Assignment 2 - Maximum Weighted Matching

108287 - Miguel Belchior Figueiredo

***Abstract* – This report presents a comparative analysis of two algorithms for solving the maximum weighted matching problem: a purely randomized algorithm and another that guides the randomization process by applying a probability distribution that biases the selection of options based on a particular heuristic. Both algorithms are examined through design, formal analysis, and experimental evaluation.**

I. Introduction

The maximum weighted matching problem is an optimization problem in the field of graph theory, with a broad range of practical applications in various fields, such as biotechnology, social analysis, etc. Formally, the problem tries to find for a given undirected graph $G(V, E)$, with n vertices and m edges the maximum weighted matching. A matching in G is a set of pairwise non-adjacent edges, i.e., no two edges share a common vertex. A maximum weighted matching is a matching for which the sum of the weights of its edges is as large as possible. In some graph instances, there may be multiple maximum weighted matchings, provided their total weight is the same.

The last report focused on solving the max weighted matching problem using two distinct approaches: an exhaustive algorithm and a greedy heuristic approach. The exhaustive method, while guaranteeing an optimal solution was found to be significantly inefficient due to its exponential time complexity of $O(m \cdot 2^m)$, with m being the number of edges. The latter, while not guaranteeing an optional solution, proved to be a more scalable alternative for larger problem instances. This report shifts focus to the design, implementation and analysis of randomized algorithms to address the maximum weighted matching problem. By incorporating randomization, these algorithms aim to strike a balance between solution precision and computational efficiency. This report presents two distinct algorithms: Randomized Max Weighted Matching and a Probabilistic Greedy approach.

II. Randomized Max Weighted Matching

A. Design

As the name suggests, the Randomized Max Weighted Matching Algorithm randomly generates different matchings from the initial set of edges. As each matching is generated, it is evaluated and kept if it corresponds to the best solution, i.e., it has the highest total weight of all encountered matchings. The algorithm's behavior can be expressed through the following pseudocode:

Algorithm 1 Random Max Weighted Matching

```
Input: Graph G(n, m), max_iter=10000, time_limit=None
Output: best_matching, max_weight

start_time ← get_current_time()
best_matching ← empty set
shuffled_edges ← empty set
edges ← G.edges
total_weight ← sum(G.edges weights)
max_weight ← 0

for i = 0 to max_iter - 1 do
    if time_limit was surpassed then
        break

    random.shuffle(edges)
    if edges shuffling repeated then
        continue
    shuffled_edges.add(edges)

    current_matching ← empty set
    matched_vertices ← empty set
    current_weight ← 0
    remaining_weight ← total_weight
    for each edge (u, v, w) in edges do
        remaining_weight -= w
        if current_weight + remaining_weight
            < max_weight then
            break
        if u not in matched_vertices and
            v not in matched_vertices then
            matched_vertices.add(u)
            matched_vertices.add(v)
            current_matching.add(edge)
            current_weight += w

    if current_weight > max_weight then
        max_weight ← current_weight
        best_matching ← current_matching

return best_matching, max_weight
```

The algorithm takes as input the graph for which the maximum weighted matching is to be found, along with the maximum number of iterations (max_iter) and a specified time limit (time_limit).

A.1 Initialization

Firstly, the algorithm initializes multiple structures and variables with the following purposes:

- `start_time` - Registers the algorithm's starting time, allowing it to be stopped if a specified time limit is exceeded. Together with `max_iter` is responsible for limiting the number of iterations of the main flow and, consequently, the number of candidate solutions that are generated and tested.
- `best_matching` - An empty set that will hold the best matching, which is the matching with the highest weight.
- `shuffled_edges` - Keeps track of all permutations of shuffled edges to avoid reevaluating repeated matchings.
- `edges` - The list of edges in the graph, which will be shuffled in each iteration.
- `total_weight` - The sum of the weights of all edges in the graph.
- `max_weight` - Stores the current maximum weight, initialized to zero.

A.2 Edge Shuffling

Then, for each iteration, as long as the time limit has not been exceeded, the algorithm shuffles the list of the graph's edges using `random.shuffle` function [1], on line 12. This approach was chosen over `combinations` function [2] from itertools as the latter would require additional validation to ensure that the subset of edges formed a matching, which would fail in most of the cases. Moreover, using combinations would require specifying the subset size explicitly. That would need to involve some additional logic to obtain this size or to randomize it. Randomizing the subset size would often result in subsets that possibly contain potencial supersets with higher weights. In contrast, `random.shuffle` avoids these challenges, offering a simpler and more effective approach, which will be explored next. It is also important to note that to ensure no matchings are tested more than once, the different permutations of edges are tracked in lines 13 to 15.

A.3 Building Matching Greedily

For each shuffled edge list, the program iterates through its edges and greedily forms a matching - in the sense that it keeps adding the next edge as long as the constraints are satisfied, i.e., the new edge doesn't share a vertex with any of the edges in the matching (lines 24 to 28). This process continues until either all edges have been processed or it is determined that the current solution cannot surpass the best-known solution with the highest weight. This is verified in line 22, where the condition checks if the combined weight of the current solution and the remaining edges cannot surpass the current maximum weight. If that's the case, the algorithm should abandon the current matching and proceed to the next iteration, where a new matching will be attempted.

A.4 Solution Evaluation

As previously mentioned, multiple iterations are performed as to explore multiple regions of the solution space and to increase the changes of finding a solution closer to the optimal solution. In each iteration, a new matching is constructed, and its weight is compared to the current maximum weight. If the weight of the new matching surpasses the current maximum, the maximum weight and the best matching are updated accordingly (lines 29 to 31).

B. Formal Analysis

As previously mentioned, the algorithm begins by initializing variables and data structures. This includes calculating the total weight of the graph's edges, which has a best, worst, and average case complexity of $O(m)$, where `m` is the number of edges.

Then, the algorithm runs a randomized process for `max_iter` iterations to maximize precision in relation to the optimal solution. During each iteration, the following main steps are executed:

1. **Time Limit Check** - Determine if the algorithm has exceeded the time limit, which only involves a few arithmetic operations and comparisons - $O(1)$;
2. **Edge Shuffling** - Randomly shuffle the list of edges in each iteration. This is accomplished by using python's `random.shuffle` function [1], which implements the Fisher-Yates shuffle algorithm [3] that has a time complexity of $O(m)$, with m being the number of edges;
3. **Verify Repeated Edge Shuffle** - This step consists of checking whether that edge permutation is already present in the set and, if not, adding it. Both operations are performed with a time complexity of $O(1)$;
4. **Building Matching Greedily** - For each shuffled edge list, the function iterates through the edges to create a matching. In the worst case, this requires examining all m edges, leading to a time complexity of $O(m)$;
5. **Solution Evaluation** - Evaluate whether the current matching has the highest weight found so far and update if so - $O(1)$.

Therefore, the time complexity of the algorithm's

is dominated by the complexity of its inner loop, resulting in $O(max\_iter \cdot m)$, where m is the number of edges. The parameter `max_iter` serves as a scaling factor, meaning that increasing its value will directly proportionally increase the running time. Since `max_iter` is a fixed number, the algorithm's complexity is linear with respect to `m`. Thus, the algorithm's performance depends on the number of edges rather than the number of nodes which is only relevant in terms of the maximum number of edges it permits.

C. Experimental Analysis

To validate the results of the formal analysys, the algorithm was applied to a range of various graphs including:

1) Randomly Generated Graphs containing varying numbers of vertices for four different edge densities. The graphs included in the experiments had the following properties:

- successively larger random graphs with $n \in [4, 154]$, with a step of 25 and n being the number of vertices in the graph. Although the algorithm could run for a larger number of vertices and edges this upper limit was decided for the experiments to minimize computation time and avoid prolonged wait periods;
- the number of edges of the generated graph corresponds to $\lfloor n \cdot edge\_density \rfloor$, with $edge\_density \in \{0.125, 0.25, 0.5, 0.75\}$ and n being the number of vertices. Only generated graphs with a positive non null number of edges were considered;
- graph vertices are 2D points on the XOY plane, with randomly generated integer valued coordinates between 1 and 1000;
- each edge was assigned a random weight ranging from 1 to 100;
- the number of edges sharing a vertex is randomly determined. However, that is only done after ensuring the connectivity of every single node, i.e., all nodes should have at least one edge, which is also attributed randomly;
- A fixed randomization seed was used to ensure reproducibility of the graph generation process - 108287.

2) SW's folder graphs, available on the course's e-learning platform, were also utilized, provided they were weighted graphs. It is important to note that these weighted graphs were also directed. However, since the algorithms reviewed do not account for directed edges, the graphs were treated as undirected, with forward and backward edges ignored - these will result in a decrease of the number of edges of the graph. These graphs will be analyzed following the analysis of the first set of randomly generated graphs.

The following metrics were registered for analysis:

- **Total Number of solutions/configurations** representing the total number of times the initial list of edges was shuffled;
- **Number of filtered solutions/configurations tested**, referring to the number of times a matching was built and evaluated;
- **Number of basic operations**, defined as the count of operations performed to check whether the nodes of the next edge in the list of edges already appeared in any of the edges of the matching;
- **Execution Time**;
- **Precision** indicating the ratio between the weight of the maximum weighted matching identified by the algorithm and the weight of an actual maximum weighted matching returned by `networkX` built-in function `max_weight_matching` [4].

The metrics, along with their corresponding values, are recorded on Table I.

TABLE I
PERFORMANCE METRICS BY NODES AND EDGE DENSITY OF RANDOMIZED MAX WEIGHTED MATCHING ALGORITHM

# Nodes	Edge Density	# Edges	#All Solut.	#Solut. Tested	#Basic Oper.	Exec. Time(s)	Precision
4	0.25	1	10000	1	2	0.002	1
4	0.5	3	10000	6	23	0.006	1
4	0.75	4	10000	24	112	0.007	1
29	0.125	50	10000	10000	648468	0.136	1
29	0.25	101	10000	10000	1209797	0.252	0.864
29	0.5	203	10000	10000	2332154	0.481	0.861
29	0.75	304	10000	10000	3404007	0.704	0.786
54	0.125	178	10000	10000	2204580	0.439	0.871
54	0.25	357	10000	10000	4190587	0.849	0.776
54	0.5	715	10000	10000	7938547	1.716	0.723
54	0.75	1073	10000	10000	11573270	2.604	0.741
79	0.125	385	10000	10000	4661626	1.156	0.752
79	0.25	770	10000	10000	8767935	1.905	0.721
79	0.5	1540	10000	10000	16657006	3.833	0.699
79	0.75	2310	10000	10000	24493167	5.665	0.699
104	0.125	669	10000	10000	7922618	1.779	0.679
104	0.25	1339	10000	10000	14951732	3.409	0.680
104	0.5	2678	10000	10000	28653518	6.706	0.664
104	0.75	4017	10000	10000	42171738	9.955	0.653
129	0.125	1032	10000	10000	11959465	2.635	0.677
129	0.25	2064	10000	10000	22681629	5.197	0.673
129	0.5	4128	10000	10000	43726022	10.136	0.667
129	0.75	6192	10000	10000	64570848	15.407	0.656
154	0.125	1472	10000	10000	16858047	3.918	0.673
154	0.25	2945	10000	10000	32001996	8.267	0.641
154	0.5	5890	10000	10000	61951523	16.327	0.656
154	0.75	8835	10000	10000	91636412	23.697	0.640

C.1 Number of Configurations Tested

The number of solutions tested over the number of edges is plotted in Figure 1 - graphs with the same number of nodes but varying edge densities are represented using the same color, a convention maintained throughout this report. It is evident that the number of solutions tested is almost always 10000 which is the value of the `max_iter` variable. The only exceptions occur with 4 nodes and 1, 3, or 4 edges, where 1, 6, and 24 different solutions were tested, respectively. These are the only cases where registering and verifying against the edge permutations yields meaningful results, as the greater the number of edges, the lower the

likelihood that shuffling them will result in the same order. As shown in Table I, the number of solutions tested and the total number of solutions remain constant and equal to 10000 for graphs with 29 nodes or more.

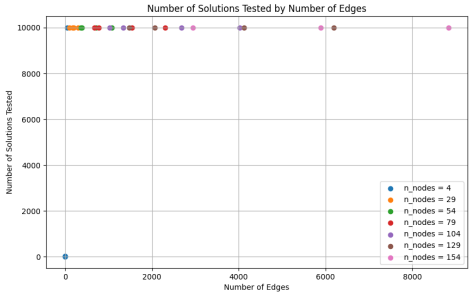

Fig. 1 - Number of Solutions Tested over the Number of Edges for the Random Max Weighted Matching Algorithm

C.2 Number Of Basic Operations

The number of basic operations by the number of edges is plotted in Figure 2. The chart also includes a linear fit, illustrating the relationship between the number of edges and the total number of basic operations performed by the algorithm.

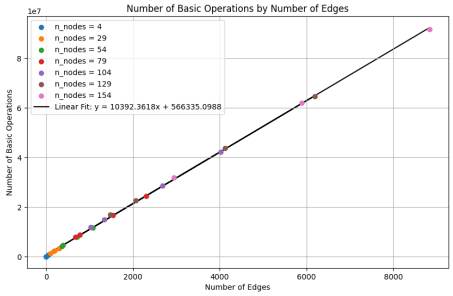

Fig. 2 - Number of Basic Operations over the Number of Edges for the Random Max Weighted Matching Algorithm

As shown in Figure 2, the number of basic operations grows linearly with the number of edges. It is also possible to conclude that the number of nodes is only relevant in terms of the maximum number of edges it permits, with the number of configurations depending solely on the number of edges of the graph. The plotted linear fit function, also corresponds to the expectations, with the slope of the line approximately equal to 10392.3618, which closely matches the value of the `max_iter` variable.

C.3 Execution Time

The execution time results over the number of edges are plotted in Figure 3. It also possesses a linear fit, which demonstrates a clear correlation between the number of edges and the execution time.

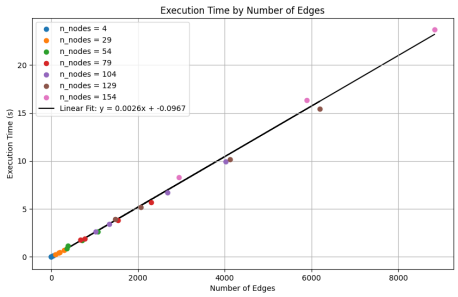

Fig. 3 - Execution Time (s) over the Number of Edges for the Random Max Weighted Matching Algorithm

As the number of edges increases the execution time grows - execution time depends linearly on the number of edges, with the number of nodes only influencing the maximum number of edges that can exist in the graph. This linear dependency supports the linear time complexity identified on the Formal Analysis section: $O(max\_iter \cdot m)$, where m is the number of edges. It is now clear that the algorithm cannot solve this problem for large problem instances within a reasonable amount of time. The experimental computer was unable to solve the maximum weighted matching problem using this approach for 504 nodes with an edge density of 0.75 (graph with 95067 edges), without taking too much time - under 5 minutes. Therefore, it is only possible to make an estimation for the execution time of the algorithm for larger problem instances based on the previously executed problems. By using the linear fit from the execution time results present on Figure 3, (though the projections may not be highly accurate due to the limited number of samples) we can project the algorithm's performance for larger graphs as following:

- For a graph with 500000 edges, it would take approximately 1299.90 seconds which is the equivalent to approximately 21.67 minutes.
- For a graph with 500000000 edges, it would take approximately 1299999.90 seconds which is the equivalent to approximately 361.11 hours or 15.04 days.
- For a graph with 500000000000 edges, it would take approximately 1299999999.90 seconds which is the equivalent to approximately 15046.30 days or 41.22 years.

C.4 Precision

Since the algorithm being analyzed is purely randomized, it was not expected to consistently produce the optimal solution. The precision of the results, relative to an optimal solution for the problem returned by `max_weight_matching` function from `networkX`, is shown in Figure 4, as well as in the last column of Table I.

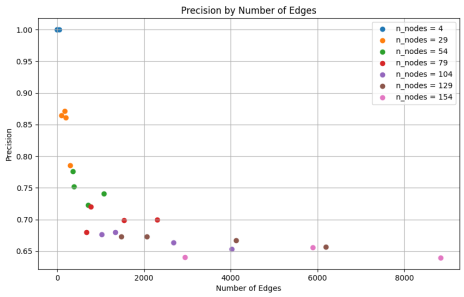

Fig. 4 - Precision over the Number of Edges for the Random Max Weighted Matching Algorithm

An analysis of the results reveals that the precision of the algorithm falls drastically as the number of edges increases. While this trend generally holds, there are instances where graphs with fewer edges exhibit lower accuracy. Such variations are expected, given the randomized nature of the algorithm. Either way, this algorithm does not demonstrate high accuracy and can only be improved by increasing further the number of iterations (`max_iter`), which is not recommended as it would extend the already long execution time to an even more prohibitive duration.

C.5 SW Graphs

This section analyzes the same metrics as the previous sections, focusing on graphs from the SW folder available on the course's e-learning platform. As noted earlier, although these weighted graphs are directed, they were treated as undirected for the analysis, which resulted in a reduced number of edges. Table II presents the different metrics for four different graphs.

TABLE II
PERFORMANCE METRICS OF RANDOMIZED MAX WEIGHTED MATCHING ALGORITHM FOR THE SW GRAPHS

Name	# Nodes	# Edges	#All Solut.	#Solut. Tested	#Basic Oper.	Exec. Time(s)	Precision
tinyEWD	8	13	10000	10000	166715	0.043	1
mediumEWD	250	1273	10000	10000	15541831	3.589	0.757
1000EWD	1000	8433	10000	10000	97880284	26.598	0.708
10000EWD	10000	61731	10000	10000	743175719	361.122	0.690

The analysis of Table II reaffirms the conclusions drawn in the previous sections:

- The number of total and filtered solutions remains the same, further demonstrating that tracking edge permutations is generally unnecessary except for very small graph instances (for the tinyEWD graph, which is the smallest graph, there are a total of 13! possible permutations).
- Both the number of basic operations and the execution time are greater for the graphs with a larger number of edges.
- Precision decreases significantly for graphs with a larger number of edges, starting at 1 on the tinyEWD graph and dropping to 0.690 on the 10000EWD graph.

III. PROBABILISTIC GREEDY

A. Design

Following the observed results of the experimental analysis of the first algorithm it is clear that, while it is indeed a straightforward approach to use randomization in solving the problem, it does not yield the best results nor the best performance. Specially as the number of edges of the graph increases and the randomization becomes more and more erratic and less predictable.

That being the case, in this section, a probabilistic greedy algorithm will be used to construct the final solution. This approach takes inspiration from Mohamed Haouari and Jouhaina Chaouachi Siala work [5] on a probabilistic greedy search algorithm for combinatorial optimisation with application to the set covering problem. The key idea drawn from their paper is the application of a probability distribution that biases/favors the selection of options based on a particular criterion/heuristic, thereby introducing an element of randomization while still guiding the search process effectively. When applied to the maximum weighted matching problem, this idea translated into assigning higher probabilities to edges with greater weights, thereby prioritizing them in the search. Therefore, the probability of each edge `e` with weight w_e being chosen abides by the following formula:

$$P(e) = \frac{w_e^{100}}{\sum_{i=1}^{m} w_i^{100}} \tag{1}$$

A polynomial function with an exponent of 100 was chosen to amplify the influence of higher weights compared to using a simple weight-to-sum ratio as this was deemed insufficient to guide the search process (this will be revisited in a later section of the report). The higher the exponent, the more pronounced the bias toward heavier weights, making the function more closely approximate to the behavior of the correspondant heuristic greedy approach. The algorithm's behavior can be expressed through the pseudocode marked as Algorithm 2.

Algorithm 2 Probabilistic Greedy

```
Input: Graph G(n, m), max_iter=1000, time_limit=2
Output: best_matching, max_weight

start_time ← get_current_time()
best_matching ← empty set
edges ← G.edges
num_edges ← len(edges)
random_indices ← empty set
total_weight ← sum(G.edges weights)
max_weight ← 0

probabilities ← compute probabilities according to P(e) formula

for i = 0 to max_iter - 1 do
    if time_limit was surpassed then
        break

    random_edge_indices ← choice(
            num_edges, size=num_edges,
            p=probabilities, replace=False)
    if random_edge_indices repeated then
        continue
    random_indices.add(random_edge_indices)

    current_matching ← empty set
    matched_vertices ← empty set
    current_weight ← 0
    remaining_weight ← total_weight
    for edge_indice in random_edge_incides do
        edge ← edges[edge_indice]
        u, v, w ← edge.unpack()
        remaining_weight -= w
        if current_weight + remaining_weight
            < max_weight then
            break
        if u not in matched_vertices and
            v not in matched_vertices then
            matched_vertices.add(u)
            matched_vertices.add(v)
            current_matching.add(edge)
            current_weight += w

    if current_weight > max_weight then
        max_weight ← current_weight
        best_matching ← current_matching

return best_matching, max_weight
```

The initialization of data structures and variables is similar to the one on the first algorithm with the addition of:

- **Pre-computing the probabilities for each edge** according to the pre-defined formula, as these remain constant accross iterations, as well as the number of edges `num_edges`.
- `random_indices` **initialization** - maintains a record of all permutations of edge indices to prevent reevaluation of matchings.

The main flow of the algorithm is also similar to the previous one differing in the following aspects:

- **Random Edge Selection** - Instead of performing a random shuffle of the edges, a list of random edge indices is generated randomly by using `np.random.choice` [6] function, according to the predefined probability distribution. Bear in mind the use of parameters `p` and `replace=False` to specify the probabilities and to ensure indices are not repeated, respectively. Since the function returns indices the corresponding edge is retrieved from the initial list of edges during the most inner loop.
- **Tracking Repeated Candidate Solutions** - A set is used to track permutations of indices to avoid re-evaluating the same matching. Unlike the previous version, this set now stores indices rather than the edges themselves.

In summary, the algorithm begins by assigning probabilities to edges, favoring those with higher weights as per the formula $P(e)$. Then, the algorithm iteratively selects random edge orderings, ensuring no vertex is matched twice and builds the matching greedily. This process continues until either all edges have been processed or it is determined that the current solution cannot surpass the best-known solution with the highest weight. If that's the case, the algorithm should abandon the current matching and proceed to the next iteration, where a new matching will be attempted. New matchings are tested iteratively until the algorithm reaches the maximum iteration count or the time limit, continually updating the maximum weight and best matching whenever a higher-weight solution is found.

B. Formal Analysis

To conduct a formal analysis, the algorithm can be divided in two sequential phases. It starts by initializing variables and data structures, where it is highlighted the **calculation of the graph's edges total weight** and the **calculation of the edge's probabilities**, both of which with time complexities of $O(m)$, where `m` is the number of edges.

Then, similarly to the first algorithm, the main inner loop is initiated, which is executed for `max_iter` iterations. During each iteration, the following primary steps are performed:

1. **Time Limit Check** - Determine if the al-

gorithm has exceeded the time limit, which only involves a few arithmetic operations and comparisons - $O(1)$;

2. **Edge Shuffling** - Randomly obtain a list of edge indices in each iteration. This is accomplished by using numpy's `np.random.choice` function [6]. After conducting some research, the complexity of the function with both parameters `p` (to specify probability) and `replace=False` (to ensure no indices are repeated) for a size n of indices could not be determined. Therefore, the complexity will be referred to as $O(q)$, with q representing an unknown function;
3. **Verify Repeated Edge Shuffle** - This step consists of checking whether that edge index permutation is already present in the set and, if not, adding it. Both operations are performed with a time complexity of $O(1)$;
4. **Building Matching Greedily** - For each shuffled edge index, the function iterates through the edges to create a matching. In the worst case, this requires examining all `m` edges, leading to a time complexity of $O(m)$;
5. **Solution Evaluation** - Evaluate whether the current matching has the highest weight found so far and update if so - $O(1)$.

By combining the complexities of the multiple phases, the overall complexity of the final algorithm is determined, where m represents the number of edges in the graph:

$$T(prob\_greedy) = O(max\_iter \cdot max(q, m)) \quad (2)$$

C. Experimental Analysis

The experimental analysis for this algorithm follows a similar framework to that of the random max weighted matching with the following differences in the first set of graphs:

- successively larger random graphs with $n \in [4, 1000]$, with a step of 25 and n being the number of vertices in the graph.

The metrics collected for analysis also remain consistent with those used in the previous algorithm:

- **Total Number of solutions/configurations**
- **Number of filtered solutions/configurations tested**
- **Number of basic operations**
- **Execution Time**
- **Precision**

C.1 Number of Configurations Tested

The number of solutions tested over the number of edges is plotted in Figure 5. The behavior depicted in the graph appears to follow an exponential decay pattern: as the number of edges increases the number of solutions tested decreases sharply before leveling off and approaching a lower asymptote. This behavior is due to the execution time limit of 2 seconds of the algorithm, which is checked at the start of each iteration before a new solution/matching is built and evaluated. As a result, graphs with fewer edges can complete multiple iterations and evaluate a greater number of solutions, while graphs with more edges require more time to process each solution, ultimately reducing the total number of solutions evaluated within the time limit. Figure 6 shows the same relationship for the same algorithm without the time limit, which resembles that of the randomized algorithm discussed in the first section, thereby corroborating this hypothesis. The outliers that escape this pattern correspond to the graph with 4 nodes as its limited number of edges doesn't allow for many different solutions to be tested, as these don't exist.

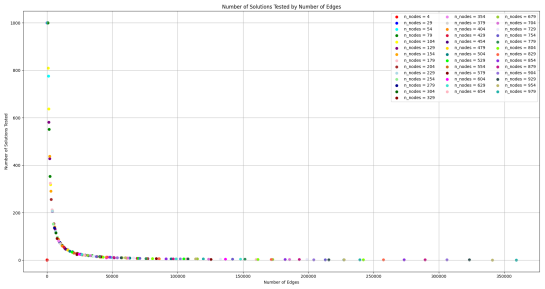

Fig. 5 - Number of Solutions Tested over the Number of Edges for the Probabilistic Greedy Algorithm

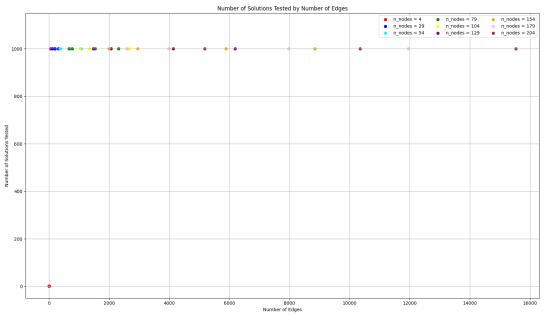

Fig. 6 - Number of Solutions Tested over the Number of Edges for the Probabilistic Greedy Algorithm without the Time Limit

Moreover, the total and filtered number of solutions tested should also be compared. In Table IV (in appendix), these metrics match perfectly in almost all cases, with the only exception being the graphs with 4 nodes. Consequently, tracking the permutations of edge indices is not particularly worthwhile, as the likelihood of generating the same permutation decreases significantly as the number of edges increases.

C.2 Number of Basic Operations

The Figure 7 represents the relationship between the number of edges and the number of basic operations, which are equivalent to the number of matching membership checks. Initially, the number of basic operations increases with the increase of the number of edges, as evident in the cases of graphs with 4, 29, 54, and 79 nodes. Another intriguing insight from this graph is the noticeable clustering of graphs with varying numbers of nodes, where the growth in the number of basic operations correlates linearly with the increase in the number of edges. However, this linear relationship holds only up to a certain threshold, after which the number of basic operations decreases briefly before resuming another linear growth pattern - the next linear pattern also appears to always have a lower slope. This cyclical behavior suggests distinct phases in the computational complexity as the graph's structure evolves and can perhaps have something to do with the maximum runtime threshold of 2 seconds imposed on the algorithm.

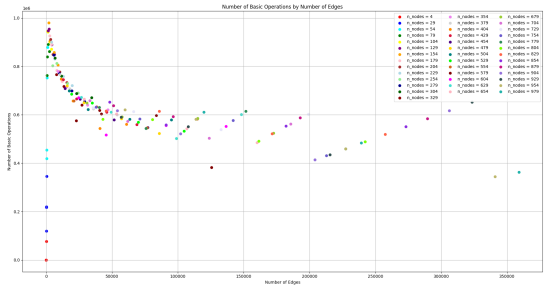

Fig. 7 - Number of Basic Operations over the Number of Edges for the Probabilistic Greedy Algorithm

Since Figure 7 raised more questions than it provided answers, a second chart of the number of basic operations over the number of edges was plotted in Figure 8 without the upper time limit threshold of 2 seconds for the algorithm.

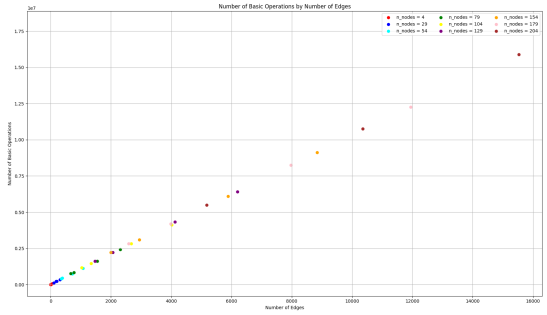

Fig. 8 - Number of Basic Operations over the Number of Edges for the Probabilistic Greedy Algorithm without the Time Limit

Figure 8 exhibits a linear dependency between the number of edges and the number of basic operations, thereby supporting the findings of the Formal Analysis and proving that the previous behavior of Figure 7 was due to the imposed time limit.

C.3 Execution Time

The Figure 9 illustrates the execution time over the number of edges. In this figure, it is possible to note that the execution time clearly grows as the number of edges grow, up to the two-second threshold imposed on the algorithm. However, for higher edge counts, the execution time exceeds this limit. This occurs because the threshold is checked at the start of each iteration, before constructing and evaluating the next matching or solution. To avoid this behavior, the threshold check could be performed before parsing each individual edge when determining whether or not it belongs to the matching.

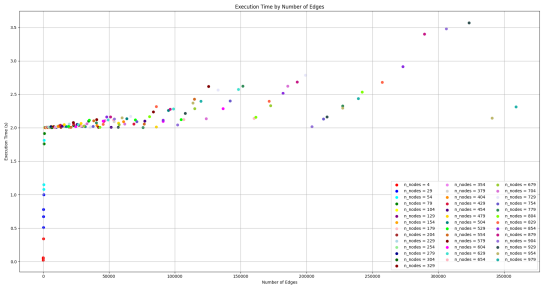

Fig. 9 - Execution Time over the Number of Edges for the Probabilistic Greedy Algorithm

In order to ease comparison with the results of the Formal Analysis, Figure 10 plots the execution time against the number of edges, without the time limit.

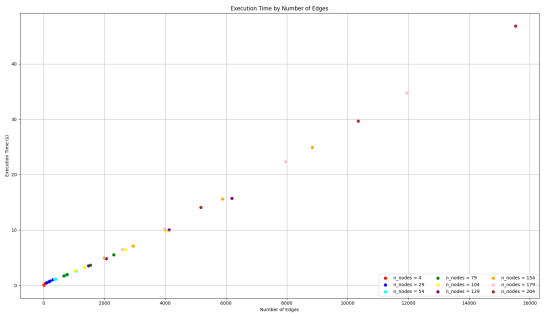

Fig. 10 - Execution Time over the Number of Edges for the Probabilistic Greedy Algorithm without the Time Limit

The complexity derived in the Formal Analysis was the following:

$$T(prob\_greedy) = O(max\_iter \cdot max(q, m)) \quad (3)$$

Since the value of `q` correspondent to the randomization of edge indices with the `np.random.choice` function, is unknown, a precise complexity could not be determined.

Therefore, it is more challenging to fully verify the validity of the formal analysis. Figure 10 appears to display an "approximately" linear relationship (Quasilinear - $O(m \cdot \log m)$) between the execution time and the number of edges, denoted by a slope differentiation at the beginning of the chart. This conclusion, neither corroborates or denies the complexity as it's not fully determined. However, being right it would suggest a complexity of $O(m \cdot \log m)$ for the undetermined value of `q`.

C.4 Precision

As previously mentioned, it was not expected for this algorithm to consistently produce the optimal solution due to its inherent randomness. However, since the probabilistic greedy algorithm guides the randomization process by giving higher probabilities to edges with higher weights this randomness can be tamed, producing encouraging results. The last column of Table IV (on appendix) shows the algorithms precision for each of the graph used in the experiments. Figure 11 displays the relationship between the number of edges and the precision of the graphs.

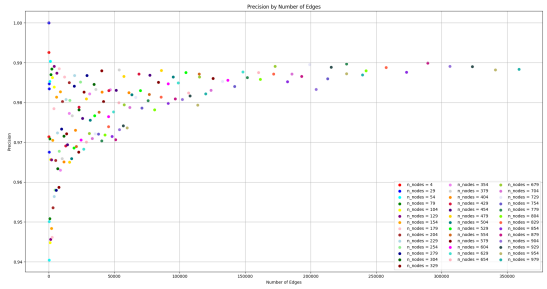

Fig. 11 - Precision over the Number of Edges for the Probabilistic Greedy Algorithm

The data presented in Figure 11 shows a higher spread of precision values for smaller edge counts, which is in part due to the algorithm's inherent randomness. However, as the number of edges increases precision also improves stabilizing between 0.98 and 0.99 for the larger graph instances.

C.5 SW Graphs

Relatively to the SW graphs used to test the randomized algorithm in the first section of this report there is something to note. As the probabilistic greedy algorithm pre-computes the probability for each edge using the below formula, which contains exponents, using edge weights smaller than 1 would lead to unintended behavior - this is the case for the weights of all of these graphs' edges.

$$P(e) = \frac{w_e^{100}}{\sum_{i=1}^{m} w_i^{100}} \tag{4}$$

Therefore, a normalization of the edge weights is required to ensure that the algorithm works as intended. This is achieved by multiplying each edge's weight by a scalar, guaranteeing that all edge weights exceed one. This adjustment is explicitly implemented and asserted in the code. Table III presents the resulting metrics for the SW graphs.

TABLE III
PERFORMANCE METRICS OF THE PROBABILISTIC GREEDY ALGORITHM FOR THE SW GRAPHS

Name	# Nodes	# Edges	#All Solut.	#Solut. Tested	#Basic Oper.	Exec. Time(s)	Precision
tinyEWD	8	13	1000	22	379	0.133	1
mediumEWD	250	1273	789	789	1217295	2.006	0.949
1000EWD	1000	8433	116	116	1142331	2.008	0.934
10000EWD	10000	61731	12	12	896611	2.021	0.922

On the one hand, the analysis of Table II reaffirms the conclusions drawn in the previous sections:

- For most graphs, the total number of solutions matches the number of solutions actually tested, with the exception of very small graph instances that have a very small number of edges (tinyEWD). This highlights that tracking permutations of edge indices to avoid evaluating a matching more than once is generally unnecessary, as such repetitions rarely occur.
- The execution times reflect the two-second threshold imposed on the algorithm, which is reached for all graphs except tinyEWD.

On the other hand, the values of precision, although high, deviate from the expected precision values that were observed on Figure 11 for graphs with a similar order of edge magnitude. Therefore, future work would include conducting a further analysis to identify potential factors influencing these deviations.

D. Choosing the Exponent for Edge Selection Probabilities

The strength of this algorithm lies in its ability to incorporate a randomized component while keeping it well-controlled and guided, preventing it from becoming chaotic. This is achieved by assigning higher probabilities to edges with greater weights, ensuring that the randomization is directed towards more optimal choices using the following formula:

$$P(e) = \frac{w_e^{100}}{\sum_{i=1}^{m} w_i^{100}} \tag{5}$$

The formula is straightforward and intuitive, as it reflects the ratio between the weight of an edge and the total weight of all edges in the graph. However, the rationale behind using an exponent and the choice of its value require further explanation and a more in-depth analysis, which will

be discussed in this section. The evaluations presented in the following subsections include experiments of the same algorithm, with variations of the exponent of the probability formula between 1, 10, and 100. When the exponent is 1, the formula is equivalent to the simple ratio of the weight of an edge over the total weight of all edges in the graph.

D.1 Precision

The primary deciding factor for selecting the exponent was the precision of the solutions produced by the algorithm.

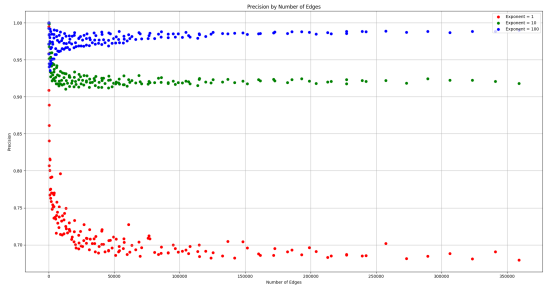

Fig. 12 - Precision over the Number of Edges for the Probabilistic Greedy Algorithm with different exponents

Figure 12 illustrates the differences in precision distributions across the various algorithm variations with exponents 1, 10 and 100:

- **1** - Shows a huge discrepancy between the precision for smaller and larger graph instances. The precision tends to decrease as the number of edges increases stabilizing at around 0.67 - values obtained through inspection of the results in Table VI;
- **10** - Similarly to the previous exponent the precision tends to decrease as the number of edges increases stabilizing at around 0.92 - values obtained through inspection of the results in Table VIII. The gap of precisions between smaller and larger graph instances has narrowed;
- **100** - This distribution exhibits a wide range of precision values across different graphs for smaller instances. As the number of edges increases the precision value stabilizes at around 0.98 - values obtained through inspection of the results in Table X. This algorithm emerges as the top performer, achieving the highest mean precision. However, it appears to present a wider range of precision values for smaller graph instances than the one with exponent 10, behaving worse than it in some cases. Therefore, a balance should be sought between exponents 10 and 100 to identify the value that achieves the highest average precision while maintaining strong performance on smaller graph instances.

The mean precision values of 0.727, 0.928 and 0.978 for exponents 1, 10 and 100 also show the difference in precision with the exponent 100 achieving the highest precision. Moreover, the probabilistic greedy with exponents 10 and 100 have a mean precision similar to the value of precision it tends to stabilize on indicating that smaller graph instances aren't positively biasing the result (as it happens for exponent 1).

D.2 Execution Time

Figure 13 plots the execution time over the number of edges for the multiple algorithm's instances. All of the instances, display a similar distribution where the execution time is capped at a threshold of 2 seconds. The execution times over that threshold are, as previously explained, due to where this limit is enforced: checked at the start of each iteration, before constructing and evaluating the next matching or solution.

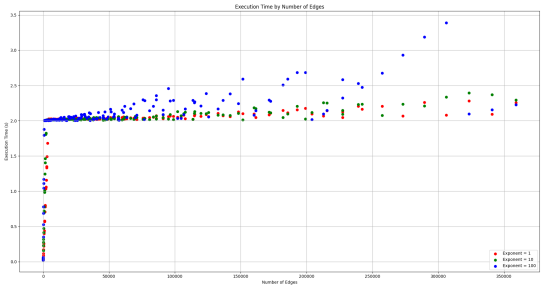

Fig. 13 - Execution Time over the Number of Edges for the Probabilistic Greedy Algorithm with different exponents

It's also possible to conclude that the larger the exponent the faster the algorithm reaches this threshold - it achieves this threshold at smaller graph instances with smaller number of edges. These results are expected and can be attributed to the increased computational time required for larger mathematical operations as the edge weights grow significantly with increasing exponents. The observed behavior aligns with the data presented in the execution times of Tables VI, VIII and X on appendix, highlighting the impact on performance for larger numerical values.

D.3 Number of Configurations Tested

Figure 14 illustrates the differences in the relationship between the number of edges and the number of solutions tested for each of the varying exponents. While all of them exhibit a similar distribution, the number of solutions tested is consistently higher for the exponent 1, followed by 10, and 100. These results are expected and can be attributed to the increased computational time required for larger mathematical operations, which was already discussed.

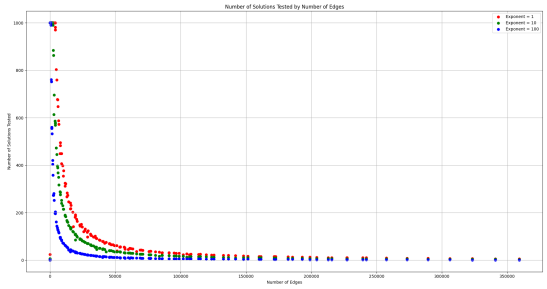

Fig. 14 - Number of Solutions Tested over the Number of Edges for the Probabilistic Greedy Algorithm with different exponents

D.4 Number of Basic Operations

Figure 15 shows the relationship between the number of edges and the number of basic operations for each of the varying exponents. All of the algorithms show a similar distribution. However, the behavior previously seen in Figure 14 can also be seen here with larger exponents exhibiting smaller number of basic operations.

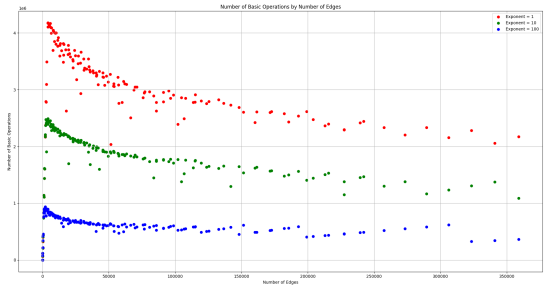

Fig. 15 - Number of Basic Operations over the Number of Edges for the Probabilistic Greedy Algorithm with different exponents

IV. Other Algorithms not covered in the report

In the attempt of improving the results obtained from the previous algorithms, a third algorithm was developed. This algorithm divided the initial set of edges in multiple subsets, randomly selected one and applied the greedy heuristic approach discussed in the last report, and then proceeded to the next subset until all were processed. However, this algorithm did not outperform the probabilistic greedy approach, with precision values exhibiting significant variability. This was likely due to the algorithm's heavy reliance on the random division of edges into subsets, which introduced significant inconsistency. As a result, no analysis of this algorithm was conducted in this report.

V. Conclusion

This report, presented a comparative analysis of two algorithms designed to solve the maximum weighted matching problem: the Randomized Max Weighted Matching algorithm and a Probabilistic Greedy approach. Both approaches were reviewed in terms of their time complexity, with both design, formal and experimental analysis being performed. The Randomized Max Weighted Matching algorithm produced not so accurate results, with the deviation between the expected and actual values increasing as the number of edges in the graph instances grew, which could be attributed to the algorithm's inherent randomization component. Moreover, the execution time of the algorithm grows linearly with the number of edges which makes the computations more and more expensive for larger graph instances. However, these results pale in comparison with the ones from the probabilistic greedy algorithm which, bounded by an upper limit of 2 seconds, could consistently deliver high precision solutions for larger graph instances in a fraction of the time. These could be achieved by assigning higher edge probabilities to edges with greater weights, ensuring that the randomization is directed towards more optimal choices.

References

[1] Python Software Foundation, "random.py — python standard library", https://svn.python.org/projects/python/trunk/Lib/random.py, 2024, [Online; accessed 2-December-2024].

[2] Python Software Foundation, "itertools — functions creating iterators for efficient looping", https://docs.python.org/3/library/itertools.html#itertools.combinations, 2024, [Online; accessed 2-December-2024].

[3] Wikipedia contributors, "Fisher–yates shuffle — Wikipedia, the free encyclopedia", https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle#The_modern_algorithm, 2023, [Online; accessed 2-December-2024].

[4] NetworkX Developers, "max_weight_matching — networkx documentation", https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.matching.max_weight_matching.html, 2024, [Online; accessed 2-December-2024].

[5] Mohamed Haouari and Jouhaina Chaouachi Siala, "A probabilistic greedy search algorithm for combinatorial optimisation with application to the set covering problem", https://www.researchgate.net/publication/245280982_A_probabilistic_greedy_search_algorithm_for_combinatorial_optimisation_with_application_to_the_set_covering_problem, 2002, [Online; accessed 2-December-2024].

[6] NumPy Developers, "numpy.random.choice — numpy documentation", https://numpy.org/doc/2.0/reference/random/generated/numpy.random.choice.html, 2024, [Online; accessed 4-December-2024].

VI. Appendix A - Probab. Greedy Results

TABLE IV
Performance Metrics by Nodes and Edge Density of Probabilistic Greedy Algorithm - Part 1

# Nodes	Edge Density	# Edges	#All Solut.	#Solut. Tested	#Basic Oper.	Exec. Time(s)	Precision
4	0.25	1	1000	1	2	0.024	1.000
4	0.5	3	1000	2	8	0.049	0.971
4	0.75	4	1000	1	6	0.058	1.000
29	0.125	50	1000	1000	76471	0.341	0.993
29	0.25	101	1000	1000	119940	0.509	1.000
29	0.5	203	1000	1000	219430	0.780	0.983
29	0.75	304	1000	1000	345053	1.000	0.985
54	0.125	178	1000	1000	217704	0.672	0.968
54	0.25	357	1000	1000	418586	1.078	0.950
54	0.5	715	1000	1000	752664	1.813	0.985
54	0.75	1073	776	776	878247	2.007	0.990
79	0.125	385	1000	1000	454326	1.151	0.940
79	0.25	770	1000	1000	839362	1.915	0.971
79	0.5	1540	552	552	890441	2.005	0.987
79	0.75	2310	354	354	861949	2.008	0.988
104	0.125	669	1000	1000	762333	1.759	0.951
104	0.25	1339	638	638	941531	2.006	0.966
104	0.5	2678	319	319	901537	2.007	0.986
104	0.75	4017	210	210	871355	2.005	0.984
129	0.125	1032	810	810	945663	2.006	0.945
129	0.25	2064	428	428	954033	2.008	0.966
129	0.5	4128	206	206	889428	2.007	0.989
129	0.75	6192	132	132	848907	2.005	0.987
154	0.125	1472	581	581	947325	2.005	0.946
154	0.25	2945	291	291	905737	2.006	0.971
154	0.5	5890	140	140	857301	2.023	0.981
154	0.75	8835	88	88	805373	2.004	0.983
179	0.125	1991	438	438	979441	2.005	0.948
179	0.25	3982	213	213	891043	2.006	0.978
179	0.5	7965	95	95	782894	2.012	0.988
179	0.75	11948	60	60	738408	2.026	0.986
204	0.125	2588	325	325	925858	2.006	0.946
204	0.25	5176	154	154	848615	2.004	0.965
204	0.5	10353	72	72	776816	2.012	0.980
204	0.75	15529	46	46	732873	2.026	0.985
229	0.125	3263	256	256	911440	2.009	0.954
229	0.25	6526	121	121	830230	2.009	0.972
229	0.5	13053	56	56	758463	2.038	0.981
229	0.75	19579	36	36	721227	2.055	0.987
254	0.125	4016	207	207	900452	2.011	0.956
254	0.25	8032	96	96	812981	2.023	0.968
254	0.5	16065	44	44	728958	2.021	0.981
254	0.75	24098	28	28	690256	2.030	0.985
279	0.125	4847	153	153	802263	2.008	0.958
279	0.25	9695	76	76	772617	2.010	0.973
279	0.5	19390	35	35	699104	2.014	0.984
279	0.75	29085	22	22	654382	2.020	0.987
304	0.125	5757	136	136	846582	2.017	0.958
304	0.25	11514	63	63	759675	2.026	0.972
304	0.5	23028	29	29	688073	2.035	0.978
304	0.75	34542	19	19	670416	2.103	0.985
329	0.125	6744	115	115	834045	2.005	0.963
329	0.25	13489	51	51	716560	2.007	0.972
329	0.5	26978	23	23	639564	2.038	0.983
329	0.75	40467	15	15	618311	2.125	0.988
354	0.125	7810	92	92	765825	2.015	0.959
354	0.25	15620	44	44	716038	2.036	0.977
354	0.5	31240	20	20	640686	2.029	0.982
354	0.75	46860	13	13	620820	2.112	0.983
379	0.125	8953	82	82	779748	2.008	0.963
379	0.25	17907	38	38	706917	2.020	0.977
379	0.5	35815	17	17	624369	2.015	0.983
379	0.75	53723	11	11	600052	2.091	0.988
404	0.125	10175	71	71	765463	2.018	0.966
404	0.25	20351	31	31	656636	2.009	0.973
404	0.5	40703	13	13	543090	2.035	0.983
404	0.75	61054	9	9	560955	2.094	0.983
429	0.125	11475	61	61	747842	2.023	0.965
429	0.25	22951	28	28	665669	2.053	0.979
429	0.5	45903	13	13	610950	2.099	0.983
429	0.75	68854	8	8	559006	2.055	0.987
454	0.125	12853	55	55	744863	2.041	0.969
454	0.25	25707	25	25	665100	2.055	0.976
454	0.5	51415	11	11	578544	2.008	0.983
454	0.75	77123	7	7	547778	2.100	0.987
479	0.125	14310	47	47	708432	2.027	0.969
479	0.25	28620	22	22	649403	2.066	0.981
479	0.5	57240	10	10	583660	2.071	0.987
479	0.75	85860	6	6	522253	2.014	0.988
504	0.125	15844	43	43	720097	2.048	0.965
504	0.25	31689	19	19	622316	2.016	0.976
504	0.5	63378	9	9	581659	2.136	0.982
504	0.75	95067	6	6	578671	2.261	0.986
529	0.125	17457	38	38	698338	2.018	0.966
529	0.25	34914	18	18	648784	2.115	0.977
529	0.5	69828	8	8	569464	2.120	0.983
529	0.75	104742	5	5	532192	2.125	0.988
554	0.125	19147	34	34	685835	2.021	0.969
554	0.25	38295	16	16	633157	2.105	0.978
554	0.5	76590	7	7	546269	2.056	0.982
554	0.75	114885	5	5	582137	2.430	0.987
579	0.125	20916	30	30	657295	2.009	0.969
579	0.25	41832	14	14	602856	2.010	0.980
579	0.5	83665	7	7	596984	2.240	0.985
579	0.75	125498	3	3	381840	2.619	0.986

TABLE V
Performance Metrics by Nodes and Edge Density of Probabilistic Greedy Algorithm - Part 2

# Nodes	Edge Density	# Edges	#All Solut.	#Solut. Tested	#Basic Oper.	Exec. Time(s)	Precision
604	0.125	22763	24	24	575392	2.079	0.968
604	0.25	45526	11	11	516514	2.101	0.977
604	0.5	91053	6	6	555943	2.095	0.985
604	0.75	136579	4	4	552585	2.288	0.986
629	0.125	24688	26	26	670699	2.016	0.971
629	0.25	49376	12	12	611206	2.117	0.978
629	0.5	98753	5	5	502083	2.280	0.985
629	0.75	148129	4	4	600768	2.574	0.988
654	0.125	26691	24	24	671683	2.050	0.968
654	0.25	53382	11	11	603201	2.119	0.980
654	0.5	106765	5	5	542370	2.123	0.982
679	0.125	28772	22	22	664609	2.026	0.970
679	0.25	57545	10	10	591522	2.050	0.979
679	0.5	115090	5	5	585415	2.286	0.986
679	0.75	172635	3	3	523771	2.332	0.989
704	0.125	30932	20	20	648289	2.027	0.972
704	0.25	61864	9	9	572820	2.058	0.979
704	0.5	123728	4	4	503396	2.136	0.983
704	0.75	185592	3	3	562470	2.623	0.987
729	0.125	33169	19	19	658264	2.061	0.971
729	0.25	66339	9	9	613817	2.171	0.981
729	0.5	132678	4	4	538242	2.566	0.985
729	0.75	199017	3	3	602689	2.786	0.990
754	0.125	35485	17	17	629681	2.009	0.972
754	0.25	70970	8	8	582309	2.091	0.979
754	0.5	141940	4	4	576741	2.400	0.984
754	0.75	212910	2	2	430890	2.132	0.986
779	0.125	37878	16	16	631244	2.021	0.972
779	0.25	75757	7	7	543697	2.006	0.980
779	0.5	151515	4	4	614963	2.622	0.986
779	0.75	227273	2	2	459073	2.325	0.990
804	0.125	40350	15	15	630240	2.075	0.970
804	0.25	80701	7	7	579692	2.168	0.978
804	0.5	161403	3	3	490820	2.159	0.986
804	0.75	242104	2	2	488951	2.534	0.988
829	0.125	42900	13	13	581700	2.009	0.972
829	0.25	85801	7	7	614241	2.316	0.981
829	0.5	171603	3	3	521690	2.400	0.987
829	0.75	257404	2	2	519455	2.682	0.989
854	0.125	45528	13	13	615504	2.055	0.974
854	0.25	91057	6	6	558433	2.112	0.980
854	0.5	182115	3	3	553112	2.518	0.985
854	0.75	273173	2	2	551399	2.915	0.988
879	0.125	48235	13	13	652755	2.162	0.972
879	0.25	96470	6	6	591929	2.277	0.981
879	0.5	192940	3	3	586981	2.686	0.987
879	0.75	289410	2	2	584083	3.402	0.990
904	0.125	51019	12	12	637295	2.164	0.971
904	0.25	102039	5	5	522114	2.043	0.981
904	0.5	204078	2	2	413756	2.018	0.983
904	0.75	306117	2	2	617323	3.483	0.989
929	0.125	53882	11	11	616475	2.131	0.973
929	0.25	107764	5	5	550816	2.217	0.982
929	0.5	215528	2	2	435353	2.163	0.989
929	0.75	323292	2	2	652025	3.568	0.989
954	0.125	56822	10	10	590095	2.008	0.974
954	0.25	113645	5	5	581804	2.371	0.979
954	0.5	227290	2	2	459729	2.298	0.987
954	0.75	340935	1	1	343746	2.146	0.988
979	0.125	59841	10	10	620374	2.150	0.974
979	0.25	119682	5	5	610930	2.398	0.982
979	0.5	239365	2	2	484013	2.436	0.987
979	0.75	359048	1	1	362403	2.314	0.988

VII. Appendix B - Probab. Greedy Results - Exponent 1

TABLE VI
Performance Metrics by Nodes and Edge Density of Probabilistic Greedy Algorithm with exponent 1 - Part 1

# Nodes	Edge Density	# Edges	#All Solut.	#Solut. Tested	#Basic Oper.	Exec. Time(s)	Precision
4	0.25	1	1000	1	2	0.026	1.000
4	0.5	3	1000	6	24	0.037	1.000
4	0.75	4	1000	24	112	0.038	1.000
29	0.125	50	1000	1000	61396	0.088	0.995
29	0.25	101	1000	1000	112905	0.118	0.958
29	0.5	203	1000	1000	217785	0.169	0.889
29	0.75	304	1000	1000	323127	0.221	0.861
54	0.125	178	1000	1000	208444	0.171	0.909
54	0.25	357	1000	1000	399891	0.254	0.841
54	0.5	715	1000	1000	757148	0.420	0.801
54	0.75	1073	1000	1000	1106810	0.578	0.815
79	0.125	385	1000	1000	440339	0.257	0.807
79	0.25	770	1000	1000	833959	0.440	0.816
79	0.5	1540	1000	1000	1602011	0.801	0.791
79	0.75	2310	1000	1000	2376324	1.157	0.792
104	0.125	669	1000	1000	750507	0.399	0.767
104	0.25	1339	1000	1000	1437886	0.709	0.763
104	0.5	2678	1000	1000	2783706	1.352	0.770
104	0.75	4017	1000	1000	4107381	2.006	0.768
129	0.125	1032	1000	1000	1140755	0.568	0.774
129	0.25	2064	1000	1000	2187431	1.060	0.770
129	0.5	4128	982	982	4173480	2.024	0.770
129	0.75	6192	648	648	4096898	2.025	0.758
154	0.125	1472	1000	1000	1607569	0.777	0.775
154	0.25	2945	1000	1000	3096019	1.493	0.754
154	0.5	5890	676	676	4104546	2.026	0.740
154	0.75	8835	449	449	4014914	2.024	0.796
179	0.125	1991	1000	1000	2164758	1.036	0.759
179	0.25	3982	1000	1000	4172021	2.017	0.752
179	0.5	7965	496	496	4047329	2.025	0.751
179	0.75	11948	322	322	3917720	2.025	0.742
204	0.125	2588	1000	1000	2792229	1.335	0.748
204	0.25	5176	760	760	4125207	2.025	0.735
204	0.5	10353	377	377	4001588	2.024	0.739
204	0.75	15529	241	241	3811717	2.026	0.730
229	0.125	3263	1000	1000	3490582	1.681	0.751
229	0.25	6526	588	588	3995572	2.027	0.744
229	0.5	13053	284	284	3784778	2.024	0.749
229	0.75	19579	186	186	3704207	2.033	0.733
254	0.125	4016	971	971	4177136	2.025	0.736
254	0.25	8032	483	483	4037325	2.023	0.738
254	0.5	16065	231	231	3799703	2.029	0.719
254	0.75	24098	151	151	3694307	2.026	0.718
279	0.125	4847	803	803	4140164	2.026	0.737
279	0.25	9695	398	398	3996368	2.023	0.732
279	0.5	19390	191	191	3784346	2.024	0.718
279	0.75	29085	124	124	3665219	2.035	0.709
304	0.125	5757	678	678	4166421	2.025	0.716
304	0.25	11514	325	325	3870084	2.026	0.734
304	0.5	23028	144	144	3384647	2.023	0.703
304	0.75	34542	101	101	3542683	2.030	0.708
329	0.125	6744	573	573	4089834	2.023	0.729
329	0.25	13489	274	274	3816764	2.030	0.722
329	0.5	26978	131	131	3596590	2.032	0.727
329	0.75	40467	79	79	3239807	2.028	0.719
354	0.125	7810	449	449	3696391	2.024	0.723
354	0.25	15620	216	216	3487192	2.029	0.721
354	0.5	31240	105	105	3342581	2.040	0.721
354	0.75	46860	65	65	3081342	2.025	0.706
379	0.125	8953	407	407	3851088	2.019	0.714
379	0.25	17907	142	142	2624061	2.027	0.711
379	0.5	35815	91	91	3316298	2.020	0.711
379	0.75	53723	57	57	3105550	2.036	0.698
404	0.125	10175	355	355	3805858	2.020	0.713
404	0.25	20351	173	173	3635533	2.023	0.700
404	0.5	40703	79	79	3262060	2.023	0.720
404	0.75	61054	51	51	3148924	2.056	0.727
429	0.125	11475	312	312	3769937	2.026	0.715
429	0.25	22951	148	148	3506585	2.020	0.694
429	0.5	45903	69	69	3217878	2.044	0.706
429	0.75	68854	43	43	2994646	2.040	0.704
454	0.125	12853	267	267	3593585	2.027	0.725
454	0.25	25707	120	120	3176023	2.037	0.698
454	0.5	51415	39	39	2036746	2.038	0.708
454	0.75	77123	38	38	2966163	2.036	0.708
479	0.125	14310	246	246	3687718	2.024	0.716
479	0.25	28620	118	118	3469326	2.031	0.709
479	0.5	57240	53	53	3082375	2.030	0.708
479	0.75	85860	32	32	2777400	2.033	0.697
504	0.125	15844	218	218	3610638	2.019	0.720
504	0.25	31689	104	104	3380280	2.032	0.708
504	0.5	63378	48	48	3095736	2.029	0.699
504	0.75	95067	31	31	2979965	2.058	0.700
529	0.125	17457	203	203	3704837	2.023	0.708
529	0.25	34914	95	95	3407636	2.021	0.702
529	0.5	69828	43	43	3049917	2.029	0.684
529	0.75	104742	27	27	2861974	2.053	0.693
554	0.125	19147	181	181	3621563	2.031	0.704
554	0.25	38295	84	84	3300125	2.035	0.698
554	0.5	76590	38	38	2944630	2.038	0.712
554	0.75	114885	24	24	2782109	2.074	0.700
579	0.125	20916	164	164	3580425	2.034	0.696
579	0.25	41832	72	72	3089386	2.025	0.701
579	0.5	83665	34	34	2891081	2.053	0.687
579	0.75	125498	22	22	2789909	2.061	0.689

TABLE VII
Performance Metrics by Nodes and Edge Density of Probabilistic Greedy Algorithm with exponent 1 - Part 2

# Nodes	Edge Density	# Edges	#All Solut.	#Solut. Tested	#Basic Oper.	Exec. Time(s)	Precision
604	0.125	22763	149	149	3531473	2.023	0.713
604	0.25	45526	69	69	3225527	2.017	0.692
604	0.5	91053	32	32	2954758	2.046	0.690
604	0.75	136579	20	20	2760586	2.073	0.705
629	0.125	24688	140	140	3599467	2.028	0.699
629	0.25	49376	62	62	3134607	2.027	0.711
629	0.5	98753	29	29	2905492	2.065	0.690
629	0.75	148129	18	18	2688831	2.128	0.704
654	0.125	26691	125	125	3457533	2.023	0.706
654	0.25	53382	56	56	3064077	2.039	0.691
654	0.5	106765	23	23	2488103	2.082	0.688
654	0.75	160148	15	15	2423786	2.098	0.688
679	0.125	28772	98	98	2933648	2.029	0.702
679	0.25	57545	47	47	2763436	2.019	0.703
679	0.5	115090	25	25	2913119	2.089	0.684
679	0.75	172635	15	15	2613651	2.112	0.695
704	0.125	30932	110	110	3543303	2.024	0.693
704	0.25	61864	49	49	3103721	2.025	0.690
704	0.5	123728	22	22	2756533	2.090	0.682
704	0.75	185592	13	13	2433959	2.114	0.693
729	0.125	33169	98	98	3368115	2.024	0.702
729	0.25	66339	37	37	2508366	2.043	0.693
729	0.5	132678	21	21	2823768	2.107	0.685
729	0.75	199017	13	13	2612967	2.176	0.696
754	0.125	35485	91	91	3352177	2.027	0.691
754	0.25	70970	42	42	3046916	2.017	0.696
754	0.5	141940	19	19	2730836	2.060	0.682
754	0.75	212910	11	11	2362986	2.070	0.683
779	0.125	37878	85	85	3336597	2.039	0.689
779	0.25	75757	37	37	2858729	2.040	0.709
779	0.5	151515	17	17	2606912	2.102	0.696
779	0.75	227273	10	10	2293340	2.126	0.687
804	0.125	40350	79	79	3300379	2.022	0.701
804	0.25	80701	36	36	2964769	2.048	0.690
804	0.5	161403	16	16	2611722	2.048	0.686
804	0.75	242104	10	10	2444101	2.163	0.685
829	0.125	42900	75	75	3325670	2.026	0.690
829	0.25	85801	30	30	2625365	2.074	0.689
829	0.5	171603	15	15	2600673	2.083	0.686
829	0.75	257404	9	9	2335728	2.208	0.702
854	0.125	45528	70	70	3296698	2.020	0.693
854	0.25	91057	30	30	2788927	2.028	0.689
854	0.5	182115	14	14	2580204	2.147	0.691
854	0.75	273173	8	8	2203049	2.069	0.681
879	0.125	48235	65	65	3239105	2.036	0.696
879	0.25	96470	29	29	2848909	2.070	0.695
879	0.5	192940	13	13	2535883	2.157	0.687
879	0.75	289410	8	8	2335772	2.263	0.685
904	0.125	51019	62	62	3270886	2.046	0.686
904	0.25	102039	23	23	2389852	2.051	0.691
904	0.5	204078	12	12	2473319	2.095	0.691
904	0.75	306117	7	7	2158203	2.081	0.688
929	0.125	53882	55	55	3058939	2.048	0.692
929	0.25	107764	26	26	2849544	2.031	0.698
929	0.5	215528	11	11	2396934	2.144	0.685
929	0.75	323292	7	7	2281795	2.284	0.681
954	0.125	56822	55	55	3218987	2.053	0.695
954	0.25	113645	24	24	2776407	2.021	0.691
954	0.5	227290	10	10	2295283	2.049	0.686
954	0.75	340935	6	6	2055890	2.093	0.690
979	0.125	59841	45	45	2778793	2.045	0.687
979	0.25	119682	23	23	2800924	2.065	0.691
979	0.5	239365	10	10	2417576	2.206	0.685
979	0.75	359048	6	6	2173804	2.258	0.680

VIII. Appendix C - Probab. Greedy Results - Exponent 10

TABLE VIII
Performance Metrics by Nodes and Edge Density of Probabilistic Greedy Algorithm with exponent 10 - Part 1

# Nodes	Edge Density	# Edges	#All Solut.	#Solut. Tested	#Basic Oper.	Exec. Time(s)	Precision
4	0.25	1	1000	1	2	0.026	1.000
4	0.5	3	1000	2	8	0.043	0.971
4	0.75	4	1000	7	28	0.055	1.000
29	0.125	50	1000	1000	67922	0.156	0.999
29	0.25	101	1000	1000	119347	0.229	1.000
29	0.5	203	1000	1000	211559	0.319	1.000
29	0.75	304	1000	1000	330261	0.413	0.998
54	0.125	178	1000	1000	206312	0.276	0.972
54	0.25	357	1000	1000	406826	0.779	0.968
54	0.5	715	1000	1000	746011	0.720	0.973
54	0.75	1073	1000	1000	1109464	1.007	0.959
79	0.125	385	1000	1000	438141	0.471	0.952
79	0.25	770	1000	1000	829918	0.780	0.952
79	0.5	1540	1000	1000	1606154	1.404	0.954
79	0.75	2310	990	990	2372188	2.013	0.960
104	0.125	669	1000	1000	737228	0.697	0.943
104	0.25	1339	1000	1000	1437517	1.245	0.949
104	0.5	2678	863	863	2411591	2.014	0.945
104	0.75	4017	579	579	2389468	2.014	0.944
129	0.125	1032	1000	1000	1142090	0.985	0.934
129	0.25	2064	1000	1000	2206992	1.823	0.942
129	0.5	4128	569	569	2440495	2.015	0.938
129	0.75	6192	368	368	2343571	2.013	0.942
154	0.125	1472	1000	1000	1613410	1.462	0.927
154	0.25	2945	614	614	1908770	2.011	0.930
154	0.5	5890	389	389	2361347	2.014	0.940
154	0.75	8835	253	253	2289635	2.012	0.935
179	0.125	1991	1000	1000	2174210	1.806	0.933
179	0.25	3982	587	587	2458414	2.016	0.934
179	0.5	7965	286	286	2352841	2.019	0.928
179	0.75	11948	184	184	2251819	2.016	0.932
204	0.125	2588	885	885	2474272	2.014	0.922
204	0.25	5176	446	446	2428871	2.015	0.927
204	0.5	10353	215	215	2290692	2.019	0.932
204	0.75	15529	139	139	2204561	2.017	0.929
229	0.125	3263	696	696	2446740	2.015	0.924
229	0.25	6526	350	350	2391085	2.017	0.930
229	0.5	13053	163	163	2186547	2.014	0.931
229	0.75	19579	85	85	1696874	2.016	0.934
254	0.125	4016	577	577	2488096	2.014	0.925
254	0.25	8032	276	276	2320299	2.013	0.922
254	0.5	16065	135	135	2230431	2.022	0.926
254	0.75	24098	87	87	2132315	2.023	0.927
279	0.125	4847	473	473	2448213	2.012	0.922
279	0.25	9695	230	230	2329487	2.019	0.922
279	0.5	19390	110	110	2186925	2.026	0.927
279	0.75	29085	71	71	2107817	2.026	0.929
304	0.125	5757	397	397	2451695	2.016	0.918
304	0.25	11514	188	188	2250846	2.015	0.926
304	0.5	23028	90	90	2116385	2.015	0.931
304	0.75	34542	54	54	1899476	2.019	0.925
329	0.125	6744	317	317	2281951	2.014	0.918
329	0.25	13489	160	160	2239833	2.020	0.922
329	0.5	26978	75	75	2072887	2.020	0.929
329	0.75	40467	48	48	1976722	2.029	0.926
354	0.125	7810	288	288	2383816	2.015	0.915
354	0.25	15620	137	137	2213246	2.026	0.927
354	0.5	31240	65	65	2073507	2.035	0.924
354	0.75	46860	40	40	1905241	2.012	0.923
379	0.125	8953	240	240	2277177	2.013	0.918
379	0.25	17907	119	119	2207598	2.023	0.922
379	0.5	35815	46	46	1679957	2.025	0.928
379	0.75	53723	35	35	1912859	2.046	0.924
404	0.125	10175	215	215	2306476	2.012	0.916
404	0.25	20351	103	103	2165078	2.028	0.921
404	0.5	40703	48	48	1993584	2.041	0.922
404	0.75	61054	30	30	1859210	2.027	0.923
429	0.125	11475	190	190	2308903	2.019	0.917
429	0.25	22951	88	88	2083586	2.011	0.923
429	0.5	45903	41	41	1917277	2.025	0.928
429	0.75	68854	26	26	1813676	2.022	0.929
454	0.125	12853	168	168	2278963	2.020	0.911
454	0.25	25707	79	79	2094640	2.024	0.923
454	0.5	51415	36	36	1887544	2.017	0.920
454	0.75	77123	23	23	1798103	2.036	0.920
479	0.125	14310	147	147	2210773	2.019	0.919
479	0.25	28620	70	70	2070646	2.013	0.920
479	0.5	57240	32	32	1863155	2.060	0.928
479	0.75	85860	20	20	1739101	2.024	0.926
504	0.125	15844	132	132	2197260	2.016	0.914
504	0.25	31689	62	62	2025673	2.022	0.919
504	0.5	63378	29	29	1869638	2.056	0.922
504	0.75	95067	18	18	1735641	2.026	0.921
529	0.125	17457	121	121	2214675	2.026	0.918
529	0.25	34914	56	56	2012176	2.026	0.922
529	0.5	69828	26	26	1848503	2.064	0.921
529	0.75	104742	13	13	1380963	2.074	0.921
554	0.125	19147	108	108	2172971	2.027	0.914
554	0.25	38295	51	51	2009129	2.045	0.920
554	0.5	76590	23	23	1790717	2.028	0.921
554	0.75	114885	15	15	1745575	2.108	0.924
579	0.125	20916	98	98	2141423	2.023	0.917
579	0.25	41832	46	46	1973103	2.041	0.926
579	0.5	83665	17	17	1445896	2.055	0.925
579	0.75	125498	13	13	1652314	2.118	0.923

TABLE IX
Performance Metrics by Nodes and Edge Density of Probabilistic Greedy Algorithm with exponent 10 - Part 2

# Nodes	Edge Density	# Edges	#All Solut.	#Solut. Tested	#Basic Oper.	Exec. Time(s)	Precision
604	0.125	22763	90	90	2148199	2.016	0.913
604	0.25	45526	41	41	1910654	2.011	0.923
604	0.5	91053	19	19	1758849	2.044	0.920
604	0.75	136579	12	12	1656312	2.085	0.924
629	0.125	24688	82	82	2109952	2.011	0.918
629	0.25	49376	36	36	1827842	2.030	0.919
629	0.5	98753	17	17	1708624	2.016	0.919
629	0.75	148129	11	11	1647830	2.106	0.921
654	0.125	26691	76	76	2114902	2.013	0.916
654	0.25	53382	35	35	1915460	2.039	0.919
654	0.5	106765	14	14	1518474	2.103	0.921
654	0.75	160148	10	10	1619915	2.185	0.923
679	0.125	28772	70	70	2096798	2.033	0.922
679	0.25	57545	31	31	1831481	2.013	0.917
679	0.5	115090	15	15	1753086	2.130	0.925
679	0.75	172635	9	9	1571744	2.116	0.922
704	0.125	30932	64	64	2064080	2.020	0.913
704	0.25	61864	29	29	1838481	2.053	0.920
704	0.5	123728	13	13	1634942	2.102	0.919
704	0.75	185592	8	8	1500848	2.095	0.923
729	0.125	33169	60	60	2063000	2.042	0.920
729	0.25	66339	27	27	1833947	2.034	0.919
729	0.5	132678	12	12	1615252	2.082	0.920
729	0.75	199017	7	7	1407067	2.028	0.922
754	0.125	35485	55	55	2029069	2.032	0.915
754	0.25	70970	25	25	1815228	2.046	0.918
754	0.5	141940	9	9	1294115	2.076	0.919
754	0.75	212910	7	7	1506586	2.255	0.920
779	0.125	37878	50	50	1969862	2.014	0.913
779	0.25	75757	23	23	1781660	2.023	0.918
779	0.5	151515	10	10	1536809	2.012	0.918
779	0.75	227273	5	5	1147768	2.149	0.918
804	0.125	40350	46	46	1929794	2.045	0.913
804	0.25	80701	21	21	1733082	2.019	0.919
804	0.5	161403	10	10	1637069	2.175	0.922
804	0.75	242104	6	6	1466223	2.237	0.921
829	0.125	42900	36	36	1600528	2.037	0.918
829	0.25	85801	20	20	1757677	2.069	0.920
829	0.5	171603	9	9	1565076	2.123	0.924
829	0.75	257404	5	5	1299087	2.077	0.922
854	0.125	45528	41	41	1938864	2.033	0.912
854	0.25	91057	19	19	1768516	2.084	0.918
854	0.5	182115	8	8	1477663	2.047	0.919
854	0.75	273173	5	5	1379810	2.238	0.918
879	0.125	48235	38	38	1900239	2.019	0.914
879	0.25	96470	18	18	1776984	2.089	0.917
879	0.5	192940	8	8	1564763	2.206	0.925
879	0.75	289410	4	4	1167038	2.209	0.925
904	0.125	51019	36	36	1904041	2.031	0.918
904	0.25	102039	17	17	1769243	2.129	0.919
904	0.5	204078	7	7	1444562	2.117	0.918
904	0.75	306117	4	4	1234765	2.337	0.923
929	0.125	53882	34	34	1897351	2.029	0.917
929	0.25	107764	16	16	1761741	2.122	0.918
929	0.5	215528	7	7	1528437	2.252	0.921
929	0.75	323292	4	4	1304996	2.394	0.923
954	0.125	56822	32	32	1879693	2.013	0.917
954	0.25	113645	14	14	1624958	2.023	0.915
954	0.5	227290	6	6	1379634	2.095	0.923
954	0.75	340935	4	4	1375634	2.368	0.921
979	0.125	59841	30	30	1857259	2.015	0.915
979	0.25	119682	14	14	1709596	2.105	0.920
979	0.5	239365	6	6	1451670	2.231	0.922
979	0.75	359048	3	3	1087031	2.297	0.918

IX. Appendix D - Probab. Greedy Results - Exponent 100

TABLE X
Performance Metrics by Nodes and Edge Density of Probabilistic Greedy Algorithm with exponent 100 - Part 1

# Nodes	Edge Density	# Edges	#All Solut.	#Solut. Tested	#Basic Oper.	Exec. Time(s)	Precision
4	0.25	1	1000	1	2	0.025	1.000
4	0.5	3	1000	2	8	0.050	0.971
4	0.75	4	1000	1	6	0.060	1.000
29	0.125	50	1000	1000	76387	0.353	0.993
29	0.25	101	1000	1000	120102	0.526	1.000
29	0.5	203	1000	1000	217475	0.780	0.983
29	0.75	304	1000	1000	344262	1.044	0.985
54	0.125	178	1000	1000	216390	0.688	0.958
54	0.25	357	1000	1000	419394	1.112	0.945
54	0.5	715	1000	1000	753188	1.878	0.980
54	0.75	1073	753	753	852209	2.006	0.992
79	0.125	385	1000	1000	454645	1.171	0.940
79	0.25	770	990	990	828520	2.006	0.975
79	0.5	1540	533	533	859008	2.006	0.988
79	0.75	2310	358	358	872187	2.011	0.985
104	0.125	669	1000	1000	763377	1.795	0.942
104	0.25	1339	560	560	825900	2.007	0.966
104	0.5	2678	273	273	775280	2.005	0.974
104	0.75	4017	202	202	839226	2.007	0.985
129	0.125	1032	761	761	891894	2.007	0.936
129	0.25	2064	405	405	900018	2.006	0.967
129	0.5	4128	196	196	846717	2.007	0.991
129	0.75	6192	127	127	815249	2.016	0.990
154	0.125	1472	556	556	902719	2.008	0.949
154	0.25	2945	281	281	873357	2.010	0.971
154	0.5	5890	129	129	788795	2.009	0.982
154	0.75	8835	85	85	775894	2.013	0.988
179	0.125	1991	421	421	939094	2.006	0.945
179	0.25	3982	197	197	827246	2.008	0.980
179	0.5	7965	96	96	792905	2.017	0.983
179	0.75	11948	60	60	738726	2.010	0.986
204	0.125	2588	274	274	778069	2.010	0.951
204	0.25	5176	144	144	792924	2.006	0.965
204	0.5	10353	71	71	764285	2.008	0.980
204	0.75	15529	44	44	701260	2.017	0.983
229	0.125	3263	253	253	902083	2.013	0.952
229	0.25	6526	119	119	816475	2.014	0.977
229	0.5	13053	55	55	743303	2.038	0.982
229	0.75	19579	35	35	700470	2.058	0.987
254	0.125	4016	204	204	892974	2.014	0.951
254	0.25	8032	92	92	775245	2.017	0.974
254	0.5	16065	43	43	714235	2.016	0.978
254	0.75	24098	27	27	666959	2.007	0.982
279	0.125	4847	162	162	845560	2.010	0.958
279	0.25	9695	73	73	741391	2.019	0.976
279	0.5	19390	32	32	640261	2.043	0.983
279	0.75	29085	22	22	655016	2.064	0.985
304	0.125	5757	136	136	844880	2.019	0.957
304	0.25	11514	63	63	759864	2.023	0.973
304	0.5	23028	29	29	686632	2.059	0.980
304	0.75	34542	18	18	634894	2.056	0.987
329	0.125	6744	115	115	834653	2.013	0.962
329	0.25	13489	52	52	731737	2.046	0.972
329	0.5	26978	24	24	665462	2.037	0.983
329	0.75	40467	15	15	619364	2.035	0.988
354	0.125	7810	97	97	811179	2.025	0.961
354	0.25	15620	36	36	585172	2.046	0.974
354	0.5	31240	21	21	672952	2.094	0.980
354	0.75	46860	13	13	619413	2.125	0.985
379	0.125	8953	83	83	789840	2.020	0.962
379	0.25	17907	38	38	703853	2.006	0.982
379	0.5	35815	18	18	660409	2.103	0.983
379	0.75	53723	11	11	600470	2.065	0.988
404	0.125	10175	72	72	777803	2.012	0.965
404	0.25	20351	33	33	698969	2.031	0.974
404	0.5	40703	15	15	624685	2.026	0.984
404	0.75	61054	8	8	497635	2.119	0.984
429	0.125	11475	63	63	773244	2.020	0.965
429	0.25	22951	29	29	688851	2.044	0.979
429	0.5	45903	13	13	610041	2.038	0.987
429	0.75	68854	9	9	629413	2.241	0.987
454	0.125	12853	56	56	759398	2.032	0.967
454	0.25	25707	25	25	666263	2.005	0.977
454	0.5	51415	12	12	631537	2.127	0.981
454	0.75	77123	8	8	625589	2.284	0.986
479	0.125	14310	43	43	650904	2.007	0.967
479	0.25	28620	22	22	650826	2.039	0.979
479	0.5	57240	10	10	585216	2.037	0.982
479	0.75	85860	7	7	610231	2.299	0.988
504	0.125	15844	44	44	737455	2.050	0.964
504	0.25	31689	20	20	655475	2.064	0.978
504	0.5	63378	9	9	581652	2.046	0.982
504	0.75	95067	6	6	579097	2.459	0.985
529	0.125	17457	39	39	715402	2.005	0.972
529	0.25	34914	18	18	649681	2.114	0.977
529	0.5	69828	8	8	568845	2.048	0.985
529	0.75	104742	5	5	531985	2.036	0.988
554	0.125	19147	36	36	726735	2.058	0.967
554	0.25	38295	16	16	632931	2.017	0.977
554	0.5	76590	7	7	545660	2.018	0.983
554	0.75	114885	5	5	583076	2.256	0.985
579	0.125	20916	30	30	658612	2.053	0.968
579	0.25	41832	15	15	647103	2.117	0.977
579	0.5	83665	7	7	597053	2.207	0.985
579	0.75	125498	4	4	509354	2.055	0.987

TABLE XI
Performance Metrics by Nodes and Edge Density of Probabilistic Greedy Algorithm with exponent 100 - Part 2

# Nodes	Edge Density	# Edges	#All Solut.	#Solut. Tested	#Basic Oper.	Exec. Time(s)	Precision
604	0.125	22763	29	29	692877	2.037	0.969
604	0.25	45526	12	12	563210	2.044	0.977
604	0.5	91053	6	6	556660	2.090	0.985
604	0.75	136579	4	4	552863	2.292	0.985
629	0.125	24688	27	27	697997	2.067	0.969
629	0.25	49376	12	12	609930	2.057	0.979
629	0.5	98753	6	6	602276	2.292	0.985
629	0.75	148129	3	3	450571	2.245	0.987
654	0.125	26691	24	24	671010	2.028	0.969
654	0.25	53382	11	11	603601	2.053	0.978
654	0.5	106765	5	5	543030	2.090	0.983
654	0.75	160148	3	3	486130	2.080	0.986
679	0.125	28772	23	23	694969	2.087	0.970
679	0.25	57545	8	8	473686	2.011	0.978
679	0.5	115090	5	5	585276	2.268	0.984
679	0.75	172635	3	3	524375	2.277	0.988
704	0.125	30932	21	21	680302	2.088	0.969
704	0.25	61864	10	10	636407	2.205	0.979
704	0.5	123728	4	4	503513	2.387	0.985
704	0.75	185592	3	3	563152	2.591	0.988
729	0.125	33169	19	19	657301	2.033	0.973
729	0.25	66339	9	9	615367	2.173	0.978
729	0.5	132678	4	4	538354	2.167	0.986
729	0.75	199017	2	2	402384	2.684	0.987
754	0.125	35485	17	17	627986	2.005	0.971
754	0.25	70970	8	8	582616	2.081	0.979
754	0.5	141940	4	4	576113	2.382	0.986
754	0.75	212910	2	2	430610	2.097	0.987
779	0.125	37878	16	16	632145	2.029	0.973
779	0.25	75757	8	8	621101	2.298	0.979
779	0.5	151515	4	4	615940	2.591	0.984
779	0.75	227273	2	2	459102	2.323	0.989
804	0.125	40350	12	12	504505	2.029	0.971
804	0.25	80701	7	7	579191	2.144	0.979
804	0.5	161403	3	3	490746	2.144	0.986
804	0.75	242104	2	2	488635	2.477	0.988
829	0.125	42900	14	14	624109	2.048	0.973
829	0.25	85801	6	6	527214	2.358	0.981
829	0.5	171603	3	3	522017	2.294	0.986
829	0.75	257404	2	2	520847	2.677	0.989
854	0.125	45528	13	13	616741	2.073	0.969
854	0.25	91057	6	6	558870	2.151	0.980
854	0.5	182115	3	3	553972	2.509	0.986
854	0.75	273173	2	2	551320	2.931	0.987
879	0.125	48235	12	12	602178	2.027	0.973
879	0.25	96470	6	6	591371	2.280	0.981
879	0.5	192940	3	3	587062	2.686	0.988
879	0.75	289410	2	2	583891	3.191	0.989
904	0.125	51019	12	12	636921	2.165	0.973
904	0.25	102039	5	5	522348	2.036	0.980
904	0.5	204078	2	2	413806	2.017	0.985
904	0.75	306117	2	2	619858	3.389	0.987
929	0.125	53882	9	9	504053	2.098	0.973
929	0.25	107764	5	5	551048	2.169	0.981
929	0.5	215528	2	2	437545	2.147	0.985
929	0.75	323292	1	1	325802	2.099	0.989
954	0.125	56822	10	10	592336	2.021	0.973
954	0.25	113645	5	5	580958	2.292	0.980
954	0.5	227290	2	2	460190	2.585	0.986
954	0.75	340935	1	1	344154	2.158	0.989
979	0.125	59841	10	10	619724	2.151	0.976
979	0.25	119682	4	4	488945	2.212	0.982
979	0.5	239365	2	2	484601	2.531	0.988
979	0.75	359048	1	1	362104	2.226	0.990