Stable Allocation Problem
A Modern Implementation

Angelo Gkikas

May 2016

1 Introduction

The purpose of this report is to describe in detail a modern implementation of the Stable Allocation Problem. The programming environment was based on Java and the project was build on Eclipse. Due to computational times and potential comparisons below we provide the system details of the personal computer that was used for the results. The execution of the program was handled by a Turbo-X desktop computer built with a Intel(R) Core(TM) i7 CPU @ 2.000 Ghz, 2.60 Ghz. The system type was 64-bit Operating System with a 64-bit processor.

2 Stable Matching Problem

The stable matching problem (also stable marriage problem or SMP) is the problem of finding a stable matching between two equally sized sets of elements given an ordering of preferences for each element. Given n men and n women, each of whom submits an ordered preference list over all members of the opposite sex, we seek a matching between the men and women that is stable. A matching is a mapping from the elements of one set to the elements of the other set. A matching is stable whenever it is not the case that both the following conditions hold. The first condition states that there is an element A of the first matched set which prefers some given element B of the second matched set over the element to which A is already matched, whereas the second condition states that B also prefers A over the element to which B is already matched. The classical stable matching (marriage) problem has been extensively studied since its introduction by Gale and Shapley in 1962. Gale and Shapley showed how to solve the problem optimally in O(n 2) time (more precisely, in O(m) time on a bipartite instance with only m edges) using a simple and natural “propose and reject” algorithm. The Gale–Shapley algorithm involves a number of ”iterations. In the first round, first a) each unengaged man proposes to the woman he prefers most, and then b) each woman replies ”maybe” to her suitor she most prefers and ”no” to all other suitors. She is then temporarily ”engaged” to the suitor she most prefers the most, and that suitor is likewise temporarily engaged to her. In each subsequent iteration, first a) each unengaged man proposes to the most-preferred woman to whom he has not yet proposed (regardless of whether the woman is already engaged), and then b) each woman replies ”maybe” if she is currently not engaged or if she prefers this guy over her current partner (in this case, she rejects her current partner who becomes unengaged). By the time the algorithm terminates, we ensure that a) there cannot be a man and a woman both unengaged, as he must have proposed to her at some point and b) it is not possible for two agents of different sets to prefer each other over their current partners. Below we provide a pseudo code for the Gale-Shapley algorithm.

Algorithm 1 Gale Shapley Algorithm

```
Initialize all m ∈ M and w ∈ W to free
while unassigned man exists do
  w = first woman on m's list to whom m has not yet proposed
  if w is free then
    (m, w) become engaged
  else
    some pair (m', w) already exists
    if w prefers m to m' then
      m' becomes free
      (m, w) become engaged
    else
      (m, w) remain engaged
    end if
  end if
end while
```

3 Stable Allocation Problem

The stable allocation problem is a recent variant of the stable marriage problem introduced by M. Baiou and N. Balinski in 2002. Although the core of the problem remains the same (how to best bring different parties together for mutual benefit), there are some significant changes that characterize the stable allocation problem. The first major change is that the stable allocation problem is a two-sided market with distinct sets of agents where each agent has strict preferences over the opposite set; but instead of assigning or matching, the question is how to allocate amount of information. Consequently, the amount of assignment between two elements i and j is no longer zero or one, but a non-negative real number. Hence, we can reasonably conclude that one element may be assigned to more than one element of the other side, which leads to the fact that the size of each set does not have to be equal. “Stability” simply asks that no pair of opposite elements can increase their time together either due to unused capacity or by giving up time with less desirable partners.

4 Preliminaries

In order to introduce the mathematical model of the stable allocation problem, let us assume we are matching N jobs indexed by [J] = 1, . . . , n to M machines indexed by [M] = 1, . . . , m. Furthermore, as mentioned before each job i is associated with a processing time p(i) and each machine j has an upper capacity c(j) of time. For each edge (i,j), connecting two nodes of different sets, we define an upper capacity u(i,j) ¡= min(p(i),c(j)), representing the maximum amount of time that can be assigned from job i to machine j. Each job/machine is also associated with a preference list, representing a unique preference order over the other set. Bear in mind, that no ties are allowed in the preference list. A bipartite graph of an instance is now created, as the jobs comprise the left side and the machines the right one respectively. Finally in order to ensure that a stable and feasible solution always exists, we add two more nodes, one in each set respectively. Both extra nodes are associated with a very large processing time and capacity. The behaviour of these extra nodes may be considered "dummy" as they do not submit a ranked preference list over the other set, neither are included in any stable solution. The dummy job is the last entry in each job's preference list and the dummy machine is the last entry in each machine's preference list. For each job i, we let q_i be the machine j most preferred by i such that x(i, j) ¡ u(i, j) and j is accepting for i. If i wishes to increase its allocation, q_i is the first machine it should logically ask. Similarly, Job r_j is the job that j would logically choose to reject first if it were offered an allocation from a more highly-preferred job.

5 Solving Methodology

As far as the solving methodology is concerned, there are three steps that need to be executed in order the program to terminate. We describe the nature and utility of each step in the next subsections.

5.1 Instance Production

A personalized computational method is responsible for the creation of an instance. This method creates a stable allocation instance I(n,m) with n jobs and m machines. This method accepts as parameters two integer number which specify the number of jobs and machines that need to be created. Afterwards, we use a number generator to assign processing times and capacities and finally we use another randomized procedure to fill the preference list of each job and machine. Each instance created and solved by our algorithm satisfies 3 constraints.

1. The total amount of process time (including all jobs) is equal to the total capacity of the machines.
$$\sum_{i=1}^{J} P_i = \sum_{j=1}^{M} C_j$$
2. Each preference list is strict, no ties are allowed.
3. Each job is able to allocate its processing time among all the other machines.
$$P_i >= \sum_{j=1}^{M} U_{ij}$$

Where P_i is the processing time of job i, C_j is the capacity of machine j, U_{ij} is the minimum amount of time of job i that can be processed in machine j U_{ij}=min(P_i,C_j).

5.2 Gale-Shapley Algorithm

After the instance is created, the Gale-Shapley algorithm runs to produce the job-optimal and machine-optimal assignments. The GS algorithm for the stable allocation problem is a natural generalization of the known GS "propose and reject" algorithm for the stable matching problem. Although the GS algorithm typically starts with an empty assignment, we start with an assignment A where every machine j is fully assigned to the dummy job and the remaining jobs are unassigned. In that way we ensure that every machine except the dummy remains fully assigned . Afterwards, in each iteration of the algorithm, we select a job i that is not yet fully assigned. We compute the amount of i's processing time that is currently unassigned T = p(i) - x(i, [J]) where x(i,[J]) is the total amount of time currently distributed by job i. Job i "proposes" T_0 = min(T, u(i, j) x(i, j)) units of processing time to machine j = q_i, which accepts. However, if j is any machine except the dummy, then it is now overfilled by T_0 units beyond its capacity, so it proceeds to reject T_0 units, starting with job r_j . During the process, x(r_j , j) may decrease to zero, in which case r_j becomes a new job higher on j's preference list and rejection continues until j is once again assigned exactly c(j) units. The algorithm terminates when all jobs are fully assigned, and successful termination is ensured by the fact that each job can send all of its processing time to the dummy machine as a last resort. Below we provide the pseudo code of our algorithm.

Algorithm 2 Instance Generator Algorithm

```
Assign all machines to the dummy job
while all jobs are not fully assigned do
    i = next not fully-assigned job
    j = $q_i$ machine on i's list to whom i has not yet proposed
    T = p(i) - x(i, [J]) compute remaining time
    j "proposes" $T_0$ = min(T, u(i, j)-x(i, j)) time to machine j
    if j accepts then
        machine j rejects $T_0$ amount of time starting from job $r_j$
        x(i,j)=$T_0$
    else
        j = next $q_i$ machine on i's list to whom i has not yet proposed
    end if
end while
```

5.3 Rotations Algorithm

The structure of the set of all stable allocations turns out to be related to the structure of the set of rotations. In the bipartite versions of matching problems, rotations can be used to represent the set of all stable allocations in a compact way and also to efficiently find a stable matching that is optimal with respect to certain objective function. For example, consider a stable allocation A, a rotation ρ consists of a sequence of job-machine pairs that participate in A together with a value v_ρ that specifies the amount of time distributes within this rotation. This sequence of pairs has the property that, starting from A and moving any amount within $(0; v_\rho]$ from the machine with which each job appears in ρ to the machine with which the previous job in the sequence appears in ρ produces a different stable allocation. Consequently, this is the main idea of our algorithm, designed to identify all stable allocations. Below we provide the pseudo code of our algorithm.

Algorithm 3 Rotations Algorithm

```
$M_J$=job optimal matching
$Q$=a new empty queue
while A job (j) that can get less happy exists do
  if $Q$ is empty then
    add Job j to $Q$
  else
    j = $Q$.remove
  end if
  m = first machine on j's list to whom j has not yet proposed
  l = least preferred machine currently assigned with j
  T= min(u(j,m),x(j,l))
  Job j proposes to Machine m T units of time
  if m prefers j from current partner j' then
    m rejects T units of time from least preferred job j'
    x(j,m)=T
    if $Q$.contains(j) then
      $Q$.empty
      eliminate rotation
    else
      add job j' to queue
    end if
  end if
end while
```

5.4 Poset Graph Algorithm

In order to produce a linear programme describing a stable allocation instance we need to create the poset graph of the rotations. In more detail the poset graph is a directed acyclic graph. Each node of the graph represents a rotation that has been exposed and eliminated according to the previous subsection. The poset graph actually saves the correct order of the rotations and more specifically, which rotations should be first exposed in order other rotations to be identified. The poset graph is created by an algorithm that firstly maintains a "book keeping" on the preference lists and afterwards extracts this information in order to create the necessary edges. The "book keeping" is comprised by three rules.

We consider an arc between two rotations, r_i and r_j, which means that r_i

must be eliminated before r_j if

$$r_i = r_i^+(j, m)$$

$$r_j = r_i^+(j, next_j(m))$$

or,

$$r_i = r_i(next_m(j), m)$$

$$r_j = r_i(j, m)$$

or,

$$r_i = r_i^-(j, m)$$

$$r_j = r_i^+(j, next_j(m))$$

Where $next_j(m)$ is the first machine less prefer ed than m by j such that (j,$next_j(m)$) is stable.