1 Computational Solution to the Assignment Model

The purpose of the assignment model is to determine the minium possible equilibrium prices, relative to some constraint price, for a set of indivisible items with heterogenous quality, and the resulting assignment of agents to items. It is assumed that agents differ with regards to their incomes and have heterogeneous preferences over divisible composite consumption good, c, and the quality of the indivisible item to which they are assigned, q. Each agent has demand for one item only.

Define a set of n items $x_i \in X$ with heterogeneous qualities, $q_i = q(x_i) : X \to Q$, where $Q \subset \mathbb{R}$. Also, let the indices of these items be ordered by quality, such that for all $x_a, x_b \in X$, $a < b$ iff $q_a < q_b$.

Also define a set of n agents $a_j \in A$ with associated incomes $y_j = y(a_j) : A \to \mathbb{R}_{>0}$. Each agent is constrained by the budget constraint $c_j + p_i \leq y_j$, where p_i is the price of the item it is assigned to. Due to monotonicity of utility, the agent will always choose to consume at the budget constraint, $c_j + p_i = y_j$. Agents also have heterogeneous utility functions, $v_j(c, q) : D_j \to U_j$, where c_j is the divisible composite consumption good, q is the item quality and $U_j \subset \mathbb{R}$. These functions must be defined over

$$D_j = \left\{(c, q) \in \mathbb{R}^2 : 0 < c \leq y_j,\ \min_i q_i \leq q \leq \max_i q_i\right\}$$

which is all positive values of consumption up to and including the income of the agent, and the range of all possible values of item quality. These functions must also be strictly increasing in quality and consumption, and strictly concave.

$$\frac{\partial v_j}{\partial c} > 0, \qquad \frac{\partial v_j}{\partial q} > 0 \tag{1}$$

$$\frac{\partial^2 v_j}{\partial c^2} < 0, \qquad \frac{\partial^2 v_j}{\partial q^2} < 0 \tag{2}$$

Finally, to always guarantee the existence of a solution, we need it to be the case that the indifference curves in the (c, q) plane generated by the function v_j approach the line $c = 0$ asymptotically, without ever intersecting it. Thus, we need

$$\forall q \in Q,\ \lim_{c \to 0} \mathrm{MRS}_j(c, q) = \lim_{c \to 0} \frac{\partial v_j(c, q)}{\partial c} \frac{\partial q}{\partial v_j(c, q)} = \infty \tag{3}$$

This means, for any level of utility in the range of $v^* \in v_j$ and any level of quality, q we can always find a price $p^* < y_j$ which an agent can afford, and which will give the utility v^*, such that $v_j(p^*, q) = v^*$. This is explained in more detail in appendix (A).

For the purposes of the assignment model, the utility functions are expressed in terms of price, p, and not consumption, c, as it allows us to compare the preferences of all agents in the same plane. Thus, we use the transformed utility function

$$u_j(p, q) = v_j(y_j - p, q)$$

moving forward.

Finally, we construct a set of **allocations**, $l_i \in \hat{L}$, which consist of a set of tuples mapping items X to agents A, such that $l_i = (p_i, a_j) \in L$, $p_i \in \mathbb{R}^+$, $x_i \in X$, $a_j \in A$ maps agent j to item i. We can consider these allocations to be like slots with an associated price p_i, where each slot represents an assignment of the agent placed into the slot to its corresponding item at the price p_i. Each slot can only contain one agent, and agents can move between slots, but l_i will always assign its agent to the same item i. Each agent and each item can each only be associated with one allocation - so multiple agents cannot be allocated to the same item and multiple items cannot be allocated to the same agent.

Define a **configuration** to be a set of allocations $\hat{L} \in \mathcal{L}$, where $\mathcal{L}$ is the set of all possible allocations. $\hat{L}$ represents the assignments of the given agents to their respective items at some prices. Thus, $\hat{L}$ in essence defines a one-to-one function from A to X, as well as giving a price for each item.

For ease of notation, the utility function of an agent at allocation l_i will be referred to as follows:

$$u(p, q;\ l_i) = u_j(p, q) \quad \text{where } (p_i, a_j) = l_i, \quad l_i \in \hat{L}$$

The goal of the algorithm is to determine an equilibrium configuration, $\hat{L}^* \in \mathcal{L}$, which is a one-to one assignment of every agent in A to an item in X at prices that yield a **minimum price non-envy equilibrium** (MPNE), which means that no agent would want to switch to another allocation, and prices are at their lowest possible values. That is, no agent would obtain a strictly higher utility by being allocated to any other item, given the prices assigned to each item by $\hat{L}^*$, and it is not possible to reduce the prices (and increase the utility of the allocated agent) of any allocation without violating the equilibrium or the constraint price. Therefore,

$$\hat{L}^* \equiv \{l_i : u(p_i, q_i;\ l_i) \geq u(p_j, q_j;\ l_i)\ \forall j \neq i\ \wedge [\forall p \in [0, p_i)\ \exists k \text{ s.t. } u(p, q_i;\ l_k) > u(p_k, q_k;\ l_k) \vee (i = 0 \wedge p_0 = p_c)]\} \tag{4}$$

where $p_0 = p_c$ acts as the constraint, constraining the price of item x_0 to p_c. The constraint price will usually be set to zero, but it can theoretically take any value less than the income of the agent with the lowest income.

$$p_c < \min_j y_j$$

If this condition is violated, no MPNE will exist.

We can permit an agent to be indifferent between other allocations, because we assume that if an agent is indifferent, they would have no incentive to switch to another allocation (which would imply excess demand). It also allows us to determine a specific value for a minimum price that we could set for an item. If we did not allow indifference, we would need to consider the limiting price that brings the utility of one allocation arbitrarily close, but still above, the utility of another, which would not technically meet the criteria for a MPNE.

We say that an allocation l_i is **invalid** if the agent that it allocates prefers some other allocation, if the allocation is not at the minimum possible price such some other agent does not prefer another allocation or if it violates the constraint price. That is

$$\exists l_j \in \hat{L} \text{ s.t. } u(p_j, q_j;\ l_i) > u(p_i, q_i;\ l_i) \vee \exists p \in [0, p_i) \text{ s.t. } \forall k,\ u(p, q_i;\ l_k) \leq u(p_k, q_k;\ l_k) \vee (i = 0 \wedge p_0 \neq p_c)$$

A configuration is in equilibrium iff it does not contain any invalid allocations. This follows from the definition of an equilibrium configuration in expression (4). An allocation is **valid** if it is not invalid.

Since we have heterogeneous preferences, the assignment of agents to items is not trivial. Intuitively, if preferences were homogeneous, houses would be assigned purely according to income, such that if we were to rank items according to quality and agents according to income, each agent would be assigned to the item at the same rank. In other words there would be positive assertive matching, which is shown by Määttänen and Terviö (2014)[**?**]. Fundamentally, this is because, in the case of homogeneous preferences, the utility function of any agent can be described as a translation of any other agent's utility function by the difference in incomes.

However, heterogenous preferences complicate things, because there is no simple way to determine which agent should be assigned to which item. The assignment depends on the income, MRS between consumption and quality, and the prices of other items. In order to solve algorithmically, we must identify a procedure for adding a new allocation to the configuration such that we will always restore the configuration to a MPNE state after its addition.

1.1 Distances

Let the distance between any two allocations is the absolute value of the difference in quality between the allocations. For some allocation l_i, some other allocation l_a is described as farther away from l_i than some other allocation l_b iff

$$|q_a - q_i| > |q_b - q_i|$$

This becomes intuitive when graphing the allocations on the p, q plane.

1.2 Graphing of Configurations

We can graph a configuration, consisting of the set of allocations in such a way that we can more easily observe the constraints and mechanics of the problem. Namely, we can plot each allocation in the p, q plane (represented by the blue circles in figure 1). For each allocation, l_i, we draw the indifference curve of the agent allocated to it - that is the locus of points (p, q) s.t. $u(p, q;\ l_i) = u(p_i, q_i;\ l_i)$. If any other allocation l_j is inside the convex set

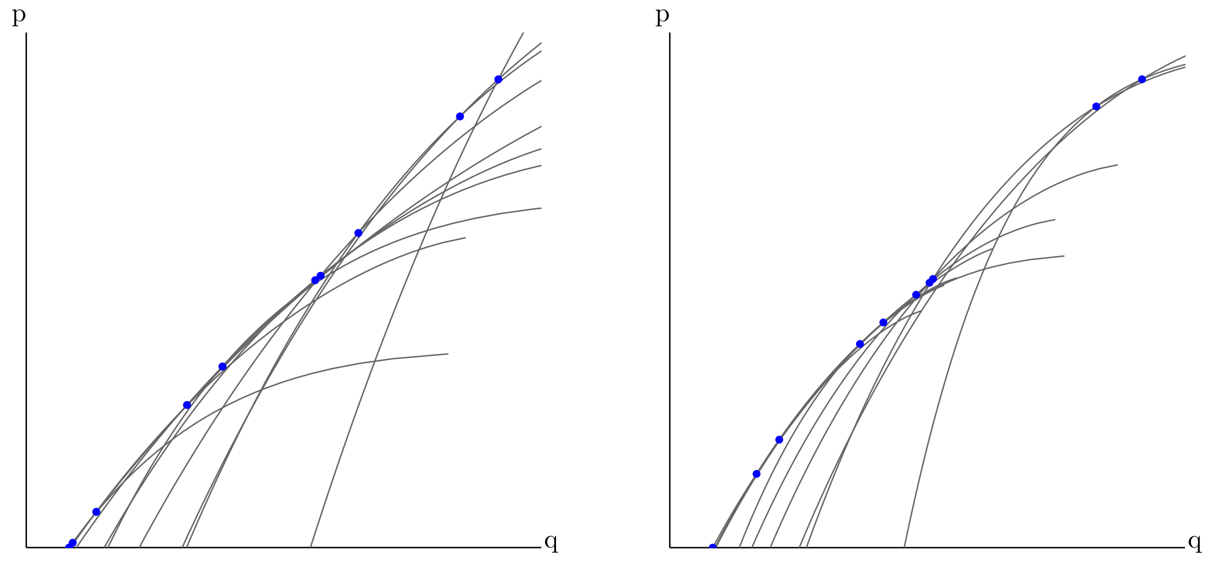

Figure 1: Graphical representation of computationally solved non-envy equilibria configurations with 10 randomly generated items and agents. Left: homogeneous preference, right: heterogeneous preferences.

bounded by indifference curve, that is, below any point along the indifference curve, then we know that we have an invalid allocation, since the agent prefers this other allocation to its own. Figure 1 shows the actual output of the algorithm on a randomly generated set of 10 items with different qualities and 10 agents with heterogeneous preferences and income.

1.3 Equivalent Price

Let the equivalent price, $V_j(u^*, q) : (U_j, Q) \to [0, y_j) \subset \mathbb{R}$ be the price that agent j would have to pay to achieve utility u^* given quality q.

$$V_j(u^*, q) = p \quad \text{s.t. } u_j(p, q) = u^*$$

Similar to utility functions, we can refer to the equivalent price in terms of the allocation:

$$V(u^*, q;\ l_i) = V_j(u^*, q) \quad \text{where } (p_i, a_j) = l_i, \quad l_i \in \hat{L}$$

1.4 Boundary

The boundary of the configuration is the function $B(q) : Q \to \mathbb{R}^+$ that is defined as follows:

$$B(q; \hat{L}) = \max_i V(u(p_i, q_i;\ l_i), q;\ l_i) \text{ s.t. } l_i \in \hat{L}$$

Essentially, this is the lowest price that can be set for each allocation so that no other agent prefers this allocation. Whenever the price of an item is calculated, it will be according to this function, since this gives the minimum price that could be supported while ensuring no other agent would want to switch to this allocation. This can be visualised on the configuration graph as the path traced by the outer most indifference curve for each level of q.

The boundary of the configuration can be also be defined as the function describing set of boundary points of the set constructed from the union of the all convex sets established by the indifference curves of the agents at their respective allocations.

That is, the boundary of the set

$$\bigcup_i \left\{ \begin{bmatrix} q \\ p \end{bmatrix} : u(p, q;\ l_i) > u(p_i, q_i;\ l_i) \right\} \subset \mathbb{R}^2$$

Intuitively, choosing a price for an allocation that lies inside this set will result in an invalid configuration, because some other agent would prefer the allocation. Similarly, setting a price such that the allocation lies outside this set, and not on the boundary, will yield an invalid configuration, since it would be possible to decrease to the price at the boundary. Figure 1 illustrates this - all the dots corresponding to allocations lie precisely on the boundary of the configuration, illustrated by the top most indifference curve at for any value of q.

1.5 The Allocation Function

All equilibrium prices will necessarily be determined relative to the prices of the other items at any given point. This is because these prices act to constrain the price that we can assign to our item such that the allocation remains valid and does not prefer any other allocation. However, when we are deciding on the price of our first item to allocate, there are no such constraints, and therefore we must arbitrarily specify the constraint price. Once we have a constraint price, it is possible to allocate subsequent items given the constraint introduced by this initial allocation. Thus, we have all the information we need to determine a unique minimum price non-envy equilibrium (MPNE) set of allocations, such that no agent prefers another allocation and it is not possible to reduce any of the allocation prices without breaking the equilibrium, given that the constrain price must apply.

An MPNE configuration is a refinement of the general concept of an equilibrium configuration. There may be an infinite number of potential equilibrium configurations in general, even when a constraint price is specified. This is because there is range of prices, $[\underline{p}_i, \bar{p}_i] \subset \mathbb{R}$ that can be set to any given allocation such that it still remains valid. Specifically, the price $\underline{p}_i$ corresponds to the minimum price that can be set for this allocation such that no other agent would prefer it. Conversely, $\bar{p}_i$ corresponds to the maximum price that can be set for this allocation such that the agent allocated does not prefer any other allocation. By imposing the condition that no price associated with any allocation could decrease without causing the allocation to become invalid, we are reducing the set of potential prices for each allocation to just the lowest price, $\underline{p}_i$.

Define the allocation function

$$f_v(\hat{X}, \hat{A}, p_c) : (\mathcal{X}, \mathcal{A}, \mathbb{R}^+) \to \mathcal{L}$$

where $\mathcal{X} \subseteq X,\ \mathcal{A} \subseteq A,\ |\mathcal{X}| = |\mathcal{A}|, \forall a_j,\ p_c < y_j$. This function takes a set of items and agents, and outputs a MPNE configuration. The allocation function takes a constraint price p_c, relative to which all other prices are determined. The behaviour of this function depends on the direction of alignment, denoted by the v subscript, that can take one of $v \in \{u, d\}$.

f_u outputs a MPNE configuration, fixing the price of the lowest quality item $x_{\min}$ to the constraint price p_c. Configurations that are allocated according to f_u are said to be **aligned upwards**. That is, $\hat{L}$ is aligned upwards iff $\hat{L} = f_u(\hat{X}, \hat{A}, p_c)$ where $p_{\min} = p_c$ and $\hat{X}$, $\hat{A}$ are the items and agents allocated in $\hat{L}$.

f_d outputs a MPNE configuration, fixing the price of the highest quality item $x_{\max}$ to the constraint price p_c. Configurations that are allocated according to f_d are said to be **aligned downwards**. That is, $\hat{L}$ is aligned downwards iff $\hat{L} = f_d(\hat{X}, \hat{A}, p_c)$ where $p_{\max} = p_c$ and $\hat{X}$, $\hat{A}$ are the items and agents allocated in $\hat{L}$.

An allocation l_i is **allocated upwards** iff its price is set to make another agent assigned to an allocation of lower quality indifferent to it. Thus,

$$\exists l_j :\ q_j < q_i \text{ s.t. } u(p_j, q_j;\ l_j) = u(p_i, q_i;\ l_j)$$

We say that allocation l_i is allocated upwards from allocation l_j.

An allocation l_i is **allocated downwards** iff its price is set to make another agent assigned to an allocation of higher quality indifferent to it. Thus,

$$\exists l_j :\ q_j > q_i \text{ s.t. } u(p_j, q_j;\ l_j) = u(p_i, q_i;\ l_j)$$

We say that allocation l_i is allocated downwards from allocation l_j.

The difference between upwards and downwards allocation is demonstrated in figure 2.

The constraint allocation set by the constraint price, p_c could either be allocated upwards, downwards or neither for either f_v, since the constraint allocation has its price set exogenously to the function. However, for $|\mathcal{X}| = |\mathcal{A}| > 1$, there must be at least one subsequent agent that is allocated upwards (for f_u) or downwards (for f_d), which the

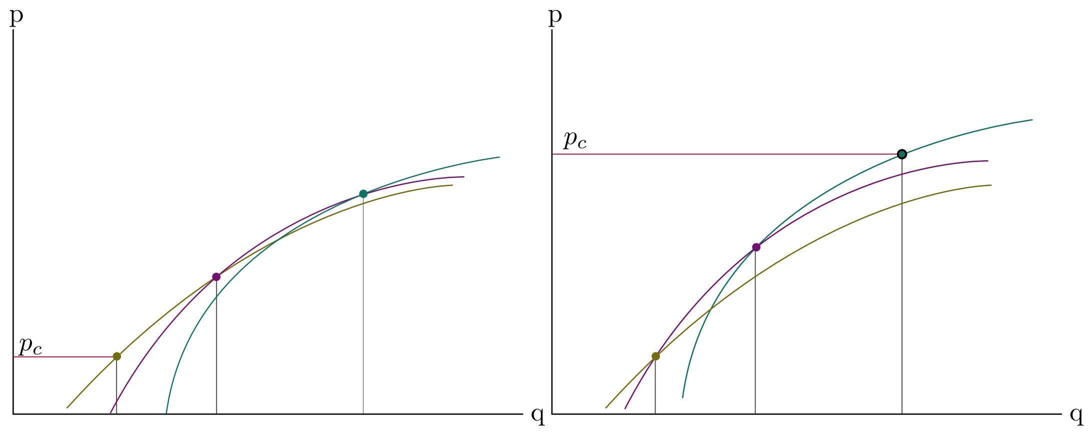

Figure 2: Agents on the left graph are allocated upwards and agents on the right graph are allocated downwards.

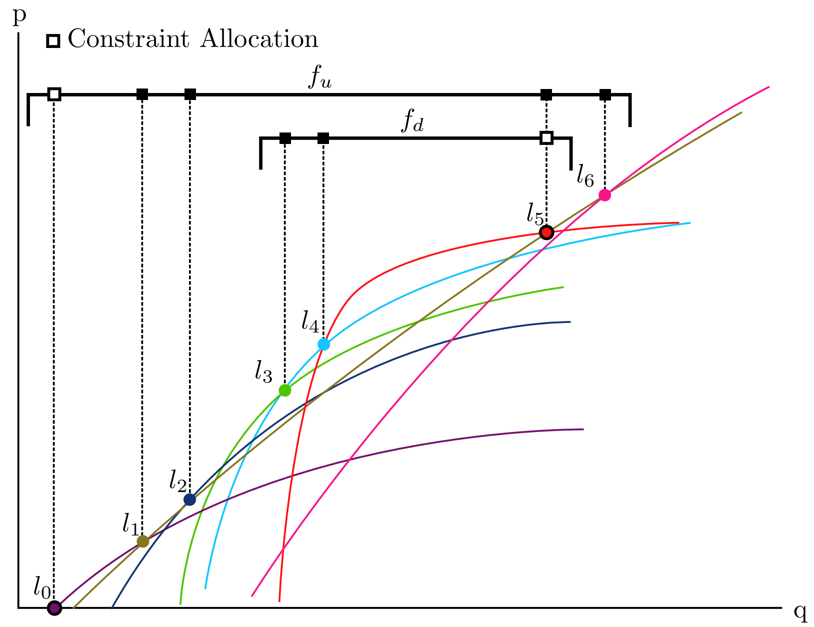

Figure 3: An illustration of how the allocation function, in this case f_u, allocates a subset of its agents in the opposite direction according to f_d.

agent at the constraint allocation will be indifferent to. However, f_u may not allocate all its allocations upwards and f_d may not allocate all its allocations downwards.

The key to understanding the algorithmic solution is the fact that, in certain scenarios, an allocation function f_v may allocate one or more subsets of $\hat{X}$ using the opposite allocation function, f_{-v}. Figure 3 demonstrates this - in order to obtain the MPNE we allocate items $\hat{X}_0 = \{x_0, x_1, x_2, x_5, x_6\}$ and the agents which are assigned to them in equilibrium upwards, according to $f_u(\hat{X}_0, \hat{A}_0, p_c)$. We then use the price p_5 as the constraint price to allocate the items $\hat{X}_1 = \{x_3, x_4, x_5\}$ downwards according to $f_d(\hat{X}_1, \hat{A}_1, p_5)$. The subsets of agents, $\hat{A}_0$ and $\hat{A}_1$ chosen to be assigned to each subset of items are determined algorithmically in such a way that is explored in section 1.7. Notice allocation l_5 may seem like it is both determined by upwards and downwards allocation, but since p_5 is set as the constraint price when allocating downwards, it is purely determined by $f_u(\hat{X}_0, \hat{A}_0, p_c)$.

1.5.1 Price Monotonicity

Proposition 1 (Price Monotonicity) *For any set of allocations $\hat{L} = f_v(\hat{X}, \hat{A}, p_c)$, if we increase (decrease) the constraint price to $p'_c > p_c$ ($p'_c < p_c$) such that $\hat{L}' = f_v(\hat{X}, \hat{A}, p'_c)$, the prices of all allocations will necessarily increase (decrease), so that $p'_i > p_i$ ($p'_i < p_i$) $\forall i$ where $(p'_i, a'_j) = l'_i \in \hat{L}'$. Therefore, the boundary will also increase* *(decrease), such that* $\forall q \in Q$, $B(q;\hat{L}') > B(q;\hat{L})$ $(B(q;\hat{L}') < B(q;\hat{L}))$.

In other words, if we are to increase (reduce) the constraint price of the constraint allocation and allocate all the agents again, then all these new prices will be necessarily higher (lower) than before. This follows from the fact that in order for us to have an MPNE, all but the 'final' agent to be allocated must be indifferent to at least one other agent. Since our first allocation l_0 has a higher (lower) price, then all points along the indifference curve established by l_0 will necessarily have a higher (lower) price, and thus the allocation to which the agent at l_0 is indifferent will be at a higher (lower) price. This process repeats itself for all agents allocated, hence all prices are higher (lower). It important to note that the order of allocations may actually change, but price monotonicity will hold regardless of whether or not agents are allocated to the same items after the change in constraint price.

1.6 Envelopes

In order to determine the equilibrium allocations, and in particular which agent should be assigned to which item, we must define the concept of an **envelope**. Define an envelope to be a set of all the allocations with quality values between the **envelope allocation**, l_e, the farthest away allocation to which the agent at l_e is indifferent - call this the source allocation, l_s. The size of the envelope is the number of allocations in the envelope, which does *not* including the envelope allocation, l_e. The allocations in the envelope between the source and the envelope allocation, are described as being *nested*.

Formally for a upwards envelope:

$$E_u(l_e;\hat{L}) \equiv \{l_i \in \hat{L} : q_e < q_i \le q_s\}$$

where

$$l_s : s = \arg\max_i q_i, \text{ s.t. } u(p_i, q_i;\ l_e) = u(p_e, q_e;\ l_e)$$

and for an downwards envelope:

$$E_d(l_e;\hat{L}) \equiv \{l_i \in \hat{L} : q_e > q_i \ge q_s\}$$

where

$$l_s : s = \arg\min_i q_i, \text{ s.t. } u(p_i, q_i;\ l_e) = u(p_e, q_e;\ l_e)$$

This means that an envelope is an upwards envelope iff the source agent, l_s is allocated upwards from the envelope agent l_e.

Similarly, an envelope is an downwards envelope iff the source agent, l_s is allocated downwards from the envelope agent l_e.

Every agent l_e which is indifferent between its own allocation and some other allocation, has an envelope associated with it, $E(l_e;\hat{L})$. This other allocation that the agent at l_e is indifferent to is the source allocation, l_s. This means that the boundary at the quality value of the source allocation is constructed from the indifference curve of the the agent at the envelope allocation.

For an envelope to be **well-formed**, we require conditions (5) and (6) to hold. The constituent agents must be in a **minimum price local non-envy equilibrium** (MPLNE) state, such that no agent in an allocation within the envelope prefers another allocation *within* the envelope, and it is not possible to reduce any of the allocation prices without breaking the equilibrium. That is, the allocations in the well-formed envelope $E_v(l_e;\hat{L})$ are in a MPLNE state. We also require that no agent outside the envelope prefers an allocation inside the envelope. Combining these, we have:

$$u(p_i, q_i;\ l_i) \ge u(p_j, q_j;\ l_i)\ \forall l_i, l_j \in E_v, j \ne i \tag{5}$$

$$u(p_i, q_i;\ l_i) \ge u(p_j, q_j;\ l_i)\ \forall l_i \in (\hat{L} - E_v), \forall l_j \in E_v \tag{6}$$

An envelope is breached if it is well-formed and at least one constituent agent in an allocation within the envelope prefers the envelope allocation to its own allocation. In this state, the overall configuration is not in equilibrium, the envelope in isolation is (remember, the envelope allocation is not part of the envelope). That is, it must satisfy expressions (5), (6) and (7).

$$\exists l_i \in E_v \text{ s.t. } u(p_e, q_e;\ l_i) > u(p_i, q_i;\ l_i) \tag{7}$$

where p_e and q_e are the price and quality of the envelope allocation respectively.

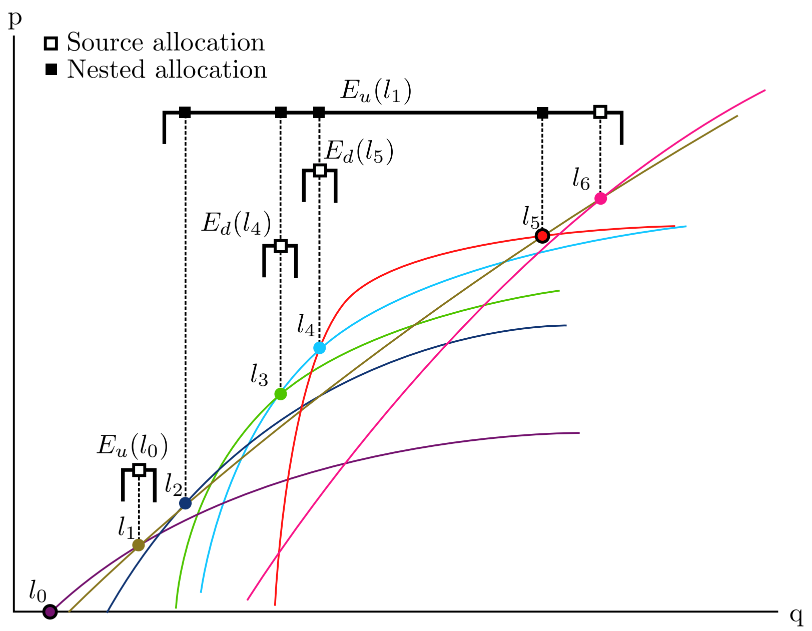

Figure 4: An illustration of all envelopes for an example configuration.

Combining these two concepts, in order for an envelope to be well-formed, we need to ensure that all constituent (sub) envelopes are well-formed and there are no envelope breaches in these sub envelopes.

There are several properties that will hold for envelopes.

(1) Any envelope of size $n > 1$ will always contain envelopes of size no more than $n - 1$. This is important in ensuring that the algorithm will not recurse indefinitely. This follows from the definition of an envelope.

(2) An upwards (downwards) envelope E_v of size $n > 1$ may contain up to $k \leq n - 1$ downwards (upwards) envelopes. If so, these envelopes will be made up of only nested allocations, $E_{-v}(l_m; \hat{L}) \subseteq E_v - \{l_s\}$ if $l_m \in E_v$, where l_s is the source agent of envelope E_v.

(3) If an envelope is breached, then the agent at the envelope allocation, l_e, is not allocated to the correct item. In order to restore the MPNE, this agent should be allocated to the source allocation, l_s, with the agent that was previously allocated to l_s and all the other agents in the envelope, $\hat{A}_{-e} \equiv \{a_j : (p_i, a_j) \in E_v(l_e; \hat{L})\}$ reallocated to items $\hat{X}_{-s} \equiv \{x_e\} \cup \{x_i : l_i \in E_v(l_e; \hat{L})\} - \{x_s\}$, one of which is the old envelope allocation, l_e. These allocations should be allocated according to $f_v(\hat{A}_{-e}, \hat{X}_{-s}, p_e)$, where v matches the direction of the envelope.

The simplest envelope is the trivial envelope of size 1. However, envelopes can be of arbitrary size. Figure 4 shows all the envelopes for the same MPNE configuration shown earlier in figure 3. As can be seen, there can exist envelopes nested in other envelopes.

Figure 5 demonstrates how the envelope of an agent can be restored to MPNE when a new allocation, l_2, is added to the boundary of the configuration. The left diagram shows the state of the configuration as l_2 is added. Since it sits on the boundary of allocation l_0, it acts as the source allocation for the envelope $E_u(l_0; \hat{L})$, where the envelope allocation, $l_e = l_0$. So the envelope is $E_u(l_0; \hat{L}) = \{l_1, l_2\}$, with size 2. By increasing the price of l_1 to the level where the agent at l_2 is indifferent, we can restore the envelope to MPLNE, and in this case, the equilibrium of the configuration. By increasing the price of l_1 in this way, we are actually allocating downwards from the source allocation l_2 (which is in the opposite direction of the upwards allocation of the prior configuration).

Figure 6 shows a similar scenario, however this time we have a breach of the envelope. So by increasing the price of l_1 to make l_2 indifferent, the agent at l_1 now prefers the envelope allocation, l_0. In this case, we restore equilibrium by promoting the agent at l_0 to allocation l_2 and allocating all the agents as shown in figure 7.

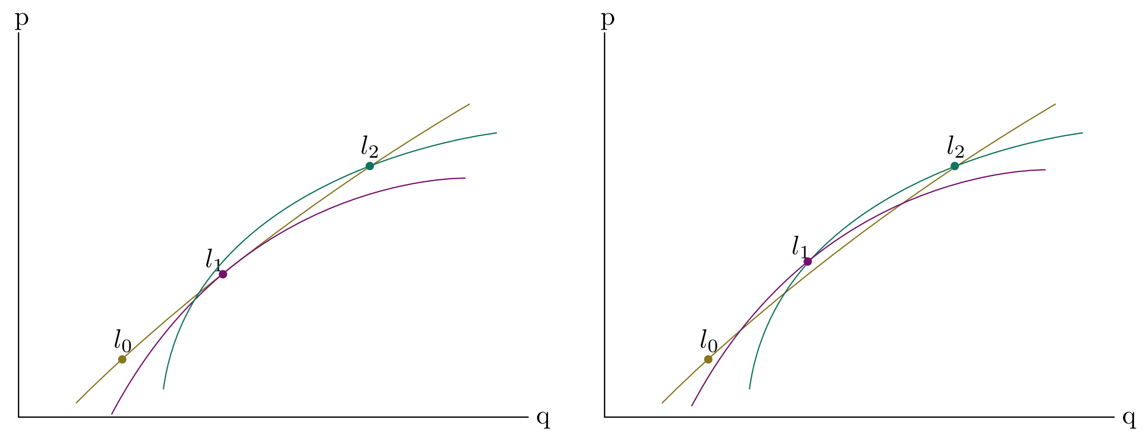

Figure 5: Allocation l_2 is added to the configuration and the envelope is restored.

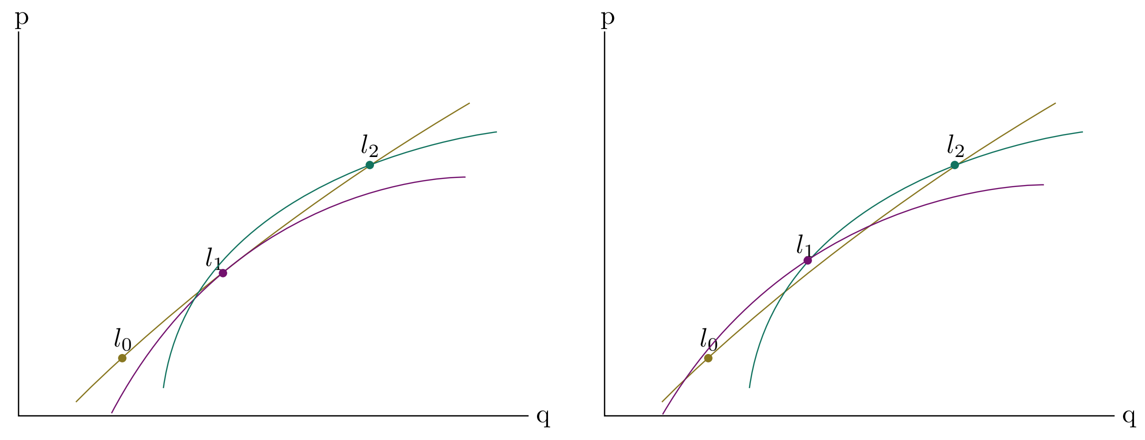

Figure 6: Envelope is breached.

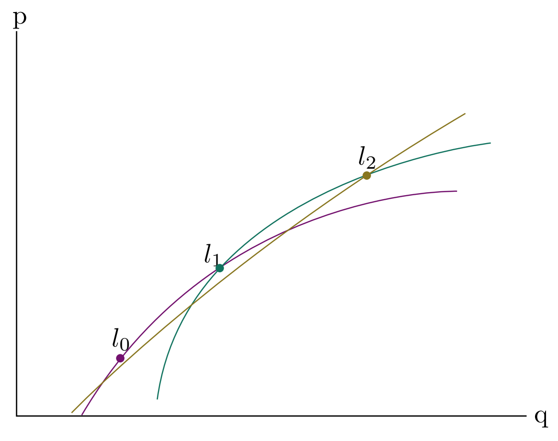

Figure 7: Agent at l_0 is promoted to l_2, restoring equilibrium.

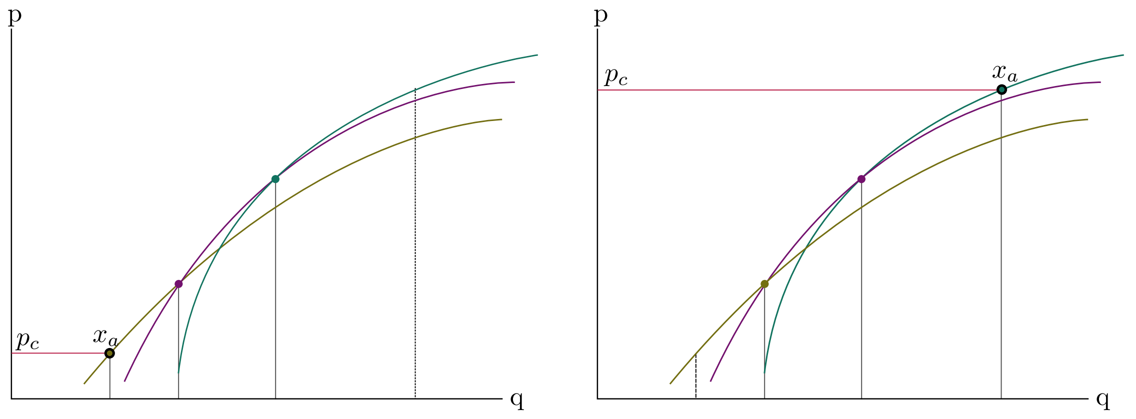

Figure 8: Illustration of shifting equivalence, left: after downward shift, right: after upward shift.

1.6.1 Shifting Equivalence

Now define $\hat{X}_{-1} \equiv \{x_a\} \cup \hat{X} - \{x_b\}$, $x_a \in X$, $x_b \in \hat{X}$ s.t. $q_a < q_i \forall x_i \in \hat{X}$, $q_b \geq q_j \forall x_j \in \hat{X}$. Essentially, we can consider $\hat{X}_{-1}$ to be the range of items $\hat{X}$ shifted down one, so we include the next item with lower quality but exclude the item with the highest quality.

Similarly define $\hat{X}_{+1} \equiv \{x_b\} \cup \hat{X} - \{x_a\}$, $x_b \in X$, $x_a \in \hat{X}$ s.t. $q_b > q_i \forall x_i \in \hat{X}$, $q_a \leq q_j \forall x_j \in \hat{X}$, which follows the same principle but shifted up one instead of down one.

Proposition 2 (Shifting Equivalence) *The boundary $B(q; \hat{L}_d)$ of any configuration $\hat{L}_d$ aligned downwards, allocating the set of agents $\hat{A} \subseteq A$ to items $\hat{X} \subseteq X$, where $\hat{L}_d = f_d(\hat{X}, \hat{A}, p_b)$ is identical to the boundary of the set of agents $\hat{X}_{-1}$, $f_u(\hat{X}_{-1}, \hat{A}, p_a)$ if $p_a = B(q_a; \hat{L}_d)$, where $\{x_a\} = \hat{X}_{-1} - \hat{X}$.*

By symmetry, the boundary $B(q; \hat{L}_u)$ of any configuration $\hat{L}_u$ aligned upwards, allocating the set of agents $\hat{A} \subseteq A$ to items $\hat{X} \subseteq X$, where $\hat{L}_u = f_u(\hat{X}, \hat{A}, p_a)$ is identical to the boundary of the set of agents $\hat{X}_{+1}$, $f_d(\hat{X}_{+1}, \hat{A}, p_b)$ if $p_b = B(q_b; \hat{L}_u)$, where $\{x_b\} = \hat{X}_{+1} - \hat{X}$.

To see why, we can consider the first case, with a downward allocation. As item x_b is now removed, the agent allocated to this item, a_i, must move to a different item. However, we know, by the definition of a downwards allocation and an envelope, that the agent a_i must be indifferent to the source agent of its envelope. In most cases (where the envelope has size $n = 1$), this will simply be the next item down. We move the agent to the source allocation of its envelope, displacing the next agent, and repeat this process for each subsequently freed agent until we reach x_a. Since we have kept every agent indifferent to its adjacent allocation (with lower quality) at all times, the boundary must remain the same.

Figure 8 demonstrates this shifting equivalence for a simple case with four items in total, one item excluded in each case to give $\hat{X}_{-1}$ allocated upwards for the left graph and $\hat{X}_{+1}$ allocated downwards for the right graph. Each agent corresponds to a different colour, and the allocation dot colour signifies which agent is assigned to each allocation. The indifference curves are also coloured according to the agent whose preferences they represent. As we can see the colours of the allocation dots shift, which shows how the agents are moved down/up. This demonstrates how shifting equivalence one way effectively implies shifting equivalence the other way - each one is just the shifted equivalent of the other. The key observation here is that the boundary is identical in both cases.

Figure 9 illustrates shifting equivalence based on the configuration in figure 3, which has non-trivial envelopes and allocations allocated upwards and downwards. Notice how nested allocations *do not change*, between the configurations. This is because the indifference curve of the agent in the envelope allocation crosses back over the to form the boundary of the configuration for values of q beyond the nested allocations in the envelope, which means that these nested allocations are not relevant in forming the boundary of the configuration *outside of the envelope*. Essentially, we can consider the nested allocations in an envelope to be an isolated group of allocations that are not relevant in determining the prices of allocations outside of the envelope (so long as the envelope is well-formed and is not breached).

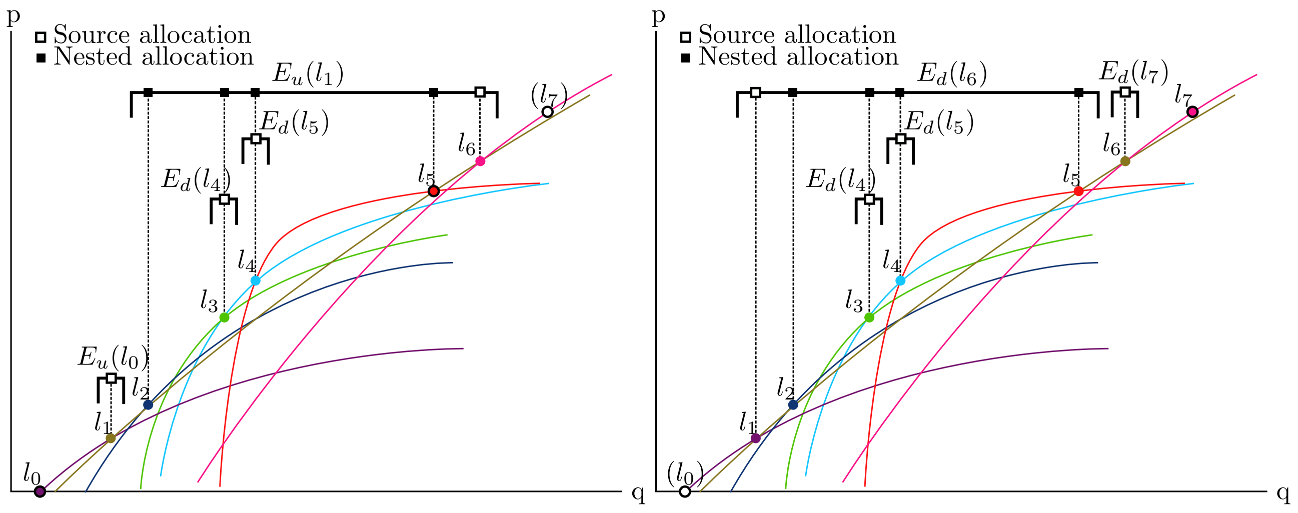

Figure 9: Illustration of shifting equivalence for a more complex configuration.

1.6.2 Proof of Property (3)

Combining shifting equivalence with price monotonicity allows us to prove property (3). Suppose that the envelope $E_v(l_e;\hat{L})$ is breached. In order for this to happen, the agents in the envelope must be the result of an MPNE allocation over those agents and items in the direction $-v$, implying that $E_v(l_e;\hat{L}) = f_{-v}(\hat{X}_e, \hat{A}_e, p_s)$, where $\hat{X}_e$ is the set items of the allocations in the envelope, and $\hat{A}_e$ is the set of agents allocated to those items. This is because, if there is an envelope breach, the boundary must be above the indifference curve of the agent at the envelope allocation for all items in the envelope if it to also prefer the envelope allocation. Thus, we can use shifting equivalence to show that, if we include x_e (equivalent to x_a in the definition of shifting equivalence) and remove x_s (equivalent to x_b in the definition of shifting equivalence) where $(p_s, a_j) \in l_s$ to give $\hat{X}_{-1} = \{x_e, \cdots, x_{s-1}\}$, then by allocating these agents in direction v on $\hat{X}_{-1}$, and setting the new constraint price p'_e as the boundary of the envelope at q_e, $B(q_e;\hat{E}_u)$, then the boundary stays the same. Thus, we would allocate according to $\hat{L}_e = f_v(\hat{X}_{-1}, \hat{A}_e, p'_e)$ Note that at this point $p'_e > p_e$, as implied by the fact that we have an envelope breach.

However, we cannot maintain this price $p'_e > p_e$ because it would not be MPNE. This is because we could reduce the price of all allocations in $\hat{L}_e$ to the point where the price of the old envelope allocation is p_e once again. By price monotonicity, if we reduce p'_e back to p_e, according to $f_v(\hat{X}_{-1}, \hat{A}_e, p_e)$ then the boundary $\hat{L}_e$ will reduce, implying that $p'_s < p_s$. Since $p'_s < p_s$ and $u(p_s, q_s;\ l_e) = u(p_e, q_e;\ l_e)$ then $u(p'_s, q_s;\ l_e) > u(p_e, q_e;\ l_e)$, and the agent at the old envelope allocation will prefer the old source allocation at the new price p'_s over its old allocation, hence it should take item x_s at the price $B(q_s;\hat{L})$.

1.6.3 Intuition for Envelopes

Fundamentally, envelopes are useful as they allow us to determine whether or not an agent is allocated to the correct allocation, and if not, where to move the agent. In the case of homogeneous preferences, all envelopes will be trivial (of size $n = 1$). This is because each allocation will always lie on the boundary formed by the indifference curve of an adjacent allocation, due to the fact that all indifference curves is simply a translations of one another. Determining whether agents should change places upon adding a new allocation to the configuration is simply a comparison between adjacent allocations, and if the agent in our new allocation prefers the adjacent allocation we know to swap them (assuming that the new allocation lies on the boundary of the old configuration). However, envelopes generalise the principle of this comparison between allocations for configurations with heterogeneous preferences, where we must determine whether the interaction of some combination of allocations (the allocations within the envelope), each with differently shaped indifference curves implies that an agent (the agent at the envelope allocation) should be moved to a different allocation (the source allocation), in the event of an envelope breach. In the context of developing an algorithm, this is crucial to giving us a mechanism via which to consistently reach the desired MPNE configuration.

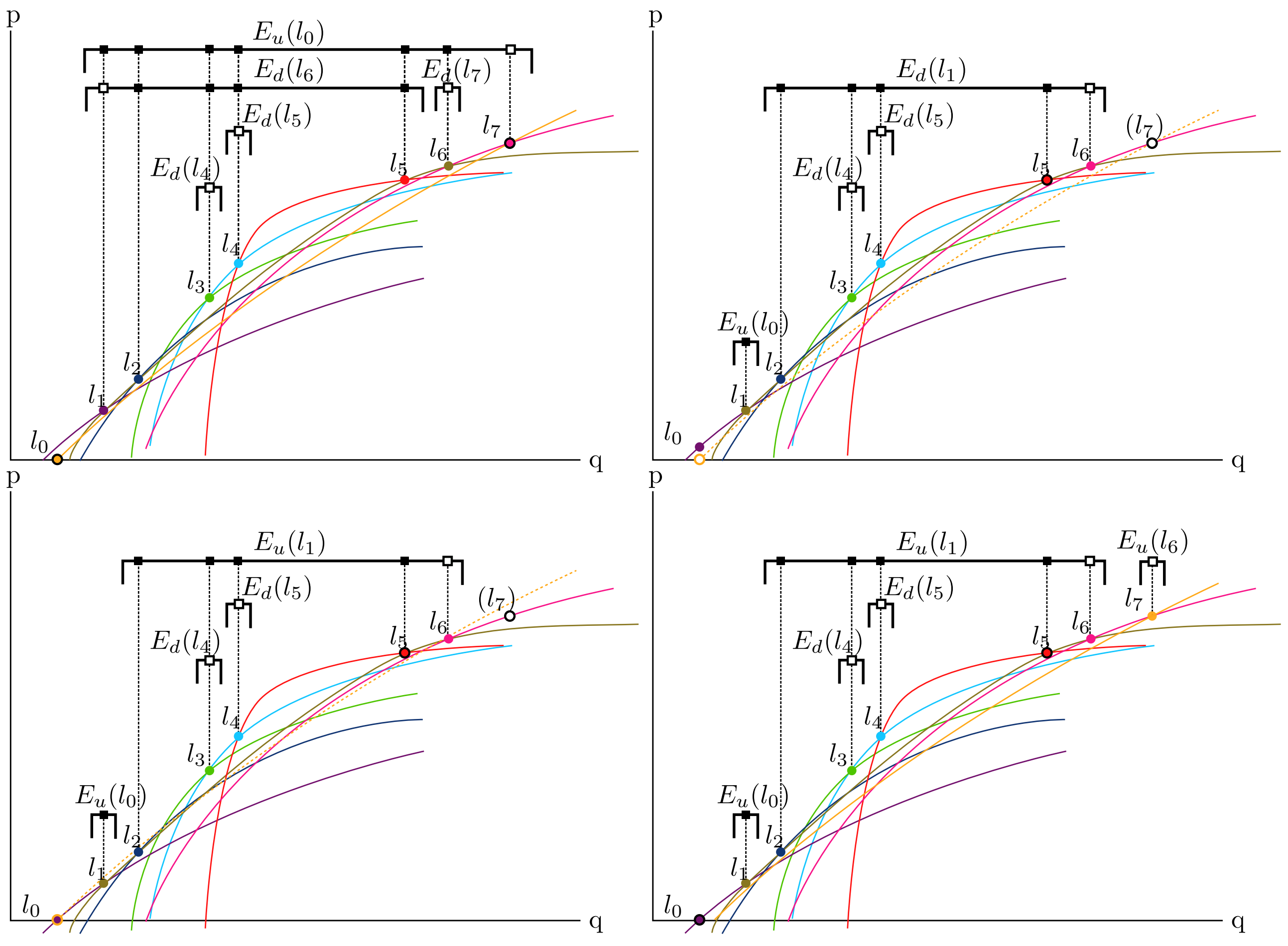

Figure 10: Illustration of property (3) for a more complex configuration. Top left: allocation l_7 added and the envelope $E_u(l_0)$ is restored so that it is well-formed, but breached (agent at l_1 prefers l_0). Top right: shifting equivalence is used to allocate all agents excluding the agent at the envelope allocation (dashed line). Bottom left: price monotonicity implies that the boundary will strictly decrease if the constraint price p_0 is decreased - l_7 is now inside the dashed indifference curve of the initial agent at l_0. Bottom right: the old agent at l_0 is shifted to l_7, since we know that it must prefer this allocation to l_0 by property (3).

1.7 The Allocation Algorithm

The allocation algorithm starts with a pool of agents, $\hat{A}$ and a configuration where no agents are assigned to any items. This means that we allow allocations to be temporarily `null`, that is we may have $l_i = \emptyset$. Define a **partial minimum price non-envy equilibrium** (PMPNE) configuration to be a configuration in which all allocations are either valid or `null`.

Agents are initially chosen for allocation according to the `next_agent` function, which determines which agent from the pool of unallocated agents, $\hat{A}$, should be initially assigned to an allocation, and determines a price for the allocation which is on the current boundary of $\hat{L}_0$. This function takes the existing configuration, $\hat{L}_0$, the pool of agents from which to choose, $\hat{A}$, the index of the allocation to allocate an agent to, i, and the direction of allocation, v. This function also returns the index of allocation, e, who's agent forms the boundary at $B(q; \hat{L})$.

The agent to be allocated is chosen to minimise the boundary in the direction of allocation. We first choose the price to allocate the agent to according to the existing boundary of the configuration. So $p_i = B(q_i; \hat{L})$.

- For $v = u$, choose the agent that minimises the equivalent price at the quality value of the *next* item (so allocation $i + 1$).
$$\arg\min_{a_j} V_j(u_j(p_i, q_i), q_{i+1}),\ \ a_j \in \hat{A}$$
- For $v = d$, choose the agent that minimises the boundary at the quality value of the *previous* item (so allocation $i - 1$).
$$\arg\min_{a_j} V_j(u_j(p_i, q_i), q_{i-1}),\ \ a_j \in \hat{A}$$

In the case of homogeneous preferences, stepping through each allocation sequentially and assigning an agent using the `next_agent` function is sufficient to produce a MPNE configuration. This is because, in the case of homogeneous preference, the boundary at any allocation is always constructed by the agent at the adjacent allocation, hence all envelopes will be trivial and we will never have an envelope breach due to how we choose the agents. We can prove this by contradiction. Suppose the agent at l_{i+1} is allocated on the boundary prefers the allocation l_i. We clearly have a contradiction as the agent allocated to l_i cannot have been the agent to minimise the boundary - the agent at l_{i+1} would have established a lower boundary if it were allocated to l_i.

However, as previously alluded to, when we generalise to heterogeneous preferences, an allocation can sit on the boundary constructed by an agent at an allocation *not* adjacent to it (see figure 5). This is because heterogeneous preferences can result in the existence of indifference curves with different degrees of curvature for the same level of consumption and quality. Therefore, the proof for the homogeneous case does not apply in all cases, because `next_agent` only chooses an agent to minimise its own equivalent price for the quality value of the next item, which may not be equal to the boundary of the configuration for that value. Even so, choosing an agent this way still maximises the chances of choosing the correct agent for that allocation and therefore not having to reallocate.

At its core, the allocation algorithm works via a single, recursive function - the `allocate` function. This function takes four parameters, the current configuration, $\hat{L}$, the index of the potentially invalid allocation to allocate, i, the index of the allocation in the configuration whose agent is indifferent to allocation i, e, and the direction of allocation, $v = \{u, d\}$.

The function assumes that the $\hat{L}$ provided is in a PMPNE state (if allocation l_i is excluded from the configuration) and modifies the configuration, to ensure that the entire configuration, including allocation l_i is a PMPNE configuration. It does this by checking to see if l_i is invalid, and if so, changing the configuration to bring about a PMPNE state.

The `allocate` function does not set any previously non-`null` allocations to be `null` in its output. This means that invoking `allocate` with a MPNE configuration will also output a MPNE configuration (so no `null` allocations). The only time that `null` is introduced is when the function is called recursively, and the allocations that were set to `null` will be restored after the recursive call has completed.

The `allocate` function performs the following steps:

(1) **Check if allocation l_i is already valid.**

If allocation l_i is already valid, then nothing needs to be done, as we know that $\hat{L}$ is PMPNE when excluding l_i, so if l_i is also valid then $\hat{L}$ is already PMPNE.

If allocation l_i is not valid, it must be that the agent allocated to l_i prefers some other allocation. In this case we need to first ensure that the envelope for which l_i is the source agent $E_v(l_e; \hat{L})$ is well-formed, so we move to step (2). The left graph in figure 5 provides an example of this scenario.

(2) **Restore the envelope to a well-formed state.**

The envelope is determined by the parameter e, which gives the index of the allocation whose agent is indifferent to l_i. This fulfils the definition of an envelope allocation, which means that l_i must be the source allocation of the envelope $E_v(l_e; \hat{L})$, where v is the parameter of the function.

However, in its the current state, the envelope may not be well-formed (if the envelope is non trivial, of size $n > 1$), because l_i is part of the envelope and may not be valid.

To restore the envelope to a well-formed state, the following procedure is performed: Start with $t = 1$,

(a) Remove the agents and items from allocations $\hat{L}_t = \{l_{i-t}, \cdots, l_{i-1}\}$ for $v = u$ or $\hat{L}_t = \{l_{i+1}, \cdots, l_{i+t}\}$ for $v = d$, and set all these allocations to `null`. Place these agents into $\hat{A}_t$.

(b) Allocate agents in $\hat{A}_t$ in the opposite direction, $-v$, to the same set of (now `null`) allocations, $\hat{L}_t$. Do this for each allocation sequentially, starting at the allocation closest to the source allocation: $l_{i-1}, \cdots, l_{i-t}$ for $v = u$ and $l_{i+1}, \cdots, l_{i+t}$ for $v = d$. For each agent to allocate, the agent is chosen according to the `next_agent` function in the (opposite) direction $-v$, from the pool of $\hat{A}_t$, using the entire configuration $\hat{L}$ to determine the boundary. Each time a new agent is allocated call the `allocate` function (recursively) on the current configuration with `null` allocations, $\hat{L}$, specifying the allocation that was just added for the i parameter, and the relevant boundary allocation for the e parameter. Essentially, this will allocate agents to items such that $\hat{L}_t = f_{-v}(\hat{X}_t.\hat{A}_t, p_i)$.

By calling the `allocate` function for each agent reallocated, we ensure that all the sub envelopes are well-formed and not breached, and if they are breached, are properly dealt with.

(c) Once complete, if any of the agents in the newly allocated allocations, $\hat{L}_t$, prefer any of the other agents in the non-reallocated nested allocations, then increment t and repeat. If not, then we have restored the envelope to MPLNE, and so our envelope is well formed.

(3) **Handle a potential envelope breach.**

Once the envelope is restored, check to see if any of the agents at the allocations in the envelope prefer the envelope allocation, l_e. If they do, then we have an envelope breach. If not, $\hat{L}$ is in a PMPNE state and there is no more to do.

If we have an envelope breach, then we remove all the agents $\hat{A}_r$ from the allocations $E_v(l_e; \hat{L})$, and we also remove the agent a_e from the envelope allocation l_e. We then sequentially reallocate agents in $\hat{A}_r$, starting with the old envelope allocation: $l_e, \cdots, l_{i-1}$ for $v = u$ or $l_e, \cdots, l_{i+1}$ for $v = d$. For each agent allocate according to the `next_agent` function, with direction v. The `allocate` function is then run recursively, with i set to the index of the allocation just added, ensuring its validity, similar to in (2)b.

Finally, the agent in the old envelope allocation l_e is allocated to the source allocation, which it l_i. The price is chosen according to $B(q_i; \hat{L})$, where $\hat{L}$ refers to the configuration modified so far by the function. Similar to when any other allocation is changed, `allocate` is (recursively) run on i again, but this time with this updated configuration.

Finally, define the `root` function that takes the complete set of agents, A, the complete set of items, X, and a constraint price of the lowest quality item, p_0, and produces the MPNE configuration equivalent to $f_u(X, A; p_0)$. This function starts with a configuration where all allocations are null, and adds the first allocation with price set to the constraint price, p_0, and with the agent chosen according to the `next_agent` function. Agents are then added to each subsequent allocation for $i = \{1, 2, \cdots, n-1\}$ sequentially, again with agents chosen according to the `next_agent` function. Each time an agent is added, the `allocate` function is called, which ensures that the configuration is PMPNE at each step. Finally, once all agents are added, we know we must have a MPNE configuration, because we have assigned an agent to every item.

Since all envelopes of size $n > 1$ must contain envelopes of size no greater than $n - 1$, we will always reach a nested envelope of size 1 when recursively calling the `allocate` function. Envelopes of size 1 do not require any reallocation of nested allocations, so the recursion via the allocation of nested envelope agents must come to an end.

Likewise, the recursion due to an envelope breach (which can occur for $n = 1$, where the next agent prefers the previous), will also come to an end, because once we shift up the agent at the envelope allocation, we know that it will always necessarily prefer this new ordering, and thus will never prefer to switch all the way back down. As there are a finite number of orderings, then the algorithm will eventually reach a point where all incorrect orderings are corrected, and thus recursion due to the agent shifting will always terminate.

It is important to note that when the agent at an envelope allocation is moved to the source allocation due to an envelope breach, this does not necessarily mean that the allocation that the agent was promoted to will be the final allocation for that agent. Although unusual, it could be the case that the order of agents in the old nested allocations change when allocating in the new direction from the old envelope allocation. As a result, once allocated to its new position, this agent may prefer an allocation previously in its envelope (but it will never prefer its old allocation, or any allocation outside its old envelope). This will still be handled appropriately by the algorithm, because `allocate` is run again on all changed allocations (which ensures PMPNE after every invocation). The key point is that even if a promoting an envelope agent is not always valid immediately, the promoted configuration is 'closer' to a valid state, and will always reach an equilibrium by the recursive invocation of the `allocate` function for each new allocation.

Finally, we must also consider the potential for an agent, j, to be allocated to an allocation i with a price exceeding its budget constraint $p_i > y_j$. For heterogeneous preference, this should never happen. This is a result of the asymptotic MRS constraint given in expression (3). However, for homogeneous preference, due to the fact that the boundary at the point of allocation could be constructed from the indifference curve of an agent at a non-adjacent allocation, the budget constraint may be invalid for a newly allocated agent. It may also happen as a result of allocating a set of agents in the opposite direction when restoring an envelope, despite it having a valid price initially. Clearly this price cannot be valid. In all cases, we must have an envelope breach, since we know that the boundary created by this agent must be below its income *for any quality*, due to the asymptotic marginal rate of substitution condition, expressed in expression (3). Therefore we know that if we allocate this agent at any price below its income, *all* points along its indifference curve will be less than its income (and therefore less than the original, invalid price), thus shifting the entire boundary by price monotonicity. This situation can be interpreted as a stronger version of the standard envelope breach.

A Asymptotic MRS Requirement

To see why we require expression (3) to hold, let us consider a counterexample, where this property does not hold. Suppose that we have two agents $A \equiv \{a_0, a_1\}$, both with identical incomes, $y_0 = y_1 = y$. Also suppose that property (3) does not hold for the utility functions corresponding to the preferences of these agents. We also have two items $X = \{x_0, x_1\}$ where q_0 and q_1 are sufficiently far apart for there to be no solution. Specifically,

$$q_1 > \max_i q : u(y, q;\ l_i) = u(p_c, q_0;\ l_i)$$

Given the constraint price $p_c = 0$, we are constrained by our price for allocation l_0. As shown in figure 12, we can choose utility functions v_0 and v_1 (that do not adhere to expression (3)) such that no matter which agent we allocate at l_0, we cannot choose a price $p_1 \leq y$ within the budget constraint of the agent allocated at l_1 that satisfies the conditions for the allocation to be valid. This is because, for any price that we choose, within the budget constraint of the agent allocated to l_1, the agent allocated to l_0 will strictly prefer l_1. Since both agents have the same budget constraint, it impossible to resolve this by setting the price p_1 such that the agent allocated at l_0 cannot afford l_1.

B Details of Computational Implementation

The algorithm was implemented in the Rust programming language as it is highly performant while also providing strict safety guarantees. Furthermore, the nature of the problem allowed Rust's move semantics to be leveraged effectively. Essentially, by not requiring the `Copy` trait for the agent type, we can guarantee that we cannot allocating the same agent to multiple items without explicitly calling `clone` on the object (the code would not compile if it were attempted).

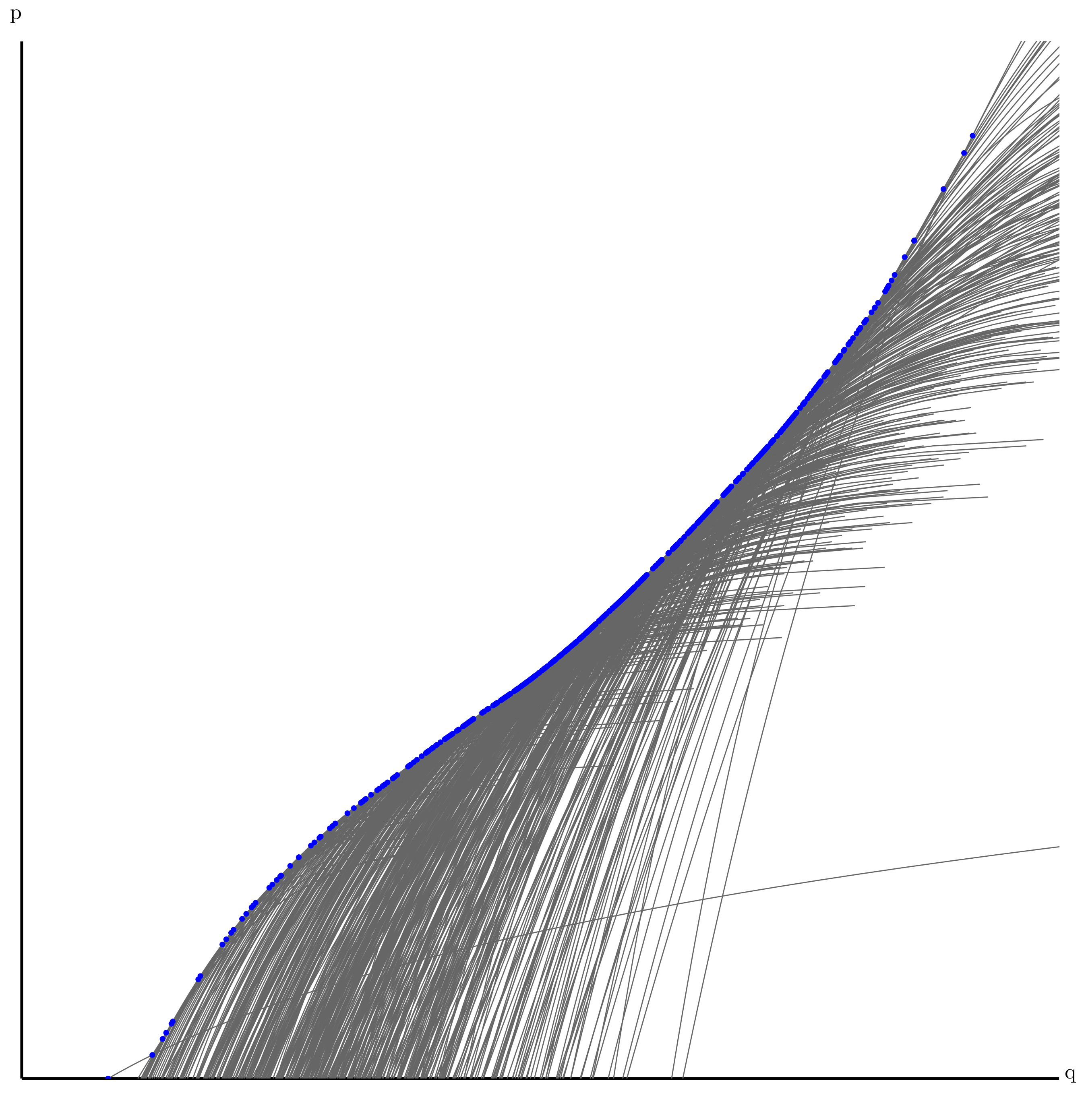

Figure 11: Graph of MPNE for 500 items and agents.

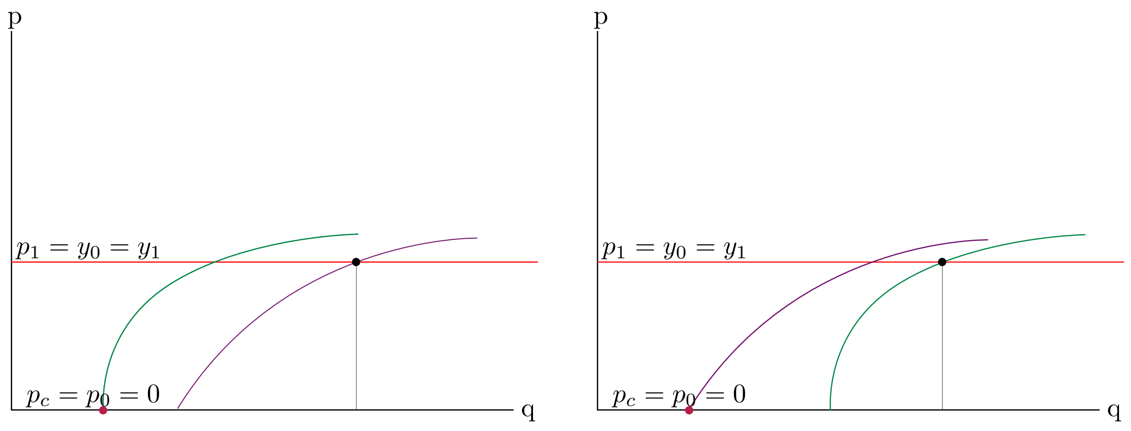

Figure 12: Counterexample configuration demonstrating why property (3) must hold for a MPNE to always exist.

The implementation is also type agnostic with regards to which floating point type it uses, thus, arbitrary precision floating point types can be used if a very high level of precision is required. It does this by using a generic floating point type in the core of the algorithm, once more leveraging one of Rust's powerful features.

The functions defined in the code correspond closely with their mathematical counterparts presented here. However, due to the finite nature of computational precision, there is a tolerance level established for comparisons between floating point values. Specifically, for the implementation of the equivalent price price function, $V_j(u^*, q)$, numerical methods are used to solve for arbitrary black box utility functions. Any arbitrary black-box utility function can be used with this implementation, because the solution of the equivalent price function (which relies on the inverse of the utility function) relies on numerical methods, rather than using some pre-specified closed form representation.

Computationally, the rough theoretical upper bound on time complexity is $O(n^3)$, but in practice the algorithm runs much faster than this, due to the fact that usually, when a new allocation is added, it is valid straight away and does not require restoring envelopes or cause envelope breaches.