On Stable Marriages and Greedy Matchings

Fredrik Manne*, Md. Naim*, Håkon Lerring*, and Mahantesh Halappanavar†

Research on stable marriage problems has a long and mathematically rigorous history, while that of exploiting greedy matchings in combinatorial scientific computing is a younger and less developed research field. We consider the relationships between these two areas. In particular we show that several problems related to computing greedy matchings can be formulated as stable marriage problems and as a consequence several recently proposed algorithms for computing greedy matchings are in fact special cases of well known algorithms for the stable marriage problem.

However, in terms of implementations and practical scalable solutions on modern hardware, work on computing greedy matchings has made considerable progress. We show that due to this strong relationship many of these results are also applicable for solving stable marriage problems. This is further demonstrated by designing and testing efficient multicore as well as GPU algorithms for the stable marriage problem.

1 Introduction

In 1962 Gale and Shapley formally defined the stable marriage problem and gave their classical algorithm for its solution [5]. Since then this field has grown tremendously with numerous applications both in mathematics and in economics. For a recent overview see the book by Manlowe [16]. Graph matching is a related area where the object is also to find pairs of entities satisfying various optimality criteria. These problems find a large number of applications. For an overview motivated from combinatorial scientific computing see [21].

While research on stable marriage problems has mainly focused on theory and mathematical rigor, work on graph matching in scientific applications has a larger practical component concerned with implementing and testing code on various computer architectures with the intent of developing fast scalable algorithms.

In this paper we investigate the connection between one type of matching problems, namely those of computing greedy weighted matchings, and algorithms for solving stable marriage problems. Although there exist exact algorithms for solving various weighted matching problems these tend to have running times that typically involve the product of the number of vertices and the number of edges. As large graph instances can contain tens of millions of vertices and billions of edges it is clear that such algorithms can easily become infeasible. For this reason there has been a strong interest in developing fast approximation algorithms and also in parallelizing these, see [18] and the references therein. Although such algorithms typically only guarantee an approximation factor of 0.5 compared to the optimal one, practical experiments have shown that they are very often only within a few percent from optimal. One such algorithm is the classical greedy algorithm applied to an edge weighted graph. Here edges are considered by decreasing weight and an edge is included in the matching if it is not adjacent to an already included edge.

The main contributions of this paper are as follows. Initially we consider implementation issues when designing efficient algorithms for the stable marriage problem. Next, we show that several recently published algorithms for computing greedy matchings are in fact special cases of classical algorithms for stable marriage problems. This also includes a generalization of the matching problem known as b-matching where a vertex can be matched with several other vertices in the final solution. Due to the strong similarities between the stable marriage problem and greedy matching, we show that one can apply recent results on designing scalable greedy matching algorithms to the computation of stable marriage solutions. This is verified by presenting efficient parallel implementations of various types of Gale-Shapley type algorithms for both multithreaded computers as well as for GPUs.

The remainder of the paper is organized as follows. In Section 2 we review the Gale-Shapley algorithm and consider implementation issues related to this. Next, in Section 3 we show that the computation of a greedy matching can be reformulated as a stable marriage problem. In Section 4 we give parallel implementations of the Gale-Shapley and McVitie-Wilson algorithms and show their scalability, before concluding in Section 5.

*Department of Informatics, University of Bergen, N-5020 Bergen, Norway. Email: {fredrikm,naim}@ii.uib.no, hakon@lerring.no

†Pacific Northwest National Laboratory, 902 Battelle Boulevard, P.O.Box 999, MSIN J4-30, Richland, WA 99352, USA. Email: mahantesh.halappanavar@pnnl.gov

2 The Stable Marriage Problem

In the following we review the stable marriage (SM) problem and how it can be solved using the Gale-Shapley algorithm and consider some implementation issues. Finally, we review some generalizations of the SM problem.

The SM problem is defined as follows. Let L and R be two equal sized sets $L = \{l_1, l_2, \ldots, l_n\}$ and $R = \{r_1, r_2, \ldots, r_n\}$. The entries in L are typically referred to as "men", while the entries in R are referred to as "women". Every man and woman has a total ranking of all the members of the opposite sex. These give the "desirability" for each participant to match with a member of the other set. The object is to find a complete matching M (i.e. a paring) between the entries in L and R such that no two $l_i \in L$ and $r_j \in R$ both would obtain a higher ranked partner if they were to abandon their current partner in M and rematch with each other. Any solution satisfying this is *stable*.

Gale and Shapley [5] defined the stable marriage problem and also proposed the first algorithm for solving it. The algorithm operates in rounds as follows. In the first round each man in L proposes to his most preferred woman in R. Each woman will then reject all proposals except the one she has ranked highest. In subsequent rounds each man that was rejected in the previous round will again propose to the woman which he has ranked highest, but now disregarding any woman that he has already proposed to in previous rounds. Gale and Shapley showed that this process will terminate with each man in L being matched to a woman in R and that this solution is stable. Although an SM instance can have many stable solutions, the Gale-Shapley algorithm will always produce the same one.

An important variant of this problem is when each participant has only ranked a subset of the opposing participants. This is known as the stable marriage problem with incomplete lists (SMI). Any solution M to an SMI instance must then in addition to being stable, also consists of mutually ranked pairs (l_i, r_j). The SMI problem is solved by the Gale-Shapley algorithm, but the solution might not be complete leaving some participants unmarried [5]. There exists a number of variants of the SM problem, for two comprehensive surveys see the books [9, 16]. In the following we will only consider the classical SM and SMI problems.

The original Gale-Shapley algorithm is described as operating in rounds, where only the men who were rejected in round t will propose in round $t+1$. It is not stated in which order the proposals in a round should be made or what kind of data structures to use. If one traverses the men in L in their original order in each round and lets each rejected man propose once it is discovered, then the men always propose in the same relative order in each round. The running time of such a scheme is $\Theta(n^2)$ even for an instance of SMI. If one is willing to forgo the requirement that the proposals in each round must be made in the same relative order then it is not hard to design an implementation of the Gale-Shapley algorithm with running time proportional to the number of actual proposals made. To do this one maintains a queue Q of men waiting to make their proposals. Initially $Q = L$ and in each step of the algorithm the man at the front of the queue gets to propose to his current best candidate $r_j \in R$, and any rejected l_i is inserted at the end of the queue. This will ensure that all men rejected in round t gets to propose before any man rejected in round $t+1$, but the relative order among the men might not always be the same. The algorithm terminates when the queue is empty.

One simple enhancement of the Gale-Shapley algorithm is that no $l_i \in L$ should propose to an $r_j \in R$ who already has a proposal from someone whom r_j ranks higher than l_i, as such a proposal will be rejected. Thus each l_i should propose to his most preferred r_j where l_i has not already been rejected and where r_j ranks l_i higher than her current best proposal (if any). This means that it is sufficient to only maintain the current best proposal for each r_j. When the algorithm terminates these proposals will make up the solution. We give our complete implementation of the Gale-Shapley algorithm in Algorithm 1.

In Algorithm 1 each r_j has a variable $suitor(r_j)$ initialized to $NULL$ that holds her current best proposal. Similarly, $ranking(r_j, l_i)$ returns r_j's ranking of l_i (as a number in the range 1 through n). We define $ranking(r_j, NULL) = n + 1$ to ensure that any proposal is better than no proposal. The function $nextCandidate(l_i)$ will initially return l_i's highest ranked woman and then for successive calls return the next highest ranked one following the last one retrieved.

For an SM instance it is straight forward to precompute the values of $ranking()$ in $O(n^2)$ time. However, for an SMI instance maintaining a complete $ranking()$ table would require $O(n^2)$ space and also proportional time to initialize it. In this case it is more efficient to store the value of $ranking(r_i, l_j)$ together with r_i in l_j's ranking list so that it can be fetched in $O(1)$ time when needed. These values can be precomputed in time proportional to the sum of the lengths of the ranking lists. To do this one first traverses the women's lists building up lists for each man l_j with the women that have ranked him along with in what position. Then using an array $position()$ of length n initially set to 0, the list of each man l_j is processed as follows. For each woman r_i that has ranked l_j we store the value l_j along with in

what position r_i has ranked l_j in $position(r_i)$. We next traverse l_j's priority list and for each r_i in the list we look up $position(r_i)$ and see if it contains l_j. If so, we fetch r_i's ranking of l_j and store it together with l_j's ranking of r_i. At the same time any r_i that has not ranked l_j but which l_j has ranked can be purged from the priority list of l_j.

Algorithm 1 The Gale-Shapley algorithm using a queue

```
1: Q = L
2: while Q ≠ ∅ do
3:    u = Q.dequeue()
4:    partner = nextCandidate(u)
5:    while ranking(partner, u) >
      ranking(partner, suitor(partner)) do
6:       partner = nextCandidate(u)
7:    if suitor(partner) ≠ NULL then
8:       Q.enqueue(suitor(partner))
9:    suitor(partner) = u
```

McVitie and Wilson [4] gave a recursive implementation of the Gale-Shapley algorithm. This algorithm also iterates over the men, allowing each one to make a proposal to his most preferred woman. But if this proposal is rejected or if it results in an existing suitor being rejected then the just rejected man recursively makes a new proposal to his best remaining candidate. The recursion continues until a proposal is made such that no man is rejected (because the last proposed to woman did not already have a suitor). At this point the algorithm will continue with the outer loop and process the next man. When all men have been processed the algorithm is finished. It is shown in [4] that the McVitie-Wilson algorithm gives the same solution as the Gale-Shapley algorithm. We note that similarly to the Gale-Shapley algorithm it is possible to avoid proposals that are destined to be rejected because the proposed to woman already has a better offer.

Comparing the two algorithms each man will consider exactly the same women before ending up with his final partner. The only difference is the order in which this is done. While the Gale-Shapley algorithm will maintain a list of men that needs to be matched, the McVitie-Wilson algorithm will always maintain a solution where each man considered so far is matched before including a new man in the solution. We note that one can implement a non-recursive version of the McVitie-Wilson algorithm simply by replacing the queue Q in Algorithm 1 by a stack and replacing the *dequeue*() and *enqueue*() operations with *pop*() and *push*() operations respectively. To see that this will result in the McVitie-Wilson algorithm it is sufficient to first note that the initial placement of L in Q is equivalent to an outer loop that processes each man once. Any rejected man will then be placed at the top of the stack and therefore be processed immediately, similarly to a recursive call in the original algorithm.

Wilson [26] showed that for any profile of womens preferences, if the men's preferences are random, then the expected sum of men's rankings of their mates as assigned by the Gale-Shapley algorithm is bounded above by $n(1 + 1/2 + ... + 1/n)$. Knoblauch [15] showed that this is also an approximate lower bound in the sense that the ratio of the expected sum of men's rankings of their assigned mates and $(n+1)((1+1/2+...+1/n)-n)$ has limit 1 as n goes to ∞. Thus if the men's preferences are random then this sum is $\Theta(n \ln n)$ for large n. However, it is not hard to design instances where this sum is $\Theta(n^2)$. One such case is when the men have identical preferences.

2.1 Generalizations of SM We next review two generalizations of the SM problem. The stable roommates (SR) problem consists of a set of n persons, each one with a complete ranking of all the others persons. The objective is now to pair two and two persons together, such that there is no pair (x, y) of persons where x is either unmatched or prefers y to its current partner, while at the same time y is either unmatched or prefers x to its current partner. Just like for the SM problem, such a solution is stable. Unlike the SM problem there might not exist a solution for an SR instance. If some persons have only ranked a subset of the other participants we get the stable roommates problem with incomplete lists (SRI). Irving gave an algorithm for computing a stable solution to an SRI problem or to determine that no such solution exists [11]. This algorithm operates in two stages, where the first one is similar to the Gale-Shapley algorithm where each person makes, accepts and rejects proposals. The second phase of the algorithm is slightly more involved but does not change the running time of $O(n^2)$. For more information on the SR and SRI problems see [9, 16].

In the last generalization each person can be matched with more than one partner. More formally, we are looking for a stable solution to an SM, SMI, SR, or SRI instance where each person v_i is matched with at most $b(v_i)$ other persons, where $b(v_i) \geq 1$. Being stable again means that no two persons v_i and v_j would both obtain a better solution if they were to match with each other, either by dropping one of their current partners or if v_i has fewer than $b(v_i)$ partners or if v_j has fewer than $b(v_j)$ partners.

For the SM problem this gives us the many-to-many stable assignment problem (MMSA), where each

"man" and "woman" can be matched with several participants of the opposite sex. This was solved by Baïou and Balinski [2] who presented a general algorithm based on modelling this as a graph searching problem. Applying the last generalization to an SR instance, gives the stable fixtures problem [12] for which Irving and Scott gave an $O(n^2)$ algorithm. Similarly to Irving's algorithm for SRI, this also consists of two stages, where the first stage is a natural extension of the Gale-Shapley algorithm to handle that each person can participate in multiple matches.

3 Matching Problems

We next explore the relationship between stable marriages and weighted matchings in graphs. A matching M on a graph $G(V, E)$ is a subset of the edges such that no two edges in M share a common end point. For an unweighted graph the object is to compute a matching of maximum cardinality. For an edge weighted graph a typical problem is to compute a matching M such that the sum of the weights of the edges in M is maximum over all matchings. Another variant could be to compute the maximum weight matching over all matchings of maximum cardinality.

We consider the GREEDY algorithm for computing a matching of maximum weight in an edge weighted graph where all weights are positive. This algorithm considers edges by decreasing weight. In each step the heaviest remaining edge (u, v) is included in the matching before removing any edge incident on either u or v. If the weights of the edges in G are unique or if a consistent tie breaking scheme is used then it follows that the solution given by GREEDY is also unique. In the following we will always assume that this is the case. It is well known that GREEDY guarantees a solution of weight no worse than 0.5 times the weight of an optimal solution. We label the problem of computing a greedy matching in an edge weighted graph as the GM problem.

Given an instance G of the GM problem one can construct an equivalent instance of the SRI problem by sorting the edges incident on each vertex u by decreasing weight, and letting this be the ranking of u's neighbors in the SRI instance. With this construction a solution to the GM problem is equivalent to a stable solution of the corresponding SRI problem. Consider the heaviest edge (u, v) in the graph. This is included in the GM solution and the corresponding vertex pair must also be part of any solution to the SRI instance, otherwise this solution would not be stable as both u and v would prefer to match with each other over any other partner. We can thus include (u, v) in the solutions to both instances and also remove u and v from further consideration. For the GM problem this means that any edges incident on either u or v are removed and for the SRI instance u and v are removed from all ranking lists. One can then repeat the argument using the heaviest remaining edge, and it follows by induction that the two solutions are identical. It is also clear that the corresponding SRI instance always has a unique stable solution.

The above construction implies that the solution given by GREEDY is stable in the sense that there does not exist an edge $(u, v) \notin M$ such that the weight of M would increase if (u, v) was added to M while removing any edges incident on either u or v from M. This observation is often stated as that the solution given by GREEDY does not contain any augmenting path containing three or fewer edges. An augmenting path of length k is a path containing k edges starting with an edge in M and then alternating between edges not in M and edges in M, such that if one was to replace all the edges on the path that belong to M with those that are not in M then the weight of the solution would increase.

We next show that the solution given by GREEDY can also be obtained by solving an associated SMI (or SM) instance. To the best of our knowledge this result has not been shown previously.

Given an instance of the GM problem on an edge weighted graph $G(V, E)$. We define an SMI instance G' from G as follows. Let L and R be the sets of men and women respectively, both of size $n = |V|$. Any man l_i will include exactly those r_j in its ranking where there is an edge $(v_i, v_j) \in E$. As the edges in G are not directed, this also means that l_j will rank r_i. Similarly, any woman r_j will include exactly those l_i in her ranking where there is an edge $(v_i, v_j) \in E$. Both men and women order their lists by decreasing weight of the corresponding edges in G. Thus every $(v_i, v_j) \in E$ gives rise to four rankings in G'. We call the two pairs (l_i, r_j) and (l_j, r_i) for the corresponding pairs of (v_i, v_j).

LEMMA 3.1. *Given a graph G with SMI instance G' as described above and let M be the greedy matching on G. Then the pairs in G' corresponding to the edges in M make up the unique solution to the SMI problem on G'.*

Proof. The proof is by induction on the edges of M considered by decreasing weight. Let (v_i, v_j) be the edge of maximum weight in G. Then $(v_i, v_j) \in M$ and it also follows from the construction of G' that l_i will rank r_j highest. Similarly, l_i will also have the highest ranking among the men ranked by r_j. Thus (l_i, r_j) must be included in any stable solution of G'. A similar argument shows that the edge (l_j, r_i) will also be included in such a solution.

Assume now that the pairs in G' corresponding to the $k \geq 1$ heaviest edges in M must be included in

any stable solution and consider the two pairs (l_s, r_t) and (l_t, r_s) corresponding to the $k+1$st heaviest edge (v_s, v_t) in M.

It is clear that any solution where l_s is matched to a woman that he has ranked after r_t while at the same time r_t is matched to a man that she has ranked after l_s, cannot be stable as both l_s and r_t would be better of if they were to match with each other. Thus if (l_s, r_t) is not included in a stable solution at least one of l_s and r_t must be matched to a partner which he or she has ranked higher than the other one of $\{l_s, r_t\}$. Assume therefore that l_s is matched to r_u and that l_s has ranked r_u higher than r_t, implying that the weight of (v_s, v_u) is greater than the weight of (v_s, v_t) in G. But since $(v_s, v_t) \in M$ it follows that $(v_s, v_u) \notin M$. Thus v_u must be matched to some other vertex v_z in M. And since $(v_s, v_u) \notin M$ the weight of (v_u, v_z) must be greater than that of (v_s, v_u). By the induction hypothesis the pairs in G' that correspond to the k heaviest edges in M must be included in any stable solution in G'. It therefore follows that l_z must be matched to r_u in any stable solution on G' contradicting that l_s is matched to r_u. A similar argument shows that r_t cannot be matched to any man in L to which she gives higher priority than she gives to l_s. Thus the pair (l_s, r_t) must be in any stable solution in G'. The argument for why (l_t, r_s) also must be included in a stable solution is analogous. It follows that any pair in G' corresponding to an edge in M must be part of a stable marriage in G'.

It only remains to show that once the pairs corresponding to the edges in M have been included in the solution M' to G', then it is not possible to match any other pairs in G'. If M' contains a pair (l_i, r_j) in addition to the pairs corresponding to the edges in M then $(v_i, v_j) \notin M$ and neither v_i nor v_j can be matched in M. But since l_i has ranked r_j (and vice versa) it follows that $(v_i, v_j) \in E$ and that M can be expanded with (v_i, v_j). This contradicts that M is maximal and the result follows.

We next consider the b-matching problem which is a generalization of the regular weighted matching problem similar to the many-to-many stable assignment problem and the stable fixtures problem. A b-matching on G is a subset of edges $M \subseteq E$ such that every vertex $v_i \in V$ has at most $b(v_i)$ edges in M incident on it. The objective is to compute the b-matching of maximum weight. A 0.5 approximation can again be computed using the greedy algorithm that selects edges by decreasing weight and whenever $b(v_i)$ edges incident on v_i have been selected, the remaining edges incident on v_i are removed [19]. Setting $b(v_i) = 1$ for all $v_i \in V$ gives a regular (one) matching.

It is straight forward to see that the stable fixtures problem is also a generalization of greedy b-matching. Given an instance of the greedy b-matching problem, one can also construct an equivalent many-to-many stable assignment instance by setting the bounds $b(l_i)$ and $b(r_i)$ equal to $b(v_i)$. A proof similar to that of Lemma 3.1 shows that these two problems have equivalent solutions.

3.1 Algorithmic Similarities As a consequence of the fact that the solution given by GREEDY can be obtained by either solving a properly designed instance of SMI or SRI, any algorithm that solves either of these two problems can also be used to compute a greedy weighted matching. This process can be simplified as it might be possible to run an SMI or SRI algorithm directly on the original graph. Let G be an instance of GM and G' its corresponding SMI instance. Also let $\{r_{s_1}, r_{s_2}, \ldots, r_{s_f}\}$ and $\{l_{t_1}, l_{t_2}, \ldots, l_{t_g}\}$ be the ranked lists of l_i and r_i respectively. Then if follows from the construction of G' that $f = g$ and that $s_k = t_k$ for all k. Thus any proposal made to r_i could be handled directly by l_i as he has the same information as r_i. It follows that one can merge l_i and r_i into one node v_i that handles making, accepting, and rejecting proposals related to l_i and r_i. In this way both the Gale-Shapley and the McVitie-Wilson algorithm can be used directly on edge weighted general graphs to compute greedy matchings, but now using edge weights to rank potential partners. Irving's algorithm [11] for solving the SRI problem consists of two stages, of which the first is exactly this algorithm used on a general graph. If the rankings in an SRI instance are based on edge weights from a GM instance then the first phase will produce the greedy solution which is stable, thus making the second phase of the algorithm redundant.

Previous efforts at designing fast parallel greedy matching algorithms have been based on the notion of dominant edges. These are edges that are heavier than any of their neighboring edges. Preis showed that an algorithm based on repeatedly including dominant edges in the matching while removing any edges incident on these will result in the same solution as GREEDY [22]. Based on this observation Manne and Bisseling developed the pointer algorithm [17], which was further enhanced by Manne and Halappanavar in the SUITOR algorithm [18]. We note that the SUITOR algorithm is identical to the McVitie-Wilson algorithm applied to a general edge weighted graph, while the pointer algorithm has strong resemblances to the Gale-Shapley algorithm as outlined in Algorithm 1.

The same type of relationship also holds true between the greedy b-matching problem and the many-to-many stable assignment problem. The algorithm pre-

sented in [2] can be instantiated to solve the b-matching problem using a Gale-Shapley type algorithm where a vertex v will accept the $b(v)$ best offers at any given time. We note that this is the same algorithm as the one presented in [7] and also [13] for computing a greedy b-matching. In [13] the authors experiment with what they call *delayed* versus *eager* rematching of rejected suitors. The difference between these two variants is the same as that between a Gale-Shapley and a McVitie-Wilson style algorithm.

4 Experiments

As shown in Section 3 much of the theory for greedy matching algorithms are mainly restricted versions of previous results from the theory of stable marriages. However, the work on greedy matchings has to a large extent been driven by a need for developing scalable parallel algorithms for use in scientific applications. This has lead to the implementation of Gale-Shapley and McVitie-Wilson type matching algorithms on a large variety of architectures, including distributed memory machines [3, 17], multicore computers [10, 14, 18], and GPUs [1, 20].

There has been less emphasis on implementations and developing working code for the stable marriages problem. We believe that much of the work done on greedy matchings can easily carry over to developing efficient code for stable marriage problems. To show the feasibility of this we have developed shared memory implementations of both the Gale-Shapley and McVitie-Wilson algorithms. We used OpenMP to parallelize the Gale-Shapley algorithm and both OpenMP and CUDA for parallelizing the McVitie-Wilson algorithm.

In weighted matching both endpoints of an edge (u, v) evaluates the importance of the edge to the same number, i.e. the weight of the edge. Whereas in the stable marriage problem both u and v assign their own ranking of the other. Thus the main difference between greedy matching algorithms such as those presented in [17, 18] and the Gale-Shapley algorithm is that in the latter, a man who makes a proposal evaluates his chance of success based on the woman's ranking, instead of on a common value. Another difference is that it is typically not assumed in weighted matching problems that the neighbor list of a vertex is sorted by decreasing weight. It was shown in [18] that when this is the case, then it both simplifies the algorithm and also speeds up the execution considerably.

Our parallelization strategy for the McVitie-Wilson algorithm using OpenMP closely follows that of the Suitor algorithm as presented in [18], while our CUDA version of the same algorithm is a simplified version of the Suitor algorithm used in [20]. In both of our OpenMP algorithms the set of men is initially partitioned among the threads who then each run a local version of the corresponding algorithm until completion. A thread will first search the list of the current man to locate the woman he gives highest priority and where the woman also prefers him to her current suitor (if any). If such a woman is discovered the thread will use a compare-and-swap (CAS) operation to become the new suitor of the woman. In this way it is assured that no other thread has changed the suitor value. If the CAS operation succeeds the previous suitor (if any) is treated according to the current strategy and is inserted in a local stack (McVitie-Wilson) or a local queue (Gale-Shapley). If the CAS operation failed because some other thread had already changed the suitor value, then if the current man can still beat the new suitor then the thread will retry with a new CAS operation, otherwise it will continue searching for the next eligible woman.

There is a difference between the algorithms in how they can handle load imbalance. For the parallel Gale-Shapley algorithm it is possible to synchronize the threads after each round of proposals and then redistribute the unmarried men to the threads before moving on to the next round. However, synchronization tends to be costly, and experiments done on greedy matching problems indicate that this is typically not worth the effort. For the McVitie-Wilson algorithm one can load balance the algorithm by using one of the dynamic load balancing strategies in OpenMP in the initial assignment of men to threads. This strategy was used successfully in experiments for the Suitor matching algorithm on graphs with highly varying vertex degrees [18].

For the McVitie-Wilson CUDA algorithm we assign one thread to each man.Each thread then executes the algorithm similarly to the OpenMP version using a CAS operation to assign a man as the suitor of a particular woman. Using only one thread per man allows for a larger number of thread blocks which the runtime environment can balance across the device. But as the threads within one physical warp operate in SIMD, the run time of all threads in the same warp will be equal to the maximum execution time of any of the threads. Similarly, the threads within the same thread block will not release resources until all threads in the block have finished executing. It would have been possible to statically assign multiple men to each thread or to design a dynamic load balancing scheme with the aim of evening out the work load. But this would have resulted in a more complicated algorithm and as our main goal is proof of concept we did not pursue this.

Implementing the Gale-Shapley algorithm on the GPU presents additional challenges compared to the McVitie-Wilson algorithm. In a Gale-Shapley algorithm

the threads would have to be grouped so that each thread group operates on one common queue, where the size of the group could be either a subset of threads in a warp or all the threads in one thread block. As the number of free men monotonically decreases between rounds, there should initially be more men than threads assigned to the same queue, something that would complicate the algorithm. Also, having several threads operate on one common queue would require synchronization which can be time consuming on the GPU. For this reason we chose not to implement the Gale-Shapley algorithm using CUDA.

As we are not aware of any sufficiently large publicly available data sets for the stable marriage problem, we have designed two different random data sets. The first set has been constructed to be relatively easy to solve, whereas the second set is intended to be more time consuming. We label these sets as *easy* and *hard* respectively. Each instance consists of n men and n women. In the *easy* data set each man is assigned a random number $\epsilon \in [0, 1]$ and then randomly picks and ranks $(1 + \epsilon) \ln n$ women. Each woman then ranks exactly the men that ranks her. With this configuration more than 98% of the participants were matched in every final solution and the total number of proposals is at most $2n \ln n$ with an average of $n \ln n$. In the *hard* data set each man has an identical complete random ranking of all the women. Similarly, all woman share the same random ranking of all the men. Thus there will always exist a complete stable solution and the total number number of considered women will always be $n(n + 1)/2$. Moreover, in the *hard* instances there will be contention among the men for obtaining the same set of women, and thus cause substantial synchronization requirements for the parallel algorithms. One obvious difference between the datasets is that the *easy* instances will require more memory access as each participant has an individual ranking list, while for the *hard* instances all rankings are stored in two shared vectors of length n. For each value of n we have generated 5 instances and for each of these we run each algorithm 3 times. For all timings we take the average of these 15 runs.

The OpenMP algorithms are run on a computer with two Intel Xeon E5-2699 processors and 252 Gbytes of memory. Each processor has 18 cores and runs at 2.30GHz. The GPU is a Tesla K40m with 12 GB of memory, 2880 cores running at 745 MHz and has CUDA compute capability version 3.5. For all parallel algorithms we measure their speedup against the fastest sequential algorithm run on the Intel Xeon machine.

In Figure 1 we present results from the *easy* instances when n varies from 5M up to 25M in steps of 5M. For the OpenMP algorithms the number of threads is set to 36. For most of these instances the running time stays well below one second. It is only for the n =25M instance that the GPU algorithm uses slightly more time than one second. This is also the largest *easy* instance that could be run on the GPU. For smaller instances the

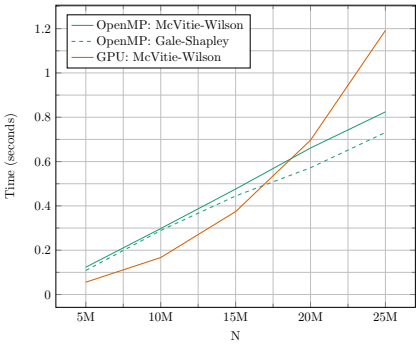

Figure 1: Running time on the *easy* dataset

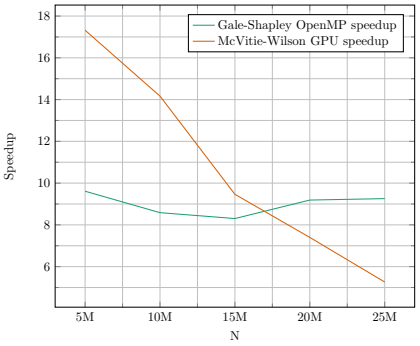

Figure 2: Speedup on the *easy* dataset

GPU algorithm is the fastest one but as the problem size increases it is slowing down compared to the OpenMP algorithms. In general the OpenMP Gales-Shapley algorithm is faster than the OpenMP McVitie-Wilson algorithm with as much as 12%. For this setup one would expect that the graph displaying the time would resemble $n \ln n$ as the computing resources is the same for each instance. This is most true for the OpenMP algorithms where the time increases close to linearly with n, whereas the time grows faster than n for the GPU algorithm. This can be seen further in Figure 2 which shows the speedup of the OpenMP Gale-Shapley algorithm and the GPU McVitie-Wilson algorithm compared to

the sequential Gale-Shapley algorithm. The OpenMP algorithm gives a constant speedup of about 9, while the speedup of the GPU algorithm starts out at about 17 but then drops sharply as the size of the instances increase. Thus this is most likely due to insufficient memory on the GPU. On these problems the sequential McVitie-Wilson algorithm was on average 27% slower than the sequential Gale-Shapley algorithm.

Figure 3 shows running times of the OpenMP algorithms using 36 threads as n increases up to 125M. It can be observed that the tendencies for the smaller instances still remain true for the larger ones. We note that the worst running time is only marginally larger than four seconds on the largest instance. Figure 4 shows the speedup of the OpenMP algorithms compared to the sequential Gale-Shapley algorithm for the three largest instances when using $t = 1, 9, 18, 27$ and 36 threads. The Gale-Shapley algorithm outperforms the McVitie-Wilson algorithm in almost all instances and reaches a speedup of almost 14 on the $n = 75$M instance.

Figure 5 shows the running time on *hard* instances where n increases from 100K up to 500K. The OpenMP codes are again run using 36 threads. As the dataset only consists of two vectors we can run the problems using all three codes, the only limiting factor being time. Since the total amount of work grows as $\Theta(n^2)$ on these instances, it is to be expected that they will require more time than the *easy* ones. From the figure it can be observed that there is little difference in the running time between the Gale-Shapley OpenMP code and the McVitie-Wilson GPU code, which both take close to 250 seconds on the largest instance. However, the McVitie-Wilson OpenMP code performs considerably better, and is a factor of 5 times faster on the largest instance. This difference is also displayed in Figure 6 which gives the speedup of the same instances compared to the sequential McVitie-Wilson algorithm. While the McVitie-Wilson algorithm reaches a speedup of close to 22 when running on the 36 threads, the parallel Gale-Shapley algorithm is never more than a factor of 2.5 faster than the sequential McVitie-Wilson algorithm. For these instances the sequential Gale-Shapley algorithm was on average 109% slower than the sequential McVitie-Wilson algorithm. We believe that some the difference between the OpenMP algorithms can be explained by how the algorithms handle the large number of rejections. While the Gale-Shapley algorithm has to store each rejected man to memory and retrieve a new one, the McVitie-Wilson algorithm can continue working on the rejected man without needing to access relatively slow memory. The poor performance of the McVitie-Wilson GPU algorithm compared to the OpenMP one is most likely due to how the machines

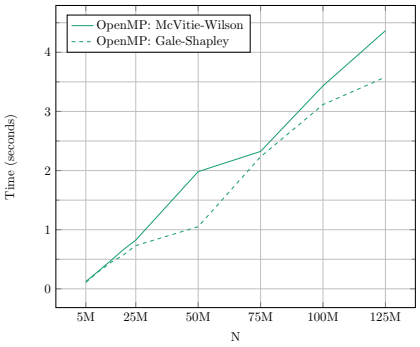

Figure 3: Running time of OpenMP algorithms on large *easy* instances

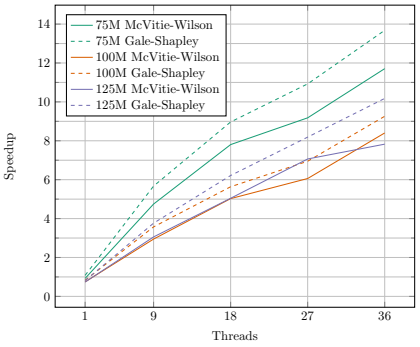

Figure 4: Speedup on large *easy* datasets

handle contention for shared resources. The GPU algorithm utilizes several thousand concurrent threads that, at least initially, will be competing for matching their man with the same set of women. Synchronizing this will lead to a much larger strain on the system compared to that of the relatively low number of threads in the OpenMP algorithm.

Finally, figures 7 and 8 show the number of considered proposals per second for both *easy* and *hard* datasets on the OpenMP algorithms. For each instance this number is given as the sum over each man of his ranking of his final partner and then divided by the total time. In sparse graph algorithms this is often referred to as the number of traversed edges per second (TEPS) and is, among other things, used to rank the performance of computers in the Graph500 challenge [8].

For the *easy* instances the TEPS rate starts out at 10M for one thread and then increases to somewhere

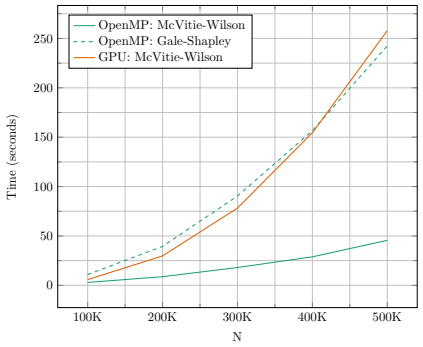

Figure 5: Running time on *hard* datasets

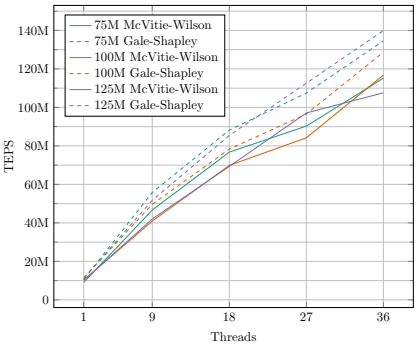

Figure 7: TEPS for *easy* datasets

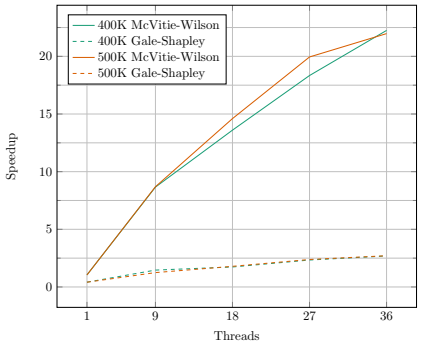

Figure 6: Speedup on *hard* datasets

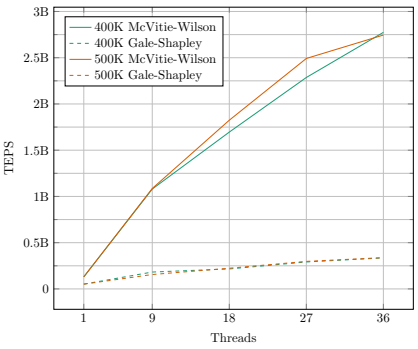

Figure 8: TEPS for *hard* datasets

between 100M to 140M for 36 threads. Thus the efficiency when using 36 threads lies somewhere in the range of 30% to 40%. For the *hard* instances the TEPS rate for the McVitie-Wilson algorithm starts out at about 150M and increases up to 2.75 billion when using 36 threads for an efficiency rate of about 50%. As already noted the Gale-Shapley algorithm does not scale well on these instances. Comparing the TEPS rate between the *easy* and the *hard* instances when using the McVitie-Wilson algorithm on the same number of threads it can be observed that the maximum TEPS rate is more than a factor of 20 larger for the *hard* instances. This is most likely because the *hard* instances are not limited by access to memory as the whole dataset only consists of two vectors.

5 Conclusion

In his book Manlove [16] lists some of the most noteworthy open problems related to SM. One of these is to determine if the SM problem is in the complexity class NC or not, that is, to determine whether the problem can be solved by an algorithm with polylogarithmic running time when using a polynomial number of processes. Efforts at designing such algorithms has mainly resulted in parallel algorithms requiring at least n^2 processes, and are thus mainly of theoretical interest [6, 25].

We are only aware of one previous attempt at implementing a parallel version of the Gale-Shapley algorithm and this did not result in any speedup [24]. Quinn [23] argues that one cannot expect a large speedup from a parallel Gale-Shapley style algorithm in practice as the algorithm cannot run faster than the maximal number of proposals made by any one man. We note that for a random instance the average number of proposals made by each man is in fact $O(\log n)$.

While the question of developing asymptotically faster parallel algorithms than those presented in Section 4 is of interest from a theoretical point of view,

we believe that this is less relevant for a practitioner. To begin with the running time of the Gale-Shapley algorithm is linear in the instance size. Thus moderate sized problem can already be solved rapidly. In addition, our current experiments on the SMI problem as well as previously experiments on GM problems shows that Gale-Shapley type algorithms scale well. One reason for this is that the size of the instance n is typically much larger than the number of threads used.

One notable difference between the formulations of the GM and the SMI problem is that for GM it is not assumed that the neighbor lists are initially sorted by decreasing weight in the same way as priority lists are ordered in SM. Thus work on developing parallel algorithms for the GM problem has focused on how one should search the neighbor lists. Suggested solutions include sorting the lists initially, searching through the list each time a new candidate is needed, or something in between. All of these strategies result in a running time that is superlinear in the input size. However, Preis's algorithm for GM has linear running time [22], but is more complicated and not suitable for parallel execution. We therefore ask if it is possible to design a linear time algorithm for the SM problem if the priority lists are not sorted, but instead given as real valued numbers such that $p_i(j)$ gives the value that person i assigns to person j of the opposite sex.

References

[1] B. O. Fagginger Auer and R. H. Bisseling. A GPU algorithm for greedy graph matching. In *Facing the multicore, Challenge II*, volume 7174, pages 108–119. LNCS, 2012.

[2] M. Baïou and M. Balinski. Many-to-many matching: stable polyandrous polygamy (or polygamous polyandry). *Disc. App. Math.*, 101(1-3):1–12, 2000.

[3] Ü. V. Çatalyürek, F. Dobrian, A. H. Gebremedhin, M. Halappanavar, and A. Pothen. Distributed-memory parallel algorithms for matching and coloring. In *IPDPS Workshops*, pages 1971–1980, 2011.

[4] L. B. Wilson D. G. McVitie. The stable marriage problem. *Comm. of the ACM*, 14(7):486–490, 1971.

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

[6] T. Feder, N. Megiddo, and S. A. Plotkin. A sublinear parallel algorithm for stable matching. *Theor. Comput. Sci.*, 233(1-2):297–308, 2000.

[7] G. Georgiadis and M. Papatriantafilou. Overlays with preferences: Distributed, adaptive approximation algorithms for matching with preference lists. *Algorithms*, 6(4):824–856, 2013.

[8] Graph 500. `http://www.graph500.org`.

[9] D. Gusfield and R. W. Irving. *The stable marriage problem.* The MIT press, 1989.

[10] M. Halappanavar, J. Feo, O. Villa, A. Tumeo, and A. Pothen. Approximate weighted matching on emerging manycore and multithreaded architectures. *Int. J. High Perf. Comput. App.*, 26(4):413–430, 2012.

[11] R. W. Irving. An efficient algorithm for the stable roommates problem. *J. Alg.*, 6(4):577–595, 1985.

[12] R. W. Irving and S. Scott. The stable fixtures problem - A many-to-many extension of stable roommates. *Disc. App. Math.*, 155(16):2118–2129, 2007.

[13] A. Khan, A. Pothen, M. M. A. Patwary, N. R. Satish, N. Sundaram, F. Manne, M. Halappanavar, and P. Dubey. Efficient approximation algorithms for weighted b-matching. *SIAM J. Sci. Comput.*

[14] A. M. Khan, D. F. Gleich, A. Pothen, and M. Halappanavar. A multithreaded algorithm for network alignment via approximate matching. *SC*, page 64, 2012.

[15] V. Knoblauch. Marriage matching: A conjecture of Donald Knuth. Economics Working Papers, http://digitalcommons.uconn.edu/econ_wpapers/200715, 2007.

[16] D. Manlove. *Algorithmics of matching under preferences.* World Scientific, 2013.

[17] F. Manne and R. H. Bisseling. A parallel approximation algorithm for the weighted maximum matching problem. In *PPAM'08*, volume 4967 of *LNCS*, pages 708–717, 2008.

[18] F. Manne and M. Halappanavar. New effective multithreaded matching algorithms. In *IPDPS*, pages 519–528, 2014.

[19] J. Mestre. Greedy in approximation algorithms. In *Algorithms - ESA 2006*, volume 4168 of *LNCS*, pages 528–539. 2006.

[20] Md. Naim, F. Manne, M. Halappanavar, A. Tumeo, and J. Langguth. Optimizing approximate weighted matching on nvidia kepler K40. In *HiPC 2015*, pages 105–114, 2015.

[21] U. Naumann and O. Schenk. *Combinatorial Scientific Computing.* CRC Press, 2012.

[22] R. Preis. Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs. In *STACS'99*, volume 1563, pages 259–269. LNCS, 1999.

[23] M. J. Quinn. *Designing efficient algorithms for parallel computers.* McGraw-Hill, 1987.

[24] J. L. Träff. A parallel approach to the stable marriage problem. In *NOAS'97*, pages 277–287, 1997.

[25] C. White and E. Lu. An improved parallel iterative algorithm for stable matching. Extended Abstract, Companion of IEEE/ACM International Conference for High Performance Computing, Networking, Storage and Analysis (SuperComputing 2013).

[26] L. B. Wilson. An analysis of the marriage matching assignment algorithm. *BIT*, 12:569–575, 1972.