Minimizing Bias in Residency Matching: A Study in Non-Standard Random Walks

Rebecca Baumher, Jeremy Bierema, Scott Buchanan, Meryem Essaidi

April 26, 2016

Advisor: Sampath Kannan
Professors: Ani Nenkova and Jonathan M. Smith

Dept. of Computer Science
University of Pennsylvania

Abstract

For problems with multiple solutions, it is often desirable to select a solution uniformly at random. This gives all possible solutions a chance of being selected, whereas deterministic algorithms might always bias the outcome towards one solution or one family of solutions. In particular, we focus on the problem of matching residents to hospitals, where deterministic algorithms are inherently biased to favor one side over the other. An unbiased solution is to pick a matching at random. We can represent all possible matchings as nodes in a graph and perform a random walk on this graph. We then output the matching represented by the node that the random walk terminates on, after a sufficient number of steps. However, a standard random walk can take very long to reach the stationary distribution. Thus, our group investigated various non-standard random walks in order to find one that would be more rapidly mixing, which would give a time improvement over standard random walks. As a result, we have been able to reduce the maximum possible deviation from the standard distribution on certain input graphs. Our results show a consistent improvement by a factor of 3 over that achieved using standard random walks.

Introduction

In order for a graduate to complete their medical training and obtain an unrestricted license to practice medicine, the graduate must first complete a residency. A residency is a component of medical training during which the graduate practices medicine under the supervision of an attending physician, which allows the resident to improve their knowledge, hone their skills, and develop key experience. Hence, each graduate needs to be matched to a hospital. In the 1940s, there was no overarching system to regulate the matching process, and so a decentralized, competitive system emerged that led to great dissatisfaction[22].

The problem was that the system caused a conflict of interest between the hospitals and residents. Namely, this was because the hospitals benefited from trying to fill their positions as early as possible by forcing the residents to make premature decisions, whereas the residents benefited from delaying their decisions as long as possible in order to accumulate all potential offers and make a well-informed choice[23]. This phenomenon led to a series of actions from both sides in attempts to game the system in their favor: Hospitals began offering resident positions up to two years in advance. In response, medical schools prohibited the release of transcripts and the writing of letters of recommendation from occurring until a medical student's final year of school. And in response to this, hospitals changed the amount of time students had to accept an offer after it was received, lowering it from ten days to twelve hours[9].

Due to the great dissatisfaction of the aforementioned system and the ill-natured tactics it gave rise to, a new system emerged. In 1952, an organization known as the National Resident Matching Program (NRMP) took on the responsibility of matching residents to hospitals. Unlike the previous system, the NRMP used an algorithm to smartly assign hospitals to residents[22]. Specifically, the NRMP collected every resident's list of hospital preferences and every hospital's list of resident preferences as input, then ran an algorithm to find a stable matching[1] between the two groups. However, the algorithm used (known as the Stable Marriage Algorithm) was inherently biased to favor one side over the other. Indeed, the execution of the algorithm is asymmetric in that it can be run in one of two ways: to generate the resident-optimal solution or to generate the hospital-optimal solution. The resident-optimal solution is the stable matching that, among all possible stable matchings, gives each resident their best possible choice of hospital. Simultaneously, it is also the stable matching that, among all possible stable matchings, gives each hospital their worst possible choice of residents. The hospital-optimal solution is similar, but with the roles of the residents and hospitals reversed[23]. Thus, no matter how the algorithm is run, it favors one side at the cost of the other. Our group sought to improve this situation by developing an improved algorithm: an algorithm that could select a matching in an unbiased fashion in a reasonable amount of time.

Approach

The goal of our algorithm is to take in a graph as input, and then output a matching on this graph uniformly at random.

Definition 1 *A* matching *is defined as a subset of a graph's edges such that no node in the graph is incident on more than one edge in the subset.*

A matching partitions the set of nodes of the graph into pairs, and two nodes belonging to the same pair represent a resident being matched to a hospital. Note that in general there is not just a single matching in a graph; many matchings are possible. Our goal is to output just one of these matchings uniformly at random. In other words, we want to give all possible matchings the same probability of being selected. This is in contrast to the NRMP's algorithm, which is only capable of ouputting two of these matchings: the resident-optimal matching and the hospital-optimal matching. The NRMP's algorithm is also deterministic, whereas our algorithm makes use of randomness in order to pick one of the possible matchings uniformly at

[1]We say a matching between residents and hospitals is stable unless there is a resident and a hospital who both would prefer to be matched with each other over those with whom they were matched.

random, thus yielding a fairer solution.

The naïve approach to selecting a matching uniformly at random would simply be to construct all possible matchings and then pick one of them uniformly at random. However, this is infeasible in practice. That is because for a graph with n nodes, there are on the order of 2^n possible matchings. The number of residents is on the order of 20,000[17], which means that there would be on the order of 10^{6020} possible matchings. To compute and store all of these would be physically impossible, especially considering that there are only 10^{80} atoms in the known universe! Thus, we used a technique that is much more space efficient to select a matching uniformly at random: we construct a meta-graph in which each node represents one possible matching, perform a random walk on this meta-graph for a sufficiently high number of steps, and then output the matching on which the walk stops. Note that we need not store this entire meta-graph at once in order to perform this random walk. Instead, we merely need to store our current node in the walk and the nodes in the local neighborhood of this node.

To construct this meta-graph, we first use a deterministic algorithm to identify one perfect matching.

Definition 2 *A* perfect matching *is a matching such that every node in the graph is incident on exactly one edge in the matching.*

To find a perfect matching, we use the Blossom Algorithm, an algorithm published by Jack Edmonds in 1965[7]. Given an input graph, this algorithm finds a maximal-sized matching in polynomial time. Thus, we use the Blossom Algorithm to isolate one node of the meta-graph, which is also the start of our random walk through the graph.

The nodes of the meta-graph represent both perfect matchings and near-perfect matchings.

Definition 3 *We say a matching is a* near-perfect matching *if it has exactly one fewer edge than a perfect matching would.*

Hence, in a near-perfect matching, exactly two nodes will not be incident to any edge in that matching, whereas in a perfect matching, there are no such nodes, since all nodes are incident on exactly one edge in the matching. We use simple rules to define the edges of the meta-graph. For perfect matchings, we draw edges to all near-perfect matchings than can be constructed by the removal of exactly one edge from the perfect matching. For near-perfect matchings, we draw edges to all matchings that can be constructed by performing the following procedure:

1. Pick one of the two unmatched nodes u in the graph.
2. Pick one of u's neighbors v, which may or may not be part of the matching.
3. If v is part of the matching, remove the edge incident to v from the matching.
4. Add an edge to the matching between u and v.

In this fashion, we draw edges from the near-perfect matching to other matchings, which may either be perfect or near-perfect.

Finally, we must explain precisely how we walk through this meta-graph. Since we are performing a random walk, at every step we randomly select one of the current node's neighbors, move to that neighbor, and then repeat. This process represents a Markov chain: a random process which moves from one state to another on a state space[13]. In this case, the nodes of the meta-graph represent the different states.

But the exact mechanism by which we assign probabilities to each neighbor of the current node for the transitions is a non-trivial matter with much importance, since it directly affects the expected time to approach the stationary distribution.

Definition 4 *A* stationary distribution *(if it exists) is a probability distribution $\boldsymbol{\pi}$ such that if the probabilities of states are chosen according to $\boldsymbol{\pi}$, then after one step the probabilities will still be distributed according to $\boldsymbol{\pi}$.*

Assume the Markov process is aperiodic[2]. If the stationary distribution exists and is unique, then we can think of it as the probability of being at each node after a random walk of infinite length. More formally, under the same conditions, if we walk on the graph for a sufficiently large number of steps, our expected probability distribution will approach the stationary distribution; at this point, if the stationary distribution is uniform, then we will have an equally likely chance of being at any particular perfect matching.

In a standard random walk, we simply assign each neighbor an equal probability of being selected, and so the next node is chosen uniformly at random from the set of neighbors of the current node. However, a standard random walk approaches the stationary distribution in polynomial time. And since the size of the meta-graph is exponential in the size of the original graph, this means a standard random walk would take exponential time to approach the stationary distribution. Consequently, we prefer a non-standard random walk: a random walk in which the each neighbor does not have an equal probability of being selected. There are infinitely many ways to implement a non-standard random walk, and so our mission was to find an implementation that converged to the stationary distribution faster than the standard random walk.

Methods

We implemented several different algorithms for choosing the probabilities of each neighbor of a node. The input into our algorithm is the local neighborhood of the current node of the walk, and from this information we output the probabilities that the random walk should apply to choose a neighbor of the current node. The fact that we limit ourselves to just the local neighborhood of the current node is essential to the feasibility of our algorithm in practice. That is because in practice, the meta-graph would have an astronomically large number of nodes, and so no global information could be discerned about the meta-graph. Only information about a local section of the meta-graph could be discerned within a reasonable amount of time.

Specifically, we limit ourselves to the nodes of the meta-graph that are within a distance k of the current node, where k is a parameter. At a high level, our approach calculates probabilities in such a way as to favor nodes that are more likely to lead to distant parts of the meta-graph. In this fashion, we effectively avoid "getting stuck" in certain parts of the meta-graph, since our probabilities lead the walk through "bottlenecks" that would otherwise limit our walk from fully exploring the graph.

We now present a low level description of our algorithms. We assume we are currently at a node s in the meta-graph, and we have an input parameter k. We only consider nodes of the meta-graph that are within a distance k of s. These nodes can be partitioned further by their distance to s, which will range from 1 to k. Let the set of nodes that are exactly j away from s be denoted by R_j. Note that R_1 is simply the neighbors of s. The general form of our algorithms goes as follows:

1. Give 1 credit to each neighbor of s. Allow each credit to remember which neighbor of s it originally assigned to (i.e., its origin).
2. The credits are currently assigned to nodes of R_j. Now consider the nodes of R_{j+1}. Every node u in R_j splits each of its credits into n equal pieces, and gives them to neighbors of u in R_{j+1}. Note that credits of different origins do not lose their identity.
3. Repeat step 2 until the credits are assigned to the nodes of R_k.
4. Normalize the total credits of each node of R_k. That is, for each node u in R_k, scale its credits such that its total amount of credit (including credits of different origins) is 1.
5. Aggregate the credits of the nodes of R_k, combining credits of the same origin. This yields a vector whose components each represent one neighbor of s.
6. Normalize this vector so that it is a unit vector. It now represents the probability distribution which we use to determine the next node of the walk.

[2] A Markov chain is aperiodic if there is no integer p greater than 1 such that repeated visits to any node are always separated by a multiple of p.

Note that we actually created several different algorithms by altering these steps in various ways. Firstly, we have 3 different ways of how node u should split its credits within step 2:

- Cloning: Do not split the credits into smaller pieces. Give each node in R_{j+1} a copy of the entire credit.
- Degree Splitting: Split the credits into d pieces, where d is the total degree of u, which includes cross edges and back edges. Give each node in R_{j+1} one piece.
- Forward Splitting: Split the credits into n pieces, where n is the number of edges between u and the nodes of R_{j+1}. Give each node in R_{j+1} one piece.

Hence, the originally mentioned step 2 was the Forward Splitting variation. Visual examples of these variations can be found in Appendix A. Secondly, we have 4 different ways of aggregating credits within step 5:

- Exactly k Away: Aggregate the credits of the nodes of R_k.
- Up to k Away: Aggregate the credits of the nodes of R_1 through R_k.
- Up to k Away Proportional: Aggregate the credits of the nodes of R_1 through R_k, scaling the credits in R_j by a factor of j.
- Mixed: We take a weighted average of the transition probabilities given by the Exactly k Away variation and the probabilities given by a standard random walk. While giving equal consideration to the two algorithms is not strictly necessary, we limited ourselves to 50–50 weights.

Hence, the originally mentioned step 5 was the Exactly k Away variation. Finally, in addition to our non-standard random walk algorithms, we implemented a standard random walk algorithm to use as a control. Consequently, we were able to measure whether our own algorithms were an improvement over the standard random walk.

Measurements and Results

Our project focused on empirical tests of our algorithms, in the hopes of showing trends that would inspire and guide theoretical research in our techniques. The format of our tests was to generate a small graph from some distribution, and then from that graph construct the meta-graph of perfect and near-perfect matchings. We then treated this meta-graph as a Markov process, using each of our different algorithms in turn to compute the transition probabilities. For each algorithm, we calculated statistics about the convergence of the Markov process, and used these statistics to analyze the effectiveness of our algorithms.

Since the number of matchings on a graph grows exponentially in the number of vertices, it was impractical to generate the entire meta-graph and run comprehensive tests on it for even moderately-sized graphs. Thus, we confined ourselves to graphs with at most 28 vertices. We chose mostly to use 3-regular graphs—graphs where every node has exactly three neighbors—in order to further restrict the total number of matchings without completely trivializing the problem, although we also tried some small 4-regular graphs. To generate the graphs, we implemented a simple algorithm by Steger and Wormald[1] that returns regular graphs from an asymptotically-uniform distribution. As a sanity check, we confirmed that all of the graphs we generated were strongly connected.

We used each of our algorithms in turn to compute the transition probabilities from every (near-)perfect matching to each neighboring (near-)perfect matching. Each of these probability vectors became a row in the transition matrix T_A for the corresponding algorithm A. The probability of being in each of the different states after s steps can then be computed as the product of the initial state vector by T_A^s. Thus, entry (i, j) of T_A^s represents the probability distribution that the Markov chain will end in state j after exactly s steps if i is the index of the initial state. We can compute T_A^s for values of s that are powers of 2 by repeatedly squaring T_A.

Our main interest was in finding an algorithm for computing transition probabilities that caused the Markov process to converge more quickly to the stationary distribution. Thus, we wanted algorithms for which the initial state vector had little impact on what the probability distribution over states would be after a fixed number of steps in the Markov process. In other words, we wanted the rows of T_A^s to be approximately equal. To quantify this property, we computed a vector such that the jth entry in the vector was set to the difference between the maximum and minimum probability in the jth column of T_A^s. This difference represents the uncertainty in the probability that we will end up in the corresponding state, depending on the initial state vector. Since we only care about perfect matchings, we do not include the uncertainties for near-perfect matchings. To condense this uncertainty vector to a single value, one option is to take the L^∞-norm (the max of the entries), which gives us an upper bound on the uncertainty of each individual probability in the distribution. Another option is to take the L^1-norm (the sum of the entries), which can be thought of as bounding the total probability that the ending state will not be drawn according to the stationary distribution. Since both norms have potential use cases, we calculated both, although the results from the two were similar.

We have included a plot of some of the output for a certain run of our tests in Appendix B. Our algorithms were run on a 24-node 3-regular graph, and we used $k = 3$. The graph had 66 perfect matchings and 4189 near-perfect matchings. We only show the forward-splitting variants of our algorithms to keep the plot simple, since the best and worst of our algorithms for this graph were both forward-splitting variants. Note that the horizontal axis shows the log of the number of steps we must take in the Markov chain before achieving the uncertainty bound plotted on the vertical axis.

The plot shows that on some graphs, if we want very small error, we can get a factor-8 improvement within the same number of steps in the Markov chain by using one of our non-standard random walks. Although we have shown one of the more optimistic outcomes, all of the tests we ran gave an error reduction by a factor of 3 or more for both the Exactly k Away and Mixed (50–50) algorithms using forward splitting once we had taken a large number of steps, assuming we had set k correctly. In particular, we set k to 3 for graphs with fewer than 24 nodes (where the meta-graph contains roughly 4000 or fewer matchings), and we set k to 4 for larger graphs, up to our limit of 28 nodes (with over 10,000 matchings in the meta-graph). Note that while our methods reduce the error for a fixed number of steps in the Markov chain, if we instead fix the desired error bound and compute the number of steps to take in the Markov chain, the improvement offered by our non-standard random walks becomes quite small, especially in light of the extra overhead needed to generate a local neighborhood in the graph and compute complicated transition probabilities.

Further cause for caution comes from putting our results back into the perspective of the original problem. We have been showing that some of our non-standard random walks converge more quickly to the stationary distribution. While the stationary distribution is uniform over all perfect matchings using standard random walks, it is no longer completely uniform using non-standard random walks. To quantify the disturbance, for each algorithm run on a graph, we calculated the maximum ratio between the probabilities of any two perfect matchings in the stationary distribution. Focusing on the two algorithms highlighted previously, we found some skews of 1.1 or higher for the exactly k-away algorithm and values at most 1.05 for the mixed (50–50) algorithm, both using forward splitting. These values rarely fell below 1.03. The largest two graphs we tested gave the highest and lowest values, so any relationship to the graph size is not clear. Nevertheless, if our goal is to generate perfect matchings uniformly at random, slight improvements in small errors will not help us unless we know something about the stationary distribution so that we can correct for these small deviations from uniformity.

Another statistic has also some bearing on the problem. If the probabilities of perfect matchings in the stationary distribution are low, then there is a greater chance that our random walk will end on a near-perfect matching, forcing us to begin another random walk in the hopes of an acceptable outcome. For the runs where we picked proper values of k, we found that the probability of ending on a perfect matching was higher for our two featured algorithms than for standard random walks, though not usually by a significant amount. Higher improvements were achieved for some of the other algorithms that did not converge as quickly. Improvements by a factor of 1.3 or more were not uncommon, and factors of up to 1.6 were reached occasionally. These improvements could be of interest in reducing the expected number of walks needed when only loose uniformity bounds are needed.

To conclude, although we found that we can indeed improve the mixing time slightly (at least on the small graphs we tested) by using non-standard random walks, some of the good properties of standard random walks are lost by doing so, especially uniformity. Nevertheless, we did find an improvement in the overall probability of selecting a perfect matching, and this is worth further study.

Discussion

We created an algorithm that takes a graph and selects a perfect matching for that graph approximately uniformly at random using less Markov chain steps than the standard random walk. Our algorithm can be applied in any situation in which it is desirable to form pairings between entities, where a deterministic algorithm may be inherently biased. For example, our algorithm can be used to match up roommates or match up competitors in a tournament. And of course, it can be used in our running example of the resident-hospital problem.

However, the details of how we create the initial graph on which we want to find the matching from everyone's list of preferences is a non-trivial matter. The edges in our model are unweighted, and so our algorithm for generating a matching does not take into consideration that certain edges may be preferred to others. So the question becomes which edges to add between nodes given every resident's and hospital's list of preferences. One approach may be to add an edge between everyone and their top n preferences for some small n, but a perfect matching is not guaranteed to exist in that case.

Instead, a better approach would be to run the resident-optimal algorithm and the hospital-optimal algorithm, and use those to determine the worst-case scenario for everyone's matchings. For example, a resident may be matched to their $4th$ preference in the resident-optimal solution, but be matched to their $26th$ preference in the hospital-optimal algorithm; since every stable marriage would match this resident to a hospital ranked $26th$ or above in their list of preferences, this is the resident's worst matching. Finally, we draw an edge between a resident and a hospital if the match is at least as good as the worst-case scenario for both parties. After doing this, we achieve a graph on which a stable matching is guaranteed to exist, since both the resident-optimal solution and hospital-optimal solution can be found on this graph. Hence, we simply run our own algorithm on this graph to generate a perfect matching uniformly at random. Although the resulting graph might not be a stable matching, we argue that it is more fair than the two aforementioned extreme solutions, since every participant will get at least as good of an assignment as if the algorithm were biased against them, and much of the time it will be better.

The algorithm used by the NRMP had been found to be hospital optimal, as it favored hospital preferences when multiple matchings were possible[25]. These findings were made public, and caused many to complain[12]. Nevertheless, the algorithm has gone through only minimal updates since it was first implemented[21], and so it is still in the hospitals' favor. Should the NRMP adopt our algorithm, the deterministic hospital-optimal nature of the end result would be no more. Instead, all reasonable matchings would have a chance of being selected, giving no party an unfair advantage. This would be especially favorable to the residents, who currently have the unfavorable side. However, it should be noted that our algorithm does not account for certain nuances that the NRMP algorithm does[20]. These include:

- Resident couples, who apply together and wish to remain in the same geographical location
- Residents who seek second-year positions
- Residency programs that exist only if all positions are filled
- Residency programs that exist only if an even number of positions are filled

Our algorithm is not able to handle these situations, though with more time perhaps we could incorporate them into our algorithm. Additionally, should we have more time, we could contact the NRMP and ask for real testing data of residents' and hospitals' preferences so that we could have a more accurate body of data to work with. Furthermore, we could compare the resulting matching from the NRMP's algorithm with the resulting matchings from our own algorithm. In this fashion, we could determine whether on average our algorithm gives residents and hospitals better results in terms of their preferences. Certainly since the algorithm is no longer hospital-optimal, the hospitals would be expected to receive worse preferences. However, the residents would be expected to receive better preferences. And so, if the benefit of the latter outweighs the detriment of the former, it may be better to use our algorithm, as it presents a fairer solution that does not serve to benefit one side at the cost of the other.

Ethics and Privacy Considerations

All of the data and graphs we used were generated by us. Hence, they do not represent real people with real preferences, and so there has been no need withhold our data or results on the basis of privacy. However, if we moved forward and tested our algorithm on real data obtained from the NRMP, then those data and results would have to be managed carefully due to privacy concerns, since we would not want to reveal any resident's or hospital's list of preferences publicly. Furthermore, if our algorithm were used to compute the matchings for NRMP, then the only piece of the solution that each resident or hospital would receive is the specific edge of the matching that pertains to them. In other words, to protect everyone's privacy, a resident would only learn which hospital they are assigned to, and each hospital would only learn which residents are assigned to it. And so, all pairings not pertinent to a resident or hospital would not be disclosed to them. Furthermore, for any resident and hospital matched to each other, we would not disclose the preference placement each had for the other. For example, a resident who ends up matched to their last preference may not want the hospital to know that it was their last choice. Hence, we would only disclose the entities within the matching, and not their preferences for each other.

References

[1] A. Steger and N. Wormald. Generating random regular graphs quickly. (English. English summary) Random graphs and combinatorial structures (Oberwolfach, 1997), *Combin. Probab. Comput. 8* (1999), no. 4, 377–396.

[2] Alvin E. Roth (1996-03-01). "The NRMP As a Labor Market: Understanding the Current Study of the Match".

[3] Archived November 25, 2010, at the Wayback Machine.

[4] Colenbrander A. Match algorithms revisited. Acad Med 1996; 71:414-415.

[5] Colenbrander A. Ophthalmology match heeds students' concerns. AUPO flyer, February 1996.

[6] Description of market based on Roth, A.E. (1984). "The evolution of the labor market for medical interns and residents: a case study in game theory". *Journal of Political Economy* 92: 991–1016. doi:10.1086/261272.

[7] J. Edmonds, "Paths, trees, and flowers," *Canadian Journal of Mathematics* 17 (1965) 449–467.

[8] Evaluation of changes to be considered in the NRMP algorithm, by Alvin E. Roth. October 24, 1995.

[9] Gusfield, Dan; Robert W. Irving (1989). "1.1.1". *The Stable Marriage Problem: Structure and Algorithms.* The MIT Press. pp. 3–4. ISBN 0-262-07118-5.

[10] Gusfield "Stable Marriage" 64 references Roth, A.E. (1984). "The evolution of the labor market for medical interns and residents: a case study in game theory". *Journal of Political Economy* 92: 991–1016.doi:10.1086/261272. as proving that the pre-1995 algorithm is essentially the hospital-optimal algorithm described in Gusfield 39. Gusfield 41 demonstrates that the hospital-optimal algorithm is also applicant-pessimal.

[11] Klaus B, Klijn F, Massó J. Some things couples always wanted to know about stable matchings (but were afraid to ask). Review of Economic Design 2007; 11:175-184.

[12] Medical seniors hit interne plan. New York Times 1951; 22 Oct:25 (col. 1).

[13] Norris, James R. (1998). Markov chains. Cambridge University Press. Retrieved 2016-03-04.

[14] NRMP Directory. Evanston, Illinois: National Intern and Resident Matching Program; 1979.

[15] NRMP homepage Retrieved on June 23, 2010.

[16] "NRMP TO IMPLEMENT MATCH WEEK CHANGES". Retrieved 2011-04-25.

[17] "Results and Data - 2015 Main Residency Match" (PDF). National Resident Matching Program. 2015.

[18] "Results of the 2010 NRMP Program Director Survey" (PDF). Retrieved 2011-04-25.

[19] Robinson, S. (August 14, 2004). "Antitrust Lawsuit Over Medical Residency System Is Dismissed". New York Times.

[20] Robinson, Sara (April 2003). "Are Medical Students Meeting Their (Best Possible) Match?" (PDF). SIAM News (3): 36. Retrieved 14 October 2010.

[21] Roth AE. The evolution of the labor market for medical interns and residents: a case study in game theory. Journal of Political Economy 1984; 92:991-1016.

[22] Roth, Alvin E. (December 8, 2012). "The Theory and Practice of Market Design" (PDF).Nobelprize.org. Nobel Media AB.

[23] Roth, Alvin; Elliott Peranson (September 1999). "The Redesign of the Matching Market for American Physicians: Some Engineering Aspects of Economic Design". *The American Economic Review* 89 (4): 748–780. doi:10.1257/aer.89.4.748. Retrieved 14 October 2010.

[24] Roth "Redesign" 748, 749, 752, 760. Wilkey, Robert N. (April 2011). "The Non-Negotiable Employment Contract: Diagnosing the Employment Rights of Medical Residents". Creighton Law Review 44: 705. Retrieved 29 August 2012.

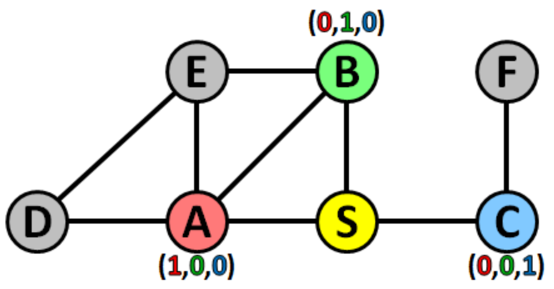
[25] Williams, K. J. (1995). "A reexamination of the NRMP matching algorithm. National Resident Matching Program". *Academic Medicine* 70 (6): 470–476; discussion 476–4. doi:10.1097/00001888-199506000-00007. PMID 7786366.

[26] Williams K. J. Examining the NRMP algorithm. Acad Med 1996; 71:310-312.

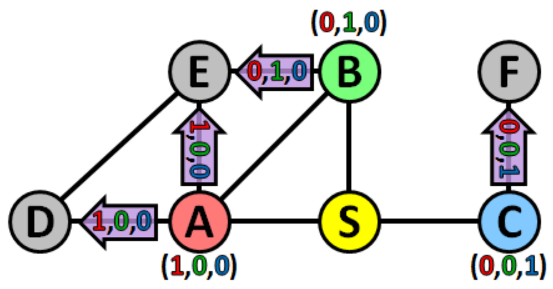
[27] Williams K. J., Werth VP, Wolff JA. An analysis of the resident match. N Engl J Med. 1981;304:1165-1166; correspondence in N Engl J Med. 1981;305:526.

A Algorithms

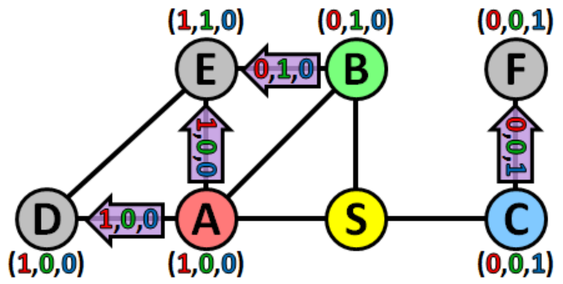
The Cloning variation for $k = 2$:



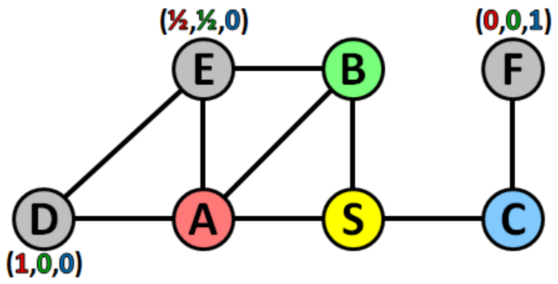
Step 1: The source node S gives each of its neighbors $\{A, B, C\}$ one credit. Each credit remembers its origin node, denoted here by the color of the node.



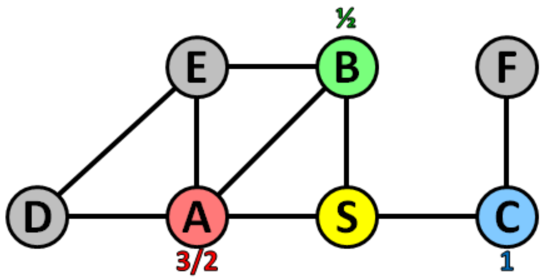
Step 2 (1/2): The nodes with the credits propagate their credits to the nodes that are 1 distance greater away from S. The credit values are copied rather than split.



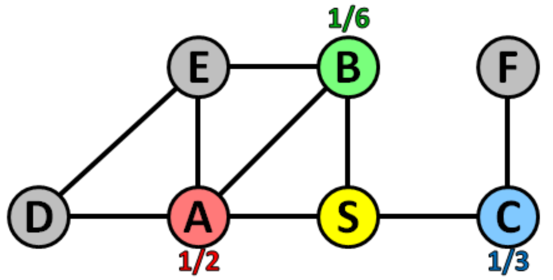
Step 2 (2/2): The receiving nodes aggregate the credits they receive, combining credits of same origin but keeping credits of different origin separate.



Step 4: The k-away nodes each normalize their credits so that the total is 1.

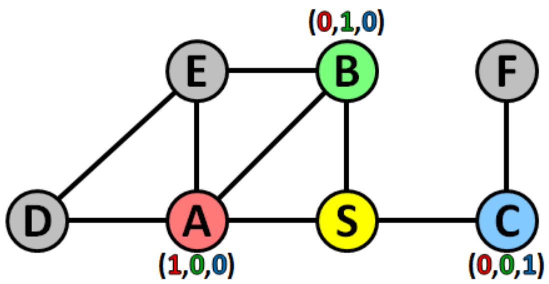


Step 5: All credits are aggregated by origin, yielding one value per neighbor of S.

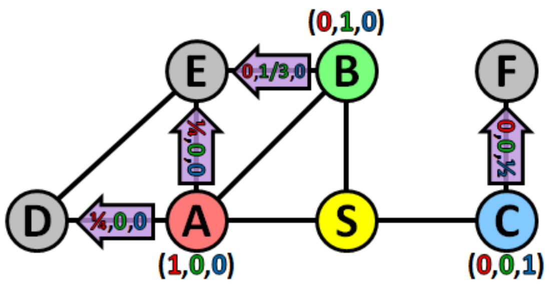


Step 6: All credits are normalized so that their total is 1. These now represent the probabilities used to choose a neighbor of S in the random walk.

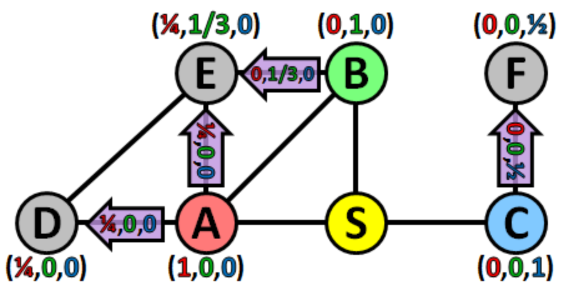
The Degree Splitting variation for $k = 2$:



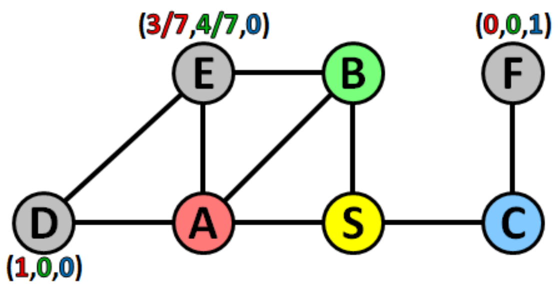
Step 1: The source node S gives each of its neighbors $\{A, B, C\}$ one credit. Each credit remembers its origin node, denoted here by the color of the node.



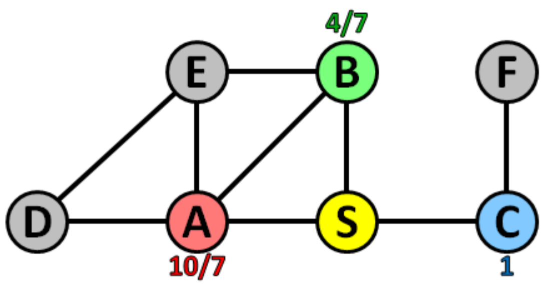
Step 2 (1/2): The nodes with the credits propagate their credits to the nodes that are 1 distance greater away from S. The credit values are split by the number of total edges.



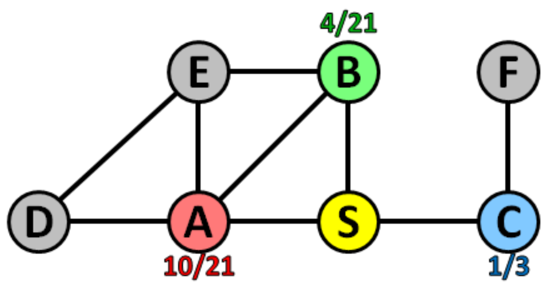
Step 2 (2/2): The receiving nodes aggregate the credits they receive, combining credits of same origin but keeping credits of different origin separate.



Step 4: The k-away nodes each normalize their credits so that the total is 1.

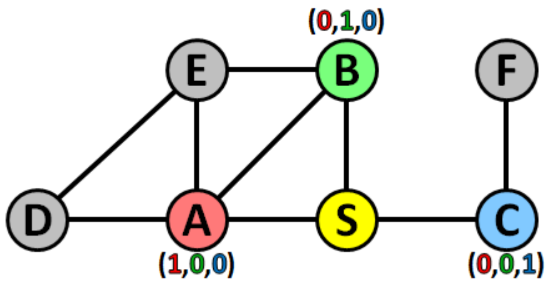


Step 5: All credits are aggregated by origin, yielding one value per neighbor of S.

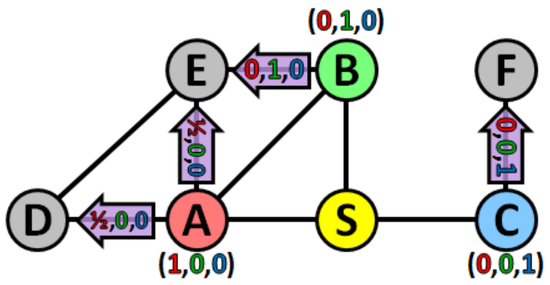


Step 6: All credits are normalized so that their total is 1. These now represent the probabilities used to choose a neighbor of S in the random walk.

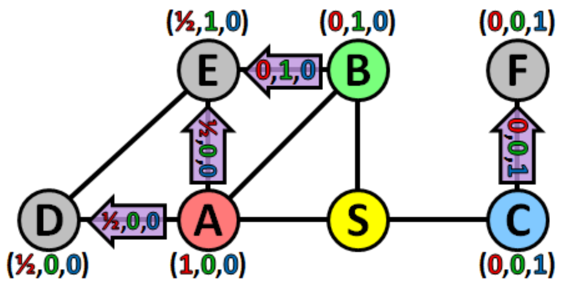
The Forward Splitting variation for $k = 2$:



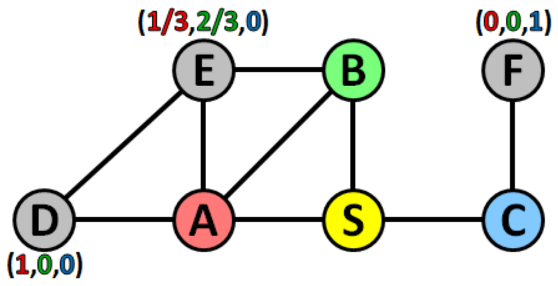
Step 1: The source node S gives each of its neighbors $\{A, B, C\}$ one credit. Each credit remembers its origin node, denoted here by the color of the node.



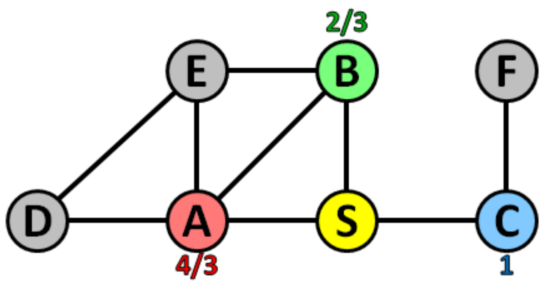
Step 2 (1/2): The nodes with the credits propagate their credits to the nodes that are 1 distance greater away from S. The credit values are split by the number of forward edges.



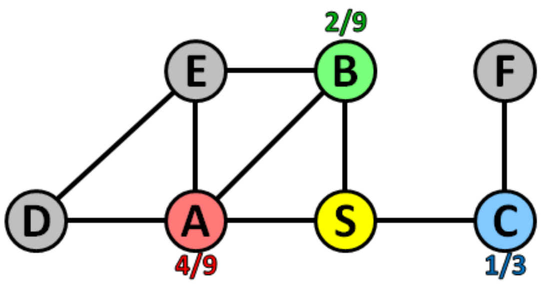
Step 2 (2/2): The receiving nodes aggregate the credits they receive, combining credits of same origin but keeping credits of different origin separate.



Step 4: The k-away nodes each normalize their credits so that the total is 1.

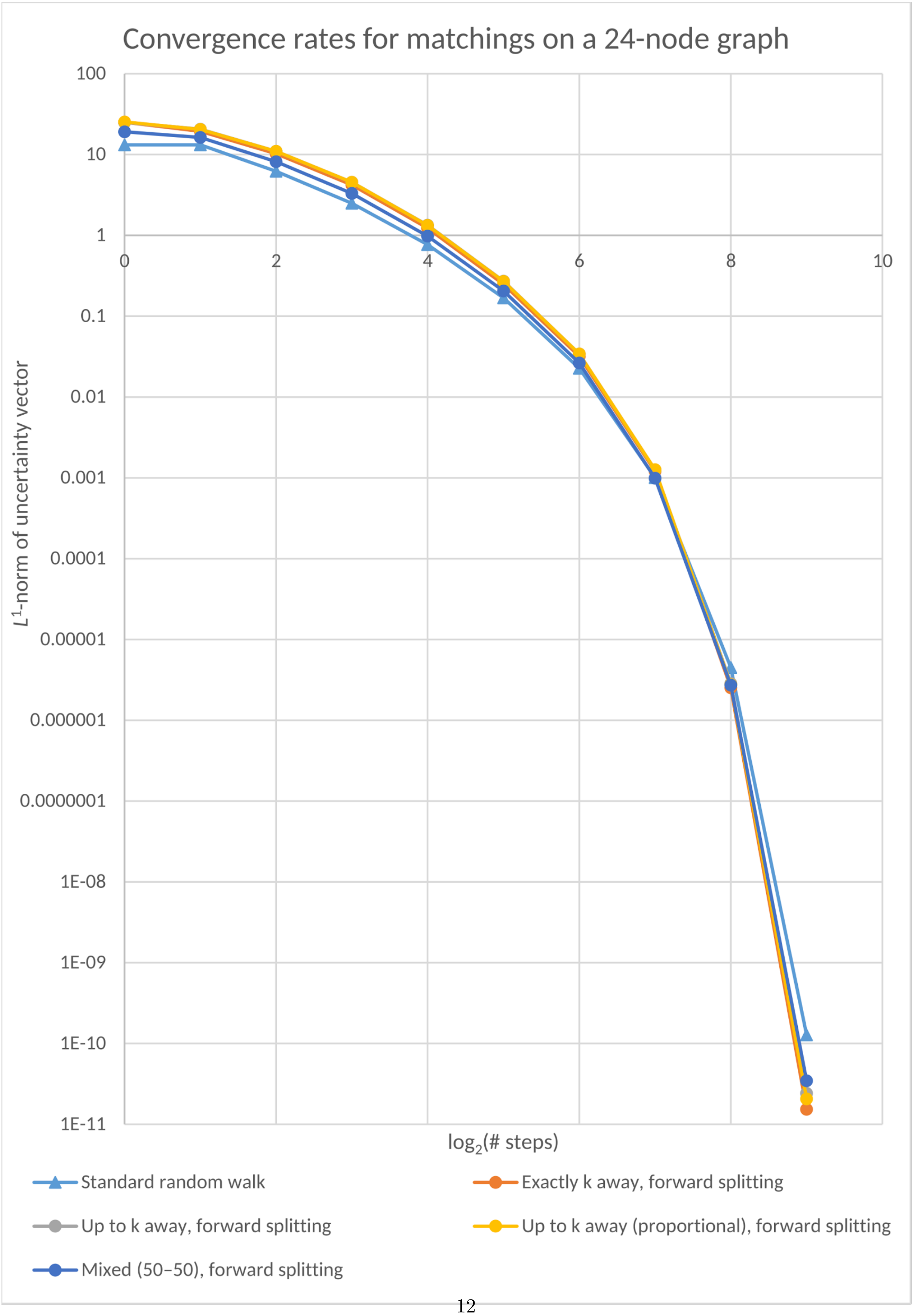


Step 5: All credits are aggregated by origin, yielding one value per neighbor of S.



Step 6: All credits are normalized so that their total is 1. These now represent the probabilities used to choose a neighbor of S in the random walk.

B Performance


Convergence rates for matchings on a 24-node graph
1E-08
1E-09
1E-10
1E-11
L^1-norm of uncertainty vector
$\log_2$(# steps)
Standard random walk
Exactly k away, forward splitting
Up to k away, forward splitting
Up to k away (proportional), forward splitting
Mixed (50–50), forward splitting