A

An Empirical Study of Two Vertex Selection Strategies for the Clique Decision and Maximisation Problems

PATRICK PROSSER, University of Glasgow

Given a graph $G = (V, E)$ a clique is set of vertices $C \subseteq V$ such that all pairs of vertices in C are adjacent. We can construct a clique using an exact algorithm that has a guessing stage, i.e. a stage where it must choose a vertex and add it to C. Intuition suggests that we choose the vertex adjacent to most others. Our study shows that intuition is incorrect and offers an explanation of why that is so. We perform an empirical study of the clique decision problem and demonstrate that it has a phase transition with a corresponding complexity peak. We then characterise this with respect to the constrainedness of the decision problem and show that we can derive a theory-based strategy for vertex selection and predict its behaviour. We show that the vertex selection strategy also works well on the optimisation problem i.e. finding the maximum clique.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures*; G.2.1 [**Discrete Mathematics**]: Combinatorics—*Combinatorial algorithms*

General Terms: Algorithms, Heuristics, Experimentation

Additional Key Words and Phrases: Maximum clique, clique decision problem, vertex selection strategy, heuristics, empirical study, phase transition, constrainedness

ACM Reference Format:
ACM J. Exp. Algor. V, N, Article A (January YYYY), 14 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

A simple undirected graph G is a pair (V, E) where V is a set of vertices and E a set of edges. An edge $\{u, v\}$ is in E if and only if $\{u, v\} \subseteq V$ and vertex u is adjacent to vertex v. A *clique* is a set of vertices $C \subseteq V$ such that every pair of vertices in C is adjacent in G. Clique is one of the six basic NP-complete problems given in [Garey and Johnson 1979]. It is posed as a decision problem [GT19]: given a simple undirected graph $G = (V, E)$ and a positive integer $k \leq |V|$, does G contain a clique of size k or more? The optimisation problem is then to find a *maximum clique*, where $\omega(G)$ is the size of a maximum clique.

We can address the decision and optimisation problems with an exact algorithm, such as a backtracking search [Pardalos and Rodgers 1992; Fahle 2002; Régin 2003; Wood 1997; Carraghan and Pardalos 1990; Segundo et al. 2011; Konc and Janezic 2007; Tomita et al. 2010]. Backtracking search incrementally constructs the set C by choosing a vertex from the *candidate set* P (where P is initially V) and adding that vertex to C. The candidate set is then updated, removing vertices that cannot participate in the evolving clique.

Author's address: School of Computing Science, University of Glasgow, Glasgow G12 8QQ, Scotland; email: Patrick.Prosser@glasgow.ac.uk.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

In the clique decision problem we should expect that a phase transition will occur when we vary k. For a graph of a given edge density, when k is small it should be easy to find a clique of size k or more, and when k is large it should be easy to determine that there is no clique of that size, and at some critical value of k it should be hard to decide if there is a clique of that size. Consequently, we investigate the phase transition in the decision problem and use theory to predict when that phase transition will occur. We then use that theory to predict and explain the behaviour of two vertex selection strategies for the decision and optimisation problem. The vertex selection strategies are essentially heuristics [Pólya 1945], i.e. *rules of thumb*, and are similar to variable ordering heuristics used in constraint programming [Rossi et al. 2006; Gent et al. 1996].

In the next section we present an exact algorithm for clique decision and two vertex selection heuristics based on vertex degree. This leads us to a characterisation of the clique decision problem's phase transition (section 4) and we use this to explain the behaviour of the heuristics. We then demonstrate that this behaviour carries over into the optimisation problem. Finally we look inside the search process, review related work and then conclude.

2. ALGORITHM FOR CLIQUE DECISION AND ITS HEURISTICS

We introduce the notation used throughout and present the algorithm and vertex selection heuristics used in the study of the clique decision problem.

2.1. Notation

Let $V(G)$ be the set of vertices and $E(G)$ the set of (undirected) edges in the graph $G = (V, E)$. Given a set of vertices P, the induced subgraph G_P is the pair $(P, E(G) \cap (P \times P))$. The neighbourhood of a vertex v is $N(v, G)$, where $N(v, G) = \{w : \{v, w\} \in E(G)\}$. The degree of vertex v in graph G_P is $\delta_{v,P}$ and is the number of vertices adjacent to v in G_P, i.e. $\delta_{v,P} = |N(v, G_P)|$.

2.2. An Algorithm for Clique Decision

Algorithm 1 is based on [Fahle 2002] and [Pardalos and Rodgers 1992]. The boolean function CD (lines 1 to 5) takes as arguments a graph G, integer k (the size of clique to be found) and a function $select$ used to choose a vertex to add to the growing clique C. In line 3, the candidate set P is initialised to be all the vertices in G and the growing clique C is initially empty (line 4). CD then calls the boolean $expand$ function (line 5).

Function $expand$ (lines 6 to 18) delivers true if the clique C is of size k, i.e. $found$ will be assigned $true$ in line 8, the while loop at line 9 immediately fails and at line 18 $found$ is returned. Otherwise $expand$ iterates while a clique of size k has not been found and vertices can be selected from candidate set P (lines 8 to 17). A vertex v is selected from P using the function $select$ (line 10), where $select(P, G)$ chooses and delivers a vertex from the induced subgraph G_P.

The selected vertex is added to the clique (line 11) and a new candidate set P' is create where P' is the set of vertices in P that are in the neighbourhood of v (line 12). In line 13 low degree vertices are removed from P', i.e. if a vertex $w \in P'$ with degree $\delta_{w,P'}$ is added to the clique it will increase C in size by at most $\delta_{w,P'} + 1$ and if this is insufficient to reach k we can remove w. This corresponds to Rule 7 in [Pardalos and Rodgers 1992] and Lemma 1 in [Fahle 2002]. In line 14 $G_{P'}$ is greedily coloured using Brelaz's heuristic [Brélaz 1979], where $colour(P', G)$ delivers as a result ω_* the number of colours used and this is an upper-bound on the size of the clique in $G_{P'}$. In line 15 a recursive call is made to $expand$ if the upper bound ω_* plus the size of the current clique is sufficient to reach k (i.e. it is assumed that if the first part of the conjunction

is false the second part of the conjunction will not be executed). The selected vertex is then removed from C and P (lines 16 and 17).

Algorithm 1: Clique Decision algorithm CD

```
1  boolean CD(Graph G, integer k, function select)
2  begin
3      Set P ← V(G)
4      Set C ← ∅
5      return expand(C, P, G, k, select)

6  boolean expand(Set C, Set P, Graph G, integer k, function select)
7  begin
8      boolean found ← |C| = k
9      while ¬found ∧ P ≠ ∅ do
10         integer v ← select(P, G)
11         C ← C ∪ {v}
12         Set P' ← P ∩ N(v, G)
13         P' ← P' \ {w : w ∈ P' ∧ δ_{w,P'} + |C| + 1 < k}
14         integer ω* ← colour(P', G)
15         found ← ω* + |C| ≥ k ∧ expand(C, P', G, k, select)
16         C ← C \ {v}
17         P ← P \ {v}
18     return found
```

2.3. Heuristics

We consider two heuristics that can be used in the guessing stage of $expand$ (line 10). The first is the most intuitive, $maxDeg(P, G)$, selecting the vertex of maximum degree in the graph G_P, (also known as best-in [Pardalos and Xue 1994]). The second, and less intuitive, is $minDeg(P, G)$ that selects the vertex of minimum degree in G_P, (i.e. worst-in [Pardalos and Xue 1994]). In Algorithm 1 the degree of vertices are maintained dynamically, consequently our heuristics are *new* (i.e. *dynamic*). A call to $CD(G, k, maxDeg)$ will be named CD_{max} and $CD(G, k, minDeg)$ will be named CD_{min}.

3. THE CLIQUE DECISION PROBLEM

Experiments were performed on Erdós-Rënyi random graphs $G(n, p)$ where n is the number of vertices and each edge is included in the graph with probability p independent from every other edge. Given a random graph $G(n, p)$ we can then have n decision problems, asking if there is a clique in the graph of size k, $1 \leq k \leq n$. The algorithms were coded in java 1.6.0 using the constraint programming toolkit choco version 2.1.0 [JCh]. The experiments were run on a machine with two Intel E5620 2.4GHz quad-core processors with 48 GB of memory, running linux centos 5.3.

One hundred instances of $G(100, 0.9)$ were generated and CD_{max} was applied to each graph for each value of k in the range 10 to 40, and this is shown graphically in Figure 1. Two contours are given: run time in milliseconds against clique size k (left y-axis) and percentage satisfiability against k (right y-axis) i.e. the percentage of graphs in the sample of $G(100, p)$ that have a clique of size k or more. We observe a phase transition in satisfiability with a corresponding computational complexity peak. At small values of k ($1 \leq k \leq 28$) it is easy to find a clique of the required size and when k is large

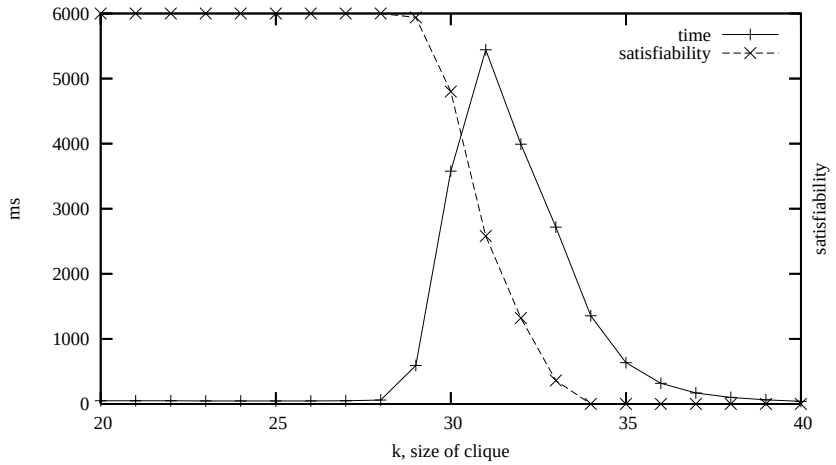

Fig. 1. $G(100, 0.9)$, sample size 100, $20 \leq k \leq 40$, CD_{max}, average run time and satisfiability.

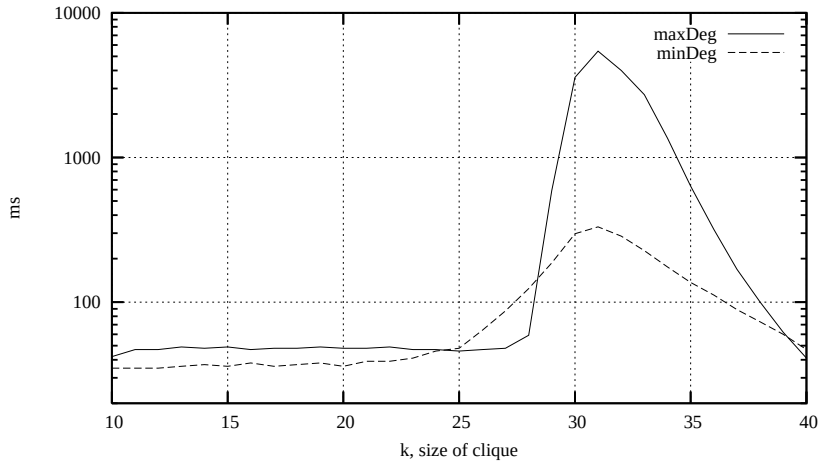

Fig. 2. $G(100, 0.9)$, sample size 100, $20 \leq k \leq 40$, CD_{max} and CD_{min}, log of average run time.

($37 \leq k \leq 40$) it is easy to determine that there is no clique of that size. However, in the range $29 \leq k \leq 36$ hard instances occur, with a complexity peak close to the 50% cross over point where on average half of the instances are satisfiable, and this is expected [Cheeseman et al. 1991; Gent et al. 1996].

The experiment above was repeated using CD_{min}. Figure 2 compares CD_{max} and CD_{min}. Again, we see a common complexity peak round about $k = 31$. But there is a dramatic difference in performance between the two vertex selection heuristics. In the hard region, $29 \leq k \leq 35$ CD_{min} outperforms CD_{max} typically by two orders of magnitude. That is, when problems are hard the best selection heuristic is to choose a vertex of minimum degree. We will now offer an explanation of why CD_{min} dominates CD_{max}, but first we must define the constrainedness of the clique decision problem.

4. THE CONSTRAINEDNESS OF CLIQUE DECISION

We now define constrainedness κ (kappa) for the clique decision problem and demonstrate that it captures the behaviour seen in Figures 1 and 2. We then use κ to explain the behaviour of the vertex selection heuristics.

4.1. Constrainedness (κ)

The phase transition has been characterised in [Gent et al. 1996] as follows

$$\kappa = 1 - \frac{\log\langle Sol\rangle}{\log|S|}$$

where $\langle Sol\rangle$ is the expected number of solutions and $|S|$ is the size of the state space, i.e. the maximum number of states that could be considered by a simple generate-and-test algorithm. When all states are solutions $\kappa = 0$ and problems are satisfiable and easy, when no state is a solution $\kappa = \infty$ and problems are unsatisfiable and easy, and when there is on average a single solution $\kappa = 1$ and problems are hard. We now define κ for the clique decision problem in $G(n,p)$ for clique size k. The size of the state space $|S|$ is equal to the number of ways we can select k vertices, hence

$$|S| = \binom{n}{k}$$

To compute the expected number of solutions $\langle Sol\rangle$ we first calculate the probability p_{sol} that a state is a solution, i.e. that having chosen k vertices they are all adjacent

$$p_{sol} = p^{\binom{k}{2}}$$

Since the probability holds for any k vertices the expected number of solutions is given by

$$\langle Sol\rangle = |S|.p_{sol}$$

Combining these results we have that for the clique decision problem in random graphs $G(n,p)$ with clique size k

$$\kappa_{clique} = \frac{\binom{k}{2}\log(\frac{1}{p})}{\log(\binom{n}{k})} \tag{1}$$

In Figure 3 we present the same data as in Figure 1 but with constrainedness (κ) on the x-axis, demonstrating that κ is a good measure for this problem.

MC_{min} was applied to $G(n,0.8)$ instances with $n \in \{100,110,120,130,140,150\}$, $10 \leq k \leq 50$ and a sample size of 100. Plotted in Figure 4 is on the left logarithm of average run time in milliseconds and on the right percentage satisfiability, both with κ on the x-axis. Ideally we would like the 50% crossover point to occur at $\kappa \approx 1$ with the complexity peak occuring simultaneously. In fact we see the crossover point and complexity peak occurring in the range $0.82 \leq \kappa \leq 0.90$, i.e. earlier than anticipated but as expected when problem size is small [Gent et al. 1996]. The evidence of Figures 3 and 4 suggest that κ characterises the phase transition phenomena in the clique decision problem for random $G(n,p)$.

4.2. Constrainedness (κ) as a Heuristic

In [Gent et al. 1996] it is proposed that κ be used as a guiding principle when designing variable ordering heuristics, i.e. to make decisions that minimise the constrainedness of the future sub-problem. In particular κ suggests how we should select a vertex to

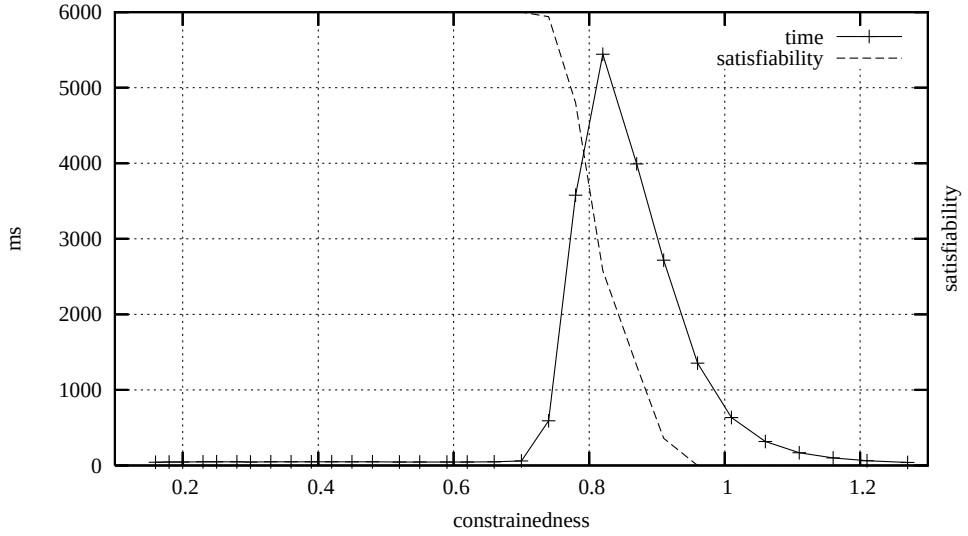

Fig. 3. $G(100, 0.9)$, sample size 100, $20 \leq k \leq 40$, CD_{max}, average run time in milliseconds (ms) and satisfiability against constrainedness (κ).

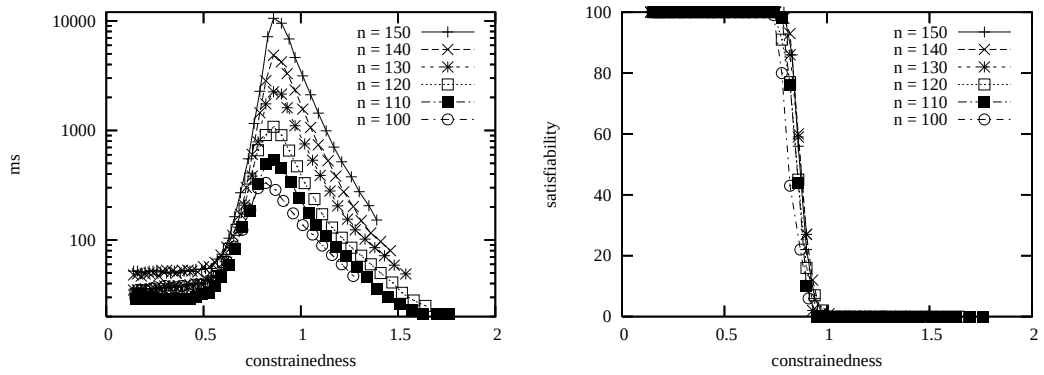

Fig. 4. $G(n, 0.9)$, sample size 100, $n \in \{100, 110, 120, 130, 140, 150\}$, $1 \leq k \leq n$, CD_{max}. On the left, logarithm of run time against κ and on the right satisfiability against κ.

add to the clique. In the setting of $G(n, p)$ with clique size k when a vertex is selected κ changes as follows:

$$\frac{\binom{k}{2}\log(\frac{1}{p})}{\log(\binom{n}{k})} \implies \frac{\binom{k}{2}\log(\frac{1}{p'})}{\log(\binom{n-1}{k})} \tag{2}$$

In (2) one vertex is selected so n becomes $n-1$, all the edges emanating from that vertex are removed and edge probability p becomes p', where p' is the number of edges remaining divided by $\binom{n-1}{2}$. Therefore the only differences between vertex selections is the resulting values of p'. If p' decreases then $\log(\frac{1}{p'})$ increases as does κ, and if p' increases then $\log(\frac{1}{p'})$ decreases and so does κ. Consequently we should choose the vertex that makes p' as large as possible and we do this by choosing the vertex that

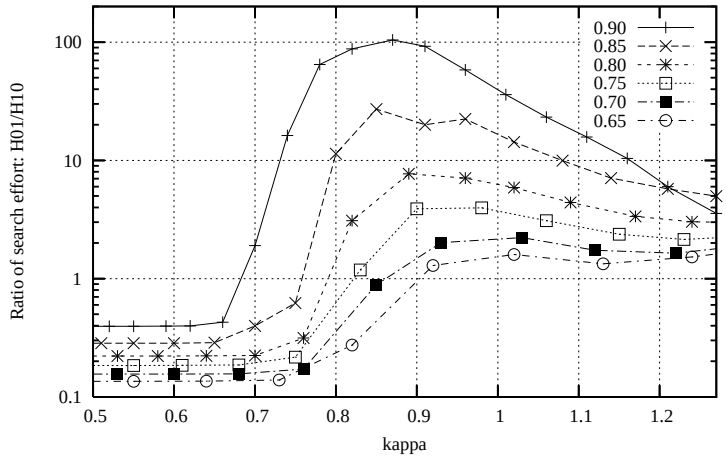

Fig. 5. Is difference in heuristics related to degree of the graph? For $G(100, p)$ with p and k varying, we apply CD_{max} and CD_{min} and compute the ratio of search effort, $r = s_{max}/s_{min}$. Plotted is average ratio of search effort against κ, one contour for each value of p. In the hard region, as graphs get denser the difference between heuristic performance increases.

removes the least number of edges and that is the vertex of minimum degree. Our experiments support this (Figure 2), i.e. our best vertex selection heuristic is $minDeg$.

For the function $f(x) = \log(x)$, when x is small, small changes to x make large changes to $f(x)$ and when x is large small changes to x make small changes to $f(x)$. Consequently we should expect that when p is large (and $\frac{1}{p}$ is small) any change of p to p' will cause large differences in $\log(\frac{1}{p'})$ and κ. Therefore in dense graphs (high values of p) we should expect large differences in performance between vertex selection heuristics, and as graphs get sparser (low values of p) differences in vertex selection heuristics should be small.

Experiments were performed on $G(100, p)$ with $0.65 \leq p \leq 0.90$ with $1 \leq k \leq 40$, sample size 100, using CD_{max} and CD_{min}. For each point we captured the ratio $r = s_{max}/s_{min}$ where s_{max} is the number of calls to $expand$ made by CD_{max} and s_{min} the number of calls to $expand$ made by CD_{min}. If $r > 1$ then heuristic CD_{max} was best and if $r < 1$ CD_{min} was best. This ratio was averaged over the 100 random graphs for each call to CD as k was varied. This is shown graphically in Figure 5, where the logarithm of the average ratio r is plotted against κ with a contour for each graph density. What we see is that as we enter the hard zone and as graphs get denser the difference between the heuristics becomes more significant: dense graphs show a large difference (up to two orders of magnitude), sparse graphs a small difference, and this is what theory predicts.

5. OPTIMISATION

The optimisation problem can be considered as a sequence of decision problems. On finding a solution search attempts to find a new solution better than the last. If search finds a clique of size k and then attempts to find a larger clique and fails, k is optimal. Therefore the search process travels through the phase transition, from the soluble to the insoluble region. Based on what we have seen we should expect that κ-based heuristics will work well on the optimisation problem. To study the optimisation problem we first investigate randomly generated problems, then the DIMACS benchmarks

[DIM] and finally the real-world instances of [Eppstein and Strash 2011] using function MC presented in Algorithm 2.

Algorithm 2: Maximum Clique algorithm MC

```
1  Set MC(Graph G, function select)
2  begin
3      Global Set C_max ← ∅
4      Set P ← V(G)
5      Set C ← ∅
6      expand(C, P, G, select)
7      return C_max

8  void expand(Set C, Set P, Graph G, function select)
9  begin
10     if |C| > |C_max| then C_max ← C
11     while P ≠ ∅ do
12         integer v ← select(P, G)
13         C ← C ∪ {v}
14         Set P' ← P ∩ N(v, G)
15         P' ← P' \ {w : w ∈ P' ∧ δ_{w,P'} + |C| + 1 ≤ |C_max|}
16         integer ω_* ← colour(P', G)
17         if ω_* + |C| > |C_max| then expand(C, P', G, select)
18         C ← C \ {v}
19         P ← P \ {v}
```

5.1. Random Problems

As before, a call to $MC(G, maxDeg)$ is named MC_{max} and $MC(G, minDeg)$ named as MC_{min}. MC_{max} and MC_{min} were used to find the maximum clique in $G(100, p)$ with $0.01 \leq p \leq 0.99$ varying in steps of 0.01, with a sample size of 100 at each value. Figure 6 shows on the left two contours, for MC_{max} and MC_{min}, with run times in milliseconds on a logarithmic scale. On the right we have size of maximum clique found with contours for minimum, median, and maximum. We show only the range $0.4 \leq p \leq 0.99$: when $p \leq 0.4$ there is essentially no search as the problems are trivial. We see that when p is very high problems are easy and this agrees with the tabulated results of [Segundo et al. 2011]. When p is low (less than 0.4) problems are trivial. However, when p is greater than 0.65 the choice of heuristic is critical. Again we see that MC_{min} outperforms MC_{max} in the really hard region and is orders of magnitude better. However, in the range $0.4 \leq p \leq 0.58$ MC_{max} becomes the heuristic of choice, but the gains to be had are modest (savings of tenths of seconds). In summary, when problems are hard MC_{min} is best.

5.2. DIMACS Instances

We now report on the DIMACS instances. Experiments were carried out to determine the relative performance of the heuristics over these standard benchmarks. The goal was to find the maximum clique, and if cpu time was exhausted, report the size of the largest clique found so far.

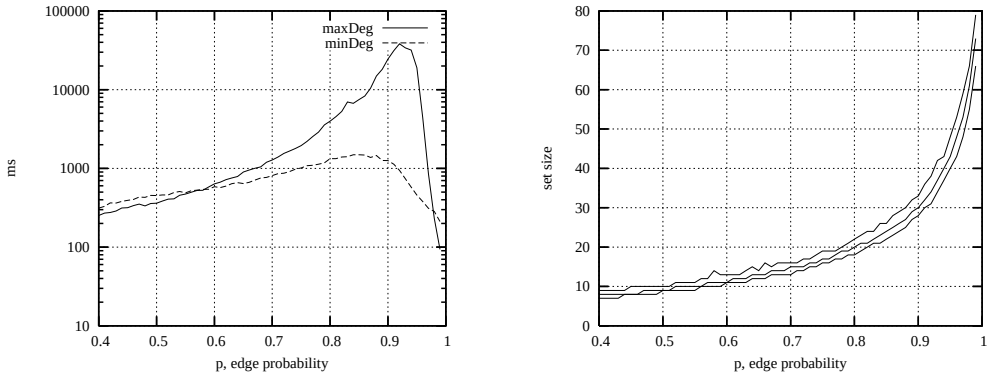

Fig. 6. $MC_{max} and MC_{min}$ applied to one hundred instances of $G(100, p)$ with $0.4 \leq p \leq 0.99$. On the left is logarithm of average search effort for MC_{max} and MC_{min}. Shown on the right is the maximum, median and minimum maximum-clique size.

The experiments were performed under similar conditions to those outlined in Section 3. Table I shows results for 59 of the 66 DIMACS maximum clique problems[1]. Run times are given in seconds and in brackets the size of the clique found. Run time was limited to 4 hours (14,400 seconds), an entry of "—" was aborted after that time and the clique size is then the largest found in that time. In the table bold entries corresponds to the best solution to non-trivial instances, and in the event that the maximum was not found we highlight the greatest clique found.

In Table I MC_{min} is most often the best. In eight instances (brock800-2/3/4, hamming-10-4, MANN-a81, p-hat100-3, p-hat1500-2/3,) MC_{max} was best. These are hard instances that exhausted the run time and a greedy construction that selects the vertex of maximum degree worked well. There was only one instance, hamming10-2, where MC_{max} significantly outperformed MC_{min} in run time.

5.3. Real-world Instances

Experiments were performed on a subset of the real-world graphs in [Eppstein and Strash 2011], the subset limited by the size of graph that could be represented in our programs (using adjacency matrix representation of the graph). Four classes of problems were investigated: BioGRID data, Mark Newman's social network data, Pajek's social and bibliographic networks, and Stanford Large Network instances (SNAP). These are shown in Table II. The first four columns describe the instance where n is the number of vertices, m the number of edges and ω the clique number. There are three columns for each heuristic: the time to find a largest clique and prove optimality (in milliseconds), the number of times a test was performed to determine if two vertices were adjacent (the most frequent activity when colouring vertices and building candidate sets) and the proportion (prop) of time that was spent in search finding the optimal solution (the remainder being spent proving optimality). There are three things to note. First, the instances are sparse graphs and we should expect the differences in performance to be relatively small. Second, we notice that the proportion of time spent finding the optimal solution (column prop) is small for MC_{max} but large for MC_{min}, and finally that MC_{min} always performed less adjacency checks than MC_{max} (column check) and is usually faster (column ms). Yet again, it appears that choosing the vertex of minimum degree is the best strategy.

[1]The c-fat instances are omitted because they are trivial.

Table I. 59 of the 66 DIMACS Instances

instance	MC_{max}		MC_{min}	
brock200-1	156.9	(21)	**28.7**	(21)
brock200-2	1.2	(12)	1.1	(12)
brock200-3	6.2	(15)	**2.3**	(15)
brock200-4	12.6	(17)	**7.3**	(17)
brock400-1	—	(24)	—	(**27**)
brock400-2	—	(24)	—	(**29**)
brock400-3	—	(31)	**14,250.4**	(31)
brock400-4	—	(25)	**8,715.8**	(33)
brock800-1	—	(20)	—	(**21**)
brock800-2	—	(**21**)	—	(20)
brock800-3	—	(**21**)	—	(20)
brock800-4	—	(**21**)	—	(20)
hamming10-2	**49.1**	(512)	361.9	(512)
hamming10-4	—	(**40**)	—	(38)
hamming6-2	0.0	(32)	0.1	(32)
hamming6-4	0.1	(4)	0.1	(4)
hamming8-2	**0.2**	(128)	1.7	(128)
hamming8-4	2.1	(16)	1.6	(16)
johnson16-2-4	47.6	(8)	**43.1**	(8)
johnson32-2-4	—	(16)	—	(16)
johnson8-2-4	0.0	(4)	0.0	(4)
johnson8-4-4	0.0	(14)	0.0	(14)
keller4	5.9	(11)	**2.1**	(11)
keller5	—	(27)	—	(27)
keller6	—	(52)	—	(**53**)
MANN-a27	—	(126)	**125.6**	(126)
MANN-a45	—	(342)	—	(**345**)
MANN-a81	—	(**1,098**)	—	(999)
MANN-a9	0.3	(16)	0.0	(16)
p-hat1000-1	96.3	(10)	**49.2**	(10)
p-hat1000-2	—	(45)	—	(**46**)
p-hat1000-3	—	(**64**)	—	(54)
p-hat1500-1	854.9	(12)	**394.5**	(12)
p-hat1500-2	—	(**63**)	—	(54)
p-hat1500-3	—	(**90**)	—	(61)
p-hat300-1	0.7	(8)	1.0	(8)
p-hat300-2	21.8	(25)	**3.1**	(25)
p-hat300-3	—	(36)	**125.7**	(36)
p-hat500-1	4.4	(9)	4.4	(9)
p-hat500-2	1,974.1	(36)	**58.5**	(36)
p-hat500-3	—	(48)	**8,394.2**	(50)
p-hat700-1	14.8	(11)	13.1	(11)
p-hat700-2	—	(44)	**433.4**	(44)
p-hat700-3	—	(62)	—	(57)
san1000	917.7	(15)	**146.5**	(15)
san200-0.7-1	0.6	(30)	0.5	(30)
san200-0.7-2	5.2	(18)	**0.8**	(18)
san200-0.9-1	82.8	(70)	**1.3**	(70)
san200-0.9-2	5,926.6	(60)	**13.7**	(60)
san200-0.9-3	—	(39)	**40.0**	(44)
san400-0.5-1	3.3	(13)	2.5	(13)
san400-0.7-1	337.1	(40)	**17.0**	(40)
san400-0.7-2	133.7	(30)	**21.5**	(30)
san400-0.7-3	934.7	(22)	**139.8**	(22)
san400-0.9-1	4,241.2	(100)	**863.6**	(100)
sanr200-0.7	40.2	(18)	**12.8**	(18)
sanr200-0.9	—	(42)	**957.9**	(42)
sanr400-0.5	55.3	(13)	**42.5**	(13)
sanr400-0.7	—	(21)	**5,208.6**	(21)

Table II. Real-world Instances

				MC_{max}			MC_{min}		
	n	m	ω	ms	checks	prop	ms	checks	prop
BioGRID									
fission-yeast	2,031	12,637	12	158	2,634,571	0.1107	120	2,169,153	0.9722
fruitfly	7,282	24,894	7	489	26,809,042	0.0022	433	26,552,801	0.9518
human	9,527	31,182	13	548	45,834,187	0.0174	619	45,408,270	0.9918
mouse	1,455	1,636	7	97	1,075,073	0.0438	52	1,059,427	0.9978
plant	1,745	3,098	9	55	1,538,071	0.0013	66	1,525,711	0.9989
worm	35,18	6,531	7	161	6,454,207	0.0226	125	6,193,196	0.9915
yeast	6,008	156,945	33	1,406	29,541,839	0.4209	438	19,354,690	0.9130
Mark Newman									
adjnoun	112	425	5	4	9,385	0.0212	4	6,872	0.8405
astro	16,706	121,251	57	3,716	140,832,589	0.0056	2,051	139,842,886	0.9983
celegens	297	1,248	5	13	62,607	0.0074	16	46,579	0.7057
condmat	40,421	175,693	30	11,299	818,015,894	0.0730	13,574	816,971,093	0.9973
dolphins	62	159	5	1	2,332	0.1591	2	2,094	0.8276
football	115	613	9	3	7,681	0.0664	3	7,390	0.2641
internet	22,963	48,436	17	3,354	274,978,783	0.0289	4,019	263,689,825	0.9999
karate	34	78	5	1	1,028	0.3132	1	645	0.8868
lesmis	77	254	10	2	4,453	0.2547	2	3,282	0.9327
netscience	1,589	2,742	20	98	1,264,942	0.0090	68	1,263,994	0.9999
polblogs	1,490	16,715	20	164	2,734,155	0.0703	109	1,284,170	0.9355
polbooks	105	441	6	3	7,222	0.0230	3	5,944	0.6743
power	4,941	6,594	6	199	12,214,950	0.0086	186	12,209,482	0.9953
Pajek									
daysall	13,308	148,035	28	14,072	623,995,011	0.0011	1,203	90,787,216	0.9921
eatRS	23,219	304,937	9	5,439	286,035,364	0.0007	3,579	271,447,076	0.9120
foldoc	13,356	91,471	9	966	90,341,738	0.0162	1,014	89,316,802	0.6584
hepth	27,240	341,923	23	6,763	395,521,531	0.0584	5,214	372,556,670	0.9551
SNAP									
email-Enron	36,692	183,831	20	6,826	692,361,478	0.0084	9,702	673,824,916	0.9997
wiki-Vote	7,115	100,762	17	572	41,138,215	0.0277	350	26,121,434	0.9703

5.4. Inside Search

MC (Algorithm 2) was modified such that $expand$ recorded the *density* of G_P (i.e. the ratio of the number of edges in G_P over m where $m = |P| \times (|P| - 1)/2$) and $|C_{max}|$. These statistics were tagged with the depth in search and the time in search, where time in search is the number of calls to $expand$ so far.

Figure 7 shows the average density of G_P at each depth in search for MC_{max} and MC_{min} over a single instance of $G(100, 0.9)$. We see that MC_{max} leaves the candidate set with ever fewer edges until search reaches a depth of about 15 when the candidate set is small and most unpromising. On the other hand, MC_{min} forces the future subproblem to become increasingly dense and increasingly promising with a low κ value.

In Figure 8 we plot the size of the largest clique found so far against the logarithm of time in search. What we see is that MC_{max} greedily builds a large clique early on in search but has to perform a huge amount of search to improve this because the candidate set is so unpromising. MC_{min} is more cautious, growing the largest clique slowly whilst still retaining a promising candidate set (Figure 7).

6. RELATED WORK

In [Carraghan and Pardalos 1990] a branch and bound algorithm is presented for maximum clique. Vertices are ordered in non-decreasing degree order at each depth in the binomial search with a cut-off based on the size of the largest clique found so far.

[Pardalos and Rodgers 1992] present a zero-one encoding , where branch and bound search selects vertices dynamically based on current degree in the candidate set: a *non-greedy* selection chooses a vertex of lowest degree and *greedy* selects highest degree.

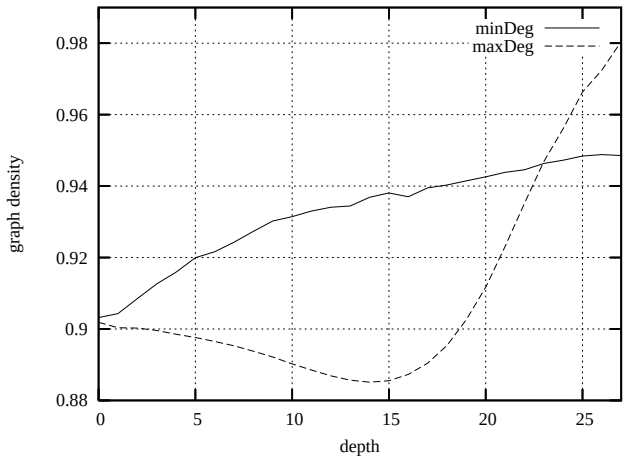

Fig. 7. $G(100, 0.9)$, graph density at depth.

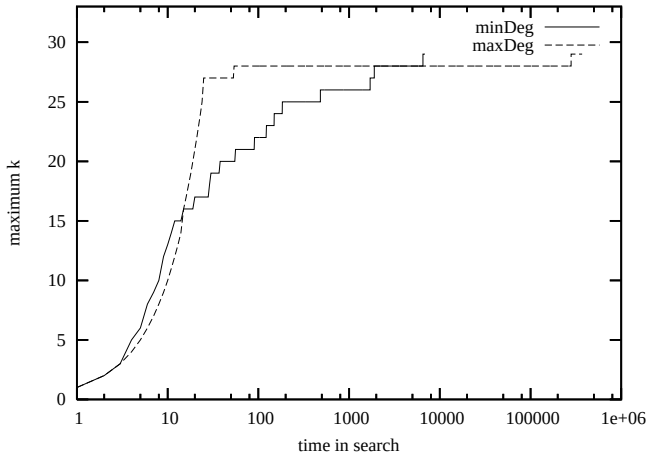

Fig. 8. $G(100, 0.9)$, size of largest clique during search.

Computational results show that greedy was good for (easy) sparse graphs and non-greedy good for (hard) dense graphs.

In [Wood 1997] graph colouring and fractional colouring is used to bound search and vertices are selected in non-increasing degree order.

Patric R. J. Östergård proposed an algorithm that has a dynamic programming flavour [Östergård 2002]. The search process starts by finding the largest clique containing vertices drawn from the set $S_n = \{v_n\}$ and records it size in $c[n]$. Search then proceeds to find the largest clique in the set $S_i = \{v_i, v_{i+1}, ..., v_n\}$ using the value in $c[i+1]$ as a bound. The vertices are ordered at the top of search in colour order, i.e. the vertices are coloured greedily and then ordered in non-decreasing colour order.

[Fahle 2002] presented a simple algorithm (Algorithm 1) (essentially the same as MC in Section 5) with a free selection of vertices. This is then enhanced (Algorithm 2) with forced accept and forced reject steps similar to Rules 4, 5 and 7 of [Pardalos and Rodgers 1992]. Fahle notes that 'The ordering in which nodes are considered during

search can have a severe impact on running time ... It is not clear though how to detect these effects while searching ...".

Jean-Charles Régin proposed a constraint programming model for the maximum clique problem [Régin 2003]. His model uses a matching in a duplicated graph to deliver a bound within search, a *Not Set* as used in the Bron Kerbosch enumeration Algorithm 457 [Bron and Kerbosch 1973] and vertex selection using the pivoting strategy similar to that in [Bron and Kerbosch 1973; Akkoyunlu 1973; Tomita et al. 2006; Eppstein and Strash 2011].

Tomita's algorithms MCQ [Tomita et al. 2003], MCR [Tomita and Kameda 2007] and MCS [Tomita et al. 2010] select vertices in non-increasing colour order, using vertex colour as a cut-off. Pablo San Segundo proposed BBMC [Segundo et al. 2011], a bit-set encoding of MCS with the colour repair step removed. An empirical study of these algorithms is reported in [Prosser 2012].

7. CONCLUSION

The purpose of this study is to better understand the behaviour of two vertex selection heuristics used in the clique decision and maximisation problems. We showed that in the decision problem there is a phase transition in satisfiability that coincides with a complexity peak, and in this region the minimum degree heuristic dominated the maximum degree heuristic, sometimes by orders of magnitude. We demonstrated that we could characterise the phase transition with respect to constrainedness, κ, and having done so use the principle of minimising κ to engineer a vertex selection heuristic for the decision problem, and that heuristic corresponded to minimum degree. With κ we predicted the behaviour of the minimum degree heuristic i.e. that performance would improve as graphs became ever more dense, and this was observed in randomly generated graphs. We also demonstrated that heuristics that are good for the decision problem are also good for the optimisation problem. This was demonstrated on random graphs, DIMACS benchmarks and real-world graphs.

Finally we looked inside the search process to see how the search process changes the structure of the candidate set at depth and how the maximum clique grows over time. This complemented the measures taken when investigating the real-world problems, i.e. MC_{max} takes a (relatively) short time to find a maximum clique but a long time to prove optimality, MC_{min} takes a long time to find a maximum clique and a short time to prove optimality, but overall MC_{min} is faster than MC_{max} on hard instances. This suggests that we might improve MC_{min} by adding a preprocessing step that greedily constructs a maximal clique. This has yet to be investigated.

REFERENCES

CHOCO Solver. http://www.emn.fr/x-info/choco-solver/.

DIMACS clique benchmark instances. ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/clique.

AKKOYUNLU, E. 1973. The enumeration of maximal cliques of large graphs. *SIAM Journal of Computing 2,* 1, 1–6.

BRÉLAZ, D. 1979. New Methods to Color the Vertices of a Graph. *Communications of the ACM 22,* 4, 251–256.

BRON, C. AND KERBOSCH, J. 1973. Algorithm 457: Finding all cliques of an undirected graph [h]. *Communications of the ACM 16,* 9, 575–579.

CARRAGHAN, R. AND PARDALOS, P. M. 1990. An exact algorithm for the maximum clique problem. *Operations Research Letters 9*, 375–382.

CHEESEMAN, P., KANEFSKY, B., AND TAYLOR, W. M. 1991. Where the really hard problems are. In *Proceedings IJCAI'91*. 331–337.

EPPSTEIN, D. AND STRASH, D. 2011. Listing all maximal cliques in large sparse real-world graphs. In *Proceedings of the 10th international conference on Experimental algorithms*. SEA'11. 364–375.

FAHLE, T. 2002. Simple and Fast: Improving a Branch-and-Bound Algorithm for Maximum Clique. In *Proceedings ESA 2002, LNCS 2461*. 485–498.

GAREY, M. AND JOHNSON, D. 1979. *Computers and Intractability*. W.H. Freeman and Co.

GENT, I. P., MACINTYRE, E., PROSSER, P., SMITH, B. M., AND WALSH, T. 1996. An Empirical Study of Dynamic Variable Ordering Heuristics for the Constraint satisfaction Problem. In *Proceedings CP 1996*. 179–193.

GENT, I. P., MACINTYRE, E., PROSSER, P., AND WALSH, T. 1996. The constrainednss of search. In *Proceedings AAAI'96*. 246–252.

KONC, J. AND JANEZIC, D. 2007. An improved branch and bound algorithm for the maximum clique problem. *Match Commun Math Comput Chem 58*, 569–590.

ÖSTERGÅRD, P. R. J. 2002. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics 120*, 197–207.

PARDALOS, P. M. AND RODGERS, G. P. 1992. A Branch and Bound Algorithm for the Maximum Clique Problem. *Computers and Operations Research 19*, 363–375.

PARDALOS, P. M. AND XUE, J. 1994. The Maximum Clique Problem. *Journal of Global Optimization 4*, 301–324.

PÓLYA, G. 1945. *How to Solve It*. Princeton University Press.

PROSSER, P. 2012. Exact algorithms for maximum clique: A computational study. *Algorithms 5,* 4, 545–587.

RÉGIN, J.-C. 2003. Using Constraint Programming to Solve the Maximum Clique Problem. In *Proceedings CP 2003, LNCS 2833*. 634–648.

ROSSI, F., VAN BEEK, P., AND WALSH, T. 2006. *Handbook of Constraint Programming*. Elsevier.

SEGUNDO, P. S., RODRÍGUEZ-LOSADA, D., AND JIMÉNEZ, A. 2011. An exact bit-parallel algorithm for the maximum clique problem. *Computers and Operations Research 38*, 571–581.

TOMITA, E. AND KAMEDA, T. 2007. An efficient branch-and-bound algorithm for finding a maximum clique and computational experiments. *Journal of Global Optimization 37*, 95–111.

TOMITA, E., SUTANI, Y., HIGASHI, T., TAKAHASHI, S., AND WAKATSUKI, M. 2003. An efficient branch-and-bound algorithm for finding a maximum clique. In *DMTC 2003, LNCS 2731*. 278–289.

TOMITA, E., SUTANI, Y., HIGASHI, T., TAKAHASHI, S., AND WAKATSUKI, M. 2010. A simple and faster branch-and-bound algorithm for finding maximum clique. In *WALCOM 2010, LNCS 5942*. 191–203.

TOMITA, E., TANAKA, A., AND TAKAHASHI, H. 2006. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science 363*, 28–42.

WOOD, D. R. 1997. An algorithm for finding a maximum clique in a graph. *Operations Research Letters 21*, 211–217.