[54] P. Crescenzi, A short guide to approximation preserving reductions, in: Proceedings of the 12th Annual IEEE Conference on Computational Complexity, CCC '97, IEEE Computer Society, 1997, pp. 262–273.

[55] L. Stockmeyer, A. Chandra, Provably difficult combinatorial games, SIAM J. Comput 8 (2) (1979) 151–174.

[56] M. Sipser, Introduction to the theory of computation, Thomson, 2006.

Appendix: Proofs of results

Proof of Theorem 1

Let us prove (ii). First, we see $\texttt{PDO} \in \texttt{Poly-APX}$. Note that any product $p_1 \in P^1$ can be represented in polynomial space with respect to the size of the full instance of

PDO. Besides, given a product $p_1 \in P^1$, we can calculate wsc_1 (the number of customers producer 1 gets by offering p_1) in polynomial time by counting the number of customers which prefer p_1 over any other product $p_2, \ldots, p_d$ or tie with them. This just requires calculating, for each customer profile $1 \leq i \leq m$, the total valuation of each product $p_1, p_2, \ldots, p_d$ for that customer profile (this requires adding the valuation of that product due to each attribute $A_1, \ldots, A_n$), and next comparing the total valuation of p_1 with the total valuation of all other given products $p_2, \ldots, p_d$. This can be easily done in polynomial time, so PDO $\in$ NPO, i.e. PDO is an NP optimization problem. Let S be the following polynomial-time strategy to find an approximate solution to any given instance X of PDO. For each customer profile $1 \leq i \leq m$, we define product $q_i = (q_1^i, \ldots, q_n^i) \in P^1$ where, for all $1 \leq h \leq n$, we have $q_h^i = argmax_{v \in A_h^1}\{val_{i,v}\}$ (that is, each product q_i is the best product producer 1 can sell to attract, specifically, customer profile i). Let the strategy S return the product p_1 in $\{q_1, \ldots, q_m\}$ which provides more customers for producer 1. Clearly, this strategy takes polynomial time. For any instance X of PDO and for any solution Y of PDO for the instance X, let $R_{\texttt{PDO}}(X,Y)$ be the ratio between the number of customers achieved by the optimal solution for X and the number of customers achieved by Y. Let us show that $R_{\texttt{PDO}}(X,S(X)) \in \mathcal{O}(N^\eta)$ for some $\eta \geq 0$, where N is the size of the instance X of PDO, which implies PDO $\in$ Poly-APX. In particular, let us show that $R_{\texttt{PDO}}(X,S(X))$ is always lower than or equal to N. According to the proposed strategy, let $p_1 = q_i$ for some $1 \leq i \leq m$. Let us suppose that this product p_1 achieves k customers from customer profile i. Note that the optimal product for producer 1 (i.e. the one achieving more customers for producer 1 from all customer profiles) cannot get more than $k \cdot m$ customers of all customer profiles together: if it did, then that product would get more than k customers from some customer profile, but this is not possible by the construction of p_1. Since the proposed strategy gets at least k customers, $R(X,S(X))$ is always lower than or equal to m. Since $N \geq m$, we have $R(X,S(X)) \leq m \leq N$. Therefore, PDO $\in$ Poly-APX.

Next we show PDO $\in$ Poly-APX-hard. Let us consider the *Maximum Independent Set* problem (MIS). This optimization problem is defined as follows: Given a graph $G = (\mathcal{V}, E)$ where $\mathcal{V}$ is the set of vertexes and E is the set of edges, find the set $\mathcal{V}' \subseteq \mathcal{V}$ with highest cardinality such that no pair of vertexes from $\mathcal{V}'$ are connected by an edge in E. Since MIS $\in$ Poly-APX-complete (see [53]), an *AP*-reduction (see e.g. [54]) from MIS to PDO will prove PDO $\in$ Poly-APX-hard. Our transformation of each MIS instance into a PDO instance will be based on the following ideas: (a) graph nodes are represented by customer types; (b) graph edges are represented by attributes whose two only possible values represent selecting each endpoint of the edge; and (c) each customer type (i.e. node) prefers our product over the product of our competitor iff, in our product, *all* attributes representing edges connected to the corresponding node have the values (i.e. edge endpoints) representing the node of *that* customer type. Hence, our product will not be able to simultaneously get the customers of two customer types represented by *adjacent* nodes. In order to make our competitor get all of these customers unless our product meets the proposed requirement, its product will achieve some high valuation which can be exceeded by our product only if our product meets the requirement. The product of our competitor will gain that high valuation

solely from its mandatory and exclusive value for the first attribute.

Formally, let function f transform MIS instances into PDO instances as follows. Let $G = (\mathscr{V}, E)$, with $\mathscr{V} = \{\mathfrak{v}_1, \ldots, \mathfrak{v}_n\}$ and $E = \{e_1, \ldots, e_m\}$, be an instance of MIS. Given $e_j \in E$, we will assume that $e_j^1 \in \mathscr{V}$ and $e_j^2 \in \mathscr{V}$ are the pair of vertexes connected by e_j. Instance G of MIS is transformed by f into the following instance of PDO:

- The attributes are $A_1, \ldots, A_m$. For all $1 \leq j \leq m$ we have $A_j = \{e^1_{j_{A_j}}, e^2_{j_{A_j}}, u_{A_j}\}$.
- There are two producers 1 and 2 whose sets of available values of attributes are $A_1^1, \ldots, A_m^1$ and $A_1^2, \ldots, A_m^2$, respectively. For all $1 \leq j \leq m$ we consider $A_j^1 = \{e^1_{j_{A_j}}, e^2_{j_{A_j}}\}$ and $A_j^2 = \{u_{A_j}\}$ (thus, $p_2 = (u_{A_1}, \ldots, u_{A_m})$).
- There are n customer profiles. For all $1 \leq i \leq n$, the value $val_{i,v}$ given by customer profile i to attribute value v is defined as follows:
  - For all $1 \leq j \leq m$ and $\mathfrak{v} \in \{e^1_{j_{A_j}}, e^2_{j_{A_j}}\}$, if $\mathfrak{v} = \mathfrak{v}_i$ then $val_{i,\mathfrak{v}_{A_j}} = 2$ else $val_{i,\mathfrak{v}_{A_j}} = 0$.
  - $val_{i,u_{A_1}} = 2 \cdot d - 1$ where d is the degree of vertex $\mathfrak{v}_i$ in graph G and, for all $2 \leq j \leq m$, $val_{i,u_{A_j}} = 0$.

  Besides, for all $1 \leq i \leq n$ we have $num_i = 1$.

It is easy to see that the size of the generated PDO instance is polynomial w.r.t. the size of the original MIS instance. Moreover, f is computed in polynomial time.

Let function g transform each solution for that PDO instance (i.e. a product $p_1 = (p_1^1, \ldots, p_m^1)$) into a solution for the original MIS instance (i.e. a set of vertexes $\mathscr{V}'$) as follows: the set $\mathscr{V}'$ consists of all vertexes $\mathfrak{v} \in \mathscr{V}$ such that, for all $e_j \in E$ connected to $\mathfrak{v}$, we have $p_j^1 = \mathfrak{v}_{A_j}$. That is, we include in $\mathscr{V}'$ all vertexes $\mathfrak{v}$ such that the values of all attributes representing edges touching $\mathfrak{v}$ are those corresponding to picking the endpoint $\mathfrak{v}$ (instead of the opposite endpoint). Let us show that p_1 gets k customers iff $\mathscr{V}' = g(p_1)$ is an independent set of size k:

$\Longrightarrow$: If p_1 gets k customers then p_1 beats p_2 for k customer profiles (note that there is a single customer in each customer profile). Beating p_2 for some customer profile i requires that, for all attributes representing edges connected to the vertex $\mathfrak{v}_i$ represented by the customer profile i, the selected value is the one representing picking $\mathfrak{v}_i$ (instead of the vertex at the opposite endpoint of the edge). This is because 2 units of valuation are gathered by each of these attribute values, so the only way to surpass the $2 \cdot d - 1$ units of valuation reached by producer 2 is getting all of these d attribute values. Since a single value can be picked for each attribute (i.e. a single endpoint can be selected for each edge), it is not possible to simultaneously beat p_2 for any two customer profiles representing the vertexes connected to same edge. Hence $\mathscr{V}'$ is an independent set. Besides, it has k vertexes, one for each customer profile achieved by p_1.

$\Longleftarrow$: Let us suppose that $\mathscr{V}'$ is an independent set of size k. By the construction of $\mathscr{V}'$, all vertexes $\mathfrak{v}$ of $\mathscr{V}'$ are such that, for all attributes representing edges

connected to $\mathfrak{v}$, the selected attribute value is the one representing picking vertex $\mathfrak{v}$. This means that the (single) customer representing vertex $\mathfrak{v}$ prefers producer 1 (which reaches a $2 \cdot d$ valuation) over producer 2 (which reaches $2 \cdot d - 1$). Thus, producer 1 gets k customers.

Hence, p_1 achieves k customers iff $\mathscr{V}' = g(p_1)$ is an independent set of size k. Let X be an instance of MIS. Note that (a) there exists an independent set of size k for X iff (b) there exists a product for producer 1 getting k customers for the PDO instance $f(X)$. In order to show the implication from (b) to (a), we just have to apply function g to get one solution from the other. Regarding the implication from (a) to (b), we can transform the solution for X into a solution for $f(X)$ by making p_1 adopt all attribute values representing the vertexes included into the solution for X. If after this we have not selected values for some attributes yet, the remaining values are selected arbitrarily. The constructed solution reaches k customers. A corollary of that double implication is that the *optimal* solution for X has k vertexes iff the *optimal* solution for $f(X)$ has k customers.

Let X be an instance of MIS and Y be a solution for that instance. Let $R_{\texttt{MIS}}(X,Y)$ be the ratio between the number of vertexes of the optimal solution for X and the number of vertexes in Y. Let $s_{f(X)}$ be any solution for $f(X)$ and let s_X be the solution for X translated from $s_{f(X)}$ as proposed (i.e. $s_X = g(s_{f(X)})$). Given the properties mentioned in the previous paragraph, for any $r \geq 1$ we have $R_{\texttt{PDO}}(f(X), s_{f(X)}) = r$ iff $R_{\texttt{MIS}}(X, s_X) = r$. This means, in particular, that for some constant $c \geq 1$ we have that $R_{\texttt{PDO}}(f(X), s_{f(X)}) \leq r$ implies $R_{\texttt{MIS}}(X, s_X) \leq 1 + c \cdot (r-1)$, as required by *AP*-reductions (we can just set $c = 1$). Also note that our definitions of functions f and g do not take any target approximation ratio as input, so their computation time obviously does not depend on that ratio. Hence, all the requirements to construct an *AP*-reduction between MIS and PDO are met, so we have $\texttt{MIS} \leq_{AP} \texttt{PDO}$. We conclude $\texttt{PDO} \in$ Poly-APX-hard, so $\texttt{PDO} \in$ Poly-APX-complete.

Regarding the proof of (i), we need $\texttt{PD} \in \texttt{NP}$ and $\texttt{PD} \in$ NP-hard. Regarding the former, we can use similar arguments as when we showed $\texttt{PDO} \in \texttt{NPO}$ in the first paragraph of this proof. Regarding the latter, we can easily adapt the proposed AP-reduction from MIS to PDO to construct a polynomial reduction from the decision problem *Independent Set* (that is, given a graph G and a natural K, is there an independent set with K or more vertexes?), which is NP-complete, into PD.

Proof of Theorem 2

First, let us show $\texttt{SPDG} \in \texttt{EXPTIME}$. Recall that a problem is in EXPTIME if it can be solved in exponential time, that is, in a time within $\mathscr{O}(2^{n^k})$ for some $k \in \mathbb{N}$ (as opposed, for instance, to doubly exponential, where a time within $\mathscr{O}(2^{2^n})$ is allowed). First, we show that the number of possible different configurations of a game is exponential w.r.t. the size of the input of SPDG. Each configuration consists of the current turn holder, the current product sold by each producer, and the numbers of customers gathered by each producer during all the last t_p turns. The current turn holder ranges within $\{1,\ldots,d\}$, each product is an element in $A_1 \times \ldots \times A_n$, all numbers of gathered customers are naturals between 0 and $\sum_{1 \leq i \leq m} num_i$, and for each producer we keep t_p of them. We deduce

that any game configuration can be expressed with polynomial number of bits w.r.t. the size of the complete problem input (also measured in bits), so the number of possible *different* game configurations is no more than exponential with respect to the size of the input. Let H be that number. We can solve SPDG by means of a brute force algorithm that explores all sequences of $min(H, t_f)$ turns from c but remembers configurations already visited, so that it avoids exploring what goes on from any configuration which was visited before during the algorithm. Since no more than H configurations will be developed in the process, this algorithm takes exponential time, so SPDG $\in$ EXPTIME.

Next we show SPDG $\in$ EXPTIME-hard. Let us consider the game G_6 as proposed in [55], which is defined as follows. Each configuration of the game is defined by a 3-tuple $(\tau, F(X,Y), \nu)$ where $\tau \in \{1,2\}$, F is a CNF formula (i.e. it is a conjunction of disjunctive clauses) over disjoint variable sets X and Y, and ν is a $(X \cup Y)$-assignment (i.e. ν is a mapping from propositional variables into $\{\top, \bot\}$). Player I (II) moves by changing the value assigned to at most one variable in X (Y); either player may pass since changing no variable amounts to a "pass." Player I wins if the formula F ever becomes true. More formally, player II can move from $(2, F, \nu)$ to $(1, F, \nu')$ if F is false under ν and ν' differs from ν in the value it assigns to at most one variable from Y, and player I can always move from $(1, F, \nu)$ to $(2, F, \nu')$ provided only that ν and ν' differ in the assignment to at most one variable in X. Note that, contrarily to *PDG*, in G_6 both players cannot win in the same game. The *problem* of determining whether there is winning strategy for player I in G_6 (hereafter called WG6) is EXPTIME-complete under logspace-reductions [55], which implies that it is also so under polynomial-reductions.

Our reduction from WG6 to SPDG will work as follows. In the SPDG instance, we will have two producers 1 and 2, and we will set $D = 1$ and $t_p = 1$. Each clause in the CNF formula required by player I to win will be represented in SPDG by a customer profile. Attributes will represent propositional variables, and attribute values will represent the truth values $\top$ and $\bot$ over those variables. For all propositional variables in set X (Y), producer 1 (2) will be able to pick the attribute values that represent making the variable true or false. Besides, each producer will only be able to pick the special value u for all attributes representing variables controlled by the *other* player. Valuations $val_{i,v}$ assigned by each customer profile (clause) to each attribute value will be defined in such a way that each customer profile will be attracted by the truth values of variables controlled by player I that make the clause true, and by truth values of variables controlled by player II which are *opposite* to those that make the clause true. Moreover, for the first attribute not controlled by producer 1 and the first attribute not controlled by producer 2, the valuations of value u will be defined ad-hoc in such a way that they give some advantage to one producer (generally producer 1). In practice, this advantage will mean that producer 2 can be the chosen one by the corresponding customer profile only if producer 2 sets all the variables it controls to values which are opposite to those required by the clause to hold (so producer 2 adds the *maximum* value it can for this customer profile), and producer 1 *also* chooses, for the variables it controls, the opposite values as those required by the clause to hold (so producer 1 adds the *mininum* value it can for this customer profile).

We polynomially reduce WG6 to SPDG as follows. Let $c = (\tau, F(X,Y), \nu)$ be an instance of WG6 where $\tau \in \{1,2\}$, $F(X,Y) = \alpha_1 \wedge \ldots \wedge \alpha_n$ is such that for all $1 \leq i \leq n$

we have $\alpha_i = a_1^i \vee \ldots \vee a_{k_i}^i$ and, for all $1 \leq j \leq k_i$, a_i^j are literals over propositional variables $V = X \cup Y$, with $X = \{x_1, \ldots, x_{m_x}\}$ and $Y = \{y_1, \ldots, y_{m_y}\}$, and ν is a valuation over X and Y. For all $1 \leq i \leq n$, let $g_{\alpha_i}^X$ and $g_{\alpha_i}^Y$ denote the numbers of variables from X and Y, respectively, appearing in α_i.

From that instance of `WG6`, we build the following instance of `SPDG`:

- Let $m = m_x + m_y$. The sets of attributes are $A_1, \ldots, A_m$, where for all $1 \leq i \leq m$ we have $A_i = \{\top_{A_i}, \bot_{A_i}, u_{A_i}\}$.
- There are two producers 1 and 2, whose sets of available values of attributes are $A_1^1, \ldots, A_m^1$ and $A_1^2, \ldots, A_m^2$, respectively, where for all $1 \leq i \leq m_x$ we have $A_i^1 = \{\top_{A_i}, \bot_{A_i}\}$ and $A_i^2 = \{u_{A_i}\}$, and for all $m_x + 1 \leq i \leq m$ we have $A_i^1 = \{u_{A_i}\}$ and $A_i^2 = \{\top_{A_i}, \bot_{A_i}\}$.
- The initial game configuration is $c_0 = (\tau, p_1, p_2, \overline{0}, \overline{0})$ with
  - $p_1 = (p_1^1, \ldots, p_{m_x}^1, p_{m_x+1}^1, \ldots, p_m^1)$
  - $p_2 = (p_1^2, \ldots, p_{m_x}^2, p_{m_x+1}^2, \ldots, p_m^2)$

  where for all $1 \leq i \leq m_x$ we have $p_i^1 = \nu(x_i)_{A_i}$, for all $m_x + 1 \leq i \leq m$ we have $p_i^2 = \nu(y_{i-m_x})_{A_i}$, and the value of p_w^q for any other values q and w is the single element in set A_w^q, that is, u_{A_w}.
- Parameters D, t_p, t_f, K are defined as follows:
  - $D = 1$,
  - $t_p = 1$,
  - $t_f = 2^{|X \cup Y|+1}$, that is, t_f equals the number of possible different configurations in the G_6 game, and
  - $K = n$.
- There are n customer profiles. For all customer profiles $1 \leq i \leq n$ we have $num_i = 1$, and the value $val_{i,v}$ this customer profile gives to variable value v is defined as follows:
  - For all $1 \leq j \leq m_x$:
    - $val_{i,\top_{A_j}} = 2$ if $l_k^i \equiv x_j$ for some $k \in \{1,2,3\}$ else $val_{i,\top_{A_j}} = 0$.
    - $val_{i,\bot_{A_j}} = 2$ if $l_k^i \equiv \neg x_j$ for some $k \in \{1,2,3\}$ else $val_{i,\bot_{A_j}} = 0$.
    - If $j = 1$ (i.e. the first attribute not controlled by producer 2) then
      - if $g_{\alpha_i}^Y > 0$ then $val_{i,u_{A_j}} = 0$, and
      - otherwise $val_{i,u_{A_j}} = 1$.

      If $j > 1$ then $val_{i,u_{A_j}} = 0$.
  - For all $m_x + 1 \leq j \leq m$:
    - $val_{i,\top_{A_j}} = 2$ if $l_k^i \equiv \neg y_{j-m_x}$ for some $k \in \{1,2,3\}$, else $val_{i,\top_{A_j}} = 0$.

* $val_{i,\bot_{A_j}} = 2$ if $l^i_k \equiv y_{j-m_x}$ for some $k \in \{1,2,3\}$, else $val_{i,\bot_{A_j}} = 0$.
* If $j = m_x + 1$ (i.e. the first attribute not controlled by producer 1) then
  - if $g^Y_{\alpha_i} > 0$ then $val_{i,u_{A_j}} = 2 \cdot g^Y_{\alpha_i} - 1$, and
  - otherwise $val_{i,u_{A_j}} = 0$.

  If $j > m_x + 1$ then $val_{i,u_{A_j}} = 0$.

We can easily see that the size of this SPDG instance is polynomial w.r.t. the size of the original WG6 instance (note that the amount $2^{|X \cup Y|+1}$ can be codified with a number of bits which is polynomial w.r.t. the size of the instance of WG6). Next we show that the answer to the original WG6 instance is *yes* iff the answer to the SPDG instance we constructed from it is *yes*.

In order to prove it, first we prove the following property. For any configuration c of G_6, let $f(c)$ be the configuration of *PDG* the previous polynomial reduction builds from c. Besides, for any configuration d of G_6 or *PDG*, let $n(d)$ denote the set of all possible new configurations of G_6 or *PDG*, respectively, we can reach from d in one move of the corresponding game. Let us see that, for any configuration $c = (\tau, F(X,Y), \nu)$ of G_6, we have $n(c) = n(f(c))$ provided that $F(X,Y)$ does not hold under assignment ν or $\tau = 1$. Let us recall that the polynomial reduction keeps the turn holder unmodified ($t = \tau$), sets the numbers of cumulated customers to $\overline{0}$ for each producer (note that $\overline{0} = 0$ in this case because $t_p = 1$), and codifies the valuation $\nu(z)$ of each propositional variable z by assigning the corresponding attribute value either $\top_{A_i}$ or $\bot_{A_i}$ to the producer representing the player that can modify that variable, and u_{A_i} to the other producer. In any move from this configuration of *PDG*, the turn is passed to the other producer (as the turn is passed to the other player in G_6), the current values of cumulated customers of each producer are reset again to $\overline{0} = 0$ (because $t_p = 1$, so only the numbers of customers gathered at the present turn matter and past customers are irrelevant), and the current turn holder can modify up to one attribute value (due to $D = 1$) of any attribute representing a propositional variable controlled by the corresponding producer (as it happens in G_6). This exactly corresponds to what the current turn holder of the original G_6 configuration can do at this configuration —provided that $F(X,Y)$ does not hold for ν or $\tau = 1$ (let us recall that, if $F(X,Y)$ holds for ν and $\tau = 2$ then the game is finished and no more moves can be done). We conclude that $n(c) = n(f(c))$ if $F(X,Y)$ does not hold under assignment ν or $\tau = 1$. By a simple inductive reasoning we deduce that, for any configuration c of G_6 and any sequence σ of consecutive configurations of G_6 reachable from c where $F(X,Y)$ is never fulfilled, there exists a sequence of consecutive configurations of *PDG* starting at $f(c)$ where the configuration traversed at each step i is the result of applying f to the configuration traversed at the i-th step of σ, and vice versa (that is, for any sequence σ' of consecutive configurations of *PDG* starting at $f(c)$ where producer 1 is never successful, there exists a sequence of consecutive configurations of G_6 starting at c where the configuration traversed at the i-th step is such that the application of f to that configuration returns the configuration traversed at the i-th step of σ').

We conclude that the *PDG* game we build from the G_6 game correctly simulates any part of the G_6 game in terms of f as long as $F(X,Y)$ does not hold during that game part. However, when $F(X,Y)$ holds at some configuration, this simulation fails

from that configuration on: in G_6, player II cannot move after $F(X,Y)$ holds (the game ends there), but in *PDG* producer 2 can keep playing after producer 1 is successful. Let us show that the following property holds:

$L \equiv$ a configuration c of G_6 allows a move that makes $F(X,Y)$ hold iff the configuration $f(c)$ allows a move that makes producer 1 be successful

It is worth to point out that, since the derived *PDG* game correctly simulates the G_6 game in terms of f as long as $F(X,Y)$ does not hold, if L holds then the following properties (a) and (b) hold:

(a) for any sequence $\sigma = c_1 \dots c_s$ of consecutive configurations of G_6 starting at c_1, if σ includes a configuration where $F(X,Y)$ holds, then the sequence of consecutive configurations $\sigma' = f(c_1) \dots f(c_s)$ of *PDG* includes a configuration where producer 1 succeeds; and

(b) for any sequence σ' of consecutive configurations of *PDG* starting at $f(c_1)$ where c_1 is a configuration of G_6, if σ' includes a configuration where producer 1 succeeds then, for some prefix σ'' of σ', the sequence $\sigma = c_1 \dots c_s$ of configurations of G_6 such that $\sigma'' = f(c_1) \dots f(c_s)$ includes a configuration where $F(X,Y)$ holds.

Note that (b) is trivially met by making σ'' be the prefix of σ' consisting of all configurations up to the first configuration where producer 1 succeeds. We can see that (a) and (b) imply that any winning strategy for G_6 is a successful strategy of *PDG* and vice versa. Thus, the answer to the original instance of `WG6` is *yes* iff the answer to the derived instance of *PDG* is *yes*.

In order to end the proof, we have to prove property L. Next we prove each implication of L:

$\Longrightarrow$: Let c be a configuration of G_6 where some move makes $F(X,Y)$ hold. In this move, either the turn holder does nothing (i.e. it passes) or it flips the value of some propositional variable belonging to the set of variables it can change. In both cases, $F(X,Y)$ holds after the move (passing or flipping a variable, respectively). Both moves can be trivially simulated by the turn holder in *PDG*, as it can also pass or flip the value of some attribute in the same way as the corresponding player flips the value of the variable. Let us see that, after the move, producer 1 succeeds. If $F(X,Y)$ holds in G_6 after the corresponding move then, for all clauses in $F(X,Y)$, at least one literal is true. By the construction of valuations $val_{i,v}$, if the attribute values of producer 1 and producer 2 represent a valuation that satisfies some clause, then the customer profile representing that clause must prefer producer 1 over producer 2, let we show it. On the one hand, even if producer 2 puts all its own variables concerning the clause to the opposite values than those required by the clause to hold (recall that the customer profile representing the clause actually *rewards* producer 2 for this), producer 1 will be ahead in the preference of the customer profile as long as at least one of its own attribute values makes the clause true. On the other hand, if at least one of the variables controlled by producer 2 has the actual value required by the clause to

hold, then producer 1 will be preferred by the corresponding customer profile even if none of the variables it controls has the value that satisfies the clause. We conclude that, if $F(X,Y)$ holds for some propositional valuation, then the corresponding equivalent selection of attribute values lets producer 1 be the selected producer for all customer profiles. Thus, producer 1 gets n customers at the current turn, and producer 1 succeeds. Hence, if the turn holder of *PDG* passes or flips some attribute value in the same way as in G_6, then *PDG* reaches a configuration where producer 1 succeeds.

$\Longleftarrow$: Let c be a configuration where producer 1 succeeds. This means that, after the current move of the turn holder of *PDG*, player 1 gets n customers. This implies that it is the preferred producer for all customer profiles. By the construction of valuations $val_{i,v}$, producer 1 can be the preferred producer for some customer profile only if the attribute values controlled by producer 1 and producer 2 represent a valuation of propositional variables that satisfies the clause represented by this customer profile (it is easy to reverse the argument given in the previous case). We conclude that all clauses of $F(X,Y)$ hold after the move. Since the move performed in *PDG* (passing or flipping some attribute value) can also be done by the corresponding player of G_6, we infer that the same move makes player I win in G_6.

Proof of Theorem 3

It is easy to see that a minimax algorithm that checks all sequences of up to $t_f \leq n$ moves solves `lSPDG` in polynomial space. For any node of the tree reached during the exploration, the algorithm just has to remember whether this node can still be part of a winning strategy, as well as where we are in the exploration tree in order to be able to backtrack. The former requires constant memory, whereas the latter requires polynomial memory (we just have to remember all moves done in the current branch). We conclude `lSPDG` $\in$ `PSPACE`.

Next we prove `lSPDG` $\in$ `PSPACE-hard`. Let us consider the `TQBF` problem (*true quantified boolean function* problem). This PSPACE-complete decision problem is defined as follows. We say that a QBF is a expression of the form

$$Q_1 x_1 Q_2 x_2 Q_3 x_3 Q_4 x_4 \ldots Q_n x_n \ \varphi$$

where, for all $1 \leq i \leq n$, $Q_i \in \{\forall, \exists\}$ and quantifiers $\forall$ and $\exists$ strictly alternate through the sequence $Q_1, \ldots, Q_n$, and φ is a propositional logic formula over propositional symbols $x_1, \ldots, x_n$ (see that all variables in the expression appear under the scope of some quantifier, i.e. there are no free variables). `TQBF` is the set of QBF expressions that evaluate to $\top$. Thus, solving `TQBF` consists in finding out whether the given expression is equivalent to $\top$ or $\bot$. Note that, since all variables in QBF expressions are quantified, all QBF expressions are equivalent to either $\top$ or $\bot$. Though the propositional expression inside each QBF, that is φ, is allowed to follow any form, from now on we will assume that φ is expressed in CNF. This variant of the problem is also known to be PSPACE-complete, as any QBF instance can be polynomially transformed into an equivalent QBF where φ is given in CNF [56].

In order to prove lSPDG $\in$ PSPACE-hard, we polynomially reduce TQBF to lSPDG.[13] First, let us informally present this reduction. Two producers 1 and 2 will play the PDG game. Each variable in the QBF expression will be represented in PDG by an attribute. The values available for each attribute A_i will be both truth values $\top_{A_i}$ and $\bot_{A_i}$, as well as two additional special values r_{A_i} and u_{A_i}. Producer 1 will be able to set the values of variables associated to *existential* quantifiers of the QBF expression, whereas producer 2 will control variables bounded to *universal* quantifiers. We will set $D = 1$, so the turn holder of PDG will be able to change only up to one attribute value in its turn. For any variable under control of some producer, that producer will be able to pick values $\top_{A_i}$, $\bot_{A_i}$, and r_{A_i} (the value selected by this producer at the beginning of the game will be r_{A_i}), whereas the other producer will only be able to pick u_{A_i}. Let n be the number of quantifiers in the QBF expression and m be the number of clauses of the CNF expression after these quantifiers. In order to succeed in the PDG game, each producer will be required to reach some average number of customers during the n turns of the game (in particular, $\frac{n^2 \cdot m}{2} + \frac{n \cdot m}{2} + m$). For the lSPDG instance we will construct, we will check whether there exists a succeeding strategy for producer 1.

Two kinds of customer profiles will be considered. On the one hand, some customers will be gained or lost by each producer just depending on the variables which already have truth values $\top_{A_i}$ or $\bot_{A_i}$ rather than special values r_{A_i} or u_{A_i}. In particular, for any variable x_i controlled by some producer, some customer profile will be attracted by the special value u_{A_i}, but even more attracted by any truth value $\top_{A_i}$ or $\bot_{A_i}$. Thus, this customer profile will prefer the producer which does not control the variable only until the producer controlling the variable sets any truth value for this variable. Therefore, in each turn of PDG, each producer will be able to gain some customer profile in its turn just by picking some truth value for some variable, and it will lose some customer profile in the turn of the other producer if the other producer also picks some truth value for some of the variables it controls. Hence, both producers will have some incentive to iteratively pick truth values for their respective variables during the n turns of the game. Note that, if this idea were implemented literally, then producers could have the *same* incentive regardless of the order in which they pick truth values for their variables, but we want variables to be set in the same order as they appear in the QBF expression. In order to avoid this problem, we introduce the following change. For the first variable appearing in the QBF expression which is controlled by some producer, this producer will gain the corresponding customer profile just by picking a truth value for that variable, as we said before. However, for the *second* variable controlled by that producer, it will gain the corresponding customer profile only when it has truth values for its *first* variable and its *second* variable. More generally, for each variable controlled by some producer, this producer will gain the corresponding customer pro-

[13] In order to prove lSPDG $\in$ PSPACE-hard, initially we considered the possibility of proving that an alternative version of the problem G_6 used in the proof of Theorem 2 (in particular, a version where the game ends after $|X \cup Y|$ turns) is PSPACE-hard by using a reduction from TQBF, and next reducing this version to lSPDG similarly as in the proof of Theorem 2. However we noticed that, in a reduction from TQBF, making variables be set in the same order as they appear in the QBF expression would be much easier if the reduction went directly to lSPDG rather than to that modified version of G_6. Thus, we chose to perform the reduction presented here.

file only if it has already adopted truth values for this variable *as well as* all previous variables in the QBF expression controlled by this producer. Note that, under these new conditions, any producer maximizes the number of gained customer profiles only if it picks truth values for the variables it controls in the same order as they appear in the QBF expression.

On the other hand, a second kind of customers will be considered. We will have a customer profile for each disjunctive clause appearing in the propositional part of the QBF expression (recall that this part of the expression is given in CNF). Similarly as in the construction given in the proof of Theorem 2, these customer profiles will be used to make sure that producer 1 gains some number of customers only if the truth values selected by both producers let all clauses hold. In order to create this effect, the expressions defining the valuations of attribute values for this customers will be taken almost literally from those we used in the proof of Theorem 2. However, they will have a difference. Note that, during the iterative process of putting truth values to each variable, some clauses could hold even *before* all variables have some truth value. In order to avoid any alteration of the numbers of customers gained by producer 1 due to this effect, we will modify the valuations of these customer profiles so that producer 1 will not be able to gain any of them until *all* variables have been assigned truth values (which cannot happen before the last turn of the PDG game). In particular, producer 2 will necessarily gain all of these customers as long as producer 1 has not put truth values to all the variables it controls. Producer 1 will gain some value for all variables with some truth value, whereas producer 2 will get some fix high value, which will be further increased if some of its own variables is still undefined. In this way, a necessary condition for producer 1 to gain these customer profiles is that all variables, both its variables and those of producer 2, have a truth value. It is worth to remark that the value gained due to having some truth value will always be higher than the value gained by fulfilling some clause (which is taken literally from the construction of the proof of Theorem 2). So, the value gained by fulfilling some clause will never compensate the value not gained by not having yet truth values for all variables controlled by producer 1 and producer 2. In this way, producer 1 will gain these customers only after all variables have truth values.

The customer profiles described in the two last paragraphs are related as follows. We will set the numbers of both kinds of customers in such a way that the numbers of customers mentioned in the paragraph before the last one will be *much more* than the numbers of customers mentioned in the last paragraph. In particular, getting all clauses (i.e. making all of them true) will never compensate not having gained all customers due to assigning all truth values to variables in the expected order. In this way, producer 1 will never be able to succeed by altering this order or by leaving some variable undefined (i.e. *passing* in some turn), and producer 2 will never be able to prevent producer 1 to be successful by altering the order in which it sets its own truth variables or by leaving undefined some of its variables: in either case, producer 1 would win regardless of whether all clauses are fulfilled or not.

Next we formally present the polynomial reduction from `TQBF` to `lSPDG`. Without loss of generality, let $\exists\, x_1 \forall\, x_2 \ldots \exists\, x_{n-1}\, \forall\, x_n\ \varphi$ be an instance of `TQBF` (note that quantifiers over useless variables can be added to make any instance of `TQBF` fit that particular quantifying pattern), where $\varphi = \alpha_1\ \wedge\ \ldots\ \wedge\ \alpha_m$ and for all $1 \leq i \leq m$ we

have $\alpha_i = a_1^i \vee \ldots \vee a_{k_i}^i$ where, for all $1 \leq j \leq k_i$, a_j^i are literals over propositional variables. For all $1 \leq i \leq m$, let $g^o_{\alpha_i}$ and $g^e_{\alpha_i}$ denote the numbers of variables with odd and even indexes, respectively, appearing in α_i.

From that TQBF instance, we build an instance of lSPDG as follows:

- The sets of attributes are $A_1, \ldots, A_n$, where for all $1 \leq i \leq n$ we have $A_i = \{\top_{A_i}, \bot_{A_i}, r_{A_i}, u_{A_i}\}$.
- There are two producers 1 and 2 whose sets of available values of attributes are $A_1^1, \ldots, A_n^1$ and $A_1^2, \ldots, A_n^2$, respectively, where for all $1 \leq i \leq n$ such that i is odd we have $A_i^1 = \{\top_{A_i}, \bot_{A_i}, r_{A_i}\}$ and $A_i^2 = \{u_{A_i}\}$, and for all $1 \leq i \leq n$ such that i is even we have $A_i^1 = \{u_{A_i}\}$ and $A_i^2 = \{\top_{A_i}, \bot_{A_i}, r_{A_i}\}$.
- The initial game configuration is $c_0 = (1, p_1, p_2, \overline{0}, \overline{0})$ with
  - $p_1 = (p_1^1, \ldots, p_n^1)$
  - $p_2 = (p_1^2, \ldots, p_n^2)$

  where for all $1 \leq i \leq n$ we have $p_i^1 = r_{A_i}$ and $p_i^2 = u_{A_i}$ if i is odd, and $p_i^1 = u_{A_i}$ and $p_i^2 = r_{A_i}$ if i is even.
- parameters D, t_p, t_f, K are defined as follows:
  - $D = 1$,
  - $t_p = n$,
  - $t_f = n$,
  - $K = \frac{n^2 \cdot m}{2} + \frac{n \cdot m}{2} + m$.
- There are $m + n$ customer profiles. For all customer profiles $1 \leq i \leq m$ (i.e. those representing the *clauses* in the CNF part of φ) we have $num_i = n$, and the valuation $val_{i,v}$ this customer profile gives to variable value v is defined as follows:
  - For all $1 \leq j \leq n$ such that j is odd,
    - $val_{i,\top_{A_j}} = 2n+2$ if $l_k^i \equiv x_j$ for some $k \in \{1,2,3\}$ else $val_{i,\top_{A_j}} = 2n$.
    - $val_{i,\bot_{A_j}} = 2n+2$ if $l_k^i \equiv \neg x_j$ for some $k \in \{1,2,3\}$ else $val_{i,\bot_{A_j}} = 2n$.
    - $val_{i,r_{A_j}} = 0$.
    - If $j = 1$ (i.e. the first attribute not controlled by producer 2) then
      - if $g^e_{\alpha_i} > 0$ then $val_{i,u_{A_j}} = n^2$, and
      - otherwise $val_{i,u_{A_j}} = n^2 + 1$.

      If $j > 1$ then $val_{i,u_{A_j}} = 0$.
  - For all $1 \leq j \leq n$ such that j is even,
    - $val_{i,\top_{A_j}} = 2$ if $l_k^i \equiv \neg y_{j-m_x}$ for some $k \in \{1,2,3\}$, else $val_{i,\top_{A_j}} = 0$.
    - $val_{i,\bot_{A_j}} = 2$ if $l_k^i \equiv y_{j-m_x}$ for some $k \in \{1,2,3\}$, else $val_{i,\bot_{A_j}} = 0$.

* $val_{i,r_{A_j}} = 2n$.
* If $j = 2$ (i.e. the first attribute not controlled by producer 1) then
  - if $g^e_{\alpha_i} > 0$ then $val_{i,u_{A_j}} = 2 \cdot g^e_{\alpha_i} - 1$, and
  - otherwise $val_{i,u_{A_j}} = 0$.

  If $j > 2$ then $val_{i,u_{A_j}} = 0$.

Besides, for all customer profiles $m+1 \leq i \leq m+n$ (i.e. those representing the *variables* in the QBF expression) we have $num_i = n \cdot m$, and $val_{i,v}$ is defined as follows:

– For all $1 \leq j \leq n$:
  * $val_{i,\top_{A_j}} = 2$ if $i - m \geq j$ else $val_{i,\top_{A_j}} = 0$.
  * $val_{i,\bot_{A_j}} = 2$ if $i - m \geq j$ else $val_{i,\bot_{A_j}} = 0$.
  * $val_{i,r_{A_j}} = 0$.
  * If $j = 1$ or $j = 2$ (i.e. the first attributes not controlled by each producer) then $val_{i,u_{A_j}} = 2 \cdot \ulcorner (\frac{i-m}{2}) \urcorner - 1$ else $val_{i,u_{A_j}} = 0$.

It is easy to see that the size of this `lSPDG` instance is polynomial w.r.t. the size of the original `TQBF` instance. Next we show that the answer to the original `TQBF` instance is *yes* iff the answer to the `lSPDG` instance we construct from it is *yes*.

$\Longrightarrow$: Let us suppose that $\exists\, x_1 \forall\, x_2 \ldots \exists\, x_{n-1} \forall\, x_n\ \varphi$ is true. That is, there exists x_1 such that, for all x_2, there exists x_3 such that, ... such that φ holds. This means that, if we view `TQBF` as a game where player I picks the values of odd variables, player II picks the values of even variables, and the goal of player I is fulfilling φ, then player I has a *winning strategy*. Let us see that the direct translation of this strategy into the derived PDG game is successful for producer 1. Obviously, in any `TQBF` game where player I follows that winning strategy, all n variables are alternatively set by each player, in the same order as they appear in the QBF expression, until all variables have a truth value. Let us note that, in each turn of producer 2 in the PDG game, producer 2 is not forced to give a truth value to the corresponding next variable in the QBF expression. Moreover, it does not have to give a truth value to *any* variable (it could pass), or even it could turn one of its variables already having a truth value $\top_{A_i}$ or $\bot_{A_i}$ back into r_{A_i} again. However, even a single "unexpected" move like this of producer 2 would let producer 1 succeed, even without the necessity to eventually fulfill all clauses of φ: if producer 2 does not gain the customer profile it is expected to gain in each of its turns by assigning a truth value to the corresponding variable, then producer 1 unexpectedly keeps that customer profile of itself and gets additional $n \cdot m$ customers from that customer profile, which equals the number of customers producer 1 would gain in the last turn if all variables had truth values and all clauses held under these values. Producer 1 can gain all the remaining customers it needs to be successful by assigning truth values to its variables in the expected order (producer 2 cannot avoid it), so producer 1 can trivially succeed. Thus,

the extra capability of producer 2 to perform new moves does not let producer 2 turn a winning strategy for player I in TQBF into a non-successful strategy for producer 1 in 1SPDG.

Let us suppose that producer 2 does not perform any of these unexpected moves and producer 1 follows in PDG the same strategy as that of TQBF. Producer 1 gets $\frac{n}{2}+1$ customer profiles after its first move: it still gains $\frac{n}{2}$ customer profiles designed to prefer producer 2 *after* producer 2 puts truth values to the corresponding variables (which still prefer producer 1 because producer 2 has not done so yet), and one customer profile of the opposite kind, which now prefers producer 1 because it has just set a truth value to the first variable it controls. These customer profiles amount $\left(\frac{n}{2}+1\right)\cdot n\cdot m = \frac{n^2\cdot m}{2}+n\cdot m$ customers. After producer 2 makes its own first move (by giving a truth value to its first variable), producer 2 gains one of the customer profiles previously owned by producer 1. Hence, after that move producer 1 has $\frac{n^2\cdot m}{2}$ customers. In subsequent moves up to the last move, producer 1 alternates $\frac{n^2\cdot m}{2}+n\cdot m$ customers in odd turns and $\frac{n^2\cdot m}{2}$ customers in even turns. Besides, since the strategy is winning in TQBF, in the last turn of any game of PDG the attribute values selected by both producers let all clauses of φ hold, so producer 1 gets m additional customer profiles (amounting $n\cdot m$ additional customers). Note that producer 1 is the producer selected by all of these customer profiles because: (a) since it has assigned a truth value to all of its variables, it gains $2n\cdot\frac{n}{2}=n^2$ valuation due to that reason, which equals the valuation of producer 2 due to variables with truth values (producer 2 has n^2 valuation by default, but it does not add any more value due to undefined variables because it assigned truth values to all of its variables); and (b) producer 1 adds more value than producer 2 due to fulfilling the corresponding clause, in the same way as we showed in the construction of the proof of Theorem 2. All in all, in the last turn of the PDG game, the number of customers collected by producer 1 during all the n turns is $\frac{n}{2}\cdot m\cdot n\cdot n+n\cdot m\cdot\frac{n}{2}+n\cdot m=\frac{n^3\cdot m}{2}+\frac{n^2\cdot m}{2}+n\cdot m$. Hence, the average number of customers of producer 1 during the last n turns is $\frac{n^2\cdot m}{2}+\frac{n\cdot m}{2}+m$, and producer 1 is successful.

$\Longleftarrow$: Let us suppose that producer 1 has a winning strategy in the derived PDG game. Let us show that a TQBF strategy derived from (part of) that PDG strategy is necessarily winning for player I. Let us note that a successful strategy for producer 1 in PDG also includes the strategy of producer 1 for cases where producer 2 does not properly set truth values to its variables (i.e. it does not assign them in the expected order or leaves some variables without truth values), but these cases are impossible in a TQBF game and cannot be included in a derived strategy for TQBF. A successful strategy for producer 1 must also be successful, in particular, for the cases where producer 2 properly assigns truth values for its variables, which are the cases actually concerned in a strategy for TQBF. Since the strategy for PDG is successful for producer 1, the average number of customers of producer 1 in any game end reached after following this strategy is at least $\frac{n^2\cdot m}{2}+\frac{n\cdot m}{2}+m$. Note that this amount is impossible to reach if producer 1 does *not* properly put truth

values to its own variables: in that case, at some turn producer 2 would get $n \cdot m$ additional customers that would have belonged to producer 1 if it had properly assigned truth values to its variables (we can show it by using a similar argument as we did in the previous case, after exchanging the roles of producer 1 and producer 2), so producer 1 would never get $\frac{n^2 \cdot m}{2} + \frac{n \cdot m}{2} + m$ average customers. We conclude that, if we take only the part of the successful strategy of PDG where producer 2 always assigns truth values to its variables properly, we get a strategy where producer 1 always assigns truth values to its variables properly. Thus, by taking that part of the successful strategy of PDG, we directly have a valid strategy for TQBF, as no player performs any invalid move (i.e. both players assign truth values to their variables in the expected order). Besides, this strategy always lets player I fulfill all clauses of φ, as producer 1 could not always get $\frac{n^2 \cdot m}{2} + \frac{n \cdot m}{2} + m$ average customers if it did not always add the $n \cdot m$ customers due to fulfilling all clauses in the last turn. We conclude that the strategy for TQBF derived from the part of the successful strategy of PDG where only valid TQBF moves are performed is winning for player I in TQBF.

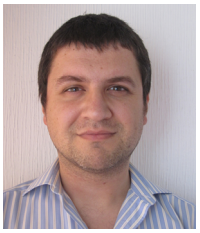
Proof of Theorem 4

Let us denote the standard version of PDO (that is, combining options (I) and (i)) by R. Problem R is Poly-APX-complete by Theorem 1, so it is also Poly-APX-hard. Problem R can be trivially AP-reduced to any other PDO variant resulting from combining (I)-(II)-(III) with (i)-(ii) in any way, as all of them are generalizations of R. Let T be any of these variants. If (II) is chosen in the definition of T, then the reduction sets $e = n$; if (III) is selected instead, then we set $feasible_j(p_j) = \texttt{true}$ for all j and $p_j \in A_1^j \times \ldots \times A_n^j$; also, if (ii) is chosen, then we just define $profit_j(p_j) = 1$ for all j and p_j. By the transitivity of the AP-reductions, we get that T is Poly-APX-hard.

Regarding the corresponding variants of SPDG, we can trivially build polynomial reductions from the standard version of SPDG to any of them by using the same ideas (i.e. setting either $e = n$ or $feasible_j(p_j) = \texttt{true}$; and/or setting $profit_j(p_j) = 1$). Since the standard version of SPDG is EXPTIME-hard, by transitivity by infer that all of these variants are EXPTIME-hard too. Besides, since it is assumed that functions $attrib_j$, $feasible_j$, and $profit_j$ take polynomial time, all of these problem variants can be solved in exponential time by following the same scheme as in the original SPDG problem (see the proof of Theorem 2). Thus, all of these variants are EXPTIME-complete.

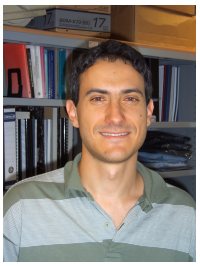
We can show that the variants of lSPDG are PSPACE-hard in the same way as we have done to show that the respective variants of SPDG are EXPTIME-hard, as the original version of lSPDG can be polynomially reduced to its variants by using the same ideas. Also, the scheme used to solve the original lSPDG problem in polynomial space (see the proof of Theorem 3) can also be followed to solve these variants of lSPDG in polynomial space (recall that $attrib_j$, $feasible_j$, and $profit_j$ are polynomial-time functions, so they can be computed with polynomial space). Hence, these lSPDG variants are PSPACE-complete.

Biosketches



Ismael Rodríguez is an Associate Professor in the Computer Systems and Computation Department, Complutense University of Madrid (Spain). He obtained his MS degree in Computer Science in 2001 and his PhD in the same subject in 2004. Dr. Rodríguez received the Best Thesis Award of his faculty in 2004. Dr. Rodríguez has published more than 100 papers in international refereed conferences and journals. His research interests cover formal methods, testing techniques, swarm and evolutionary optimization methods, and functional programming.

Pablo Rabanal is an Assistant Professor in the Computer Systems and Computation Department, Complutense University of Madrid (Spain). He obtained his MS degree in Computer Science in 2004 and his PhD in the same subject in 2010, devoted to the development of nature-inspired techniques to solve NP-complete problems. Dr. Rabanal has published more than 30 papers in international refereed conferences and journals. His research interests cover swarm and evolutionary optimization methods, formal methods, testing techniques, and web services.



Fernando Rubio is an Associate Professor in the Computer Systems and Computation Department, Complutense University of Madrid (Spain). He obtained his MS degree in Computer Science in 1997, and he was awarded by the Spanish Ministry of Education with "Primer Premio Nacional Fin de Carrera". He finished his PhD in the same subject four years later. Dr. Rubio received the Best Thesis Award of his faculty in 2001. Dr. Rubio has published more than 80 papers in international refereed conferences and journals. His research interests cover formal methods, swarm and evolutionary optimization methods, parallel computing, and functional programming.