DIMACS Technical Report 93-69
October 1993

Some Experimental and Theoretical Results on Test Case Generators for the Maximum Clique Problem

by

Laura A. Sanchis[1,2]
Department of Computer Science
Colgate University
Hamilton, NY 13346

Arun Jagota
Department of Mathematical Sciences
Memphis State University
Memphis, TN 38152

[1]Supported in part by DIMACS.
[2]Supported in part by NSF grant CCR-9101974.

DIMACS is a cooperative project of Rutgers University, Princeton University, AT&T Bell Laboratories and Bellcore.

DIMACS is an NSF Science and Technology Center, funded under contract STC–91–19999; and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

We describe and analyze test case generators for the maximum clique problem (or equivalently for the maximum independent set or vertex cover problems). The generators produce graphs with specified number of vertices and edges, and known maximum clique size. The experimental hardness of the test cases is evaluated in relation to several heuristics for the maximum clique problem, based on neural networks, and derived from the work of A. Jagota [9], [11]. We also compare the performance of the different algorithms on the generated graphs.

Our results show that the hardness of the graphs produced by this method depends in a crucial way on the construction parameters; for a given edge density, challenging graphs can only be constructed using this method for a certain range of maximum clique values; the location of this range depends on the expected maximum clique size of random graphs of that density; the size of the range depends on the density of the graph. In addition, one of the algorithms, based on reinforcement learning techniques, has more success than the others at solving the test cases produced by the generators.

We also present some NP-completeness results showing that (in spite of what might be suggested by the results just mentioned) the maximum clique problem remains NP-hard even if the domain is restricted to graphs having a given constant edge density and a constant ratio of the maximum clique size to the number of vertices, for all valid combinations of such values.

1 Introduction

Test cases with known answers can be useful for the testing of approximation or exact algorithms for NP-hard problems. In this paper we describe and analyze test case generators for the maximum clique problem. Because of the equivalence among the maximum clique, independent set, and minimum vertex cover problems, the generators produce test cases with known answers for all three of these problems. The original generating procedure was described in [17] in terms of the vertex cover problem. Some preliminary analysis and modifications based on evaluating the test cases by using some simple approximation algorithms, may be found in [18]. In this report we present further modifications to the generator, and the results of running more sophisticated approximation algorithms for the maximum clique problem on the generated test cases. The approximation algorithms are based on neural network heuristics and were first described in [9] and [11].

The next section describes the approximation algorithms to be tested. We then describe the test case generators and their properties. The following section presents experimental results derived from running the algorithms on test cases produced by the generators. It will be seen that challenging graphs for these algorithms can only be obtained within a certain range of parameter values. The hardness of the graphs depends to a large extent on the edge density and on the relation of the maximum clique size to the expected clique size for random graphs of that density.

In spite of these results, limiting the experimental hardness of the graphs generated by this method to certain parameter values, it is possible to show that hard graphs must exist for all valid constant edge densities with a variety of different maximum clique sizes. This is shown through NP-hardness reductions. Furthermore, we show that the sets of graphs produced by the generators at various edge densities and with various maximum clique sizes, are also NP-hard. Thus although for certain parameter values most of the graphs produced by our implementation of the generators are apparently easily solvable, the set of all graphs with these parameter values which can be produced by the generators must include hard graphs.

2 Approximation algorithms

The approximation algorithms were devised by A. Jagota, and are based on the Hopfield network. They were first described in [9] and [11].

The Hopfield Network [7, 8] is structurally an N-vertex complete weighted graph. $W = (w_{ij})$ is the symmetric zero-diagonal matrix of real-valued weights, w_{ij} is the weight of the edge $\{v_i, v_j\}$. Vertices have binary-valued (in $\{0, 1\}$) or continuous (in $[0, 1]$) labels S_i on them. $I = (I_i)$ is the vector of external biases to vertices. We refer the reader to [6] for further details.

We encode the Maximum Clique problem in a Hopfield Network as follows [9]. Given an N-vertex graph G, construct a Hopfield network instance H with N vertices, one for each vertex v_i in G. Let $w_{ij} = 1$ if vertices v_i, v_j are adjacent in G, $w_{ij} = \rho$ otherwise. We normally choose $\rho = -4N$. Let $I = (w_0)$ where $w_0 = |\rho|/4$.

Earlier encodings of the Maximum Clique problem in Hopfield networks, different than the one above, are in [BGS87, 13, 19]. Encoding Maximum Clique in a Hopfield Network is equivalent to formulating it as an unconstrained quadratic minimization problem.

We now present two algorithms for approximately solving Maximum Clique on instances encoded in Hopfield Networks. (One of them, Discrete Descent, is in fact an algorithm-schema, which is instantiated to give several algorithms.) These algorithms work with any of the above-mentioned encodings though our previous and current experiments employ only our encoding. These algorithms are inherently suited for parallel (SIMD) implementation, and we describe them in a form which emphasizes this. In particular we make heavy use of vector operations.

2.1 Discrete Descent

Data Structures

The vector $\vec{S} \in \{0,1\}^N$ represents the vertex states. The vectors $\vec{n} \in Z^N$ and $\vec{\Delta E} \in Z^N$ represent internal states of vertices. The vector $\vec{V} \in \{-1,0,1\}^N$ represents the directions in which vertex states can switch. The vector $\vec{W_i} = (w_{ji})_i \in Z^N$ is the column vector of vertex v_i's edge-weights. The set U consists of indices of vertices whose states are ONE.

Algorithm Discrete Descent(U).
$n \leftarrow I;\quad S \leftarrow \vec{0}$
for $i \in U$ **do** $S_i \leftarrow 1;\quad n \leftarrow n \,+\, W_i$ **endfor**
loop

For $j = 1, \ldots, N$ $V_j \leftarrow \begin{cases} 1 & n_j > 0 \text{ and } S_j = 0 \\ -1 & n_j < 0 \text{ and } S_j = 1 \\ 0 & \text{otherwise} \end{cases}$

For $j = 1, \ldots, N$ $\Delta E_j \leftarrow -V_j \times n_j$
$k \leftarrow F(\vec{\Delta E})$;
if $\Delta E_k \geq 0$ **then exit** /* no vertex label can switch, fixed point */
if $V_k = 1$ **then** $\{\ S_k \leftarrow 1; n \leftarrow n + W_k\}$
else $\{\ S_k \leftarrow 0; n \leftarrow n - W_k\ \}$

endloop

F is a function that returns the index i of a vertex v_i such that $\Delta E_i < 0$ or the index of any vertex if this condition cannot be satisfied. By such a choice of F, the algorithm minimizes the "energy" function $-(1/2)\vec{S}^T W \vec{S} - \vec{I}^T \vec{S}$ during its operation and hence always terminates [7]. Different instantiations of F lead to different instances of the generic algorithm "Discrete Descent".

It is useful to note that the above algorithm (hence its SD and SSD instantiations presented below) is problem-independent, and is applicable to any problem encoded in the weights W and biases I (in our case, the Maximum Clique problem).

Steepest Descent

Steepest Descent (SD) is a greedy deterministic instance of Discrete Descent in which F returns the index of the vertex whose ΔE_i is minimum. Ties are broken lexicographically.

Let SD(U_0) denote starting with the initial state U_0 (i.e., U_0 represents an initial assignment of labels to the vertices) and terminating at a local minimum of the earlier-mentioned energy function (in our case, a maximal clique). SD(U_0), on our Maximum Clique encoding, emulates the following greedy clique-finding algorithm. First, vertices from U_0 are deleted one by one until the state, call it U_1, is a clique; then vertices in $V \backslash U_1$ are added one by one to U_1 until the state, call it U_2, is a maximal clique. Consider a point in the deletion phase when the state is S. If S is not a clique, the vertex deleted from S in the next step has least degree in $G[S]$, the subgraph induced by S.

Stochastic Steep Descent

Stochastic Steep Descent (SSD) is a randomized instance of Discrete Descent in which F returns the index based on the following probability distribution on ΔE:

$$P[i \text{ is returned}] \quad := \quad \begin{cases} 0 & \text{if } \Delta E_i \geq 0 \\ \frac{\Delta E_i}{\sum_{j:\Delta E_j < 0} \Delta E_j} & \text{otherwise} \end{cases} \tag{1}$$

Let SSD(U_0, M, k) denote up to M runs of SSD, each run starting from the same initial state U_0 (i.e., U_0 represents an initial assignment of labels to the vertices) and terminating at a state representing a clique. If a clique of size k is found in any run, then the algorithm is terminated after that run and the clique is output. If not, then the largest clique found in the M runs is output. As an example, SSD($\emptyset, 1, \infty$), on our Maximum Clique encoding, emulates in parallel form the following randomized clique-finding algorithm. The initial state is the empty set. It is extended to a maximal clique by adding randomly selected vertices one by one while keeping every intermediate state a clique.

We also examine the performance of a variant of SD which we call GSD (greedy steepest descent) which behaves exactly like SD with the following greedy modification when vertices are being added to the clique: if W is the set of eligible vertices to be added, then the vertex in W with highest degree in the subgraph induced by W, is chosen. The algorithm GSD(U, M) consists of running GSD(U) M times, randomly permuting the graph before each repetition, and returning the maximum value found during the M runs. (By a random permutation of the graph we mean a random relabelling of the vertices.)

Stochastic Steep Descent with Reinforcement Learning

Stochastic Steep Descent with Reinforcement Learning (SSD$_{\mathrm{RL}}$) is a new variant of SSD formulated by A. Jagota that incorporates the idea of reinforcement learning—an important learning paradigm in both biological and artificial neural networks—into a Hopfield network. It is designed for optimization purposes. Reinforcement learning is useful in learning difficult sequential tasks, for example to balance a pole, in which usually the only feedback from the environment comes in the form of a reinforcement (or penalty) at the end of one trial on the task (e.g. when the pole drops). One of the first influential papers on this topic is [3].

Our algorithm is as follows. Let $\mathrm{SSD}_{\mathrm{RL}}(U_0, M, k)$ denote up to M runs of $\mathrm{SSD}_{\mathrm{RL}}$, each run starting from the same initial state U_0 and terminating at a state representing a clique. If a clique of size k is found in any run, then the algorithm is terminated after that run and the clique is output. If not, then the largest clique found in the M runs is output. $\mathrm{SSD}_{\mathrm{RL}}$ differs from SSD in that after every run, the external vertex biases I_i are adjusted based on the quality of the clique found in that run. Before the first run, $I := \vec{1}, \alpha_1 = \alpha_2 := 0.5$ where I is the external biases vector, and α_1, α_2 are real-valued scalars. Let $k^{(i)}$ denote the size of the largest clique found after i runs of $\mathrm{SSD}_{\mathrm{RL}}(U_0)$. Let C be the clique found in the $(i+1)^{th}$ run.

$$\begin{aligned}\text{IF } |C| > k^{(i)} \text{ THEN } & I := \vec{\min}(3N, I + \lfloor \alpha_1^{(i)}(|C| - k^{(i)})\vec{S} \rfloor) \\ \text{ELSE } & I := \vec{\max}(1, -1 + I + \lfloor \alpha_2^{(i)}(|C| - k^{(i)})\vec{S} \rfloor)\end{aligned}$$

$$\alpha_1^{(i+1)} := \alpha_1^{(i)} + 20.0/N, \quad \alpha_2^{(i+1)} := 0.95 \times \alpha_2^{(i)}$$

We write this algorithm using vector operations to emphasize its suitability for parallel SIMD implementation. Note that $\vec{\min}$, $\vec{\max}$, and $\lfloor \ldots \rfloor$ are all vector functions, generalized from their scalar versions in the usual way. Consider $|C| > k^{(i)}$. The statement after the THEN part is executed, which increases I_i only for those components i whose $S_i = 1$, that is whose vertices are in the clique C. Thus the vertices in C are rewarded for belonging to a clique larger than one previously found. Similarly, if $|C| \le k^{(i)}$, the ELSE part penalizes the vertices in C. Note that I_i are kept in the set $\{1, 2, \ldots, 3N\}$, which, when $\rho < -4N$, ensures that only cliques are stable states.

Note that SSD_{RL}, as described above, is a Maximum Clique algorithm. However, it is straight-forward to make it problem-independent by using energy values instead of the clique sizes.

The following computer experiment illustrates how the manner in which SSD_{RL} uses information from previous runs to bias its search space helps it find better cliques in certain graphs. A 200-vertex graph called san200-0.9-2 was chosen for this study; its maximum clique size was known to be 60. $\mathrm{SSD}(\emptyset,)$ found a clique of size 42 in 1000 runs. As shown in Table 1, $\mathrm{SSD}_{RL}(\emptyset,)$ found a maximum clique in 160 runs. Of special interest is the behavior. There was some progress early (runs 1 to 23) and no progress for a long while (runs 24 to 140). Then a larger clique was found in run 141 following which much larger cliques were found rapidly in the next few runs. This shows how biasing the search into the neighborhood of a recently found improved clique helps.

3 Generators

The generators produce graphs with a specified number of vertices and edges, and a given minimum vertex cover size. The complement of each generated graph thus has a known

Table 1: The behavior of $\mathrm{SSD}_{RL}(\emptyset,)$ on the graph san200-0.9-2

i	1	2	14	23	141	142	148	152	155	156	161
$\|C\|$	36	37	38	39	40	41	46	48	55	59	60

maximum clique size. The main idea behind the construction procedure is as follows. To produce a graph with n vertices, m edges, and minimum vertex cover size c, first partition the n vertices into $k = n - c$ cliques; choose all but one of the vertices in each clique to be in the cover; finally add additional edges, such that each added edge is adjacent to at least one vertex in the chosen cover. Taking the complement of this graph, we obtain a graph with maximum clique size equal to k.

In [18] it was reported that if the vertex partition into the k cliques, and the additional edges, are chosen at random, then the resulting graphs tend to be easily solvable by even quite simple greedy algorithms. However, if the additional edges are chosen in such a way that the average degrees of cover and noncover vertices are made equal (or as close as possible), then the resulting graphs appear to be experimentally harder. Choosing the k clique sizes as close to each other as possible also appears to produce harder graphs (relative to these greedy algorithms). The generators to be discussed here incorporate both of these variants. As explained in [18], equalizing the degrees of cover and noncover vertices serves to foil greedy approximation algorithms which select vertices with high degrees to be in the cover. Partitioning the vertices into k cliques of approximately equal sizes allows greater choice in choosing the additional edges, and also makes it easier to adjust the average degrees as described above. See [18] for more details about this construction method.

It is also shown in [18] that the set of graphs produced by these generators is hard with respect to the minimum vertex cover problem, in the sense that no polynomial-time approximation algorithm can find the optimal answer for all graphs in the set, unless P=NP. A similar statement must then apply with regard to the maximum independent set problem, and for the maximum clique problem in relation to the complemented graphs.

Preliminary experimentation showed that the algorithms described in the previous section were more successful than the simpler algorithms considered in [18], at solving the test cases. Some further modifications were made to the generators in an effort to produce experimentally harder graphs. One approach involved building extra independent sets into the constructed graph, or equivalently extra cliques into the complemented graph. We suspected that the presence of relatively large cliques in the graph would tend to misdirect the stochastic algorithms towards finding these (smaller) cliques instead of the maximum clique. A few variants of this idea were considered.

One simple technique for accomplishing this is as follows. After the k cliques have been formed, and before the additional edges are added, randomly choose sets of vertices composed of at most one vertex from each clique. When the additional edges are added, do not add any edge connecting two vertices in the same set. Thus a number of additional independent sets are explicitly constructed into the graph; equivalently, additional cliques are built into the complemented graph. Graphs incorporating this modification seem in fact to be more challenging for some of the algorithms. However, the number of built-in independent sets and/or their sizes have to be relatively large in order to achieve this effect. The relatively large independent sets apparently also make the graphs easier for exact enumerative algorithms such as the one in [4]. The reason for this may be that once a relatively large clique is found by the enumerative algorithm, many paths can be pruned from its search tree, since it is clear that no larger clique can be found within those paths.

Another approach was suggested by looking at the reduction from 3SAT to the minimum vertex cover problem which is found in [5]. In this reduction, a graph is constructed which is composed of cliques of sizes 2 and 3, with additional edges connecting vertices in the different cliques. In fact each additional edge joins a vertex from one of the 2 cliques to a vertex in one of the 3 cliques. This construction suggested the following modification to our procedure: after the vertices have been partitioned into the k cliques, divide these cliques into two groups; when adding the additional edges, do not add any edges joining two vertices belonging to cliques in the same group. Of course this limits the number of additional edges which may be added, and thus may result in an overconstrained and hence easier graph. We therefore adjust the process as follows.

After the k cliques are formed, partition the cliques into two classes A and B. Choose a parameter r (to be discussed later) and let A_r be the vertex set consisting of all but r of the cover vertices in each clique in A. Define B_r similarly. When the additional edges joining vertices in different cliques are chosen, do not add any edge joining two vertices in A_r or two vertices in B_r. Also do not add any edges joining a noncover vertex in one of the A cliques to a vertex in A_r, or a noncover vertex in one of the B cliques to a vertex in B_r.

Note that this construction again has the effect of building extra cliques into the complemented graph. However, these cliques are not randomly distributed as in the first approach, but rather are concentrated on the vertex sets A_r and B_r. Apparently this construction causes the enumerative algorithm to do a lot of searching within these vertex sets, resulting in a greater running time for the algorithm.

The number and sizes of the built-in cliques depend on the relative sizes of the classes A and B, and on the choice of the parameter r. Through experimentation we determined that choosing the class sizes to be roughly equal to each other appears to produce the hardest graphs. The best choice of r depends on the parameters of the graph, specifically on the edge density and on the sizes of the k cliques used in the construction. A small value for r results in more cliques being built into the (complemented) graph, but also constrains the structure of the graph: too much constraint results in less choice in the construction and in "easier" graphs. We have chosen different values of r experimentally for each graph parameter set; these are included in the data. In general, lower values of r can be used, and thus more cliques incorporated, for the denser (complemented) graphs. Also the value of r can be larger for graphs with smaller maximum clique values, since in this case the size of each of the k cliques is larger.

We denote the original generator from [18] as SimParD. The generator just described will be denoted as SimParD(r). Note that if $r_1 < r_2$ then the set of graphs generated by SimParD(r_1) for given parameter values is a subset of the set generated by SimParD(r_2), and both are subsets of the set generated by SimParD. It is shown in a later section that the set of graphs produced by the SimParD(0) generator is also hard with respect to the maximum clique problem (or equivalently the minimum vertex cover problem).

4 Experimental results

In this section we present the results of running the algorithms described in a previous section on graphs produced by the generators. We first concentrate on the original SSD and GSD algorithms, and then present results for the SSD algorithm with reinforcement learning.

Our experiments were run on a NeXT 68040 machine. Part of our objective was to evaluate the hardness of the instances produced, and in particular to determine whether these algorithms can find the optimal answer for different types of generated graphs, and if so how many iterations are necessary to do so.

4.1 Experiments on the SSD and GSD algorithms

Tables 2 and 3 show the results of running the GSD and SSD algorithms on 200 vertex graphs constructed by SimParD and SimParD(r), respectively. Each line in the tables represents an average taken over 20 different graphs of the type described: N is the number of vertices, p is the edge density (of the complemented graph), Opt is the value of the max clique (values marked with a * are those which random graphs of the given density are expected to have). For each algorithm we give both the average value obtained over all iterations, and the Max value obtained. The algorithms were run until either 1000 iterations were performed or until the value of the max clique was found. Ignore for the moment the last column. The next to the last column gives the average over the 20 graphs of the largest clique size found by any of the algorithms for each graph.

For a given vertex count and edge density, we find that challenging graphs can only be obtained, using these generators, for a particular range of max clique values. These ranges get smaller and move downwards as the density of the graph decreases. They comprise graphs whose maximum clique is equal to or larger than the expected sized of the maximum clique for random graphs of that density. The tables include data for graphs within and around these ranges. As mentioned previously, we found that the GSD and SSD algorithms, when iterated many times, perform much better on the generated graphs than the simple greedy strategies investigated in [18]. Also it can be seen that the graphs in Table 3 are in general somewhat more challenging for the algorithms than those in Table 2. Looking more closely at the data it appears that the SSD(V) and GSD(V) algorithms, particularly the latter, have dropped their performance the most in going from Table 2 to Table 3. Note particularly that the GSD(V) algorithm appears to perform well for several of the graph types in Table 2, but for hardly any in Table 3. Also in Table 3, the SSD algorithms seem to perform better than the GSD algorithms on the lower density graphs, while on the higher density graphs SSD($\emptyset$) and GSD($\emptyset$) have better performance than the SSD(V) and GSD(V) algorithms. It can also be seen that the SSD algorithms have more variance between the average and max

Table 2: Experiments on 200 vertex graphs generated by SimParD

N	p	Opt	SSD($\emptyset$, 1000)		SSD(V, 1000)		GSD($\emptyset$, 1000)		GSD(V, 1000)		max	SSD($\emptyset$)
			Max	Avg	Max	Avg	Max	Avg	Max	Avg		20000
200	0.5	11*	9.90	7.13	9.90	7.11	9.15	8.84	10.10	8.41	10.35	10.80
200	0.5	20	19.30	7.52	19.30	7.54	10.50	10.12	17.35	9.75	20	20
200	0.5	30	30	8.56	30	9.20	12.50	11.98	28.15	20.49	30	-
200	0.7	18*	15.20	11.72	15.40	11.85	15.30	14.59	16.40	13.79	16.40	16.40
200	0.7	30	24.85	12.34	23.30	12.40	17.30	16.39	26.05	17.42	29.40	30
200	0.7	45	45	13.35	45	14.38	22.85	20.69	42.90	23.83	45	-
200	0.9	25	25	23.98	25	24.13	25	25	25	24.92	25	-
200	0.9	35	32.20	27.13	32.85	27.72	34.75	32.92	34.60	31.49	34.85	33.25
200	0.9	44*	34.45	28.27	35.25	28.92	37.90	35.28	37.80	33.39	38.40	36.70
200	0.9	60	47.90	29.47	45.30	30.27	49.40	41.84	51.25	39.58	56.25	55.10
200	0.9	70	58.45	30.56	61.60	31.35	52.80	46.23	62.80	54.94	69.75	69.15
200	0.9	80	75.40	32.78	73.95	33.46	59.80	52.10	73.05	63.29	79.75	80

values found during the iterations. This is to be expected since in the SSD algorithm the choice of vertex to add or delete from the clique at each step is less restricted than in GSD, so a wider variety of cliques can be obtained. This suggests that using additional iterations may be more profitable with these algorithms than with the GSD algorithms. The fact that the overall max average is sometimes higher than the max obtained by any one algorithm seems to imply that in some way the algorithms may complement each other, one doing better when another does worse, or viceversa, depending on the graph.

Table 4 shows the results of experiments on 400 vertex graphs generated by SimParD(r). Here each line represents the average taken over only 10 graphs with the specified parameter values. Because of the slowness of the algorithms SSD(V), GSD($\emptyset$), and GSD(V), these algorithms were again run for only 1000 iterations. (Timing results are discussed below.) The faster SSD($\emptyset$) algorithm, however, was run for 2000 iterations. From this data it can be seen that the algorithms have more difficulty with these larger graphs, especially at the higher densities; this applies even to the 2000 iteration runs of SSD($\emptyset$). We suspect that the number of iterations must be increased much more than linearly, in order to achieve the same type of performance that was observed with the 200 vertex graphs, when the number of vertices is increased. We observe again the same pattern in the relative performances of the algorithms, as that described for Table 3.

Table 5 presents timing results from running each algorithm for 100 iterations on several graphs having 200 and 400 vertices and different edge densities and max cliques. As can be seen, the SSD($\emptyset$) algorithm is by far the most efficient. Its runtime does vary considerably, however, depending on the type of graph, increasing with both edge density and value of the max clique. The other algorithms are much slower, and their runtimes vary little with respect to edge density and max clique. Notice the quadratic behavior evidenced by a quadrupling of the time when the number of vertices is doubled, for each of these algorithms. The SSD($\emptyset$)

Table 3: Experiments on 200 vertex graphs generated by SimParD(r)

N	p	Opt	r	SSD($\emptyset$, 1000)		SSD(V, 1000)		GSD($\emptyset$, 1000)		GSD(V, 1000)		max	SSD($\emptyset$)
				Max	Avg	Max	Avg	Max	Avg	Max	Avg		20000
200	0.5	11*	7	8.90	6.11	8.75	6.18	8.40	7.93	7.75	6.05	9.15	10.40
200	0.5	20	4	19	9.64	17.15	9.79	11	10.62	10.70	10	19.55	20
200	0.5	30	2	30	14.18	30	14.60	15	15	15	15	30	-
200	0.7	18*	2	14.20	10.49	14.45	10.48	14.40	13.41	12.15	9.36	14.90	15.65
200	0.7	30	1	23.75	14.67	22.20	14.88	17.25	16.32	15.95	15	24.20	30
200	0.7	45	0	45	22.43	44.55	21.71	30.85	29.30	22	22	45	-
200	0.9	25	0	25	23.77	25	24.15	25	25	25	25	25	-
200	0.9	35	0	31.45	26.19	32.30	26.84	34.65	32.55	33.05	26.95	34.65	32.80
200	0.9	44*	0	33.15	27.40	34.15	27.68	37.60	34.20	28.60	23.72	37.60	36.10
200	0.9	60	0	45.90	30.67	42.60	31.52	53.75	47.80	32.85	30.26	54.05	54.20
200	0.9	70	0	62.35	33.22	58.35	34	60.40	53.29	37.15	35.09	66.00	69.65
200	0.9	80	0	78.10	38.24	67.75	40.68	79.70	78.96	41.20	40.02	80	80

algorithm's growth with respect to vertex count is definitely slower than quadratic but larger than linear.

Because of the very efficient performance of SSD($\emptyset$,), and its tendency to suddenly come up with a large clique after many iterations yielding small clique sizes, it seemed to be a good candidate for investigating the benefits of allowing many more iterations of the algorithm. The last column in Tables 2, 3, and 4 give the average maximum clique size found after 20000 iterations of SSD($\emptyset$). It can be seen that a definite improvement in the clique size is attained, and in a few cases the maximum clique size is found where fewer iterations had been insufficient to do so. Table 6 provides additional data for some selected parameters, using 40000 and 80000 iterations. As can be seen a doubling of the number of iterations appears to produce only a constant increase in the max clique size returned. Thus for the dense 400 vertex graphs where the max clique returned after 2000 iterations is quite far from optimum, it would appear that only an extremely large number of iterations could hope to achieve the optimum value.

For purposes of comparison Table 7 gives the results of running all four algorithms on random graphs. It can be seen that the algorithms perform better on the random graphs than on constructed graphs having the same density and maximum clique size. Timings on random graphs showed very similar results for the last three algorithms; SSD($\emptyset$) was somewhat slower on random graphs than on the constructed graphs, although still much faster than the other algorithms.

4.2 Experiments with the reinforcement learning algorithm

The reinforcement learning algorithm, SSD_{RL}, was run on the same types of generated graphs. It performed very well, and greatly reduced the range of maximum clique sizes for

Table 4: Experiments on 400 vertex graphs generated by SimParD(r)

N	p	Opt	r	SSD($\emptyset$, 2000)		SSD(V, 1000)		GSD($\emptyset$, 1000)		GSD(V, 1000)		max	SSD($\emptyset$)
				Max	Avg	Max	Avg	Max	Avg	Max	Avg		20000
400	0.5	13*	4	8.90	6.72	8.60	6.94	9.10	8.33	7.50	7	9.10	10.20
400	0.5	25	4	18.90	12.28	16.60	12.65	13.90	13.39	13	13	19.40	24.50
400	0.5	40	2	40	19.40	38.40	19.78	20	20	20	20	40	-
400	0.7	22*	3	15.90	12.24	15.70	12.04	16.20	15.15	13.20	11.12	16.40	16.50
400	0.7	40	1	24.20	19.59	23	19.83	22.40	21.35	20	20	24.50	33.50
400	0.7	60	1	52.20	27.46	45.80	29.15	32.50	31.11	30	30	53.60	60
400	0.7	80	1	80	34.25	80	36.75	40.80	40.14	40	40	80	-
400	0.9	35	4	35	30.87	35	31.34	35	35	35	34.11	35	35
400	0.9	55*	0	39.70	33.47	40.10	33.71	45.80	41.74	33.40	28.56	45.80	40.80
400	0.9	75	0	44.80	38.16	45.10	38.74	54.80	48.22	39.50	37.14	54.90	47.90
400	0.9	100	0	62.50	44.41	66.30	46.04	75.80	64.93	51.10	50.01	81.20	80.90
400	0.9	120	0	85.40	50.33	74.40	58.43	103	96.55	60.50	60	106.80	108.50

which challenging graphs (for this algorithm) can be found. Table 8 presents results for 400 vertex graphs. Each line represents data computed for 10 graphs of the type described, although the graphs used for the 20000 iterations, when they were performed, were actually a different set of 10 graphs.

Both for the algorithms presented in the previous sections, and for the algorithm in this section, giving the average maximum value found by the algorithm on 10 different graphs can be misleading. This is because the algorithm tends to either provide a relatively low suboptimal clique value, or to find the optimum. This is especially true for the data presented in this section. We therefore give, for each set of graphs and number of iterations, the average of the max values found for all graphs in the set (Max (all)), the number of graphs for which the optimum was found (Opt Found), and the average of the max values for all graphs for which the optimum was not reached (Max (nonOpt)).

Comparisons with the data in Table 4 show that, as mentioned above, the range of max clique values where challenging graphs can be found for the algorithm has been reduced for all three densities considered.

We also performed experiments on larger graphs with 800 vertices. The results are presented in Table 9. Here we see the range of challenging graphs increasing again. For example, notice that for density 0.9, generated graphs with 800 vertices and maximum clique size equal to 140, are challenging for the algorithm; 140 is more than twice the expected clique size for graphs of this density (66). However, graphs of the same density with 400 vertices and maximum clique size 85 already succumb to 10000 iterations, whereas 85 is less than twice the expected max clique size for this type of graph (55).

Table 10 contains running times for the algorithm on a single graph of each type specified, using 1000 iterations. As can be seen, this algorithm is slower than SSD($\emptyset$), but in most cases faster than the other three algorithms considered in the previous subsection. However, for

Table 5: Running times in CPU seconds, 100 iterations of each algorithm

N	p	Opt	r	SSD($\emptyset$)	SSD(V)	GSD($\emptyset$)	GSD(V)
200	0.5	11	7	2.0	67.5	76.2	86.5
200	0.5	20	4	3.0	68.3	76.4	86.1
200	0.5	30	2	4.2	68.8	76.5	85.6
200	0.7	18	2	3.2	66.4	73.7	86.4
200	0.7	30	1	4.3	66.8	73.9	85.7
200	0.7	45	0	6.1	66.7	74.1	85.3
200	0.7	60	0	8.3	67.9	74.2	84.2
200	0.9	25	0	6.7	63.7	71.6	85.2
200	0.9	35	0	7.4	64.0	71.8	86.0
200	0.9	44	0	8.2	63.6	71.8	85.8
200	0.9	60	0	8.6	64.2	72.1	84.7
200	0.9	70	0	9.2	64.9	72.4	84.2
200	0.9	80	0	10.6	63.1	73.7	83.7
400	0.5	13	4	4.6	270.3	305.2	347.0
400	0.5	25	4	7.7	271.4	305.6	346.0
400	0.5	40	2	11.7	273.5	306.3	344.4
400	0.7	22	3	7.5	266.7	295.2	346.8
400	0.7	40	1	11.5	267.8	295.6	345.0
400	0.7	60	1	15.6	270.3	296.4	343.3
400	0.7	80	1	19.4	275.3	296.9	340.6
400	0.9	35	4	17.4	260.2	286.0	343.8
400	0.9	55	0	18.8	261.3	286.2	345.1
400	0.9	75	0	21.5	261.2	286.8	343.0
400	0.9	100	0	24.7	263.4	286.1	340.3
400	0.9	120	0	27.9	260.2	290.5	338.6

Table 6: Results of 20000, 40000, and 80000 iterations of SSD($\emptyset$) on graphs constructed by SimParD(r)

N	p	Opt	20000	40000	80000
200	0.9	60	54.20	57.70	58.55
400	0.9	75	47.90	49.50	53.70
400	0.7	40	33.50	35.70	37.80

Table 7: Experiments on random graphs - 1000 iterations of SSD($\emptyset$) on 200 vertex graphs, 2000 iterations on 400 vertex graphs

N	p	Opt (Exp)	SSD($\emptyset$,) Max	SSD($\emptyset$,) Avg	SSD(V, 1000) Max	SSD(V, 1000) Avg	GSD($\emptyset$, 1000) Max	GSD($\emptyset$, 1000) Avg	GSD(V, 1000) Max	GSD(V, 1000) Avg	max	SSD($\emptyset$) 20000
200	0.5	11	10.35	7.36	10.25	7.45	9.90	9.50	10.55	8.84	10.85	10.75
200	0.7	18	16.20	12.25	16.30	12.47	16.80	15.94	17.60	14.90	17.65	17.15
200	0.9	44	36.10	29.50	37.00	30.49	39.80	37.56	39.95	35.80	40.10	37.75
400	0.5	13	11.60	8.39	11.30	8.45	11.60	11.03	12.20	9.89	12.40	12.00
400	0.7	22	18.20	14.15	18.00	14.30	19.40	18.55	19.80	16.66	19.90	19.00
400	0.9	55	43.40	36.02	43.50	36.84	50.00	46.72	48.90	43.52	50.10	44.10

Table 8: Results of running SSD_{RL} on 400 vertex graphs generated by SimParD(r)

N	p	Opt	r	2000 iterations			10000 iterations			20000 iterations		
				Max (all)	Opt Found	Max (nonOpt)	Max (all)	Opt Found	Max (nonOpt)	Max (all)	Opt Found	Max (nonOpt)
400	0.5	13*	9	9.30	0	9.30	10.40	2	9.75	11.10	4	9.83
400	0.5	19	6	16.10	6	11.75	17.60	8	12.00	18.60	9	15.00
400	0.5	25	4	25	10	-	25	10	-			
400	0.7	19	6	16.30	0	16.30	16.50	0	16.50			
400	0.7	22*	5	16.00	0	16.00	17.10	0	17.10	17.50	0	17.50
400	0.7	31	3	20.40	1	19.22	26.20	6	19.00	27.60	7	19.67
400	0.7	37	2	32.60	7	22.33	37	10	-			
400	0.9	37	2	36.30	3	36.00	36.80	8	36.00			
400	0.9	49	1	39.90	0	39.90	41.40	0	41.40			
400	0.9	55*	1	40.80	0	40.80	42.00	0	42.00	43.80	0	43.80
400	0.9	75	0	56.60	4	44.33	63.30	6	45.75	66.40	7	46.33
400	0.9	85	0	70.80	6	49.5	85	10	-			

Table 9: Results of running SSD_{RL} on 800 vertex graphs generated by SimParD(r)

N	p	Opt	r	2000 iterations			10000 iterations			20000 iterations		
				Max (all)	Opt Found	Max (nonOpt)	Max (all)	Opt Found	Max nonOpt	Max (all)	Opt Found	Max (nonOpt)
800	0.5	15*	17	10.10	0	10.10	10.90	1	10.44			
800	0.5	30	8	20.70	3	16.71	26.10	7	17.00			
800	0.7	25*	9	18.10	0	18.10	18.80	0	18.80			
800	0.7	50	4	27.70	0	27.70	30.10	1	27.89			
800	0.7	75	2	54.40	4	40.67	68.20	8	41.00			
800	0.9	66*	3	47.80	0	47.80	48.30	0	48.30			
800	0.9	80	2	50.20	0	50.20	50.80	0	50.80			
800	0.9	100	1	57.20	0	57.20	58.00	0	58.00	58.40	0	58.40
800	0.9	120	1	65.60	0	65.60	76.90	2	66.13	77.20	2	66.50
800	0.9	140	0	81.50	1	75.00	94.70	3	75.29	101.30	4	75.50
800	0.9	160	0	85.10	0	85.10	160	10	-			

the highest density, the runtime of SSD_{RL} is quite close to that of the three slower algorithms from the previous section.

Table 11 contains the results of running SSD_{RL} on random graphs with 400 vertices. Each line lists the average max value found using 2000, 10000, and 20000 iterations on 10 different graphs with the specified parameter values. (Again, a different set of graphs was used for the 20000 iterations). It can be seen that the algorithm does better on the random graphs than on generated graphs with the same parameter values.

4.3 Remarks

As the results of this section show, constructed graphs may present more difficulties for certain approximation algorithms than for others. There may in fact exist other types of algorithms which solve the instances not solved by the algorithms considered here. In particular, preliminary experiments indicate that the algorithm provided as a benchmark program for the Second Dimacs Challenge (1993), is able to solve most of the instances constructed by the generator within 1000 iterations. This algorithm is based on the one for finding large independent sets used in the graph coloring algorithm XRLF described in Johnson, Aragon, McGeoch, and Schevon [12], originally suggested by Matula and Johri. It should be noted, however, that this algorithm is considerably slower than $\mathrm{SSD}(\emptyset)$ and SSD_{RL}. Further investigation may involve modifying the generating procedures to attempt to produce harder graphs for these and other algorithms.

Table 10: Running times in CPU seconds for SSD_{RL} - 1000 iterations

N	p	Opt	r	Time (secs.)
400	0.5	13*	9	64.3
400	0.5	19	6	84.2
400	0.5	25	4	107.6
400	0.7	19	6	102.6
400	0.7	22*	5	108.3
400	0.7	31	3	133.4
400	0.7	37	2	151.7
400	0.9	37	2	249.2
400	0.9	49	1	263.4
400	0.9	55*	1	272.8
400	0.9	75	0	307.3
400	0.9	85	0	319.8
800	0.5	15*	17	156.0
800	0.5	30	8	262.4
800	0.7	25*	9	256.2
800	0.7	50	4	417.3
800	0.7	75	2	738.5
800	0.9	66*	3	670.9
800	0.9	80	2	717.7
800	0.9	100	1	802.0
800	0.9	120	1	887.3
800	0.9	140	0	1001.0

Table 11: Results of running SSD_{RL} on random graphs

N	p	Exp.	iterations		
		Opt	2000	10000	20000
400	0.5	13	11.30	11.80	12.00
400	0.7	22	18.60	19.20	19.50
400	0.9	55	43.10	45.80	47.60

5 NP-Hardness Results

The generators examined in the previous sections are apparently not able to produce experimentally hard graphs outside of a certain range of maximum clique values; the location and size of this range are dependent on the edge density of the graph. In particular, graphs constructed to have very large maximum clique values appear to be always easily solvable by simple greedy algorithms. Also at lower edge densities experimentally hard graphs are difficult to obtain.

This leads one to consider the question of whether hard graphs for this problem are restricted to certain parameter values, or whether other types of generators could produce experimentally hard graphs at a variety of parameter settings. The results in this section show that there do in fact exist hard graphs with all valid combinations of constant edge density and ratio of the max clique value to the number of vertices in the graph.

Moreover, it can be shown that the set of all graphs which can be produced by the generator SimParD or SimParD(0), for a fixed edge density and fixed ratio of max clique value to the number of vertices, is also hard with respect to the maximum clique problem. This holds for a variety of edge densities and max clique values, in spite of the fact that for many of these parameter combinations, most graphs produced by our implementation of the generator appear to be quite easy.

The following definition is from [16].

Definition 5.1 Let Π be an NP-hard optimization problem. A set B of instances of Π is *hard* with respect to Π if no polynomial-time approximation algorithm for Π can give the optimal answer for all instances in B, unless P=NP.

In the following we state all results in terms of the maximum independent set problem, as this makes the proofs easier to present. It is clear that a set of graphs is hard with respect to the maximum independent set problem if and only if the set of complemented graphs is hard with respect to the maximum clique problem.

First of all we must determine for what values of n, d, p there exist graphs with n vertices, edge density d, and maximum independent set size equal to pn. Let $ME(n)$ denote the maximum number of edges a graph with n vertices can have, namely $n(n-1)/2$. Also let $IR(G)$ denote the ratio of the size of the maximum independent set to the number of vertices, for the graph G.

Given a real number d, a graph cannot have edge density exactly equal to d unless $dME(n)$ is an integer. We therefore have the following definition.

Definition 5.2 Let d be a real number. We say a graph with n vertices has near-edge-density equal to d if it has $\lfloor dME(n) \rfloor$ edges.

Definition 5.3 Let n, k be positive integers, $1 \leq k \leq n$. Let q, r be the quotient and remainder, respectively, when n is divided by k. So $n = kq + r$, $0 \leq r < k$. Define

$$\kappa(n, k) = r(q+1)q/2 + (k-r)q(q-1)/2$$

Theorem 5.1 *Let p be a rational number, $0 < p < 1$, and d a real number such that $0 < d \leq 1 - p^2$. Then for sufficiently large n, if pn is an integer, there exists a graph with n vertices, maximum independent set of size pn, and near-edge-density equal to d.*

Proof: Let n be a positive integer such that pn is an integer. If G is a graph with n vertices and maximum independent set of size pn, then G can have any number of edges between the values $Min(n,p)$ and $Max(n,p)$, where $Min(n,p) = \kappa(n,pn)$ and $Max(n,p) = (n - pn)(n + pn - 1)/2$ (see [17]). We have

$$Min(n,p) \leq pn(n/pn + 1)(n/pn)/2 = (p+1)n/(2p)$$

So $Min(n,p)/ME(n) \to 0$ as $n \to \infty$. Also

$$Max(n,p)/ME(n) = (n - pn)(n + pn - 1)/(n^2 - n)$$

$$= \frac{(1-p^2)n^2 - (1-p)n}{n^2 - n} > 1 - p^2$$

Hence it is possible to choose n large enough so that

$$Min(n,p) \leq \lfloor dME(n) \rfloor < Max(n,p)$$

and the result follows. □

Corollary 5.2 *Let p be a rational number, $0 < p < 1$, and d a real number such that $0 < d \leq 1 - p^2$. Then for sufficiently large n, if pn is an integer, there exists a connected graph with n vertices, independent set of size pn, and near-edge-density equal to d.*

Proof: If G is connected, then $Min(n,p) = \kappa(n,pn) + pn - 1$ and $Max(n,p)$ stays the same. So we still have $Min(n,p)/ME(n) \to 0$ as $n \to \infty$ and the rest of the proof is unchanged. □

Corollary 5.3 *Let p be a rational number, $0 < p < 1$, and $\epsilon > 0$. Then for sufficiently large n, if pn is an integer, and G is a graph with n vertices and maximum independent set of size pn, then the edge density of G is at most $1 - p^2 + \epsilon$.*

Proof: From the above proof it can be seen that $Max(n,p)/ME(n) \to 1 - p^2$ as $n \to \infty$. □

Theorem 5.4 *Let p be a rational number, $0 < p < 1$, and d a real number such that $0 < d < 1 - p^2$. Then the set of all graphs having n vertices, maximum independent set of size pn, and near-edge-density equal to d, is hard with respect to the maximum independent set problem.*

Proof: We will use a reduction from the 3SAT problem (CNF satisfiability with length three clauses) to the maximum independent set problem. This reduction is modelled on the reduction from 3SAT to the minimum vertex cover problem which is found in [5].

The construction uses an integer parameter K where $K \geq 4$ and $K \geq 1/p$. The value of K is dependent only on the value of p and its choice will be considered later. Let $p = p_1/p_2$ where p_1, p_2 are integers which are relatively prime. Given a 3SAT formula F with n variables and m clauses the reduction produces a graph G_F as follows. First the formula is changed so that the number of variables equals the number of clauses, and such that this number is a multiple of $p_2 - p_1$. This can be done by adding additional variables and/or repeating clauses. Let N be the number of clauses (or variables) in the modified formula. Let $u_1, ..., u_N$ denote the variables and $c_1, ..., c_N$ the clauses in the formula.

For each variable u_i in the formula, form 2 cliques A_i and B_i of size K. In the A_i clique, label two of the vertices u_i and $\overline{u_i}$. In the B_i clique, label one of the vertices x_i. Add edges joining each vertex in B_i to each of the vertices in A_i except for u_i and $\overline{u_i}$.

For each clause c_j form two cliques C_j and D_j of size K. Label three of the vertices in C_j r_j, s_j, and t_j. Label one of the vertices in the D_j clique y_j. Add edges joining each vertex in D_j to each of the vertices in C_j except for r_j, s_j, and t_j. In addition if the first literal in c_j is u_i ($\overline{u_i}$), then add an edge connecting r_j to vertex u_i in A_i (or vertex $\overline{u_i}$). Proceed similarly for the second literal and s_j, and the third literal and t_j.

Let $R = 4N(Kp-1)/(1-p)$. It can be checked that because N is a multiple of $p_2 - p_1$, R must be an integer. Add R additional isolated vertices to the graph. Denote these vertices as $v_1, ..., v_R$. Call the graph constructed so far L_F. Note that the original formula is satisfiable if and only if L_F has a maximum independent set of size $4N+R$. The proof of this fact is similar to that given in the reduction from 3SAT to the minimum vertex cover problem given in [5]. In fact, it can be seen that if the formula is satisfiable, then L_F has a maximum independent set which includes the following vertices: the R isolated vertices; for each variable u_i, the vertex x_i, and exactly one of u_i or $\overline{u_i}$; and for each clause c_j, the vertex y_j and exactly one of r_j, s_j, or t_j. Note also that if the formula is satisfiable, then

$$IR(L_F) = (4N + R)/(4NK + R) = \frac{1 + (Kp - 1)/(1 - p)}{K + (Kp - 1)/(1 - p)} = p$$

Denote the number of edges in L_F by $L(N, K)$. It is not hard to check that $L(N, K) = T(K)N$ where T is a function of K and independent of N.

Define the vertex set W as follows:

$$W = (\bigcup_{i=1}^{N} \{u_1, \overline{u_i}, x_i\}) \cup (\bigcup_{j=1}^{N} \{r_j, s_j, t_j, y_j\}) \cup (\bigcup_{k=1}^{R} \{v_k\})$$

Let $\mathcal{E}$ denote the set of all edges in the complement of L_F, except for those joining any two vertices in W. Specifically, $\mathcal{E}$ consists of the union of all of the following sets:

1. For each i_1, i_2, $1 \leq i_1 < i_2 \leq N$, the set of all edges joining each vertex in $A_{i_1} \cup B_{i_1} - \{u_{i_1}, \overline{u_{i_1}}, x_{i_1}\}$ to each vertex in $A_{i_2} \cup B_{i_2} - \{u_{i_2}, \overline{u_{i_2}}, x_{i_2}\}$.

2. For each j_1, j_2, $1 \leq j_1 < j_2 \leq N$, the set of all edges joining each vertex in $C_{j_1} \cup D_{j_1} - \{r_{j_1}, s_{j_1}, t_{j_1}, y_{j_1}\}$ to each vertex in $C_{j_2} \cup D_{j_2} - \{r_{j_2}, s_{j_2}, t_{j_2}, y_{j_2}\}$.

3. For each i, j, $1 \leq i,j \leq N$, the set of all edges joining each vertex in in $A_i \cup B_i - \{u_i, \overline{u_i}, x_i\}$ to each vertex in $C_j \cup D_j - \{r_j, s_j, t_j, y_j\}$

4. For each i_1, i_2, $1 \leq i_1 \neq i_2 \leq N$, the set of all edges joining each of x_{i_1}, u_{i_1}, and $\overline{u_{i_1}}$ to each vertex in $A_{i_2} \cup B_{i_2} - \{u_{i_2}, \overline{u_{i_2}}, x_{i_2}\}$.

5. For each j_1, j_2, $1 \leq j_1 \neq j_2 \leq N$, the set of all edges joining each of y_{j_1}, r_{j_1}, s_{j_1}, and t_{j_1} to each vertex in $C_{j_2} \cup D_{j_2} - \{r_{j_2}, s_{j_2}, t_{j_2}, y_{j_2}\}$.

6. For each i, j, $1 \leq i,j \leq N$, the set of all edges joining each of x_i, u_i, and $\overline{u_i}$ to each vertex in $C_j \cup D_j - \{r_j, s_j, t_j, y_j\}$

7. For each i, j, $1 \leq i,j \leq N$, the set of all edges joining each of y_j, r_j, s_j, and t_j to each vertex in $A_i \cup B_i - \{u_i, \overline{u_i}, x_i\}$.

8. For each i, k, $1 \leq i \leq N$, $1 \leq k \leq R$, the set of all edges joining v_k to each vertex in $A_i \cup B_i - \{u_i, \overline{u_i}, x_i\}$.

9. For each j, k, $1 \leq j \leq N$, $1 \leq k \leq R$, the set of all edges joining v_k to each vertex in $C_j \cup D_j - \{r_j, s_j, t_j, y_j\}$.

So the number of edges in $\mathcal{E}$ is

$$N(N-1)(2K-3)^2/2 + N(N-1)(2K-4)^2/2 + N^2(2K-3)(2K-4)$$

$$+3N(N-1)(2K-3) + 4N(N-1)(2K-4) + 3N^2(2K-4) + 4N^2(2K-3)$$

$$+RN(2K-3) + RN(2K-4)$$

$$= (N^2-N)((2K-3)^2 + (2K-4)^2)/2 + N^2(2K-3)(2K-4) + (N^2-N)(14K-25)$$

$$+N^2(14K-24) + 4N^2(4K-7)(Kp-1)/(1-p)$$

Let H_F be the graph formed by adding to L_F all of the edges in $\mathcal{E}$. Let $H(N,K)$ be the number of edges of H_F. Note that because no edges have been added joining any two vertices in W, it is again the case that F is a satisfiable formula if and only if H_F has a maximum independent set of size $4N+R$ with the same properties as described above for L_F.

For fixed K, $L(N,K)/ME(4NK+R) \to 0$ as $N \to \infty$. It can be checked that $H(N,K)/ME(4NK+R) \to D(K)$ as $N \to \infty$, where

$$D(K) = \frac{(2K-3)^2/2 + (2K-4)^2/2 + (2K-3)(2K-4) + 2(14K-25) + 4(4K-7)(Kp-1)/(1-p)}{8K^2 + 16K(Kp-1)/(1-p) + 8(Kp-1)^2/(1-p)^2}$$

It can also be checked that as $K \to \infty$, $D(K) \to 1-p^2$. We have $0 < d < 1-p^2$. Let ϵ be a positive number less than $(1-p^2)-d$. Choose K large enough so that $D(K) > d + \epsilon/2$. We then have that for this choice of K (which depends only on p and d) for large enough N,

$$L(N,K)/ME(4NK+R) < d < H(N,K)/ME(4NK+R)$$

To finish the construction of G_F, if N is large enough, let $E = \lfloor dME(4NK + R) \rfloor$ and add exactly $E - L(N, K)$ of the edges in $\mathcal{E}$ to L_F. (If N is not large enough, add as many edges as possible). Call this graph G_F.

So we have a reduction which given a formula F, produces a graph G_F, such that F is satisfiable if and only if G_F has a maximum independent set of size pn where n is the number of vertices of G_F. Moreover, for large enough formulas F, the graph G_F has near-edge-density equal to d. □

Corollary 5.5 *Let p be a rational number, $0 < p < 1$, and d a real number such that $0 < d < 1 - p^2$. Then the set of all connected graphs having n vertices, maximum independent set of size pn, and near-edge-density equal to d, is hard with respect to the maximum independent set problem.*

Proof: It is not hard to see that enough edges may be added to the graph L_F to make it connected, while preserving the limit $L(N, K)/ME(4NK + R) \to 0$ as $N \to \infty$. □

The generators considered in this paper produce graphs for which the average degree of vertices in the (constructed) maximum independent set equals the average degree of vertices not in this set. We therefore next consider graphs having n vertices and a given maximum independent set I of size pn, where the average degree of vertices in I is equal to the average degree of vertices not in I. Let the graph $G = (V, E)$ have n vertices, m edges, and maximum independent set I of size pn. Let m_1 be the number of edges joining two vertices in $V - I$, and m_2 be the number of edges joining a vertex in $V - I$ to a vertex in I. We will refer to the edges in the first set as type 1 edges, and to the edges in the second set as type 2 edges. If the average degree of vertices in I and $V - I$ are equal, we must have $(2m_1 + m_2)/(n - pn) = m_2/(pn)$, and hence if $p \neq 1/2$

$$m = \frac{m_2}{2p} = \frac{m_1}{1 - 2p}$$

In particular, $2pm$ must be an integer. Also $p \leq 1/2$, and if $p = 1/2$, then $m = m_2$ and $m_1 = 0$. The first fact motivates the following definition.

Definition 5.4 Let d be a real number, p a rational number. Let $2p = r_1/r_2$ where r_1 and r_2 are integers which are relatively prime. We say a graph with n vertices has p-close-density equal to d if it has $r_2\lfloor dn(n-1)/(2r_2) \rfloor$ edges.

(In other words, the number of edges should be a multiple of r_2).

Theorem 5.6 *Let p be a rational number, $0 < p \leq 1/2$, and d a real number such that $0 < d \leq 1 - p$. Then for sufficiently large n, if pn is an integer, there exists a graph $G = (V, E)$ with n vertices, p-close-density d, and maximum independent set I of size pn, such that the average degree of vertices in I is equal to the average degree of vertices in $V - I$.*

Proof: Let n be a positive integer such that pn is an integer. From [17] we know that there exists a graph $G = (V, E)$ with n vertices and maximum independent set I of size pn, having m edges, m_1 type 1 edges, and m_2 type 2 edges, as long as m_1's value is between $Min_1(n, p) = \kappa(n - pn, pn)$ and $Max_1(n, p) = (n - pn)(n - pn - 1)/2$, and m_2's value is between $Min_2(n, p) = n - pn$ and $Max_2(n, p) = pn(n - pn)$.

If the average degree of vertices in I is equal to the average degree of vertices in $V - I$, then as seen above we must have $m_1 = ((1-2p)/2p))m_2$. It can be checked that $Min_1(n, p) \geq ((1-2p)/2p)Min_2(n, p)$, and that $Max_1(n, p) > ((1-2p)/2p)Max_2(n, p)$. Hence (assuming for the moment that $p < 1/2$), m_2 is actually restricted to values between $(2p/(1 - 2p))Min_1(n, p)$ and $Max_2(n, p)$. Therefore m can take any value which is between $Min_1(n, p)/(1 - 2p)$ and $Max_2(n, p)/(2p)$ as long as $2pm$ is an integer. It is not hard to see that

$$\frac{Min_1(n, p)/(1 - 2p)}{ME(n)} \to 0 \text{ as } n \to \infty$$

Also

$$\frac{Max_2(n, p)/(2p)}{ME(n)} = \frac{(n - pn)(pn)}{(2p)n(n - 1)/2} = \frac{(1 - p)n^2}{n^2 - n} > 1 - p$$

Hence it is possible to choose n large enough so that

$$\frac{Min_1(n, p)}{1 - 2p} \leq \lfloor dME(n)/r_2 \rfloor r_2 < \frac{Max_2(n, p)}{2p}$$

(Recall $2p = r_1/r_2$ where r_1, r_2 are relatively prime). When $p = 1/2$, we have $Min_1(n, p) = 0$ and in fact we must have $m_1 = 0$, so m_2's lower bound is $Min_2(n, p)$; the result still follows. □

Corollary 5.7 *Let p be a rational number, $0 < p \leq 1/2$, and d a real number such that $0 < d \leq 1 - p$. Then for sufficiently large n, if pn is an integer, there exists a connected graph $G = (V, E)$ with n vertices, p-close-density d, and maximum independent set I of size pn, such that the average degree of vertices in I is equal to the average degree of vertices in $V - I$.*

Proof: The proof is similar to that of Corollary 5.2. □

Corollary 5.8 *Let p be a rational number, $0 < p \leq 1/2$, and $\epsilon > 0$. Then for sufficiently large n, if pn is an integer, and $G = (V, E)$ is a graph with n vertices and maximum independent set I of size pn, where the average degree of vertices in I equals the average degree of vertices in $V - I$, then the edge density of G is at most $1 - p + \epsilon$.*

Proof: From the above proof it can be seen that $(Max_2(n, p)/(2p))/ME(n) \to 1 - p$ as $n \to \infty$. □

When $p = 1/2$, graphs of this type are easily solvable. If the graph $G = (V, E)$ has n vertices, and maximum independent set I of size $n/2$, such that the average degree of vertices in I equals that of vertices in $V - I$, then there are no type 1 edges, and G must in fact be bipartite. Hence the set I may be found in polynomial time.

Theorem 5.9 *Let p be a rational number, $0 < p < 1/2$, and d a real number such that $0 < d < 1-p$. Then the set of all graphs $G = (V, E)$ having n vertices, maximum independent set I of size pn, such that the average degree of vertices in I is equal to the average degree of vertices in $V - I$, and p-close-density d, is hard with respect to the maximum independent set problem.*

Proof: Let $\mathcal{S}$ be the set of graphs described in the statement of the theorem. We will show that if there is a polynomial-time approximation algorithm A which gives the correct answer for all graphs in $\mathcal{S}$, then 3SAT can be solved in polynomial time.

Let F be a 3SAT formula. Construct the graph L_F as described in the proof of Theorem 5.4. Let the set of edges $\mathcal{E}$ be defined as in that proof as well, and let H_F be constructed as in that proof.

If F is satisfiable then both L_F and H_F have a maximum independent set I of size $4N+R$ as described in the previous proof. Let $L_1(N, K)$ be the number of type 1 edges in L_F, and $L_2(N, K)$ be the number of type 2 edges in L_F, with respect to the set I. We have

$$L_1(N, K) = 4N(K-1)(K-2)/2 + N(K-1)(2K-5) + t_1$$

$$L_2(N, K) = 4N(K-1) + N(2K-5) + t_2$$

The quantities t_1, t_2 are unknown since we do not know which of the vertices u_i or $\overline{u_i}$, or which of r_j, s_j, or t_j are in I. But we do know that $t_1 + t_2 = 3N$ and that $t_2 \geq N$. In any case, for fixed K, we have

$$\frac{Max(L_1(N, K)/(1-2p), L_2(N, K)/2p)}{ME(4NK + R)} \to 0 \text{ as } N \to \infty$$

Assuming again that F is satisfiable, let $H_1(N, K)$ and $H_2(N, K)$ be the number of type 1 and type 2 edges, respectively, in the graph H_F, with respect to the independent set I. So $H_1(N, K)$ equals $L_1(N, K)$ plus the number of edges included in items (1), (2), and (3) and some the edges included in items (4), (5), (6), and (7) of the definition of the set $\mathcal{E}$; while $H_2(N, K)$ equals $L_2(N, K)$ plus the number of edges included in items (8) and (9) and some of the edges included in items (4), (5), (6), and (7) of the definition of $\mathcal{E}$.

It can be checked that if K is large enough, then for sufficiently large N, $H_1(N, K)/(1-2p) \geq H_2(N, K)/2p$. We also have that $(H_2(N, K)/2p)/ME(4NK+R) \to B(K)$ as $N \to \infty$ where

$$B(K) = \frac{(16K-28) + 4(4K-7)(Kp-1)/(1-p)}{2p(8K^2 + 16K(Kp-1)/(1-p) + 8(Kp-1)^2/(1-p)^2)}$$

and $B(K) \to 1-p$ as $K \to \infty$. We have $0 < d < 1-p$. Let ϵ be a positive number less than $(1-p)-d$. Choose K large enough so that $B(K) > d + \epsilon/2$, and also large enough so that $H_1(N, K)/(1-2p) \geq H_2(N, K)/2p$ (for sufficiently large N). We then have that for this choice of K (which depends only on p and d) for large enough N,

$$Max(L_2(N, K)/(2p), L_1(N, K)/(1-2p)) < \lfloor dME(4NK + R)/r_2 \rfloor r_2 < H_2(N, K)/(2p)$$

Let $m = \lfloor dME(4NK+R)/r_2 \rfloor r_2$, $m_1 = (1-2p)m$ and $m_2 = 2pm$. Then it is possible to add enough edges of each type from $\mathcal{E}$ to L_F, so that the graph will have m_1 edges of type 1, and m_2 edges of type 2. There are however two problems.

The first is that we do not know how many of the edges described in items (4), (5), (6), and (7) of the definition of $\mathcal{E}$ are actually type 1 or type 2 edges. However, it can be checked that the total number of edges included in these items, when divided by $ME(4NK+R)$, has a limit equal to $C(K)$ as $N \to \infty$, where $C(K) \to 0$ as $K \to \infty$. Hence if K is chosen large enough, there is no need to use the edges described in these items.

The second problem is that we do not know exactly how many edges of each type L_F has (in the case that F is in fact satisfiable), since we do not know the values of t_1 and t_2. We therefore try all possibilities.

For $r = N, N+1, ..., 3N$, let $E_{2r} = m_2 - 4N(K-1) - N(2K-5) - r$, and let $E_{1r} = m_1 - 4N(K-1)(K-2)/2 - N(K-1)(2K-5) - (3N-r)$. Let G_F^r be the graph obtained by adding E_{1r} edges of type 1 and E_{2r} edges of type 2 from $\mathcal{E}$ to L_F. Run algorithm A on each graph G_F^r.

Now if the formula F is not satisfiable, none of these graphs will have a maximum independent set of size $4N+R$, so the algorithm A must return a value smaller that this. On the other hand, if F is satisfiable, then at least one of the graphs is in the set $\mathcal{S}$, and therefore if A gives the correct answer for graphs from $\mathcal{S}$, A must return $4N+R$ for at least one of the graphs G_F^r. Thus we may determine whether or not F is satisfiable. $\square$

Theorem 5.10 *Let p be a rational number, $0 < p < 1/2$, and d a real number such that $0 < d < 1-p$. The set of graphs produced by SimParD having n vertices, maximum independent set of size pn, and p-close-density d, is hard with respect to the maximum independent set problem.*

Proof: The result will follow if we can modify the construction of the graphs in the preceding proof, so that, in the case when the formula F is satisfiable, the constructed graphs can be seen to be composed of the disjoint union of cliques of at most two sizes differing by 1, with additional edges, each one incident on at least one of the cover (non independent set) vertices.

We do this by first adding some edges to L_F. Let $p = p_1/p_2$ where p_1 and p_2 are integers and relatively prime. Recall that K is chosen large enough so that $B(K)$ is close enough to its limit $1-p$. We now also require that $K = l(p_2 - p_1) + 1$ where l is some integer. It can then be shown that $(Kp-1)/(1-p) = lp_1 - 1$, and hence $R = 4N(lp_1 - 1)$. There are $4N$ cliques of size K in the graph. We will connect $lp_1 - 1$ of the R isolated vertices to each of the cliques.

Without loss of generality, let Q be one of the cliques, and $v_1, ..., v_{lp_1 - 1}$ the vertices to be connected to Q. Let z_1, z_2, z_3 be three of the vertices in Q, chosen such that: if $Q = A_i$ for some i, then $z_1 = u_i$ and $z_2 = \overline{u_i}$; if $Q = B_i$, then $z_1 = x_i$; if $Q = C_j$ for some j, then $z_1 = r_j$, $z_2 = s_j$, and $z_3 = t_j$; and if $Q = D_j$, then $z_1 = y_j$. The vertices in Q plus the isolated vertices comprise $K + lp_1 - 1 = lp_2$ vertices. Let q and r be the quotient and remainder when lp_2 is divided by lp_1. So $lp_2 = qlp_1 + r$ where $0 \le r < lp_1$.

Divide up the lp_2 vertices into r groups of size $q+1$, and $lp_1 - r$ groups of size q in such a way that: all but one of the groups contain exactly one of the isolated vertices; and the remaining group contains the vertices z_1, z_2, z_3. This can be done as long as $q \geq 2$, and if $q = 2$ then $r > 0$. We have

$$q = 1/p - r/(lp_1) > 1/p - 1 > 1$$

since $p < 1/2$. Suppose that $q = 2$. Then we have $lp_2 = 2lp_1 + r$, implying that $p = 1/2 - r/(2lp_2)$ and hence $r > 0$ since $p < 1/2$.

Finally, connect each of the isolated vertices to each vertex in Q belonging to its group.

Let L'_F be the graph obtained by adding edges connecting all of the isolated vertices to their respective cliques in this way. The edges added are all type 2 edges in the case that F is satisfiable. Their number is less than RK and hence a constant multiple of N once K is chosen. It follows that the limits derived in the above proof continue to hold and the rest of the proof proceeds in a similar manner. □

We next show that the graphs produced by SimParD(0) are NP-hard. Since to construct these graphs, a number of edges must be left out of the graph, we cannot achieve all valid densities as in previous proofs. The interested reader may calculate the upper bound on the densities that can be achieved, and check that it is a function of p.

Theorem 5.11 *Let p be a rational number, $0 < p < 1/2$. The set of graphs produced by SimParD(0) having n vertices, and maximum independent set of size pn, is hard with respect to the maximum independent set problem.*

Proof: We modify the above proof as follows. After constructing the graph L'_F, we remove from each clique Q any edges joining vertices assigned to different groups. Denote by L''_F the graph formed in this manner. Let P_1 consist of all vertices originally in the A_i and D_j cliques, together with the (originally isolated) vertices attached to these cliques. Let P_2 consist of all other vertices. Note that all edges of L''_F which are not within the cliques, join a vertex in P_1 to a vertex in P_2. Thus F is satisfiable if and only if L''_F has an independent set of size $4NK + R$, and in this case L''_F belongs to the set generated by SimParD(0). □

Erratum

Concerning the remark in section 4.3 regarding the performance of the benchmark algorithm for the Second Dimacs Challenge, further experimentation has shown that experimentally hard graphs for this algorithm can be generated by using higher values of r than those used in the preliminary experiments.

References

[BGS87] D.H. Ballard, P.C. Gardner, and M.A. Srinivas. Graph problems and connectionist architectures. Technical report, Department of Computer Science, University of Rochester, Rochester, NY, 1987.

[1] M. Brockington, J.C. Culberson, and P. VanBeek, 1993. Camouflaging Independent Sets in Quasi-Random Graphs. Presented at the Second DIMACS Algorithm Implementation Challenge Conference, October 1993.

[2] B. Bollobás and P. Erdös. Cliques in random graphs. *Proc. Camb. Phil. Soc.*, 80:419–427, 1976.

[3] A.G. Barto, R.S. Sutton, and C.W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:834–846, 1983.

[4] R. Carraghan and P.M. Pardalos. An exact algorithm for the maximum clique problem. *Operations Research Letters*, 9:375–382, 1990.

[5] M.R. Garey and D.S. Johnson. *Computers and intractability, a guide to the theory of NP-completeness*. W.H. Freeman, New York, 1979.

[6] J. Hertz, A. Krogh, and R.G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, 1991.

[7] J.J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences, USA*, 79, 1982.

[8] J.J. Hopfield. Neurons with graded responses have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences, USA*, 81, 1984.

[9] A. Jagota. Efficiently approximating Max-Clique in a Hopfield-style network. In *International Joint Conference on Neural Networks*, volume 2, pages 248–253, New York, June 1992. Baltimore, June, IEEE.

[10] A. Jagota. *The Hopfield-clique Network, Associative Memories, and Combinatorial Optimization*. PhD thesis, State University of New York at Buffalo, Department of Computer Science, 224 Bell Hall, Amherst, NY 14260, 1993. Also available as Technical Report TR 93-12 from above address.

[11] A. Jagota and K.W. Regan. Performance of max-clique heuristics under description-length weighted distributions. Technical Report 92-24, Department of Computer Science, SUNY at Buffalo, Buffalo, NY, September 1992. Available from author.

[12] D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by Simulated Annealing: An Experimental Evaluation, Part II (Graph Coloring and Number Partitioning). Operations Research, 39:378–406, 1991.

[13] J.H.M. Korst and E.H.L. Aarts. Combinatorial optimization on a Boltzmann machine. *Journal of Parallel and Distributed Computing*, 6:331–357, 1989.

[14] S. Kirkpatrick, C.D. Gelatt Jr., , and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220, 1983.

[15] E.M. Palmer. *Graphical evolution*. Wiley, New York, 1985. (Matula's theorem is on page 76).

[16] L. Sanchis. On the complexity of test case generation for NP-hard problems. *Information Processing Letters*, 36:135–140, 1990.

[17] L. Sanchis. Generating hard and diverse test sets for NP-hard graph problems. *Discrete Applied Mathematics*, to appear.

[18] L. Sanchis. Test case construction for the vertex cover problem. In *Proceedings of the DIMACS Workshop on Computational Support for Discrete Mathematics*, March 1992.

[19] Y. Shrivastava, S. Dasgupta, and S.M. Reddy. Neural network solutions to a graph theoretic problem. In *Proceedings of IEEE International Symposium on Circuits and Systems*, pages 2528–2531, New York, 1990. IEEE.