International Journal of Computer Applications (0975 - 8887)
Volume 186 - No.43, September 2024

Selection Improvements on the Parallel Iterative Improvement Algorithm for Stable Matching

Scott Wynn
Department of Computer Science and Engineering
University of Washington, Seattle, WA 98185, USA

Alec Kyritsis
Department of Computer Science
Middlebury College, Middlebury, VT 05753, USA

Stephora Alberi
Department of Computer Science
Salisbury University, Salisbury, MD 21801, USA

Enyue Lu
Department of Computer Science
Salisbury University, Salisbury, MD 21801, USA

ABSTRACT

Sequential algorithms for the Stable Matching Problem are often too slow in the context of some large scale real-time applications like switch scheduling. Parallel architectures can offer a notable decrease in runtime complexity. This paper proposes a stable matching algorithm that runs in $O(nlog(n))$ time using n^2 processors. The proposed algorithm is structurally based on the Parallel Iterative Improvement (PII) algorithm, where we suggest alternative selection methods for pairs, called Right-Minimum and Dynamic Selection, as well as a faster preprocessing step, called Quick Initialization. The proposed algorithm improves the convergence rate from 90% in the original PII algorithm to 100% in the proposed algorithm over more than 3.6 million trials and significantly improves runtime.

General Terms

Stable Matching, Parallel Computing, High Performance Computing, Algorithms

Keywords

Stable Matching, Parallel Algorithms, Parallel Iterative Improvement Algorithm, Matching Algorithms, Preference Matrices, Blocking Pairs, Right-Minimum Selection, Dynamic Selection

1. INTRODUCTION

Originally developed to model the college admissions process, the Stable Matching Problem [6] has spawned numerous applications in the social, physical and computational sciences [1, 4, 5, 8, 14]. One such application, switch scheduling in large data centers, is of particular interest due to the exponential increase in network throughput over the recent decade [18].

In the context of these large scale implementations, the classic Gale-Shapley algorithm from [6] can be too slow, requiring $O(n^2)$ runtime to complete. Parallel and Distributed architectures can greatly decrease the computational complexity of solving the Stable Matching Problem [3, 7, 16]. Without considering some of the practical constraints on parallel architecture, [3] showed that a sublinear algorithm for stable matching is possible. This paper studies an augmented version of the Parallel Iterative Improvement (PII) algorithm proposed in [12], which improves on previous ideas in [17] and preliminary work in [10] and offers significant runtime improvement while requiring only $O(n^2)$ processors, making the parallel architecture required for the algorithm much more feasible for large scale applications.

In its initial implementation, the PII algorithm finds a stable matching within $5n$ iterations and a total of $O(n\ log(n))$ runtime in approximately 90% of cases when tested on input size of $n = \{10, 20, ..., 100\}$. In the remaining 10% of cases, the algorithm cycles indefinitely, repeatedly returning to a previous iteration. This paper shows that applying new iteration methods can significantly improve runtime and convergence rate of the PII algorithm.

1.1 Preliminaries

Denote the sets of n men and n women by $M = \{m_1, m_2, ..., m_n\}$ and $W = \{w_1, w_2, ..., w_n\}$ respectively. Let P be the $n \times n$ matrix, or *preference matrix*, with each row i corresponding to the ith element of M and each column j corresponding to the jth element of W, as used in [12]. Hence, $p_{i,j} \in P$ represents the respective rankings of man i with woman j. For a given pairing $p_{i,j}$, denote the preference ranking of man i for woman j and the preference ranking of woman j for man i by $L(p_{i,j})$ and $R(p_{i,j})$ respectively. Denote these quantities the *left value* and *right value* of $p_{i,j}$ respectively. Then, man i prefers woman m to woman j if $L(p_{i,m}) < L(p_{i,j})$, and similarly woman j prefers man l to man i if $R(p_{l,j}) < R(p_{i,j})$.

Let μ be a one-to-one mapping between M and W. Then, μ is called a *matching* and $p_{i,j}$ is considered to be *matched* under μ if and only if $\mu(i) = j$ and $\mu(j) = i$. For convenience, this

paper will often say $p_{i,j} \in \mu$. For a given matching μ, if man i and women j prefer each other to their current partners, then $p_{i,j}$ is a *blocking pair* and the current pairs are called *unstable pairs*. Formally, in a preference matrix P, a pair $p_{i,j}$ is a blocking pair with respect to a matching μ and unstable pairs $p_{i,l}, p_{m,j} \in \mu$ if and only if $L(p_{i,j}) < L(p_{i,l})$ and $R(p_{i,j}) < R(p_{m,j})$. If a matching μ on preference matrix P contains any unstable pairs, μ is classified as an unstable matching.

Finally, let $\mu_1, \mu_2, ..., \mu_n$ denote a set of matchings on preference matrix P. Suppose μ_k is an unstable matching with blocking pair $p_{i,j}$. Then μ_{k+1} *satisfies* $p_{i,j}$ if and only if $p_{i,j} \in \mu_{k+1}$. In the following sections it will be useful to track general sequences of blocking pairs.

1.2 Parallel Architecture

This paper assumes n^2 processors (PE's) with hypercube or multiple broadcast bus architecture. This enables row and column broadcasts and find minimum operations to be completed with at most $O(log(n))$ complexity, as outlined in [12]. It may be useful to consider the PE's arranged on an $n \times n$ mesh A such that $PE_{i,j}$ corresponds to pair $p_{i,j}$ in preference matrix P.

1.3 Statement of Results

This paper's main contribution to the problem lies in the presentation and analysis of two new pair selection methods for the iteration step of the PII algorithm that significantly improve the convergence rate.

Method 1: The first proposed method is Right-Minimum Selection, which forces termination of the algorithm by only choosing blocking pairs in which one side's preferences improve.

Method 2: The second proposed method is Dynamic Selection, which optimally chooses the subsequent matching by considering all previously selected blocking pairs as a strong combination.

This paper also proposes a novel initialization method, which runs in $O(n)$ time and improves the proposed algorithm's convergence speed.

The resulting new variant of the PII algorithm converges faster and more often than previous iterations of the PII algorithm (showing 100% convergence over more than 3.6 million trials across different n values), and the convergence rate scales significantly better with higher values of n.

1.4 Overview

The rest of the paper will proceed as follows. Section 2 will discuss related work, including a brief overview of the PII Algorithm and previous work based on the PII Algorithm. Section 3 presents the proposed new selection methods, Right-Minimum and Dynamic Selection, as well as the new preprocessing method, Quick Initialization. Section 4 presents the proposed modification of the original PII algorithm using the new methods. Section 5 analyzes the results of the proposed methods and modified algorithm, and Section 6 outlines future work.

2. PREVIOUS WORK

2.1 Iterative Stable Matching Algorithms

When observed in real life through social networks, matchings attempt to converge toward stability through pairing together blocking pairs. Roth and Vande Vate have shown that swapping along blocking pairs at random from any initial matching will almost surely result in a stable matching eventually [13]. However, it has also been shown that iterative methods for swapping along specific blocking pairs each iteration may result in indefinite cycling, resulting in a failure to ever converge to stability [9].

2.2 The PII Algorithm

The PII Algorithm presented in [12] is one such iterative algorithm, which uses parallel architecture to improve convergence runtime. The algorithm begins with a randomly generated matching μ_0. The algorithm then executes the following steps each iteration, terminating when a stable matching is achieved:

1) Find all blocking pairs

2) In each row of M with a blocking pair, select the one with the lowest left value as an NM1-generating pair

3) In each column of M with an NM1-generating pair, select the one with the lowest right value as an NM1 pair

4) Remove all pairs in the current matching in the same row or column as an NM1 pair, and add all NM1 pairs to the matching

5) Fill in any open columns and rows with pairs as described in [12].

The PII algorithm requires n^2 parallel processors, and each iteration will complete in $O(log(n))$ complexity. Lu empirically observed that the convergence rate of the PII algorithm on randomly generated preference matrices steadily decreases from 99% at $n = 10$ to 86% at $n = 100$. In cases where the PII algorithm did not converge, it continued cycling indefinitely.

2.3 The PII-SC Algorithm

White proposed the Smart Initialization and Cycle Detection methods to improve the convergence of the PII Algorithm in [17].

In Smart Initialization, each man m_i in M proposes to his top choice woman. If multiple men propose to the same women, she is matched with the man she most prefers. Those men left unmatched then propose to their next choice woman, excluding any women matched in a previous round, and a similar process ensues. This occurs until all men have been matched with a woman, and runs with $O(n\ log(n))$ complexity in parallel as described in [17].

White observed that the majority of cycles were formed by a single $NM1$ pair was alternating between pairs in two distinct sub stable matchings each iteration. A stable matching could then be constructed by including every other $NM1$ pair in a cycle and the set of pairs that remained in the matching throughout the cycle. Some more prominent edge cases for detecting cycles were also added to the cycle detection method proposed in [17].

The PII-SC Algorithm combines the PII Algorithm with Smart

Initialization and Cycle Detection, and fails to converge in approximately 1 in 1 million randomly generated preference matrices from $n = 10$ to $n = 100$, with failure to converge significantly higher than 1 in 1 million for $n = 100$, suggesting some scalability issues. Due to the existence of numerous cases where cycles were not formed as described above, the PII-SC algorithm and subsequent improvements have continued improving the observed convergence by patching edge cases. As a result, the convergence rate of the PII-SC algorithm decreases significantly on larger n, as more frequent edge cases occur, illustrated by the results in [17].

3. SELECTION METHODS

This section contains the main contribution to the line of PII algorithms: Updated selection methods.

3.1 Right-Minimum-Selection

This paper proposes a novel selection method to enhance the algorithm's efficiency, termed Right-Minimum-Selection. Define **Right-Minimum-Selection** as follows:

DEFINITION 1 RIGHT-MINIMUM-SELECTION. *Let S be a preference matrix with n men and n women, and let μ be an unstable matching on S. Suppose $p_{i,j}$ is an unstable pair on μ with corresponding blocking pair $p_{i,l}$. Then the algorithm will consider $p_{i,l}$ as a potential NM1-generating pair if and only if $R(p_{i,j}) > R(p_{i,l})$.*

Informally, Right-Minimum selection will only select NM1-generating pairs where for a given row the woman prefers her new matching, demonstrated in figure 1.

Often, the PII algorithm was observed to keep unstable pairs in the matching by satisfying a blocking pair that does not improve the matching overall, leading to cycles as described in [12]. Intuitively, Right-Minimum selection forces the algorithm to terminate by continually iterating on the matching in favor of women similar to the process in a similar manner to the original Gale-Shapley algorithm. It should be noted that a Left-Minimum selection defined in a similar manner would yield similar results (with random initialization).

Right-Minimum Selection leads to the following results justifying the algorithm will terminate:

LEMMA 2 $NM1$-CYCLE FREENESS. *Let A be an instance of the PII algorithm. If A uses Right-Minimum-Selection over its entire duration, then A is $NM1$-cycle free.*

PROOF. Let $S = \{p_1, p_2, ..., p_k\}$ be a sequence of NM1-pairs such that p_{i+1} is a blocking pair of p_i. Observe that p_{i+1} may only replace p_i if it appears in an identical row or column. If replacement occurs in a column, then $R(p_{i+1}) < R(p_i)$ by the definition of a blocking pair. If replacement occurs in a row, $R(p_{i+1}) < R(p_i)$ by the definition of Right-Minimum Selection. Thus, a chain of NM1 pairs R satisfying $R(p_1) > R(p_2) > ... > R(p_k)$ is produced. For a cycle to occur, p_k must return to a row or column of p_1, violating the monotonicity of R and the results follow. □

The following result have also been proven:

LEMMA 3. *Let $S = \{p_1, p_2, ..., p_k\}$ be an $NM1$-path taken by the algorithm. Then each time S is traversed, it's length decreases by at least two.*

PROOF. Let P be an $n \times n$ preference matrix and S be an $NM1$ path such that $S = \{p_1, p_2, ..., p_k \;\; p_i \in S\}$. Assume that after traversing P once, the algorithm arrives at v_k. Observe that by definition v_k may not be replaced by an NM1 or NM2 pair. Since v_{k-1} lives in a row or column of v_k, upon a second traversal of S the algorithm may not arrive at v_{k-1} since it would have to replace v_k. Now define $S' = S \setminus \{v_k, v_{k-1}\}$ and observe that S' is also an $NM1$ path. Then, by similar logic from above, P' must also satisfy this property. Hence, a sequence of paths $P \supset P' \supset P'' \supset ...$ can be constructed such that the cardinality of each path decreases by at least two upon each traversal.

□

Combining Lemmas 2 and 3, the following theorem can be proven:

THEOREM 4 CYCLE FREENESS. *Let A be an instance of the PII algorithm. If A uses Right-Minimum Selection over its entire duration, then A is cycle free.*

PROOF. Assume the contrary. Lemma 2 implies that such a cycle must be composed of both $NM1$ and $NM2$ pairs. Consider the $NM1$ path S that takes the shortest iterations to traverse in the cycle with length k. Suppose S spawns $NM2$ pair u_i at $NM1$ pair $p_i \in S$. By lemma 3 and the assumption of cycling, S may be traversed until v_i is deleted. Then p_i may not spawn u_i, contradiction.

□

Complexity
It is straightforward to see that the additional check in each step when choosing NM1-generating pairs can be achieved in $O(1)$ time, and thus the per iteration time complexity remains $O(log(n))$.

Note that while Right-Minimum Selection has the strong property that it may not cycle, in practice it does not necessarily yield a stable matching and often is supplemented with a standard $NM1$-selection method. This occurs when all blocking pairs remaining in a matching do not satisfy the condition to be considered under Right-Minimum Selection.

It has been empirically determined that, if Right-Minimum Selection terminates without successfully finding a stable matching, there are very few blocking pairs remaining and continuing with the original PII selection method often finds a stable matching.

3.2 Dynamic Selection

This paper also proposes Dynamic Selection, a novel algorithm for cycle prevention. Unlike cycle detection, which requires original PII method selection, Dynamic Selection is compatible with Right-Minimum Selection. Define **Dynamic Selection** as follows:

DEFINITION 5 DYNAMIC SELECTION. *Let S be a preference matrix with unstable matching μ_k. Assume NM1 selection as proposed in the original PII algorithm in [12], and let N_i be the set of NM1 pairs in row i the algorithm chooses over matchings $\mu_1, \mu_2, ..., \mu_{k-1}$. Suppose $p_{i,j}$ is a blocking pair of μ_k. Then $p_{i,j}$ is an NM1-generating pair if and only if $L(p_{i,j}) < \min\{L(p), p \in N_i\}$.*

Intuitively, in an iteration where Dynamic Selection is used, the algorithm will only select a blocking pair as a new NM1-generating pair if its left value is less than all previously selected NM1-generating pairs, as shown in figure 2.

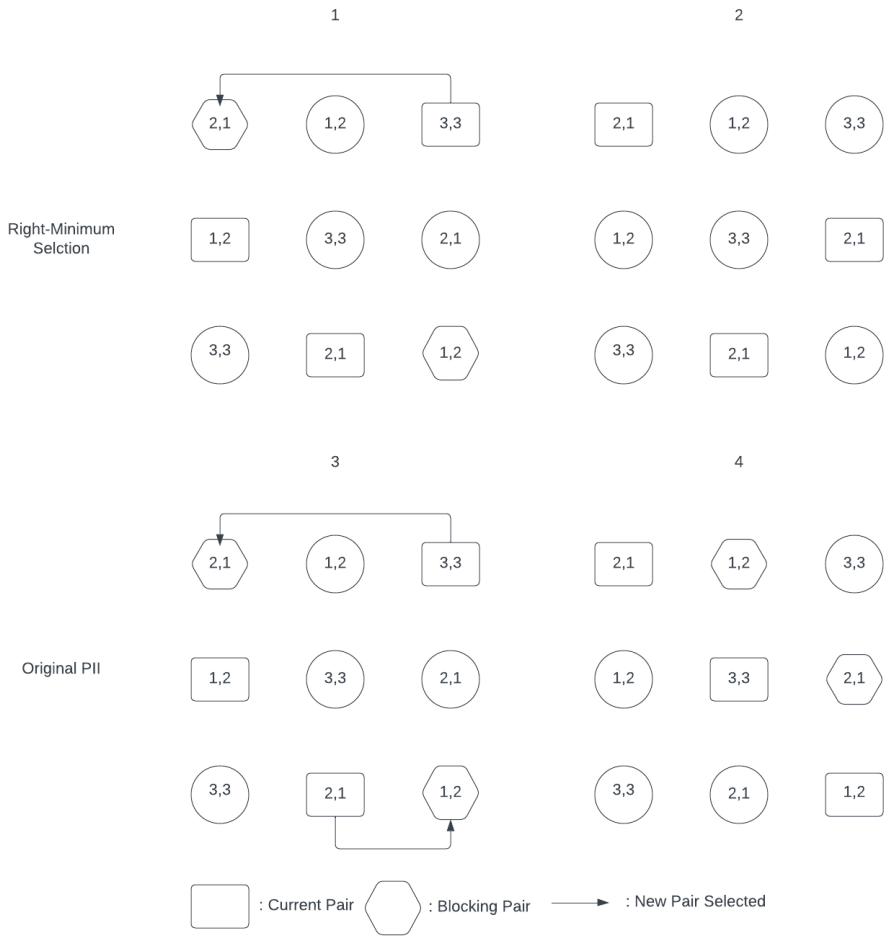

Fig. 1. Comparison of a single iteration of Right-Minimum Selection (from 1 to 2) and a single iteration of the original PII algorithm (from 3 to 4). Each sub-figure (1,2,3,4) contains the entries of the preference matrix, with shapes indicating the matching at a given iteration. The restriction imposed by Right-Minimum selection allows immediate convergence, while PII continues to allow multiple blocking pairs.

Complexity
The following provides a justification that the per iteration time complexity remains $O(log(n))$ with Dynamic Selection:

Each PE in row i maintains a pointer to $PE_{i,l}$ corresponding to $L(p_{i,l}) = \min\{L(p), p \in N_i\}$. For convenience, term this as the *minimum pointer* and $PE_{i,l}$ as the *left-minimum processor*. $PE_{i,l}$ points to itself. To avoid instances in which the algorithm consecutively selects that pair with the minimum left value, and to facilitate discovery of a greater number of NM1 pairs, the algorithm will impose a wait time $\mathcal{W} \in Z^+$. Hence, $PE_{i,l}$ also logs the last iteration at which it was compared c, and may enter the comparison process again when $k - c > \mathcal{W}$ where k is the current iteration. If this is the case, the *wait condition* is considered satisfied.

Intuitively, the wait condition ensures that the algorithm will only use dynamic selection when there are enough previously selected pairs to make the step effective, and when the standard iterations are taking too long to converge, suggesting the algorithm is likely in a cycle.

It is straightforward to see that the additional variables may be stored with $O(1)$ space complexity. The per iteration time complexity remains $O(log(n))$, as justified below:

PROOF. Assume at iteration k there is an unstable pair $p_{i,j}$ in row i. Dynamic Selection may be achieved with the following steps:

(1) If $N_i = \emptyset$ and $p_{i,j}$ is selected as an NM1 pair, proceed to 4. Otherwise, continue.
(2) If the wait condition is satisfied, left-minimum processor $PE_{i,l}$ enters the comparison procedure outlined in section 5.2. Otherwise, NM1 selection proceeds as normal. Row-wise find-minimum operations may still be achieved in $O(log(n))$ time.
(3) If $p_{i,j}$ is selected as an NM1 pair, $PE_{i,j}$ compares its left value to $PE_{i,l}$. This is achieved in $O(1)$ time. If $L(p_{i,j}) \geq L(p_{i,l})$,

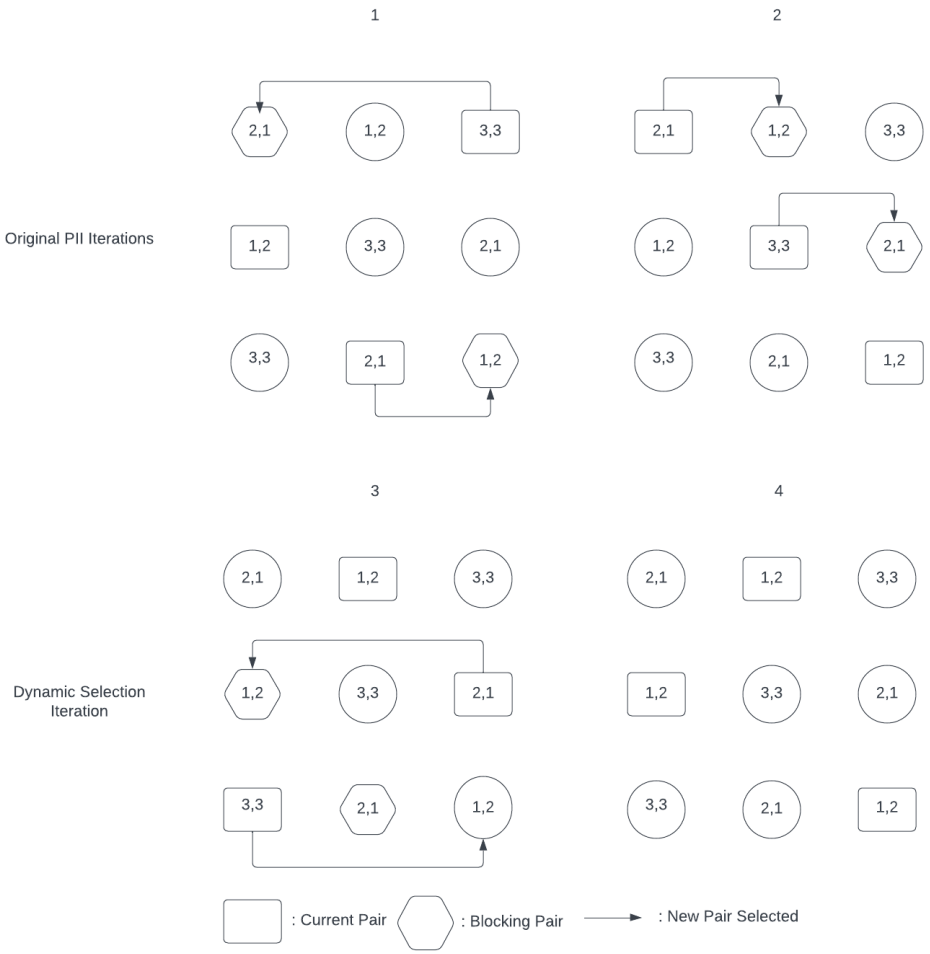

Fig. 2. Example of Dynamic Selection. Each sub-figure (1,2,3,4) contains the entries of the preference matrix, with shapes indicating the matching at a given iteration. Iterations of the original PII algorithm are done for the first 2 iterations (from 1-2 and 2-3). Dynamic Selection is done for the 3rd iteration (from 3-4) leading to convergence. Note that another iteration of the original PII algorithm after the matching at 3 would return to the initial matching (1) causing a cycle.

no change occurs. Otherwise, $L(p_{i,j}) < L(p_{i,l})$ and proceed to 4.

(4) $PE_{i,j}$ sets its minimum pointer to itself, and broadcasts its position along row i. Each processor then proceeds to set its minimum pointer to $PE_{i,j}$. This is achieved in $O(log(n))$ time.

Each step occurs in at most $O(log(n))$ time. Hence, per iteration complexity remains $O(log(n))$.

□

It has been empirically determined that incrementing $\mathcal{W}$ each time the left-minimum processor is selected as an NM1 pair yields the best results. The algorithm will then set $\mathcal{W}$ to 2 when a new a left-minimum processor is chosen.

3.3 Preprocessing

This paper also proposes new quick initialization method. Each man m_i in M sequentially proposes to his top choice woman remaining. When a woman receives a proposal, they are matched and the woman is removed from the preference lists of all other men. After all men have proposed once, all men will have been matched. Each woman receives exactly one proposal, so proposals can be done in $O(1)$ runtime. In parallel with n^2 processors, removing a woman from the preference lists of other men can also be done in $O(1)$ runtime, allowing quick initialization to run with $O(n)$ complexity in parallel.

4. THE PII-RMD ALGORITHM

Define the PII-RMD algorithm as the PII algorithm augmented with Right-Minimum Selection and Dynamic Selection. It has been empirically determined that beginning Right-Minimum Selection after

Probability of Convergence across Varying N

N	Original PII	Right-Minimum	Dynamic	PII-RMD (Random)	PII-RMD (Quick)	PII-RMD (Smart)
10.0	0.99028	0.99962	0.99999	1.0	1.0	1.0
20.0	0.95848	0.99841	0.9999	1.0	1.0	1.0
30.0	0.92675	0.99817	0.9999	1.0	1.0	1.0
40.0	0.90177	0.99853	0.99974	1.0	1.0	1.0
50.0	0.88343	0.99891	0.99976	1.0	1.0	1.0
60.0	0.87356	0.9989	0.9996	1.0	1.0	1.0
70.0	0.86709	0.99922	0.9997	1.0	1.0	1.0
80.0	0.86566	0.99935	0.99939	1.0	1.0	1.0
90.0	0.86306	0.99938	0.99944	1.0	1.0	1.0
100.0	0.86464	0.99949	0.99944	1.0	1.0	1.0

Fig. 3. Probability of successfully finding a stable matching within $5n$ iterations with $100,000$ trials for various n ranging from $10-100$. All methods were tested using random initialization, except PII-RMD (Quick) and PII-RMD (Smart) which used Quick and Smart Initialization respectively.

Probability of Convergence within Varying Numbers of Iterations

Iterations	Original PII	Right-Minimum	Dynamic	PII-RMD (Random)	PII-RMD (Quick)	PII-RMD (Smart)
0.5n	0.8246	0.82445	0.82277	0.8229	0.87148	0.8229
1n	0.86406	0.86483	0.86224	0.86397	0.95655	0.86397
1.5n	0.86462	0.97983	0.99711	0.98005	0.99235	0.98005
2n	0.86464	0.99562	0.99899	0.99598	0.99827	0.99598
2.5n	0.86464	0.99886	0.99931	0.99922	0.99973	0.99922
3n	0.86464	0.99913	0.99939	0.99958	0.99984	0.99958
3.5n	0.86464	0.99915	0.99943	0.99999	1.0	0.99999
4n	0.86464	0.99915	0.99944	1.0	1.0	1.0
4.5n	0.86464	0.99915	0.99944	1.0	1.0	1.0
5n	0.86464	0.99915	0.99944	1.0	1.0	1.0
Failed	0.13536	0.00085	0.00056	0.0	0.0	0.0

Fig. 4. Probability of successfully finding a stable matching with $n = 100$ with $100,000$ trials within various numbers of iterations ranging from $0.5n - 5n$. All methods were tested using random initialization, except PII-RMD (Quick) and PII-RMD (Smart) which used Quick and Smart Initialization respectively.

n iterations and initially setting the wait time $\mathcal{W}$ of Dynamic Selection to n produced the fastest convergence.

5. RESULTS

The complete PII-RMD algorithm, along with Right-Minimum Selection, Dynamic Selection, and the original PII algorithm, have been implemented for comparison. Each algorithm has been tested for convergence and speed using Randomized Initialization, Quick Initialization, and Smart Initialization.

In each trial, randomized preference lists are created, done by generating a random permutation of n preference lists of length n, and run each of the different algorithms with each initialization method on the created preference matrix. In figure 3.3, the success rate of each algorithm in finding a stable matching within 5n iterations was computed, where n ranges from 10 to 100, running 1 million trials for each algorithm variant. In figure 4, the number of iterations each algorithm required to find a stable matching was computed using n = 100, varying the number of iterations from 0.5n to 5n and running 100,000 total trials for each algorithm variant.

Right-Minimum Selection directly replaces iterations of the original PII algorithm. From figure 3.3 (or figure 7 for visualization), using Right-Minimum Selection instead of the original PII selection improves convergence at larger n from 86% to over 99.9%. Cases where Right-Minimum Selection alone fails to find a stable matching still terminate as shown in section 3.1, and the final matching has always been observed to contain very few unstable pairs.

Dynamic Selection and Cycle Detection both force pairs from a single stable matching to be chosen when the algorithm is cycling between pairs from multiple separate matchings. Comparing Dynamic Selection with the Cycle Detection data in [17], both methods perform similarly in improving algorithm convergence rate when allowed to run the full $5n$ iterations, as shown in figure 6. It should be noted that Cycle Detection does very slightly outperform Dynamic Selection at higher iterations (converging in approximately 3 more trials per 10,000), but due to incompatibility with Right-Minimum selection as noted in section 3.2, Dynamic Selection is preferred for the PII-RMD algorithm. Additionally, the cycle detection method does not begin until iteration 3n, causing a very significant convergence runtime improvement when using Dynamic Selection over Cycle Detection, indicating the large difference in convergence rates at lower iterations in figure 6. For the same reason, the PII-RMD algorithm showed a significant improvement in convergence speed compared to the

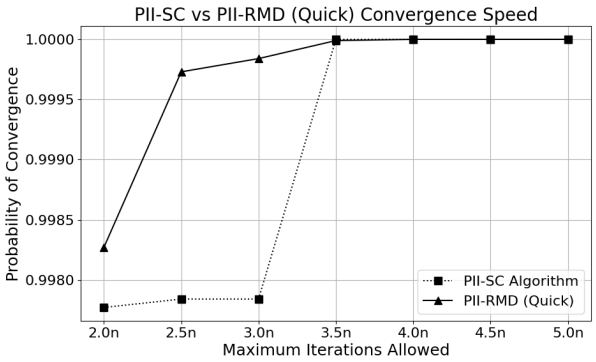

Fig. 5. Comparison of PII-SC vs PII-RMD (with Quick Initialization) Convergence Speed. Note that this plot starts from iteration 2n instead of 0.5n for easier visualization due to the magnitude of convergence rates.

PII-SC algorithm as shown in figure 5.

Overall, the PII-RMD algorithm converged in all cases for random, quick, and Smart initialization, with fastest convergence when using quick initialization (see figure 4) removing the need for an expensive $O(n\ log(n))$ runtime for preprocessing. All previous iterations of the PII algorithm, most notably the PII-SC algorithm, still failed to converge in select edge cases, as shown in figure 6, which worsened with larger n.

To assess the scalability of the PII-RMD algorithm with higher values of n, the PII-RMD algorithm has also been tested for n=100, 110, ..., 200. For each initialization method 10,000 tests were run for each value of n, resulting in 330,000 total trials which fully converged, suggesting the PII-RMD algorithm does not see the same steep decline with increasing n observed in the PII-SC algorithm even for $n \leq 100$ in [17]. Furthermore, Right-Minimum Selection was individually tested with limited trials (1000 trials for each value of n) for n ranging from 100 to 1000, and also saw no performance decrease at higher values of n compared to those in figure 3.3. These results support the scalability of the PII-RMD algorithm, achieving the original PII algorithm's purpose of an efficient in-practice stable matching algorithm on a reasonable number of processors.

Overall, the PII-RMD algorithm is empirically fully convergent within $5n$ iterations in all $3,630,000$ trials across all tested values of n, including testing at higher values of n compared to previous PII algorithm iterations. The runtime required for preprocessing and the number of iterations required for convergence were also reduced compared to previous PII algorithm iterations.

6. CONCLUDING REMARKS AND FUTURE WORK

The results suggest the PII-RMD algorithm is a significant improvement over previous PII algorithm variations in both runtime and convergence. Empirically, the PII-RMD algorithm exhibits complete convergence within 5n iterations, and thus has always been observed to converge in $O(n\ log(n))$ runtime.
The ultimate goal of research in the PII algorithm is to determine a fully convergent stable matching algorithm on n^2 processors with a theoretical worst-case runtime of $O(n\ log(n))$, to provide

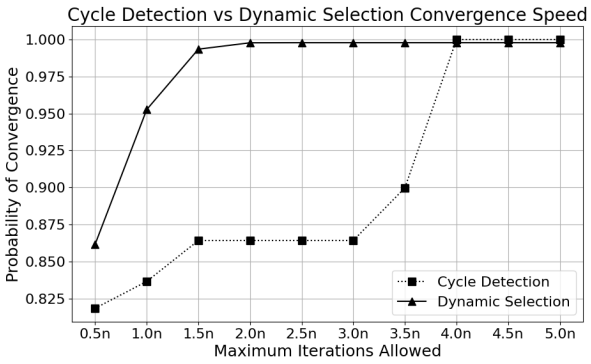

Fig. 6. Comparison of Cycle Detection vs Dynamic Selection Convergence Speed.

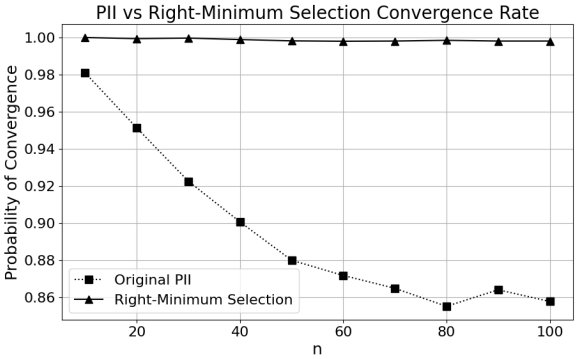

Fig. 7. Comparison of Original PII vs Right-Minimum Selection Convergence Rate.

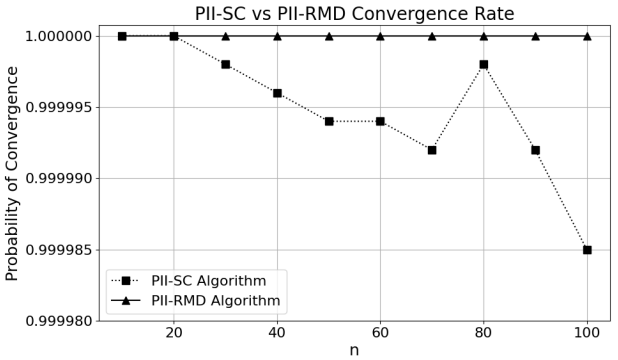

Fig. 8. Comparison of PII-SC vs PII-RMD Convergence Rate.

an efficient algorithm with a low requirement on the number of processes available for large scale applications. It remains unclear whether PII-RMD algorithm fully converges theoretically.

Building toward proving the convergence of the PII-RMD algorithm or another algorithm satisfying the same runtime and parallel architecture constraints is a key objective of this work. In doing so, this research also aims to better understand the properties of instability for the stable matching problem, as very limited research has been done in this area [2].

7. ACKNOWLEDGEMENTS

This work is proudly supported by NSF grant CNS-2149591, and was conducted in part as part of the REU EXERCISE program at Salisbury University.

The code for this work can be accessed on GitHub at `https://github.com/salberi1/Parallelizing_Stable_Matching`

8. REFERENCES

[1] S.T. Cao, L.V. Thanh, and H.H. Viet. Finding Maximum Weakly Stable Matchings for Hospitals/Residents with Ties Problem via Heuristic Search. In *AI 2023: Advances in Artificial Intelligence, Lecture Notes in Computer Science*. Springer, Singapore, 2024. doi:10.1007/978-981-99-8388-9_36.

[2] Kimmo Eriksson and Olle Häggström. Instability of matchings in decentralized markets with various preference structures. *International Journal of Game Theory*, 36(3–4):409–420, 2008. doi:10.1007/s00182-007-0110-0.

[3] T. Feder, N. Megiddo, and S.A. Plotkin. A sublinear parallel algorithm for stable matching. *Theoretical Computer Science*, 233(1–2):297–308, 2000.

[4] Enrico Maria Fenoaltea, Izat B. Baybusinov, Jianyang Zhao, Lei Zhou, and Yi-Cheng Zhang. The stable marriage problem: an interdisciplinary review from the physicist's perspective. *Physics Reports*, 917:1–79, 2021.

[5] Patrik Floréen, Petteri Kaski, Valentin Polishchuk, and Jukka Suomela. Almost stable matchings by truncating the Gale–Shapley algorithm. *Algorithmica*, 58:102–118, 2010.

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

[7] C.C. Huang, T. Kavitha, J. Mestre, and others. On the dynamics of distributed computing and stable matchings. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 287–296. ACM, 2007.

[8] R.W. Irving, P. Leather, and D. Gusfield. An efficient algorithm for the "stable roommates" problem. *Journal of the ACM*, 34(3):532–543, 1987.

[9] D.E. Knuth. *Mariages Stables*. Les Presses de L'Université de Montréal, Montréal, 1976.

[10] Alec Kyritsis, Scott Wynn, Stephora Alberi, and Enyue Lu. Dynamic and Right-Minimum Selection for the Parallel Iterative Improvement Stable Matching Algorithm. In *Proceedings of Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing (ACM MOBIHOC)*, series REUNS workshop, pages 568–570, October 2023. ACM. doi:10.1145/3565287.3617979. Extended Abstract and Poster.

[11] Enyue Lu, Mei Yang, Yi Zhang, and SQ Zheng. Design and implementation of an acyclic stable matching scheduler. In *GLOBECOM'03. IEEE Global Telecommunications Conference (IEEE Cat. No. 03CH37489)*, volume 7, pages 3938–3942. IEEE, 2003.

[12] Enyue Lu and SQ Zheng. A parallel iterative improvement stable matching algorithm. In *High Performance Computing-HiPC 2003: 10th International Conference, Hyderabad, India, December 17-20, 2003. Proceedings 10*, pages 55–65. Springer, 2003.

[13] A.E. Roth and J.H. Vande Vate. Random paths to stability in two-sided matching. *Econometrica*, 58(6):1475–1480, 1990.

[14] Alvin E. Roth and Elliott Peranson. The NRMP matching algorithm revisited: theory versus practice. *Academic Medicine*, 70(6):477–484, 1995. URL: `https://journals.lww.com/academicmedicine/abstract/1995/06000/The_NRMP_matching_algorithm_revisited__theory.8.aspx`.

[15] Youcef Saad and Martin H. Schultz. Data communication in parallel architectures. *Parallel Computing*, 11(2):131–150, 1989.

[16] S. Subramanian. A parallel approach to stable marriage problem. *Information Processing Letters*, 41(5):227–233, 1992.

[17] Colin White. An Improved Parallel Iterative Algorithm for Stable Matching. *SuperComputing 2013*, 2013. Extended Abstract and Poster.

[18] Yuxiang Zhang, Lin Cui, and Yuan Zhang. A stable matching based elephant flow scheduling algorithm in data center networks. *Computer Networks*, 120:186–197, 2017.