Greedy Matching Algorithms, an Experimental Study

Jakob Magun
Swiss Federal Institute of Technology Zürich

We conduct an experimental study of several greedy algorithms for finding large matchings in graphs. Further we propose a new graph reduction, called *k-Block Reduction*, and present two novel algorithms using extra heuristics in the matching step and *k-Block Reduction* for $k = 3$. Greedy matching algorithms can be used for finding a good approximation of the maximum matching in a graph G if no exact solution is required, or as a fast preprocessing step to some other matching algorithm. The studied greedy algorithms run in $O(m)$. They are easy to implement and their correctness and their running time are simple to prove. Our experiments show that a good greedy algorithm looses on average at most one edge on random graphs from $G_{n,p}$ with up to 10,000 vertices. Furthermore the experiments show for which edge densities in random graphs the maximum matching problem is difficult to solve.

1. INTRODUCTION

Let $G = (V, E)$ be a graph with vertex set V and edge set E and let $n = |V|$ and $m = |E|$. A *matching* is a subset M of the edge set E such that no two edges are incident to the same vertex. Edges in M are said to be *matched*. A set M of maximum size is called a *maximum matching* of G. The maximum matching problem is one of the classic problems of algorithmic graph theory. Applications of the maximum matching problem can be found in many textbooks, such as [Lawler 1976].

Several algorithms have been proposed to solve the maximum matching problem on general graphs. The first polynomial algorithm was found by Edmonds [Edmonds 1965]. The complexity of Edmonds' algorithm has been improved from $O(n^4)$ to $O(n^3)$ by Gabow [Gabow 1976] and using Union and Find data structure [Gabow and Tarjan 1983] to $O(mn)$. The algorithm with the best asymptotic running time $O(\sqrt{n}m)$ is the algorithm of Micali and Vazirani [Micali and Vazirani 1980].

Unfortunately the algorithm of Micali and Vazirani is rather difficult to understand and implement. It took nearly 10 years after the initial publication to prove its correctness [Vazirani 1989]. In LEDA, the popular library/framework for efficient data types and algorithms developed by Mehlhorn and Näher [Mehlhorn and Näher 1995], the algorithm of Gabow and Tarjan [Gabow 1976; Tarjan 1983] with running time $O(\alpha(n, m)mn)$ is used for the implementation of the maximum matching algorithm on general graphs.

A different approach is taken by greedy matching algorithms. Each matched edge is never reconsidered. This results in linear running time, if each edge is

Address: magun@inf.ethz.ch; Institut für Theoretische Informatik, ETH Zürich, CH-8092 Zürich, Switzerland
This work was supported in part by the Schweizerischer Nationalfond, Grant NF 2000-422 44.94.

inspected a constant number of times. Furthermore the algorithms are simple to understand and easy to implement. However the matching produced by a greedy matching algorithm is not necessarily a maximum matching. Greedy algorithms are often used as a preprocessing step for some other maximum matching algorithm. The preprocessing step of LEDA either uses the *SimpleGreedy* matching algorithm or a special heuristic that shrinks small blossoms [Mehlhorn et al. 1996]. The straightforward approach of a greedy algorithm, *SimpleGreedy*, chooses vertices at random. This basic algorithm can be improved considerably without sacrificing the $O(m)$ running time. In the past few years there has been much research in the field of greedy matching algorithms on random graphs, see [Goldschmidt and Hochbaum 1990; Dyer et al. 1993; Frieze et al. 1993; Aronson et al. 1997].

Although there are many theoretical papers on greedy matching algorithms, we do not know of any recent experimental study. The main contribution of our work is to close this gap by evaluating and comparing different greedy matching algorithms on random graphs and a small selected number of nonrandom graphs proposed in [Crocker 1993; Mattingly and Ritchey 1993].

As a further contribution we propose a new graph reduction step, the *k-Block Reduction*. We use this reduction for $k = 3$ together with a new heuristic to construct two new greedy matching algorithms, *MinRed12* and *BlockRed*. A further aim of our work is to determine the range of random graphs for which the maximum matching problem is hardest to solve. The difficulty of the maximum matching problem depends strongly on the average number of edges per vertex. Eg. if the average number of edges per vertex is one we have a very sparse graph with lots of isolated vertices and many chains. For such a graph, the matching problem becomes trivial.

The studied algorithms are implemented in C++ using LEDA, the library of efficient data types and algorithms [Mehlhorn and Näher 1995]. The source code of the implemented algorithms and the experimental test environment are available from `http://www.inf.ethz.ch/personal/magun`.

This paper is organized as follows. In Section 2 several graph reduction techniques are developed and two possible heuristics are presented. In Section 3 the algorithms used in the experiments are introduced. In Section 4 and 5 the experimental setup is described and the experimental results are discussed. The paper is concluded with final remarks in Section 6.

2. GRAPH REDUCTIONS AND MATCHING HEURISTICS

The following theorems prove certain structural properties of maximum matchings. These properties allow the transformation of a graph $G = (V, E)$ to a new graph G' with $|V'| < |V|$ and $|E'| < |E|$ without changing the size of a maximum matching in G' relative to the size of the maximum matching in G. Such graph reductions can be used to simplify the original input graph G without loosing any edges relative to a maximum matching of G. A subset $U \subseteq V$ is a *stable set* if and only if there are no edges between the vertices in U. Given a graph G and a matching M we call a vertex v *saturated* if and only if v is an endpoint of an edge in M. Let uv be an undirected edge between the vertices u and v and let $d(u)$ be the degree of vertex u. Let $d_T(u)$ be the degree of the vertex u in the subgraph T. By $N(U)$ we denote the set of vertices adjacent to $U \subseteq V$, excluding U itself. For $U = \{u\}$ we write $N(u)$. Given a graph G and a vertex set U, we define $G - U$ as the graph obtained by removing the vertices in the set U together with all edges incident to U. For a graph G and a vertex set W, we define G/W by shrinking the vertices w in W to a new vertex v, such that each edge wx is replaced by an edge vx. Parallel edges and loops are removed.

THEOREM 1. **Degree 1 Reduction [Karp and Sipser 1981]** *Given a graph $G = (V, E)$ and vertices $u, w \in V$ with $d(u) = 1$ and $N(u) = \{w\}$, there is always a maximum matching M on G with $uw \in M$.*

PROOF. Let M' be a maximum matching on G. If $uw \in M'$, then we have $M := M'$. If $uw \notin M'$ there has to be a matched edge e incident to w, as M' is a maximum matching. Replacing e with uw we have $M := M' - \{e\} + \{uw\}$. □

Using Theorem 1, a maximum matching M_1 on $G_1 = G - \{u, w\}$ can always be extended to a maximum matching $M := M_1 + \{uw\}$ on G by adding the edge $\{u, w\}$ to the matching M_1.

THEOREM 2. **Degree 2 Reduction [Karp and Sipser 1981]** *Given a graph $G = (V, E)$ and vertices $u, w_1, w_2 \in V$ with $d(u) = 2$ and $N(u) = \{w_1, w_2\}$, there is always a maximum matching M on G with $uw_1 \in M$ or $uw_2 \in M$.*

PROOF. Let M' be a maximum matching on G. If $uw_1 \in M'$ or $uw_2 \in M'$, then we have $M := M'$. If $uw_1 \notin M'$ and $uw_2 \notin M'$ there has to be a matched edge e in M' incident to w_1 or w_2, as M' is a maximum matching. If e is incident to w_1 we have $M := M' - \{e\} + \{uw_1\}$, otherwise e is incident to w_2 and we have $M := M' - \{e\} + \{uw_2\}$. □

Using Theorem 2, a maximum matching M_1 on $G_1 = G/\{w_1, w_2\} - \{u\}$ can always be extended to a maximum matching $M := M_1 + \{uw_1\}$ or a maximum matching $M := M_1 + \{uw_2\}$ on G. After a maximum matching for G_1 has been found, we expand the vertex w to the original vertices w_1 and w_2. Either the edge $\{uw_1\}$ or the edge $\{uw_2\}$ can then be included in the maximum matching M.

Definition 1. Let $G = (V, E)$ be a graph and (X, Y) an induced bipartite graph containing only edges between X and Y. X and Y are disjoint subsets of V. We call the bipartition (X, Y) *Hall-perfect* if and only if every bipartition $(X, Y'), Y' = Y - \{u\}, u \in Y$, has a matching that saturates every vertex in X. The Theorem of Hall [Hall 1935] states that (X, Y') has a matching that saturates every vertex in X if and only if $|N(S)| \geq |S|$ for all $S \subseteq X$.

COROLLARY 1. *The bipartition $B(U, W)$ is* Hall-perfect *if and only if $|N(U')| \geq |U'| + 1$ for all $U' \subseteq U$.*

PROOF. This follows directly from the Theorem of Hall and the definition of *Hall-perfectness.* □

THEOREM 3. **k-Block Reduction** *Given a graph $G = (V, E)$ and a stable set $U \subset V$ with $W = N(U)$ such that $k = |W| = |U| + 1$. If the induced bipartition (U, W) is* Hall-perfect*, there exists a maximum matching M on G such that $k - 1$ edges $uw, u \in U, w \in W$, are matched.*

PROOF. Given a maximum matching M' on G. Let S be the set of edges in M' incident to at least one vertex w in W. We have $|M' \cap S| = k$ or $|M' \cap S| = k-1$.

Case 1: If $|S| = k-1$ we set $T :=$ maximum matching on the bipartition (U, W) and $M = M' - S + T$.

Case 2: If $|S| = k$ we choose an edge $wx \in M, w \in W, x \in V - U - W$, and set $T := \{wx\}$ + maximum matching on the bipartition $(U, W - \{w\})$. A matching of size $k-1$ exists on $(U, W - \{w\})$ for every $w \in W$, as (U, W) is *Hall-perfect* and $M = M' - S + T$. $\square$

THEOREM 4. *Given a graph $G = (V, E)$ and a stable set $U \subset V$ with $W = N(U)$ such that $k = |W| = |U| + 1$. The induced bipartition (U, W) is* Hall-perfect *if and only if there is a spanning tree T in (U, W) with $d_T(u) = 2, \forall u \in U$.*

PROOF. Given a spanning tree T on the bipartition $B = (U, W)$ with $d_T(u) = 2, \forall u \in U$. Let B_i be a bipartition $B_i = B - \{w_i\} = (U, W - \{w_i\}), w_i \in W$ and T_{B_i} the spanning forest defined by restricting T to the bipartition B_i. The spanning forest T_{B_i} has t trees, $t = d_T(w_i)$, as t edges are removed from T. Each subset $U' \subseteq U, r = |U'|, s = |N_{B_i}(U')|$ defines a bipartition $B_i' = (U', N_{B_i}(U'))$ and the spanning forest T'_{B_i} defined by restricting T_{B_i} to the bipartition B_i'. The number of edges in the spanning forest T'_{B_i} is bound by the number of edges in a spanning tree $(r+s) - t$ and the number of existing edges $2r - t$. Combining the upper and lower bounds we have $(r+s) - t \geq |E_{T'_{B_i}}| \geq 2r - t$ and therefore $s \geq r$. Since we choose w_i arbitrarily, $|N_{B_i}(U')| \geq |U'|, \forall\ U' \subseteq U$ for every vertex $w_i \in W$, thus (U, W) is *Hall-perfect.*

We now show that every *Hall-perfect* bipartition has a spanning tree $T, d_T(u) \geq 2$ using induction over all bipartitions $B(U, W), |U| = k$. Let $2ST$ be a spanning tree T with $d_T(u) \geq 2$.

$k = 1$: For $k = 1$ we have $|U| = 1, |W| = 2$ and trivially $B(U, W) = T$.

k: Choose a pair of adjacent vertices $u \in U, w \in W$ and consider the bipartition $B(U - u, W - w)$.

Case 1: $B(U - u, W - w)$ is *Hall-perfect.* The $2ST$ in $B(U - u, W - w)$ can be extended with v, w to a $2ST$ on $B(U, W)$ as $deg(u) \geq 2, u \in U$.

Case 2: $B(U - u, W - w)$ is not *Hall-perfect.* In this case we first show that there exists a *Hall-perfect* subgraph $B(X, Y)$ with $N(X) = Y$. In a second step we shrink the bipartition $B(X, Y)$ to a new vertex, showing that the resulting graph is *Hall-perfect* again.

Step 1: We show that there exists a *Hall-perfect* subgraph $B(X, Y)$ with $N(X) = Y$. If $B(U - u, W - w)$ is not *Hall-perfect* the negation of Corollary 1 holds. Therefore a subset $U' \subseteq U$ exists such that $|N(U' - u) - w| < |U' - u| + 1$. On the other hand we know that $|N(U' - u)| \geq |U' - u| + 1$ for $B(U, W)$. Combining both inequalities, we have $|N(U' - u)| = |U' - u| + 1$ and $w \in N(U' - u)$. By Corollary 1, $B(U' - u, N(U' - u))$ is *Hall-perfect* together with $N(X) = Y$, hence $B(X, Y) = B(U' - u, N(U' - u))$.

Step 2: As $B(X, Y)$ is *Hall-perfect*, we can shrink $B(X, Y)$ to a new vertex y, by Block Reduction of $B(X, Y)$ in $B(U, W)$. We claim that the new bipartition is *Hall-perfect.* Let $B(U - X, W - Y + y)$ be the graph after the substitution. For $z \in W - Y$ a perfect matching in $B(U, W - z)$ induces a perfect matching in $B(U - X, W - Y + y - z)$ as $N(X) = Y$ holds. For $z \in Y$ a perfect matching in $B(U, W - z)$ induces a perfect matching in $B(U - X, W - Y)$ as $N(X) = Y$ holds. Hence $B(U - X, W - Y - y)$ is *Hall-perfect.* By induction hypothesis there exists a $2ST$ in $B(X, Y)$ and a $2ST$ in $B(U - X, W - Y + y)$, from which it is straightforward to construct a $2ST$ in $B(U, W)$ □

Using the equivalence stated in Theorem 4 we can detect a reducible situation by the existence of a spanning tree with $d_T(u) = 2, \forall u \in U$. A maximum matching on a tree T can then be easily found with Degree 1 reduction. Using Theorem 3 together with Theorem 4, a maximum matching M_1 on $G_1 = G/W - U$ can always be extended to a maximum matching on G by adding $k - 1$ edges of the tree T to M_1.

Let us consider the case $k = 3$ in more detail.

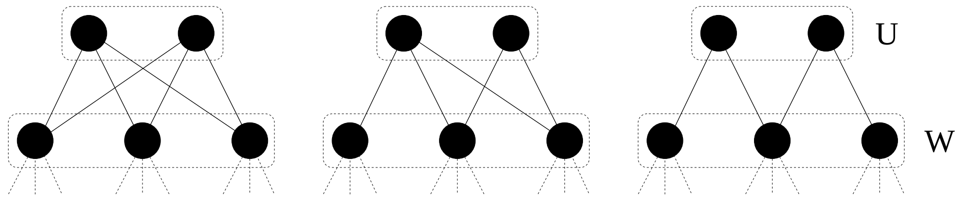

Fig. 1. All *Hall-perfect* situations for $k = 3$. a) 6 edges, b) 5 edges, c) 4 edges

THEOREM 5. *Given a graph $G = (V, E)$ and a stable set $U \subset V$ with $W = N(U)$, $k = 3$, $|U| = 2$ and $|W| = 3$. There are exactly three different bipartitions (U, W), see Figure 1 a), b) and c), which are* Hall-perfect.

PROOF. Each graph in Figure 1 a), b) and c) has a spanning tree T with $d_T(u) \geq 2$ and is therefore *Hall-perfect.* As a spanning tree in (U, W) has at least $(|U| - |W|) - 1 = 4$ edges, no graph with fewer edges can be *Hall-perfect.* All bipartitions, $|U| = 2, |W| = 3$, with 4, 5 or 6 edges are isomorph to one of the three bipartitions in Figure 1. □

The implemented 3-Block reduction algorithm can safely skip situations b) and c) as these cases are already handled by two consecutive Degree 2 reduction steps. The recognition of situation a) can be implemented in linear time using a trie to keep track of the vertex triples adjacent to the degree 3 vertices.

It is an open problem if the construction of the spanning tree with $d_T(u) \geq 2$ and the *k-Block Reduction* can be performed in their full generality in the $O(m)$ running time imposed by the greedy algorithms. The experiments show that the maximum matching problem becomes challenging if the average degree is in the range $c = 2.6 \ldots 3.8$. The *k-Block Reduction* for $k = 3$ improves the performance of the greedy algorithms for this range as it removes vertices of degree 3 through an exact graph reduction step. In Figure 1 the graphs a) has two vertices of degree 3 and graph b) has 1 vertex of degree 3 which will be removed.

The presented graph reductions simplify the input graph G without loosing any edges relative to the maximum matching. However at some stage of the algorithm, no reduction technique might be applicable. At such a stage the use of good heuristics can minimize the number of lost edges considerably. We present two such heuristics:

HEURISTIC 1. *When the first vertex u of the new matching edge is selected, choose a vertex of minimum degree.*

HEURISTIC 2. *When a vertex u for the new matching edge has been selected, choose an adjacent vertex v of low degree.*

Intuitively a vertex of low degree is harder to match than a vertex of high degree as it has fewer matching candidates: Matching a vertex of high degree can isolate many other vertices as many edges are removed from the graph, whereas matching a vertex of low degree will only remove few candidates from other vertices. Our experiments support this intuitive view.

3. ALGORITHMS

We investigate five different greedy algorithms with linear running time in our study. The algorithms have the following basic structure.

```
G_0 ← G = (V, E)
WHILE |E_i| ≠ Ø DO
   IF Degree 1 reduction step is possible AND enabled THEN
      Select vertex u and w from G_i.
      G_{i+1} ← G_i − {u} − {w}
   ELSE IF Degree 2 reduction step is possible AND enabled THEN
      Select vertex u and w_1, w_2 from G_i.
      G_{i+1} ← G_i/{w_1, w_2} − {u}
   ELSE IF 3-Block reduction step is possible AND enabled THEN
      Select vertices u_1, u_2 and w_1, w_2, w_3 from G_i.
      G_{i+1} ← G_i/{w_1, w_2, w_3} − {u_1, u_2}
   ELSE /* Matching Step */
      Select vertices u, v with uv ∈ E
      G_{i+1} ← G_i − {u, v}
   END
END
```

We have listed the algorithms in ascending performance.

—**SimpleGreedy** The matching step of this algorithm chooses a vertex u at random and matches u with an arbitrary adjacent vertex v. There is no reduction step. *SimpleGreedy* is used as the default algorithm in the preprocessing step of the maximum matching algorithm of LEDA.

—**MinGreedy** This algorithm chooses a node u of minimum degree in G_i according to Heuristic 1 and matches it with one of the adjacent vertices of u. The *MinGreedy* algorithm performs implicitly Degree 1 reductions. For each $1 \leq d < n$ the algorithm maintains a doubly linked list of all vertices in G_i with $d = d(v)$. The minimum degree of G_i is reduced at most by two in each step. Therefore the algorithm has $O(m)$ running time.

—**Red12** The *Red12* algorithm is an extension of *MinGreedy*. If the graph G_i has vertices v, $d(v) \leq 2$, the reduction step of *Red12* performs Degree 1 and Degree 2 reductions. If no Degree 1 or Degree 2 reductions are possible, the matching step of *MinGreedy* is applied.

We propose the following two new algorithms for finding a matching in a graph G.

—**MinRed12** The algorithm is very similar to *Red12*. The reduction step consists of Degree 1 and 2 reductions. The matching step is improved by applying Heuristic 2. Instead of matching the vertex u of minimum degree with one of the adjacent vertices, u will be always matched to an adjacent vertex of minimum degree. This extension still has $O(m)$ running time, as each edge in G is visited once to find an adjacent vertex of lowest degree

—**BlockRed** This algorithm improves the reduction step of *Red12* to perform 3-Block reductions in addition to the Degree 1 and 2 reductions. If no reduction step is possible for a certain G_i, the algorithm uses the matching step of *MinRed12*. For each new vertex u in G_i with $d_i(u) = 3$ and $d_{i-1}(u) \neq 3$, we sort the set $N(u)$ using the ordering of the vertices and insert $N(u)$ in a trie. If $N(u)$ is already contained in the trie for some other vertex v, $d(v) = 3$, we have $N(u) = N(v)$ and we can apply 3-Block reduction. Insert, delete and lookup in the trie take $O(1)$ time, as the depth is bound by $|N(u)| = 3$. At most n vertices are inserted in the trie giving a total running time of $O(m)$. The current implementation uses a standard LEDA dictionary instead of the trie. Using a dictionary with $O(\log n)$ time for `lookup`, `insert` and `delete`, gives a worst case bound of $O(m + n \log n)$ on the total running time.

4. EXPERIMENTS

We have evaluated five different greedy algorithms on random graphs from $G_{n,p}$ and on a collection of nonrandom graphs chosen from the literature [Crocker 1993; Mattingly and Ritchey 1993]. Initial experiments on the nonrandom graphs showed that the algorithms did not loose any edges at all. Except *SimpleGreedy* all algorithms found the maximum matching for these graphs. We have therefore restricted our experiments to random graphs.

The performance of the different algorithms is compared using the number of edges lost by each algorithm relative to the maximum matching of G. Measuring the number of lost edges has the advantage of being independent of factors such as cpu clock speed, memory cache effects, implementation details or compiler options. These factors often make runtime measurements very difficult to handle and to compare. The algorithms are implemented in C++ and use the library of efficient data types and algorithms LEDA to represent the graph and to implement the basic data structures.

4.1 Random Graphs

For the experiments on random graphs, we have chosen the $G_{n,p}$ model. A graph from $G_{n,p}$ has n vertices and every possible edge uv is independently included with probability p. The edge probability p is an arbitrary function. In order to keep the difficulty of the matching problem equal for different graph sizes n, we have kept the average number of adjacent vertices c independent of n by choosing $p = \frac{c}{n}$. If

p is chosen differently, eg. $p = \frac{c \log n}{n}$ the average number of edges per vertex grows for $n \to \infty$, thus simplifying the maximum matching problem for large graphs. If the function p grows for $n \to \infty$ faster or slower than the function $f(c, n) = \frac{c}{n}$ the maximum matching problem becomes easier for graphs above a certain size n. We have chosen $p = \frac{c}{n}$ for $c = 2.0, \ldots, 6.0$, as the results of the experiments indicate that matchings on random graphs with $c \geq 6.0$ are easily solved.

For each of the five algorithms we have generated graphs with $n = 500, \ldots, 10000$ vertices and average number of incident edges per vertex $c = 1.0...6.0$. For each triple $(f_i, n, c), i = 1 \ldots 5$ we have generated 600 graphs. For each graph, the difference between the size of the matching found by the algorithm f_i and the maximum matching of G is measured. We then calculate the expectation and the standard deviation of the number of edges lost over the 600 graphs. All experimental results are available in full detail from `http://www.inf.ethz.ch/personal/magun`.

5. EXPERIMENTAL RESULTS

We compare the five greedy algorithms for fixed average degree c and for fixed graph size n. Especially the experimental results of the two new algorithms *MinRed12* and *BlockRed* are compared to the other algorithms. Further we count the number of exact reduction steps made by the algorithms and how often a maximum matching is found.

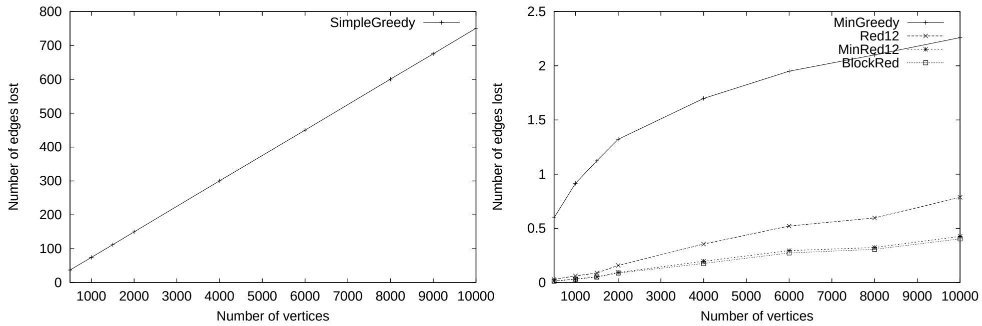

Fig. 2. Number of lost edges at $c = 3.0$

Figure 2 shows the number of edges lost for a fixed average degree $c = 3.0$. At $c = 2.8 \ldots 3.6$ the number of lost edges is maximal, as can be seen from Figure 3. Considering the absolute number of lost edges, *SimpleGreedy* looses on average around 750 edges for a graph with $n = 10{,}000$ vertices, whereas *MinGreedy* looses only 2.25 edges for the same graph size. This is an improvement by a factor of over 300. The number of lost edges is further reduced by *Red12* to 0.75 and with *MinRed12* and *BlockRed* to 0.5. *SimpleGreedy* looses $O(n)$ edges in the experiments. Tinhofer [Tinhofer 1984] has shown that the number of edges lost by *SimpleGreedy* is bound by $O(n)$ for graphs from $G_{n,\frac{c}{n}}$. The experiments indicate that this bound is sharp. For *MinGreedy* the experiments indicate that the number of lost edges is sublinear. Aronson, Frieze and Pittel [Aronson et al. 1997] show that on a graph from the $G_{n,m}$ model with $m = \lfloor cn/2 \rfloor$ a simplified *MinGreedy* algorithm looses on

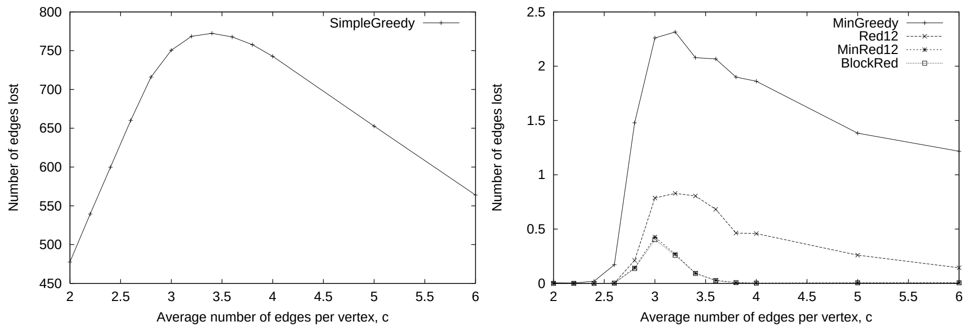

Fig. 3. Number of lost edges at $n = 10{,}000$

average a constant number of edges for $c < e$ and with high probability $n^{1/5+o(1)}$ edges for $c > e$. See Figure 3 b) for the case $c < e$. The asymptotic behavior of *Red12*, *MinRed12* and *BlockRed* is hard to determine from the experimental data, as the standard deviation of the number of lost edges is around 1 to 1.5 times the average number of lost edges for a graph of $n = 10{,}000$ vertices.

The average number of lost edges and the standard deviation for *SimpleGreedy*, *MinGreedy*, *Red12*, *MinRed12* and *BlockRed* are 750.63 ± 16.50, 2.26 ± 1.27, 0.78 ± 1.11, 0.42 ± 0.82 and 0.40 ± 0.80 for $c = 3.0$ and $n = 10{,}000$. The standard deviation of all algorithms except *SimpleGreedy* is around the average number of lost edges. This can be explained by the fact that very few edges are lost on average, while each run may only loose complete edges.

Figure 3 illustrates the difficulty of the matching problem for different values of c. The Figure shows the number of edges lost by *SimpleGreedy* and the other algorithms for a fixed graph size of $n = 10{,}000$. The number of lost edges has a clear peak at $c = 2.8 \ldots 3.6$ independent of the algorithm. All algorithms except *SimpleGreedy* show a sharp rise at $c = 2.4 \ldots 2.6$. Note that for *MinGreedy* the rise starts at $c = 2.4$, whereas for *Red12*, *MinRed12* and *BlockRed* this critical point is moved to $c = 2.6$. For $c > 3.2$ the number of lost edges declines again. *MinRed12* and *BlockRed* only loose edges in the range $2.6 \leq c \leq 3.8$, whereas the other algorithms keep loosing edges for $c \geq 6.0$.

In Figure 4 and 5 a) we have the number of edges lost by the two new algorithms, *MinRed12* and *BlockRed*. Figure 4 shows the number of edges lost by *MinRed12* for different values of c and the number of edges lost at $n = 10{,}000$ for different c compared to *Red12*. *MinRed12* looses fewer edges than *Red12* for all values of c. Moreover in the range $c = 3.2 \ldots 6.0$, where the number of lost edges is declining, the use of Heuristic 2 in *MinRed12* reduces the number of lost edges much faster than *Red12*. At $c = 4.0$ *MinRed12* nearly always finds a maximum matching, whereas *Red12* still looses as many edges as *MinRed12* at $c = 3.0$. This is a considerable improvement for the range $c > 3.0$.

Figure 5 a) shows the number of edges lost by *BlockRed*. Using *3-Block Reduction* reduces the number of edges lost compared to *MinRed12*. However the improvement is small. This can be explained by the fact that the number of executed reductions is very small, the relative number of k-Block reduction steps is less than 0.001,

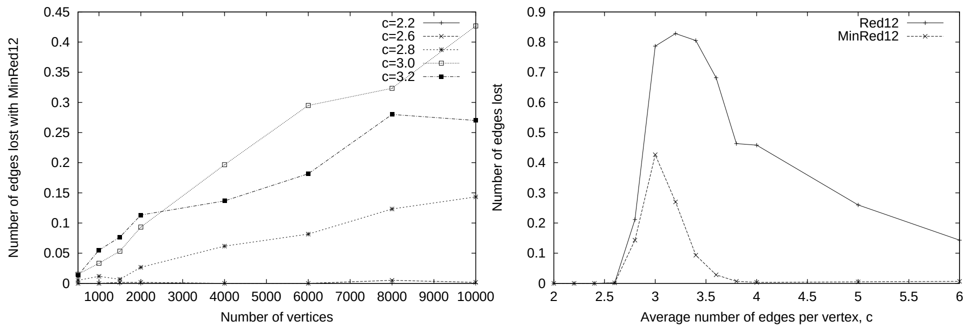

Fig. 4. Number of lost edges for algorithm *MinRed12*

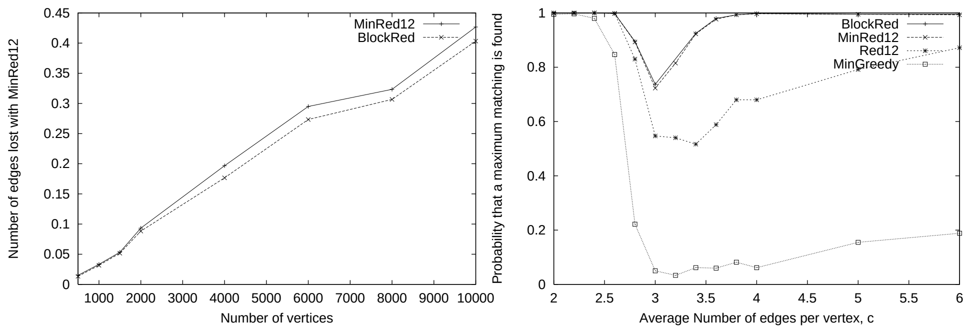

Fig. 5. a) Number of edges lost by *BlockRed* b) Probability to find a maximum matching

compared to the number of Degree 1 and Degree 2 reductions. Considering the small number of reductions, the difference in the number of lost edges indicates that the k-Block reduction steps merge situations where an edge is likely to be lost. Figure 5 b) shows the probability to find a maximum matching using a greedy algorithm. For the algorithms *BlockRed* and *MinRed12* and $c = 2.8 \ldots 3.4$, at least 75% of all runs will return a maximum matching. Outside this small range a maximum matching is found in over 95% of all runs.

Figure 6 a) depicts the relative number of times Degree 1 and Degree 2 reduction has been used in *BlockRed*. The relative number of k-Block reductions is below 0.001. We see that for $c = 2.8$ nearly all steps are exact reductions. The small remaining number of steps using Heuristic 2 are responsible for the lost edges. At $c = 2.6$ the Degree 1 reductions decline sharply, while the Degree 2 reductions slowly become dominating. The total number of exact reduction steps steadily declines from $c = 2.8$ on, while the number of lost edges still increases until $c = 3.0$. For $c > 3.0$ the number of lost edges starts declining. This is surprising, as the number of exact reduction steps declines as well. This suggests that the structural properties of the graphs simplify the maximum matching problem for $c > 3.0$. Figure 6 b) illustrates this point. At $c = 4.0$ the average size of the maximum matching is nearly $n/2$. From the point where a maximum matching is perfect on average, any

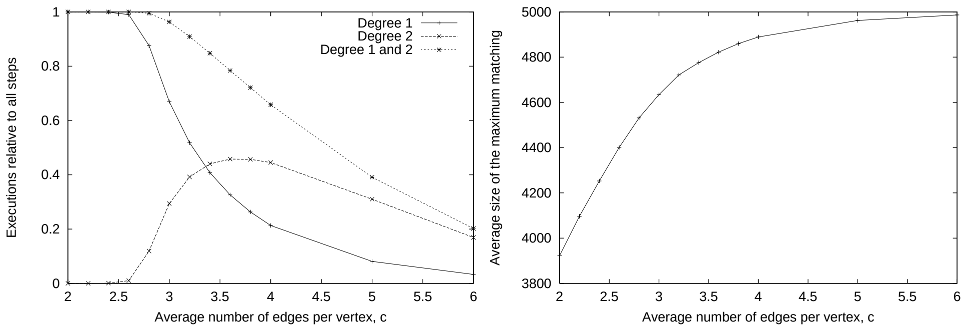

Fig. 6. a) Number of exact reduction steps b) Size of maximum matching at $n = 10{,}000$

increase of c will insert additional candidate edges into the graph, while the size of the maximum matching will remain constant at $n/2$. This simplifies the matching problem for $c > 4.0$. However the number of lost edges already starts declining for $c = 3.0$, where the maximum matching is far from being perfect. This suggests that there are other structural properties responsible for the peak at $c = 3.0$.

5.1 Running Times

In addition to the number of lost edges, we measured the absolute running times of the investigated greedy-algorithms. We have included the running times for the exact matching algorithm provided by LEDA as well. All experiments were run on a Sun Ultra 1, 143 MHZ, 64 MB Ram. All C source code was compiled with `g++` using `-O` optimization and LEDA Version 3.4. The running times in milliseconds for three different graph sizes are given in Figure 7.

	SimpleGreedy	MinGreedy	Red12	MinRed12	BlockRed	Leda
$n = 1{,}000$	0.031	0.032	0.055	0.054	0.085	0.150
$n = 4{,}000$	0.092	0.095	0.227	0.223	0.369	2.816
$n = 8{,}000$	0.198	0.204	0.511	0.493	0.826	14.537

Fig. 7. Average running times over 600 runs at $c = 3.0$

For graphs $n \geq 1{,}000$ all greedy matching algorithms have a competitive running time compared to LEDA. The performance advantage increases for large graphs as the LEDA implementation has asymptotic running time $O(\alpha(n,m)mn)$. The algorithmic steps needed for Degree 1 and Degree 2 reduction double the running time of *Red12* and *MinRed12* compared to *SimpleGreedy* and *MinGreedy*.

6. CONCLUSIONS

The experiments show that simple and robust greedy algorithms will find a nearly optimal matching on random graphs in $O(m)$. Such a matching provides an excellent starting point for augmenting-path based methods. The matching can thus be extended to a maximum matching by finding the nearly constant number of remaining augmenting paths in the graph. As the number of phases of the augmenting-path based algorithms are practically constant, the complicated implementation of the algorithm of Micali and Vazirani can be replaced with a simpler algorithm, such as the algorithm of Gabow.

All studied greedy algorithms improve the performance of the *SimpleGreedy* algorithm. The use of reduction techniques and matching heuristics will result in a significant reduction of the number of edges lost by the algorithm. The proposed *MinRed12* algorithm, using Heuristic 2, gives a considerable improvement for the range $c > 3.0$. For $c \geq 4.0$ it nearly always finds a maximum matching. The *k-Block Reduction* for $k = 3$ has a small influence on the number of lost edges as the number of situations where the reduction can be applied is small. However if only a few steps using Heuristic 2 can be replaced by block reduction steps, the number of lost edges is further reduced. For a wide range of edge densities, $c < 2.6$ and $c > 3.8$, a good greedy algorithm, such as *MinRed12* or *BlockRed*, will nearly always find the maximum matching of a random graph. For the remaining range $c = 2.6 \ldots 3.8$, the maximum matching problem becomes difficult and the algorithms start loosing edges. The most practical algorithm is *MinGreedy* extended by Heuristic 2. This combination allows a straightforward implementation as no vertices have to be merged by the algorithm.

Several interesting questions arise from our work. Is it possible to implement *k-Block reduction* for $k = 4$ in linear time or for all $k \geq 4$ in $O(m \log n)$? Would *k-Block reduction* for $k = 4$ or $k > 4$ give a significant improvement in the number of lost edges? Can graph reductions be found which are not always exact, but still reduce the average number of edges lost?

ACKNOWLEDGMENTS

I wish to thank my colleague Thomas Raschle for his helpful comments and suggestions during preparation of this paper.

REFERENCES

ARONSON, J., FRIEZE, A. M., AND PITTEL, B. G. 1997. Maximum matchings in sparse random graphs: Karp-sipser re-visited. preprint.

CROCKER, S. T. 1993. An experimental comparison of two maximum cardinality matching programs. *Network Flows and Matching, DIMACS Series in Discrete mathematics and Theoretical Computer Science 1*, 519–537.

DYER, M. E., FRIEZE, A. M., AND PITTEL, B. 1993. On the average performance of the greedy algorithm for finding a matching in a graph. *Ann. Appl. Probab. 3*, 526–552.

EDMONDS, J. 1965. Paths, tress, and flowers. *Canad. J. Math. 17*, 449–467.

FRIEZE, A. M., RADCLIFFE, A. J., AND SUEN, S. 1993. Analysis of a simple greedy matching algorithm on cubic random graphs. Volume 4 of *Proc. Fourth. Annual ACM-SIAM Symposium on Discrete Algorithms* (1993), pp. 341–351.

GABOW, H. 1976. An efficient implementation of Edmonds' maximum matching algorithm. *J. ACM 23*, 221–234.

GABOW, H. AND TARJAN, R. 1983. A linear time algorithm for a special case of disjoint set union. Volume 15 of *Proc. of the Ann. ACM Symp. Theory of Computing* (Boston, 1983), pp. 246–251.

GOLDSCHMIDT, O. AND HOCHBAUM, D. S. 1990. A fast perfect-matching algorithm in random graphs. *SIAM J. Discrete Math. 3*, 48–57.

HALL, P. 1935. On representatives of subsets. *J. London. Math. Soc. 10*, 26–30.

KARP, R. M. AND SIPSER, M. 1981. Maximum matchings in sparse random graphs. Volume 22 of *Proc. of the Ann. IEEE Symp. Foundations of Computer Science* (Nashville, 1981), pp. 364–375.

LAWLER, E. L. 1976. *Combinatorial Optimization: Networks and Matroids.* Holt, Reinehart and Winston, New York.

MATTINGLY, B. AND RITCHEY, N. P. 1993. Implementing an $O(\sqrt{N}M)$ cardinality matching algorithm. *Network Flows and Matching, DIMACS Series in Discrete mathematics and Theoretical Computer Science 1*, 539–556.

MEHLHORN, K. AND NÄHER, S. 1995. Leda: A platform for combinatorial and geometric computing. *Communications of the ACM 38*, 96–102.

MEHLHORN, K., NÄHER, S., AND UHRIG, S. 1996. Leda version 3.4-R source code, module _mc_matching.cc.

MICALI, S. AND VAZIRANI, V. 1980. An $O(\sqrt{V}E)$ algorithm for finding maximum matching in general graphs. Volume 21 of *Proc. of the Ann. IEEE Symp. Foundations of Computer Science* (Syracuse, 1980), pp. 17–25.

TARJAN, R. E. 1983. Data structures and network algorithms. *CBMS-NSF Regional Conference Series in Applied Mathematics 44.*

TINHOFER, G. 1984. A probabilistic analysis of some greedy cardinality matching algorithms. *Annals of Oper. Res. 1*, 239–254.

VAZIRANI, V. 1989. A theory of alternating paths and blossoms for proving correctness of the $O(\sqrt{V}E)$ general graph matching algorithm. Technical Report TR-89-1035 (Sept), Cornell University.