DIMACS Series in Discrete Mathematics
and Theoretical Computer Science
Volume 00, 0000

An Experimental Comparison of Two Maximum Cardinality Matching Programs

STEVEN T. CROCKER

ABSTRACT. We report results of experiments comparing the run time behavior of Micali and Vazirani's $O(m\sqrt{n})$ algorithm (MV) and Gabow's $O(n^3)$ algorithm (G) for maximum cardinality matching. These experiments used randomly generated graphs with vertex set sizes between 1024 and 8196 as input. Within the sequence of graphs of an experiment, some parameter was kept constant, such as the expected degree of a vertex or the number of vertices in the graph. The results show the dependence of run time on expected degree of a vertex, the correlation of blossom formation and run time, and the significant cost of beginning a new phase in algorithm MV. Also, they show thresholds and characteristics of blossom formation in both algorithms and of the number of phases in algorithm MV.

1. Introduction

In this paper, we report the results of experiments investigating the timings and combinatorial features found by two algorithms that solve the maximum cardinality matching problem on nonbipartite graphs. For a given graph $G = (V, E)$, let n be the size of the vertex set V, and m be the size of the edge set E. The experiments used two algorithms, one discovered by Micali and Vazirani[**4**] (referred to as MV), which has a bounded running time of $O(m\sqrt{n})$ and one discovered by Gabow[**2**] (referred to as G), which has a bounded running time of $O(n^3)$.

The emphasis of the project consisted of two experiments, *constant number of vertices (CED)* and *varying expected degree (VED)*. They were designed to investigate the dependence of algorithm performance on parameters of random graphs. The purpose of experiment *CED* was to look at algorithm performance as it depended on the size of a graph. This experiment gathered results of the algorithms running on graphs with different sized vertex sets but with constant expected degrees for their vertices. It was used to determine how significantly

1991 *Mathematics Subject Classification.* Primary 05C85; Secondary 05C70, 68Q20.

the initialization phases of the algorithms contributed to the run times, how rapidly the algorithms moved towards their asymptotic performance, and how their asymptotic performance compared to their theoretical bounds.

The purpose of experiment *VED* was to investigate the combinatorial and timing characteristics of the algorithms on random graphs with a constant number of vertices but over a range of edge densities. The multiple runs of this experiment using different vertex set sizes gave us a basis for claiming the observed performance characteristics were due to the density of the graph and not to the size of the graph. We also performed a bipartite version of experiment *VED* to investigate more thoroughly the effects of blossom formation on run time.

In the next section we give a brief description of the two algorithms used in this experiment and some details of the implementations. Section 3 discusses the design of the experiments and sections 4 and 5 contain the discussion of their results. Section 6 contains the results of running the two algorithms on another of the DIMACS Implementation Challenge's core experiments for maximum cardinality matching.

2. Algorithms and Their Implementations

In this section we give high level descriptions of the Micali-Vazirani and the Gabow matching algorithms. We follow each description with some implementation details.

2.1. The Micali-Vazirani Algorithm (MV). Algorithm MV developed by Micali and Vazirani[**4**] proceeds in phases, where each phase increases the current partial matching by finding a maximum set of vertex disjoint shortest length augmenting paths. The algorithm halts when either there are no more free vertices or a phase finds no augmenting paths. A phase operates in two alternating modes, incrementing the depth of the breadth first search (*bfs*) trees and processing bridges. First, the algorithm synchronously builds breadth first search trees along alternating paths from each free vertex. All trees are built to the same level before the synchronous bfs continues onto the next level. At each level, the synchronous bfs discovers a (possibly empty) set of bridges. Bridges are edges, (u, v), which are used in both directions, as (u, v) and (v, u), by the same level of the synchronous bfs. Thus, a bridge is the edge at which two bfs tree's meet or at which a bfs tree rejoins itself. Before the phase extends the bfs to the next higher level, the phase switches to its other mode and examines each bridge discovered at the current level. For a given bridge, a synchronous double depth first search seeks disjoint alternating paths from the two vertices of the bridge, down their respective bfs trees. If the two depth first searches (dfs) find two vertex disjoint alternating paths to free vertices, then an augmenting path has been found. This path is augmented, increasing the matching by one, and then topologically erased from the bfs trees for the remainder of the phase. This erasing forces all augmenting paths found from bridges of this level to be vertex disjoint. If instead, in the double depth first search, the two dfs trees become bottlenecked at some vertex because neither dfs can find another path around that vertex, then a blossom has been discovered and the bottleneck is the blossom's base vertex. Defining a blossom requires traversing the ddfs tree to the bottleneck, and adding to the new blossom each vertex that is not already in a blossom. After all bridges have been examined, if no augmenting paths were found then the phase returns to building the synchronous bfs trees to the next higher level. If, however, one or more augmentations were done, then the phase is over. A new phase is begun by restarting the bfs trees at level 0 from the unmatched vertices in the original graph with the new matching.

Each phase of the algorithm examines a vertex or edge $O(1)$ times. By a result of Hopcroft and Karp[**3**], there are at most $\sqrt{n}$ phases. This gives the $O(m\sqrt{n})$ run time of the algorithm.

2.2. Implementation of Micali and Vazirani's Algorithm. Algorithm MV was implemented in slightly more than 1000 lines of Pascal code by the author. The graph to be matched is represented as a modified adjacency list. Each vertex is represented by a record which contains a pointer to a list of incident edges. Each edge is represented by a unique record that contains pointers to both of its vertices and pointers to the next edge in each of their respective edge lists. The vertices' and edges' records also contain fields for the data needed to build and retain the bfs and the dfs trees, the blossoms, and the augmenting paths. Thus, the algorithm allocates almost all the storage space it needs dynamically, but it does so during initialization when reading the problem graph.

2.3. Gabow's Algorithm. Gabow's algorithm, which is a variant of Edmonds' matching algorithm, is simpler. [**1, 2**] Algorithm G builds a search tree rooted at each free vertex, but it only builds one tree at a time. The search tree for this implementation is built in a breadth first manner in the sense that unexamined edges which would extend the current alternating search tree are examined in a first in, first out order; it is not a bfs where the alternating paths all have equal length, as in algorithm MV (however, in a bipartite graph where no blossoms occur, the two would be the same). Each time the current search tree is grown, it is extended by two edges: first by an unmatched edge that leaves the current search tree and then by the matched edge outside of the search tree that shares an endpoint with the unmatched edge. The vertex through which the matched edge was reached is called an *inner* vertex. The other vertex of the matched edge is *outer*. The algorithm has found an augmenting path when it reaches a free vertex; it has found a blossom when it reaches an outer vertex via an unmatched edge. A scheme using labels kept at each vertex allows the algorithm to reconstruct an augmenting path through blossoms after it reaches a free vertex.

2.4. Implementation of Gabow's Algorithm. Algorithm G was implemented in slightly under 300 lines of C code by Ed Rothberg and modified slightly by the author. It was made available by the DIMACS Implementation Challenge organizing committee. The implementation uses array structures to store the graph and the data structures of the algorithm. The problem graph is represented as an adjacency list with two records allocated for each edge. The space requirements of this algorithm are considerably smaller than those of MV because of the simpler data structures involved. Besides the storage needed to store the original graph, algorithm G requires only five additional arrays of integers of length n. These arrays are allocated dynamically during program initialization.

3. Experiment Designs

The main project consisted of a number of separate experiments. Each experiment was made up of a number of individual runs. A run involved executing both algorithms on a sequence of related graphs and plotting the data gathered. A data point was generally the average of the statistics gathered by executing an algorithm on twenty random graphs chosen from $G_{n,m}$ where $G_{n,m}$ is the class of all graphs with n vertices and m edges. An algorithm's run time did not include the time to read the graph, to build its adjacency list, nor to print the maximum matching.

The design of an experiment was fixed by choosing the way in which n and m varied throughout a run. In the first experiment, called CED for *C*onstant *E*xpected *D*egree, the ratio $2m/n$ was fixed and n increased by factors of 2. The seven runs in this experiment used graphs with expected degrees of 2^i for $0 \leq i \leq 6$. In the second experiment, called VED for *V*arying *E*xpected *D*egree, n was fixed and m increased by factors of $2^{1/16}$. The four runs in this experiment had n fixed at values of 2^{10+i} for $0 \leq i \leq 3$. Every data point shown is the mean of the statistics gathered from executing the algorithm on twenty random graphs generated with the same parameters n and m. Data points from each run are connected by lines in the figures of the experimental data that follow.

The random graphs of these experiments were generated by a random graph generator supplied by DIMACS. This generator worked by fixing the number of vertices and edges a graph would have and then, for each potential edge in the graph, randomly deciding whether or not that edge is in the graph with probability $\frac{\text{number of edges left to choose}}{\text{number of potential edges remaining}}$. The two experiments and the values of their key parameters are summarized in figure 1.

Lines connect data points of a particular run. In a figure with plots of multiple runs, the shape of the symbol used to denote data points for a particular run is the key to the value of the invariant for that run. Symbols are only printed at the data points which have a value on the abscissa that is an integral power of two. The experiments were conducted on a Sun 4/40 ipc workstation with 24 meg of

EXPERIMENT		Run		*Data Points*		
				Minimum	Maximum	*Increment*
Name	*Description*	*Size*	*Symbol*	*Size*	*Size*	*Factor*
CED	Constant	deg = 1	□	$\|V\|$=256	$\|V\|$=32768	2
	Expected	2	○	256	32768	
	Degree	4	△	256	32768	
		8	+	256	32768	
		16	×	128	16384	
		32	◊	128	16384	
		64	▽	128	4096	
VED	Varying	$\|V\|$ =1024	□	$\|E\|$=512	$\|E\|$=92681	$2^{1/16}$
	Expected	2048	○	512	220435	
	Degree	4096	△	2048	193570	
		8192	◊	2048	65536	

FIGURE 1. Parameters of the two experiments. Each experiment kept the invariant parameter constant while increasing the variant parameter by the increment factor.

memory. The cpu was rated at 15 MIPS. The programs used for the experiments described in this paper are available via anonymous ftp from DIMACS.

4. Experiment Constant Expected Degree (CED)

The main purpose of this experiment was to study the effect of graph size on run time. We attempted to eliminate systematic structural changes within the graphs of a run and to isolate the effects of graph size by fixing the degree of all the graphs within a run. For example, one effect graph size could have on run time is the overhead of initialization of an algorithm after it has read the graph. A log-log plot of runtime versus number of vertices for polynomial time algorithms such as these two, produce linear graphs. The slope of these graphs give a means to compare actual performance with asymptotic bounds. Figure 2 shows a log-log plot of runtime vs. number of vertices for algorithm MV.

Figure 3 shows a log-log plot of runtime vs. number of vertices for algorithm G. Each line in the graph connects the data points for one run of the experiment. Vertices within all graphs of a run have the same expected degree. the symbol marking each data point denotes the expected degree of the graphs of that run.

The near linearity of all lines in figure 2 is also evident from the 0.99 or larger correlation coefficients of the data from these lines. The slopes of these lines, given in figure 4, which are all between 1.05 and 1.25 indicate the algorithm's runtime is growing superlinearly, but not as fast as the asymptotic bound of $O(m\sqrt{n})$ which would have yielded slopes for these sparse graphs of 1.5. The bunching of the lines of the runs with expected degrees of 4, 8, and 16 indicates that in this range, the runtime of the algorithm is not very affected by the

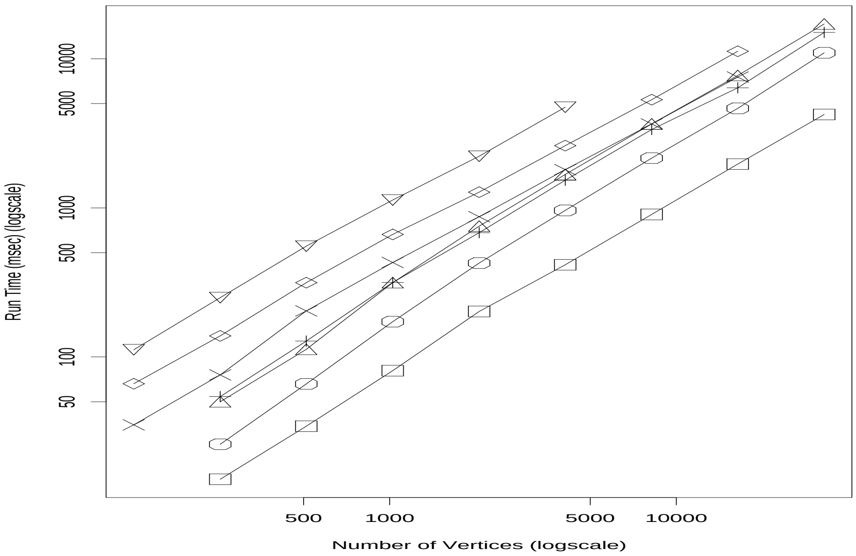

FIGURE 2. Experiment CED for algorithm MV showing run times versus number of vertices on a log-log scale. Symbols mark data points at vertex set sizes which are powers of 2. The symbol denotes the expected degree of a random vertex from any graph in the run. They are: expected degree 1 : $\square$, 2 : $\bigcirc$, 4 : $\triangle$, 8 : $+$, 16 : $\times$, 32 : $\Diamond$, and 64 : $\bigtriangledown$.

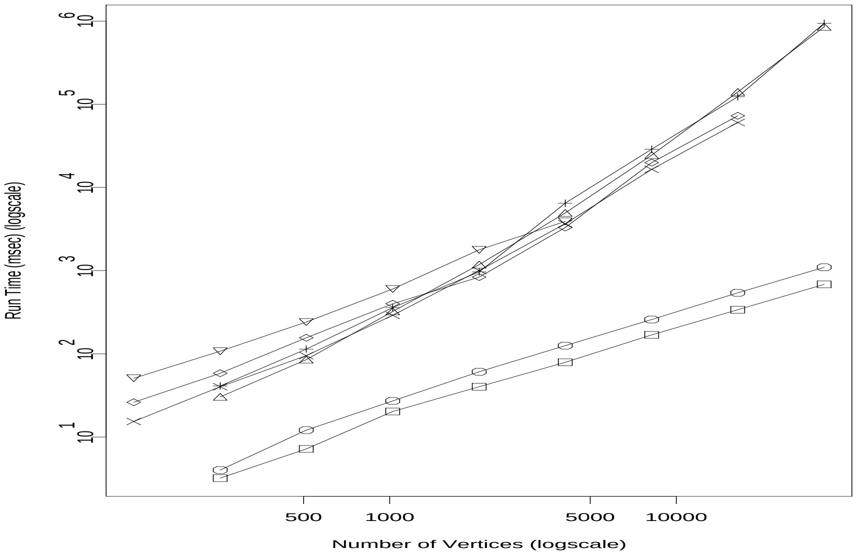

FIGURE 3. Experiment CED for algorithm G showing run times versus number of vertices on a log-log scale. Each run has an expected degree and symbol from $1 : \square ; 2 : \bigcirc ; 4 : \triangle ; 8 : + ; 16 : \times ; 32 : \Diamond ; 64 : \bigtriangledown$.

FIGURE 4. Slopes of the line fit by least squares through the points in Figures 2 and 3.

Expected	*Algorithm*	
Degree	*MV*	*G*
1	1.16	1.10
2	1.23	1.13
3	1.20	2.12
4	1.15	2.07
5	1.10	1.72
6	1.05	1.62
7	1.07	1.28

expected degree. This phenomenon and others will be clearly seen in the results of the next section.

The corresponding graph for algorithm G shows more complex shape. The data for the degree 1 and 2 runs show linear behavior. These runs have slopes similar to those of the MV algorithm (figure 4). The data from the runs of higher expected degree show a slightly convex shape. The slope of the next line, for graphs of expected degree 3, and the gap between that line and the line for graphs of expected degree 2 indicates a threshold exists for the algorithm on random graphs with expected degree between 2 and 3. The slopes of the lines from the runs with expected degree 3 and 4 show the algorithm runtimes are more than quadratic in the number of vertices over that range but significantly less than cubic. The gradual reduction of the slopes as the expected degree increases further indicates something else is affecting the average length of an augmenting path. Again, these phenomena will be more clearly seen in the next section.

5. Experiment Varying Expected Degree (VED)

In this experiment the densities of graphs in a particular run varied by keeping the number of vertices constant and increasing the number of edges. We looked at the relations of run time, blossom formation, and number of phases to the expected degree of the random graphs for algorithm MV and at the relations of run time and blossom formation to expected degree for algorithm G. Also, we compared the run times of the two algorithms and the number of blossoms they found.

Figure 5 shows the runtimes versus expected degree plotted on a log-log scale for the four runs of algorithm MV with 2^i, $10 \leq i \leq 13$, vertices. The relatively even spacing between all four lines is predictable from the linearity results of experiment CED in the previous section. The spike in the runtime of all four runs occurs at a degree in the range between 2.7 and 2.8. The period of flat runtimes between degree's 4 and 10 corresponds to the bunching of lines for degrees 4 and 8 (and at larger graph sizes, 16) in figure 2. By degree 20, the log-log plots have near linear growth with a slope slightly less than 1.

Figure 6 shows the run times versus the expected degree of algorithm G on a log-log plot for runs with graphs having vertex sets of size 2^i, $10 \leq i \leq 13$. As in the previous plot, the nearly even spacing of the runs corresponds to the near linearity found in experiment CED. The jump in run time in that experiment that occurred from expected degree 2 to 3 occurs here as a very sharp rise in run time as the expected degree increases through that range. This jump in run time is centered between 2.7 and 2.8, roughly the same location as the peak in run time which occurs in algorithm MV. From expected degree 3 up to much larger expected degrees, the run times are almost flat, which again corresponds with the bunching of the lines from the different runs of experiment CED. For the

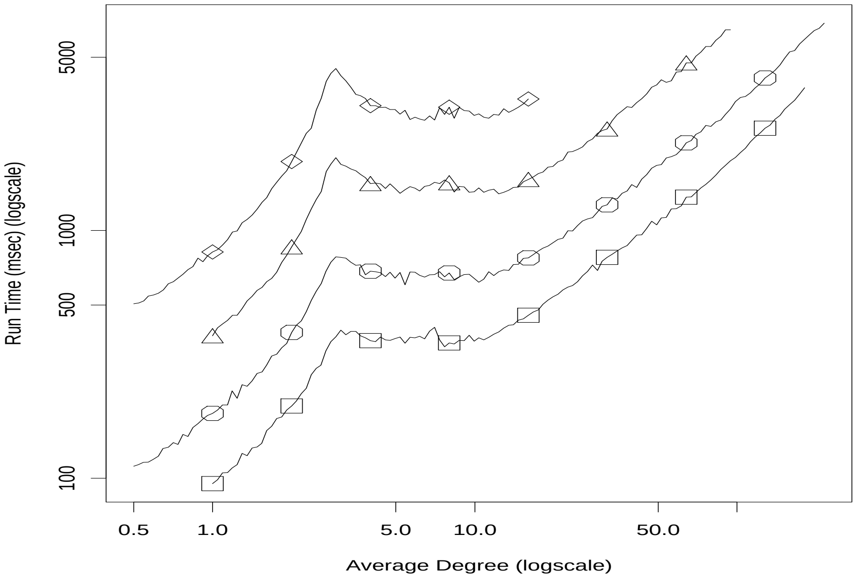

Figure 5. Experiment VED for algorithm MV showing run time versus expected degree on a log-log scale for graphs with 2^{10},2^{11},2^{12} and 2^{13} vertices.

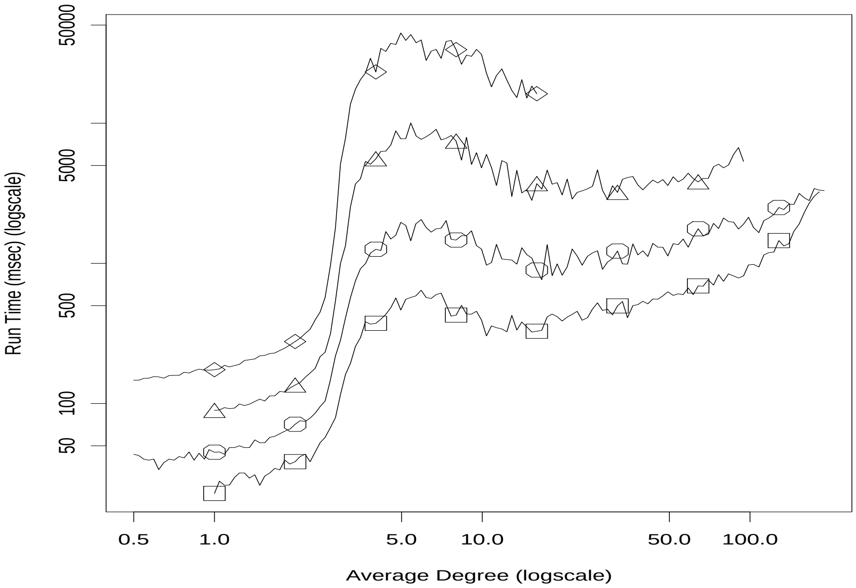

Figure 6. Experiment VED for algorithm G showing run time versus expected degree on a log-log scale for graphs with 2^{10},2^{11},2^{12} and 2^{13} vertices.

FIGURE 7. Mean runtimes and variances of the data from graphs which had 4096 vertices and average degrees that were a power of two.

	Algorithm			
Average	*MV*		*G*	
Degree	*Mean*	*Var*	*Mean*	*Var*
1	376	28	89	11
2	858	52	135	12
4	1547	171	5574	2144
8	1559	271	7428	4104
16	1608	75	3702	3580
32	2569	135	3203	1676
64	4749	483	3818	2136

smallest run, with vertex sets of size 1024, run times start rising more sharply as the graphs approach the size limit of the execution environment.

Figure 7 lists the mean runtimes and their variances from graphs which had expected vertex degrees which were a power of two. For very small degree graphs there is small variance in the runtimes because almost all the work of the algorithm is accomplished in the initial phase (algorithm MV) or greedy matching (algorithm G). With larger degrees, the variance in MV in comparison to the variance in G is quite small.

Figure 8 shows the relationships between the runtime, number of blossoms found, and number of phases for algorithm MV. The data in this figure is from the run with 1024 vertices; however the other three runs had similar plots. Unless otherwise noted, the discussion of this plot will refer generally to all of the four runs. The peak in the number of phases consistently occurs between the expected degrees of 2.7 and 2.8 for all runs. The simultaneous peaks in the number of phases and in the run time indicate there is a high overhead with initializing a phase. The initialization of a phase requires that the whole breadth first search structure and the depth first search structure from the previous phase must be cleared. The large number of data items stored with these structures leads to the large overhead. Blossoms are first discovered with graphs of expected degree near 2.5. Eventually, at higher expected degrees, blossoms are no longer found. For the graphs with vertex sets of 1024 vertices, this upper limit to blossom formation is around expected degree 10, while for the 8192 vertex run, the upper limit is near 20.

Figure 9 plots run times and blossoms found versus expected degree for algorithm G on graphs with 1024 vertices. Here again, blossoms discoveries begin with a sharp threshold at expected degrees near 2.7. They have a broad peak in the range of expected degrees 5 to 7 then decline rapidly until an expected degree near 10 from which point the number of blossoms discovered declines very gradually. From the sharp rise in run time between degrees 2.7 and 4 and

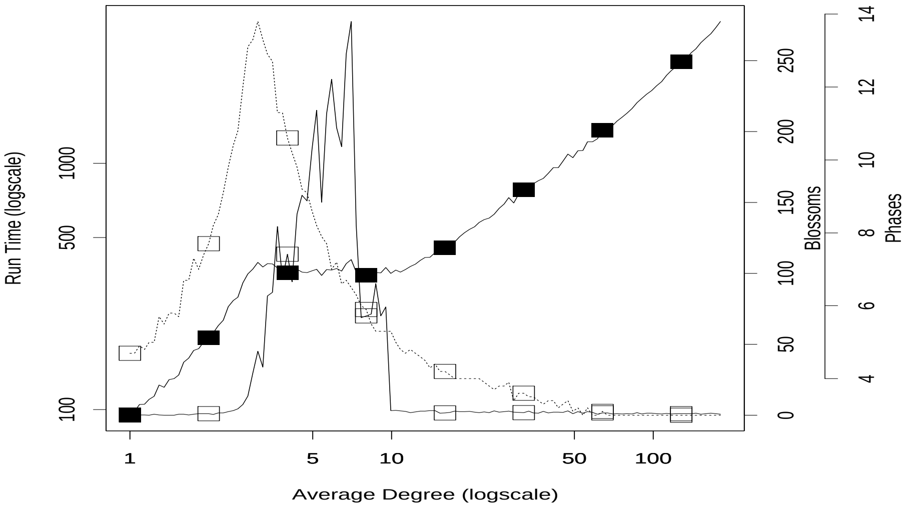

FIGURE 8. Experiment VED for algorithm G with 1024 vertices showing run times and number of blossoms discovered versus expected degree on a log-log scale. Run time: solid □, blossoms: hollow □ and solid line, phases: hollow □ and dashed line

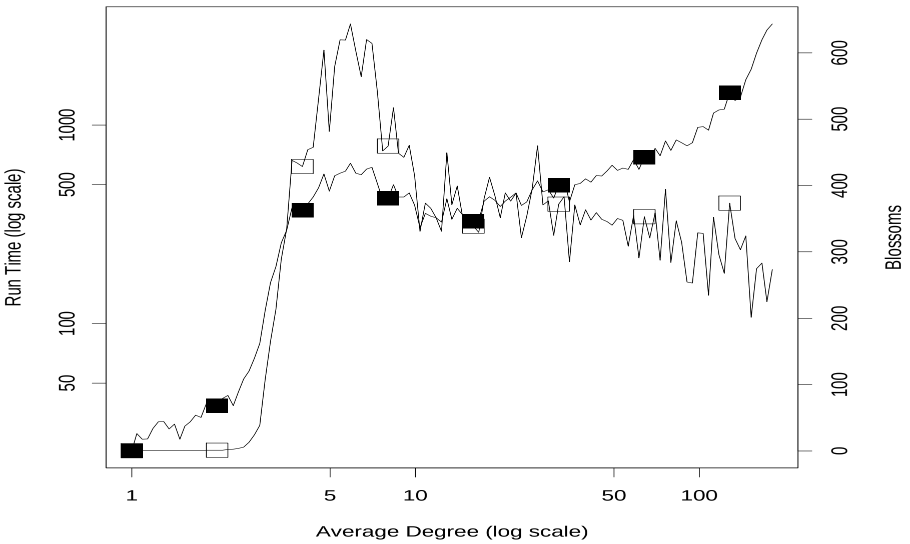

FIGURE 9. Experiment VED for algorithm G with 1024 vertices showing run times versus expected degree on a log-log scale. Run time: solid □, blossoms: hollow □

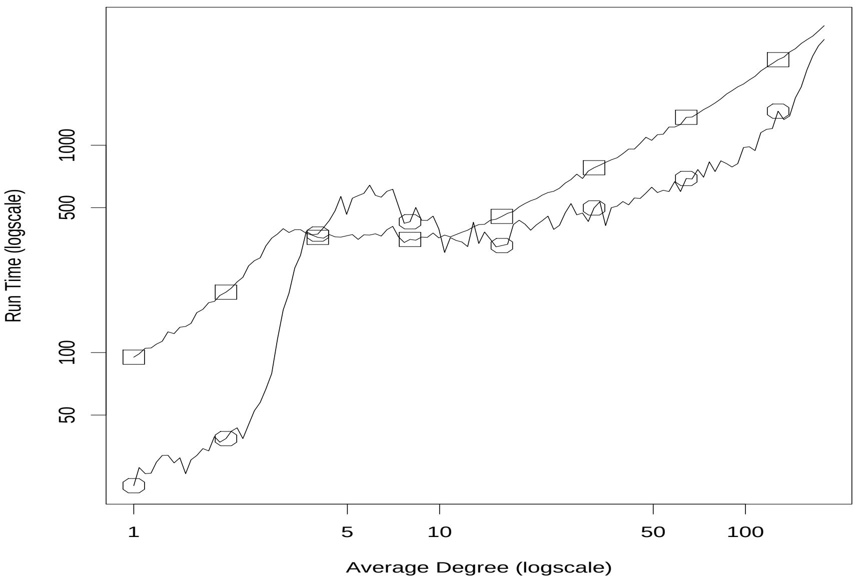

FIGURE 10. Experiment VED for vertex set sizes of 1024. Run time versus expected degree. Algorithm MV: □; algorithm G: ○.

the run time's local maxima near degree 6, it is apparent that their is a strong positive correlation between run time and blossom formation. For degrees larger than 100, the run time seems to rise rapidly with expected degree but the high density graphs for which these timings were obtained were very close to a size limit of what either the machine or compiler could handle. Beyond that limit, the programs either slowed way down due to disk operations (thrashing) or they crashed. Therefore this final rise in runtime is likely an artifact of the execution environment on which the experiment was run.

Figures 10 and 11 show the run times for both algorithms plotted against expected degree for the runs with vertex set sizes of 1024 and 8192. The two intermediate runs with vertex set sizes of 2048 and 4096 had similar plots with curves that fell, as expected, between the curves of these plots. In all four runs, algorithm G significantly outperformed algorithm MV for expected degrees less than 3. This is approximately where the local peak in algorithm MV's run time occurs. Eventually, with higher expected degree graphs, algorithm G's run times fall below those of algorithm MV. The interval of degrees where algorithm MV's run times are faster than algorithm G's grows with the size of the vertex set of the random graphs. Most of the increase in this interval occurs on the higher end. At the lower end, the crossover point seems to be bounded at graphs of expected degree 2.7. This is where algorithm MV has a peak in its run time and algorithm G has a sharp rise in its run time.

Experiment VED also led to some surprising discoveries of when blossoms are

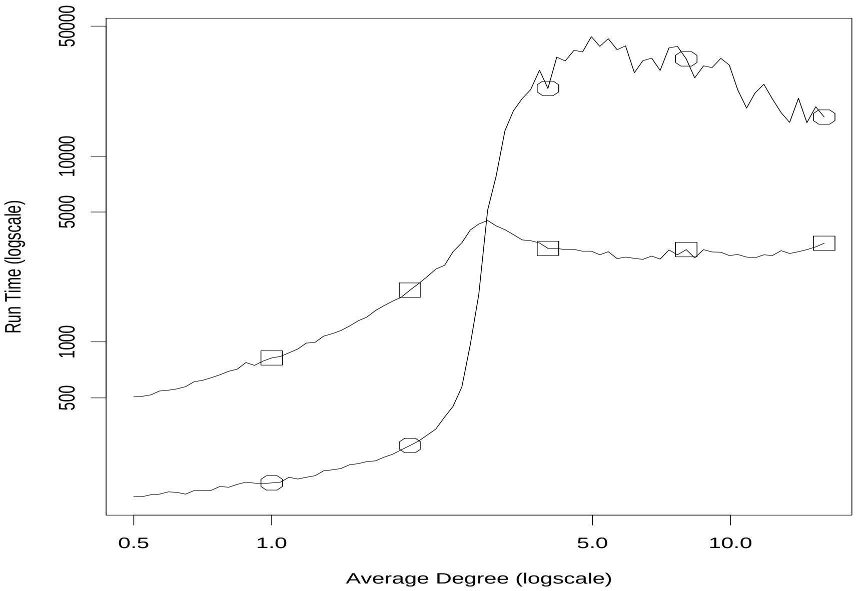

FIGURE 11. Experiment VED for vertex set sizes of 8192 showing run time verses expected degree. Algorithm MV: □, algorithm G: ○.

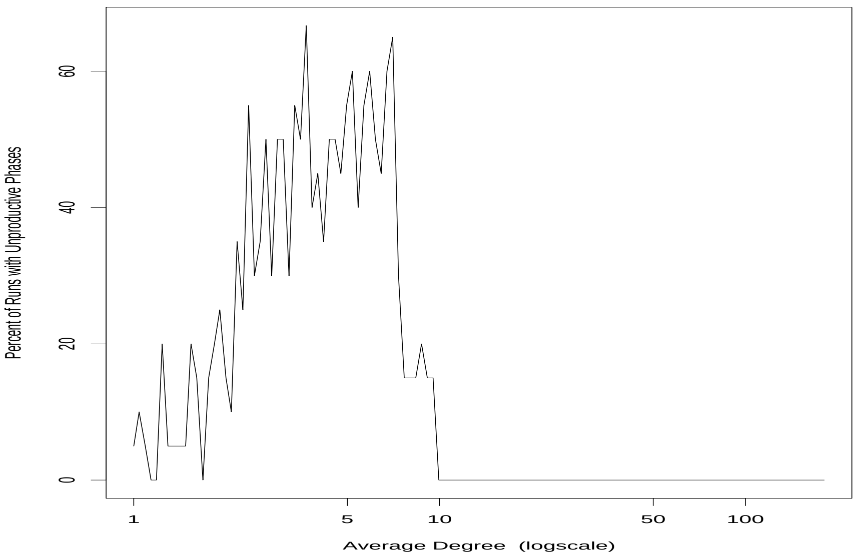

FIGURE 12. Experiment VED on MV for vertex set sizes of 1024, Fraction of blossoms found in unproductive phases versus Expected Degree.

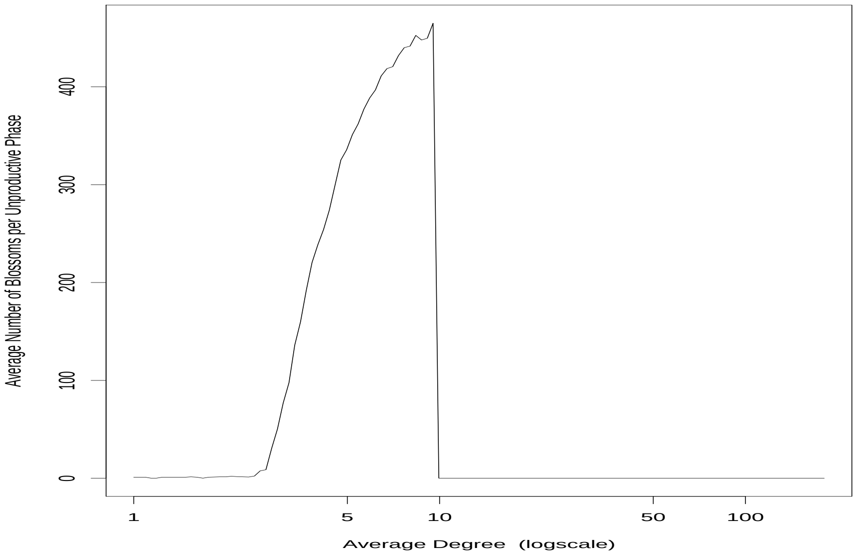

FIGURE 13. Experiment VED on MV for vertex set sizes of 1024, Average number of blossoms found in an unproductive phase versus Expected Degree.

discovered. Define an *unproductive phase* as a phase of the MV algorithm for which no augmenting paths were found. A *productive phase* is one that is not unproductive. If an unproductive phase occurs, it must be the final phase of the algorithm on a graph without a perfect matching. The way blossoms were found by algorithm MV on a random graph can be characterized in two ways. Either there were very few blossoms discovered or many blossoms were discovered but almost all of them came in an unproductive phase. Figure 12 shows the percent of all blossoms found by the algorithm which were found in an unproductive phase. Figure 13 shows the average number of blossoms discovered in a final phase for final phases in graphs of varying expected degree. The lack of blossoms discovered at degrees less than 2.5 is an indication that most of the components in the graph at these degrees are small trees. Between degrees 2.7 and 10 the average rises along a smooth curve. The smoothness of this curve is a result of a small variance in the raw data. After expected degree 10, unproductive phases cease to occur. Figure 14 shows the average number of blossoms discovered in a productive phase. The low values in this curve compared to the values in the curve in figure 13 show that when the algorithm starts finding many blossoms, it is almost surely in the final, unproductive phase.

To see the effects of blossom formation on run time, experiment VED was re-run using bipartite graphs. These graphs were generated in a similar manner as the general graphs except the set of possible edges to choose from was restricted to the edges in a complete bipartite graph with the partitions of the vertex set

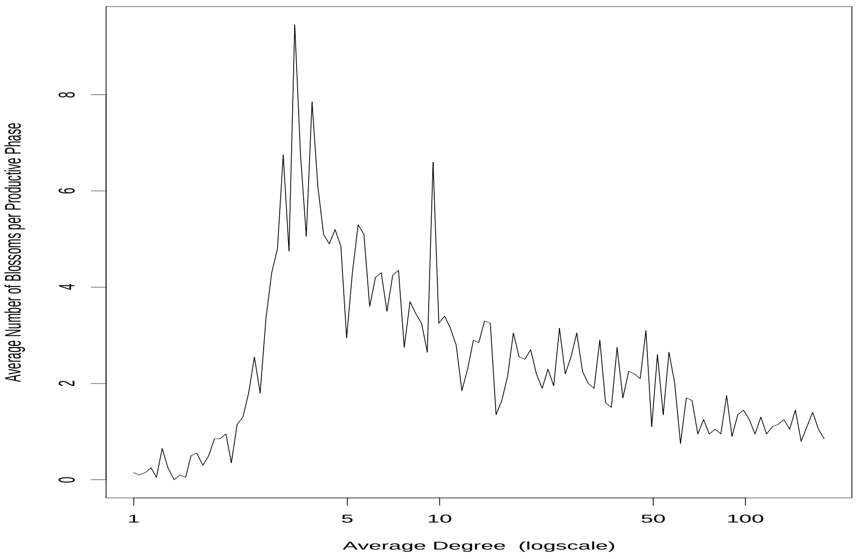

FIGURE 14. Experiment VED on algorithm MV for vertex set sizes of 1024 showing average number of blossoms found in productive phases versus expected degree.

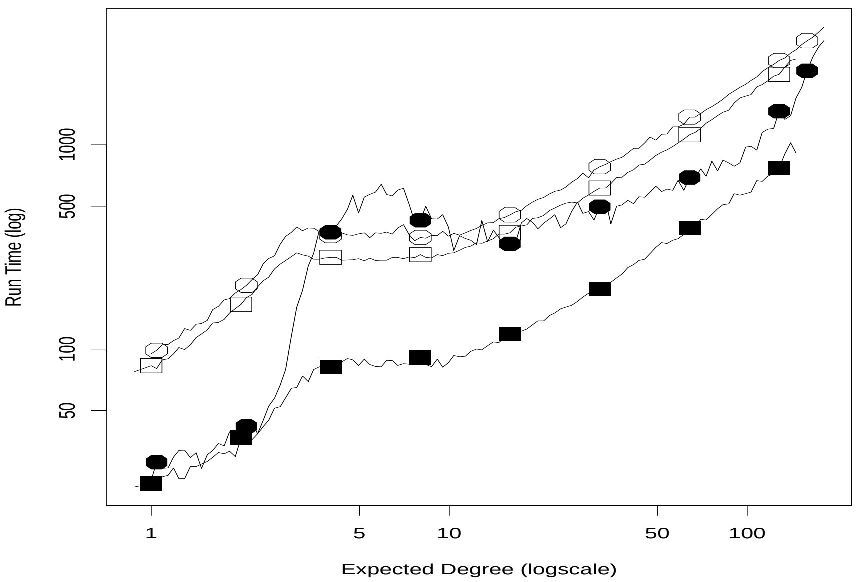

FIGURE 15. Experiment VED for both general and bipartite graphs with vertex set sizes of 1024. Run time verses expected degree. Algorithm MV: □; Algorithm G: ◯; bipartite graphs: solid; general graphs: hollow.

having equal size. Figure 15 shows the results of these runs in comparison with the previous runs on general graphs. Algorithm MV is shown with hollow symbols; algorithm G has solid symbols. Algorithm MV shows no significant change in its run time on bipartite graphs from its run time on general graphs. This is a consequence of the main contribution of the MV algorithm to the development of matching algorithms, which is its efficient handling of blossoms. In contrast, algorithm G's run time is significantly faster for the bipartite graphs above average degrees of 2.7, where blossoms are first discovered in general graphs. The discovery of blossoms by algorithm G considerably lengthens its execution time. It is interesting to note that without the formation of blossoms, the shape of algorithm G's run time plot takes on a similar shape to that of the plots for algorithm MV on general and bipartite graphs.

6. Triangle Graphs

This section contains the results of another of the "core" experiments for the DIMACS Implementation Challenge. The experiment consisted of running both algorithms on two types of triangle graphs: singly and triply connected. The structure of both types on a vertex set of size $3k$ begins with k vertex disjoint triangles labeled $0 \ldots k-1$. To complete the triply connected triangles, we add edges from each vertex to the corresponding vertex in the next larger triangle for triangles 0 to $k-2$. The edge set of a triply connected triangle consists of these edges on the vertex set $\{0 \ldots 3k-1\}$:

$(3i, 3i+1)$, $(3i+1, 3i+2)$, $(3i+2, 3i)$ $\quad 0 \le i \le k-1$

$(3i, 3i+3)$, $(3i+1, 3i+4)$, $(3i+2, 3i+5)$ $\quad 0 \le i \le k-2$

The singly connected triangle graphs have only one edge from a triangle to the next larger triangle. Such a graph on the same vertex set would have an edge set of:

$(3i, 3i+1)$, $(3i+1, 3i+2)$, $(3i+2, 3i)$ $\quad 0 \le i \le k-1$

$(3i + (i \bmod 3), 3i + 3 + (i \bmod 3))$ $\quad 0 \le i \le k-2$

Figure 16 shows the run times of the triangle graphs and triple triangle graphs for both the MV and G algorithms. These graphs were proposed as an example for which the MV algorithm would find many blossoms.

The run times and blossoms found by both the MV and G algorithms on these graphs are given in figure 17. For singly connected triangles, the general direction an alternating path may take from an unmatched vertex is either left or right. Once the direction is chosen, there is very little variation in the way the search may progress. Thus, algorithm G, with its fewer data structures and the simpler process by which it takes a step in the alternating search away from the unmatched vertex, outperforms algorithm MV. The performance of G is consistently better than the performance of MV by a factor of at least three for these graphs. With the triply connected triangles, where there is more freedom to choose an alternating search path, the performance of algorithm MV improves to slightly less than a factor of three worse than algorithm G. The structure in

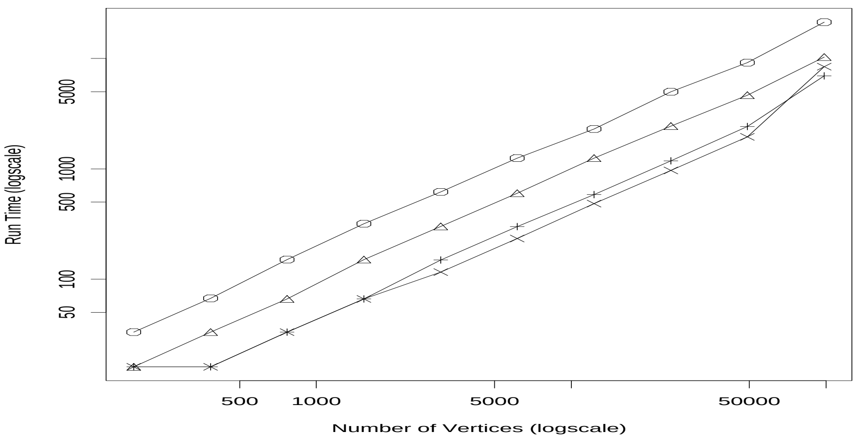

FIGURE 16. Experiment Tri, Triangle graphs' run times versus number of vertices on a log-log scale. MV on singly connected triangle graphs (*stg's*): ○; MV on triply connected triangle graphs (*ttg's*): △; G on STG's: +; G on TTG's: ×.

these graphs give quite a different outcome in the relative performance of the two algorithms than in the experiment using random graphs. There, for graphs of average degree four, algorithm MV had better performance than algorithm G (figures 10 and 11).

7. Conclusions

The experiments we ran have shown how the run times of the Micali-Vazirani and the Gabow algorithms for maximum matching in general graphs behave when the algorithms are executed on random graphs. We have also seen some of the combinatorial execution characteristics of these algorithms. For example, the number of phases which algorithm MV undergoes exhibits a sharp peak for random graphs which have an expected vertex degree of between 2.7 and 2.8. The number of blossoms discovered by algorithm MV takes on one of two classes. Either there are very few blossoms found, or there are a large number found, almost all of which are found during the last, unproductive phase.

The number of blossoms discovered by both algorithms running on random graphs show a sharp peak at expected degrees near 2.7. By expected degree 8, MV essentially discovered no more blossoms, while the blossom discoveries by G were reduced but almost level. The two algorithms exhibited opposite effects on their performance during the period for which large numbers of blossoms were being discovered. The peak in MV's run time near an expected degree of 3 is a result of the simultaneous peak in the number of phases. There is considerable overhead in initiating a phase, and the growth in the number of phases before

FIGURE 17. Experiment Triangles. Run times and blossom discoveries for algorithms MV and G on singly and triply connected triangle graphs.

Graph		Run Time (msec)		Blossoms	
Type	Triangles	MV	G	MV	G
Singly	64	33	16	41	31
Connected	128	67	16	82	64
Triangle	256	150	33	169	127
	512	317	66	338	256
	1024	617	149	681	511
	2048	1250	299	1362	1024
	4096	2300	583	2729	2047
	8192	4983	1183	5458	4096
	16384	9184	2416	10921	8191
	32768	21350	6966	21842	16384
Triply	64	16	16	0	0
Connected	128	33	16	0	0
Triangle	256	66	33	0	0
	512	150	66	0	0
	1024	300	116	0	0
	2048	600	233	0	0
	4096	1250	483	0	0
	8192	2433	966	0	0
	16384	4650	1949	0	0
	32768	10300	8432	0	0

average degree 3 explains the rise in run times while the decrease in the number of phases after degree 3 explains the corresponding plateau in run times.

The run times for algorithm G were highly affected by the formation of blossoms. At average degree 2.7 the run times rose sharply with the number of blossoms formed. By average degree 8 blossom discoveries were at half their peak level and had entered a long period of a slow decline. The run times over this same period declined at first and then entered an extended period of gradually increasing values.

References

1. J. Edmonds, *Paths, trees and flowers*, Canad. J. Math. **17** (1965), 449–467.
2. H. Gabow, *Implementation of algorithms for maximum matching on nonbipartite graphs.*, J. Assoc. Comput. Mach. **23** (1976), 221–234.
3. J. E. Hopcroft and R. M. Karp, *An $n^{5/2}$ algorithm for maximum matching in bipartite graphs*, SIAM J. Comput. **2** (1973), 225–231.
4. S. Micali and V. Vazirani, *An $o(|e|\sqrt{|V|})$ algorithm for finding maximum matching in general graphs*, Proc. 21st Annual IEEE Symposium on Foundations of Computer Science, 1980, pp. 17–27.

Department of Computer Science, University of Chicago, Chicago, Illinois 60637
E-mail address: crocker@cs.uchicago.edu