Efficient and Effective Pair-Matching Algorithms for Microsimulations

Nathan Geffen[1] **and Stefan Scholz**[2]

[1]*Department of Computer Science, University of Cape Town, Rondebosch, Cape Town, 7700, South Africa*
[2]*Center for Health Economics Research Hannover (CHERH), Leibniz University of Hannover, Universitätstr. 25, 33615 Bielefeld, Germany*
Correspondence should be addressed to *nathangeffen@gmail.com*

Journal of Artificial Societies and Social Simulation 20(4) 8, 2017
Doi: 10.18564/jasss.3485 Url: http://jasss.soc.surrey.ac.uk/20/4/8.html

Received: 16-11-2016 Accepted: 11-05-2017 Published: 31-10-2017

Abstract:

Microsimulations and agent-based models across various disciplines need to match agents into relationships. Some of these models need to repeatedly match different pairs of agents, for example microsimulations of sexually transmitted infection epidemics. We describe the requirements for pair-matching in these types of microsimulations, and present several pair-matching algorithms: Brute force (BFPM), Random (RPM), Random k (RKPM), Weighted shuffle (WSPM), Cluster shuffle (CSPM), and Distribution counting (DCPM). Using two microsimulations, we empirically compare the speeds, and pairing quality of these six algorithms. For models which execute pair-matching many thousands or millions of times, BFPM is not usually a practical option because it is slow. On the other hand RPM is fast but chooses poor quality pairs. Nevertheless both algorithms are used, sometimes implicitly, in many models. Here we use them as yardsticks for upper and lower bounds for speed and quality. In these tests CSPM offers the best trade-off of speed and effectiveness. In general, CSPM is fast and produces stochastic, high quality pair-matches, which are often desirable characteristics for pair-matching in discrete time step microsimulations. Moreover it is a simple algorithm that can be easily adapted for the specific needs of a particular domain. However, for some models, RKPM or DCPM would be as fast as CSPM with matches of similar quality. We discuss the circumstances under which this would happen.

Keywords: Agent-Based Modelling, Pair-Matching, Partner Matching, Sexually Transmitted Infections, HIV

Introduction

1.1 Microsimulations and agent-based models (ABMs) are increasingly used across a broad area of disciplines, i.e. biology (Gras et al. 2009), sociology (Macy et al. 2002), economics (Deissenberg et al. 2008) and epidemiology (Gray et al. 2011). In many of these applications an artificial society of agents, usually representing humans or animals, is created, and the agents need to be paired with each other to allow for interactions between them.

1.2 Zinn (2012) addresses some of the conceptual challenges of finding suitable pairs of agents, particularly with respect to closed continuous ABMs. The author differentiates between stochastic versus stable matching rules, discusses different measures of compatibility between agents (which we call distance functions) and which agents choose their partners and which only get chosen. While the paper provides a conceptual framework of matching procedures, computational algorithmic aspects are left out and only little research can be found on this topic. Bouffard et al. (2001) presents an algorithm for finding suitable pairs for marriage. However this algorithm would be too slow for the repeated pair-matching required in many microsimulations of sexually transmitted infections (STIs).

1.3 The identification of suitable pairs of agents for partnerships may be difficult to compute efficiently. If a simulation does not measure the compatibility of two agents, i.e. it assumes that agents behave uniformly and match randomly with one another, then it risks failing to model essential features that determine the outcomes being studied.

1.4 On the other hand if a model attempts to measure the compatibility of agents, so that it produces sets of agent partnerships that very closely match the population being studied, it may become too computationally slow to study large populations; it may even be too slow for small populations where agents seek new partners repeatedly. The choice of pair-matching algorithm therefore needs to consider the trade-off between accuracy and speed to ensure the feasibility of the microsimulation for the research question at hand. This is especially true if the model is stochastic and needs to be run many times for the construction of confidence intervals or sensitivity analyses.

1.5 Two algorithms are commonly used for pair-matching in microsimulations. One randomly matches agents in a mating pool into pairs. While this does not usually yield good quality matches with respect to the actual distribution of partnerships in the population being studied, it is fast: the time of the pair-matching procedure is linear with the number of agents being matched.

1.6 The second algorithm is a brute force approach (using common computer science terminology, e.g. Levitin (2011)). This algorithm compares each agent to every other unmatched agent in the mating pool, using a compatibility or distance function, in order to choose appropriate partnerships. While this method usually yields good quality matches, it is slow, with execution time quadratic with the number of agents being matched.

1.7 The aim of our study is to present several alternative pair-matching algorithms for microsimulations and ABMs, which allow for a better trade-off between good quality matches and computation time than the random and brute-force methods. After the general problem statement and a description of general characteristics of pair-matching algorithms, we provide a detailed description of the matching algorithms. As this paper focuses on the computational features, we analyse the respective relative efficiency and effectiveness of the algorithms compared to brute force and random matching algorithms for one abstract microsimulation, and one microsimulation that models some features of a natural population in the field of epidemiology.

1.8 We have provided open-source C++ implementations of the algorithms, although modellers will likely wish to adapt them for their domains. We hope that these algorithms will lead to faster more accurate, easier-to-implement microsimulations, especially, but not only, of STI epidemics.

Background

General problem statement

2.1 The typical structure of a discrete microsimulation or ABM is that the simulation is divided into time steps, and in each time step events execute over all the agents. For example, see Algorithm 1.

Algorithm 1 Structure of a discrete microsimulation

```
for each time step do
    for each event e do
        for each agent a do
            if e has to be applied to a then
                Apply e to a
            end if
        end for
    end for
end for
```

2.2 The practicability of a simulation is dependent on the efficiency of the events. A particularly simple event is to age each agent. This merely requires incrementing an age property for each agent. The execution time is linear with the number of agents. We usually want the efficiency class of events to be linear, or at worst linearithmic (i.e. the execution time is proportional to $n \log n$, where n is the number of agents being matched). An event whose time efficiency is quadratic (n^2) with the number of agents being matched will slow simulations with large numbers of agents to the point that it may become unfeasible to generate confidence intervals or conduct sensitivity testing.

2.3 Pair-matching represents a more complex event involving the properties of more than one agent and can be stated formally as follows: We have a set of n agents eligible for pairing $a_1, a_2 ... a_n$. We have a distance function $distance$ which takes two agents as its parameters such that if $distance(a, b) < distance(a, c)$ then b is

a more likely, compatible, suitable or appropriate match for a than c. (If it is possible that $distance(a,b) = distance(a,c)$ the modeller must decide on a tie-breaking mechanism.) A pair-matching solution is a set of matches such that every agent is paired with exactly one other agent. This can be recast as a fully-connected graph problem such that every vertex is an agent and every edge is a distance between two agents.

2.4 There exists for such a graph with n vertices multiple sets of distinct pair-matchings. One or more of these is a minimum-distance — or perfect — set in that the sum of all the distances between the pairs is less than or equal to every other pair-matching set. There are algorithms that find the minimum-distance set of pair-matchings, such as Blossom V (Kolmogorov 2009; Cook & Rohe 1999). However, Blossom V suffers from two serious problems: it is far too slow for most microsimulations, and it is not stochastic — it always produces the same set of pairs, unless there are multiple perfect pair-matching sets, in which case the algorithm could be modified to produce a random tie-breaking mechanism. Usually, though, more stochasticism than this is required. Nor is reproducing the expected value of a probabilistic distribution with certainty usually a desirable statistical attribute of a microsimulation.

2.5 Instead we are interested in pair-matching algorithms that approximate the underlying distribution, and do so quickly. This paper presents several such pair-matching algorithms, some of them novel. They are tested, compared and analysed, including against the Blossom V algorithm.

Characteristics of pair-matching algorithms

2.6 Pair-matching algorithms may be described and categorized in general by the following characteristics:

Distance function If agents are not matched randomly, a distance function must be defined. This function indicates the compatibility of a pair of agents for a partnership based on the distribution of partnerships in the population being modelled, as described in 2.3.

Exact vs approximate In some applications, agents are only paired if they are an exact match, i.e. for two agents a and b, they are paired if and only if $distance(a,b)$ equals a defined value. In other applications, only an approximate match is necessary. In this paper the algorithms are compared using approximate matching applications. Nevertheless, the algorithms can all be adapted to do either approximate or exact matching.

Stochasticism Stochasticism in partner choice is usually desired in microsimulations, for example so that multiple executions of the simulation differ from each other. As agents are usually stored in an array data structure, or similar, pair-matching algorithms will process the array from front to back. If the agents that are processed first are more likely to find compatible partners than those processed last, as the pair-matching event is repeated in subsequent time-steps, the partner selections will become increasingly biased. An easy way to avoid this "storage" bias is through the introduction of randomness by shuffling agents using the technique described in Knuth (1997, pp.142–146) and that is implemented in most modern programming language standard libraries. Further randomness can be introduced if desired. For example, the maximum number of evaluated potential matching candidates or the threshold of the value of the distance function for accepting partners may be stochastic.

Effectiveness We call an algorithm's success at generating matches its effectiveness. We evaluate this using two measures. When exact matching is used, the first effectiveness measure is calculated as the proportion of the agent population, who are supposed to be in relationships, that are actually in relationships after the algorithm is executed across all agents. For approximate matching, effectiveness is the mean or median distance between all paired agents. The second effectiveness measure is defined as follows: For any agent a we can rank all other agents in order of distance from a. Then the effectiveness is the mean or median rank of all agent partners. Both measures are relative, if the best possible matching result is unknown (because it is too computationally demanding to calculate).

Initialisation vs in-simulation All the pair-matching algorithms discussed here are intended for execution as events during a simulation as in Algorithm 1. However, in some simulations, it is necessary to pair agents before the simulation starts. An example of an algorithm to do this is discussed in Scholz et al. (2016).

Distance functions

2.7 Euclidean distance would be a convenient measure of the suitability of pairing two agents a and b with m appropriately scaled properties $a_1, b_1, a_2, b_2 ... a_m, b_m$. For multi-dimensional space, the equation for Euclidean distance is:

$$d(a,b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + ... + (a_m - b_m)^2}$$

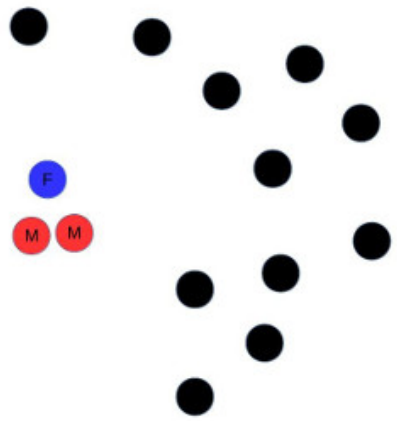

Figure 1: Heterosexual pair-matching violates the Triangle Inequality and does not map to a Euclidean plane. The two male agents, marked M, are closer on the Euclidean plane than the female agent, marked F, but if we are modelling heterosexual pair-matching then the distance function will record the two males as being further apart than the agent marked F.

2.8 Algorithms exist that efficiently but approximately find the nearest neighbour to a point in high-dimensional Euclidean space (Indyk & Motwani 1998; Beis & Lowe 1997; Arya et al. 1998). It is unclear how easily these can be adapted to microsimulations, but perhaps if we could map the properties of agents to Euclidean space, we could use one of these algorithms (or a variation thereof) to find suitable partners.

2.9 To map the matching properties to metric space, of which Euclidean space is one example, the distance function would have to satisfy the triangle inequality (Weisstein 2016). For agents a, b and c this is:

$$distance(a, b) + distance(b, c) \geq distance(a, c)$$

2.10 In some simulations the agent properties violate the triangle inequality. An example is the primarily heterosexual HIV epidemic in South Africa. Consider the agent properties we are interested in for measuring pairing suitability:

- age
- sex
- desire for a new partnership at this time
- risk behaviour propensity (including whether or not the agent is a sex worker)
- relationship status (including whether or not the agent is married)
- location

2.11 The more similar two individuals the closer they are to each other in Euclidean space, but this is not the case for heterosexual agents in an HIV epidemic: If two people share the same sex they are not suitable partners in this model. See Figure 1 for a graphical depiction of how this violates the triangle inequality.

2.12 The problem is not confined to modelling heterosexual relationships. Even if we want to model only men who have sex with men (MSM) with respect to an STI, it would be difficult to map our agents to Euclidean space. We might want our microsimulation to make it less likely for married agents seeking a new partnership to partner with other married agents, or with other agents currently in a relationship. When searching for a new relationship, we might want the distance to be great between otherwise well-matched agents who have been in a partnership with each other previously. We might also want to extend our model to account for people looking for partners of different age, wealth or education from themselves.

2.13 In our model's set of agent properties, age, risk behaviour propensity and geographical location may be attributes that map well to Euclidean space; we could define our distance function so that the more similar these are between agents, the more likely they are to form partnerships. However, sex for heterosexual agents does not map to Euclidean space, and, depending on the model's assumptions, neither might several other agent

attributes. (While it might be argued that sexual orientation is a categorical variable, a distance function simply returns a real number, and therefore has to represent incompatible categories of agents by calculating a very high value for such pairs of agents.)

2.14 We call a model's agent properties which do map well to Euclidean space *attractors*, and attributes that do not *rejectors*. Two of the algorithms we present (CSPM and WSPM) work by clustering agents together based on their attractors. Therefore clustering functions have to be defined by modellers who use these algorithms. However, when the rejectors dominate the distance function, the cluster function is less useful and the effectiveness of these two algorithms degrades.

Pair-matching algorithms

2.15 Besides the above mentioned Brute force pair-matching (BFPM) and Random pair-matching (RPM) algorithms, we describe the following four pair-matching algorithms.

- Random k pair-matching (RKPM)
- Weighted shuffle pair-matching (WSPM)
- Cluster shuffle pair-matching (CSPM)
- Distribution counting pair-matching (DCPM)

Brute force pair-matching (BFPM)

2.16 This algorithm works as follows: For every agent a in a set of agents, it calculates the distance to every remaining unpaired agent after a in the set. The agent b with the smallest distance to a is marked as a's partner (and vice-versa).

2.17 This is a naive algorithm that is very effective. However, unless pair matching only needs to be executed a few times, or on small populations, it is impractically slow — quadratic with the number of agents being paired. For example to run one simulation of a subset of the South African population to model the HIV epidemic for five years, we iterate over tens of thousands of agents daily or weekly, executing pair-matching on each iteration. Numerous simulations need to be run in order to build confidence intervals or to perform sensitivity testing (see for example Hontelez et al. (2013) and Johnson & Geffen (2016)). The *BFPM* algorithm is usually too slow for this purpose.

2.18 As BFPM, or variations thereof, is widely implemented in microsimulations and ABMs, we use it here as a reference against which to measure the effectiveness and speed of the other algorithms.

2.19 The pseudocode for *BFPM* is provided in Algorithm 2.

Algorithm 2 Brute force pair-matching (BFPM)

Parameters: $Agents$, an array of agents, with subscripts $0..n-1$, where n is the number of agents. If n is uneven, one agent will remain unmatched.

```
1:  function BruteForceMatch(Agents)
2:      shuffle(Agents)                          ▷ So that the algorithm is stochastic
3:      for each unmatched agent a in Agents do
4:          best ← ∞
5:          for each unmatched agent b after a in Agents do
6:              d ← distance(a, b)
7:              if d < best then
8:                  best ← d
9:                  bestPartner ← b
10:             end if
11:         end for
12:         Make a and bestPartner partners
13:     end for
14: end function
```

Random pair-matching (RPM)

2.20 Many deterministic models that use differential equations to model population dynamics or epidemiology, as well as simple microsimulations make the simplifying assumption that people are randomly paired with their sexual partners. This corresponds to a random matching algorithm using no distance function. Although this is extremely fast, with execution speed linear with the number of agents being matched, by ignoring the compatibility of matched agents it may result in the simulation producing outputs that differ wildly from the population being modelled.

2.21 As with BFPM, this algorithm's primary purpose is as a yardstick against which to measure the other algorithms. A good algorithm will execute much faster than BFPM but perhaps considerably slower than RPM. The effectiveness of a good algorithm should far exceed RPM. Algorithm 3 presents the pseudocode for this algorithm.

Algorithm 3 Random pair-matching (RPM)

Parameters: $Agents$, an array of agents, with subscripts $0..n-1$, where n is the number of agents. For simplicity assume n is even.

```
1: function RANDOMMATCH(Agents)
2:     shuffle(Agents)
3:     for i ∈ 0, 2, 4.., n − 4, n − 2 do
4:         make Agents[i] and Agents[i + 1] partners
5:     end for
6: end function
```

Random k pair-matching (RKPM)

2.22 This algorithm is a small conceptual advance on RPM and BFPM (Pelillo 2014; Larose & Larose 2014). Instead of only considering all agents following a, the one under consideration (as in BFPM), or matching with the adjacent agent (as in RPM), we consider the k adjacent agents after a in the array of agents. We partner a with the agent with the smallest distance. Assuming k is a constant (which it is in our implementation), the efficiency of this algorithm is linear with the number of agents.

2.23 The pseudocode for *RKPM* is provided in Algorithm 4.

Algorithm 4 Random k pair-matching (RKPM)

Parameters:
$Agents$, an array of agents, with subscripts $0..n-1$, where n is the number of agents. For simplicity assume n is even.
k, the number of adjacent agents to consider when finding a suitable partner

```
1: function RANDOMKMATCH(Agents, k)
2:     shuffle(Agents)
3:     for each unmatched agent a in Agents do
4:         best ← ∞
5:         for each unmatched agent b in one of up to k positions in the array after a do
6:             d ← distance(a, b)
7:             if d < best then
8:                 best ← d
9:                 bestPartner ← b
10:            end if
11:        end for
12:        Make a and bestPartner partners
13:    end for
14: end function
```

2.24 Surprisingly, unlike RPM the effectiveness of this simple algorithm is quite good in our applications. In effect, assuming n agents in the mating pool, the algorithm randomly draws k elements without replacement from an

ordered list of n elements, and selects l, the lowest of the k elements in the ordering. The mean position of l is $\frac{1}{k+1} \times n$ in the ordered list, and the median is $1 - e^{-\frac{\ln(2)}{k}} \times n$.

2.25 This means, for example, if we have $k >= 100$, then on average the partner chosen will be ranked in the top one percent in suitability from the remaining available agents.

Weighted shuffle pair-matching (WSPM)

2.26 This algorithm is an extension of RKPM, except that instead of ordering the agents entirely randomly, those with compatible pair-matching attributes are more likely to be clustered together. The algorithm works as follows:

2.27 Agents are assigned a cluster value. Then the cluster value is multiplied by a uniform random number — giving a random value weighted towards the cluster of the agent — to introduce stochasticism into the algorithm. Then the agents are sorted on this weighted value. Finally for each agent a, the agent with the smallest distance to a in the k adjacent agents is selected.

2.28 To calculate the cluster value for each agent, we need a domain-specific clustering function. This would typically be based solely on the attractor attributes, or a subset thereof, of the distance function. Despite being domain specific, a programmer without knowledge of the application can work out a cluster function solely by examining the distance function. However, it is likely that with greater domain knowledge, a more sophisticated cluster function can be defined.

2.29 Algorithm 5 is an example of a simple cluster function using some of the attractor attributes described above.

Algorithm 5 Example of a cluster function

1: **function** CLUSTER(a) ▷ a is an agent
2: **return** $AGE\_FACTOR * a.age + ORIENTATION\_FACTOR * a.orientation + RISK\_FACTOR * a.riskiness$
3: **end function**

2.30 Algorithm 6 provides pseudocode for WSPM.

Algorithm 6 Weighted shuffle matching (WSPM)

Parameters:
$Agents$**, an array of agents, with subscripts** $0..n-1$**, where** n **is the number of agents. For simplicity assume** n **is even.**
k**, the number of adjacent agents to consider when finding a suitable partner.**
The function `rand()` **returns a random number in the range** $[0..1)$**.**

1: **function** WEIGHTEDSHUFFLEMATCH($Agents$, k)
2: **for** each agent a in $Agents$ **do**
3: $a.weight \leftarrow cluster(a) * rand()$
4: **end for**
5: sort $Agents$ by weight
6: Execute lines 3 to 13 of *RKPM* (Algorithm 4).
7: **end function**

2.31 Assuming k is a constant, which it is in our implementation, then sorting has the slowest efficiency of the steps in this algorithm, linearithmic with the number of agents being matched. Therefore the efficiency of the entire algorithm is linearithmic with the number of agents.

Cluster shuffle pair-matching (CSPM)

2.32 This algorithm is a conceptual advance on WSPM. As with WSPM, it also extends RKPM and uses a cluster function. It works as follows:

2.33 The agents are sorted by the value returned by the cluster function (as opposed to a cluster weighted random number as in WSPM). They are then divided into a user-specified number of clusters. Then each cluster is shuffled to introduce stochasticism. (In contrast to WSPM, agents in the same cluster always remain relatively close

to each other.) Finally, just as with WSPM and RKPM, it finds the best of k agents after the agent under consideration.

2.34 Algorithm 7 provides the pseudocode for this algorithm.

Algorithm 7 Cluster shuffle pair-matching (CSPM)

Parameters:
$Agents$**, an array of agents, with subscripts** $0..n-1$**, where** n **is the number of agents. For simplicity assume** n **is even.**
c**, the number of clusters to divide the agents into. For simplicity assume** c **divides into** n**.**
k**, the number of adjacent agents to consider when finding a suitable partner.**

```
1:  function CLUSTERSHUFFLEMATCH(Agents, c, k)
2:      for each agent, a, in Agents do
3:          a.weight ← cluster(a)
4:      end for
5:      sort Agents by weight
6:      clusterSize ← n/c
7:      i ← 0
8:      for each cluster do
9:          first ← i * clusterSize
10:         last ← first + clusterSize
11:         shuffle Agents[first...last − 1]          ▷ to introduce stochasticism
12:         i ← i + 1
13:     end for
14:     Execute lines 3 to 13 of RKPM (Algorithm listing 4).
15: end function
```

2.35 As with WSPM, the efficiency of the entire algorithm is linearithmic with the number of agents, because the only step that is not linear is the sort, which is linearithmic.

Distribution counting pair-matching (DCPM)

2.36 This algorithm is influenced by sorting by counting (Knuth 1998, p 75-79), also called distribution counting sort (Levitin 2011, p 283). It is only useful if the agents can be distributed into buckets that exactly or approximately correspond to the properties of the partners with whom they need to be matched. Therefore a domain specific function that places a given agent into one of a set of predefined buckets must be defined.

2.37 The algorithm works as follows: Pointers to the shuffled agents are copied into an array. The copy is sorted using distribution counting which makes use of the bucket function (Levitin 2011, p 283). This sorting has linear efficiency — as opposed to the linearithmic efficiency of standard sorting algorithms — because it uses information about the underlying distribution of the objects provided by the bucket function.

2.38 A table with the same number of entries as there are buckets is then constructed so that given the "desired" partner properties of an agent, a, we can directly access the first agent with those characteristics in the copied agents array. We then linearly examine up to k agents, selecting the partner with the closest distance. The algorithm uses a defined $getBucket$ function which assigns an agent, or its "desired" partner to a bucket.

2.39 To understand how agents are assigned to buckets consider the agent properties of age, sex, and sexual orientation. If an agent representing a female "desires" a 25-year-old male, then it is assigned to the bucket of agents looking for 25-year-old heterosexual males. When looking for a partner, it will search up to k agents in this bucket and choose the one with the lowest distance to it. If the model's agents are either male or female, heterosexual or homosexual, and any age from 16 to 50, then there are $2 * 2 * 35 = 140$ buckets.

2.40 One caveat: Because age is a flexible characteristic (e.g. if the desired partner of a is 25 it does not mean that if b is a 24-year-old, it should have no prospect of partnership with a), the algorithm can be adapted so that it searches for a partner in several adjacent buckets, which differ by one or more years of age.

2.41 Assuming k is a constant, which it is in our implementation, the algorithm's execution time is linear with the number of agents being matched.

2.42 The pseudocode for the unadapted algorithm is provided in Algorithm 8.

Algorithm 8 Distribution counting pair-matching (DCPM)

Parameters: $Agents$**, an array of** $0..n-1$ **pointers to agents,**
$buckets$**, the number of buckets the agents can be distributed into**
k**, the number of agents in a bucket to search for a partner.**
There is also a domain-specific $getBucket$ **function which assigns an agent to its correct bucket in the range** $[0..buckets-1]$**.**

function DISTRIBUTIONCOUNTINGMATCH($Agents$, $buckets$, k)
  $shuffle(Agents)$
  $copyAgents \leftarrow Agents$
  Sort $copyAgents$ using sort by counting and the $getBucket$ function
  for $i = 0$ to $buckets - 1$ **do**
    $Table[i].start \leftarrow 0$
    $Table[i].entries \leftarrow 0$
  end for
  for each a in $Agents$ **do**
    $bucket \leftarrow getBucket(a)$
    $table[bucket].entries \leftarrow table[bucket].entries + 1$
  end for
  $lastIndex \leftarrow 0$
  for $i = 0$ to $buckets - 1$ **do**
    $table[i].start \leftarrow lastIndex$
    $lastIndex \leftarrow lastIndex + table[i].entries$
  end for
  for each unmatched agent a in $Agents$ **do**
    $bucket \leftarrow getBucket(a's\ desired\ partner)$
    $startIndex \leftarrow table[bucket].start$
    $endIndex \leftarrow startIndex + table[bucket].entries$
    $lastIndex \leftarrow min(startIndex + k, endIndex)$
    Find agent b that returns the smallest $distance(a, b)$ in $copyAgents[startIndex..lastIndex - 1]$
    Make a and b partners
    $swap(copyAgents[index\ of\ agent\ b], copyAgents[lastIndex - 1]$
    $table[bucket].entries \leftarrow table[bucket].entries - 1$
  end for
end function

Methods

Simulation experiments

General set-up

3.1 To test and analyze the algorithms, two different simulations have been programmed that use the structure depicted by Algorithm 1. The two models differ in the properties of the agents and the distance functions. While the first (ATTRACTREJECT) is somewhat abstract and intended to explore the relative effect of attractor and rejector attributes, the second (STIMOD) resembles a more concrete application in the field of epidemiology taken from research on the HIV-epidemic in South Africa (Eaton et al. 2012; Hontelez et al. 2013; Johnson & Geffen 2016).

3.2 The purpose of these simulations is solely to test and compare the algorithms; the simulations are not intended to model the natural world. They also differ from simulations of the natural world in that every agent on every iteration (or time step) is in the mating pool, so that comparisons of the algorithms remain as fair as possible.

3.3 Both experiments were coded in C++ (ISO C++11) and compiled with the GNU C++ compiler version 4.8.4. Processing of results was conducted using R. The experiments were executed on a machine with an Intel Xeon with 20 cores running at 2.3 GHZ and with 32GB RAM, under the GNU/Linux operating system. The source code is available on Github at `https://github.com/nathangeffen/pairmatchingalgorithms`.

ATTRACTREJECT experiment

3.4 In the first microsimulation, which we call *ATTRACTREJECT*, every agent has two properties: *attractor* and *rejector*. When executing the simulation, the user sets two constants, *ATTRACTOR_FACTOR* and *REJECTOR_FACTOR*, to values between 0 and 1, such that they add to 1. The distance function is simple and the pseudocode for it is provided in Algorithm 9.

Algorithm 9 Example of a distance function

Parameters a and b are agents with properties *attractor* and *rejector, both in the range* $[0, 1]$. *ATTRACTOR_FACTOR* and *REJECTOR_FACTOR* are user defined positive constants whose sum is 1.

1: **function** DISTANCE(a, b)
2: $\quad attraction \leftarrow ATTRACTOR\_FACTOR * |a.attractor - b.attractor|$
3: $\quad rejection = REJECTOR\_FACTOR * |a.rejector - (1 - b.rejector)|$
4: $\quad$ **return** $attraction + rejection$
5: **end function**

3.5 This distance function captures the issue of attractors and rejectors that affect the difficulty of the pair-matching problem when the triangle inequality is violated. The closer to 1 ATTRACTOR_FACTOR is set and the closer to 0 REJECTOR_FACTOR is set, the more clustered agents suitable for pairing will be. Conversely, the closer to 0 ATTRACTOR_FACTOR, the less useful clustering is, because agents that are clustered together will not be suitable matches.

STIMOD experiment

3.6 The *ATTRACTREJECT* model is not intuitive. The model in the second microsimulation (*STIMOD*), is intended to be more so. It is loosely based on previously published microsimulations (Johnson & Geffen 2016; Hontelez et al. 2013) that model the South African HIV epidemic, but much simpler than these. The pair matching characteristics of the agents are typical for a model of STIs: a list of previous partners, age, sex, sexual orientation and a variable that represents at how much risk of infection their sexual behaviour places them. We have also included an additional attractor factor in the distance function: location, described by x and y co-ordinates on a Euclidean plane, representing how far agents live from one another. An overview of the characteristics of the starting population can be found in Table 1.

3.7 We emphasise that the model and distance function used here are solely for comparing the algorithms. "Production" standard models of the natural world would need to be more complex.

3.8 As with the ATTRACTREJECT model there are user defined constants that specify the weighting of each of these factors in the final distance calculation. The pseudocode for this distant function is provided in Algorithm 10.

Variable	Range	Amount that distance function increases by (higher scores are poorer matches)
Age	Uniformly distributed over the ages 15 to 25	AGE_FACTOR x the difference in age
Sex	50% male/50% female	If mismatch on sexual orientation, add
Sexual orientation	90% of males are men who have sex with men, 90% of females are women who have sex with women	ORIENTATION_FACTOR x difference in (which is always 1 sexual orientation
Risk index	Uniformly distributed over [0-1]	RISK_FACTOR x the absolute difference between two agents
Location	Uniformly distributed over a Euclidean plane with co-ordinates (0-10,0-10)	0.1 x the Euclidean distance (DISTANCE_FACTOR)

Table 1: Agent attributes in STIMOD microsimulation

Algorithm 10 Distance function used in *STIMOD*

Parameters a and b are agents.
Upper case names are user defined constants.
The higher PREV_PARTNER_FACTOR the lower the
probability of previous sexual partners rematching.
The higher AGE_FACTOR, ORIENTATION_FACTOR and RISK_FACTOR the
lower the probability of agents of different age, incompatible
sexual orientation and risk behaviour are to match.
The heterosexual property is 1 if the agent is heterosexual else 0.

function DISTANCE(a, b)
  if a and b have been partners before **then**
    $prev\_partner \leftarrow PREV\_PARTNER\_FACTOR$
  else
    $prev\_partner \leftarrow 0$
  end if
  $age\_diff \leftarrow AGE\_FACTOR * |a.date\_of\_birth - b.date\_of\_birth|$
  if $a.sex = b.sex$ **then**
    $sex\_diff \leftarrow ORIENTATION\_FACTOR * (a.heterosexual + b.heterosexual)$
  else
    $sex\_diff \leftarrow ORIENTATION\_FACTOR * ((1.0 - a.heterosexual) + (1.0 - b.heterosexual))$
  end if
  $risk\_diff \leftarrow RISK\_FACTOR * |a.riskiness - b.riskiness|$
  $distance\_diff = EuclideanDistance(a_x, a_y, b_x, b_y) * DISTANCE\_FACTOR$
  return $prev\_partner + age\_diff + sex\_diff + risk\_diff$
end function

3.9 In all tests, we set PREV_PARTNER_FACTOR to 500, ORIENTATION_FACTOR to 100, AGE_FACTOR to 1, RISK_FACTOR to 1, and DISTANCE_FACTOR to 0.1 (see Table 2). Agents with similar ages, riskiness and location are more likely to partner, so these are attractors. Since 90% of agents are set to heterosexual, having the same sex is usually a rejector. Two agents who have been partners are also very likely to reject each other.

Effectiveness and efficiency measures

Measuring speed

3.10 To compare the speed of the algorithms, we ran 10 simulations of the STIMOD model for each algorithm using 20,000 agents. Each simulation was executed for 20 iterations (i.e. time steps). In other words, every algorithm was executed 200 times.

3.11 We are also interested in how algorithms perform as the number of agents increases, or as the size of k increases for the algorithms that depend on this parameter. For the CSPM algorithm we are also interested in how the number of clusters affects speed. Since this algorithm achieved the best balance of effectiveness and speed in our tests, we ran tests on it varying the number of agents up to 10 million. We also ran tests varying k from 50 to 500, and tests varying c from 50 to 500.

Measuring effectiveness

3.12 We ran the following tests to compare the effectiveness of the algorithms in the STIMOD model:

- STIMOD with 5,000 agents, simulated 36 times with different random number seeds, with each simulation having 20 iterations. In other words every algorithm executes 720 times. Algorithms were compared against Blossom V for effectiveness.
- STIMOD with 20,000 agents, simulated 36 times, with each simulation having 20 iterations.
- Using CSPM with 20,000 agents, we varied k, the number of adjacent agents to consider, from 50 to 1,000 stepping up by 50 on each execution of the microsimulation, while holding the number of clusters constant, to see the effect on the mean ranking and distance. We then similarly varied the number of clusters, c, initially setting it to 1 and then from 50 to 1,000 stepping up by 50 on each execution of the microsimulation, while holding k constant at 200.

3.13 We ran the following tests to compare the effectiveness of the algorithms in the ATTRACTREJECT model:

- ATTRACTREJECT microsimulation with 5,000 agents, with each simulation executed 20 times with different random seeds with the following combinations of ATTRACTOR_FACTOR and REJECTOR_FACTOR respectively:
  - 0.0 and 1.0
  - 0.25 and 0.75
  - 0.5 and 0.5
  - 0.75 and 0.25
  - 1.0 and 0.0
- ATTRACTREJECT microsimulation with 20,000 agents, with each simulation executed 20 times with the above combinations of ATTRACTOR_FACTOR and REJECTOR_FACTOR respectively.

Best-possible result

3.14 As mentioned above, the *mean distance* and *mean ranking* measures can only be used for the relative comparison of the different algorithms. Furthermore, the set of partnerships with the minimum distance possible is not known a priori. This can be calculated using the Blossom V algorithm. A prerequisite for the application of Blossom V is the creation of a fully connected undirected graph where the vertices represent the agents and the edges represent the distances between them, a process with a quadratic speed increase with the number of agents being matched. The Blossom V algorithm itself is not stochastic and is very slow, with speed worse than cubic with the number of agents being matched. It is therefore only used for reference purposes in the simulations of 5,000 agents.

Mean of mean distances

3.15 In the first measure we calculate mean distance of all the pairings for a single execution of the pairing algorithm. Since every algorithm is executed multiple times, we calculate the mean of these means. The measure of effectiveness for an algorithm is then the mean of its mean distances divided by the mean of the mean distances calculated by Blossom V.

3.16 However, when we use 20,000 agents Blossom V is too slow to use. On state of the art consumer hardware (an Intel Xeon running 20 i7 cores) a single execution of Blossom V over 20,000 agents, including creating the graph, takes over 2 hours — and we have had to run hundreds of simulations. Instead, effectiveness of the algorithm with the lowest mean of mean distances (usually BFPM) is assigned the value of 1. The effectiveness of the remaining algorithms is the ratio of their mean of mean distances to this algorithm's mean of mean distances.

Mean of mean rankings

3.17 In the second measure for every agent we calculate the distance to every other agent, creating a fully connected undirected graph. For each agent, a, we can then order every other agent by its suitability as a partner to a. The ranking of a pairing for a given agent is its partner's place in this ordering. If an agent chooses its ideal partner the rank is 0. If it chooses the least desirable partner, the rank is $n - 2$, where n is the number of agents being matched. We can then calculate the mean ranking over all the agents being matched for a single execution of an algorithm. We compare the algorithms by calculating the mean of its mean rankings over many executions. The lower the mean of mean ranking the better the algorithm has done.

3.18 Effectiveness is calculated analogously to the way it is done for the mean of mean distances measurement. Likewise with 5,000 agents effectiveness is established with Blossom V (although Blossom V guarantees lowest mean distance and not lowest mean ranking, in practice it generally returns the lowest ranking). With 20,000 agents, the method of assigning 1 to the effectiveness of the best algorithm is used.

Results

Speed tests

4.1 Table 2a lists the results of the speed tests. The fastest algorithm, RPM, has an average speed more than 5,700 times faster than the slowest algorithm, BFPM. The former completes the entire pair-matching event (i.e. match every agent for an iteration) in less than a millisecond on average, while the latter takes over three seconds on average. CSPM, RKPM, WSPM and DCPM have a mean speed between 27 and 39 milliseconds, and are all approximately an order of two magnitudes faster than BFPM.

Algorithm	Mean speed (ms)	Speedup (BFPM ref)
RPM	0.6	5,715
RKPM	27	115
CSPM	29	109
WSPM	29	109
DCPM	39	81
BFPM	3,143	1

(a) Comparison of algorithms by speed. Each algorithm was executed 20 times per simulation, and each simulation was run 10 times. Each simulation consisted of 20,000 agents. For algorithms that take a k parameter, k was set to 200. For CSPM, c was set to 100.

Number of agents	Mean speed (ms)
20,000	29
50,000	53
100,000	109
500,000	608
1,000,000	1,263
5,000,000	6,842

(b) Speed of CSPM with increase in agents. The algorithm was executed once per simulation, and each simulation was run 10 times. The k parameter was set to 200, and c was set to 100.

k	Mean speed (ms)
50	301
100	410
200	606
300	808
400	1,006
500	1,230

(c) Change of speed of CSPM as k increases. The algorithm was executed once per simulation, and each simulation was run 10 times. 500,000 agents were used.

Table 2: Speed comparisons of algorithms

	Distance				Rank				Speed
Algorithm	Mean	SD	Median [IQR]	Effectiv.	Mean	SD	Median [IQR]	Effect.	Time
Blossom V	2.7	14.3	0.32 [0.27;0.41]	1.00	40	225	1.7 [0;5.4]	1.00	6.5 minutes
BFPM	2.8	19.2	0.32 [0.26;0.44]	1.03	63	328	1.6 [0.0;6.7]	1.6	345 ms
CSPM	3.3	17.1	0.46 [0.34;0.84]	1.21	138	507	15.6 [3.1;74.9]	3.5	15 ms
WSPM	3.4	18.3	0.79 [0.48;1.38]	1.25	180	361	64.7 [20.1;187.6]	4.6	14 ms
RKPM	3.4	18.3	0.79 [0.48;1.39]	1.25	181	362	65.5 [20.4;188.7]	4.6	14 ms
DCPM	3.8	20.5	0.57 [0.37;0.98]	1.40	125	347	29.7 [9.4;91.2]	3.2	17 ms
RPM	105.6	93.1	104.17 [4.99;203.29]	38.6	2501	1444	2503.4 [1251.7;3752.2]	63	0.3 ms

Table 3: Distance and rank results on 5,000 agents for STIMOD. The mean, standard deviation, median, IQR and time columns are the means of these values over all executions. The effectiveness is the ratio of the algorithm's mean distance to Blossom V's.

4.2 As the number of iterations increases, thereby creating more previous partners, the mean speed of the algorithms (except RPM) increases slightly, because the distance calculation has to search for more previous partners. For example BFPM increased from a mean of over 2 seconds on the first iteration of a simulation to 4.6 seconds on the 20th, and CSPM increased from 19 to 43 milliseconds.

4.3 The speed of the CSPM algorithm increased linearly with the number of agents, as Table 2b indicates. This is expected because even though our analysis above shows that its speed increases linearithmically with the number of agents being matched, this is due to its sorting operation, which is a small contributor to the overall time of the algorithm with these small numbers of agents.

4.4 Modifying the number of clusters also shows a linear effect on the speed of the algorithm. This is expected because modifying the value of c in Algorithm 7 increases the number of iterations of the for loop at line 8 although it correspondingly decreases the number of agents to be shuffled at line 11.

4.5 The speed of the CSPM algorithm also increased linearly with the size of k, as Table 2c indicates. This too is expected as increasing k proportionately increases the number of comparisons made for each agent a with the following agents in the array.

Effectiveness tests

4.6 Table 3 shows the results of the comparison of the algorithms on the STIMOD microsimulation for 5,000 agents. With this few agents, it is still feasible to identify the theoretically lowest mean distance using the Blossom V algorithm. In these tests CSPM, followed by DCPM algorithm, offers the best trade-off of speed and effectiveness. Nevertheless RKPM and WSPM have similar effectiveness and speeds.

4.7 Table 4 shows the distance and ranking results respectively of the ATTRACTREJECT microsimulation. In contrast to the STIMOD microsimulation the DCPM algorithm is entirely unsuited to this distance function and performs poorly, although much better than RPM. The remaining algorithms improve their relative effectiveness as ATTRACTOR_FACTOR rises to 1 and REJECTOR_FACTOR declines to 0.

4.8 Interestingly CSPM exceeds the effectiveness of BFPM for higher values of ATTRACTOR_FACTOR. A possible explanation for this is that agents at the front of the array processed by BFPM will be well-matched, but agents nearer the back of the array have fewer partner options that give low distance measurements. CSPM, on the other hand, by first clustering, ensures the agents at the back of the array are nearer likely partners.

4.9 This raises a key weakness of BFPM, CSPM, WSPM, RKPM and possibly even DCPM: the distribution of their matches varies greatly in quality between the front and back of their arrays, and there is room for further research as to how they can be adapted to have a more equal distribution of quality.

4.10 Consider Table 6 which shows the mean median ranking, mean interquartile range (IQR) and mean standard deviation across the ATTRACTREJECT simulations with ATTRACTOR_FACTOR set to 0.5. The median ranking for BFPM is much lower than the mean ranking shown in Table 4. It is even better than the median ranking of Blossom V. In fact at least 75% of the rankings are better than the mean ranking, implying that some very poor rankings toward the back of the agent array bring down the mean ranking. This skewed quality is also a problem for CSPM but, as shown by its smaller standard deviation and median ranking closer to its mean ranking, not as profoundly as it is for BFPM, WSPM, RKPM and DCPM (see also Figure 2).

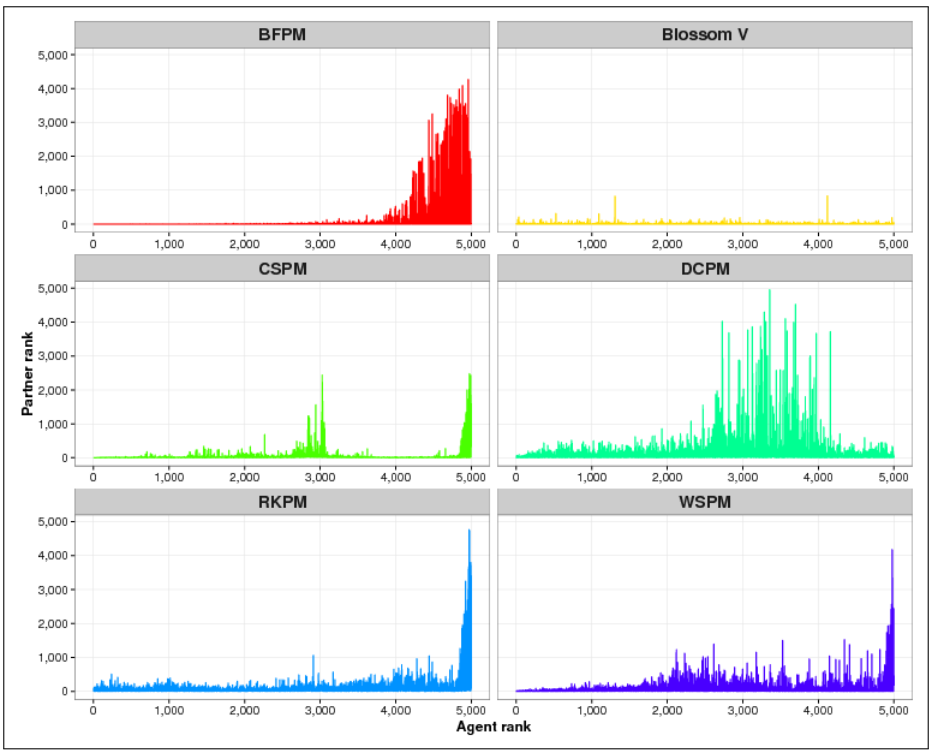

Figure 2: Distribution of rankings for single execution of each algorithm
This figure shows the partner rank for a single run of each algorithm (except RPM). Notice how Blossom V's poorly ranked agents are distributed uniformly across the array, but the poorly ranked agents for the other algorithms are distributed towards the back of the array.

	Attractor:	0	0.25	0.5	0.75	1
	Rejector:	1	0.75	0.5	0.25	0
	Algorithm	Mean (ratio to best)				
Distance	Blossom V	0.006 (1)	0.009 (1)	0.010 (1)	0.008 (1)	0.0002 (1)
	BFPM	0.008 (1.3)	0.015 (1.6)	0.016 (1.6)	0.012 (1.6)	0.0007 (3.5)
	CSPM	0.016 (2.6)	0.018 (1.9)	0.018 (1.8)	0.015 (1.9)	0.0004 (2.0)
	WSPM	0.016 (2.6)	0.032 (3.3)	0.033 (3.4)	0.026 (3.3)	0.0025 (12.2)
	RKPM	0.015 (2.5)	0.036 (3.8)	0.037 (3.9)	0.030 (3.8)	0.0032 (15.8)
	DCPM	0.018 (2.9)	0.041 (4.3)	0.044 (4.6)	0.039 (4.9)	0.0135 (65.9)
	RPM	0.334 (53.6)	0.334 (35.3)	0.334 (34.3)	0.334 (42.3)	0.3337 (1,633)
Rank	Blossom V	61 (1)	8 (1)	5 (1)	4 (1)	1 (1)
	BFPM	63 (1.1)	47 (6.1)	37 (7)	25 (5.8)	5 (5.4)
	CSPM	148 (2.4)	39 (5.1)	22 (4)	15 (3.5)	3 (2.9)
	WSPM	147 (2.4)	82 (10.5)	64 (12)	52 (12.2)	23 (23.3)
	RKPM	139 (2.3)	93 (12)	76 (14)	60 (14.2)	30 (30.5)
	DCPM	161 (2.7)	129 (16.5)	112 (21)	112 (26.4)	127 (127.4)
	RPM	2501 (41)	2502 (321)	2502 (472)	2501 (588)	2499 (2499)

Table 4: Distance and rank results for 5,000 agents for ATTRACTREJECT. Ranks and ratios are rounded.

4.11 Table 4 also shows that when REJECTOR_FACTOR is much larger than ATTRACTOR_FACTOR, then as expected RKPM is as effective as CSPM, DCPM and WSPM.

4.12 When the number of agents in the STIMOD model is increased to 20,000 we do not see any substantial changes in effectiveness across the algorithms compared to when 5,000 agents are used, except that CSPM clearly outper-

	Distance				Rank				Speed
Algorithm	Mean	SD	Median [IQR]	Effect.	Mean	SD	Median [IQR]	Effect.	Time
BFPM	1.7	14.9	0.2 [0.2;0.3]	1	130	971	1.5 [0;6]	1	7 seconds
CSPM	2.0	15.3	0.3 [0.2;0.4]	1.2	276	1,315	14.6 [3.6;62.7]	2.1	51 ms
DCPM	2.7	15.2	0.6 [0.4;1.1]	1.6	520	1,207	141 [44.6;448.9]	4	61 ms
RKPM	2.8	13.6	0.8 [0.5;1.5]	1.7	865	1,504	312 [96.3;910]	6.7	47 ms
WSPM	2.8	13.7	0.8 [0.5;1.5]	1.7	869	1,508	312.6 [95.6;916.9]	6.7	51 ms
RPM	104.7	91.3	104.08 [5;203]	63	10,000	5,773	10,001 [5,002;14,999]	77	2 ms

Table 5: Distance and rank results for 20,000 agents for STIMOD.

Algorithm	ATTRACTREJECT		STIMOD			
	Median rank [IQR]	SD	Rankings			
			Mean	Best	Worst	SD
Blossom V	1.4 [0;5.1]	17	–	–	–	–
BFPM	1 [0;5.3]	225	130	43	254	51
CSPM	7.4 [2.7;16.1]	14	276	66	537	107
WSPM	22.9 [7.2;62.3]	165	869	354	1684	276
RKPM	36 [13.8;80]	190	865	345	1673	275
DCPM	44 [15.5;112]	250	520	283	957	113
RPM	2,502 [1,259;3,746]	1,440	10,000	9,797	10,192	56

Table 6: Selected statistics for 5,000 agents for ATTRACTREJECT with attractor and rejector set to 0.5, and volatility of the results as demonstrated for STIMOD with 20,000 agents

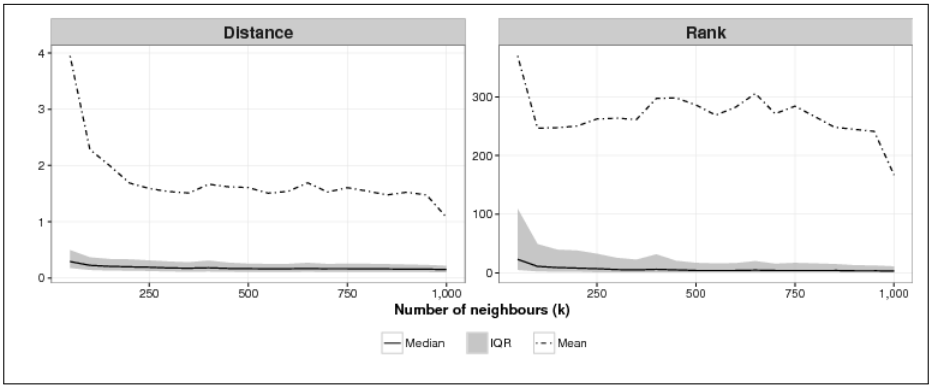

Figure 3: Finding the ideal number of neighbours to minimize distance and rank (displayed on the respective y-axes).

forms DCPM on both mean distance and mean ranking. Whether this is due to something qualitatively different happening as the number of agents increases or due to random fluctuations of the results is unclear. The results from simulation to simulation are indeed volatile as Table 6 shows, with BFPM being the most stable algorithm followed by CSPM.

4.13 We also attempted to find the ideal values of k, the number of neighbours, and c, the number of clusters for the CSPM algorithm. As Figure 3 shows, the relationship between the value of k and effectiveness is unclear for this application. For $k = 50$, the lowest value of k we tried, the mean of the mean rankings over 40 simulations was 370. For $k = 1,000$, the highest value of k we used, the mean ranking was 167, the lowest, but as the graph shows for all values in between, there is no discernible pattern. On other test runs we got different results where $k = 1000$ was not the best and $k = 50$ was not the worst. Results are domain dependent, and modellers who use CSPM will have to experiment to find the best value of k.

4.14 We are unsure what the relationship between c and effectiveness is as Figure 4 shows. For $c = 1$ (essentially an array sorted on the cluster function), the mean of the mean rankings is poor: 791. For all values of c tested from 50 to 1,000 we were unable to identify a discernable difference in effectiveness across the simulations. As with

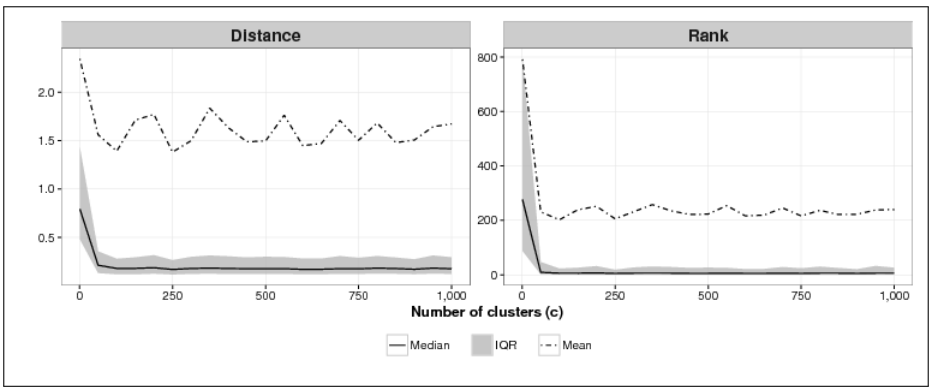

Figure 4: Finding the ideal number of clusters to minimize distance and rank (displayed on the respective y-axes).

the ideal value of k modellers who use CSPM will have to experiment to find the best value of c.

Discussion

5.1 We have used two microsimulations to show how four algorithms CSPM, DCPM, WSPM and RKPM can match agents much better than chance (RPM) yet much faster than the BFPM algorithm and Blossom V — which finds a perfect set of pairs without desired stochasticism.

5.2 CSPM performed consistently well across our tests. For example in the STIMOD microsimulation it only had a mean ranking four times worse than Blossom V, whereas random matching was more than 61 times worse. When the number of agents was increased to 20,000, a number beyond which it is practical, at least on our hardware, to check effectiveness against Blossom V, CSPM's mean ranking was only half as poor as that of BFPM, more than double that of the next best algorithm DCPM and 45 times better than random matching. Yet it executed on average in just over 50 milliseconds versus 7 seconds for BFPM.

5.3 However, while CSPM has done well here, it is likely that there are other applications where the other fast approximation algorithms, DCPM, WSPM and RKPM, will be equally good or better choices. We are currently exploring an application where DCPM appears to be better. In this application, the agents fall neatly into a prespecified set of buckets, which is ideal for DCPM.

5.4 As the tests with the ATTRACTREJECT microsimulation show, when rejector attributes of agents dominate, the RKPM algorithm might as well be used: it is faster than CSPM, trivial to implement and will produce results that are at least as good as CSPM because clustering achieves nothing in such pair-matching scenarios.

5.5 CSPM requires the user to decide the values of two parameters, k and c. We cannot currently offer good heuristics for choosing these values other than to suggest preliminary empirical tests for the specific microsimulation being used.

5.6 We have not yet found a situation where WSPM outperforms all of CSPM, DCPM and RKPM. It is possible that it is simply an inferior algorithm, but it is not inconceivable that a useful microsimulation application exists in which WSPM makes sense as the pair-matching algorithm.

5.7 One serious limitation of BFPM, CSPM, RKPM and WSPM, at least in the applications considered here, is that the quality of their matches are skewed to the front of the array of agents that they process. After shuffling, agents near the front of the array will tend to be matched with partners with smaller distances than the agents at the back of the array. This is reflected in the fact that the mean of the median rankings — and even the 75% quartile — is smaller than the mean of the mean rankings in the simulations where we examined this. Further research is required to improve one or more of CSPM, RKPM or WSPM to reduce this skewed matching.

5.8 While we are interested in microsimulations that simulate sexual pairing, it is possible there are uses of these algorithms in other microsimulations which require agent interaction. However, for such applications the algorithms would need to be evaluated to see if they produce results that compare well to observed data.

5.9 Further research is also needed to see how useful an algorithm such as CSPM is in the context of modelling sexually transmitted infections. For example, the most cited model of the South African HIV epidemic is a deterministic equation-based one that implicitly assumes people are randomly matched (Granich et al. 2009). Other

deterministic models of the epidemic make more complex assumptions, e.g. by dividing the population into a small number of sexual groups (Johnson 2014). It would be interesting to compare the outputs of a full-fledged microsimulation of an STI epidemic using random matching versus CSPM. If the results of such a microsimulation are similar irrespective of the pair-matching algorithm then the speed of random matching means it is the better choice. If however, the results are vastly different, it has implications for how we model, not only using agents, but also with deterministic differential-equation models.

References

Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R. & Wu, A. Y. (1998). An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, *45*(6), 891–923

Beis, J. S. & Lowe, D. G. (1997). Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *CVPR '97 Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition*

Bouffard, N., Easther, R., Johnson, T., Morrison, R. J. & Vink, J. (2001). Matchmaker, matchmaker, make me a match. *Brazilian Electronic Journal of Economics*, *4*(2)

Cook, W. & Rohe, A. (1999). Computing Minimum-Weight Perfect Matchings. *INFORMS Journal on Computing*, *11*(2), 138–148

Deissenberg, C., van der Hoog, S. & Dawid, H. (2008). EURACE: A massively parallel agent-based model of the European economy. *Applied Mathematics and Computation*, *204*(2), 541–552

Eaton, J. W., Johnson, L. F., Salomon, J. A., Bärnighausen, T., Bendavid, E., Bershteyn, A., Bloom, D. E., Cambiano, V., Fraser, C., Hontelez, J. A. C., Humair, S., Klein, D. J., Long, E. F., Phillips, A. N., Pretorius, C., Stover, J., Wenger, E. A., Williams, B. G. & Hallett, T. B. (2012). HIV treatment as prevention: Systematic comparison of mathematical models of the potential impact of antiretroviral therapy on HIV incidence in South Africa. *PLoS medicine*, *9*(7), e1001245

Granich, R. M., Gilks, C. F., Dye, C., De Cock, K. M. & Williams, B. G. (2009). Universal voluntary HIV testing with immediate antiretroviral therapy as a strategy for elimination of HIV transmission: A mathematical model. *The Lancet*, *373*(9657), 48–57

Gras, R., Devaurs, D., Wozniak, A. & Aspinall, A. (2009). An individual-based evolving predator-prey ecosystem simulation using a fuzzy cognitive map as the behavior model. *Artificial Life*, *15*(4), 423–463

Gray, R. T., Hoare, A., McCann, P. D., Bradley, J., Down, I., Donovan, B., Prestage, G. & Wilson, D. P. (2011). Will changes in gay men's sexual behavior reduce syphilis rates? *Sexually transmitted diseases*, *38*(12), 1151–1158

Hontelez, J. A. C., Lurie, M. N., Bärnighausen, T., Bakker, R., Baltussen, R., Tanser, F., Hallett, T. B., Newell, M.-L. & de Vlas, S. J. (2013). Elimination of HIV in South Africa through expanded access to antiretroviral therapy: A model comparison study. *PLoS medicine*, *10*(10), e1001534

Indyk, P. & Motwani, R. (1998). Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, (pp. 604–613). New York, NY, USA: ACM

Johnson, L. (2014). *THEMBISA version 1.0: A model for evaluating the impact of HIV/AIDS in South Africa*. Centre for Infectious Disease Epidemiology and Research working paper

Johnson, L. F. & Geffen, N. (2016). A Comparison of Two Mathematical Modeling Frameworks for Evaluating Sexually Transmitted Infection Epidemiology. *Sexually Transmitted Diseases*, *43*(3), 139–146

Knuth, D. E. (1997). *The Art of Computer Programming, Volume 2: (2Nd Ed.) Seminumerical Algorithms*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc.

Knuth, D. E. (1998). *The Art of Computer Programming, Volume 3: (2Nd Ed.) Sorting and Searching*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc.

Kolmogorov, V. (2009). Blossom V: A new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, *1*(1), 43–67

Larose, D. T. & Larose, C. D. (2014). *k-Nearest Neighbor Algorithm*, (pp. 149–164). John Wiley & Sons, Inc.

Levitin, A. (2011). *Introduction to the Design and Analysis of Algorithms*. Boston: Pearson, 3 edition edn.

Macy, M. W., & Willer, R. (2002). From factors to actors: Computational sociology and agent-based modeling. *Annual Review of Sociology*, *28*(1), 143–166

Pelillo, M. (2014). Alhazen and the nearest neighbor rule. *Pattern Recognition Letters*, *38*, 34 – 37

Scholz, S., Surmann, B., Elkenkamp, S., Batram, M. & Greiner, W. (2016). Creating populations with partnerships for large-scale agent-based models – A comparison of methods. In *Proceedings of the Summer Computer Simulation Conference (SCSC 2016)*, Simulation Series, (pp. 351–356). Red Hook, NY, USA: Curran Associates, Inc.

Weisstein, E. W. (2016). *Metric Space*. From MathWorld – A Wolfram Web Resource. `http://mathworld.wolfram.com/MetricSpace.html`. Accessed: 16 November 2016

Zinn, S. (2012). A Mate-Matching Algorithm for Continuous-Time Microsimulation Models. *International Journal of Microsimulation*, *5*(1), 31–51