Multidimensional Mechanism Design in Single Peaked Type Spaces

Debasis Mishra, Anup Pramanik, and Souvik Roy

March 1, 2014

Abstract

We consider implementation of a deterministic allocation rule using transfers in quasi-linear private values environments. We show that in multidimensional single peaked type spaces, an allocation rule is implementable if and only if it satisfies a familiar and simple condition called 2-cycle monotonicity.

Keywords: implementation, 2-cycle monotonicity, revenue equivalence, single-peakedness.

JEL codes: D44, D47, D71, D82, D86.

*We are grateful to an editor and two anonymous referees for constructive comments. We are also thankful to Rudolf Muller, Arunava Sen, Rakesh Vohra, and audience at the 2013 biennial conference of the Society for Economic Design for useful comments.

†All authors are affiliated to Economics and Planning Unit, Indian Statistical Institute, New Delhi. Corresponding author: Debasis Mishra, Economics and Planning Unit, Indian Statistical Institute, 7, SJS Sansanwal Marg, New Delhi - 110016, India, Email: dmishra@isid.ac.in.
1 Introduction

An enduring theme in mechanism design is to investigate conditions that are necessary and sufficient for implementing an allocation rule. We investigate this question in private value and quasi-linear utility environment when the set of alternatives is finite and the allocation rule is deterministic (i.e., does not randomize). An allocation rule in such an environment is implementable if there exists a payment rule such that truth-telling is a dominant strategy for the agents in the resulting mechanism. Our main result is that in multidimensional single peaked type spaces, implementability is equivalent to a simple condition called 2-cycle monotonicity. Our proof techniques can also be used to derive this result in other type spaces where ordinal restrictions are present. By virtue of revenue equivalence, which holds in these type spaces, we are able to characterize the entire class of dominant strategy incentive compatible mechanisms. The 2-cycle monotonicity condition requires the following: given the types of other agents, if the alternative chosen by the allocation rule is $a$ when agent $i$ reports its type to be $t$ and the alternative chosen by the allocation rule is $b$ when agent $i$ reports its type to be $s$, then it must be that

$$t(a) - t(b) \geq s(a) - s(b),$$

where for any alternative $x$, $t(x)$ and $s(x)$ denote the values of alternative $x$ in types $t$ and $s$ respectively.

We use a novel method to impose ordinal restriction on type spaces. To see how ordinal restrictions can be imposed in a cardinal environment like ours, note that a type in our environment is a vector in $\mathbb{R}^{|A|}$, where $A$ is the set of alternatives. Now, let us restrict attention to strict types, where value of no two alternatives is the same. Such a type must induce a strict linear ordering on $A$. We put restrictions by allowing only a subset of strict linear orderings that can be induced by any strict type. The set of all strict types in $\mathbb{R}_+^{|A|}$ that induce a strict linear ordering belonging to a set of permissible strict linear orderings define a strict type space. To allow for indifferences, we take the closures of such type spaces. We call such type spaces ordinal type spaces. Though our result extends to a large class of ordinal type spaces, our main result in this paper concerns the type space induced by all single peaked strict linear ordering on a tree graph, where the graph consists of alternatives as nodes and a strict linear ordering must be single peaked along paths of the tree.\(^1\) Single peakedness on a tree is due to Demange (1982), and it is a generalization of classical single peaked preference orderings.\(^2\)

---

\(^1\)Roughly, a single peaked type is defined using a strict and complete order on the set of alternatives. A type is single peaked if the values of alternatives decrease as we go to the left or right (where left and right are defined with respect to the given order) of the peak (the highest valued alternative).

\(^2\) In the working paper version of this paper, we identify more ordinal type spaces where our result
One of the earliest papers to pursue the question of identifying necessary and sufficient conditions for implementability was Rochet (1987), who proved a very general result. He showed that a significantly stronger condition called cycle monotonicity is necessary and sufficient for implementability in any type space - see also Rockafellar (1970). Myerson (1981) formally establishes that in the single object auction set up, where the type is single dimensional, 2-cycle monotonicity is necessary and sufficient for implementation (in Myerson’s set up, this is true even if we consider randomized allocation rules) - see also Spence (1974); Mirrlees (1976). When the type space is multidimensional, if the set of alternatives is finite and the type space is convex, 2-cycle monotonicity implies cycle monotonicity (Bikhchandani et al., 2006; Saks and Yu, 2005; Ashlagi et al., 2010). Though convexity is a natural geometric property satisfied in many economic environments, it excludes many interesting type spaces. Moreover, how far this result extends to type spaces that do not satisfy convexity remain an intriguing question - we discuss this issue in detail in Section 2. A primary objective of this paper is to formulate restrictions on type spaces without the convexity assumption made in the literature and answer the question of implementability in such multidimensional type space. Indeed, our multidimensional single peaked type space on a tree is a non-convex type space. To our knowledge, this paper is the first to identify such an interesting non-convex type space where 2-cycle monotonicity characterizes implementability.

A characterization of implementability using 2-cycle monotonicity is useful because the cycle monotonicity condition, which can be used to characterize implementability in any type space, is a difficult condition to use and interpret. On the other hand, 2-cycle monotonicity is a simpler condition. It is also the appropriate extension of the monotonicity condition used by Myerson (1981) to characterize implementability in the single object auction model. For this reason, 2-cycle monotonicity is often referred to as weak monotonicity (Bikhchandani et al., 2006; Saks and Yu, 2005) or monotonicity (Ashlagi et al., 2010).

An important objective in mechanism design is to design expected revenue maximizing mechanisms. While Myerson (1981) solved this problem for the sale of a single object, the problem remains unsolved for multidimensional problems - see a recent take on this topic in Manelli and Vincent (2007); Hart and Nisan (2012); Hart and Reny (2012). However, as Myerson illustrates, there are two important steps in solving the optimal auction problem: (a) characterizing the implementable allocation rules using a monotonicity property and (b) establishing revenue equivalence to pin down the payments. Though the eventual optimization problem remains illusive in the multidimensional type spaces, the literature has made significant progress in advancing these two steps for multidimensional type spaces. Our results add to this literature and we hope that these advances will eventually help us solve the

holds (Mishra et al., 2013).
revenue maximization problem in the multidimensional type spaces.

The recent literature on multidimensional mechanism design started with the paper of Jehiel et al. (1999) who observed that besides 2-cycle monotonicity, an integral condition is required to ensure Bayesian implementability in multidimensional environments with randomization. There are many papers which characterize different extensions of implementability in convex type spaces using 2-cycle monotonicity and additional technical conditions - for Bayes-Nash implementation, see Jehiel et al. (1999) and Muller et al. (2007); for randomized implementation, see Archer and Kleinberg (2008); for implementation with general value functions, see Berger et al. (2010) and Carbajal and Ely (2013); for extension of cycle monotonicity to general environments, see Rahman (2011). Our result is an extension of these results to a non-convex type space. Mishra and Roy (2013) also consider a non-convex type space, which they call rich dichotomous type space, and show that 3-cycle monotonicity, a stronger condition than 2-cycle monotonicity but weaker than cycle monotonicity, is sufficient for implementability in their type space but 2-cycle monotonicity is not sufficient.

A parallel literature in multidimensional mechanism design pursues type spaces where revenue equivalence result in Myerson (1981) holds. Contributions to this are Krishna and Maenner (2001); Milgrom and Segal (2002); Chung and Olszewski (2007); Heydenreich et al. (2009); Carbajal (2010); Kos and Messner (2013). Revenue equivalence holds in our type space and helps us to characterize the set of payments that implement a 2-cycle monotone allocation rule.

Most of the type space restrictions considered in the multidimensional mechanism design literature is geometric (using assumptions like convexity or connectedness in topological spaces). Our idea of imposing ordinal restrictions follow a rich tradition in social choice theory. For instance, the single peaked type spaces and their generalizations are widely studied in strategic social choice theory without transfers - seminal work include Moulin (1980) and Sprumont (1991). However, allowing for transfers in such type spaces is practical in many contexts. Hence, our results extend this literature to the case of transfers.

2 IMPLEMENTATION AND CYCLE MONOTONICITY

We consider a model with a single agent. As is well known in this literature, this is without loss of generality. All our results generalize easily to a model with multiple agents. The single agent is denoted by \( i \). The set of alternatives for agent \( i \) is denoted by \( A \). In an \( n \)-agent model, \( A \) denotes the possible allocations of agent \( i \). The type (private information) of

\[3\]For instance, in a model with \( n \) agents and \( n \) objects, where each agent can be assigned exactly one object and there is no externality in allocations across agents, \( A \) will be the set of objects and not the set of matchings.
agent $i$ is a vector $t \in \mathbb{R}^{|A|}$. If agent $i$ has type $t$, then $t(a)$ will denote the value of agent $i$ for alternative $a$. We assume private values and quasi-linear utility. This means that if alternative $a$ is chosen and agent $i$ with type $t$ makes a payment of $p$, then his net utility is given by $t(a) - p$.

Not all possible vectors in $\mathbb{R}^{|A|}$ can be a type of agent $i$. Let $D \subseteq \mathbb{R}^{|A|}$ be the type space of agent $i$ - these are the permissible types of agent $i$. An allocation rule is a mapping $f : D \rightarrow A$. We will assume that $f$ is onto. This is standard in the literature - if $f$ is not onto, then all the results can be restated in terms of range of $A$.

A payment rule of agent $i$ is a mapping $p : D \rightarrow \mathbb{R}$. A mechanism consists of an allocation rule and a payment rule.

**Definition 1** An allocation rule $f$ is implementable if there exists a payment rule $p$ such that for every $s, t \in D$, we have

$$s(f(s)) - p(s) \geq s(f(t)) - p(t).$$

In this case, we will say that $p$ implements $f$ and $(f, p)$ is an incentive compatible mechanism.

The primary objective of this paper is to give a simple necessary condition on the allocation rule that is also sufficient for implementability in a large class of interesting type spaces. For this, we revisit a classic condition that is already known to be necessary and sufficient for implementability in any type space.

**Definition 2** An allocation rule $f$ is $K$-cycle monotone, where $K \geq 2$ is a positive integer, if for every finite sequence of types $(t^1, t^2, \ldots, t^k)$, with $k \leq K$, we have

$$\sum_{j=1}^{k} \left[ t^j(f(t^j)) - t^j(f(t^{j-1})) \right] \geq 0,$$

where $t^0 \equiv t^k$. An allocation rule $f$ is cyclically monotone if it is $K$-cycle monotone for every positive integer $K \geq 2$.

It is well known that implementability is equivalent to cycle monotonicity (Rochet, 1987; Rockafellar, 1970). This result is very general - it works on any type space $D$ and does not even require $A$ to be finite. However, cycle monotonicity is a difficult condition to use and interpret since it requires verifying non-negativity of Inequality 1 for arbitrary length

---

4 When the set of alternatives is finite, this result can be slightly strengthened to say that implementability is equivalent to $|A|$-cycle monotonicity (Mishra and Roy, 2013).
sequences of types. In a series of papers, it has been established that a significantly weaker condition than cycle monotonicity is sufficient for implementation in various interesting type spaces. Bikhchandani et al. (2006) showed that 2-cycle monotonicity is sufficient for implementability if $D$ is an order-based type space - this includes many interesting type spaces in the context of multi-object auctions. Saks and Yu (2005) show that 2-cycle monotonicity is sufficient for implementation if $D$ is convex - this extends the result in Bikhchandani et al. (2006) because an order-based type space is convex. Ashlagi et al. (2010) extend this result to show that if the closure of $D$ is convex, then 2-cycle monotonicity is sufficient for implementation.

However, Mishra and Roy (2013) show that there are interesting non-convex type spaces where 2-cycle monotonicity is not sufficient for implementation. Further, they identify an interesting class of non-convex type spaces where 3-cycle monotonicity is sufficient for implementation but 2-cycle monotonicity is not sufficient.

Interestingly, Ashlagi et al. (2010) establish a surprising result by allowing for randomization, i.e., an allocation rule picks a probability distribution over alternatives. They show that if every 2-cycle monotone randomized allocation rule is also cyclically monotone in a type space $D$ of dimension at least 2, then the closure of $D$ must be convex.

It is not clear how far this result is true if $f$ is allowed to be deterministic. Vohra (2011) contains a simple example of a non-convex type space with four alternatives where every deterministic allocation rule satisfying 2-cycle monotonicity is implementable. In his example, Vohra (2011) considers the sale of two objects $\alpha$ and $\beta$ to agents. The set of alternatives is the set of all subsets of $\{\alpha, \beta\}$. The restriction on values of agents is the following: $t(\{\alpha, \beta\}) = \max(t(\{\alpha\}), t(\{\beta\}))$ and $t(\emptyset) = 0$. Hence, each agent desires at most one object, though he may be assigned both the objects. The type space here is non-convex. To see this, consider two types of agent $i$

\[
\begin{align*}
t(\emptyset) &= 0, t(\{\alpha\}) = 3, t(\{\beta\}) = 4, t(\{\alpha, \beta\}) = 4 \\
s(\emptyset) &= 0, s(\{\alpha\}) = 5, s(\{\beta\}) = 4, s(\{\alpha, \beta\}) = 5.
\end{align*}
\]

A convex combination of $(0.5, 0.5)$ of these two types generates values 4 for objects $\alpha$ and $\beta$ but a value of 4.5 for the bundle of objects $\{\alpha, \beta\}$. This violates the restriction on the type space.

Note that if we allow at most one object to be assigned to an agent, then the type space becomes convex, and we can apply earlier result to conclude that 2-cycle monotonicity is sufficient for implementation. However, by allowing the alternative $\{\alpha, \beta\}$, but still having a restriction that agents desire at most one object, we get to a non-convex type space. The result in Vohra (2011) shows that 2-cycle monotonicity is sufficient for implementation in such an example. It is not clear on how to extend the proof of this example if there are more
than two objects.

2.1 A Motivating Example

Since the type space in the example in Vohra (2011) seems to be a slight modification of a convex type space, it is still unclear whether there are interesting non-convex type space where 2-cycle monotonicity is sufficient for implementation. The result in Ashlagi et al. (2010) shows that if every 2-cycle monotone randomized allocation rule is implementable in a multidimensional type space, then it must be convex. This shows that there is a significant gap in understanding implementability of deterministic allocation rules in non-convex multidimensional type spaces. We give below a motivating example to show that there are interesting non-convex type spaces where the current results are silent. Our results in the paper will apply to such type spaces.

Consider a general scheduling problem as follows. A number of firms procure products/parts from a supplier over a time horizon. In each time period, the supplier can only supply to one firm. Every firm \( i \) has a time period \( \tau^* \) where it gets the maximum value from getting its products supplied. The firms have single peaked preference over time, i.e., for any time periods \( \tau, \tau' \), if \( \tau < \tau' < \tau^* \) or \( \tau > \tau' > \tau^* \), then a firm values supply of its products at time period \( \tau' \) to time period \( \tau \) (this may be due to inventory carrying cost and delivery delay costs).

The type space in this example is non-convex. To see this, suppose there are just three time periods \{1, 2, 3\} and consider two single peaked types of an agent (firm): \( s := (6, 4, 3) \) (peak value is period 1) and \( t := (3, 4, 6) \) (peak value is period 3). A convex combination \( \frac{s+t}{2} \) produces the type \((4.5, 4, 4.5)\), which is no longer single peaked.

There are other problems where one encounters single peaked preferences. For instance, consider an agent who is being sold multiple products/objects, but the agent is interested to buy at most one object. Each object has a quality and a maintenance cost, which depends on the quality. The value for an object depends on the trade off between maintenance cost due to quality and value for quality. Under reasonable assumptions on these cost and value functions, one gets single peaked preferences, i.e., an optimum level of quality such that below and above that quality level, value for the objects decline. If we consider the standard multi-object auctions in this framework, one gets a restriction on type space that types have to be single peaked.

In such non-convex type spaces, we characterize implementability using 2-cycle monotonicity and apply revenue equivalence to obtain a complete characterization of dominant

\(^5\)For instance, if an object with quality \( q \) generates a net value of \( vq - q^2 \), where \( v \) is per unit value of quality and \( q^2 \) is the maintenance cost, then the optimal quality level is \( \frac{v}{2} \).
strategy incentive compatible mechanisms. Thus, there are interesting type spaces where earlier results are silent and our results provide sharp characterizations of implementability and incentive compatibility.

3 The Single Peaked Type Space on a Tree

We start off by considering the problem of choosing an alternative (a location) over a tree network. Our network $G$ is given by a finite set of nodes $A$ and a set of undirected edges $E$ between these nodes. The set $A$ is the set of alternatives or outcomes from which one of the alternatives must be chosen. We will assume that $G$ is a tree, i.e., a graph whose edges do not form any cycles and there is a unique path between every pair of alternatives/nodes. The private information or type of the agent is a vector $t \in \mathbb{R}^{|A|}_+$, where $t(a)$ denotes the value for alternative $a$ at this type. The set of possible types (type space) of each agent will be determined by $G$.

We define the type space by imposing ordinal restriction on type spaces. Notice that each type induces a weak ordering on the set of alternatives. We call a type $t$ strict if $t(a) \neq t(b)$ for all $a \neq b$. A strict type induces a strict linear order on the set of alternatives. Let $\mathcal{P}$ be the set of all strict linear orders over $A$. Given a linear order $P \in \mathcal{P}$, we denote the $k$-th ranked alternative in $P$ as $P(k)$. Given any pair of alternatives $a, b \in A$, there exists a unique path in $G$ between $a$ and $b$, and we denote this unique path as $\Pi(a, b)$. With a slight abuse of notation, we let $\Pi(a, b)$ to denote also the set of alternatives (including $a$ and $b$) in the unique path from $a$ to $b$ in $G$. A strict linear order $P \in \mathcal{P}$ is single peaked with respect to $G$ if for every $a \in A$ and every $b \neq a$ but $b \in \Pi(a, P(1))$, we have $bPa$. Let $\mathcal{D} \subseteq \mathcal{P}$ be the set of all single peaked strict linear orders in $\mathcal{P}$. If $G$ is a line graph, this reduces to the standard definition of single peaked preference ordering. This generalization of single peakedness is due to Demange (1982).

Definition 3 The strict single peaked type space $T^G$ (with respect to $G$) is the set of all non-negative type vectors that induce a strict linear order in $\mathcal{D}$, i.e.,

$$T^G := \{ t \in \mathbb{R}^{\left|A\right|}_+ : t \text{ induces } P \text{ for some } P \in \mathcal{D} \}.$$  

The single peaked type space is $cl(T^G)$, where $cl(T^G)$ denotes the closure of the set $T^G$.

To give an example, consider $A = \{a, b, c\}$ and the line graph $G$ with edges $\{a, b\}, \{b, c\}, \{c, a\}$. This corresponds to the following four strict linear orderings in $\mathcal{D}$: $aPbPc, bPaPc, bPcPa$, and $cPbPa$. Then, $T^G$ consists of four cones in $\mathbb{R}^3$, each corresponding to a strict linear ordering in $\mathcal{D}$. Figure 1 shows the projection of $T^G$ onto the hyperplane $t(a) + t(b) + t(c) = 1$. 

8
Figure 1: Single peaked type space with three alternatives

The dashed region in Figure 1 corresponds to $T^G$ with the projection of each cone separated by dark lines. It is clear that $T^G$ is non-convex.

The main result of this paper is the following.

**Theorem 1** An allocation rule $f : \text{cl}(T^G) \rightarrow A$ is implementable if and only if it is 2-cycle monotone.

**Remark.** In many contexts, it is natural to assume that there is an alternative whose value is always zero (for instance, in auction problems, the alternative of not getting any object gives zero value to the agent). Though we do not explicitly allow this in our model, all our proofs can be modified straightforwardly to accommodate the fact that there is an alternative which is worst ranked and has value zero at every type.

**Remark.** The characterization of the set of all payment rules that implement an allocation rule is done using the revenue equivalence principle.

**Definition 4** An allocation rule $f$ satisfies **revenue equivalence** if for all payment rules $p, q$ that implement $f$, there exists a constant $\alpha \in \mathbb{R}$ such that for all $t \in D$

$$p(t) = q(t) + \alpha.$$

The revenue equivalence holds in the single peaked type space on a tree since it is a connected subset of $\mathbb{R}^{|A|}$ - this follows from Chung and Olszewski (2007). In the proof of Theorem 1, we construct a payment rule that implements a 2-cycle monotone allocation rule. Using revenue equivalence, we thus characterize the entire class of dominant strategy
incentive compatible mechanisms.

**Remark.** We give an example to illustrate how tight Theorem 1 is. Below, we consider an example with three alternatives and \( D \) that contains one more preference ordering than the set of all single peaked orderings. We show that 2-cycle monotonicity is not sufficient for implementability in this domain. This highlights the fact that 2-cycle monotonicity is not sufficient for implementability in *every* domain defined by ordinal restrictions.

**Example 1**

Let \( A = \{a, b, c\} \) and \( D \) consists of the following strict linear orderings shown in Table 1.

<table>
<thead>
<tr>
<th></th>
<th>( P^1 )</th>
<th>( P^2 )</th>
<th>( P^3 )</th>
<th>( P^4 )</th>
<th>( P^5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>c</td>
<td>b</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>c</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: 2-cycle monotonicity is not sufficient for implementability

Notice that \( D \setminus \{P^5\} \) is the set of all single peaked strict linear orderings with respect to the graph with edges \( \{a, b\}, \{b, c\}, \{c, a\} \) (i.e., the line graph with \( a \) and \( c \) as endpoints). Let \( D \) be the set of all non-negative vectors in \( \mathbb{R}^3_+ \) that is consistent with a strict linear ordering in \( D \). We now define an allocation rule \( f : cl(D) \rightarrow A \) as follows.

\[
    f(t) = \begin{cases} 
    a & \text{if } t(a) - t(b) \geq 1, t(a) > t(b) \geq t(c) \\
    b & \text{if } t(a) - t(b) < 1, t(a) \geq t(b) \geq t(c), t(a) \neq t(b) \neq t(c) \\
    b & \text{if } t(b) \geq t(a) \geq t(c), t(a) \neq t(b) \neq t(c) \\
    b & \text{if } t(b) \geq t(c) \geq t(a), t(a) \neq t(b) \neq t(c) \\
    c & \text{if } t(c) \geq t(b) \geq t(a), t(a) \neq t(b) \neq t(c) \\
    c & \text{if } t(c) > t(a) \geq t(b) \\
    a & \text{if } t(c) = t(a) > t(b), t(a) - t(b) \geq 1 \\
    c & \text{if } t(c) = t(a) > t(b), t(a) - t(b) < 1 \\
    c & \text{if } t(a) = t(b) = t(c). 
\end{cases}
\]

Note that \( d(a, b) = -1, d(b, a) = 1, d(b, c) = d(c, b) = d(c, a) = d(a, c) = 0. \)

Now, to verify if \( f \) is \( K \)-cycle monotone for each \( K \), we use the following equivalent definition of \( K \)-cycle monotonicity. An allocation rule \( f \) is \( K \)-cycle monotone if for every integer \( k \leq K \) and every sequence of alternatives \( (a_1, \ldots, a_k) \), we have \( d(a_1, a_2) + \ldots + d(a_{k-1}, a_k) + d(a_k, a_1) \geq 0 \). The equivalence of this definition of \( K \)-cycle monotonicity and
the earlier definition of $K$-cycle monotonicity is well known, and can be found, for instance in (Vohra, 2011; Heydenreich et al., 2009).

Using this, we see that $f$ is 2-cycle monotone. But $d(a,b) + d(b,c) + d(c,a) = -1$ implies that $f$ is not 3-cycle monotone, and hence, not implementable.

4 Conclusion

We have shown that in the multidimensional single peaked type space on a tree, 2-cycle monotonicity is a necessary and sufficient condition. Our proof technique is quite general and can be applied to other type spaces also. In particular, Lemmas 2 and 3 (see Appendix below) are fundamental steps in our proof. In the working paper version of this paper (Mishra et al., 2013), we identify more type spaces where these lemmas continue to hold, and as a result, the characterization holds. The type spaces we identify are derived by imposing ordinal restrictions on the set of permissible strict linear orderings that a type can induce. Hence, they need not be convex. At a technical level, it will be interesting to identify necessary and sufficient conditions on the ordinal restrictions that allow for such a result. In future, we will like to identify applications of these results to construct specific mechanisms.

Appendix: Proof of Theorem 1

The proof of Theorem 1 will be done using a series of Lemmas. These lemmas will reveal the underlying structure of the type space. Further, we will show how these steps can be used in other type spaces to extend Theorem 1.

Denote by $D \equiv cl(T^G)$. First, by Rochet (1987), if $f : D \to A$ is implementable, then it is 2-cycle monotone. Next, again by Rochet (1987), if $f$ is cyclically monotone, then it is implementable. So, we will show that if $f$ is 2-cycle monotone, then it is cyclically monotone. In the remainder of the section, we assume that $f$ is 2-cycle monotone.

For every $a \in A$, define $D(a)$ as follows.

$$D(a) := \{ t \in D : f(t) = a \}.$$  

Since $f$ is onto, $D(a)$ is non-empty. Next, for every $s, t \in D$, define $\ell(s, t)$ as follows.

$$\ell(s, t) := t(f(t)) - t(f(s)).$$

Notice that 2-cycle monotonicity is equivalent to requiring that for every $s, t \in D$, we have $\ell(s, t) + \ell(t, s) \geq 0$. Now, for every $a, b \in A$, define $d(a, b)$ as follows.

$$d(a, b) := \inf_{t \in D(b)} [t(b) - t(a)].$$
We state below a well known fact - see, for instance, Lemma 6 in Bikhchandani et al. (2006).

**Lemma 1** For every \( a, b \in A \), \( d(a,b) + d(b,a) \geq 0 \).

For any \( a, b \in A \), we say \( a \) and \( b \) are **G-neighbors** if the unique path between \( a \) and \( b \) in \( G \) is a direct edge between \( a \) and \( b \) in \( G \). The following facts will be useful throughout the proofs. These facts are true due to the single peakedness of the type space.

**Fact 1** For any \( a, b \in A \), if \( a \) and \( b \) are G-neighbors, then there exists a strict linear order \( P \in D \) such that \( P(1) = a, P(2) = b \).

Fact 1 says that if \( a \) and \( b \) are G-neighbors then there is some strict linear ordering where they are ranked first and second.

**Fact 2** For any \( a, c \in A \) and \( b \in \Pi(a,c) \) such that \( b \) is a G-neighbor of \( a \), there exists a strict linear order \( P \in D \) such that \( \{P(1), P(2)\} = \{a,b\} \), \( xPc \) for all \( x \in \Pi(a,c) \setminus \{c\} \), and \( cPx \) for all \( x \notin \Pi(a,c) \).

Fact 2 says that if \( a \) and \( c \) are any pair of alternatives with \( b \) being a G-neighbor of \( a \) in \( \Pi(a,c) \), then there is some strict linear ordering where \( a \) and \( b \) are first and second ranked, followed by the other alternatives in \( \Pi(a,c) \), and followed by the remaining alternatives outside \( \Pi(a,c) \). The first step of the proof of Theorem 1 is the following lemma.

**Lemma 2** If \( a, b \) are G-neighbors, then \( d(a,b) + d(b,a) = 0 \).

**Proof:** Consider \( a, b \in A \) such that \( a \) and \( b \) are G-neighbors. By Lemma 1, \( d(a,b) + d(b,a) \geq 0 \). Assume for contradiction \( d(a,b) + d(b,a) = \epsilon > 0 \). Then, either \( d(a,b) > \frac{\epsilon}{2} \) or \( d(b,a) > \frac{\epsilon}{2} \). Suppose \( d(a,b) > \frac{\epsilon}{2} \) - a similar proof works if \( d(b,a) > \frac{\epsilon}{2} \). Then, there is a type \( s \in D(b) \) such that \( d(a,b) \leq s(b) - s(a) < d(a,b) + \epsilon_1 \), for any \( \epsilon_1 > 0 \) arbitrarily close to zero, in particular \( \epsilon_1 < \frac{\epsilon}{2} \). Hence, \( s(b) - s(a) > \frac{\epsilon}{2} \). We now choose a \( \delta \in (2\epsilon_1, s(b) - s(a)) \) but arbitrarily close to \( 2\epsilon_1 \). Since \( a \) and \( b \) are G-neighbors, by Fact 1, there exists a \( P \in D \) such that \( b \) is top ranked and \( a \) is second ranked. We can construct a type \( u \in D \) that induces \( P \) and

\[
  u(x) = \begin{cases} 
    s(x) + \delta & \text{if } x = a \\
    s(x) + \frac{\delta}{2} & \text{if } x = b \\
    \min(s(x), s(a)) & \text{if } x \notin \{a, b\}.
  \end{cases}
\]

Notice that since \( s(b) > s(a) \), we have \( u(b) > u(a) \) for sufficiently small \( \delta > 2\epsilon_1 \). Also, alternatives other than \( b \) and \( a \) are ordered according to \( P \) but their values are not increased.

We will now argue that \( f(u) = a \). First, if \( f(u) = x \notin \{a, b\} \), we have \( u(x) - u(b) \leq s(x) - s(b) - \frac{\delta}{2} < s(x) - s(b) \), which violates 2-cycle monotonicity. Second, if \( f(u) = b \), we
have \( u(b) - u(a) = s(b) - s(a) - \frac{\delta}{2} < d(a, b) - (\frac{\delta}{2} - \epsilon_1) < d(a, b) \), which violates the definition of \( d(a, b) \). Hence, \( f(u) = a \).

But this implies that \( d(b, a) \leq u(a) - u(b) = s(a) - s(b) + \frac{\delta}{2} \leq -d(a, b) + \frac{\delta}{2} \). Hence, \( d(b, a) + d(a, b) \leq \frac{\delta}{2} \). Since \( \delta, \epsilon_1 \) can be chosen arbitrarily close to zero, this contradicts the fact that \( d(a, b) + d(b, a) = \epsilon > 0 \).

The next step is to show that for any pair of alternatives \( a \) and \( c \), there is some alternative \( b \in \Pi(a, c) \) such that a version of the reverse triangle inequality holds between \( a, b, \) and \( c \) using \( d(\cdot, \cdot) \).

**Lemma 3** For any pair of alternatives \( a, c \in A \) such that \( a \) and \( c \) are not \( G \)-neighbors, there exists an alternative \( b \in \Pi(a, c) \) such that

\[
d(a, b) + d(b, c) \leq d(a, c).
\]

**Proof:** Fix \( a, c \in A \) such that \( a \) and \( c \) are not \( G \)-neighbors. Choose an \( \epsilon > 0 \) and arbitrarily close to zero and a \( t \in D(c) \) such that \( d(a, c) \leq t(c) - t(a) \leq d(a, c) + \epsilon \). We consider two cases.

**Case 1.** \( t(c) \geq t(a) \). Choose \( b \in \Pi(a, c) \) such that \( b \) is a \( G \)-neighbor of \( c \). By single peakedness, for every \( x \in \Pi(a, c) \), we have \( t(x) \geq t(a) \). Then, we can construct a new type in which \( b \) and \( c \) occupy the top two ranks. We construct such a new type \( s \) as follows. Choose \( \epsilon' > 0 \) but arbitrarily close to zero and let \( \delta := t(c) - t(b) - d(b, c) + 2\epsilon' \). Note that since \( t \in D(c) \), we have \( t(c) - t(b) \geq d(b, c) \), and this implies that \( \delta > 0 \).

\[
s(x) = \begin{cases} 
  t(x) + \epsilon' & \text{if } x = c \\
  t(a) & \text{if } x \in \Pi(a, c) \setminus \{b, c\} \\
  t(x) + \delta & \text{if } x = b \\
  \leq \min(t(x), t(a)) & \text{if } x \notin \Pi(a, c).
\end{cases}
\]

By Fact 2, we can define \( s \) such that it is in \( cl(T^G) \). We argue that \( f(s) = b \). First, suppose \( f(s) = x \notin \{b, c\} \). Then, \( s(x) - s(c) < t(x) - t(c) \), and this contradicts 2-cycle monotonicity. Next, suppose \( f(s) = c \). Then, \( d(b, c) \leq s(c) - s(b) = t(c) - t(b) - \delta + \epsilon' = d(b, c) - \epsilon' < d(b, c) \), a contradiction. Hence, \( f(s) = b \).

Now, \( d(a, b) \leq s(b) - s(a) = [t(b) - t(a) + \delta] = t(c) - t(a) - d(b, c) + 2\epsilon' \leq d(a, c) - d(b, c) + 2\epsilon' + \epsilon \). Since \( \epsilon \) and \( \epsilon' \) can be chosen arbitrarily close to zero, we conclude that \( d(a, b) + d(b, c) \leq d(a, c) \).

**Case 2.** \( t(c) < t(a) \). Let \( b \in \Pi(a, c) \) be the \( G \)-neighbor of \( a \). Define the subset of alternatives \( C \) as follows: \( C := \{c' \in \Pi(b, c) : t(c') = t(c) \text{ and } \forall c'' \in \Pi(c', c), t(c'') = t(c) \} \). In other
words, \( C \) is the set of “contiguous” alternatives in \( \Pi(b,c) \) starting from \( c \) which have the same value as \( t(c) \) in type \( t \). Now, construct a new type \( s \) as follows. Choose an \( \epsilon' > 0 \) but arbitrarily close to zero. Note that \( \epsilon' \) can be chosen sufficiently close to zero such that for all \( x \in \Pi(a,c) \setminus C \), we have \( t(x) > t(c) + \epsilon' \). Also, choose \( \delta = t(c) - t(b) - d(b,c) + 2\epsilon' \). As before, \( \delta > 0 \).

\[
 s(x) = \begin{cases} 
 t(c) + \epsilon' & \text{if } x \in \Pi(a,c) \setminus \{a,b\} \\
 t(x) & \text{if } x = a \\
 t(x) + \delta & \text{if } x = b \\
 \leq \min(t(x), s(c)) & \text{if } x \notin \Pi(a,c). 
\end{cases}
\]

Again, by Fact 2, such a type \( s \) can be found in \( cl(T^G) \). If \( f(s) = x \notin C \cup \{b\} \), then \( s(x) - s(c) < t(x) - t(c) \), which violates 2-cycle monotonicity. Hence, \( f(s) \in C \cup \{b\} \). If \( f(s) = c \), then \( d(b,c) \leq s(c) - s(b) = t(c) - t(b) - \delta + \epsilon' = d(b,c) - \epsilon' < d(b,c) \), a contradiction. So, \( f(s) \in (C \setminus \{c\}) \cup \{b\} \). We consider two subcases.

**Case 2A.** Suppose \( f(s) = c' \in C \setminus \{c\} \). Then, \( d(c,c') \leq s(c') - s(c) = 0 \). But \( f(t) = c \) implies that \( d(c',c) \leq t(c) - t(c') = 0 \). This implies that \( d(c,c') + d(c',c) \leq 0 \). By Lemma 1, \( d(c,c') + d(c',c) = 0 \). Since, \( d(c,c') \leq 0 \) and \( d(c',c) \leq 0 \), we conclude that \( d(c,c') = d(c',c) = 0 \). Further, since \( f(s) = c' \), \( d(a,c') \leq s(c') - s(a) = t(c) - t(a) + \epsilon' \leq d(a,c) + \epsilon + \epsilon' \). Since \( \epsilon \) and \( \epsilon' \) can be chosen arbitrarily small, \( d(a,c') \leq d(a,c) \). Hence, \( d(a,c') + d(c',c) \leq d(a,c) \), where we used the fact that \( d(c',c) = 0 \). This completes the proof of this case.

**Case 2B.** Suppose \( f(s) = b \). Then, \( d(a,b) \leq s(b) - s(a) = t(b) - t(a) + \delta = t(c) - t(a) - d(b,c) + 2\epsilon' \leq d(a,c) - d(b,c) + 2\epsilon' + \epsilon \). Since \( \epsilon \) and \( \epsilon' \) can be chosen arbitrarily small, \( d(a,b) + d(b,c) \leq d(a,c) \). This completes the proof of this case.

Lemmas 2 and 3 are the foundations of our proof. The next lemma (and many subsequent lemmas) is a consequence of these two lemmas. Lemmas 2 and 3 are the only place where we use the fact that the type space is \( cl(T^G) \). This implies that as long as we can prove analogues of Lemmas 2 and 3 in a type space, Theorem 1 continues to hold. Now, consider the following lemma.

**Lemma 4.** For any pair of alternatives \( a_1, a_k \in A \), let \( \Pi(a_1,a_k) = (a_1, a_2, \ldots, a_k) \) with \( k > 2 \). Then, the following are true.

\[
 d(a_1,a_2) + d(a_2,a_3) + \ldots + d(a_{k-1},a_k) \leq d(a_1,a_k) \\
 d(a_k,a_{k-1}) + d(a_{k-1},a_{k-2}) + \ldots + d(a_2,a_1) \leq d(a_k,a_1).
\]
Proof: Consider any pair of alternatives \(a_1, a_k \in A\) and let \((a_1, a_2, \ldots, a_k)\) be the sequence of alternatives on \(\Pi(a_1, a_k)\). We do the proof using induction on \(k\). If \(k = 3\), then the claim is true due to Lemma 3. Suppose the claim is true for all \(k < K\). If \(k = K\), then by Lemma 3, there is an alternative \(a_r \in \{a_2, \ldots, a_{K-1}\}\) such that \(d(a_1, a_r) + d(a_r, a_K) \leq d(a_1, a_K)\). The paths \((a_1, \ldots, a_r)\) and \((a_r, \ldots, a_K)\) each contain less than \(K\) nodes. By our induction hypothesis, \(d(a_1, a_2) + \ldots + d(a_{r-1}, a_r) \leq d(a_1, a_r)\) and \(d(a_r, a_{r+1}) + \ldots + d(a_{K-1}, a_K) \leq d(a_r, a_K)\). Hence, \(d(a_1, a_2) + \ldots + d(a_{K-1}, a_K) \leq d(a_1, a_K)\).

A similar argument shows that \(d(a_k, a_{k-1}) + d(a_{k-1}, a_{k-2}) + \ldots + d(a_2, a_1) \leq d(a_k, a_1)\). \(\blacksquare\)

The following lemma is well known - see, for instance, Heydenreich et al. (2009).

**Lemma 5** Suppose for every sequence of alternatives \((a_1, \ldots, a_k)\), we have

\[
\sum_{j=1}^{k} d(a_j, a_{j+1}) \geq 0,
\]

where \(a_{k+1} \equiv a_1\). Then, \(f\) is cyclically monotone.

At this point, it will be useful to consider another graph \(G^f\). \(^6\) The set of nodes in \(G^f\) is the set of alternatives \(A\). It is a complete directed graph. Hence, for every pair of alternatives \(a, b \in A\), there is an edge from \(a\) to \(b\) and an edge from \(b\) to \(a\). A path from an alternative \(a\) to another alternative \(b\) in \(G^f\) is a directed path. Note that for every path \((a_1, a_2, \ldots, a_k)\) in \(G^f\) from \(a_1\) to \(a_k\), the corresponding undirected path may or may not exist in \(G\). For any pair of alternatives \(a_1, a_k \in A\), denote by \(\text{dist}^f(a_1, a_k)\) the shortest path length from \(a_1\) to \(a_k\) in \(G^f\).

The next lemmas shows that the shortest path in \(G^f\) between a pair of alternatives \(a\) and \(b\) is the unique path \(\Pi(a, b)\) in \(G\).

**Lemma 6** For any pair of alternatives \(a, b \in A\), let \(\Pi(a, b) \equiv (a \equiv a_1, a_2, \ldots, b \equiv a_k)\). Then,

\[
\sum_{j=1}^{k-1} d(a_j, a_{j+1}) = \text{dist}^f(a, b).
\]

Proof: Fix \(a, b \in A\) and choose a shortest path from \(a\) to \(b\) in \(G^f\). Note that \(G^f\) consists of directed edges and, hence, this path consists of directed edges. Let this path be \((a'_1, \ldots, a'_h)\), where \(a'_1 \equiv a\) and \(a'_h \equiv b\). Also, there is a unique path \(\Pi(a, b)\) between \(a\) and \(b\) in \(G\), and this consists of undirected edges in \(G\). We will sometimes refer to this path in \(G^f\) by turning them into directed edges - this will give rise to two directed paths, one from \(a\) to \(b\) and the

\(^6\)In Heydenreich et al. (2009), this graph is called the allocation graph.
other from $b$ to $a$. We will denote these two directed paths corresponding to $\Pi(a, b)$ in $G^f$ as $\Pi(a, b)$ and $\Pi(b, a)$.

Now, take any (directed) edge $(x, y)$ in the path $(a'_1, \ldots, a'_h)$. If $x$ and $y$ are not $G$-neighbors, then we can pick the path $\bar{\Pi}(x, y) \equiv (x, c_1, \ldots, c_r, y)$ from $x$ to $y$, and $d(x, y) \geq d(x, c_1) + d(c_1, c_2) + \ldots + d(c_{r-1}, c_r) + d(c_r, y)$ by Lemma 4. Combining such paths $\bar{\Pi}(a'_j, a'_{j+1})$ for all $j \in \{1, \ldots, h - 1\}$, we get the path $\bar{\Pi}(a, b)$ from $a$ to $b$ in $G$, which we denote by $(a_1, \ldots, a_k)$ with $a \equiv a_1$ and $b \equiv a_k$, and some cycles in $G^f$. Since these cycles all consist of edges from $G$, and $G$ does not have a cycle, it must be that these cycles are 2-cycles. By Lemma 2, these cycles have zero length (according to weights defined in $G^f$). Hence, $\text{dist}^f(a, b) = \sum_{j=1}^{k-1} d(a_j, a_{j+1})$.

This leads to the final lemma in the proof of Theorem 1.

**Lemma 7** Every cycle of $G^f$ has non-negative length.

**Proof:** Consider a cycle $(a_1, \ldots, a_k, a_1)$ in $G^f$. By Lemma 6, the unique path $\Pi(a_1, a_k) \equiv (a_1, b_1, \ldots, b_r, a_k)$ in $G$ satisfies $d(a_1, b_1) + d(b_1, b_2) + \ldots + d(b_{r-1}, b_r) + d(b_r, a_k) = \text{dist}^f(a_1, a_k) \leq d(a_1, a_2) + \ldots + d(a_{k-1}, a_k)$. This shows that

$$d(a_1, a_2) + \ldots + d(a_{k-1}, a_k) \geq d(a_1, b_1) + d(b_1, b_2) + \ldots + d(b_{r-1}, b_r) + d(b_r, a_k).$$

Now, consider the path $(a_k, b_r, \ldots, b_1, a_1)$ from $a_k$ to $a_1$. By Lemma 4,

$$d(a_k, a_1) \geq d(a_k, b_r) + d(b_r, b_{r-1}) + \ldots + d(b_2, b_1) + d(b_1, a_1).$$

Adding the previous two inequalities, we get

$$\sum_{j=1}^{k} d(a_j, a_{j+1}) \geq [d(a_1, b_1) + d(b_1, a_1)] + [d(b_1, b_2) + d(b_2, b_1)] + \ldots$$

$$+ [d(b_{r-1}, b_r) + d(b_r, b_{r-1})] + [d(a_k, b_r) + d(b_r, a_k)]$$

$$= 0,$$

where $a_{k+1} \equiv a_1$ and the last equality follows from Lemma 2 and the fact that consecutive alternatives on the path $(a_1, b_1, \ldots, b_r, a_k)$ are $G$-neighbors.

**Lemmas 7 and 5** establish that $f$ is cyclically monotone, and hence, implementable. This completes the proof of Theorem 1.

A final remark about the payment that implements a cyclically monotone allocation rule. It is well known that by fixing some $a \in A$ and letting $p_i(t_i) = 0$ if $f(t_i) = a$ and
\[ p_i(t_i) = \text{dist}^f(a, f(t_i)) \] if \( f(t_i) \neq a \) is a payment rule that implements a cyclically monotone allocation rule (Vohra, 2011; Kos and Messner, 2013). Lemma 6 shows the specific structure of \( \text{dist}^f \) in our type space. Using revenue equivalence, we have a complete description of the payments rules that implement a 2-cycle monotone allocation rule in our type space.

**REFERENCES**


