Randomized Local Approximations
with applications to the MAX-CLIQUE problem

R. Bar-Yehuda [*], *V. Dabholkar, K. Govindarajan and D. Sivakumar*
Department of Computer Science,
State University of New York at Buffalo,
Buffalo, NY 14260
e-mail: `sivak-d@cs.buffalo.edu`

August 10, 1993

Abstract

We present a heuristic scheme for finding a clique of maximum size or weight. Our heuristic scheme uses approximation techniques for the weighted vertex cover problem, due to R. Bar-Yehuda and S. Even [BE85]. The approach is based on the notion of making incremental progress towards finding a clique. At each step of our algorithm, one or more *local optimization* techniques are attempted, and when these techniques do not make progress, we perform *local approximations*. In the local approximation step, the algorithm selects a heuristic from a set of heuristics, depending on the characteristics of the graph at that stage. Once a solution is constructed, we attempt to iteratively refine the solution. Randomization plays a key role at various steps of our algorithm.

1 Introduction

Finding a clique of maximum size or maximum weight (MAX-CLIQUE) in a graph is a well known NP-hard problem [GJ79]. In fact, results of [ALMSS92] indicate that there is an $\epsilon > 0$ such that it is NP-hard to approximate MAX-CLIQUE to within a factor of n^ϵ. In this paper, we present a scheme where we combine various heuristics to find cliques of large size or weight.

The motivation for our approach is the following: a clique K of maximum size or weight in a graph $G = (V, E)$ with weight function $w{:}V \rightarrow N^+$ is an independent set I of maximum size or weight in the complement graph $\overline{G} = (V, \overline{E})$, $w{:}V \rightarrow N^+$, and $V \backslash I$ is a vertex cover of minimum size or weight in $\overline{G}$. An appealing property of the vertex cover problem is that there are efficient approximation algorithms for this problem. In particular, R. Bar-Yehuda and S. Even [BE85] give an approximation algorithm for the

[*] On leave from Department of Computer Science, Technion, Haifa, Israel 32000

weighted vertex cover problem that guarantees an approximation ratio no worse than $2 - (\log\log n / 2\log n)$. Therefore, to approximate MAX-CLIQUE, we can use efficient heuristics for finding a minimum vertex cover.

It is evident that the approximation ratio is not preserved when we find an independent set through an algorithm for vertex cover ([GJ79], page 134). However, this does not imply that approximation heuristics for the vertex cover problem are useless in approximating the maximum independent set. We argue that for practical inputs, heuristics for the vertex cover problem are useful tools in approximating the maximum independent set, and substantiate our claim in the rest of this paper.

Our scheme uses the framework of local optimizations (cf. [NT75]) and local approximations (see, for example, [BE85]), together with a number of greedy and randomized heuristics. Briefly described, a *local optimization* step in an optimization algorithm is one in which a partial solution is found. This partial solution has the property that there is an optimal solution which extends this partial solution. Ideally, one would like to find locally optimal solutions in each step and construct an optimal solution eventually. Obviously, this idea does not work if we want to solve the optimization problem in polynomial time (unless P = NP). Typically, local optimization algorithms have the following drawbacks : (a) they fail to make progress after a few iterations, and (b) to be successful, they require the input to have specific structure or properties. A *local approximation* step is used when the local optimization technique fails to make progress due to either of these reasons. Local approximation is performed by considering a small part of the input, and finding an approximate solution to this part "without sacrificing too much" in the overall solution. Besides providing an approximation algorithm for finding maximum cliques, our approach illustrates several techniques and paradigms for tackling combinatorial optimization problems.

In the next section, we introduce the necessary terminology and discuss the motivations for our approach. In section 3, we present the algorithm in full detail. In section 4, we list our findings based on experiments, and analyze the merits and demerits of our heuristics based on experimental observations. This section also contains the experimental results indicating the performance of our approach to the MAX-CLIQUE problem. Finally, we summarize our contributions and outline future possibilities in section 5.

2 Preliminaries and Motivations

2.1 Notations and definitions

In this section we review some standard graph-theoretic definitions and introduce basic notation that will be used throughout this paper.

Given a graph $G = (V, E)$, we will use $N(v)$ to denote the *neighborhood* of a vertex v and $d(v)$ to denote the vertex *degree* of v. That is,

- $N(v) = \{w \in V | (v, w) \in E\}$, and

- $d(v) = |N(v)|$.

When a weight function $w:V \rightarrow N^+$ is defined on the vertices, the *dw-ratio* of a vertex v is $d(v)/w(v)$.

For a set S of vertices in G, the graph $H = G|_S$ is the subgraph of G induced by the vertices in the set S and $N(S)$ denotes the neighborhood of the vertices in S. That is,

- $G|_S = (S, \{(i,j) \in E | i,j \in S\})$,
- $N(S) = \bigcup_{v \in S} N(v)$.

A *clique* in a graph G is a complete subgraph K of the graph G. An *independent set* in a graph G is a set I of vertices no two of which are adjacent to each other. A *vertex cover* in a graph $G = (V, E)$ is a set VC of vertices such that $V \backslash VC$ is independent.

Clearly, a clique K_{max} of maximum cardinality in $G = (V, E)$ is an independent set of maximum cardinality in the complement graph $\overline{G}$, and $V \backslash K_{max}$ is a vertex cover of minimum cardinality in $\overline{G}$. Observe that the independent set and the vertex cover problems are natural *duals* of one another. Analogous descriptions apply to the case where a weight function $w:V \rightarrow N^+$ is defined on the vertices of the graph.

Given a minimization problem Π, an approximation algorithm A for Π is said to achieve an *approximation ratio* R_A if, for all instances I_Π of Π, $C/C^* < R_A$, where C is the cost of the solution provided by A on input I_Π and C^* is the cost of an optimal solution.

2.2 Local Optimization and Local Approximation

We now introduce the notions of local optimization and local approximation. Given any optimization problem Π, consider an algorithm A for solving Π (wlog. A could be an exact algorithm or an approximation algorithm). At any step, we say that algorithm A performs a *local optimization* if the partial solution S constructed in that step satisfies $S \subseteq S^*$, for some optimal solution S^* of Π. Note that the term "local optimization" has also been used in literature to denote improving a solution S to a solution S' by performing local operations. An algorithm is said to perform a *local approximation* step if, in that step, the algorithm restricts its attention to only a portion of the input and finds an approximate solution to the sub-problem induced by this portion of the input.

We illustrate the above concepts with respect to the vertex cover problem. Given $G = (V, E)$, a local optimization step would be to find a subset $S \subseteq V$ such that there exists an optimal vertex cover V^* and $S \subseteq V^*$. For example, if we find a leaf-vertex v which has one edge (u, v) incident on it, we can perform local optimization by taking the vertex u to the vertex cover. Let $T = \{V_1, V_2, V_3\}$ be a triangle in G. Clearly, any optimal vertex cover V^* of G has to contain at least two vertices from T. A local approximation step would be to take all the vertices in T to the vertex cover. Another example of local approximation for vertex cover is to identify a clique K_p in G, and adding to the vertex cover all the p vertices in K_p. The attractive characteristic of local approximation is brought out by the following theorem from [BE85]:

Theorem 1 ([BE85]) *Let $\hat{G}$ be an unweighted graph of $\hat{n}$ vertices, whose optimum vertex cover contains $\hat{c}^*$ vertices. Define $\hat{r} = \hat{n}/\hat{c}^*$. Let A be an approximation algorithm for the vertex cover problem with approximation ratio R_A. Let $L_{\hat{G}}$ be the following algorithm: given a graph G, identify a subgraph $\tilde{G} = (\tilde{V}, \tilde{E})$ of G that is isomorphic to $\hat{G}$, and return $\tilde{V} \bigcup A(G \backslash \tilde{G})$. Then*

$$R_{L_{\hat{G}}} \leq \max(\hat{r}, R_A).$$

The above theorem guarantees that the overall approximation ratio is no worse than the worst local approximation ratio. For example, a local approximation by triangle-elimination guarantees an approximation ratio no worse than 1.5. Eliminating a clique of size p guarantees a local ratio of $p/(p-1)$.

2.3 Motivations

Here we briefly describe the reasons why we decided to pursue an approach based on approximating the vertex cover problem. As mentioned before, the [BE85] approximation algorithm for the weighted vertex cover problem achieves an approximation ratio of $2 - (\log\log n/2\log n)$. The ratio is not preserved if we construct an independent set using this algorithm. In fact, Arora and Safra [AS92] have proved that no constant ratio is possible, unless P = NP.

Nevertheless, we pursue the idea for the following reasons. First, we note that the approximation ratio in [BE85] is based on worst-case analysis, and it is not unreasonable to expect the algorithm to work well in practice, at least for specific families of graphs. Secondly, the vertex cover problem and the independent set problem are natural duals of one another. Therefore, using an approach based on vertex cover helps us take advantage of being able to devise efficient heuristics in either domain. Another advantage of this duality is as follows: given a graph G, to approximate the vertex cover of the complement graph $\overline{G}$, we can find large cliques in $\overline{G}$ and perform local approximations. To find large cliques in $\overline{G}$, we can use the algorithm recursively.

3 Description of the algorithm

In this section, we give a complete description of our algorithm. We first describe the various components of our algorithm and then proceed to discuss their combinations. We feel that each of these techniques merit discussion in their own right, and are of sufficiently general interest in solving hard combinatorial problems. In the rest of this paper, we will mostly talk about identifying independent sets in graphs.

3.1 The Nemhauser-Trotter local optimization algorithm

Our main technique for performing local optimization is due to G.L. Nemhauser and L.E. Trotter [NT75]. This technique was used before to obtain the $2 - (\log\log n/2\log n)$ approximation algorithm for the weighted vertex cover problem [BE85].

Algorithm NT

Input:	$G = (V, E)$, $w{:}V \rightarrow N^+$
Step 1:	Define a bipartite graph $B = (V, V', E_B)$ where $E_B = \{(u, v') \mid (u, v) \in E\}$.
Step 2:	$VC_B \leftarrow VC^*(B)$.
Output:	$VC_0 \leftarrow \{u \mid u \in VC_B$ AND $u' \in VC_B\}$
	$V_0 \leftarrow \{u \mid u \in VC_B$ XOR $u' \in VC_B\}$
	$I_0 \leftarrow \{u \mid u \notin VC_B$ AND $u' \notin VC_B\}$

Given a graph $G = (V, E)$, we first create a bipartite graph by making two copies of V and adding two copies of each edge in E, as given in Step 1 of Algorithm NT. Note that the number of edges in B is twice the number of edges in E. Step 2 of Algorithm NT computes the minimum vertex cover $VC^*(B)$ of the bipartite graph B. To solve the minimum vertex cover problem on the bipartite graph produced, we can use maximum matching algorithms for the unweighted case, and maximum flow algorithms for the weighted case. The worst case time complexity of this step is, therefore, $O(E\sqrt{V})$ for the unweighted case and $O(V^3)$ for the weighted case. The outputs VC_0 and V_0 of Algorithm NT satisfy the following properties:

Theorem 2 ([NT75])
(i) *If a set $D \subseteq V_0$ covers $G|_{V_0}$ then $VC = D \bigcup VC_0$ covers G.*
(ii) *There exists a minimum vertex cover $VC^*(G)$ such that $VC_0 \subseteq VC^*(G)$, and there exists a maximum independent set $I^*(G)$ such that $I_0 \subseteq I^*(G)$.*
(iii) $w(VC^*(G|_{V_0})) \geq w(V_0)/2$

For our task of identifying a large independent set, we say that NT makes progress whenever $VC_0 \bigcup I_0$ is non-empty. We repeatedly apply NT till it ceases to make progress, collecting as many vertices from VC_0 and I_0 as possible.

Our experiments indicate that NT makes progress only when the graph has one or more vertices that are much better than the others in terms of the dw-ratio. Such vertices occur commonly in very sparse graphs and graphs where the weights of vertices are highly disparate and they occur very rarely in uniform random unweighted graphs. Therefore, the NT algorithm is particularly attractive for the former class of graphs, more so for instances from real world problems where the vertices often have a "natural" weight defined on them.

3.2 A randomized divide-and-conquer heuristic using NT

Here we describe a randomized divide-and-conquer heuristic for identifying partial vertex covers, using Algorithm NT. This heuristic is motivated by the idea of *randomized rounding*, as described in [Rag90], and it exploits the fact that Algorithm NT works well for sparse graphs.

Given a dense graph $G = (V, E)$, we partition it into k graphs $G_1, G_2, \ldots, G_k$, where $G_i = (V, E_i)$ such that $\bigcup_{i=1}^{k} E_i = E$ and $\forall i \neq j, E_i \bigcap E_j = \phi$. This partitioning is performed by assigning each edge to a randomly chosen partition. We choose k large enough to make each graph very sparse so that we can use NT to find small vertex covers. Once we find the vertex cover for each G_i, we combine these vertex covers probabilistically to find a good partial vertex cover for G. We propose three heuristics to perform this probabilistic combination.

(a) For each vertex v, let $count(v) :=$ the number of graphs G_i for which v was included in the vertex cover of G_i, and define $p(v) := count(v)/k$. With probability $p(v)$, add v to the vertex cover for G.

(b) This heuristic attempts to eliminate a drawback of (a), viz., each vertex is added to the vertex cover independently of its neighbors. For example, suppose vertex 17 in G has neighbors 1, 4, 7, 9 and 45, and let $p(17) = 0.6$. If vertices 1, 4 and 9 have been added to the vertex cover, vertex 17 is not a very attractive choice to be added to the vertex cover. Therefore, we modify the heuristic as follows: define an initial probability $p_0(v)$ for each vertex as in (a). Let $d_0(v)$ be the initial degree of vertex v in G and let $d(v)$ be its degree when it is considered for inclusion in the vertex cover. Add v to the vertex cover with probability $p(v) = p_0(v)d(v)/d_0(v)$. In the example given, $p_0(17) = 0.6, d_0(17) = 5$ and $d(17) = 2$; therefore, vertex 17 is added to the vertex cover with probability 0.24.

(c) This heuristic attempts to construct a vertex cover by considering each edge (u, v) and adding one of u or v with probabilities $p'(u) = p(u)/(p(u) + p(v))$ and $p'(v) = 1 - p'(u)$, respectively.

We observed experimentally that heuristic (b) performs best of the three for dense random graphs. Heuristic (c), although intuitively appealing because it proceeds through a "fractional cover", did not perform well on random graphs. The applicability of these heuristics to different families of graphs remains to be studied. Another open issue is to identify better estimates for the probability $p(v)$.

3.3 Local approximations

Local approximation involves identifying and eliminating a particular type of sub-graph; for our experiments, we used triangles, cliques and edges. The choice of any one depends on the density and other characteristics of the graph at any stage of the algorithm.

3.3.1 Cliques: the ping-pong game

When the graph at any stage is sufficiently dense, we complement the graph and identify a large independent set in the complement. To identify large independent sets in the complement, we use our algorithm recursively. This is a very attractive feature of the scheme of local optimizations and local approximations – when no progress could be made on a dense graph, it lets us switch to the complement graph where progress is more likely to happen. An optional parameter to the algorithm specifies the threshold density above which this local approximation is performed.

3.3.2 Triangles and edges

When the graph is not sufficiently dense, identifying large cliques could be an expensive operation. Therefore, we attempted to eliminate triangles (cliques of size 3) for local approximation. We sorted the vertices on their dw-ratio, and picked triangles such that the sum of the dw-ratios of the three vertices is maximum. However, eliminating a few triangles was not sufficient for NT to make any progress, and even for graphs with density 0.1, we had to remove almost all triangles before NT made any progress. Moreover, triangle elimination is a worst-case $O(n^3)$ operation (without using special data structures, etc.); therefore, we decided to discard the triangle elimination step.

Unlike triangles, edges could be easily identified and eliminated. Here again, we sort the vertices on their dw-ratio and pick the edges to be removed in a greedy fashion.

3.4 Randomized local approximations

If, in the local approximation step, we identify a clique of size k, taking all k vertices to the vertex cover guarantees an approximation ratio no worse than $k/(k-1)$. However, it is possible to improve upon this ratio. We begin with the following observation:

> The local ratio becomes closer to 1 if k is very large. Since local approximation has to be a very fast operation, we might not be able to identify large cliques always. For small values of k, however, the ratio becomes unattractive for approximating the independent set. Moreover, even for large values of k, it suffices to include only $k-1$ vertices in the vertex cover.

The foregoing observation leads to the idea of randomized local approximation. When we identify a clique of size k, rather than adding all k vertices to the vertex cover, we add each vertex independently with probability $(k-1)/k$. The expected number of vertices added is $k-1$ which is sufficient to cover the edges of the clique.

From our experiments, we observed that this technique performed considerably better than deterministic local approximation.

3.5 Branch-and-Bound

After a number of local optimization and local approximation steps, if the graph becomes sufficiently small, we employ a straightforward branch-and-bound algorithm to find the maximum independent set.

3.6 Putting the pieces together

At this point, our algorithm looks as follows:

```
Algorithm BDGS
Input: G = (V, E), w:V→N+

while TRUE do
    Step 1: if G is small enough then
                        perform Branch-and-Bound;
                        return
            else if G is sparse enough then
                        repeat Algorithm NT until no progress
            else if G is moderately sparse then
                        apply randomized divide-and-conquer
                        using Algorithm NT (cf. Section 3.2)
            else if G is dense enough then
                        perform randomized local approximation
                        by finding large cliques
    Step 2: if no progress made in Step 1 then
                        perform randomized local approximation
                        by removing edges
end while
```

Once we obtain a solution using Algorithm BDGS, we attempt to improve it as follows. On input $G = (V, E)$, let I be the independent set constructed by Algorithm BDGS. We construct an independent set $I' \subseteq I$ by adding each vertex of I to I' with probability 0.5. The expected size of I' is half the size of I. Let $S = V \backslash (I' \cup N(I'))$ and $\hat{G} = G|_S$, where $N(I')$ denotes the neighborhood of the vertices in I'. Observe that $I' \bigcup \hat{I}$ is an independent set of G, for any independent set $\hat{I}$ of $\hat{G}$. We exploit this idea by applying Algorithm BDGS recursively on $\hat{G}$, and iterate this process a number of times.

4 Experiments and analysis

In this section, we present experimental results of our algorithm on two sets of instances of the MAX-CLIQUE problem. The first set of instances are drawn from the family $G(n; p)$ – random graphs with n vertices where each edge occurs independently with probability p. The second set of experiments were conducted with the benchmark instances supplied by DIMACS. We performed our experiments on Sun SPARC-10 workstations operating at 80 million instructions per second.

4.1 Random graphs

We tested our algorithm on random graphs $G(n;p)$ for $n = 400$, 700 and 1000 and $p = 0.2$, 0.4, 0.6 and 0.8. We present our results averaged over five instances from each set $G(n;p)$. To exploit the advantages of randomization, we ran the algorithm three times on each instance and have reported the best performance. When reporting the CPU time, however, we have *not* reported the sum of the running time for the three trials. Our results are summarized in Table 1. The algorithm was run in various "modes", each of which is described below:

- **RLA (edges) only**: only randomized local approximations on edges are performed.
- **RLA (edges) and NT**: randomized local approximations on edges are performed, and Algorithm NT was applied whenever the graph had $|E| < 4|V|$.
- **RLA (edges) and RR/CL**: randomized local approximations on edges are performed, and one of (a) the randomized divide-and-conquer heuristic using NT or, (b) randomized local approximations on cliques was performed, depending on the density of the graph. If $|E| < 0.3|V|^2$, the randomized divide-and-conquer heuristic (RR) is attempted, and if $|E| > 0.5|V|^2$, randomized local approximation with cliques is attempted.

The entry $E(\omega(G(n;p)))$ gives the value $z(n;p)$, which is an approximation to the expected size of the largest clique in $G(n;p)$ (cf. [BE76]).

n	p	$E(\omega(G(n;p)))$	RLA (edges) only	CPU time (secs)	RLA (edges) and NT	CPU time (secs)	RLA (edges) and RR/CL	CPU time (secs)
400	0.2	7.19	6.00	146.02	6.20	140.70	6.20	390.80
400	0.4	10.65	9.80	84.74	9.80	81.88	9.40	160.32
400	0.6	16.02	15.20	45.14	15.20	45.94	15.00	55.32
400	0.8	27.96	28.40	21.16	28.40	30.46	28.20	126.02
700	0.2	7.78	7.00	761.78	6.60	790.38	6.00	1531.46
700	0.4	11.68	10.00	436.80	10.00	449.04	10.00	828.90
700	0.6	17.86	16.80	214.08	16.60	219.94	16.80	258.86
700	0.8	32.18	30.40	82.02	31.20	121.80	31.40	691.60
1000	0.2	8.16	7.00	2594.52	7.00	2254.60	-	-
1000	0.4	12.34	10.80	1347.34	10.80	1284.08	-	-
1000	0.6	19.05	17.40	602.84	17.20	601.82	17.40	696.62
1000	0.8	34.90	31.60	207.46	32.80	295.66	32.20	2103.18

Table 1: Experimental results on unweighted random graphs $G(n;p)$

Since our algorithm works largely on the complement graph (to find an independent set), the running times are much better for dense graphs. In most cases, performing randomized local approximations on edges yields the best results. Algorithm NT makes a marginal improvement in very dense graphs ($G(n; 0.8)$). This could be explained as follows: when the input graph G is very dense, the complement $\overline{G}$ is very sparse, and Algorithm NT is invoked after a few applications of randomized local approximations. If, on the other hand, $\overline{G}$ is dense to start with, randomized local approximations add more vertices to the vertex cover, limiting the contribution that NT could make. The randomized divide-and-conquer heuristic and local approximation using cliques did not improve the performance for this family of graphs. Finally, some entries for $G(1000; 0.2)$ and $G(1000; 0.4)$ are incomplete because of memory limitations.

4.2 DIMACS Benchmarks

We ran our algorithm on the set of benchmark instances made available by DIMACS ([DI93]). The results are summarized in Table 2. Since the size of the largest clique is not known for all the benchmark graphs, our analysis is mostly restricted to the relative performance of the four heuristics.

CFat: On graphs from this family, three of the four heuristics (RLA(edges), RLA(edges) and NT, RLA(edges) and RR) performed well. RLA with cliques seemed to perform much worse than the others. It can be observed that the heuristic based on randomized divide-and-conquer (RR) performed the best on all graphs in this family.

Hamming: Graphs in this family are highly regular and symmetric. All our heuristics perform very well on graphs from this family. For the denser graphs, algorithm NT was observed to perform very well. Specifically, when NT was used in conjunction with RLA on edges, the algorithm converges to the best value much faster (this fact is not reflected in the timings because the timings are given for a fixed number (30) of iterations).

Johnson: This is another family of highly regular graphs, and again, all our heuristics performed very well. It could be noted, however, that the simplest heuristic (RLA on edges) was sufficient to achieve the best performance for each graph. We further observed that algorithm NT did not make any contribution.

Keller: All four heuristics got a clique of size 11 on the smallest graph in this family. On the next larger instance, the best performance was observed for RLA with edges. The largest clique in the graph has 27 vertices, and the algorithm reported a clique of size 24. Once again, all other heuristics did not improve the performance. Therefore, we only attempted RLA on edges for the largest graph in this family and found a clique of size 37.

Sanchis: Our algorithm did not perform well on this specially constructed family of test graphs. Our conjecture is that these graphs have very few cliques which are much larger than the expected clique size. Our probabilistic approach could identify the large cliques for two of the eleven cases. None of NT, RR or RLA with cliques had any impact on the performance of the algorithm.

Graph	Optimum	RLA (edges) only	CPU time (secs)	RLA (edges) and NT	CPU time (secs)	RLA (edges) and RR	CPU time (secs)	RLA (edges) CL	CPU time (secs)
c-fat200-1		12	12.3	12	13.4	12	22.9	12	17.1
c-fat200-2		23	10.4	24	11	24	20.1	22	15.8
c-fat200-5		56	6.3	57	6	58	11.3	42	31.1
c-fat500-1		14	310.5	14	518.6	14	494.9	14	385.2
c-fat500-10		120	112.2	123	195.8	126	191.9	71	352.1
c-fat500-2		23	460.9	26	296.2	26	296.9	24	451.7
c-fat500-5		49	208.1	64	217.3	64	350.5	40	296
hamming10-2		512	99.2	512	96.1	173	109	512	84.5
hamming10-4		32	126.4	34	255.7	32	260.8	33	288.2
hamming6-2	32	32	0.2	32	0.2	32	0.3	32	0.3
hamming6-4	4	4	0.4	4	0.4	4	0.7	4	0.6
hamming8-2	≥ 79	128	3.7	128	5.2	128	5.8	128	4.9
hamming8-4	16	16	7.5	16	8.5	16	13.2	16	7.6
johnson16-2-4	8	8	1.2	8	1	8	2.1	8	1.1
johnson32-2-4		16	17	16	31.6	16	41.3	16	27.4
johnson8-2-4		4	0	4	0	4	0.1	4	0
johnson8-4-4	14	14	0.3	14	0.2	14	0.8	14	0.3
keller4	11	11	2.9	11	3	11	4.9	11	2.9
keller5	27	24	91	22	182.6	21	249	22	145.7
keller6	≥ 59	37	5400	-	-	-	-	-	-
san1000	15	8	484.5	8	515.8	8	592.2	8	507
san200_0.7_1	30	15	4.1	15	8.5	15	9.6	15	8.5
san200_0.7_2	18	12	3.8	12	7.9	12	8.4	12	6.4
san200_0.9_1	70	43	2.2	45	10.4	39	4.7	45	10
san200_0.9_2	60	36	4.1	36	12.7	35	4.6	36	9.1
san200_0.9_3	44	36	2.4	44	7.2	36	5	34	9.2
san400_0.5_1	13	7	29.8	7	30.6	7	51.6	7	31.4
san400_0.7_1	40	20	19.3	20	48.7	20	48.5	20	48.2
san400_0.7_2	30	15	19.2	15	36.9	15	49	15	38.7
san400_0.7_3	22	12	18.7	12	30	12	48.3	12	31.1
san400_0.9_1	100	50	10.9	51	55.2	49	23.9	100	16.7
sanr200_0.7		18	6	17	3.8	17	7.2	18	4.2
sanr200_0.9		41	2.4	41	8.9	39	4.3	42	5.6
sanr400_0.5		13	35.9	12	33.5	12	41.5	12	37.9
sanr400_0.7		19	19.2	20	21.5	19	49.9	19	21.5

Table 2: Experimental results on DIMACS benchmark instances

5 Concluding Remarks

• Randomized local approximations is a powerful technique in finding approximate solutions to hard combinatorial problems like MAX-CLIQUE. This technique has the merits of being very simple and efficient, yet general enough to be applied for many problems.
• The idea of discarding each vertex from the independent set with probability 0.5 and iterating Algorithm BDGS proved very useful in every instance. This offsets to some extent the effect of greedier local decisions made early in the algorithm.
• We observed that NT did not make as much progress as expected both in random as well as specific benchmark instances. Whenever NT made progress, the convergence was achieved very fast.
• In algorithm BDGS, the branch and bound method was called when the graph size was reduced to 30 or fewer vertices. This threshold could have been increased. In fact, we observed that branch and bound alone gave optimum results in reasonable time for the graph $G(400; 0.5)$. But since the idea was to see how useful randomized approximations and algorithm NT are in finding large cliques, we decided to perform branch and bound on only very small instances.
• Replacing NT by other local optimization techniques is a potentially useful idea.
• The performance of the four heuristics (and their combinations) on these and other families of graphs remains to be studied in greater detail. More importantly, we could not evaluate the technique on weighted graphs due to lack of adequate benchmark instances.

References

[ALMSS92] S. Arora, C. Lund, R. Motwani, M. Sudan, M. Szegedy, "Proof Verification and Hardness of Approximation Problems", In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 14-23, 1992.

[AS92] S. Arora, S. Safra, "Probabilistic Checking of Proofs", In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 2-13, 1992.

[BE85] R. Bar-Yehuda, S. Even, "A Local-Ratio Theorem For Approximating the Weighted Vertex Cover Problem", *Annals of Discrete Mathematics*, Vol 25, Elsevier Science Publishers B.V., pages 27-46, 1985.

[BE76] B. Bollobás, P.Erdös, "Cliques in Random Graphs", *Mathematical Proceedings of the Cambridge Philosophical Society*, Vol 80, pages 419-427, 1976.

[DI93] DIMACS, NSF Center for Discrete Mathematics and Theoretical Computer Science, *Benchmark instances made available by electronic transfer at* `dimacs.rutgers.edu`, Rutgers Univ., Piscataway, NJ, 1993.

[GJ79] M.R. Garey, D.S. Johnson, *Computers and Intractability - A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., 1979.

[NT75] G.L. Nemhauser, L.E. Trotter, Jr., "Vertex Packing: Structural Properties and Algorithms", *Mathematical Programming*, Vol 8, pages 232-248, 1975.

[Rag90] P. Raghavan, "Lecture Notes on Randomized Algorithms", Research Report, IBM RC 15340, 1990.