Discrete Optimal Matching with Dynamic Characteristics

Antoine CHAPEL

June 2021

Abstract

We develop an extension of the static model of optimal matching between two lists of types to a dynamic setting where types' characteristics evolve through time. We consider the case where breaking an existing assignment is costly.

1 Introduction

In this research report, We will seek to transcribe the results of modelling experiments with an optimal matching problem in dynamic setting. The optimal matching described in Galichon (2016) aims at describing the surplus-maximising matching between two populations, relying on the theory of Optimal Transport. In the original version of the optimal matching problem, the characteristics of the two populations are static. If the population sizes are reasonable, it is possible to compute the coupling matrix that maps the optimal matching between the two populations. This paper aims at providing the solution to a version of this problem where the characteristics of the individuals evolve through time.

It is conceivable that, although humans do not radically change from one day to another, their characteristics may evolve through time. Among the characteristics considered in Dupuy and Galichon (2014), one may consider for example the Body Mass Index, which is likely to change through a person's life. From this it follows that a dynamic version of the optimal matching model must allow for reassignment. Since individuals' characteristics vary slowly through time, they should be reassigned to form a new optimal matching, the only rigidity being the speed of adjustment of the characteristics. However, it is unrealistic that the reassignment of the matching formed between humans is cost-less. We should thus be able to account for the various costs (financial, emotional, among others) that would follow from a reassignment.

In this paper, we will start by reminding briefly the formulation and solution of the static model, in order to keep a consistent notation between the static and the dynamic formulation. Secondly, we will describe and solve a first version of the dynamic model, in which there is no cost to reassign the agents, using

linear programming and vectorisation. Thirdly, we will generalise the dynamic model in order to allow for a reassignment cost, and compare the cases of full information and no information of the central planner regarding the individuals' evolution. We will rely on dynamic programming and the Bellman-Ford path-finding algorithm to solve the full information case[1]

2 Static Model

In this whole paper, unless specified otherwise, we will make the assumption that there are n types in each of the two populations, and that each type is fully described fully with k characteristics. Each type contains one and only one agent. Unless specified otherwise, we will not allow for unassigned agents as this is not the focus of this paper, but the algorithms described in this paper may be generalised to account for this possibility.

2.1 Data description

The data at the basis of the static optimal matching model consists in two tables of dimension (n x k), containing the list of n individuals on the vertical axis, and their k characteristics on the horizontal axis. Each table X and Y contains the types' characteristics. In addition, there is an affinity matrix A, of dimension (k x k) which allows us to compute the surplus generated by the matching of two types. For an arbitrary row of the matrix X, x_i, and an arbitrary row of the matrix Y, y_j, we write the surplus generated by their matching as: $\phi_{ij} = x_i \cdot A \cdot y_j^T$. Applying this computation for all x_i and y_j, we can generate a Φ matrix, of dimension (n x n) whose typical element ϕ_{ij} represents the surplus generated by matching male i with female j. The computation is the following: $\Phi = X \cdot A \cdot Y^T$

2.2 Problem formulation

The goal of optimal matching formulated in matrix algebra terms is to find a matrix Π of dimension (n x n) that maps individuals to each other. Since we are in the symmetrical case, an arbitrary row i or an arbitrary column j of matrix Π must contain one and only one value different from 0, meaning that every individual is mapped to exactly one individual. The optimal matrix Π^* is such that the matching is surplus maximising.

The typical element of the matrix Π^* is π_{ij}^*, which can take two values: if individual i must be matched to individual j, $\pi_{ij}^* \neq 0$. In particular, in the symmetrical case, $\pi_{ij}^* = 1/n$, meaning that the sum of the types is normalised to 1. We can incorporate the constraints and formulate the problem as:

[1]The code corresponding to all computations referred to in this paper is available at the following GitHub repository: `https://github.com/AntoineChapel/DOT-with-dynamic-characteristics`

$$\begin{aligned} \max_{\pi_{ij} \geq 0} \quad & \sum_{ij} \pi_{ij} \cdot \phi_{ij} \\ \text{s.t.} \quad & \sum_{j} \pi_{ij} = (1/n) \cdot \mathbf{1}_n, \\ & \sum_{i} \pi_{ij} = (1/n) \cdot \mathbf{1}_n \end{aligned}$$

It should be noted that the two constraints correspond to the requirements, mentioned above, that characterise the Π matrix. In order to be solvable through linear programming, the maximisation program above must be vectorised. It should be noted at this point that we will use along this paper the convention that two-dimensional vectorisation is done on the row-major order, unless specified otherwise. Explicitly:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

$$vec(A) = \begin{pmatrix} a_{11} & a_{12} & a_{21} & a_{22} \end{pmatrix}$$

With z = vec(Π), the optimal matching program can be vectorised as the following maximisation program, which can then be passed to a linear programming solver.

$$\begin{aligned} \max_{z \geq 0} \quad & vec(\Phi)^T \cdot z \\ \text{s.t.} \quad & (\mathbf{1}_n^T \otimes \mathbf{I}_n) \cdot z = vec(p), \\ & (\mathbf{I}_n \otimes \mathbf{1}_n^T) \cdot z = vec(q^T) \end{aligned}$$

3 Dynamic Model without reassignment cost

3.1 Data Generation

Let us start by defining the generation of the dynamic data at the core of this problem. Indeed, the static Φ matrix defined earlier must be adapted to a dynamic setting. Since we are starting from the assumption that individuals' characteristics vary through time, we can work only on the characteristics matrix and make the additional assumption that individuals' preferences are static through time. After standardising some initial static data, we create three-dimensional matrices in which each individual's set of characteristics becomes a time-series with random evolution.

A being an arbitrary matrix of dimension (n x k), the formula used to generate these time-series is the following, where t is the number of time periods, s is the speed of characteristics evolution and R is a (n x k) matrix containing draws from a normal distribution $\mathcal{N}(0, 1)$:

$$A^0 = A$$
$$A^t = A^{t-1} + s \cdot R^t$$

The resulting object is a three-dimensional matrix of vertical dimension n, horizontal dimension k, and "depth" t. We adopt the convention that such object has the shape (t x n x k). For later computations, we need to define the matrix product between two of such three-dimensional objects. Let A be of dimension (t x n x k) and B of dimension (t x k x l). The matrix product $A \cdot B$ is a three-dimensional matrix C of dimension (t x n x l), where all computations are dot products between (n x k)^t and (k x l)^t sub-matrices, with no interactions that cross the depth dimension.

After generating this dynamic data starting from the existing static one, we also need a dynamic affinity matrix. Since we made the assumptions that preferences wouldn't evolve, the three-dimensional affinity matrix is simply a (t x n x n) matrix consisting of t (n x n) identical affinity matrices concatenated with each other along the depth axis.

Then, we obtain easily a dynamic $\Phi_{(dyna)}$ matrix. Let $X_{(dyna)}$, $Y_{(dyna)}$ be the dynamic characteristics tables, and $A_{(dyna)}$ be the dynamic affinity matrix. We have:

$$\Phi_{(dyna)} = X_{(dyna)} \cdot A_{(dyna)} \cdot Y_{(dyna)}^T$$

With the three-dimensional matrix multiplication defined earlier, the resulting $\Phi_{(dyna)}$ object is of shape (t x n x n), and its typical element ϕ_{ij}^t corresponds to the surplus resulting from matching individual i to individual j at time t.

3.2 Formulation of the problem

Our goal is to find a dynamic $\Pi_{(dyna)}$ matrix of dimension (t x n x n) where every layer along the time axis is a coupling corresponding to the criteria defined in subsection 2.2. The problem can thus be expressed as follows:

$$\begin{aligned} \max_{\pi_{ij}^t \geq 0} \quad & \sum_t \sum_{ij} \pi_{ij}^t \cdot \phi_{ij}^t \\ \text{s.t.} \quad & \sum_j \pi_{ij}^t = (1/n) \cdot \mathbf{1}_n \quad \forall t, \\ & \sum_i \pi_{ij}^t = (1/n) \cdot \mathbf{1}_n \quad \forall t \end{aligned}$$

Since there are no reassignment cost, we could essentially obtain this dynamic assignment matrix simply by running a loop and solving t consecutive static problems. However, we will attempt to use a new vectorisation in order to solve the whole dynamic problem as one linear programming optimisation problem. At this stage, the formulation of the problem is quite close to the original static problem. The only issue is thus the vectorisation of this dynamic problem into a linear optimization program. It is straightforward to see that the objective function can be vectorised as follows:

$$\sum_t \sum_{ij} \pi_{ij}^t \cdot \phi_{ij}^t = vec(\Pi_{(dyna)})^T \cdot vec(\Phi_{(dyna)})$$

Regarding the constraints, we can start by representing them under matrix form, following the notation defined earlier.

$$\Pi_{(dyna)} \cdot \mathbf{1}_{n(dyna)} = p_{(dyna)}$$
$$\mathbf{1}_{n(dyna)}^T \cdot \Pi_{(dyna)} = q_{(dyna)}^T$$

Here, $p_{(dyna)}$ and $q_{(dyna)}^T$ are three-dimensional arrays of dimension (t x n x 1) and (t x 1 x n) respectively, containing only positive entries of value $1/n$. Since the Transpose operator is not self-evident in three-dimensional setting, we should precise that it switches only the two last dimension. If shape(A) = (a x b x c), shape(A^T) = (a x c x b), a being the time dimension in our problem.

The vectorisation of the constraints is slightly more difficult than the one of the objective function. In the static version of the problem, we could apply the vectorisation formula

$$vec(AXB) = (B^T \otimes A) \cdot vec(X)$$

This formula is unfortunately unavailable to us due to the additional dimension of the problem at hand. We should thus build our own formula, based on an arbitrary (t x n x n) matrix. Here, in order to make the problem easy to visualise, we will use a (3 x 2 x 2) matrix P, represented as three consecutive (2 x 2) matrices.

$$P = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & f \\ g & h \end{pmatrix} \begin{pmatrix} i & j \\ k & l \end{pmatrix}, \text{ such that } p = vec(P) = \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \\ j \\ k \\ l \end{pmatrix}$$

We thus have shape$(vec(P)) = ((n^2 \cdot t)$ x 1).

3.2.1 Vectorisation of the first constraint

The target vector is a vector d of dimension $(n \cdot t)$ that sums over the **rows** of each (n x n) component of the P matrix. Explicitly:

$$d = \begin{pmatrix} a+b \\ c+d \\ e+f \\ g+h \\ i+j \\ k+l \end{pmatrix}$$

We should now find a matrix V such that $V \cdot p = d$. We can build this matrix manually and see that it is:

$$V = \begin{pmatrix} 1&1&0&0&0&0&0&0&0&0&0&0 \\ 0&0&1&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&1 \end{pmatrix}$$

Matrix V has a shape (nt x n^2t), and can be rewritten as $\mathbf{I}_{nt} \otimes \mathbf{1}_n^T$.

3.2.2 Vectorisation of the second constraint

The target vector for the second constraint is a vector e of dimension $(n \cdot t)$ that sums over the **columns** of each (n x n) component of the P matrix. Explicitly:

$$e = \begin{pmatrix} a+c \\ b+d \\ e+g \\ f+h \\ i+k \\ j+l \end{pmatrix}$$

We should now find a matrix W such that $W \cdot p = e$. We can build this matrix manually and see that it is:

$$W = \begin{pmatrix} 1&0&1&0&0&0&0&0&0&0&0&0 \\ 0&1&0&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&1&0&0&0&0&0 \\ 0&0&0&0&0&1&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0&1&0 \\ 0&0&0&0&0&0&0&0&0&1&0&1 \end{pmatrix}$$

Matrix W has a shape (nt x n^2t), and can be rewritten as $\mathbf{I}_t \otimes \mathbf{1}_n^T \otimes \mathbf{I}_n$.

3.3 Resolution of the problem

We are now able to write the dynamic optimal assignment problem without reassignment cost, as one only linear programming optimization problem. With $z = vec(\Pi_{(dyna)})$ and $vec(p_{(dyna)}) = vec(q_{(dyna)}^T) = \frac{1}{nt}\mathbf{1}_{nt}$, the optimisation program can be written as follows. For easy interpretation, we may remind that t is the discrete number of time periods, and n is the number of matchings that should take place (there are 2 populations of n individuals).

$$\begin{aligned} \max_{z \geq 0} \quad & vec(\Phi_{(dyna)})^T \cdot z \\ \text{s.t.} \quad & (\mathbf{I}_{nt} \otimes \mathbf{1}_n^T) \cdot z = \frac{1}{nt}\mathbf{1}_{nt}, \\ & (\mathbf{I}_t \otimes \mathbf{1}_n^T \otimes \mathbf{I}_n) \cdot z = \frac{1}{nt}\mathbf{1}_{nt} \end{aligned}$$

This optimisation program can be passed through a linear programming solver such as Gurobi, which yields both the resulting optimal value and optimal $\Pi^*_{(dyna)}$ matrix. The optimal value of this program corresponds to the total surplus generated by the matchings across all time periods, and the t^{th} (n x n) layer of $\Pi^*_{(dyna)}$ corresponds to the optimal coupling at time period t.

Using marriage market data restricted to n = 10 individuals for easy visualisation, t = 15 time periods and a characteristics adjustment speed s = 1, we can obtain the following graph, which represents the reassignments taking place. Since the data represents the heterosexual marriage market, each line corresponds to a male i, the value on the vertical axis corresponds to the index j of the female with which male i is matched at the period t indicated on the horizontal axis. We may see on this graph that the matching does not appear to be completely random: some individuals remained matched to the same person across several time periods. This apparent rigidity is only caused by the slow evolution in individuals' characteristics, and unrelated to the assignment rigidity that we will introduce later in the model.

It is also possible to represent the matching of the entire population n = 1158. Although computationally expensive, a commercial solver such as Gurobi is able to compute a dynamic matching on up to 10 time periods in reasonable time (less than 5 minutes on a domestic laptop).

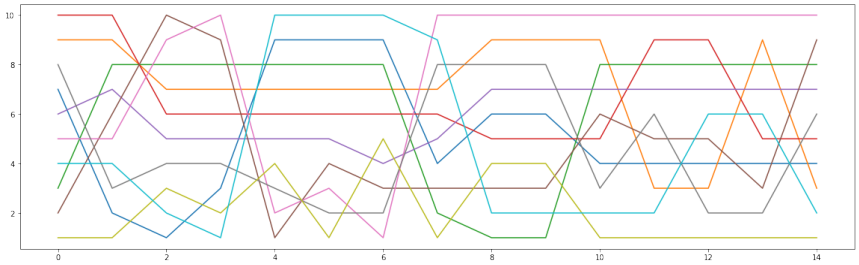

Figure 1: Dynamic optimal matching n = 10, t = 15

The resulting graph is evidently more difficult to interpret, but we may however generate it.

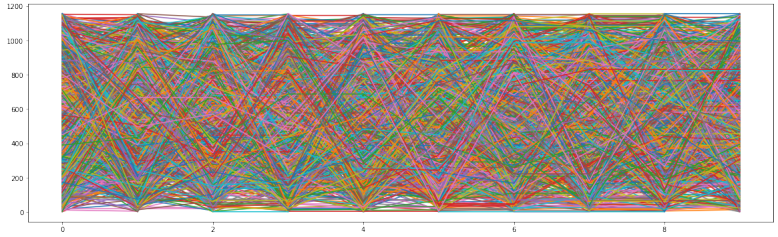

Figure 2: Dynamic optimal matching n = 1158, t = 10

4 Dynamic Matching with reassignment cost

Now that we have introduced a dynamic setting, it would be necessary to make this setting valuable in terms of analysis of the optimal matching problem. Indeed, the model of dynamic matching without reassignment cost could be assimilated to t consecutive static optimal assignments. The dynamic setting becomes interesting if the matching at time t has an impact on the problem at the dynamic level. If a matching is formed, be it between two individuals engaging in marriage or between a job offer and a job seeker, we may realistically make the assumption that there is a cost to breaking this matching, inducing some rigidity in the model. On the job market, this cost may be the loss of a trained employee replaced by an untrained one for the firm, the cost of adapting to a new firm for the job seeker... On the marriage market, a divorce may represent a financial cost but also a source of unhappiness for both individuals. All of it leads us to believe that once a matching is formed, it is unrealistic that individuals would break it immediately in order to obtain a welfare gain that may be marginal.

We need however to find a way to represent this cost in a way that can be solved numerically. The easiest way to do so is to *trick* the optimisation

algorithm into computing the optimal assignment where, if matching Π_t took place at time t, the value of Φ_{t+1} is inflated as follows, $C \geq 0$ being an arbitrary fixed matching rigidity:

$$\Phi_{t+1} = \Phi_{t+1} + C \cdot \Pi_t$$

Doing this, the matching realised at period t is made more attractive to re-conduct at period $t + 1$. We can also control the cost to reassignment by modifying C: if C tends to 0, we find the perfectly competitive optimal assignment defined earlier. If, on the other side, C tends to $+\infty$, there must be one only matching for all time periods, which logically reduces the welfare generated.

Although this formula is helpful due to its simplicity and efficiency, it raises the issue that there is no easy method to turn such problem into an optimisation program that a LP solver could solve, at least in one only program. We will consider two different algorithms, which correspond to two perspective on the problem. First, we will solve this problem under a no-information assumption, meaning that the central planner has no information at time t about the future characteristics evolution of the populations. In this framework, we will also explore a small variation of the adjustment formula in which the cost of breaking a matching is proportional to the surplus it has generated, and increasing for each period it is maintained. Secondly, we will use dynamic programming to compute a solution in which we assume that the central planner has full access to the information regarding the population's future evolution at each time period.

4.1 No information case

If we make the assumption that the central planner has no information about the population's future evolution, it follows that the central planner is simply optimising the matching, at each period, and suffers the reassignment cost at each period without attempting to limit the number of reassignments. We can see intuitively why this perspective on the problem leads to a sub-optimal result compared to a full-information one: if the reassignment cost C were set to $+\infty$, the optimal assignment realised during the first period would be maintained automatically during all following periods, while this particular coupling has no reason to be the best-performing coupling.

4.1.1 Fixed reassignment cost

The easiest algorithm to generate a dynamic coupling matrix under the no-information assumption is to simply run a loop over t consecutive static optimal matching problems and update the values inside Φ_{t+1} at each iteration. The algorithm for this first version of the no-information case is as follows:

Algorithm 1: No-information case with fixed reassignment cost

Result: Optimal dynamic coupling matrix and surplus generated
$\Pi^*_{(dyna)}$ = empty (t x n x n) matrix;
ω = empty vector of length t containing optimal values;
$\Phi_t = \Phi_{(dyna)(0)}$;
for *iteration in range(t)* **do**
  Solve the Static Optimal Assignment problem with Φ_t using Linear Programming;
  if *Optimal matching has been found* **then**
    Fill the t^{th} layer of $\Pi^*_{(dyna)}$ with $\Pi^*_{(dyna)(t)}$;
    Fill the t^{th} value of ω with the surplus generated by $\Pi^*_{(dyna)(t)}$;
  else
    Declare failure of the algorithm
  end
  Update the Φ matrix for next iteration using $\Phi_t = \Phi_{t+1} + C \cdot \Pi^*_{(dyna)(t)}$
end
return $\Pi^*_{(dyna)}$ and the sum of all elements in ω

The difference between the results yielded by different algorithms is best seen when comparing the surplus generated and the resulting matching graph. If we run this algorithm on the same data that had generated the first matching graph, only adding a reassignment cost of $C = 20$, we obtain a welfare that is lower by approximately 4.34%.

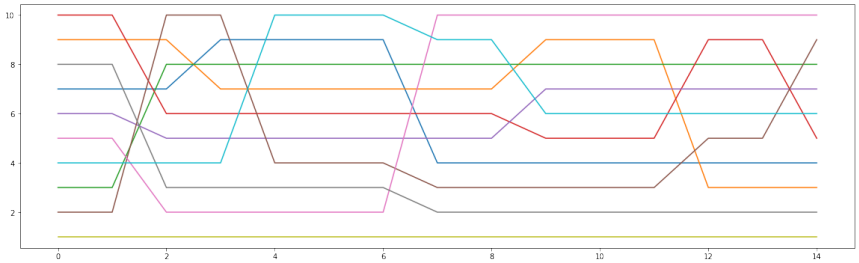

Figure 3: Dynamic optimal matching n = 10, t = 15, C = 20, with fixed cost

To obtain this result, we need to compute the surplus that was *actually* generated, meaning the surplus generated by the couplings yielded by the algorithm, and the original non-inflated $\Phi_{(dyna)}$ matrix.

4.1.2 Variable reassignment cost

We can now consider a small variant to the algorithm presented above, which may have some applicability. The fixed reassignment cost used above implies

that all couplings are broken with the same cost, no matter how long they held and the surplus they had generated. We can run the same algorithm using a slightly modified Φ matrix updating, using the following formula. ($\odot$ represents the element-wise/Hadamard product)

$$\Phi_t = \Phi_{t+1} + C \cdot \Pi^*_{(dyna)(t)} \odot \Phi_{(dyna)(t)}$$

Using this formula, the matchings that generate more surplus are more *solid* as the cost to breaking them is increased proportionally to the surplus they generated in the preceding period. This formula also has the consequence that a matching that endures time accumulates increasing surplus, making it increasingly difficult to break. This may reflect some aspects of matching markets. For example, on the job market, an employee that works for a long time in a firm acquires a lot of knowledge about the firm and becomes harder to replace. The employee may also become loyal to the firm and be reluctant to leave it. As we may expect, the matching graph with such algorithm concentrates the noise at the beginning of the matching process, and reassignments become rarer through time.

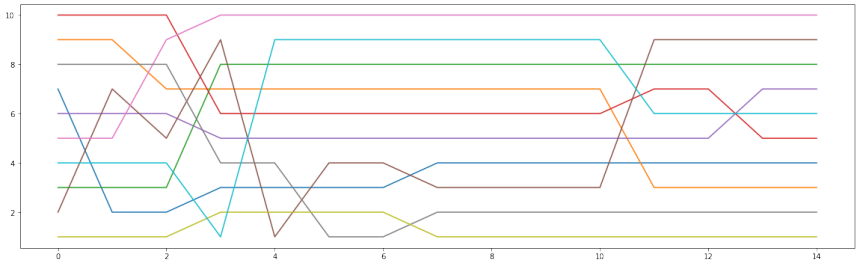

Figure 4: Dynamic optimal matching n = 10, t = 15, C = 20, with variable cost

As one may observe from the colours of the matching graphs, both matchings for fixed cost and variable cost start with the same coupling. This coupling is also identical to the first coupling generated in the first model without reassignment cost. This fact is revealing of why we must consider a case of full-information. Indeed, the fact is that the central planner in the reassignment cost models developed above is unable to anticipate necessary reassignments, and is thus unable to limit the number of reassignments if the reassignment cost becomes very high.

4.2 Dynamic Optimal Matching using dynamic programming: the full information case

In order to consider a full information case of the dynamic optimal matching problem, it is natural to consider finite-horizon dynamic programming. We can see that it applies to the present case as there are:

1. A finite series of T states, which are the inflated values of the $\Phi_{(dyna)}$ matrix, that we denote X_t, with $X_0 = \Phi_{(dyna)(0)}$
2. A series of control variables, the transpose of the potential couplings, that we denote u_t, U being the set of (n x n) matrices that correspond to the definition of a coupling.
3. A state transition function: $X_{t+1} = \Phi_{(dyna)(t+1)} + C \cdot u_t$
4. An objective function $\sum_{t=0}^{T-1} \mathrm{Tr}\,(u_t \cdot X_t)$

The dynamic programming optimization problem can thus be written as follows, given a matrix $\Phi_{(dyna)}$:

$$\begin{aligned} \max_{\{u_t\}_t : \forall t, u_t \in U} \quad & \sum_{t=0}^{T-1} (\mathrm{Tr}\,(u_t \cdot X_t)) \\ \text{s.t.} \quad & X_{t+1} = \Phi_{(dyna)(t+1)} + C \cdot u_t, \\ & X_0 = \Phi_{(dyna)(0)} \end{aligned}$$

There exist many different approaches as to how to solve a dynamic programming optimisation problem. The simple treatment familiar to economists used in Sydsaeter and al. (2008) relies primarily on the backwards induction algorithm, whose implementation is not natural here given the nature of the objective function. Indeed, each control u_t is a particular type of matrix, a coupling, for which the computation of a gradient would not be pertinent. The second issue to which we may be confronted here is the factorial nature of the problem. Given a (n x n) surplus matrix, there are $n!$ possible couplings. For example:

$$n = 2 : \begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \end{pmatrix} \begin{pmatrix} 0 & 1/2 \\ 1/2 & 0 \end{pmatrix}$$

$$n = 3 : \begin{pmatrix} 1/3 & 0 & 0 \\ 0 & 1/3 & 0 \\ 0 & 0 & 1/3 \end{pmatrix} \begin{pmatrix} 1/3 & 0 & 0 \\ 0 & 0 & 1/3 \\ 0 & 1/3 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1/3 & 0 \\ 1/3 & 0 & 0 \\ 0 & 0 & 1/3 \end{pmatrix} \begin{pmatrix} 0 & 1/3 & 0 \\ 0 & 0 & 1/3 \\ 1/3 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1/3 \\ 1/3 & 0 & 0 \\ 0 & 0 & 1/3 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1/3 \\ 0 & 1/3 & 0 \\ 1/3 & 0 & 0 \end{pmatrix}$$

Thus, only generating the set of couplings may become a challenge if there are more than 10 couples to form.

As one can read in Bertsekas (2017), deterministic dynamic programming problems can be reformulated under a shortest-path problem. In the present

case, we are dealing with a deterministic problem: each coupling can lead to one and only one Φ state in the next period, without probabilities being involved in any way. Therefore, we should be able to represent and solve the dynamic programming problem by using a shortest-path reformulation. Although this solves the first issue, the second issue is exacerbated by this solution. Indeed, most basic path-finding algorithms have to explore every node in order to find the shortest path. Although this limits the resulting algorithm to small-scale problems, it provides us with a basic algorithm able to solve the problem submitted. We may expect that the resulting dynamic matching causes less reassignments, but also more clever assignments, as they would be able to predict the future state of individuals' characteristics.

Let us reformulate the dynamic problem as a network optimisation problem. Let nodes be the states of the $\Phi_{(dyna)}$ matrix, the original node 0 be $\Phi_{(dyna)(0)}$, and the value of edges from node a to node b represent the surplus generated by the matching realised at node a with the matching leading to node b. At each period of time except the first one, there will thus be $n!$ nodes, corresponding to the $n!$ possible couplings. For t periods of time, there will be $(t-1) \cdot n! + 1$ nodes and $n! \cdot (t-2) + n!$ edges. As one can see on the graph below, there are also $n!$ additional terminal nodes which are not states of the Φ matrix but simply represent the end of the optimisation process, the last edge containing the surplus generated by the last matching.

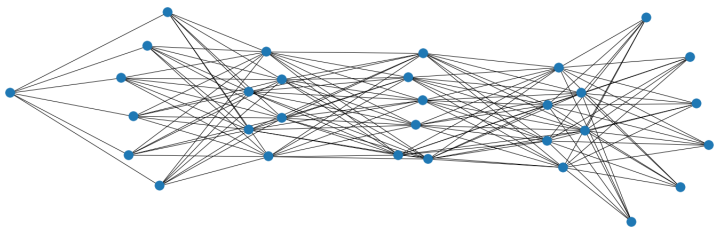

Figure 5: Network graph of the problem for n = 3, t = 5

We have here a directed graph, without any cycles. Thus, we can rely on a path-finding algorithm such as the Bellman-Ford algorithm in order to find the sequence of nodes that maximises the surplus. It has to be slightly modified in order to encompass the fact that we are seeking the *longest* path, as every edge contains the surplus generated by a matching, and we seek to maximise the surplus. One solution could be to transform the payoffs into negative payoffs and keep the minimising formulation of the Bellman-Ford algorithm. Another is to transform the Bellman-Ford algorithm to find the longest path. We followed the latter approach in this paper, noted that we do not have to worry about negative weight cycles (a source of failure for the Bellman-Ford algorithm) given that our graph is globally directed and does not allow for cycles.

The Bellman-Ford algorithm returns a path of nodes from the source node to every node of the graph, and the surplus generated by each path. First, we

restrict the result to the $n!$ terminal nodes, and apply a map from the set of nodes to the set of couplings. The best performing series of couplings is the optimal $\Pi^*_{(dyna)}$ dynamic coupling matrix. It is then possible to interpret this dynamic coupling matrix as done with the two other algorithms.

Additionally, it is possible to derive the value of information in the problem by comparing the surplus generated by the dynamic programming algorithm and the one generated by the loop algorithm. As one may expect, the value of information increases as the reassignment cost increases, up to the point (when it exists) where both algorithms yield a stable coupling without reassignment, causing the value of information to stagnate. As one can see, the value of information is punctually negative for small reassignment cost values, indicating a flaw in the latter algorithm, which is caused by the *trick* used to inflate the value of the Φ matrix. A simpler algorithm that directly penalises changes in assignment would not fail punctually, but it would not be possible then to compare it with the loop algorithm and compute the value of information.

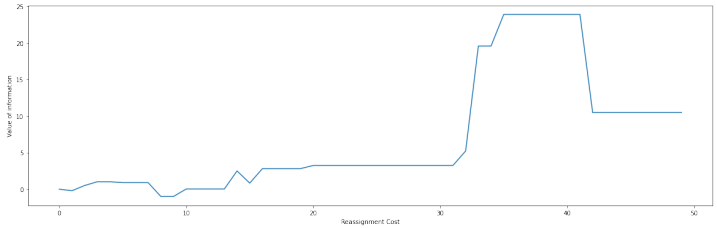

Figure 6: Value of information for varying reassignment costs, n = 4, t = 20

We may also compare the surplus loss caused by the reassignment cost, by testing the three algorithms on identical data. As one can expect, the dynamic programming algorithm largely outperforms the loop algorithm as the reassignment cost increases, due to the difference in information between the two models. However, one can observe that the loop algorithm punctually generates a higher surplus than the dynamic programming one, which should not happen in theory. This corresponds to the punctual negative value of information, and flaw in the dynamic programming algorithm mentioned above.

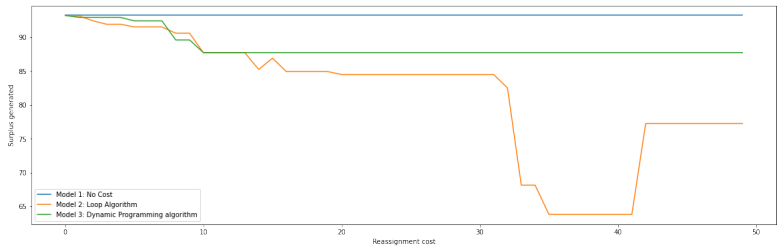

Figure 7: Comparison of the algorithms' performance on n = 4, t = 20

5 Conclusion

This paper explores three different algorithms for the resolution of dynamic matching problems. The first is limited to a case where there is no cost to reassigning the agents, but is able to solve mid-scale problems as it relies on an efficient linear programming algorithm. The second one relies on a loop of static problems. It is an approximation of the optimal solution for small reassignment costs, but its inability to predict the necessary reassignments makes it unsuited for large reassignment costs. The last and most general one relies on dynamic programming and the Bellman-Ford path-finding algorithm to compute a globally optimal solution. It is particularly efficient on large reassignment costs, compared to the loop algorithm. Unfortunately, the algorithm used is only useful for small-scale problems and fails as soon as $n > 5$.

The dynamic perspective chosen in this paper opens the way to many additional problems. First, on the level of economic modelling, it would be a necessary addition to drop the deterministic assumption made on the evolution of individuals' characteristics. Indeed, it is unlikely that a coupling between two individuals or between a firm and an employee has no consequence on either or both stakeholders of the coupling. We may, for example, make the assumption that a coupling leads to a convergence over time between the characteristics of the stakeholders, when the characteristics are identical on both sides. Otherwise, one may make a stochastic assumption on the evolution of characteristics, an individual i remaining the same between two periods with probability p_i and evolving with probability $1 - p_i$. Finally, it would also be necessary to drop the assumption of static preferences made in this paper. It is indeed unlikely that individuals' preferences remain fixed, especially in the marriage market.

The most direct challenges raised from this paper are algorithmic and computational. The dynamic matching model with reassignment cost gives rise to a huge number of potential couplings sequences. It would thus be necessary to design a more efficient, non-brute-force algorithm. One possibility could be the use of path-finding algorithms that rely on heuristics in order to avoid exploring every path, such as the A^* algorithm. However, this may not lead to a significant improvement given the nature of the network and the fact that we would still have to begin with generating all possible couplings and coupling sequences. A more interesting option could be the use of sub-optimal dynamic programming algorithms, such as reinforcement learning, which are targeted to large-scale dynamic programming problems.

References

[1] A. Galichon, 'math+econ+code' masterclass on optimal transport and economic applications. January 2021

[2] Bertsekas, D., 2017. Dynamic programming and optimal control. 4th ed. Belmont: Athena Scientific.

[3] Dupuy and Galichon, December 2014. Personality Traits and the Marriage Market. Journal of Political Economy, Vol. 122, No. 6, pp. 1271-1319

[4] Sydsaeter et al, 2008. Further Mathematics for Economic Analysis. 2nd ed.

[5] Galichon, A., 2016. Optimal transport methods in economics. 1st ed. Princeton: Princeton University Press.