An algorithm to solve combinatorial discrete choice problems over an interval

Implemented in Julia

Rowan Shi

February 5, 2019

Contents

1 Introduction

This document describes the `cdc_solver.jl` module, which implements an algorithm to solve combinatorial discrete choice optimisation in Julia. It is heavily inspired by the theoretical work by Arkolakis & Eckert (2017).

1.1 A framing example

To better interpret the formalisation below, consider the problem which inspired this module. An economic model describes (potentially) multi-national firms deciding in which countries to place affiliates. The firms choose this set of countries to maximise expected profits. An extra affiliate provides benefits to the firm – for example, more seamless access to the country's consumers. However, it is also costly to open. In addition, the locations of the firm's other affiliates could affect the new affiliate's potential value – for example, it would decrease if the firm chooses to have an affiliate in a nearby location with adequate access to the country's consumers. For this reason, each location

cannot be evaluated independently; instead, the firm must optimise over the *power set* of affiliate locations – that is, it must choose a set of locations to place its affiliates. This choice set represents the "combinatorial discrete choice" (CDC).

The key parameter in the model is firm productivity, which affects expected profits positively. Therefore, the central question in the model is how the optimal choice of affiliate locations vary with firm productivity. In particular, it is intuitive that more productive firms may choose to place more affiliates (larger optimal sets) or affiliates that are high-cost, high-value.

To this end, the algorithm is designed not to solve the CDC problem for a specific firm productivity, but the return the optimal policy as a function of firm productivity.

1.2 What the algorithm solves

The algorithm solves maximisation problems where the objective function has the form $f(\cdot;\theta) : \mathcal{P}(J) \to \mathbb{R}$, and is optimised over its first argument. In other words, the problem is to select a subset $\mathcal{J} \subseteq J$, which maximises f. The second argument θ is a parameter governing the level of f. The algorithm calculates $\mathcal{J}(\theta)$, the optimal $\mathcal{J}$ for each level of the parameter θ, under the following assumptions:

- $f(\mathcal{J}, \cdot\,)$ is weakly increasing in θ for all $\mathcal{J}$
- if $D_j(\mathcal{J},\theta) = f(\mathcal{J} \cup \{j\},\theta) - f(\mathcal{J}\backslash\{j\},\theta)$ for $j \in J$,
  - $D_j(\mathcal{J}_1,\theta) \geq D_j(\mathcal{J}_2,\theta)$ if $\mathcal{J}_1 \subseteq \mathcal{J}_2$
  - $D_j(\mathcal{J},\theta)$ is increasing in φ
- for each $\mathcal{J}$, there is a unique θ solving $D_j(\mathcal{J},\theta) = 0$
- for any pair $\mathcal{J}_1 \neq \mathcal{J}_2$, there is a unique θ solving $f(\mathcal{J}_1,\theta) = f(\mathcal{J}_2,\theta)$

1.3 The assumptions

The first assumption characterises the objective function's first derivative. In the baseline firm example, this assumption corresponds to the intuition that expected profits increase as firm productivity increases.

The second assumption can be loosely interpreted as governing the objective function's second derivatives. D_j captures the marginal value of adding location j, evaluated at $(\mathcal{J},\theta)$ – a representation of f's first derivative along $\mathcal{J}$. Then, the first part of the assumption asserts that this marginal value decreases as $\mathcal{J}$ grows, holding θ constant – in effect, that the second derivative along $\mathcal{J}$ is negative. For the multi-national firm, if the value of an additional affiliate comes from proximity to consumers, then this value would be the largest when the firm has no other affiliates. It would then decrease as the firm considers larger and larger sets of affiliates in the area.

The second part of the assumption concerns the "cross-partial" of f – the marginal value of adding any location j to any $\mathcal{J}$ increases in θ. Cast in the multi-national firm

model, not only does firm productivity increase expected profits – it also increases the expected value of adding an additional affiliate.

The third and fourth assumptions are regularity assumptions on the objective function. Since $D_j(\mathcal{J}, \theta)$ is monotonic in θ, the solution to $D_j(\mathcal{J}, \theta) = 0$ is unique if it exists – the assumption ensures existence.

2 Using the module

The module exposes one function: `solve_cdc`, which accepts as named arguments:

- `N`, the cardinality of J
- `obj`, the objective function f, with positional arguments `(J, prod)` where
  - `J` is a `N`-length boolean array representing the set $\mathcal{J}$
  - `prod` is a scalar representing θ
- `equalise_obj`, a function with positional arguments `(J1, J2)`, which returns θ solving $f(\mathcal{J}_1, \theta) = f(\mathcal{J}_2, \theta)$
- `zero_D_j_obj`, a function with positional arguments `(J, j)` (where `j` is an integer between `1` and `N` indexing j), which returns θ solving $D_j(\mathcal{J}, \theta) = 0$

Then, `solve_cdc` runs the algorithm and returns a 2-tuple containing

- an `M`-length array, with each element a `N`-length boolean array
- an `(M+1)`-length vector of scalar breakpoints, dividing the range of θ

Suppose the results are stored in `opt_J` and `breakpoints`, respectively. Then, `opt_J[m]` is the optimal choice for values of θ in the interval `(breakpoints[m], breakpoints[m+1])`.

3 The algorithm

This section details the algorithm and its theoretic grounding. A summary of the algorithm which solves the CDC for a specific value of θ described in Arkolakis & Eckert (2017) is provided first. Next, its logic is extended to solve the problem for a range of θ.

3.1 Arkolakis & Eckert (2017): solving the CDC for a single value of θ

The logic of the algorithm builds on the work of Arkolakis & Eckert (2017), which develops a theoretically-justified algorithm to solve the CDC problem for a specific value of θ. In this section, θ is therefore dropped for notational simplicity.

Their method consists of an inner loop and an outer loop. The inner loop iterates on a pair $(\underline{\mathcal{J}}_i, \overline{\mathcal{J}}_i)$, where $\underline{\mathcal{J}}_i \subseteq \overline{\mathcal{J}}_i$:

1. define the set $\text{maybe}_i = \overline{\mathcal{J}}_i \backslash \underline{\mathcal{J}}_i$

2. calculate

$$\begin{aligned} \text{in}_i &= \{j \in \text{maybe} \quad | \quad D_j(\overline{\mathcal{J}}_i) > 0\} \\ \text{out}_i &= \{j \in \text{maybe} \quad | \quad D_j(\underline{\mathcal{J}}_i) \leq 0\} \end{aligned}$$

3. update $\underline{\mathcal{J}}_{i+1} = \underline{\mathcal{J}}_i \cup \text{in}_i$ and $\overline{\mathcal{J}}_{i+1} = \overline{\mathcal{J}}_i \backslash \text{out}_i$

4. if $\underline{\mathcal{J}}_{i+1} = \underline{\mathcal{J}}_i$ and $\overline{\mathcal{J}}_{i+1} = \overline{\mathcal{J}}_i$, stop; otherwise, repeat

They show that when the algorithm is initialised with any $\underline{J}_0 \subseteq \overline{\mathcal{J}}_0$, it converges at a finite I and that $\underline{\mathcal{J}}_I \subseteq \mathcal{J}^* \subseteq \overline{\mathcal{J}}_I$. For this reason, let the pair $(\underline{\mathcal{J}}_I, \overline{\mathcal{J}}_I)$ resulting from this algorithm be called "I-converged bounds".

The intuition relies on the "concavity" of f with respect to $\mathcal{J}$. The pair $(\underline{\mathcal{J}}_i, \overline{\mathcal{J}}_i)$ represent upper and lower bounds on $\mathcal{J}^*$. Then, D_j as calculated at $\overline{\mathcal{J}}_i$ are the lowest possible marginal values for each j, since $\overline{\mathcal{J}}_i$ is the largest possible set in consideration. The items j for which this marginal value is positive will have positive marginal values for any other set under consideration, and are therefore always beneficial to add. For this reason, these js are added to the lower bound $\underline{\mathcal{J}}_{i+1}$ for the next iteration. Similarly, marginal values D_j evaluated at $\underline{\mathcal{J}}_i$ are the highest possible values for each j. If they are not positive, then they will not be positive for any other set under consideration – so they are excluded from the upper bound $\overline{\mathcal{J}}_{i+1}$ for the next iteration. The details of the proof can be found in the original paper.

This inner iteration has a key property which will be used in the `cdc_solver` algorithm: successive lower bounds are increasing while successive upper bounds are decreasing: $\underline{\mathcal{J}}_0 \subseteq \underline{\mathcal{J}}_1 \subseteq \cdots \subseteq \underline{\mathcal{J}}_I \subseteq \mathcal{J}^* \subseteq \overline{\mathcal{J}}_I \subseteq \cdots \subseteq \overline{\mathcal{J}}_1 \subseteq \overline{\mathcal{J}}_0$.

However, this iteration does not necessarily find $\mathcal{J}^*$, since $\underline{\mathcal{J}}_I$ could be a strict subset of $\overline{\mathcal{J}}_I$. The outer loop solves addresses this possibility. Consider an initial lower and upper bound pair $(\underline{\mathcal{J}}^0, \overline{\mathcal{J}}^0)$, which will likely be $(\emptyset, J)$:

- run the inner iteration to get I-converged bounds $(\underline{\mathcal{J}}_I, \overline{\mathcal{J}}_I)$; if they are equal, stop
- if they are not equal, choose a $j \in \overline{\mathcal{J}}_I \backslash \underline{\mathcal{J}}_I$ and create two pairs: $(\underline{\mathcal{J}}_I, \overline{\mathcal{J}}_I \backslash \{j\})$ and $(\underline{\mathcal{J}}_I \cup \{j\}, \overline{\mathcal{J}}_I)$
- feed both pairs through the previous steps repeatedly to obtain a list of I-converged bounds where $\tilde{\mathcal{J}} \equiv \underline{\mathcal{J}} = \overline{\mathcal{J}}$; evaluate f at each to determine the global maximum

This outer loop forces inclusion and exclusion of items j successively as needed, calculates local maximands for each case, then compares these to find the global maximand.

3.2 Solving the CDC over a range of θ

When the goal of the algorithm switches from solving the CDC for one value of θ to characterising the response of $\mathcal{J}^*$ to θ, the main unit of interest becomes *intervals* of θ since $\mathcal{J}^*(\theta)$ is ultimately a piece-wise function, which partitions the range of θ according

to optimal $\mathcal{J}$. Intuitively, the "moving variable" is no longer $\mathcal{J}$, but θ – the problem become finding, for each possible value of $\mathcal{J}$, the values of θ for which it is optimal.

To this end, the fundamental data structure used in `cdc_solver` is a bundle of four items: $(\theta_l, \theta_r, \underline{\mathcal{J}}, \overline{\mathcal{J}})$. The first two items define the left and right endpoints of the interval. The last two items are the lower and upper bounds associated with this interval. The bundle is referred to a `prod_interval` in the module. Then, the algorithm has three overall blocks, described below.

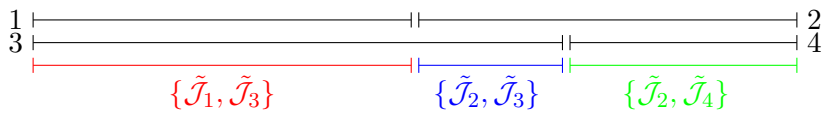
The inner block accepts a bundle $(\theta_l, \theta_r, \underline{\mathcal{J}}, \overline{\mathcal{J}})$ and returns a set of new bundles where the intervals defined by these bundles partition of the original interval. In addition, each of the new bundles $(\theta_l', \theta_r', \underline{\mathcal{J}}', \overline{\mathcal{J}}')$ are such that, for any $\theta \in (\theta_l', \theta_r')$, the I-converged bounds from the first step in Arkolakis & Eckert (2017) are $(\underline{\mathcal{J}}', \overline{\mathcal{J}}')$. In other words, this step finds the function over the range (θ_l, θ_r) mapping θ to its I-converged bounds, given initialisation $(\underline{\mathcal{J}}, \overline{\mathcal{J}})$.

If all the resulting intervals from this step have $\underline{\mathcal{J}} = \overline{\mathcal{J}}$, then the function $\mathcal{J}^*(\theta)$ has been found. If not, then the subsequent steps of the algorithm, modelled after the second iterative block from Arkolakis & Eckert (2017), are necessary.

The next procedure in `cdc_solver` takes a bundle $(\theta_l, \theta_r, \underline{\mathcal{J}}, \overline{\mathcal{J}})$ with I-converged bounds where $\underline{\mathcal{J}} \subset \overline{\mathcal{J}}$. Paralleling the second iteration in the original Arkolakis & Eckert (2017) algorithm, it chooses a $j \in \overline{\mathcal{J}} \backslash \underline{\mathcal{J}}$, then creates two new bundles: $(\theta_l, \theta_r, \underline{\mathcal{J}}, \overline{\mathcal{J}} \backslash \{j\})$ and $(\theta_l, \theta_r, \underline{\mathcal{J}} \cup \{j\}, \overline{\mathcal{J}})$. These two new bundles are fed through the first step and broken down into subintervals with I-converged bounds. Then, it continually iterates on any resulting bundles whose I-converged bounds are not equal to each other.

Finally, we are left with a collection of possibly overlapping interval bundles with I-converged bounds equal to each other. Any bundles which do not overlap with others are considered "solved", in the sense that each bundle's associated $\tilde{\mathcal{J}}$ is the optimal choice for θ in the bundle's interval. The remaining bundles have overlapping intervals, but each bundle is only associated with one choice $\tilde{\mathcal{J}} = \underline{\mathcal{J}} = \overline{\mathcal{J}}$. These $\tilde{\mathcal{J}}$ are the local maximands from the second step in Arkolakis & Eckert (2017). Let the span of the intervals from these bundles be Θ. Then, $\mathcal{J}^*$ remains unsolved for Θ.

In the original algorithm solving the CDC for a single value of θ, the local maximands are compared and the best is selected as the global maximand. When solving the CDC for the range of θ, the task is to instead find the intervals in Θ for which each local maximand is the global maximand. The first step is to partition Θ according to the set of local maximands available for each interval. As an example, suppose the previous processes from the algorithm return the four black intervals numbered $1-4$ below. Each m-numbered black interval has corresponding local maximand $\tilde{\mathcal{J}}_m$. Then, Θ is partitioned into the coloured intervals, where the red interval has $\{\tilde{\mathcal{J}}_1, \tilde{\mathcal{J}}_3\}$ as possible maximands; the blue has $\{\tilde{\mathcal{J}}_2, \tilde{\mathcal{J}}_3\}$; and the green has $\{\tilde{\mathcal{J}}_2, \tilde{\mathcal{J}}_4\}$.



Next, each of these new intervals is considered in succession. Given an interval (θ_l, θ_r)

with its set of local maximands $\{\tilde{\mathcal{J}}_n\}_{n=1}^N$, the algorithm calculates the values of $\theta_{n,n'}$ for which $f(\tilde{\mathcal{J}}_n, \theta) = f(\tilde{\mathcal{J}}_{n'}, \theta)$ for all pairs $\{\tilde{\mathcal{J}}_n, \tilde{\mathcal{J}}_{n'}\}$. Any $\theta_{n,n'} \notin (\theta_l, \theta_r)$ are discarded and the remaining values further partition (θ_l, θ_r) into cells indexed by k. By construction, the global maximand is constant within each cell. The algorithm therefore picks the midpoint θ_k, calculates $f(\tilde{\mathcal{J}}_n, \theta_k)$ for each $\tilde{\mathcal{J}}_n$, and compares these values to find the global maximand for each cell. This process concludes the calculation of $\mathcal{J}^*(\theta)$.

As a final aesthetic step, the algorithm pastes together any adjacent intervals with the same optimal $\mathcal{J}$.