

Characterizing the Search Space of the Clique Problem in Random Graphs using “Go with the Winners”

Tassos Dimitriou*

Abstract

We consider the problem of investigating the structure of the space of all possible cliques in random graphs generated according to the $G_{n, \frac{1}{2}, K}$ model. This model consists of all graphs on n nodes and edge probability $1/2$, in which a random set of K nodes is forced to be a clique.

In this work we make a first attempt to explain the hardness of the problem by revealing the *combinatorial characteristics* of the space of all possible cliques for graphs generated according to the above distribution. Our main tool is “Go with the winners”, an optimization heuristic that uses many particles that independently search the space of all possible solutions. In particular, we consider how the search space decomposes into smaller regions of related solutions by imposing a quality threshold to them. If these regions possess a combinatorial property, the so called “local expansion”, then these regions can be effectively sampled by using enough particles and thus discover the optimal solution.

Most importantly however, sampling can be used to deduce properties of the search space. These properties can then help optimize heuristic performance and design heuristics that take advantage of this information. Thus the goal of this work is not to compare clique-finding heuristics but to *exhibit a way to reveal the combinatorial characteristics of the search space, verify this information experimentally and use it to design good heuristics.*

1 Introduction

A *clique* of size K in a graph G is a complete subgraph on K nodes, that is a set of nodes any two of which are connected by an edge. The clique problem is that of determining the largest K for which there exists a clique of size K in the graph.

While this problem is one of the fundamental problems in Computer Science, the problem is known to be NP-complete[17, 12], so we cannot expect to have good performance in the worst case. The best known approximation algorithm for general graphs is due to Boppana

and Halldórsson[6] and has a performance guarantee of $O(n/(\log n)^2)$, where n is the number of vertices in the graph.

On the negative side, it was shown by a series of results[3, 4] that it is impossible to approximate, in polynomial time, the size of the largest clique within a factor of n^ϵ , for some constant ϵ , assuming $P \neq NP$. As these results apply to the worst case it is only natural to study the problem for random graphs drawn from specific distributions. The $G_{n,p}$ model consists of all graphs G with n vertices and edge probability p . The most frequently examined case is for $p = \frac{1}{2}$. It is known (see for example[5]) that almost surely the size of the largest clique in the $G_{n, \frac{1}{2}}$ model is $2 \log_2 n - O(\log \log n)$.

There are several simple polynomial time algorithms that find a clique of size $\log_2 n$, that is about half the size of the largest one. However, many attempts at designing randomized algorithms that can beat this bound, i.e. discovering a clique of size at least $(1 + \epsilon) \log n$, for any fixed $\epsilon > 0$, have met no success. The problem of finding such an algorithm was suggested by Karp[18]. His results implied that several algorithms do not achieve this bound and was conjectured that no polynomial time algorithm can find, with high probability, a clique of size bigger than $(1 + \epsilon) \log_2 n$. In support of this conjecture, Jerrum[13] demonstrated the existence of an initial state for which the Metropolis algorithm (Simulated Annealing with a fixed temperature) needs super-polynomial time $n^{\Omega(\log n)}$ to find a clique of size $(1 + \epsilon) \log_2 n$, for any constant $\epsilon > 0$.

A similar model to the previous one is $G_{n, \frac{1}{2}, K}$ which assumes the existence of a clique of size K in the graph. The graph is generated according to $G_{n, \frac{1}{2}}$ and a random subset of K nodes is forced to be a clique. It is easy to find the largest clique in this model for $K = \Omega(\sqrt{n \log n})$ as this clique almost surely contains the vertices with the largest degrees[19]. Recently, Alon, Krivelevich and Sudakov[2] improved this bound to $K = \Omega(\sqrt{n})$ using an algorithm based on the spectral properties of the graph. It is still open however to find algorithms that work for smaller values of K . Even the problem of finding a clique of size $(1 + \epsilon) \log_2 n$ in $G_{n, \frac{1}{2}, K}$, for $K = o(\sqrt{n})$, as suggested by Jerrum[13], remains a challenge. The sig-

*Department of Mathematics, University of Athens and Technical University of Chania, Greece. E-mail: tdimitr@math.uoa.gr

nificance of studying the $G_{n, \frac{1}{2}, K}$ model arises from the fact that the two models induce the same probability distribution on random graphs when $K \approx 2 \log_2 n$ [16]. This way, by knowing the hidden clique size, we can evaluate the performance of any algorithm in discovering large cliques. It is thus verified that approximating the largest clique in a graph remains a challenging problem and further results will provide significant breakthroughs. Moreover, the difficulty of the problem seems to be confirmed by a number of experimental results, see for example [15].

In this work we make a first attempt to explain the hardness of the problem by discovering the *combinatorial characteristics* of the space of all possible cliques for graphs generated according to the above distributions. We do this by considering how this search graph decomposes into smaller regions of related solutions by imposing a quality threshold to them. Our main tool will be the “Go with the winners” (GWW) strategy, introduced in [1] as a method of searching trees for good nodes and modified in [8] as an optimization heuristic.

The algorithm uses many particles that independently search the search graph for a solution of large value. Dimitriou and Impagliazzo [9] were able to relate the performance of GWW with the existence of a combinatorial property of the search graph, the so called “local expansion”. Intuitively, local expansion means that a random walk on the space of solutions quickly converges to the stationary distribution and never restricts itself into a small portion of the search space.

The main result of [9] was that if the subgraphs of high value solutions have this property, then these subgraphs can be effectively sampled by using enough particles and thus locate the optimal solutions. Furthermore, if local expansion holds, particles remain uniformly distributed and sampling can be used to deduce properties of the search space. These properties can then help optimize heuristic performance and design heuristics that take advantage of this information.

The goal of this work is then not just to compare the performance of various heuristics in finding cliques but *provide information about the combinatorial characteristics of the search space, verify this information experimentally and use it to design effective algorithms.*

The rest of this abstract is organized as follows: In Sections 2 and 3 we review the basic algorithm and the type of random graphs we’ll be working with. Sections 4 and 5 are implementation oriented. The former describes the search space by defining the neighbors of a potential solution, while the latter describes the particular implementation of the algorithm. Finally, Section 6 contains the various experiments that we use to characterize the search space.

2 Go with the winners Strategy (GWW)

Most optimization heuristics (Simulated Annealing [14], Greedy, WalkSat [20], etc.) can be viewed as *strategies*, possibly probabilistic, of moving a particle in a search graph, where the goal is to find a solution of optimal value. The placement of the particle in a node of a search graph, corresponds to an examined solution.

The GWW algorithm was first introduced by Aldous and Vazirani [1] as a method for *searching trees* for deep nodes. A population of particles was initialized at the root and at every stage of the algorithm these particles advanced to deeper levels of the tree. If, however, a particle ended its search at a leaf, this particle was moved to the position of a particle that continued its exploration of the tree. Aldous and Vazirani determined the population size, in terms of a measure of the tree’s imbalance so that the deepest node can be found with high probability.

Dimitriou and Impagliazzo [8] modified this algorithm so that it can be used directly for *combinatorial optimization*. The algorithm has the same flavor as the algorithm for trees and is shown below for the concrete setting of searching for large cliques.

GWW for Cliques

Denote by \mathcal{N}_i the part of the solutions space during stage i of the algorithm, i.e. the space of K -node sets with number of edges greater than or equal to i , where K is the size of the hidden clique.

- Stage 0 (Initialization): Generate B random solutions and place particles on each one of them.
- Stage i (Main loop): Proceed in two phases.
 1. In the *redistribution* phase, replace each particle of value $i - 1$ with a duplicate of a random particle of value $\geq i$. If all particles have value $i - 1$, stop.
 2. In the *randomization* phase, for each particle, perform an S steps *random walk* in \mathcal{N}_i as follows: Select a neighbor of the current solution. If its value is $\geq i$, move the particle to this neighbor, otherwise leave the particle in the current solution.
 3. Go to the next stage, by raising the threshold i .
- Output the best solution found.

The algorithm uses B particles that independently

search the space of all solutions. The particles however interact with each other in the following way: each “dead” particle (a particle that ended its search at a local optimum) is moved to the position of a randomly chosen particle that is still “alive”. The goal is to discover a K -set (a set of K nodes) with maximum number of edges in it (see also Section 4 for the definition of the neighborhood structure of solutions). Thus the algorithm tries to maximize the number of edges in sets of predefined size.

The algorithm uses two parameters: the number of particles B and the length S of the random walk each particle performs. The intuition of the algorithm is that in the i -th stage and beyond we are implicitly deleting all K -sets of size (number of edges) smaller than i from the space of all possible solutions. This divides the solutions space into components of potential cliques with number of edges bounded below by i . The redistribution phase will make sure that particles in locally optimal solutions will be distributed among non-local solutions, while the randomization phase will ensure that particles remain *uniformly distributed* as they advance to deeper and deeper levels of the search space. The invariant the algorithm maintains is that at the end of stage i , all particles are uniformly distributed inside \mathcal{N}_i , the set of all solutions with value $\geq i$.

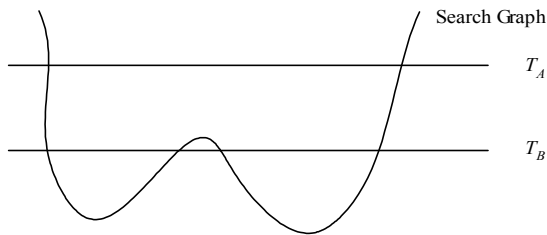


Figure 1: Decomposing the search graph by increasing the threshold.

This is shown abstractly in Figure 1 for a part of a search space and two thresholds T_A and T_B . When the algorithm is in stage T_A , all solutions of value larger than T_A form a connected component and particles are uniformly distributed inside this large component. However, when the threshold is increased to T_B , the search graph decomposes into smaller components and the goal is to have particles in each one of these in *proportion* to their sizes.

In [9], Dimitriou and Impagliazzo were able to characterize the search spaces where GWW works well in terms of a combinatorial parameter, the *local expansion* of the search space. Intuitively, this property suggests that if a particle starts a random walk in \mathcal{N}_i , then the resulting solution will be uncorelated to its start. If the search space has the local expansion property, particles will remain uniformly distributed inside the space and

it will be unlikely that they will get trapped into small regions of it. This property was proved to be true for the problem of bisections of a graph and in the subsequent work of [7], it was also verified experimentally, giving rise to heuristics that take advantage of the good expansion properties of the space of bisections.

Preliminary work[10] has shown that the space of all cliques has local expansion, although in not such a strong form as the space of bisections. In this work we would like to test these findings by revealing the characteristics of the space of all possible cliques.

Studying the behavior of the algorithm with respect to its two important parameters, population B and random walk length S , will indicate if the search graph has good expansion properties. In particular, performance of the algorithm as a function of the population size will provide information about the connectedness of \mathcal{N}_i , while performance as a function of the walk length will indicate if \mathcal{N}_i locally expands. Studying the relative importance of the two parameters will offer a tradeoff between using larger populations or longer walks, which may give rise to implementations of the algorithm which produce high quality solutions, even when some of these parameters are kept small.

3 Choice of Distribution

Since we want to evaluate the performance of an algorithm, the choice of a distribution to generate the random instances is of uttermost importance.

The $G_{n,p}$ model consists of all graphs G with n vertices in which each edge appears with probability p . When $p = \frac{1}{2}$, the size of the largest clique in a graph generated according to this distribution is almost surely $2 \log_2 n - O(\log \log n)$. A similar model to the previous one is $G_{n,\frac{1}{2},K}$ which assumes the existence of a clique of size K in the graph. The graph is generated according to $G_{n,\frac{1}{2}}$ and a random subset of K nodes is forced to be a clique. The two models can be shown to be equivalent when $K \approx 2 \log_2 n$. In particular, in [16] it was shown that if there exists an algorithm that finds a clique of size $(1+\epsilon) \log_2 n$ in $G_{n,\frac{1}{2},K}$, then the same algorithm can find one of size $(1+\epsilon) \log_2 n$, in $G_{n,\frac{1}{2}}$, with high probability.

Our graphs will be generated according to the $G_{n,\frac{1}{2},K}$ model. This way, we will be able to evaluate the quality of the solutions found by the algorithm by comparing them to the hidden clique. However, the choice of K is crucial. The expected number of cliques of value K in the $G_{n,\frac{1}{2}}$ model, is simply

$$E_K = \binom{n}{K} \times 2^{-\binom{K}{2}},$$

since $2^{-\binom{K}{2}}$ is the probability that a random set of K nodes forms a clique. In our experiments, we will set

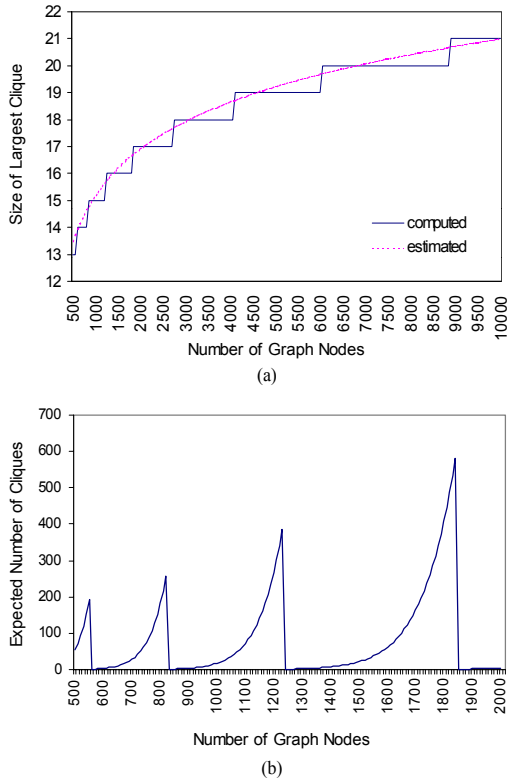


Figure 2: a) Computed and estimated clique sizes. b) Expected number of cliques.

K equal to the *largest* value such that $E_K > 1$. The computed K is shown in Figure 2(a) for various graph sizes. A good estimate for K is given by the quantity $2 \log(ne) - 2 \log \log(n) - 1$, which is plotted against the computed value of K in the same figure. The reason for choosing K such that $E_K > 1$ is simply because we want the two distributions $G_{n, \frac{1}{2}}$ and $G_{n, \frac{1}{2}, K}$ to be comparable. If the size K of the hidden clique is such that $E_K \ll 1$, then the graphs generated according to $G_{n, \frac{1}{2}, K}$ will differ substantially from those generated according to $G_{n, \frac{1}{2}}$, at least with respect to the size of the largest clique.

The expected number of cliques, for this choice of K , is shown in Figure 2(b). As it can be seen from the figure, this number can be much greater than 1 for certain graph sizes. So in order to force the algorithm work hard to find the (possibly unique) largest clique, we will work with graphs in which these cliques are rare. So, for example, a graph with 553 nodes will have on the average 208 cliques of size 14, while a graph with 554 nodes will have only one.

This distribution will help us evaluate the performance of the algorithm by knowing what to expect for the optimal solution. The parameters of the distribution where

chosen this way based on preliminary experiments, which showed that GWW easily locates the hidden clique when its size is different than the one selected above.

4 Search Graphs

“Go with the Winners” is an optimization heuristic that tries to locate optimal solutions in a *search graph*, whose nodes represent all feasible solutions for the given problem. Two nodes in the search graph are connected by an edge, if one solution results from the other by making a small *local* change. Such a search graph should not be confused with the input graph, which is usually exponentially smaller than the former. The search graph is implicitly defined by the problem at hand and doesn't have to be computed explicitly. The only operations required by the algorithm are i) generating a random node (solution) in the search graph ii) compute the value of a given node and iii) list all the neighboring solutions.

In the case of cliques, we will study the search graph whose nodes correspond to all sets of K nodes (we will call such potential solutions, K -sets or potential cliques). Two such sets are connected by an edge in the search graph, if one can result from the other by swapping a vertex in the K -set with one of the $n - K$ vertices not in the set. Thus the number of neighbors of any K -set is at most $K(n - K)$. The *value* of the solution is simply the number of edges in the K -set. Thus the goal of the algorithm is to maximize the number of edges in any set, finding one with $\binom{K}{2}$ edges.

The choice of the search graph is crucial to the performance of the algorithm. Here, we choose to study the search graph defined above. Another search graph for the same problem is one in which neighboring solutions are not defined by swaps of vertices, but by simple additions or deletions of a single vertex. Thus if X is a potential clique and $u \in X, v \notin X$, then $X - \{u\}$ and $X + \{v\}$ are neighbors of X . Thus any set X can have at most n neighbors. The value of solutions has to be changed however. Since our goal is to find a clique on K vertices, one has to penalize solutions of different sizes. This can be done by adding a penalty function that takes into account the number of extra vertices.

This approach can be equally effective because the extra solutions (total 2^n instead of just $\binom{n}{K}$) provide more ways out of local optima. Another benefit is that the smaller neighborhoods allow for faster implementations of the algorithm. This search graph however, will be the subject of subsequent research.

5 Implementation Details

The most time consuming part of the algorithm is the randomization phase in which each particle has to perform an S steps random walk inside the space of solutions, as is determined by the particular stage of the algorithm.

To perform a random step one has to compute explicitly the neighborhood set, store the solutions that have values greater than the required threshold and then select one of them at random and put the particle there. This requires an explicit enumeration of $K(n-K)$ potential neighbors, and so the running time is dominated by this quantity. (We don't count here the time to evaluate the edges of each K -set, which depends on the particular data structure used). Thus, the total time of the algorithm is given by the quantity

$$STAGES \times B \times S \times K(n-K), \quad (1)$$

(modulo the observation above) where $STAGES$ is the total number of stages of the algorithm, which in this case is bounded by $\binom{K}{2}$.

It is obvious that if we have a quick way to pick one of the $K(n-K)$ neighboring solutions, then the computation of all the $K(n-K)$ swaps will no longer be necessary. This can be done by a "randomized approach" to picking the right neighbor. Remember that our goal is to perform an S step random walk, in the space of all solutions with size (number of edges) greater than some threshold T . To perform one step using the "randomized approach", we simply pick one of the $K(n-K)$ potential neighbors at random and check its value. If it is at least T we go there, otherwise we repeat the experiment. How many times do we have to do this in order to perform S valid steps? This depends on the number of neighbors at each stage T . If this number is x_T , then the probability of hitting one of them is $x_T/K(n-K)$ and thus we need $K(n-K)/x_T$ trials on the average just to hit one of them. So, in order to perform S valid steps, we simply need to do this $SK(n-K)/x_T$ times, on average.

As an approximation to x_T we computed the expected number of neighboring solutions at each threshold T using the fact that we are working with the $G_{n, \frac{1}{2}}$ model. This number is plotted in Figure 5 together with the statistics collected by a particular run of the algorithm. As it can be seen from the figure, the two curves match, proving the validity of this approach.

The savings are great: At early stages of the algorithm, where the value of x_T is close to $K(n-K)$, each random step takes constant time! At the last stages, where this value drops to 1, we follow a deterministic approach, computing all $K(n-K)$ neighbors, as the randomized approach cannot tell us if a solution has zero neighbors, no matter how much we try.

6 Experiments

The two important parameters of GWW is population size and random walk length. The algorithm uses many particles, instead of just one, that independently search the space of all possible solutions. The effect of using many particles was studied in [1] in the context of searching for a deep leaf in a tree. It was found there that the use of many particles acts as a probability amplification technique. This means that the probability of finding the deep node in the tree is boosted when having more than one particle interacting with each other.

The other mechanism used by GWW is the random walk that is performed by the particles at every stage of the algorithm. This allows the algorithm to escape from local optima that are encountered when one is using only greedy moves. The use of these two ingredients has the effect of maintaining a uniform distribution of particles throughout the part of the search graph that results by increasing the quality threshold at each stage of the algorithm.

The goal of the experiments is to understand the relative importance of these parameters and the structure of the search graph. The first type of experiments will study the quality of the solutions found by the algorithm as a function of the population size and random walk length. This will reveal the importance of each parameter. (There is also a third parameter that affects the performance of the algorithm, the increase of the threshold at each stage. One can argue that the algorithm may benefit if this change happens according to some schedule and not be fixed as in our case. Due to space requirements however we stick with the smooth increase and leave this question for future consideration.) The purpose of the second type of experiments is to validate the implementation choices of the algorithm and to reveal the expansion characteristics of the search graph. If the search graph has good expansion properties, this will be revealed by a series of tests that indicate that sufficiently long random walks uniformly distribute the particles. Once we have found the optimal random walk length that uniformly distributes the particles, we can study how the search graph decomposes into smaller components at various thresholds. Then, hopefully, these observations can be used to design heuristics that take advantage of the structure of the search space.

In the following experiments the random graphs were first generated according to the $G_{n, \frac{1}{2}}$ model and then a random set of K nodes was forced to be a clique. In particular, we tested a 554 nodes graph with a hidden clique of size $K = 14$. We called this graph G'_{554} . The behavior of the algorithm was found to be similar for many instances of random graphs with 554 nodes, so we decided to proceed with this particular instance. The

expected number of cliques of size 14 on graphs with 554 nodes is 1, and our implementation of GWW found the hidden clique several times.

6.1 Trading Random Steps for Particles

The effect of varying the number of particles B and random walk steps S can be seen in Figure 3. In Figure 3(a), we used different length random walks ($S = 200, 50, 10$ and 1) and we plotted the average number of edges in the K -sets found for increasing number of particles (1 to 100). The size of the solutions found corresponds to the edges of the K -set, with 91 being the maximum number of edges in any clique of size 14. In Figure 3(b) is shown the effect of increasing the length of the random walk, for various population sizes ($B = 100, 50, 10, 5$ and 1). Incidentally, the algorithm discovered three cliques: the hidden one, which always lies, through a renaming of the graph nodes, in the first K vertices of the graph and two others, which consist of the vertices (3, 39, 59, 81, 82, 118, 158, 293, 346, 366, 446, 450, 506, 528) and (128, 139, 157, 185, 287, 291, 330, 352, 378, 379, 427, 437, 462, 533), respectively.

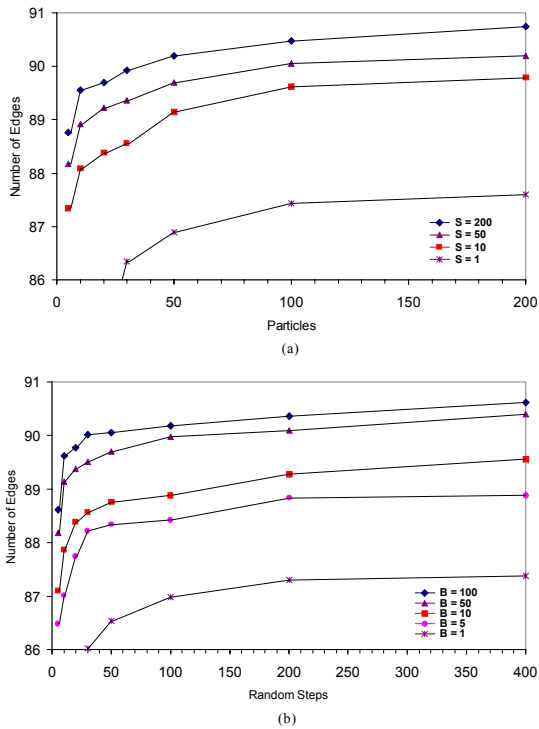


Figure 3: a) Average solution value vs number of particles for various walk lengths. b) Average solution value vs walk steps for various number of particles.

It is evident from these figures that increasing the computational resources (particles or walk steps) has the

effect of improving the quality of the solutions found. Thus, it is not clear if any of these resources is more important than the other. So, in the next type of experiments we proceeded to examine the effect on the quality of solutions found by varying the number of particles while keeping the product $B \times S$ (particles \times random steps) and hence the running time (see equation (1)) constant. The results are shown in Figure 4.

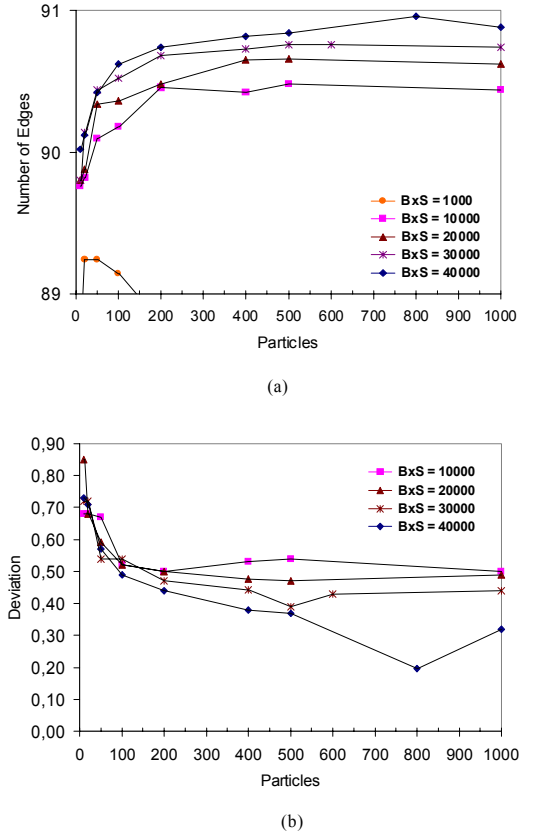


Figure 4: a) Average solution value vs number of particles for different values of $B \times S$. b) Deviation vs number of particles for different values of $B \times S$.

For each value in these figures we performed 50 runs of the algorithm and computed the average value and the deviation of the solutions found. In Figure 4(a) we plotted the sizes of the K -sets found for various values of $B \times S$. It is clear that as the number of particles increases the average value also increases, but there is a point where this behavior stops and the value of solutions starts to decline. This is well understood. As the population becomes large, the number of random steps decreases and this has the effect of not allowing the particles to escape from bad regions.

What is counterintuitive perhaps is that we expected

the execution of longer walks to be more beneficial to the algorithm than having large population sizes. We thought that having a few particles and investing in longer walks would result in discovering better solutions. But this is not the case as it can be seen in the left side of Figure 4(a). Fewer particles and longer walks result in poorer solutions. Thus, it turns out that you need the opposite: Large number of particles and only a few steps. And most surprisingly, the peek value in all the curves, from one extreme $B \times S = 1000$ to the other $B \times S = 40000$, happens when the number of steps is about 40 or 50. This is also true for the deviation in Figure 4(b). The deviation becomes smaller for the same number of steps.

To verify that this type of behavior was not an accidental one and had nothing to do with our choice of graph G_{554} , we repeated the experiments with a second random graph, this one having 830 nodes and a hidden clique of size 15. Again the same pattern of behavior was observed. We computed the average value of solutions found for values of $B \times S$ equal to 50000, 60000 and 90000. The peek value happened again when the number of steps was about 50.

One may deduce from this that the number of steps is not related to the size of the graph but the size of the hidden clique K . So at this point, we would like to *risk* an explanation of this pattern of behavior. Suppose you have a particular solution (a set with K nodes) A and the goal is to obtain from this, through random swaps of vertices, another random K -set B . How many swaps do you need?

First, let's find what would be the properties of such B . We know by standard probabilistic arguments that this set would have an expected K^2/n nodes in common with A , where n is the number of nodes in the graph. Thus, our goal is to perform the walk so that $r = K - K^2/n$ random vertices get removed from A and be replaced with random vertices from the rest of graph nodes.

This walk is reminiscent of sampling with replacement from a population of M distinct elements until r different elements are obtained. In our case, $M = K$ and $r = K - K^2/n$, the vertices that have to be removed. The expected length of random walk in a situation like this is about $M \ln \frac{M+1/2}{M-r+1/2}$ (see also [11, Chapter IX.3]). Replacing the values of M and r , we find that the length of the walk has to be

$$K \ln \frac{K + 1/2}{K^2/n + 1/2} < K \ln \frac{n}{K}.$$

Substituting the values $(K, n) = (14, 554)$ and $(15, 830)$ for the two random graphs we have considered, we obtain respectively 40 and 45 random steps. A very close match to be an accidental one! So we conclude that when the resources are limited, the best behavior occurs when the

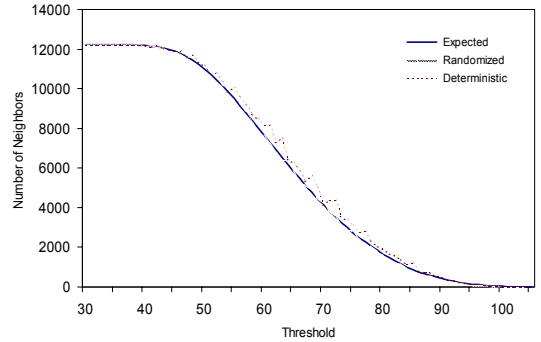


Figure 5: Comparison of number of neighbors using the randomized and deterministic approaches.

number of steps is about $K \ln(n/K)$, where K is the size of the hidden clique.

Incidentally, the algorithm discovered three cliques in G_{830} but this time the two cliques $(0, 1, 3, 4, 6, 7, 8, 9, 10, 13, 14, 290, 564, 663, 680)$ and $(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 663)$ relied heavily on the existence of the hidden clique on the first 15 nodes. While it is not at all certain that these are the only cliques of size 15 in the graph, this is an indication of the validity of our approach to work with graphs where cliques are rare (the expected number of size-15 cliques in G_{830} is one).

6.2 Characterising the Search Graph

In this part we would like first to validate the choice we made about our “randomized approach” to finding neighbors and second, to test if the space of cliques has good expansion properties. In all experiments that follow we used the G_{830} random graph.

The results of the first experiment are shown in Figure 5. We computed the average number of neighbors (degree) of a small population of particles (B is set to 100 particles) as a function of the threshold and compared this with the number of neighbors when the algorithm deterministically computes neighborhood sets. In both runs, S was set to 100. As it can be seen from the figure the two curves match, proving the validity of the randomized implementation. Also in the figure is shown the *expected* number of neighbors a K -set has in the $G_{N, \frac{1}{2}}$ model. Even with this small population, the algorithm is staying close to random.

While the above discussion is an indication that the “randomized approach” works fine, perhaps more important are the tests that examine the uniform distribution of particles inside the search graph. If the graph has good expansion properties, the population will remain uniformly distributed and this can be shown by a series of experiments that measure the right walk length that

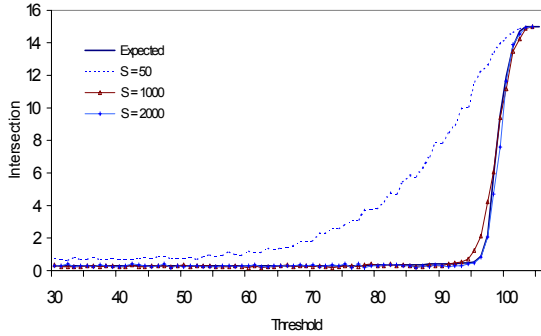


Figure 6: Choosing the right walk length that achieves uniformity: Average intersection between start and end of the random walk.

achieves uniformity.

A good test towards this direction is a one that determines the appropriate walk length inside the space \mathcal{N}_i so that the average number of *common* nodes between the original solution and the one obtained from that solution after performing this random walk matches the one expected in the $G_{n, \frac{1}{2}}$ model. We have argued in the previous section that when the resources are limited, $K \ln(n/K)$ steps are sufficient to achieve good performance. Are they also sufficient to achieve uniformity? The answer is yes, but only in the earlier stages of the algorithm as it is implied by Figure 6 and the curve when $S = 50$. At later stages, when the neighbors become scarce, more steps are required to achieve uniformity. As it can be seen by direct comparison with the expected curve in Figure 6, the results are very close to being uniform when $S = 1000$.

We performed another test to examine the hypothesis that 1000 steps are sufficient to uniformly distribute the particles. In this test (not presented due to space restrictions) we examined the average size (number of edges) of the GWW population at various thresholds and compared with the expected size of K -sets in the $G_{n, \frac{1}{2}}$ model. Again the two curves matched showing that particles remain well distributed.

In Section 6.1, when we studied the tradeoff between particles and random steps, we found that the optimal number of steps is about 50. And we've already seen that the algorithm benefits from using more particles than random steps. So one may ask why this discrepancy with the number of random steps required to achieve uniformity? The answer must lie in the structure of the search graph. We believe that as the space decomposes into components by imposing the quality threshold to solutions, good solutions must reside in pits with high barriers that render long random walks useless. Thus by using more particles the algorithm is able, first, to hit

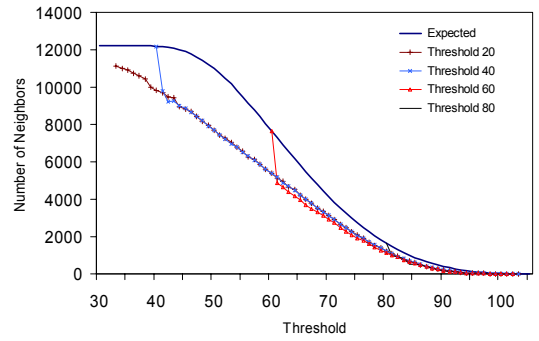


Figure 7: Number of neighbors of greedy search starting from particles at various thresholds.

these pits and then search the neighborhood for good solutions.

We tried to test this hypothesis by performing simple greedy optimization starting from particles at various thresholds and counting the number of neighbors of each greedily obtained solution. If this number is smaller than the expected number of neighbors found by GWW, this is an indication that solutions lie inside deep pits. The results are shown in Figure 7, where the average number of neighbors of greedily obtained solutions is plotted against the expected number of neighbors each solution has in the $G_{n, \frac{1}{2}}$ model. It is clear from the figure that while GWW stays close to random (see also Figure 5), greedily obtained solutions deviate substantially from the expected curve.

We also tried to examine in more detail the neighborhoods of local optima to see if they have the form of a plateau, i.e. if they consist of neighbors with the same size. Our findings support the previous claim that solutions lie in deep pits; these optima seldom had more than a few neighbors with the same value. Walking sideways (visiting solutions with the same value) never uncovered any better optima.

To summarise these findings we have constructed the following picture of the search space: In the early stages of the algorithm solutions form one big component, and therefore a few steps are sufficient to distribute the particles. As the threshold increases, solutions decompose into smaller components in which optima are hidden into deep pits. At these stages more steps are required to achieve good mixing of the population but this process never finds better optima. Furthermore, particles remain trapped in these regions even if we allow random walks that stay at the threshold. Hence, if time permits, it's better to increase the number of particles at the expense of random steps.

6.3 Heuristics Comparison

Based on these observations we believe that heuristics that utilize some form of a threshold to make progress towards better solutions will have better chances to uncover good solutions than heuristics that mix greedy with random steps. Greedy moves tend to trap the particles into deep regions of the search graph from which the algorithm cannot escape, even with the help of random moves, while thresholded moves allow for a more progressive exploration of the search space.

The heuristics we propose to examine are described below:

Greedy with sideways moves: This is a simplified (no restarts) version of the GSAT algorithm of [20]. A particle is maintained which starting from a random solution either moves to a neighbor of better cost or, if none exists, to a neighbor of equal cost. The length of this second type of moves is determined by a parameter which is set to 1000.

Simulated annealing: One particle is maintained which starting from a random solution either moves to a random neighbor of better cost, or if none exists, to a bad neighbor with probability $e^{\Delta/T}$. T is a parameter called temperature which controls the annealing process and Δ is the negative difference of costs. The implementation is based on the description of [14] with parameters set as follows: TEMPFACOR = 0.95, SIZEFACTOR = 5, MINPERCENT = 1, INITPROB = 0.5.

Go with the Winners: The population is set to 1800 particles and the random walk length to 50 steps.

τ -algorithm (τ for threshold): This is based on a simplified variant of GWW[7]. Three particles are maintained starting from a random solution and a threshold initialized to the particle with minimum size. In each step, the particles are moved to a random neighbor provided it is below the threshold. Progress is made by increasing the threshold at each step with a probability that may depend on the average size so that the expected increase is +0.01.

We performed an initial run of these algorithms on the G_{830} random graph and computed the average degrees of solutions discovered at various thresholds. The results are shown in Figure 8. We can see that GWW and τ stay close to random while simulated annealing and greedy deviate, with the latter being the worst of all. Based on this comparison we expect the algorithms' performance to be inversely proportional to their deviation from random.

The most time consuming algorithm was GWW. One run of GWW with the parameters defined previously took approximately 900 seconds. Annealing took 130

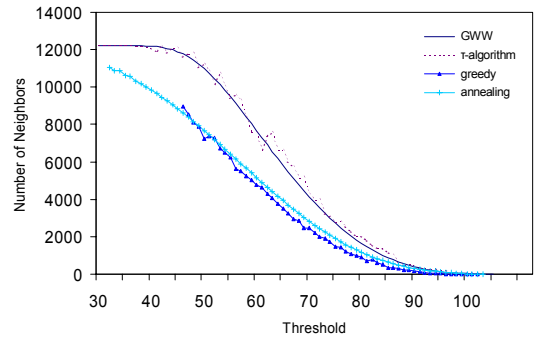


Figure 8: Comparison of number of neighbors for various algorithms. Algorithms that stayed closer to random behaved better.

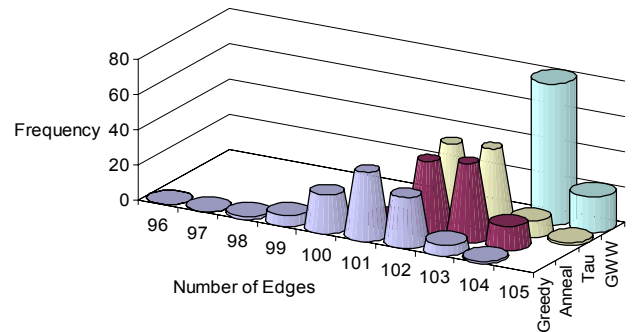


Figure 9: Distribution of solution values found by algorithms on G_{830} normalized for running time.

seconds, the τ -algorithm took 120 seconds and greedy being the fastest of all, only 3.5 seconds.

In order to be fair in our comparisons we normalized for running time. We executed 10 runs of GWW that took approximately 9000 seconds, and we performed the analogous number of runs for each algorithm so that the total time matched that of GWW. Specifically, we performed 70 runs of Annealing, 75 runs of τ and 2600 runs of Greedy. The results are depicted in Figure 9 under the same scale so that comparisons are made possible.

As was expected the worst algorithm is Greedy with solutions ranging from 96 to 104 edges and a mass of 85% located around 101 edges. The mean of the solutions found by Greedy was 1.04 and the standard deviation 1.1. Annealing behaved a lot better and surprisingly matched the behavior of the τ -algorithm. Both algorithms had solutions ranging between 101 and 104 edges with the exception of τ that discovered the largest clique once. (The statistics of these two algorithms are so similar that we believe if more experiments were conducted, annealing would also have discovered the largest clique). Both algorithms had a mean of 102.64 and Annealing's deviation 0.75 was slightly larger than tau's 0.74. Finally,

GWW discovered the hidden clique two times in 10 runs. All solutions found by GWW had sizes between 104 and 105 edges. To verify that this concentration was not an accidental one, we conducted another 30 runs of GWW and again the same behavior was observed: the largest clique was discovered 7 times.

7 Conclusions

In general, the algorithms behaved as we predicted. The performance of Greedy showed that sideways moves are not helpful because there are no plateaus that lead to better solutions. Annealing's behavior was somewhat of a surprise. We didn't expect it to perform so well especially since the average degree of solutions discovered deviated substantially from random. One explanation is perhaps the abundance of cliques of size 14 (one less) than the one we were looking for. Another might be the use of random moves that helps the particle escape from bad regions. Or, there might be other features of the search space that boost the performance of the algorithm. All these observations suggest that more experiments need to be done to reveal more characteristics of the search graph.

The τ -algorithm behaved reasonably well, even though the number of particles used was so limited and convergence was restricted to a small portion of the space. The use of small populations like this is another indication that the likelihood of getting trapped into a local optimum (as is the case for Greedy) becomes smaller as the population increases. This also suggests that the use of GWW with small parameters maybe another good alternative for combinatorial optimization.

GWW performed as expected. Large populations help exploring more of the search space, while the number of steps seems not to be a problem. $K \ln(n/K)$ steps are sufficient to uniformly distribute the particles in the early stages of the algorithm so that they hit the interesting regions of the space later on. More experiments however need to be done to reveal the decomposition of the state space into components at these later stages as well as the neighborhoods of local optima.

Although a complete map would be possible only for smaller search graphs where it is computationally feasible, one can use large populations in bigger graphs to infer the structure of components according to how the particles remain connected as a function of the threshold. We believe that good solutions must lie in components with high barriers but this remains to be shown.

Another interesting line of research is to understand how the use of different search graphs may influence the quality (and the running time) of the solutions found. As we suggested in Section 4 the use of unbalanced

solutions may provide more ways out of local optima and will certainly speedup the algorithm considerably (this was shown experimentally for the problem of graph bisections[14]). Are the expansion properties preserved in this case as well? What about the distribution of particles? Is the overall performance of the algorithm better? These are some more important questions that need to be answered.

Finally, one may use the GWW approach to study the search spaces of other problems. A good candidate would be the search space of assignments of random SAT formulas. Characterization of this space would perhaps explain the apparent success of WALKSAT[20] in discovering good satisfying assignments.

Acknowledgements

The author wishes to thank the reviewers for their many useful comments.

References

- [1] D. Aldous and U. Vazirani. Go with the winners Algorithms. In *Proc. 35th FOCS*, pages 492–501, 1994.
- [2] N. Alon, M. Krivelevich, B. Sudakov. Finding a large hidden clique in random graph. In *Proc. 9th SODA*, 1998.
- [3] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy. Proof verification and intractability of approximation problems. In *Proc. 33rd FOCS*, pages 14–23, 1992.
- [4] Probabilistic checking of proofs; a new characterization of NP. In *Proc. 33rd FOCS*, pages 2–13, 1992.
- [5] B. Bollobás. *Random Graphs*. Academic Press, 1985.
- [6] R. Boppana and M. M. Halldórsson. Approximating maximum independent sets by excluding subgraphs, *BIT*, 32, pages 180–196, 1992.
- [7] T. Carson and R. Impagliazzo. Determining regions of related solutions for graph bisection problems. In *Inten. Joint Conference on AI, Workshop ML-1: Machine Learning for Large Scale Optimization*, 1999.
- [8] T. Dimitriou and R. Impagliazzo. Towards an analysis of local optimization algorithms. In *Proc. 28th STOC*, 1996.

- [9] T. Dimitriou and R. Impagliazzo. Go with the Winners for Graph Bisection. In *Proc. 9th SODA*, 510–520, 1998.
- [10] T. Dimitriou. Finding Hidden Cliques in Random Graphs. *Preliminary work*, 2001.
- [11] William Feller. *An Introduction to Probability Theory and its Applications*. Vol. 1, 3rd Edition, 1968, John Wiley & Sons.
- [12] M. R. Garey and D. S. Johnson. *Computers and Intractability*. Freeman, San Francisco, CA, 1979.
- [13] M. R. Jerrum. Large cliques elude the Metropolis process. In *Random Structures and Algorithms*, 3(4), 347–359, 1992.
- [14] D. S. Johnson, C. R. Aragon, L. A. McGeoch and C. Schevon. Optimization by Simulated Annealing: An experimental evaluation, Part I, in *Oper. Research*, 37 (1989), pp. 865–892.
- [15] D. S. Johnson and M. Trick, editors. Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge. American Mathematical Society, 1996.
- [16] A. Juels and M. Peinado. Hiding cliques for cryptographic security. In *Proc. 9th SODA*, 678–684, 1998.
- [17] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Communications*, pages 85–103, 1972.
- [18] R. M. Karp. Probabilistic analysis of some combinatorial algorithms. J. F. Traub, editor, *Algorithms and Complexity: New Directions and Recent results*. Academic Press, 1976.
- [19] L. Kučera. Expected complexity of graph partitioning problems. *Discrete Applied Math.* 57, pages 193–212, 1995.
- [20] B. Selman, H. A. Kautz and B. Cohen. Local search strategies for satisfiability testing. In *Second DIMACS Challenge on Cliques, Coloring and Satisfiability*, October 1993.