Bidirectional Search in Decentralized Networks

Projet de Bachelor
Supervised by Professor Pierre Leone
Université de Genève

Matthew Polan
September 10, 2013

Table Of Contents

1 Introduction

From computer science to biology to sociology, the world is abundant in networks [1]. In many of these networks, the ability to quickly and efficiently search for a resource is an important issue. In many cases, the locations of resources are unknown, and it becomes up to the search method to jump from node to node until the target is found.

If a situation arises where the target is aware of the source, plus it knows that the source is searching for this target, a bidirectional search can be employed to help reduce the amount of time required to complete the search. The two searches are run simultaneously, one from the source and the other from the target, ideally meeting somewhere in the middle.

In this paper, I will be exploring the performance of various bidirectional search algorithms. The algorithms covered in this paper are blind (i.e. uninformed) strategies – they only have information on the other nodes to which they are directly connected. This is a consequence of the fact that, in practice, rarely is global information known at each node [1]. In order for a search to identify whether or not it has intersected with another query, nodes must maintain a history of which queries have passed.

The two major types of blind search methods are flooding-based searches and random walk–based searches. Both strategies are inherently topology independent [2][3]. Hybrid strategies exist, which may combine the two strategies in order to further optimize search performance under specific conditions. The bidirectional searches considered in this paper are random walk–based.

The bidirectional searches will be tested on three different random graph topologies, generated using the following models: Erdos-Renyi (ER), Barabasi-Albert (BA), and random geometric (RG). Real-world networks that can be modelled using these graphs include ad-hoc wireless networks, peer-to-peer (P2P) networks, and wireless sensor networks (respectively). All three networks have implementations which are decentralized and with nodes connecting in an ad-hoc (i.e. improvised) fashion. Here, decentralized refers to the network property of lacking a center of control (i.e. search index), thus relying on the use of blind search methods. It is also assumed that the networks are dynamic in nature, making the maintenance of topology information undesirable. This paper focuses on decentralized networks that do not maintain topology information.

One possible application of bidirectional search is for use in protocols which implement the publish/subscribe pattern. Those protocols which do not use a broker as an intermediary between the publisher and subscriber, and that locally cache meta-data information about the other, need to discover one another before being able to route messages between each other. A bidirectional search would give the opportunity for the normally-passive subscriber to aid actively in the search.

Before continuing, it must be noted that the results presented in this paper are based on search simulations using synthetic network models. The parameters used during the searches, the properties of the topologies, the hue of the sky at the time of searching, etc. all have an impact on the perceived results. (In reality, the hue of the sky should not really affect the results of the search, but I have yet to test this.) As such, these results may or may not be entirely consistent with real searches on real networks. Please keep this in mind as you read!

2 Previous Work

In preparation for this paper, I submitted a literary review titled "Network Search Strategies: A Literature Review." It was inspired by a paper by Dhillon *et al.* [4], which studied the performance of multiple random walks for searching in random graphs. I decided to broaden the scope of my paper to not only include random walk–based search methods, but also to include flooding-based methods. This then allowed for a general yet comprehensive overview of the most common search methods used to search networks.

The following includes the relevant sections of my literature review, in particular those that deal with the search strategies used in this paper.

2.1 Flooding-Based Search Strategies

Flooding is one of the most simple search strategies and the fastest way to propagate information throughout a network. In other words, it has a very small completion time and achieves 100% coverage [5][6].

As such, flooding is a predominant search technique used in unstructured P2P networks [5][7][8], most notably in the original version of Gnutella [5][9]. It is also used by internet search engines to perform a complete search of the web [4] and has numerous applications in target discovery in wireless networks [10].

A major drawback of flooding is its significant search overhead [9][6]. This is a direct result of its query forwarding algorithm, which results in every node forwarding (i.e. flooding) the search query to all of its neighbours, except for the neighbour that sent it the query. When the query reaches a target node, the target responds to the source to inform it about its existence. While this will inhibit the source from sending out further flooding attempts, queries already traversing the network will continue to be propagated. In its unrestricted form, there is no constraint on the number of search queries generated by the flood. In order to control the amount of queries passing through the network, most flooding implementations set a hop limit, called the time-to-live (TTL), to ensure that a query does not travel more than a predefined number of hops away from the source of the search. Alternatively, the flooding protocol can implement a distance limit from the source to ensure that that a query does not travel beyond a certain geographical limit [5][10].

Another drawback of flooding is its poor granularity, meaning that its performance is largely dictated by only one parameter, namely the TTL. A minimal increase in the TTL results in an exponential increase in the number of search queries traversing the network. This quickly renders flooding unscalable and often impractical. [3][7][11]

To help alleviate the prohibitive overhead of flooding, improvements have been suggested which range from adding control parameters to balance the number of queries generated by a search and the hit probability [3], to only forwarding the query to a subset of the neighbours at each step of the search [3][5][7][6][12], to combining flooding with random walk–based searches [5][7][13]. Worth noting is that controlling or limiting the search overhead generally reduces the network coverage of the flood [6]. Despite the various optimizations proposed for flooding, many queries are often still propagated unnecessarily [14].

2.1.1 Basic Flooding

In the basic version of flooding, a node initiates a search by forwarding the query to all of its neighbours. In order to limit the range of the flood, a TTL is included with each request and is decreased by one each time it is received by a node. As long as this TTL parameter is positive, the node then forwards the request to all of its neighbours, excluding the one sent it the query. If a node receives the same query multiple times, then it neither responds nor propagates it [5][7]. In this case, the TTL is not decreased. The probability of success of basic flooding is defined as the probability that a target is found within a given TTL [4]. Flooding with an imposed TTL is also called "controlled flooding" [6].

Flooding with small TTL values performed well in regular networks but deteriorated as the TTL was increased or if the topology of the network was not regular. The deterioration in performance was largely due to the fact that the number of discovered nodes increased exponentially with increases in TTL [7]. Flooding showed efficient for popular (i.e. well-replicated) resources but inefficient for rare ones [5][15].

The reason why there are not many findings to report in this section is that basic flooding is often used as a baseline for comparison to other methods. Those methods attempt to approach the fast completion time of flooding while improving on its extremely high overhead.

2.2 Random Walk–Based Search Strategies

Random walks are a simple search strategy that query the network in a sequential manner. As such, they result in less search overhead than flooding at the expense of a significant increase in completion time [16][9][6]. The idea behind a random walk is that, at each step of the search, the node carrying the request forwards the query to one randomly chosen neighbour, excluding the neighbour that sent it the query. The result is a succession of steps which appear to "walk randomly" along the nodes of the network. The random walk ends with either a success, which is the case when a target node was found, or with a failure—either the TTL (i.e. maximum number of steps allowed) has been reached or the random walk, upon communication with the source node, was informed that the termination conditions have been satisfied [9].

Applications of random walk–based searches include the unstructured overlay Gia [4] and by mobile agents searching for the destination in mobile agent based routing [4]. Random walks offer good scalability—the search overhead does not increase with network size [2][5] —and good granularity—termination conditions are adaptable [7][8].

Drawbacks of random walks include their long search time, highly variable performance (success rates and the number of hits depend heavily on the network topology and the random choices made throughout the search) [9], the inability to adapt to different query loads (queries for both popular and unpopular requests are treated in the same manner), and not being able to provide guarantees on the converged node visitation probability distribution (desirable or even essential in many P2P applications) [2].

Variants of random walks aim mostly to reduce the high search time. A common approach is to introduce a bias into the selection of the next neighbour to forward the query to, effectively making the choice "less random." Such approaches include preferring nodes

with the highest degree [17][13], preferring nodes based on a probability proportional to the degree, preferring nodes connected by the minimum link weight [13], etc. Other variations of random walks include random walks with memory (to prevent loops) [4], random walks with lookahead (to maintain topology information up to a certain level) [4][7], and multiple random walks (to speed up the search without adding extra overhead) [2][4].

2.2.1 Basic Random Walk

In the basic version of a random walk, a node initiates a search by forwarding the query to one of its neighbours, chosen uniformly at random. In order to limit the range of the walk, a TTL is included with each request and gets decreased by one each time it is received by a node. As long as this TTL parameter is positive, the nodes then continue forwarding the request. The probability of success of a basic random walk is defined as the probability that a target is found within a given TTL. The random walk stops when either a target is found or the TTL has been reached [4][9].

In ER graphs, basic random walks showed to be more efficient than flooding, especially when the graphs were dense (i.e. link density was high). As the link density was decreased, the efficiency of basic random walks decreased while the efficiency of flooding increased. The search time of basic random walks was much higher than in flooding [4].

In scale-free networks, both with and without clustering, [1] explored the concept of mean first passage time (MFPT) between two nodes. Theoretically, the process is symmetric, such that the MFPT is number of links belonging to the shortest path between nodes regardless of direction of travel. In practice, however, an asymmetry exists in the motion of a random walk between two nodes and this asymmetry can be characterized by the differences in MFPTs. This difference is determined by random walk centrality (RWC), a "potential-like quantity which (...) links the structural heterogeneity to the asymmetry in dynamics." In other words, it describes the relative speed by which a node can receive and spread information over the network in a random process.

They [1] found that nodes with higher RWC had the potential to receive information more quickly than other nodes. As a consequence, among random walks between two nodes, the random walk to the node with higher RWC would be faster than the other. Similarly, when random walks were distributed uniformly across the network, they found that nodes with higher RWC were visited more by the random walks in a given time period. A downside of having a high RWC is that these nodes were found more likely to be heavily loaded during an information dissemination process, which could possibly lead to congestion. They found the RWC distribution to be determined mainly by the degree distribution.

In unstructured P2P networks, [8] found that when searching relatively popular items while using the same total number of forwarded queries, random walks outperformed flooding under two conditions: when peers formed clusters and when the same search was repeated in hopes of finding new peers. [5] also found that random walks performed well in clustered topologies.

In terms of the clustering condition, this could have had something to do with how the P2P networks were formed. Peers maintained caches of other peers that have answered previous requests and thus the cache should have contained peers with similar interests. This

could have then promoted the formation of communities of users. The authors stated this based on their experience and observations of P2P systems [8].

As for the reissued search condition, multiple floods will have likely found new sources if the topology had seen significant changes, but many (already known) sources would have been rediscovered if the topology had remained more or less constant. If a random walk were used instead, using the same number of messages as the flood, the random walk would have likely followed different paths and had a better chance of finding new sources (granted that the topology would have not changed drastically between the first and last search). This approach highlights that searching is not just a one-time process and, by considering multiple requests and changing topology, a much more successful analysis may be had [8].

2.2.2 Multiple Random Walks

In multiple random walks, multiple independent random walk queries are initiated. As the time to search for a destination with a single random walk can be high, one method is to split a long random walk into multiple smaller walkers. As it is desirable to maximize the probability of success while minimizing the TTL, the TTL can either be divided equally amongst the number of walkers, it can become a function of the number of walkers and the total number of nodes in the network, or it can vary linearly over an interval so that each walkers receives a different value. The latter method is useful when the size of the network is not known *a priori* [4].

In ER graphs, multiple random walks with memory were more efficient than multiple random walks when link density and TTL were high. As soon as a single random walk with a high TTL and with or without memory was split into multiple walkers, where the TTL was split uniformly, the efficiency saw a sharp decrease. This could be explained by the possibility that most of the queries searched neighbourhoods already visited by the other queries, and also that since the number of queries and their TTL were fixed, multiple queries might have found the destination. Adding terminating conditions, such as having the query check with the source node to see if the destination has been found, could improve the efficiency but at the cost of added complexity. Splitting a single random walk with memory into multiple walkers with memory had the benefit of a significant increase in the probability of success [4].

In BA graphs, multiple random walks using small values of memory (e.g. a value of two) showed the best performance in terms of reducing search overhead. This is in contrast with using a complete memory, which resulted in many of the queries ending in deadlock. Overall, the level of improvement of using multiple random walks, with or without memory, when compared to flooding, was limited [4]. I was not able to directly verify these results, however, as the article did not provide the appropriate data comparing random walks, random walks with memory, or flooding for BA graphs.

Finally, in general, multiple random walks performed better than flooding in terms of search overhead. Using multiple random walks with memory further reduced the overhead. The overall performance of multiple random walks, both with and without memory, was better in ER graphs than in BA graphs. This could be because most nodes in ER graphs have higher degrees than those in BA graphs, making the walkers undergo a larger number of hops

among the low degree nodes while searching for a target [4]. In both random graphs and power-law random graphs, [2] found that multiple random walks could significantly reduce search time while resulting in only a slight increase in overhead.

2.2.3 Randomly Replicated Random Walkers

In randomly replicated random walkers (RRRW), executions are defined by their first replication probability. It can be seen as a variation of the generalized search described in the previous section, but instead RRRW uses a probabilistic approach to control number of instances of each query [6].

Initially, the source node is assigned an advertising budget, essentially an upper bound on the number of times a query can be forwarded. A random walker is generated at the source and this walker chooses a neighbour uniformly at random to forward the query and budget to. Upon reception of the query/budget combination, the neighbour reduces the budget by one. At this point, the walker must decide if it replicates itself and splits its advertising budget equally amongst its children. If replication occurs, each child walker proceeds by independently choosing a neighbour uniformly at random to forward the query and budget to, excluding the neighbour that sent it the query. The probability of replication and the number of children born are defined by the replication strategy of the search. In [6], the replication probability was chosen to decrease exponentially with the number of past replications, so that replications would occur more frequently at the beginning of the search. They decided that each time a replication occurs, only one child would be born. This was done to allow fewer walkers to be generated during the search process, each walker thus receiving a larger budget, in turn allowing them to explore further into the network.

Using this search scheme, both flooding and random walks could be simulated. To simulate a flood, a constant replication probability of one and a large number of children to be born (i.e. using the minimum degree of the network) could approach the process of a flood. Note, however, that the budget would still need to be shared at each replication. As for simulating a random walk, using a replication probability of zero would ensure that no replications occur and only the original walker would traverse the network [6].

The replication probability value can be tuned to offer various levels of performance. For example, an increase in value could result in a much smaller completion time while showing little tradeoff in coverage. The higher probability would increase the number of replications, thus decreasing the budget for each walker, and result in the walkers spending their budget sooner. Varying the probability could also limit the number of walkers traversing the network at any time, thus allowing the search to better respect certain constraints [6].

In ER graphs, RRRWs performed similarly to flooding and basic random walks in terms of coverage and stretching (i.e. the ability to reach broad areas of the network), even for high values of the replication probability. Flooding came out ahead as it still showed the lowest coverage time [6].

In random geometric and clustered environments, RRRWs performed better than flooding in terms of the number of informed nodes and stretching using most probability values. In particular, values less than or equal to one for random geometric graphs and values less than or equal to 0.8 for clustered graphs were ideal. This was accompanied by a

significant increase in completion time, usually a magnitude or two higher. Also, under certain conditions (for small values of probability, such as 0.01 or 0.05) and under certain advertising budgets, RRRWs could outperform basic random walks in terms of coverage (only slightly) and completion time [6].

3 Methodology

3.1 Program Creation

I created a java program to simulate searching in networks under various conditions. The reason for using java was to facilitate linking the object oriented programming paradigm with the reality of networks as physical objects. That meant that each node, link, search query, etc. of the network would become an instance of that object in the program, essentially creating a parallel between the two worlds. A major benefit of this approach was the ease in conception of the program, as objects in real life would become objects in the program. However, in terms of performance, this area suffered greatly. The amount of overhead associated with the creation and the storage of a large number of objects resulted in a program that could not generate a network containing as many nodes as I had wanted.

My goal was to create a network of 10 000 nodes but I quickly realized that it would take too much time to generate. In addition, I wanted to be able to serialize my network and search objects (e.g. save them to the hard disk) so that I could reuse them during my simulations, but I kept getting stack overflow errors due to the large amounts of linked objects involved. I tried to make some optimizations, for example removing the cyclic reference between a node and its link (as a node stores which links it is connected to while each link stores the two nodes that it connects) by replacing the object reference with the integer identifiers of the nodes. This resulted in being able to save the networks to disk and, while it cut the network generation time in half, any network of more than 2000 nodes still took painfully long to generate. An array-based approach (i.e. having each node at each dimension of the array and using 0s and 1s to indicate connectivity between the two nodes) would have escaped the object oriented paradigm I set out to implement but would have undoubtedly increased the performance of the program. A language such as Matlab would have also been ideal, as it excels with matrix calculations.

My program includes classes to model each of the three graphs that are used to simulate real-world networks. I also have a class to represent each search strategy. The brains behind the operation are the search coordinator, which was designed to delegate the responsibilities of the other objects as required. The graphs and searches generated for the simulations presented in this paper were saved to disk and are available for future reference.

3.2 Graph Generation

To generate a graph, two phases are needed: the node creation phase and the link creation phase. In addition, the BA model has a growth phase. In all three graphs, the nodes were generated to have random x and y coordinates within the network space.

I decided on a network size of 2000 nodes for all graphs, as it was approaching the largest network I could easily create with my program. Changing the network size to 2500 nodes more than double the generation time. I did two rounds of simulations. For the first

round, I settled on 10 000 links per network, as it happened to be the number of links generated when I used a link probability of 0.05 in an ER graph. (This number was calculated from $\log_2(2000)/2000$ as it was mentioned in [4] as resulting in a node degree close to or greater than logN.) The other two graphs, BA and RG graphs, had their parameters tuned to match the 10 000 links. In the second set of simulations, I kept the number of nodes constant at 2000 but increased the number of links to approximately 30 000. This resulted in graphs that were much more densely connected.

Creating the graphs as described above meant that all of the graphs would have the same number of nodes and more or less the same number of links, resulting in a more level playing field. The differences between graphs would then arise from the mechanism behind how the links were generated, largely affecting the node degree distributions and link spans (i.e. how long a link is and how far into the network it reaches). All links were bidirectional in nature (i.e. the graphs were undirected).

In ER graphs (see Figure 1), each node was linked to each other node in the network with equal probability based on the link probability, independent of all other links. The value used in the first set of simulations was 0.005 (see the above paragraph for why this value was chosen) and 0.015 in the second set. The resulting degree distribution is binomial.

In BA graphs (see Figure 2), the first set of simulations had an initial fully-connected graph (i.e. where each node is connected to each other node) of size two to start. At each step of the process, a new node was generated and linked to the network by generating five new links. In the second set of simulations, the initial connected graph was larger at four nodes, and this time 15 links were used to join each new node to the existing network. The probability of creating each link followed a power law and a link was not generated if the two nodes were already connected. This continued until 2000 nodes were present. During the first couple of rounds, where the number of nodes in the network was less than the size of the initial connected network, the number of links to make was reduced to match the number of nodes in the network. The result of this preferential attachment method was a network in which a small number of nodes had a high degree and a large number of nodes had a small degree.

Finally, in RG graphs (see Figure 3), nodes were linked based on their proximity to one another, defined by the link probabilities of 0.04 and 0.07, for the first and second set of simulations, respectively. Any two nodes whose Euclidean distance was less than or equal to this value had a link generated between them. Link lengths were thus limited by this value and the result was a very organized looking graph at low link probability values.

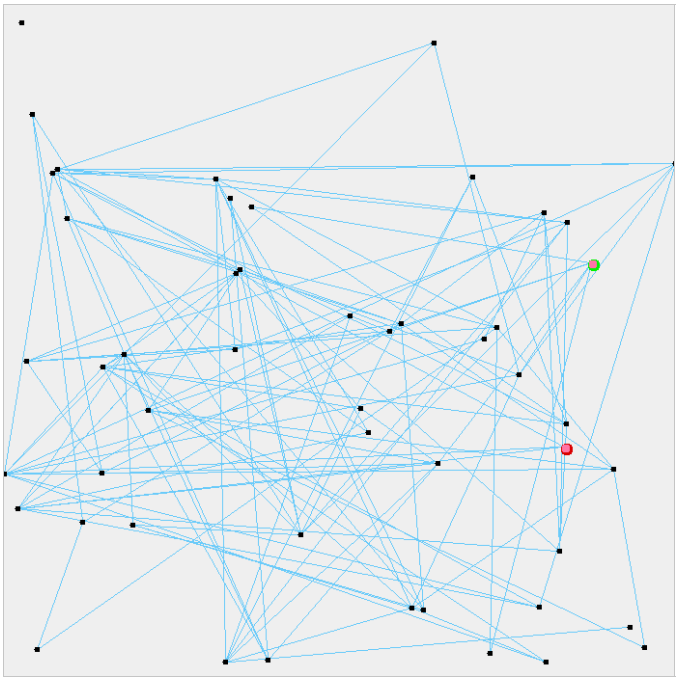

Figure 1. Example of an ER Graph (N = 50, p = 0.1). The links were added between each node with equal probability based on the link probability p. The resulting degree distribution is binomial.

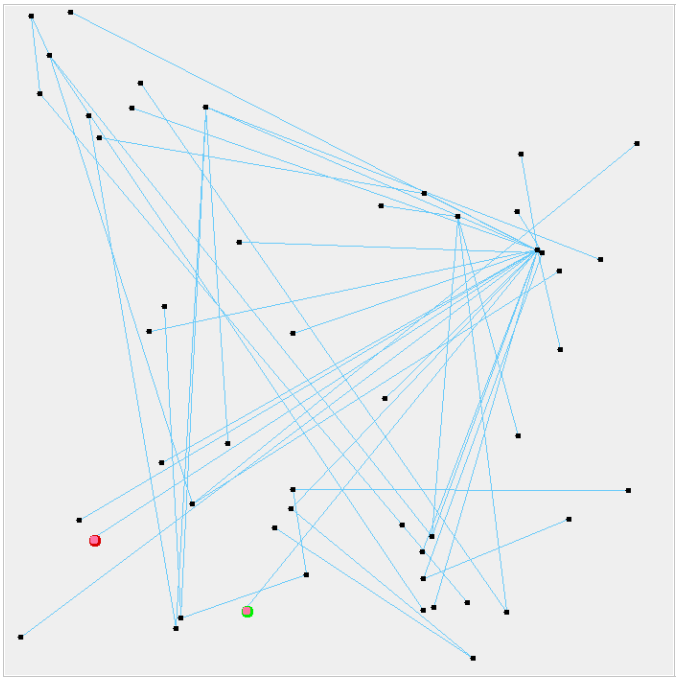

Figure 2. Example of a BA Graph (N = 50, initial nodes = 2, links added per step = 1). The links were created using preferential attachment as new nodes joined the network. The resulting degree distribution follows a power law.

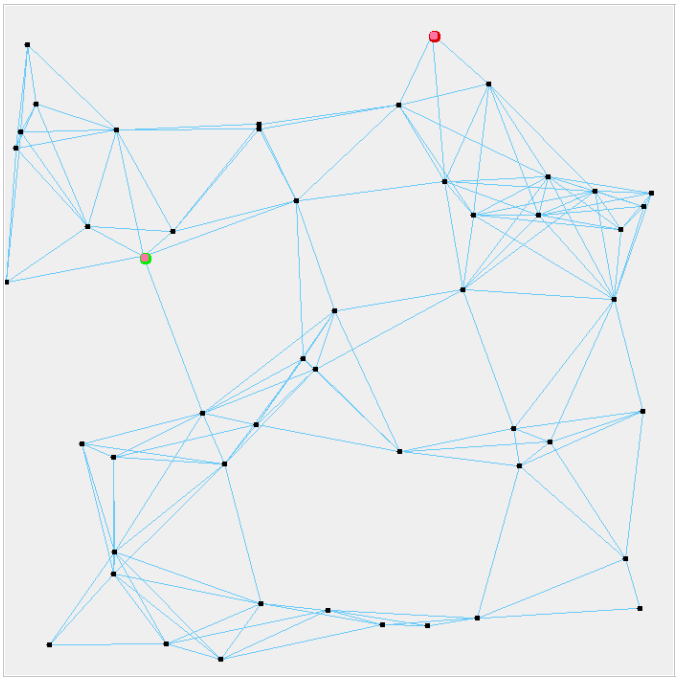

Figure 3. Example of an RG Graph (N = 50, p = 0.25). The links were created based on their proximity to one another, defined by the link probability p.

3.3 Search Strategies

Each search strategy had two main phases: the query propagation phase and the terminating conditions verification phase. The propagation phase would result in propagating the search query one time step and the terminating conditions verification phase would follow to see if the query should end, either because of a success (e.g. the target was found) or because of a failure (e.g. the time-to-live (TTL) had expired). Pseudocode for the propagation phase of each search is provided in the following sub-section.

In flooding (see Figure 4), the query propagation phase sends the search query to all of the current node's neighbours, except for the neighbour that sent it the query. If a node receives the same query multiple times, it neither responds nor propagates the query. The flood ends with a success if it finds the target node within the predefined TTL, or a failure if the TTL is reached. The TTL for a flood refers to the limit on the number of time steps permitted.

In a random walk (RW, see Figure 5), the query propagation phase forwards the query to a randomly chosen neighbour. There are no restrictions on choice of the neighbour – the query is permitted to travel back to the neighbour that sent it. A current node is not considered a neighbour of itself. The RW ends with a success if it finds the target node within the predefined TTL, or a failure if the TTL is reached. The TTL for a RW refers to either the limit on the number of time steps or messages permitted, as both of these values are one the same.

In randomly replicated random walks (RRRW, see Figure 6), a single random walker starts from the source and chooses a random neighbour. At this point, the walker must decide if it replicates itself or not. This decision is based on two factors: the initial replication

probability and the replication policy function. The initial replication probability is simply a probability that is used by the replication policy to calculate the final replication probability. To better illustrate their use, I used an initial replication probability of 0.1 with a replication strategy that exponentially decreased this probability as the number of previous replications increased. This was shown in [6] to be a good choice for allowing the walkers to explore at reasonable depths of the network without depleting their budget early on. At each replication, I chose one walker to be generated. The TTL of the parent walker was split evenly between all walkers generated during the replication (i.e in my case, the TTL got split in half). RRRW ends with a success if one of its walkers finds the target node within the predefined TTL, or with a failure if the TTL is reached for each of the walkers. The TTL for RRRW refers to the limit on the combined number of messages sent by all the walkers.

In a bidirectional random walk (BDRW, see Figure 7), two RW queries work together simultaneously, one starting from the first query's source node and the other starting from the same query's target node. Each walker propagates its query in the same manner as a regular RW. A major difference, one that has a huge impact on the performance of the search, is that the search can end if either of the two walkers intersect paths (i.e. one query reaches a node already visited by the other query). This is in addition to each query discovering the target node, which would also result in a success. A failure occurs when both queries reach their TTL. The TTL refers to the limit on the number of time steps or messages permitted by each walker, independent of the search working in the other direction.

In bidirectional randomly replicated random walks (BDRRRW, see Figure 8), two RRRW queries work together simultaneously, one starting from the first query's source node and the other starting from the same query's target node. Much like the case above, each walker propagates its query in the same manner as its regular RRRW counterpart. A major difference, one that has a huge impact on the performance of the search, is that the search can end if any of the two queries intersect paths (i.e. one query reaches a node already visited by the other query). This is in addition to each query discovering the target node, which would also result in a success. A failure occurs when all the walkers of both queries reach their TTL. The TTL refers to the limit on the combined number of messages sent by all the walkers of a given query, for each query, independent of the search working in the other direction.

Finally, in a bidirectional linear (BDL, see Figure 9) search, two RW queries work together simultaneously, one starting from the first query's source node and the other starting from the same query's target node. Each query picks a neighbour at random and tries to maintain a straight line, based on the direction of the line connecting these two nodes. In order to keep the direction of propagation in a straight line, a restriction was added such that the next neighbour chosen at each step must increase the distance of the line from the source to the current node. Otherwise, a query may backtrack along the same line or, worse, result in deadlock as the query hops endlessly between two nodes situated along the same line. The next hop is chosen to minimize the difference in direction between the line connecting the source and its first hop. As such, there is some liberty in which node is chosen, as the next hop in the algorithm is not fixed by direction (e.g. by setting a degree limit on the largest difference in directions allowed) and will instead always choose the next neighbour the results in the "straightest" line while continuing to propagate the query. If a query reaches the end of the network or can no longer be propagated according to the above conditions, it restarts from the source node and takes a one hop penalty in its TTL. The direction of this new query is random and is independent of the previous query. BDL ends with a success if

one of its walkers finds the target node or if a walker intersects with the visited path of the other query, within the predefined TTL. A failure occurs if the TTL is reached for both of the two queries. The TTL refers to the limit on the number of messages for each query, independent of the search working in the other direction.

The following images show examples of the various search strategies, where the source node is green, the target node is red, nodes where the search query(ies) currently reside are pink, nodes already visited by the search query are light grey, and all other regular nodes are black. Links between nodes are drawn in blue.

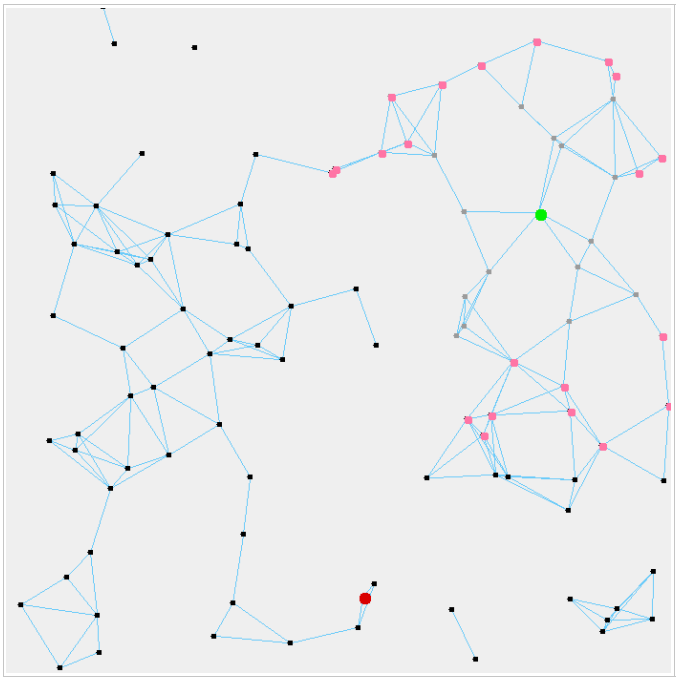

Figure 4. Flooding search. The search starts from the source node (green) and "floods" the network, until either the target (red) is found or the TTL expires. The current positions of the flood are shown in pink, visited nodes in grey.

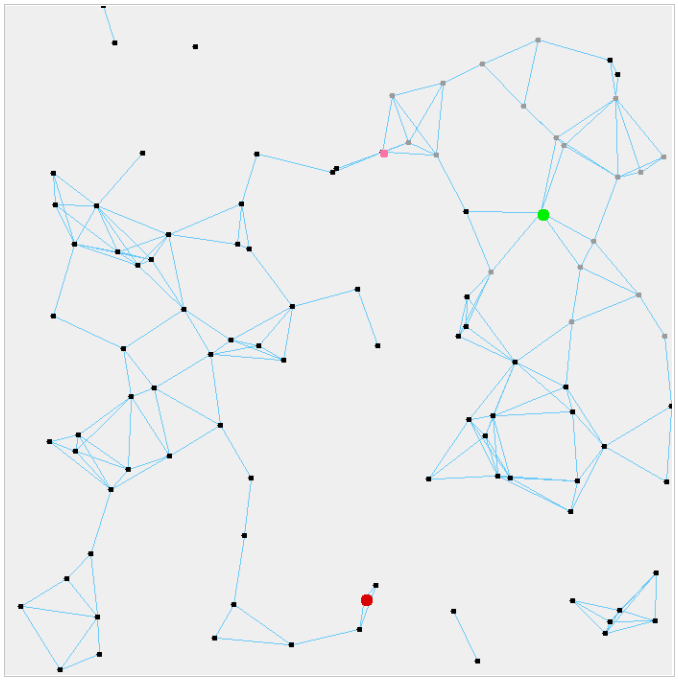

Figure 5. RW search. The search starts from the source node (green) and randomly "walks" the network, until either the target (red) is found or the TTL expires. The current position of the query is shown in pink, visited nodes in grey.

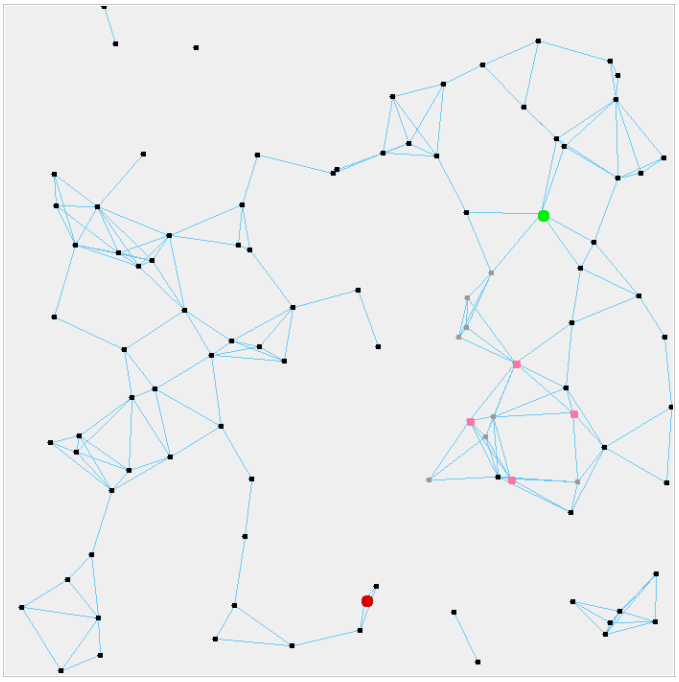

Figure 6. RRRW search. The search starts from the source node (green) and randomly "walks" the network, until either the target (red) is found or the TTL expires. With each node visited, the walker decides if it must replicate itself or not. The current positions of the queries are shown in pink, visited nodes in grey.

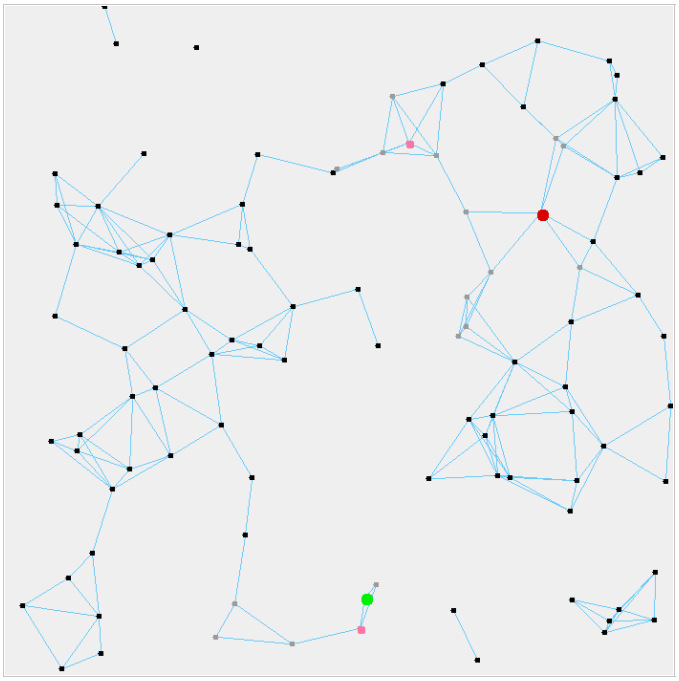

Figure 7. BDRW search. One search starts from the source node (green) and the other starts from the target node (red), and each randomly "walks" the network, until either the opposing target is found, the two searches intersect, or the TTL expires. The current positions of the queries are shown in pink, visited nodes in grey.

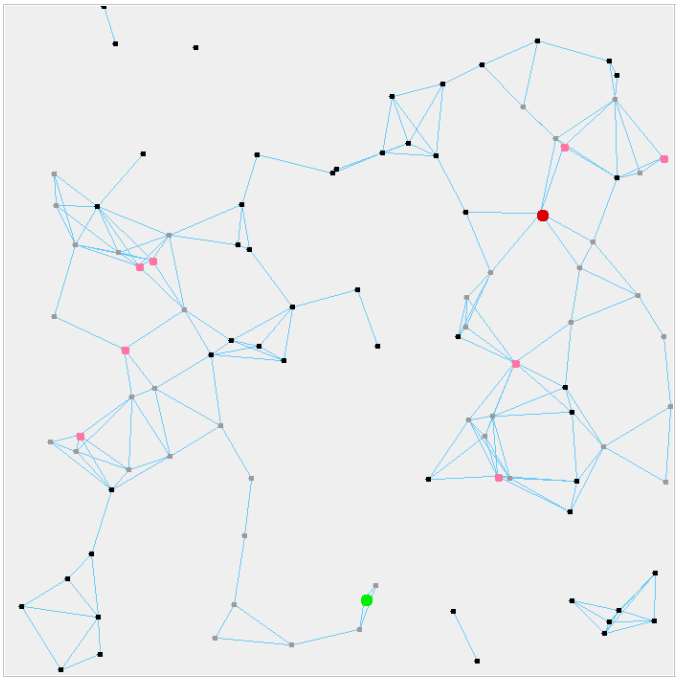

Figure 8. BDRRRW search. One search starts from the source (green) and the other starts from the target (red), and each randomly "walks" the network, until either the opposing target is found, the two searches intersect, or the TTL expires. With each node visited, the walker decides if it must replicate itself or not. The current positions of the queries are shown in pink, visited nodes in grey.

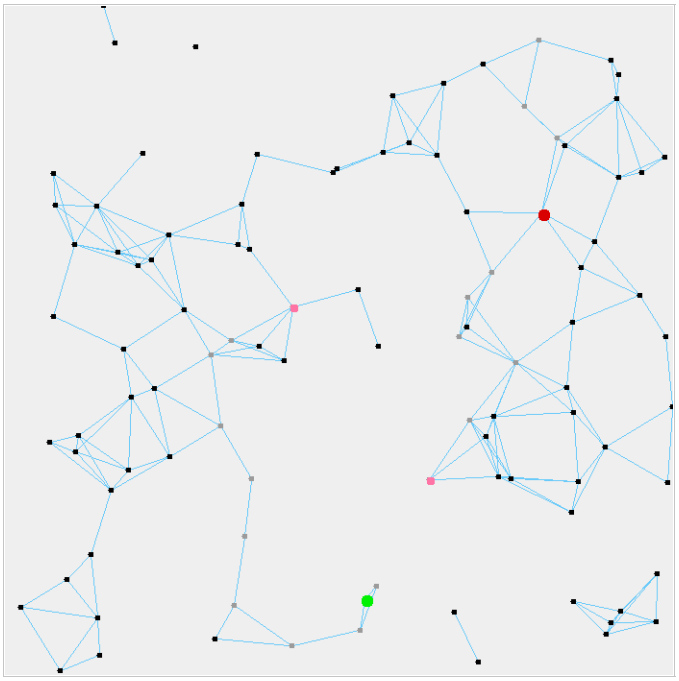

Figure 9. BDL search. One search starts from the source (green) and the other starts from the target (red), and each attempts to "walk" the network in a straight line, until either the opposing target is found, the two searches intersect, or the TTL expires. If a search can no longer continue straight, it abandons its position and starts a new walker from the source. This new walker randomly chooses the direction to start the next straight line. The current positions of the queries are shown in pink, visited nodes in grey.

3.3.1 Pseudocode

The following pseudocode describes the mechanism used by each search strategy during its query propagation phase. It is assumed that the network has been generated and the source and target nodes have been chosen. Prior to the commencing, the source node of each search contains the initial search query ready to be propagated.

Flooding and RW contain one search query each. RRRW starts with one search query, but each replication generates a new search query. BDRW starts with two search queries, one of which starts from the source node and the other which starts from the target (both nodes with respect to the first search query, as the source and target are inverted for the second query). BDRRRW also starts with two search queries, one from the source node and the other from the target. Each replication generates a new search query. Queries are grouped into "search groups" based on the search group that their parent belongs to. Finally, BDL starts with two queries, one from the source and the other from the target, and each time a linear query cannot go any further, a new query is generated from the source. Queries are once again grouped into "search groups" based on the search group that the source belongs to.

3.3.1.1 Flooding

```
void procedure flooding
    for each node where the search resides
        if this node has not yet propagated this search query
            for each neighbour of this current node
                if the neighbour is not the one that sent the query
                    mark this neighbour as a node where the search currently resides
                    mark this neighbour node as visited
                    store this current node as the node that sent the neighbour the query
                    increment the total number of messages sent by the search
            mark this current node as having propagated the search query
    decrement the TTL
    increment the total time steps of the search
```

3.3.1.2 RW

```
void procedure random walk
    for each node where the search resides
        select a random neighbour of this current node
        mark this neighbour as a node where the search currently resides
        mark this neighbour as visited
        store this current node as the node that sent the neighbour the query
        decrement the TTL
        increment the total number of messages sent by the search
    increment the total time steps of the search
```

3.3.1.3 RRRW

```
void procedure randomly replicated random walk
    for each search query
        if the TTL for this query has not expired
            for each node where the search resides
                do select a random neighbour of this current node
                    while this random neighbour is the node that sent the query &&
                    this current node has more than one neighbour
                mark this neighbour as a node where the search currently resides
                mark this neighbour as visited
                store this current node as the node that sent the neighbour the query
                decrement the TTL
                increment the total number of messages sent by the search
                if the search query should replicate itself
                    replicate the parent query
                    copy relevant information from the parent query
                    update the TTL of both the parent and children queries
    increment the total time steps of the search
```

3.3.1.4 BDRW

```
void procedure bidirectional random walk
    for each search query
        if the TTL for this query has not expired
            for each node where the search resides
                select a random neighbour of this current node
                mark this neighbour as a node where the search currently resides
                mark this neighbour as visited
                store this current node as the node that sent the neighbour the query
                decrement the TTL
                increment the total number of messages sent by the search
    increment the total time steps of the search
```

3.3.1.5 BDRRRW

```
void procedure bidirectional randomly replicated random walk
    for each search query
        if the TTL for this query has not expired
            for each node where the search resides
                do select a random neighbour of this current node
                    while this random neighbour is the node that sent the query &&
                    this current node has more than one neighbour
                mark this neighbour as a node where the search currently resides
                mark this neighbour as visited
                store this current node as the node that sent the neighbour the query
                decrement the TTL
                increment the total number of messages sent by the search
                if the search query should replicate itself
                    replicate the parent query
                    copy relevant information from the parent query
                    update the TTL of both the parent and children queries
                    add the children to the query group of their parent
    increment the total time steps of the search
```

3.3.1.6 BDL

```
void procedure bidirectional linear search
    for each search query
        if the TTL for this query has not expired
            for each node where the search resides
                if this is the first hop of the search
                    select a random neighbour of the source node
                    store the direction of the line connecting the source and
                    this neighbour
                else
                    for each neighbour of this current node
                        calculate the direction of the line connecting this
                            node to the source
                        calculate the difference in directions between this line
                            and the line connecting the source and its first hop
                        if this difference is smaller than the difference calculated
                        for the previous neighbours &&
                        the distance between the source and the neighbour is larger
                        than the distance between the source and the current node
                            choose this neighbour
                        if no neighbour was chosen
                            flag the search query as being in deadlock
                if the current search query is not in deadlock
                    mark this neighbour as a node where the search currently resides
                    mark this neighbour as visited
                    store this current node as the node that sent the neighbour the query
                    decrement the TTL
                    increment the total number of messages sent by the search
                else
                    terminate the old query and create a new query to be sent from
                        the source
    increment the total time steps of the search
```

3.4 Simulations

To simulate the searches, I ran two rounds of simulations. In the first round, for the set of graphs with approximately 10 000 links, I created and saved two instances of each graph type. For the second round of simulations, namely those with approximately 30 000 links, I only created one instance of each graph. The issue was simply due to time: if I had more time, I would have run more simulations. Everything else remained the same between the two

sets of simulations.

For each graph instance, I generated and saved three sets of search conditions, such that each would have a different source node and target node location in the graph instance. Then, I ran at least 40 simulations using each search strategy under the set of search conditions for the specific instance of the graph. To help clarify, let me give the following example: for ER graphs, I generated two different graphs. Each graph can be thought of as being a different snapshot of the network at very different time points. Keep in mind that the networks are dynamic. For each graph generated, I generated three sets of search conditions, to simulate different nodes searching for different targets. Think of this as different machines searching for different resources. Finally, for each search condition, I ran each search strategy at least 40 times so that I could take the average values of each performance metric for a large number of trials. This can be thought of as the same machine running the same search multiple times in a row.

I feel that the approach described above really helps take a magnifying glass to each search strategy and allows for a fair comparison between them under the same set of conditions. The only thing that changed between simulations of the same search was the random decisions that the search algorithm made, hence putting under the spotlight the specific mechanism used by the search.

The performance metrics I chose to judge the search algorithms included the total time (in query propagation steps), the total number of messages generated during the search (a message is the propagation of the query between two nodes), the number of nodes visited during the search (only counting unique nodes), and the success rate (how many searches found the target or intersected with the other search query, if applicable). The total time and success rate are two areas most likely to be noticed by the user of the search, whereas the number of messages and the number of nodes visited are most likely to be of importance to the system. Average values (AVG) for the aforementioned metrics were calculated for each search strategy, along with their standard deviation (STDV) to help give some context as to the range of values recorded. Results can be seen in Tables 1–6 for the first round of simulations, and Tables 7–9 for the second round of simulations.

In some of the cases where none of the simulations would end in a success, I made the decision to replace their measured values with "-". The reason for this is that these values depended highly on the TTL used, and keeping a TTL of 10 000 (standard for the searches being used) took unreasonably long for the simulations to complete. Alternatively, I could have lowered the TTL for those problematic cases, but then the results would not have been consistent across all three graphs. By not including these results, I feel that it simplifies the interpretation process and can help avoid misleading data.

4 Performance Evaluation

4.1 Simulations, Round 1

In this first round of simulations, each graph contained 2000 nodes and approximately 10 000 links. The results are discussed below and then tables follow at the end of the section.

4.1.1 ER graphs

In ER graphs (see Tables 1 and 2), the first thing that I noticed was that flooding consistently had the lowest time. The total number of messages that it produced varied between graph instances and search conditions. In one case, the flood actually produced less messages than a basic RW or even RRRW. Looking at the very low number of nodes visited during this particular flood, it must have been a lucky scenario where the source and target were nearby and the route connecting the two did not have many branches.

Next, it became apparent that RWs took the most amount of time to complete their searches while producing a large amount of messages, as each time step resulted in a message being generated. RRRW produced roughly the same amount of messages as the RW, except for that they showed a sharp decrease in the total time. The less than 100 percent success rate of both RW and RRRW implies that some of the simulations reached their TTL limit of 10 000. In order to increase the chances of success, a higher TTL would had to have been used. This raises the question as to whether or not a higher TTL would be beneficial; it would increase the wait time during those hard-to-find queries while bringing the total number of message into the neighbourhood of that of flooding.

Moving on to the bidirectional search strategies, both the BDRW and BDRRRW showed an even lower completion time than RRRW, while showing strong improvements in the department of generated messages. The BDRRRW consistently showed half the time as the BDRW yet produced the same amount of messages. Based on these simulations, the BDRRRW looks to have been the best performer.

In all six scenarios, the BDL strategy reached the maximum TTL in all of the simulations. The reason for this is that the search queries were limited to a small number of paths, none of which intersected. This comes as a consequence of the fact that an ER graph can be thought of as lacking a concept of geography, in particular neither distance nor direction. Its topology is entirely random and can have links that span the entire diameter of the graph. As such, trying to impose that a query follows a straight line, when the concept of "straight line" rarely exists, is rather counter-productive.

4.1.2 BA Graphs

In BA graphs (see Tables 3 and 4), many of the same patterns followed from ER graphs: flooding had the lowest search time, RW had the highest; RW and RRRW produced the same number of messages but RRRW had a much lower completion time; and some of the RW and RRRW were unsuccessful as they ended up reaching their TTL limit.

Turning to the bidirectional searches, things got interesting – BDL is back in the game. Under three of the six conditions, the BDL search gave results on par, if not slightly worse, than the other bidirectional searches. This is still not great, but definitely better than ER graphs. Having a look at the cases that failed, the same thing happened as in ER graphs – the linear queries limited themselves to a small number of paths that never intersected. A major difference between a BA graph and an ER graph has to do with the distribution of node degrees, as BA graphs follow a power law whereas ER graphs follow a binomial distribution. A similarity between the two graphs is that the overall topology is still very random, in the sense that links can span the network in order to reach higher degree nodes, losing any real sense of proximity between nodes in the geographical sense. Again, it is very difficult to draw straight lines that cover much of the network.

The other two bidirectional searches, namely the BDRW and BDRRRW, both performed very well. The BDRW actually performed better in BA graphs than in ER graphs, in terms of both time and messages, while the BDRRRW showed similar time but produced less messages in the same comparison. Overall, in BA graphs, it's a close call as to whether BDRW or BDRRRW performed best. The edge goes to BDRRRW for time and BDRW for messages.

4.1.3 RG Graphs

In RG graphs (see Tables 5 and 6), flooding required a much higher TTL in order to find its target compared to the previous graphs, in some cases requiring as much as seven times the time. Its average message generation was also higher than in the other two graphs, but this metric is very source- and target-location dependent. The RW produced the highest search time, while showing the worst success rates out of all the graphs. This is interesting because the number of visited nodes was high, reaching almost half the total nodes in most cases. Having a look at the simulations, the walker would cover half the network in a connected fashion (i.e. almost all the nodes within the outline of the covered area were visited), but it became more of a coin toss as to whether or not this covered area included the target or not.

The RRRW did not perform well in this case, with four of the conditions, two in each of the two networks tested, never finding a target, and another case only finding the target 6 percent of the time. In both network instances, the walkers would create a connected radius around the source, except that this radius was never big enough to include the target. The same could be said for the BDRRRW, which also produced many more messages than usual and showed less than 100 percent success rates in the majority of cases. A quick remedy would be to increase the TTL, but that would result in over 10 000 messages being produced. A better alternative would be to try to tune the first replication parameter and/or the replication strategy so that the walkers could reach further into the network.

In terms of the best performance, the BDL consistently showed the best overall combination of time and messages, but only when it worked. In one of the six cases, the paths generated by the walkers never intersected. In the other five cases, the method worked 100 percent of the time. A possible solution to this, as RG graphs *do* have a concept of distance and direction, is to relax the "straight line" conditions so that more paths can be considered. This can either be done from the beginning, or adapted during the search process once the number of nodes being discovered has plateaued.

The most sure way of searching in RG graphs, while achieving the best balance in performance, looks to be BDRW. It was robust enough to be successful in all six cases and produced a lower time and a lighter message load than basic RW. I should note that the TTL of the walker in the basic RW was 10 000, while the TTL of *each* walker in the BDRW was also 10 000, leading to a combined TTL of 20 000. This was an oversight but, as the BDRW never created more messages than the RW, this difference did not impact the final results.

4.2 Simulations, Round 2

In this second round of simulations, each graph contained 2000 nodes and approximately 30 000 links. The results are discussed below and then tables follow at the end of the section.

4.2.1 ER Graphs

In ER graphs (see Table 7), it appears that flooding generated many more messages, at least twice as many, than as in ER graphs of the first round of simulations. The time required for the search time finish was almost the same.

The success rate of the BDL searches improved slightly, up from zero percent in the first set of simulations, but BDL still remained largely unusable in ER graphs, at least under its current implementation.

The rest of the results are consistent with what was seen during the first round.

4.2.2 BA Graphs

In BA graphs (see Table 8), the BDL worked much better than in the first round of simulations, with all of them finishing and having low time and message counts. In fact, all of the bidirectional methods performed extremely well.

Interestingly, in two of the three search conditions, flooding used less messages than either of the two regular random walk–based methods, RW and RRRW. In the third case, however, the number of messages produced by the flood exploded to at least 25 times the previous values.

The rest of the results are consistent with what was seen during the first round.

4.2.3 RG Graphs

In RG graphs (see Table 9), the RRRW could have used a higher TTL, as seen by their less-than-ideal success rates. In more than 40 percent of the cases under each search condition, the queries reached their 10 000 message limit. The time to each the TTL, however, remained low.

The rest of the results are consistent with what was seen during the first round.

Table 1. Erdos-Renyi graph, instance 1. The graph was generated using N=2000 nodes and link probability p=0.005. The resulting graph had 10 170 links. The TTL of flooding was 5 and all other searches had a TTL of 10 000.

	Total Time (steps)		Total Messages		Number of Nodes Visited		Success Rate (%)
	AVG	(STDV)	AVG	(STDV)	AVG	(STDV)	
Search Conditions 1							
Flooding	4	0	9946	0	1988	0	100
RW	1586	1463	1586	1463	828	501	100
RRRW	54	19	1379	1281	764	462	100
Bidirectional RW	49	28	99	55	86	48	100
Bidirectional RRRW	19	7	104	60	90	51	100
Bidirectional Linear	-	-	-	-	-	-	0
Search Conditions 2							
Flooding	3	0	874	0	703	0	100
RW	1898	1474	1898	1474	950	503	100
RRRW	61	20	2089	1830	978	498	100
Bidirectional RW	45	27	91	54	79	45	100
Bidirectional RRRW	18	6	90	56	79	48	100
Bidirectional Linear	-	-	-	-	-	-	0
Search Conditions 3							
Flooding	4	0	9745	0	1983	0	100
RW	2904	2638	2904	2638	1117	573	98
RRRW	72	33	3576	3172	1191	635	93
Bidirectional RW	44	30	88	60	78	52	100
Bidirectional RRRW	18	7	101	66	90	57	100
Bidirectional Linear	-	-	-	-	-	-	0

Table 2. Erdos-Renyi graph, instance 2. The graph was generated using N=2000 nodes and link probability p=0.005. The resulting graph had 9 912 links. The TTL of flooding was 5 and all other searches had a TTL of 10 000.

	Total Time		Total Messages		Number of Nodes Visited		Success Rate (%)
	AVG	(STDV)	AVG	(STDV)	AVG	(STDV)	
Search Conditions 1							
Flooding	4	0	8945	0	1986	0	100
RW	2684	2454	2684	2454	1075	582	97
RRRW	72	29	3317	2694	1229	536	95
Bidirectional RW	50	24	100	48	87	42	100
Bidirectional RRRW	19	8	102	73	89	61	100
Bidirectional Linear	-	-	-	-	-	-	0
Search Conditions 2							
Flooding	4	0	5547	0	1870	0	100
RW	2862	2445	2862	2445	1131	574	97
RRRW	67	27	2650	2422	1075	549	97
Bidirectional RW	49	23	97	45	85	40	100
Bidirectional RRRW	19	7	112	71	97	58	100
Bidirectional Linear	-	-	-	-	-	-	0
Search Conditions 3							
Flooding	4	0	8796	0	1980	0	100
RW	2859	2587	2859	2587	1101	610	98
RRRW	65	31	2939	2748	1100	594	98
Bidirectional RW	31	22	61	44	54	37	100
Bidirectional RRRW	21	7	113	57	101	47	100
Bidirectional Linear	-	-	-	-	-	-	0

Table 3. Barabasi-Albert graph, instance 1. The graph was generated using N=2000 nodes, 2 initial nodes, and 5 links created per step. The resulting graph had 9 985 links. The TTL of flooding was 4 and all other searches had a TTL of 10 000.

	Total Time		Total Messages		Number of Nodes Visited		Success Rate (%)
	AVG	(STDV)	AVG	(STDV)	AVG	(STDV)	AVG
Search Conditions 1							
Flooding	3	0	3025	0	1469	0	100
RW	3539	3063	3539	3063	1122	590	93
RRRW	68	29	2970	2800	1004	594	97
Bidirectional RW	30	19	60	37	53	32	100
Bidirectional RRRW	16	6	74	39	65	33	100
Bidirectional Linear	50	36	79	57	8	2	100
Search Conditions 2							
Flooding	4	0	13558	0	2000	0	100
RW	1119	1081	1119	1081	591	429	100
RRRW	52	16	1255	1171	645	401	100
Bidirectional RW	33	17	66	33	57	29	100
Bidirectional RRRW	16	6	69	40	60	33	100
Bidirectional Linear	83	73	131	114	7	2	100
Search Conditions 3							
Flooding	4	0	12700	0	2000	0	100
RW	3958	3270	3958	3270	1215	517	88
RRRW	91	45	4965	3578	1330	571	80
Bidirectional RW	29	17	58	34	52	30	100
Bidirectional RRRW	18	6	77	43	68	35	100
Bidirectional Linear	-	-	-	-	-	-	0

Table 4. Barabasi-Albert graph, instance 2. The graph was generated using N=2000 nodes, 2 initial nodes, and 5 links created per step. The resulting graph had 9 985 links. The TTL of flooding was 4 and all other searches had a TTL of 10 000.

	Total Time		Total Messages		Number of Nodes Visited		Success Rate (%)
	AVG	(STDV)	AVG	(STDV)	AVG	(STDV)	AVG
Search Conditions 1							
Flooding	3	0	7077	0	1931	0	100
RW	3742	3261	3742	3261	1143	588	90
RRRW	80	36	4146	3235	1232	557	90
Bidirectional RW	28	15	56	29	48	25	100
Bidirectional RRRW	16	7	74	47	64	39	100
Bidirectional Linear	52	42	80	65	7	2	100
Search Conditions 2							
Flooding	3	0	4102	0	1684	0	100
RW	3093	2784	3093	2784	1073	524	92
RRRW	72	34	3320	2854	1096	576	94
Bidirectional RW	31	19	62	38	54	32	100
Bidirectional RRRW	14	6	63	39	55	33	100
Bidirectional Linear	-	-	-	-	-	-	0
Search Conditions 3							
Flooding	3	0	3446	0	1561	0	100
RW	4334	3300	4334	3300	1262	591	88
RRRW	72	28	3286	3065	1083	535	95
Bidirectional RW	28	19	56	39	50	34	100
Bidirectional RRRW	15	6	68	39	61	34	100
Bidirectional Linear	-	-	-	-	-	-	0

Table 5. Random geometric graph, instance 1. The graph was generated using N=2000 nodes and geographic link probability p=0.04. The resulting graph had 9 587 links. The TTL of flooding was 40 and all other searches had a TTL of 10 000.

	Total Time		Total Messages		Number of Nodes Visited		Success Rate (%)
	AVG	(STDV)	AVG	(STDV)	AVG	(STDV)	AVG
Search Conditions 1							
Flooding	17	0	6834	0	1047	0	100
RW	5965	3460	5965	3460	1021	433	74
RRRW	160	24	9857	663	576	106	6
Bidirectional RW	994	711	1988	1422	543	285	100
Bidirectional RRRW	69	20	6513	4520	619	199	99
Bidirectional Linear	99	76	190	145	42	16	100
Search Conditions 2							
Flooding	23	0	8367	0	1257	0	100
RW	5993	3453	5993	3453	1007	424	71
RRRW	162	21	10000	0	466	86	0
Bidirectional RW	977	777	1954	1554	546	289	100
Bidirectional RRRW	81	29	9097	5103	745	198	94
Bidirectional Linear	115	119	222	227	47	14	100
Search Conditions 3							
Flooding	29	0	10991	0	1628	0	100
RW	7425	3010	7425	3010	1144	317	58
RRRW	164	16	10000	0	415	76	0
Bidirectional RW	432	595	864	1189	253	283	100
Bidirectional RRRW	29	25	979	1849	132	149	100
Bidirectional Linear	81	46	158	88	51	16	100

Table 6. Random geometric graph, instance 2. The graph was generated using N=2000 nodes and geographic link probability p=0.04. The resulting graph had 9 725 links. The TTL of flooding was 40 and all other searches had a TTL of 10 000.

	Total Time		Total Messages		Number of Nodes Visited		Success Rate (%)
	AVG	(STDV)	AVG	(STDV)	AVG	(STDV)	AVG
Search Conditions 1							
Flooding	7	0	1880	0	318	0	100
RW	3442	3438	3442	3438	667	518	90
RRRW	82	37	4063	3288	375	176	88
Bidirectional RW	421	542	842	1083	284	268	100
Bidirectional RRRW	37	12	894	929	212	114	100
Bidirectional Linear	42	29	81	56	38	14	100
Search Conditions 2							
Flooding	28	0	12520	0	1800	0	100
RW	7133	3356	7133	3356	1175	374	52
RRRW	162	22	10000	0	503	104	0
Bidirectional RW	941	539	1883	1079	582	221	100
Bidirectional RRRW	138	43	17262	4115	853	137	53
Bidirectional Linear	-	-	-	-	-	-	0
Search Conditions 3							
Flooding	22	0	5690	0	860	0	100
RW	4535	3239	4535	3239	843	420	88
RRRW	164	19	10000	0	399	88	0
Bidirectional RW	482	810	965	1619	260	321	100
Bidirectional RRRW	24	18	441	815	99	99	100
Bidirectional Linear	101	78	190	145	34	16	100

Table 7. Erdos-Renyi graph, instance 1. The graph was generated using N=2000 nodes and link probability p=0.015. The resulting graph had 29 900 links. The TTL of flooding was 5 and all other searches had a TTL of 10 000.

	Total Time (steps)		Total Messages		Number of Nodes Visited		Success Rate (%)
	AVG	(STDV)	AVG	(STDV)	AVG	(STDV)	
Search Conditions 1							
Flooding	3	0	26171	0	2000	0	100
RW	1901	1750	1901	1750	969	561	100
RRRW	59	22	2021	2329	922	603	100
Bidirectional RW	41	19	81	37	78	35	100
Bidirectional RRRW	16	7	88	59	83	55	100
Bidirectional Linear	-	-	-	-	-	-	13
Search Conditions 2							
Flooding	3	0	21103	0	2000	0	100
RW	1411	1230	1411	1230	844	473	100
RRRW	57	18	1814	1677	979	449	100
Bidirectional RW	45	25	89	50	85	47	100
Bidirectional RRRW	18	7	79	39	76	37	100
Bidirectional Linear	-	-	-	-	-	-	10
Search Conditions 3							
Flooding	3	0	21922	0	2000	0	100
RW	3386	2738	3386	2738	1254	661	98
RRRW	63	28	2468	2458	1061	615	100
Bidirectional RW	35	21	70	41	67	38	100
Bidirectional RRRW	17	8	77	57	74	54	100
Bidirectional Linear	-	-	-	-	-	-	10

Table 8. Barabasi-Albert graph, instance 1. The graph was generated using N=2000 nodes, 2 initial nodes, and 15 links created per step. The resulting graph had 29 880 links. The TTL of flooding was 4 and all other searches had a TTL of 10 000.

	Total Time		Total Messages		Number of Nodes Visited		Success Rate (%)
	AVG	(STDV)	AVG	(STDV)	AVG	(STDV)	AVG
Search Conditions 1							
Flooding	3	0	35876	0	2000	0	100
RW	3640	2710	3640	2710	1272	477	95
RRRW	79	36	3682	3409	1161	604	88
Bidirectional RW	32	18	63	36	60	34	100
Bidirectional RRRW	16	6	65	33	62	30	100
Bidirectional Linear	22	13	38	22	12	5	100
Search Conditions 2							
Flooding	2	0	1415	0	925	0	100
RW	2447	2153	2447	2153	1031	494	98
RRRW	66	27	2846	2550	1080	549	98
Bidirectional RW	27	12	54	23	52	22	100
Bidirectional RRRW	16	6	67	40	64	36	100
Bidirectional Linear	30	20	51	33	11	3	100
Search Conditions 3							
Flooding	2	0	957	0	718	0	100
RW	1356	1140	1356	1140	752	422	100
RRRW	61	26	1680	1628	815	485	100
Bidirectional RW	28	18	55	36	52	34	100
Bidirectional RRRW	16	9	70	52	66	47	100
Bidirectional Linear	26	18	44	31	11	4	100

Table 9. Random geometric graph, instance 1. The graph was generated using N=2000 nodes and geographic link probability p=0.07. The resulting graph had 29 006 links. The TTL of flooding was 40 and all other searches had a TTL of 10 000.

	Total Time		Total Messages		Number of Nodes Visited		Success Rate (%)
	AVG	(STDV)	AVG	(STDV)	AVG	(STDV)	AVG
Search Conditions 1							
Flooding	9	0	22375	0	1156	0	100
RW	2201	2004	2201	2004	871	510	100
RRRW	113	51	6461	3791	960	355	58
Bidirectional RW	167	153	333	305	230	188	100
Bidirectional RRRW	29	18	705	1106	276	279	100
Bidirectional Linear	53	30	102	58	39	10	100
Search Conditions 2							
Flooding	9	0	27348	0	1419	0	100
RW	2620	2102	2620	2102	999	540	100
RRRW	115	48	6997	3539	1075	354	60
Bidirectional RW	147	173	293	345	197	184	100
Bidirectional RRRW	34	16	646	892	303	261	100
Bidirectional Linear	43	23	84	44	42	10	100
Search Conditions 3							
Flooding	12	0	37709	0	1842	0	100
RW	4083	3487	4083	3487	1164	592	88
RRRW	132	46	8432	2709	1243	281	35
Bidirectional RW	127	132	255	265	182	160	100
Bidirectional RRRW	28	18	509	674	238	242	100
Bidirectional Linear	41	24	81	47	52	15	100

5 Conclusions

In this paper, an investigation of the performance of bidirectional search algorithms was presented. Their effectiveness in various graph topologies under varying search conditions was considered. Based on the data collected during numerous simulations, it appears that bidirectional searches have the potential to offer a significantly lower combination of time and messages generated as compared to standard flooding- and random walk–based strategies. Their performance and rate of success are highly dependent on graph topology and search conditions.

BDRW performed consistently well in all three graph topologies tested, showing perfect success rates, lower completion time than basic RW, and often an acceptably low message count. BDRRRW showed the lowest search times overall, but had some problems in RG graphs that lead to high message counts and lower success rates. Very impressive. Finally, BDL showed the most variable performance; it refused to work in ER graphs and caused problems once in the other two graphs. It has the highest potential to benefit RG graphs, due to their inherent geographic nature, and I am confident the algorithm can be tweaked to show even better performance.

Future work could involve running simulations on larger networks (e.g. in the neighbourhood of 10 000 nodes or more), considering the impact of multiple targets on search performance, performing additional simulations on different network topologies and under varying search conditions, etc. There is *always* more that can be done!

References

[1] J. D. Noh and H. Rieger, "Random walks on complex networks," Physical Review Letters, vol. 92, no. 11, Mar. 2004.

[2] M. Zhong, K. Shen and J. Seiferas, "The convergence-guaranteed random walk and its applications in peer-to-peer networks," Computers, IEEE Transactions on, vol. 57, no. 5, May 2008, pp. 619–633.

[3] R. Gaeta and M. Sereno, "On the evaluation of flooding-based search strategies in peer-to-peer networks," Concurrency and Computation: Practice and Experience, vol. 20, no. 6, Apr. 2008, pp. 713–734.

[4] S. Dhillon and P. Van Mieghem, "Searching with multiple random walk queries," in 18th Ann. IEEE Int. Symp. Personal, Indoor and Mobile Radio Communications (PIMRC'07), Athens, Sep. 2007, pp. 1–5.

[5] R. Dorrigiv, A. Lopez-Ortiz and P. Pralat, "Search algorithms for unstructured peer-to-peer networks," in Local Computer Networks, 2007. LCN 2007. 32nd IEEE Conf. on, Oct. 15–18, 2007, pp. 343–352.

[6] D. Kogias, K. Oikonomou and I. Stavrakakis, "Study of randomly replicated random walks for information dissemination over various network topologies," in Wireless On-Demand Network Systems and Services, 2009. WONS 2009. 6th Int. Conf., Feb. 2–4, 2009, pp. 53–60.

[7] C. Gkantsidis, M. Mihail and A. Saberi, "Hybrid search schemes for unstructured peer-to-peer networks," in INFOCOM 2005. 24th Ann. Joint Conf. IEEE Computer and Communications Societies. Proc. IEEE, vol. 3, Mar. 13–17, 2005, pp. 1526–1537.

[8] C. Gkantsidis, M. Mihail and A. Saberi, "Random walks in peer-to-peer networks," in INFOCOM 2004. 23rd Ann. Joint Conf. IEEE Computer and Communications Societies, vol. 1, Mar. 7–11, 2004.

[9] D. Tsoumakos and N. Roussopoulos, "Analysis and comparison of P2P search methods," in Proc. 1st Int. Conf. Scalable Information Systems, Hong Kong, May 30–Jun. 1, 2006.

[10] Z. Cheng and W. B. Heinzelman, "Flooding strategy for target discovery in wireless networks," Wireless Networks, vol. 11, no. 5, Sep. 2005, pp. 607–618.

[11] A. S. Maiya and T. Y. Berger-Wolf, "Expansion and search in networks," in Proc. 19th ACM Int. Conf. Information and Knowledge Management, Toronto, Ont., 2010, pp. 239–248.

[12] V. Kalogeraki, D. Gunopulos and D. Zeinalipour-Yazti, "A local search mechanism for peer-to-peer networks," In Proc. of the 11th Int. Conf. on Information and Knowledge Management, 2002, pp. 300–307.

[13] S. S. Dhillon, Ant Routing, Searching and Topology Estimation Algorithms for Ad Hoc Networks, Amsterdam, Netherlands: Ios Press, 2008.

[14] Z. J. Haas, J. Y. Halpern and L. Li, "Gossip-based ad hoc routing," in INFOCOM 2002. 21st Ann. Joint Conf. IEEE Computer and Communications Societies. Proc. IEEE, vol. 3, 2002, pp. 1707–1716.

[15] M. Zaharia and S. Keshav, "Gossip-based search selection in hybrid peer-to-peer networks," Concurrency and Computation: Practice and Experience, vol. 20, no. 2, May 2007, pp. 139–153.

[16] R. Rakesh and S. Sujata, "Reducing redundancy of random walk search on grid topology for unstructured P2P network," International Journal of Computer Applications, vol. 34, no. 5, 2011, pp. 34–36.

[17] L. A. Adamic et al., "Search in power-law networks," Physical Review E, vol. 64, no. 4, Sep. 2001.