用于匹配问题的新算法

卢嘉伟和马克·威尔逊

2017年3月14日

抽象

标准的双面和单面匹配问题，和密切相关的学校选择问题，已经从公理视角被广泛研究。少数的算法占主导地位的文献。对于双边匹配，使用**Gale-Shapley**算法**;** 单向匹配，（随机）串行专用和概率串行规则**;** 为学校选择，**Gale-Shapley**和波士顿机制。这些算法的主导地位的主要原因是它们在效率和战略性概念方面的良好（最坏情况）公理行为。然而，如果我们将焦点转移到公平，社会福利，不兼容的公理之间的权衡和平均案例分析，这些算法是最优的还远远不清楚。

我们调查新的算法，其中几个没有出现（在我们的知识）在文学前。我们给出一个统一的演示，其中用于双侧匹配的算法以系统的方式产生单侧匹配算法。除了公理性质，我们使用理论和计算方法研究代理福利。我们发现，某些新算法值得在某些应用中考虑。特别是，当考虑真实偏好下的福利时，一些新的算法胜过经典的算法。

**1** 介绍

标准的双边和单方面匹配问题，以及密切相关的学校选择问题，从公理的角度被广泛研究。少数算法支配文献。对于双边匹配，使用**Gale-Shapley**算法**;** 单向匹配，（随机）串行排序和概率串行规则**;** 为学校选择，**Gale-Shapley**和波士顿机制。

这些算法的主导地位的主要原因是它们在效率和战略性概念方面的良好的公理行为。然而，如果我们将焦点转移到公平，社会福利或不兼容的公理之间的折衷，这些算法是最优的还远远不清楚。

**1.1** 我们的贡献

在第**3**节中，我们介绍了几个（在我们看来）单向匹配的自然算法，其中几个在文献中没有出现（我们的知识）。我们给出了使用**Gale-Shapley**算法**[]**的专门化的双向匹配的一致性推导，其包括在统一框架中的公知算法串行专家**[]**和朴素波士顿**[]**。在第**4**节，除了公理性质，如效率和战略性，我们调查福利损失使用计算方法。我们发现，在真实的偏好下，一些新的算法明显优于传统的算法。特别是，我们推荐一些新的算法为一些应用程序。

**2** 定义和术语

令**A = {a1**，**...**，**an}**是一组有限的代理，**O = {o1**，**...**，**om}**一组有限的项。

一般来说，项目和代理的数量可能不相等。我们集中在本文中的情况**m = n**。更普遍的情况涉及相当复杂：在**O**或**A**通向**strategyproofness**的不同的概念的子集分配喜好，例如不同的方式。但是，为了在一般情况下工作，可以轻易地修改算法。

在标准的双边匹配问题中，**O**的每个元素具有完全严格的优先顺序。对于**A**的元素，反之亦然，而对于单侧匹配，仅需要后者的信息。对于学校选择，项目优先于代理的顺序取决于代理对项目的偏好 **-** 双方绝对不是独立的（学校通常必须允许有资格的学生，只要有能力，并且只有当必须打破领带）。我们旨在统一这三种情况，并限制为严格线性顺序的情况。

令**L**（**O**）（分别为**L**（**A**））表示**O**（或代理）上的所有严格线性阶的集合。偏好简档是函数**f**：**O**↔**A**。偏好简档是函数**f**：**O**↔**A**。令**S**是所有双随机**n×n**矩阵的集合，行由代理和列按项目索引。随机分配是**S**的元素。匹配问题是简单地输出给定输入简档的匹配。比例分配是每个矩阵项等于**1 / n**的随机分配。

当然，有**n**！离散分配和发现一个是微不足道的。关键是找到一个具有期望属性的。如果没有其他分配改进某些代理的结果并且不使任何代理恶化它，则离散分配是有效的。如果每个以肯定概率发生的匹配是有效的分配，则用于随机分配的算法是事后有效的。

我们现在回顾一些标准的算法从文学。

如果两个算法对于每个输入产生相同的输出，则相同匹配问题的两个算法是等效的。所研究的所有算法是匿名的，这意味着玩家的排列导致分配的相同排列。换句话说，只有偏好，而不是代理的身份。当使用离散赋值时，这意味着当我们有两个具有相同偏好的代理时，必须羡慕对方的赋值。更强的条件是对称性（也称为**“**等于的平等处理**”**），其表示具有相同偏好的代理接收相同的赋值。显然，这只能在随机分配的框架中得到满足。

使用代理的固定初始顺序并产生匹配的每个算法可以产生产生随机匹配的算法，简单地通过对初始顺序进行随机化。这通常根据均匀分布进行，以便保持试剂之间的对称性。因此，第**3**节中讨论的每个算法都具有随机化版本，我们通过在其名称前面加上**“R”**来表示。

**2.1** 单边算法

单向匹配是指忽略项目对代理的偏好的情况。最常被讨论的算法是串行专有技术**[]**。

示例**2.1**。（**SerialDictatorhip**）

修正**A**上的任意线性排序。关于此排序的串行专用算法（**SD**）

将项目分配给代理，如下：在步骤**i**，向代理分配其尚未分配给先前代理的最优选项目。

如上所述，随机化版本表示为**RSD**。串行专制满足重要的公理性质，如事后效率和战略性（所有公理性质在第**4**节中定义和讨论）。

下一个算法由**Bogolmanaia**和**Moulin [BM01]**提出。除了事后有效率之外，它是有效的，并且具有弱的战略性性质。它可以使用蛋糕吃类比来描述。

实施例**2.2**。（概率序列）概率序列规则是固有地随机化的，并且生成如下的随机分配。我们将项目**j**的分数**x**的分配解释为代理**i**，意味着我接收**j**的分数**x**（换句话说，我们假设项目是无限可分的）。所有代理同时开始以单位速度**“**吃**”**，每个时间点的每个代理从其最优选的项目中选择尚未完全消耗的那些，在终止时，我们具有随机分配。

与单边匹配密切相关的问题是房地产市场问题**[SS74]**。不同之处在于，假定每个代理具有初始分配项，并且我们寻求用于找到在某种方式上最优的分配的方法。

实施例**2.3**。（顶部交易周期）当每个代理被认为最初分配了整个项目时，代理可以如下在他们之间进行交易。每个代理指向**i**的首选项列表顶部的当前拥有该项的代理。由于有限性，并且由于每个节点具有偏度**1**，所以该有向图必须包含一个周期。根据循环中的弧重新分配项目，并从进一步考虑中删除这些代理和项目。重复（如有必要，使用指向下一级别首选项的指针），直到没有项目**/**代理程序保留。这个**TTC**算法**[SS74]**，由**Shapley**和**Scarf**归因于**David Gale**，总是产生一个有效的离散赋值，机制是战略性和个别理性。此外，算法在多项式时间中运行。

实施例**2.4**。（**TTC**的样本执行）考虑其中代理**1**和**2**具有优先级**b> c**并且代理**3**具有优先权**b> a> c**。分配**1**：**c**，**2**：**b**，**3**：**a**不是事后有效的，因为**2**和**3**可以交易到它们的互惠。在第一轮**TTC**中，代理**1**和**2**指向代理**3**，而代理**3**指向代理**2.**代理**2**和**3**之间有一个周期。代理在周期中交换，并从考虑中移除。代理**1**然后在下一轮中指向自身，沿着周期交换，保持与项**c**，并且从考虑中移除。没有任何代理，**TTC**算法停止。输出是事后有效分配**1**：**c**，**2**：**a**，**3**：**b**。

当使用**TTC**时，我们有自由选择初始分配。例如，随机选择并运行**TTC**产生与**RSD [AS98]**等价的算法，并且当在比例禀赋**[Kes09]**上运行时，**TTC**对交易单位份额的适应产生等同于**PS**的算法。

我们发现在第**3**节中的一些算法的输出上运行**TTC**是有用的（满足事后效率增益的算法没有受益于运行**TTC**，其由于没有周期而立即终止）。所得到的组合算法（表示为**XG**，其中**X**是基本算法的名称）是事后有效的，并且似乎具有比原始算法明显更好的整体性能。

**2.2** 双边算法

在这种情况下，代理对项目具有完全严格的偏好，反之亦然。最知名的算法属于延迟接受算法类。项目和代理暂时匹配，但这些**“**约定**”**可能会中断。事实上，在算法过程中，每个项目可以附加到最多**n**个代理。

在**Gale-Shapley**算法**[GS62]**中，代理反过来接近先前未被批准的项目，他们喜欢他们当前分配的项目（每个代理优选每个项目没有项目，并且每个项目优选每个代理不被代理持有）。如果当前提议代理优于当前与项目匹配的代理，则项目将拒绝其对于提议代理的当前合作伙伴。没有代理可以接近已经拒绝的项目（这种方法将导致上述规则的另一个拒绝）。这确保在最多**n2**个提议之后终止。

实施例**2.5**。这来自**[GS62]**中的实施例**2**。假设提议者的偏好如下：

1: *a*>*b*>*c*>*d* 2: *a*>*d*>*c*>*b* 3: *b*>*a*>*c*>*d* 4: *d*>*b*>*c*>*a*

*and the proposees’ preferences are given by*

*a*: 4>3>1>2 *b*: 2>4>1>3 *c*: 4>1>2>3 *d*: 3>2>1>4

最终匹配为1：c，2：d，3：a，4：b。 在执行算法期间提出了9个建议。

Gale-Shapley算法具有众所周知的属性，即输出匹配是稳定的，意味着没有不匹配（代理，项目）对，每个对都彼此偏爱他们当前的伙伴。 此外，输出匹配对于提议者是最佳的，意味着每个提议器接收在稳定匹配中可以接收的最佳可能项目。 因此，算法的输出不依赖于代理提出的建议的顺序。 注意，相比之下，第3节中的算法的输出将强烈依赖于提议的顺序。

我们可以通过强制项目具有特定（虚构）偏好来从双侧产生单侧匹配算法。 我们在第3节系统地使用这个想法。

2.3 学校选择算法

在单面和双面匹配之间的中间情况，这对于后来很重要，是学校选择的情况。 Gale-Shapley算法适用于代理数量超过项目数量的情况，只要物品（学校）具有一定数量的代理人（学生）的能力。 如果学生有能力剩余，或者学生优于已经暂时接受的学生，则学校暂时接受学生的提议。 每个学校的容量为1，学校和学生的数量相等的情况是第2.2节所述的情况。

还有其他算法为学校选择使用立即接受。 在这种情况下，每个学生首先申请她的第一选择学校。 每个学校根据能力排名申请人和选择尽可能多的人。 学生不接受已经然后申请他们的第二选择学校等。该描述隐含地使用由所有不匹配的代理同时提出，并称为波士顿机制[AS03，MS14]。 还有一个顺序版本，其中一次一个代理一个提议。 在这种情况下，提案的顺序显然改变了最终分配，因为不能破坏任何参与。

波士顿机制的特殊情况是每个学校具有容量1并且代理和项目的数量相等是如上定义的双侧匹配算法。 对于与实例2.5相同的输入，使用1,2,3,4顺序提供的最终分配是1：a，2：d，3：b，4：c。 使用同时报价的最终分配是1：a，2：c，3：b，4：d。

4 新算法

对于其余的分析，我们通过放宽双侧算法构造单侧匹配算法。 Gale-Shapley算法基于代理和项目的固定偏好，使用一系列建议生成稳定匹配。 给定代理对项目的偏好作为对单向匹配问题的输入，我们构造代理上的项目的虚构偏好。 例如，我们可以假设所有项目具有固定的公共偏好。

命题3.1。 标准化。

证明。 假设所有项目具有与代理相同的偏好顺序，在不失一般性的情况下，我们写1> 2> ...> n。 我们声称Gale-Shapley将输出与代理订单1,2，…，n相同的序列专有权的输出。

证明是归纳的。 基本情况是代理1将得到他的第一选择与GS。 这是非常真实的，因为代理1将建议其最优选的项目，并且由于每个项目优选代理1到任何其他代理，它们不能以后被拒绝。 因此，代理1将在GS或SD下分配相同的项目。

GS的稳定性意味着对于代理我想要多于它们被分配的项目的每个项目，该项目必须由该项目排名更高的代理持有。 如果每个代理在我之前获得其根据SD选择，代理我将最终提出在SD下的选择。 因为每个代理人在i之后被所有项目排在i之下，他们不能导致该项目拒绝i。 因此，代理人i将在GS或SD下具有相同的项目。

我们可以用（同时）朴素波士顿算法做同样的事情。 给定一个单侧匹配的实例，我们创建虚拟偏好，其中每个项目具有相同的偏好，例如1> 2> ...> n。 得到的算法称为单边朴素波士顿算法。 注意，如果我们确保代理顺序是1,2，…，n，但是否则顺序和同时形式通常会不同，我们也可以顺序解释这个算法。

下面，我们概括了这种虚拟偏好方法，允许每个项目动态构建虚拟偏好，使用一些固定的规则，仅基于从代理接收的建议序列。 回想一下，任何提案顺序对于具有固定首选项的Gale-Shapley算法都有相同的结果，但是正如我们在下面看到的，在我们的轻松设置中不是这样。

定义3.2。 在本文的其余部分中，为了说明和比较算法，我们使用我们所谓的标准简档，其中代理1,2和3具有偏好a> b> c> d，代理4具有偏好b> a > c> d。

3.1“永久记忆”案例

由于代理人的建议的顺序影响项目的偏好，建议的顺序影响最终分配。 这有两个后果。 第一是处理被拒绝的代理人。 在代理被拒绝后，或者因为项目优选其当前代理或者断开其暂时的接合，可能不一定是下一个代理提出。 我们考虑两种可能性：堆栈（被拒绝的代理到堆栈的顶部）或队列（被拒绝的代理到队列的后面）。 另一个结果是，如上所述，代理的初始顺序对最终分配有影响。 为了分析的目的，算法将修正任意的初始顺序。

我们考虑动态构建首选项的两个规则。 这些是早期提议偏好（或Accept-First）和晚期提议偏好（或Accept-Last）。 Accept-First表示接受项目的第一个代理，并拒绝后续提议。 使用接受最后，如果尚未由新提议者持有，则项目总是断开对新提交者的约束。 关于通常用于描述Gale-Shapley算法的婚姻解释，对于Accept-First算法，被提议者忠实地忠于他们的第一个请求者，而对于Accept-Last算法，被提议者总是比他们的 当前未婚夫。

当结合永久和临时存储器（在3.2节中解释）之间的区别时，这两个二分法（堆栈/队列，接受首先/接受最后）产生8个算法。 我们用三个字母的缩写来表示它们。 例如，PFS指的是永久内存，Accept-First，堆栈。

在下面介绍的所有8种算法的标准曲线上的样本执行可以在附录A中找到。结果总结在表1中。

我们考虑Accept-First算法。 算法PFS等效于串行专用。 有趣的是，将堆栈切换到队列导致一个等效于朴素波士顿算法的算法，所以我们在这种情况下没有新的不等式算法，只是一个统一的陈述。 我们给出的细节如下。

命题3.3。 具有代理1,2，...，n的顺序的PFS等效于具有相同代理顺序的串行专用。

证明。 BydefinitionPFS是对于代理的项目，因为交互永远不会被断开（通过Accept-First和永久存储器），其中这些代理的参数顺序为1> 2> ...> n。 通过命题3.1，后者等价于代理顺序为1,2，…，n的SD。

命题3.4。 PFQ，其中，项目的共同偏好顺序是1> 2> ...> n，其中项目1,2，...，n等价于1-SideNiveiveBostonal算法。

证明。 将上述算法视为发生在轮次中是有用的。 当所有剩余的代理提出了他们的第i个选择时，轮次结束。 我们通过归纳显示，在第一轮：

•提前准确地提前准备的试剂;

•每个实体，

由于这是（同时）单边波束算法的行为，并且由于分配在任一算法中从不改变，因此命题如下。

当i = 1时，所有代理都提出了他们的第一选择。 因此，每个代理正好做出一个提议，因为接受的任何代理都不会提出另一个提议，任何被拒绝的代理都必须到队列的后面，并等待所有其他代理提交第一个提议。

假设结果对于i之前的所有回合成立，则所有剩余的代理已经提出并且被所有项目拒绝排名为i-1。在回合i中，他们必须建议他们的第i个选择。 这发生正好一次，因为队列纪律。

实施例3.5。 （Accept-Lastpermanentmemoryalgorithmsversusaccept-first）

对于标准简档，使用PFS的最终分配是1：a，2：b，3：c，4：d，

d，2：c，3：a，4：b。 如果针对PFQ的最终分组标志，最终分配：a，2：c，3：d，4：b，并且对于PLQ，1：d，2：c，3： 注意，对于这些偏好，一些代理必须接收其第四选择，并且不超过两个代理可以接收他们的第一选择。 只有PFQ实现后一种条件。

虽然接受最后算法不是事后有效的，但经验结果表明（除其他事项外）他们的输出当用作TTC的初始禀赋时，导致比先前高效的Accept-First算法更好的福利性能。 我们在第4节详细讨论这个问题。

表1：其中n = 4的标准简档上的算法的行为

算法 输出匹配 提案数量

|  |  |  |
| --- | --- | --- |
| PFS | 1:a, 2:b, 3:c, 4:d | 10 |
| PFQ | 1:a, 2:c, 3:d, 4:b | 9 |
| PLS | 1:d, 2:c, 3:a, 4:b | 9 |
| PLQ | 1:d, 2:c, 3:b, 4:a | 10 |
| TFS | 1:d, 2:a, 3:c, 4:b | 18 |
| TFQ | 1:a, 2:b, 3:d, 4:c | 33 |
| TLS | 1:b, 2:a, 3:d, 4:c | 18 |
| TLQ | 1:a, 2:b, 3:d, 4:c | 21 |

3.2 “临时存储”情况

先前的算法假定每个项目在算法的整个执行期间保持其偏好。 通过放宽该要求可以生成其他有趣的算法。 每当项目重置其内存时，代理程序建议对之前拒绝它们的项目有意义。 使用项目的临时内存的任何规则都必须确保算法将停止与匹配。 为了确保算法停止，我们只允许项目在暂定匹配的数量增加时重置其存储器。 由于项目不会从匹配到不匹配，这恰好在匹配新项目时发生。

每当代理人建议一个不匹配的项目，所有项目，包括新项目，失去他们的偏好的记忆。 当代理向没有偏好的匹配项目建议时，项目优选建议代理而不是匹配的代理。 由于在算法的执行期间暂时匹配的数量增加，所以只能有n个偏好的重置，因此建议的数量由n3限制。

使用上述规则，可以构造类似于3.1节中的那些的四个新算法。

我们首先呈现Accept-First算法。 串行专用的临时存储器模拟，即TFS，是有趣的。 该算法的每一轮操作类似于串行专家，其中代理的子集重复地“偷取”链中的项目，直到一些代理选择不匹配的项目，于是新的轮次开始于开始窃取的新代理。代理选择的顺序 我们建议对这个算法使用替代名称“Iterative Dictatorhip”，算法TFQ是单边波形算法的临时存储器模拟，像所有基于队列的算法一样， 比基于堆栈的算法更难解释。

接受最后的算法更难理解（但见下面的方解释，这是我们整个研究计划的灵感）。

实施例3.6。 （临时存储器，Accept-First和Accept-Last）对于标准简档，TFS下的最终分配为1：d，2：a，3：c，4：b，使用TFQ的最终分配为1：a ，2：b，3：d，4：c。 相比之下，使用TLS的最终分配是1：b，2：a，3：d，4：c，而TLQ下的最终分配是1：a，2：b，3：d，4：c。

3.3 进一步意见

我们在统一的框架中提出了8个算法，对应于二分法存储器/无存储器，堆栈/队列，接受第一/接受最后。 表1中显示了它们在标准配置文件上的行为的基本描述。注意，除了TFQ和TLQ之外，所有输入都有不同的输出，这通常是不同的。 一个强有力的声明，即所有随机化版本都是不等式算法，见附录A.9中的例子。 还要注意，在这个输入上，只有PLQ无法给出有效的分配。 在PLQ的输出上运行TTC产生分配1：a，2：c，3：b，4：d。

所有的算法都可以用聚会游戏来解释，主机提供了一个礼物。 当每个人到达聚会（说，虽然一个狭窄的门），他们从藏匿或（在某些情况下）从另一个人的礼物。 永久存储器算法具有单个回合，并且临时存储器算法在每次获得新的存在时开始新的回合。 对于Accept-First算法，在每个回合中，每个存在可以至多采取一次。 对于接受最后算法，在每个回合中，每个（人，当前）对可以最多出现一次。 队列或堆栈规则确定当队友失去队伍的现状时会发生什么：立即选择替换队列，或者到队列的后面。 基于接受优先级队列的算法将是无趣的，就像PFS，但其他似乎我们值得尝试。

TLS算法在这个解释中与聚会游戏扬基交换或白色电子[Con17]密切相关，这是我们的研究计划的灵感。 在真实的游戏中，礼物是由聚会游客贡献的，并且是包装的，所以没有人有自己的偏好的完整信息。 我们不知道基于其他算法的任何其他真实的聚会游戏。

4算法的属性

我们的大多数算法不能满足任何常见的公理性质。 然而，一些具有良好的平均情况行为。 有趣的是，他们的行为相当不同。

4.1事后效率

命题4.1。 AllAccept - Firstalgorithms areex-postefficient。

证明。我们通过归纳显示轮次数，到目前为止构建的部分分配是事后有效的（在每种情况下，每当先前不匹配的项目被选择时，轮次结束 - 注意，这是“轮”的不同用法第3.3节，其中永久存储器算法具有单轮）第一轮总是以第一代理作为其首选的终止，并且这显然是事后有效的结果。假设结果对于i之前的所有轮次成立并且考虑轮回i。输入代理选择项目并且在整个轮次中保留它（通过接受第一策略）。在该轮期间采取不匹配项目或窃取项目的每个代理（“活动代理”）选择可用的最佳项目。这样的代理j将只希望与从最后一次存储器复位之后选择的代理k（在永久存储器情况下，自算法的开始;在临时存储器情况下，因为该轮的开始）进行交易。然而在这种情况下，k不希望与j交易。因此，在活性剂组内没有相互有益的交易周期。通过归纳假说，在非活性剂组内也没有这样的循环。活动组中的每个代理将其当前项优先于由非活动组保存的所有内容，因为这些项可用于窃取。结果如下，因为TTC将终止，没有交易。

实施例4.2。 所有Accept-Last算法失败事后效率。 为了看到这一点，考虑其中代理1,2,3具有各自优选顺序a> b> c，a> b> c，b> a> c的简档。 直接计算显示对于每个算法X，存在一些初始代理顺序，使得通过在X的输出上运行TTC形成的算法XG给出不同的结果。 因此，X不能是事后有效的。

4.2有序效率

如果没有另一个随机分配，则随机分配S在没有另一个随机分配的情况下是有效的，S'每个代理i SD优选Si'到Si，至少1个代理严格地SD优选它们在S'下的分配。 有序效率意味着事后效率，反之亦然[BM01]。

实施例4.3。 AllAccept-Firstalgorithmsfailordinalefficiency.Acounterexample（详细说明）：假设代理1,2优选a> b> c> d，而代理3,4具有优选a> b> d> c。 这个相同的反例也适用于事后有效算法PLSG和PLQG。 具有n = 4的不同反例对于TLSG和TLQG起作用，并且对于这两种算法，顺序效率似乎较少被违反。

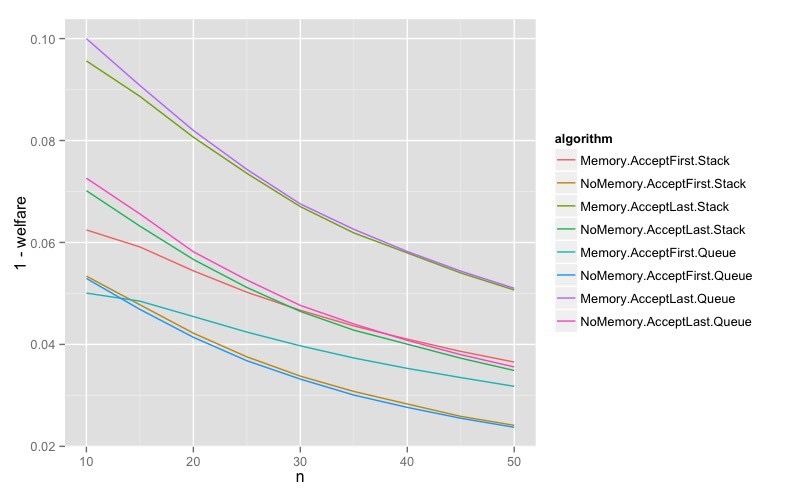
4.3战略性

如果没有代理具有错误报告其偏好的激励，而不管其他代理做什么，则分配算法是策略。换句话说，真实报告是每个代理的严格主导策略。这种情况当然相当强烈，但已知是由串行专制，因此由PFS满足。这里讨论的所有其他算法不能满足它，如下所示。对于随机分配算法，激励被理解为以通常方式的预期效用来表示。

算法概率序列满足弱策略性[BM01]，其表示没有代理有偏差的激励，其中激励以一阶随机优势表示。换句话说，对于每个简档和每个代理，对于与真实性的偏离是不利的代理的效用函数存在一些一致的选择。注意，这比策略性略弱，其表示对于每个简档和代理，以及对于该代理的效用函数的每个一致选择，与真实性的偏离是不利的。

显然，如果算法X对于代理的每个初始次序是策略性的，则RX是RX。

图1：我们的8个基本算法的功利福利损失



我们通过示例显示，我们的算法的任何随机化版本都不是弱策略。 考虑代理1,2,3,4都优选具有a> b> c> d的情况。 如果一些代理提交而另一些代理提交，而其他代理保持真实，则算法TLS，TLG，PLS，PLQ，TLQ，TLQG获得该代理的优选结果。

类似的实施例表明，PLQG，PLSG，TFS和TFQ以及PFQ都失败了弱的策略性。

4.4功利福利

我们在这里给出一个基本分析，并引用读者对我们更广泛的分析[LW16]。我们在这里只分析算法的随机化版本在真实行为下的平均情况性能。我们通过要求它们都具有相同的（Borda）效用函数来估算代理的效用值，其中第i个选择对应于效用n-1 + i。我们考虑功利性社会福利，这是所有代理人对其分配效用的总和。功利性社会福利的最优值是可有效计算的，例如使用匈牙利算法[]。这允许我们量化每个算法平均损失的最大可能社会福利的分数。我们使用K. Stern [Ste]的Java实现。

结果如图1所示。他们表明，RSD实质上优于PFQ，TFQ和TFS，但优于Accept-Last算法。在图2中，我们展示了当TTC在我们的Accept-Last算法的输出上运行时，福利是如何显着地改善的。在图3中，我们展示了我们最好的算法，即TLQG和TLQS，与标准算法RSD，PS和天真波士顿相比较。事实上，所有具有TTC的Accept-Last算法优于那些标准算法。

4.5平均福利

我们也考虑平等主义的福利，即最坏的代理人的福利。 由于最优的精确计算很难[]，所以我们用n代替精确的最优。

结果与实用案例类似，显示了RSD和Naive Boston的非竞争性，GTTC的积极效应，以及TLSG和TLQG的整体优势。 在图4中，TLSG和TLQG的行是不可区分的。 我们不与PS比较，因为它是固有随机的，因此比较对于其他算法是不公平的 - 对最小福利的期望值小于期望值的最小值。 注意，新算法的福利增加为n，而旧算法的福利减少。

图2：TTC的Accept-Last的实用性福利改善

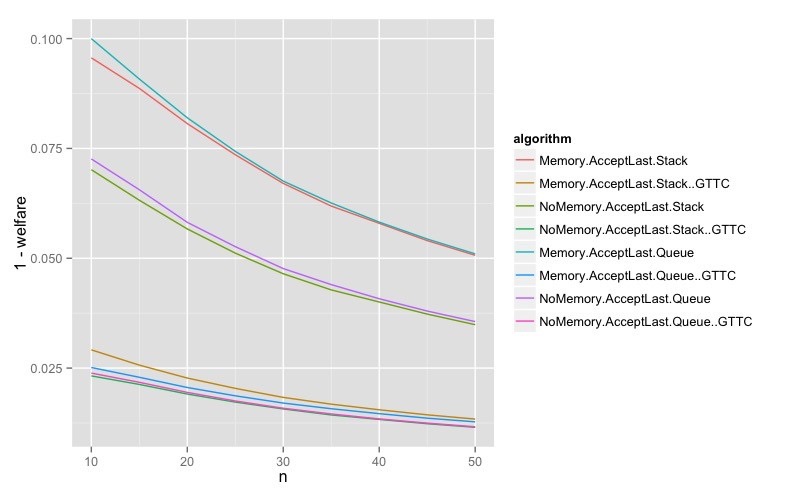


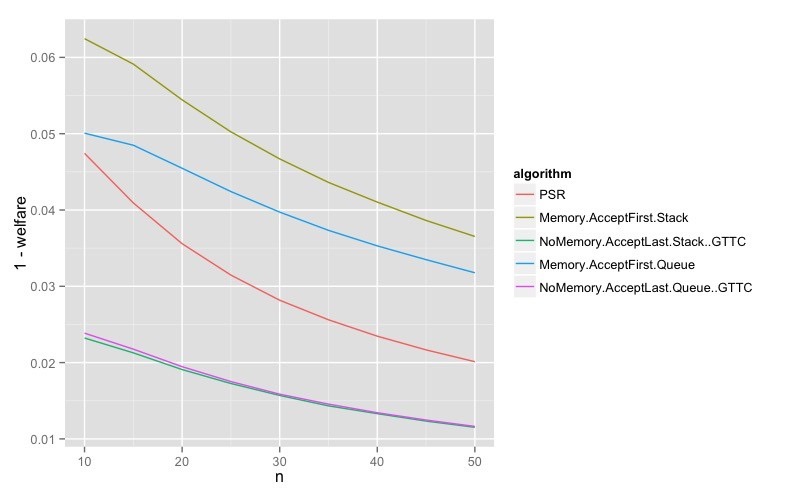
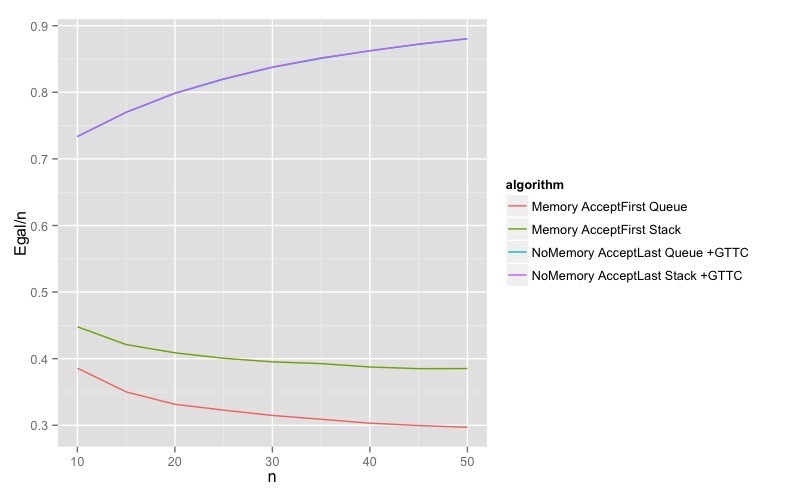
图3：实用性福利损失比较

图4：归一化的平均福利比较

4.5.1平均福利界限

说一个算法满足k的条件性平均福利边界，如果每个代理接收到它的前k个选择之一，只要在某种分配下是可能的。 如果k恰好是最小可能值，则说明该算法在该输入上产生最优的平均borda福利。 所有合理的算法（包括本文中的所有算法）满足1的边界，因为如果所有代理具有不同的顶部选择，则每个代理都接受它的最高选择。 我们调查了情况k = 2，发现我们的算法没有一般满足它（细节省略）。 然而，当n = 3时，TLS满足k = 2的界限，其它基于队列的对等物TLQ也是如此，而没有其他算法。 因此，对于n = 3，这些算法是均等最优的。

4.6定单偏差

在所有代理对项目具有相同偏好的情况下，一些算法（例如串行特殊符号）显然偏向第一代理，而其他算法（接受最后算法）明显偏向最后代理。 我们将算法的阶偏差定义为所有预期（在偏好上的均匀分布）下的所有代理对上的最大值，Borda福利增加，n由正态化。

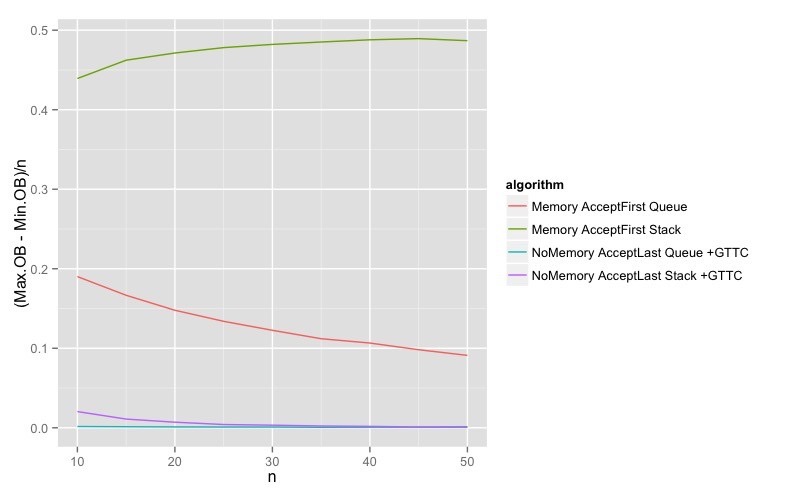
结果表明，基于队列的算法的顺序偏差显着小于基于堆栈的算法。 我们最好的福利算法，即TLSG和TLQG，具有几乎零阶偏差（PS具有零阶偏差的定义），但随机序列独裁和天真波士顿算法具有实质的阶次偏差，前者明显比所有 其他算法。 向接受最后算法添加TTC大大减少了阶次偏差（未示出）。

5 结论

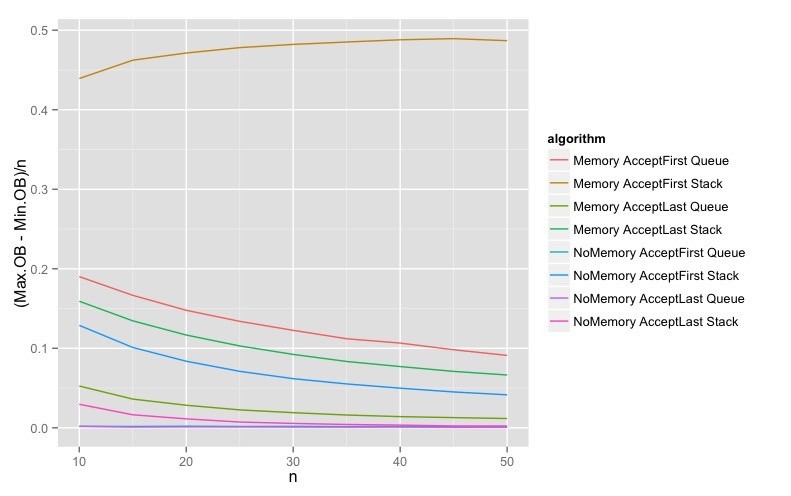
5.1结果总结

我们已经为我们的知识引入了10个用于单边匹配的新算法，它们都不等同于文献中的任何算法。 他们使用Gale-Shapley框架的推导给出了统一的描述。 每个算法在次序n2（永久存储器）或n3（临时存储器）的最差情况下运行。 虽然他们缺乏强大的公理基础，几个算法在标准上表现良好，如平均福利，功利福利和秩序偏差，与标准

图5：我们的8个基本算法的归一化阶偏差

图6：归一化阶偏差比较

串行专用或波士顿算法。算法TLSG和TLQG比标准算法Serial Dictatorhip，Naive Boston和Probabilistic Serial在这些措施上的表现总体上更好。

在本文中，我们引入了一个新的（对我们的知识）与对称性相关的性能标准，即顺序偏差。基于队列的算法总体上比基于堆栈的算法执行的更好。算法TLQ（和TLQG）具有显着小的阶偏差。我们的直觉是，队列模拟由代理选择的顺序的随机化。

我们使用的考虑不是事后有效的算法，然后在其输出上运行TTC的方法似乎是我们新的。事后效率大致来说是局部最优标准。通过避免过早锁​​定效率，我们的新的Accept-Last + TTC算法似乎能够实现高效的结果与更高的全球福利。

例如，与使用TTC所获得的增益相比，使用临时与永久存储器的好处相对较小。然而，它们是真实的，并且对于解决方案质量实质上比运行时更重要的情况，我们建议使用它。在解决方案质量方面最好的整体算法可以是TLSG和TLQG。后者具有较低阶偏差和前者较高的福利，尽管算法之间的差异较小。

我们的一些新算法可能对特殊情况（除了聚会游戏）有用。例如，TFS似乎在福利中平等地对待所有代理，除了最后一个，他有明确的优势 - 这可能是有用的，例如，当一个代理是一个小孩子。具有（几乎）零阶偏差是在一些应用中非常重要的强公平条件。我们的单边朴素波士顿算法PFQ最大化接受他们的第一选择的代理的数量。

扩展基本算法的库存比仅仅允许我们另外10个算法可以选择更多的好处。 Mennle＆Seuken已经使用了杂交的概念[MS13]。这简单地通过取随机矩阵输出的凸组合（1-p）MA + pMB从随机分配算法A和B和固定的p∈[0,1]形成新的算法（1-p）A + A和B.这使我们能够以受控的方式在期望的属性（例如战略性和效率）之间进行权衡。 Mennle＆Seuken只考虑RSD，PS，Naive Boston和最大化功利福利的算法作为它们的基本算法。我们相信，我们的新算法将证明作为具有良好整体行为的混合算法的构建块是有用的。

5.2未来工作

自适应波士顿学校选择算法通过允许代理跳过显然将被拒绝，因为学校已经达到容量的建议，改善了天真波士顿。它满足了薄弱的战略性和战略性之间的属性中介，被称为部分战略性的根据Mennle＆Seuken [MS14]。虽然我们发现了一个顺序解释的天真波士顿，避免讨论同时提出，我们还没有这样做适应波士顿。

我们的每个算法都延伸到学校选择的情况。 “存储器”组件和“数据结构”组件直接转换，不需要更改。接受政策如果更复杂。在学校选择，学校暂时接受申请，直到达到能力，然后每个新的临时招生要求取消现有的招生。在上述能力1案例中，我们的接受最后和接受第一政策也可以被称为“拒绝 - 当前”或“拒绝新”。在一般的学校选择情况下，我们需要为学校创造虚拟的偏好决定哪个当前学生拒绝，有些明显的方法包括FIFO或LIFO。

物品的数量超过代理的数量，并且代理接收一次选择一个的物品的捆的情况是复杂的。代理在每个回合中应该选择的顺序（这里，当所有代理已经将它们以前的项目总数增加1时，回合结束）必须被指定。例如，当我们具有超过4个项目的偏好a> b> c> d的2个代理，并且拾取序列1221，串行专家向代理1授予a，d，向代理2授予b，c。然而，在PFQ代理2下尝试得到项目a和失败，去到队列的后面，因此失去了这一回合。在下一轮中，它被分配b，则代理1尝试b并且失败。因此，挑选顺序必须扩展。接受 - 最后的算法在这种情况下工作得更好。

我们用m̸= n对这些有趣的案例进行进一步探索，以进行未来的工作。

参考文献

[AS98] Atila Abdulkadirog ̆lu and Tayfun Sönmez. Random serial dictatorship and the core from random endowments in house allocation problems. *Econometrica*, 66(3):689–701, 1998.

[AS03] Atila Abdulkadirog ̆lu and Tayfun Sönmez. School choice: A mechanism design approach. *The American Economic Review*, 93(3):729–747, 2003.

[BM01] Anna Bogomolnaia and Hervé Moulin. A new solution to the random assignment problem. *Journal of Economic theory*, 100(2):295–328, 2001.

[Con17] Wikipedia Contributors. White elephant gift exchange — Wikipedia, the free encyclopedia, 2017. [Online; accessed 2017-03-13].

[GS62] David Gale and Lloyd S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

[Kes09] Onur Kesten. Why do popular mechanisms lack efficiency in random environments? *Journal of Economic Theory*, 144(5):2209–2226, 2009.

[LW16] [MS13] [MS14] [SS74] [Ste]

Jacky Lo and Mark C. Wilson. Average-case analysis of random assignment algorithms. *Preprint*, 2016.

T. Mennle and S. Seuken. Hybrid Mechanisms: Trading off Efficiency and Strategyproofness in One-Sided Matching. *ArXiv e-prints*, March 2013.

Timo Mennle and Sven Seuken. The Naive versus the Adaptive Boston Mechanism. *arXiv preprint arXiv:1406.3327*, 2014.

Lloyd Shapley and Herbert Scarf. On cores and indivisibility. *Journal of mathematical Economics*, 1(1):23–37, 1974.

Kevin Stern. Hungarianalgorithm.java. https://github.com/KevinStern/ software-and-algorithms/blob/master/src/main/java/blogspot/software\_and\_ algorithms/stern\_library/optimization/HungarianAlgorithm.java.

附录

A算法的示例执行

对于其中代理1,2和3具有偏好a> b> c> d并且代理4具有偏好b> a> c> d并且代理的初始排序按照数字顺序递增的简档，提议顺序和 使用各种算法的拒绝将如下。

A.1永久内存，早期建议首选，堆栈

|  |  |  |  |
| --- | --- | --- | --- |
| Proposal | Outcome | Order of Remaining Agents | Current Partial Matching |
| *a*1 *=>* a | tentatively matched | *a*2, *a*3, *a*4 | *a*1:a |
| *a*2 *=>* a | *a*2 rejected | *a*2, *a*3, *a*4 | *a*1:a |
| *a*2 *=>* b | tentatively matched | *a*3, *a*4 | *a*1:a, *a*2:b |
| *a*3 *=>* a | *a*3 rejected | *a*3, *a*4 | *a*1:a, *a*2:b |
| *a*3 *=>* b | *a*3 rejected | *a*3, *a*4 | *a*1:a, *a*2:b |
| *a*3 *=>* c | tentatively matched | *a*4 | *a*1:a, *a*2:b, *a*3:c |
| *a*4 *=>* b | *a*4 rejected | *a*4 | *a*1:a, *a*2:b, *a*3:c |
| *a*4 *=>* a | *a*4 rejected | *a*4 | *a*1:a, *a*2:b, *a*3:c |
| *a*4 *=>* c | *a*4 rejected | *a*4 | *a*1:a, *a*2:b, *a*3:c |
| *a*4 *=>* d | tentatively matched |  | *a*1:a, *a*2:b, *a*3:c, *a*4:d |

A2永久内存，早期建议首选，队列

|  |  |  |  |
| --- | --- | --- | --- |
| Proposal | Outcome | Order of Remaining Agents | Current Partial Matching |
| *a*1 *=>* a | tentatively matched | *a*2, *a*3, *a*4 | *a*1:a |
| *a*2 *=>* a | *a*2 rejected | *a*3, *a*4, *a*2 | *a*1:a |
| *a*3 *=>* a | *a*3 rejected | *a*4, *a*2, *a*3 | *a*1:a |
| *a*4 *=>* b | tentatively matched | *a*2, *a*3 | *a*1:a, *a*4:b |
| *a*2 *=>* b | *a*2 rejected | *a*3, *a*2 | *a*1:a, *a*4:b |
| *a*3 *=>* b | *a*3 rejected | *a*2, *a*3 | *a*1:a, *a*4:b |
| *a*2 *=>* c | tentatively matched | *a*3 | *a*1:a, *a*2:c, *a*4:b |
| *a*3 *=>* c | *a*3 rejected | *a*3 | *a*1:a, *a*2:c, *a*4:b |
| *a*3 *=>* d | tentatively matched |  | *a*1:a, *a*2:c, *a*3:d, *a*4:b |

A3 永久内存，最后建议优先，堆栈

|  |  |  |  |
| --- | --- | --- | --- |
| Proposal | Outcome | Order of Remaining Agents | Current Partial Matching |
| *a*1 *=>* a | tentatively matched | *a*2, *a*3, *a*4 | *a*1:a |
| *a*2 *=>* a | *a*1 rejected | *a*1, *a*3, *a*4 | *a*2:a |
| *a*1 *=>* b | tentatively matched | *a*3, *a*4 | *a*1:b, *a*2:a |
| *a*3 *=>* a | *a*2 rejected | *a*2, *a*4 | *a*1:b, *a*3:a |
| *a*2 *=>* b | *a*1 rejected | *a*1, *a*4 | *a*2:b, *a*3:a |
| *a*1 *=>* c | tentatively matched | *a*4 | *a*1:c, *a*2:b, *a*3:a |
| *a*4 *=>* b | *a*2 rejected | *a*2 | *a*1:c, *a*3:a, *a*4:b |
| *a*2 *=>* c | *a*1 rejected | *a*1 | *a*2:c, *a*3:a, *a*4:b |
| *a*1 *=>* d | tentatively matched |  | *a*1:d, *a*2:c, *a*3:a, *a*4:b |

A4 永久内存，首选最后建议，队列

|  |  |  |  |
| --- | --- | --- | --- |
| Proposal | Outcome | Order of Remaining Agents | Current Partial Matching |
| *a*1 *=>* a | tentatively matched | *a*2, *a*3, *a*4 | *a*1:a |
| *a*2 *=>* a | *a*1 rejected | *a*3, *a*4, *a*1 | *a*2:a |
| *a*3 *=>* a | *a*2 rejected | *a*4, *a*1, *a*2 | *a*3:a |
| *a*4 *=>* b | tentatively matched | *a*1, *a*2 | *a*3:a, *a*4:b |
| *a*1 *=>* b | *a*4 rejected | *a*2, *a*4 | *a*1:b, *a*3:a |
| *a*2 *=>* b | *a*1 rejected | *a*4, *a*1 | *a*2:b, *a*3:a |
| *a*4 *=>* a | *a*3 rejected | *a*1, *a*3 | *a*2:b, *a*4:a |
| *a*1 *=>* c | tentatively matched | *a*3 | *a*1:c, *a*2:b, *a*4:a |
| *a*3 *=>* c | *a*1 rejected | *a*1 | *a*2:b, *a*3:c, *a*4:a |
| *a*1 *=>* d | tentatively matched |  | *a*1:d, *a*2:c, *a*3:b, *a*4:a |