J Heuristics (2011) 17: 181–199
DOI 10.1007/s10732-010-9131-5

Cooperating local search for the maximum clique problem

Wayne Pullan · Franco Mascia · Mauro Brunato

Received: 9 March 2009 / Revised: 18 February 2010 / Accepted: 22 March 2010 /
Published online: 2 April 2010

Abstract The advent of desktop multi-core computers has dramatically improved the usability of parallel algorithms which, in the past, have required specialised hardware. This paper introduces cooperating local search (CLS), a parallelised hyper-heuristic for the maximum clique problem. CLS utilises cooperating low level heuristics which alternate between sequences of iterative improvement, during which suitable vertices are added to the current clique, and plateau search, where vertices of the current clique are swapped with vertices not in the current clique. These low level heuristics differ primarily in their vertex selection techniques and their approach to dealing with plateaus. To improve the performance of CLS, guidance information is passed between low level heuristics directing them to particular areas of the search domain. In addition, CLS dynamically reconfigures the allocation of low level heuristics to cores, based on information obtained during a trial, to ensure that the mix of low level heuristics is appropriate for the instance being optimised. CLS has no problem instance dependent parameters, improves the state-of-the-art performance for the maximum clique problem over all the BHOSLIB benchmark instances and attains unprecedented consistency over the state-of-the-art on the DIMACS benchmark instances.

Keywords Local search · Maximum clique · Parallel algorithm

W. Pullan (✉)
School of Information and Communication Technology, Griffith University, Gold Coast, Australia
e-mail: w.pullan@griffith.edu.au

F. Mascia · M. Brunato
Dipartimento di Ingegneria e Scienza dell'Informazione, Università di Trento, Trento, Italy

F. Mascia
e-mail: mascia@disi.unitn.it

M. Brunato
e-mail: brunato@disi.unitn.it

1 Introduction

The maximum clique problem calls for finding the maximum sized sub-graph of pairwise adjacent vertices in a given graph. Maximum clique is a prominent combinatorial optimisation problem with many applications, for example, information retrieval, experimental design, signal transmission and computer vision (Balus and Yu 1986). More recently, applications in bioinformatics have become important (Pevzner and Sze 2000; Ji et al. 2004). The search variant of maximum clique can be stated as follows: Given an undirected graph $G(V, E)$, where V is the set of all vertices and E the set of all edges, find a maximum size clique in G, where a clique in G is a subset of vertices, $K \subseteq V$, such that all pairs of vertices in K are connected by an edge, *i.e.*, for all $v, v' \in K$, $\{v, v'\} \in E$, and the size of a clique K is the number of vertices in K. Maximum clique is NP-hard and the associated decision problem is NP-complete (Garey and Johnson 1979); furthermore, it is inapproximable in the sense that no deterministic polynomial-time algorithm can find cliques of size $|V|^{1-\epsilon}$ for any $\epsilon > 0$, unless NP $=$ ZPP (Håstad 1999).[1] The best polynomial-time approximation algorithm for maximum clique achieves an approximation ratio of $O(|V|/(\log|V|)^2)$ (Boppana and Halldórsson 1992). Therefore, large and hard instances of maximum clique are typically solved using heuristic approaches, in particular, greedy construction heuristics and stochastic local search heuristics such as simulated annealing, genetic algorithms and tabu search. For an overview of these and other methods for solving maximum clique problems, see Bomze et al. (1999). It may be noted that the maximum clique problem is equivalent to the independent set problem as well as to the minimum vertex cover problem, and any heuristic for maximum clique can be directly applied to these equally fundamental and application relevant problems (Bomze et al. 1999).

From the recent literature on maximum clique heuristics, it seems that, somewhat unsurprisingly, there is no single best heuristic. Although most heuristics have been empirically evaluated on benchmark instances from the Second DIMACS Challenge (Johnson and Trick 1996), it is quite difficult to compare experimental results between studies, mostly because of differences in the respective experimental protocols and run-time environments. Nevertheless, particularly considering the comparative results reported by Pullan and Hoos (2006), it seems that there are five stochastic local search maximum clique heuristics that achieve state-of-the-art performance: *Reactive Local Search* (RLS) (Battiti and Protasi 2001), an advanced and general tabu search method that automatically adapts the tabu tenure parameter; *Deep Adaptive Greedy Search* (DAGS) (Grosso et al. 2004) which uses an iterated greedy construction procedure with vertex weights; *k-opt* heuristic (Katayama et al. 2004) is based on a conceptually simple Variable Depth Search (VDS) procedure that uses elementary search steps in which a vertex is added to, or removed from, the current clique; *VNS* (Hansen et al. 2004) is a basic variable neighbourhood search heuristic that combines greedy search with simplical vertex tests in its descent steps; and *Dynamic Local Search–Maximum Clique* (DLS-MC) (Pullan and Hoos 2006), a heuristic which alternates between phases of iterative improvement, during which suitable

[1] ZPP is the class of problems that can be solved in expected polynomial time by a probabilistic algorithm with zero error probability.

vertices are added to the current clique, and plateau search, during which vertices of the current clique are swapped with vertices not contained in the current clique. The selection of vertices is solely based on vertex penalties that are dynamically adjusted during the search, and a perturbation mechanism is used to overcome search stagnation. The behaviour of DLS-MC is controlled by a single parameter, penalty delay, which controls the frequency at which vertex penalties are reduced. Unfortunately, the DLS-MC heuristic is sensitive to the penalty delay parameter, and its calibration time-consuming and instance-dependent. This weakness of DLS-MC has been explicitly stated (Pullan and Hoos 2006), where elimination of tuning for penalty delay is indicated as a valuable improvement. This deficiency in DLS-MC was subsequently rectified in *Phased Local Search* (PLS) (Pullan 2006) which sequentially cycles through greedy vertex degree selection, random selection and vertex penalty based selection heuristics where the penalty delay parameter was adaptively modified to an appropriate value, thus eliminating the need for any run-time parameters. The performance of PLS is comparable to that of DLS-MC and both are currently state-of-the-art local search heuristics for the maximum clique problem.

In this work, a new, parallel, hyper-heuristic (Burke et al. 2003) for maximum clique, dubbed *Cooperating Local Search* (CLS) is introduced. Hyper-heuristics are defined as heuristic search methods that seek to automate the process of selecting, combining, generating or adapting a number of low level heuristics to efficiently solve computational search problems and are normally classified into one of two groupings. The first grouping, of which CLS is a member, are those hyper-heuristics that are methodologies to select low level heuristics while the other grouping is those hyper-heuristics that are methodologies to generate low level heuristics. A recent review of hyper-heuristics is presented in Chakhlevitch and Cowling (2008).

CLS is targeted at the recently available, and rapidly becoming desk-top standard, multi-core computers. The relatively recent advent of these computers as the standard desktop (and laptop) computer has strengthened the case for the development of parallel heuristics as now, unlike in the past, a specialised computer or cluster of computers is no longer required for parallel computing. CLS controls copies of four low level heuristics which are allocated to available processor cores with cooperation between low level heuristics materially benefiting the overall performance of CLS. The low level heuristics differ primarily in their vertex selection methods and also the perturbation mechanisms used to overcome search stagnation. CLS requires no run-time parameters and is able to dynamically reconfigure the use of low level heuristics to better match the maximum clique instance being optimised.

The remainder of this paper is structured as follows. In Sect. 2, an overview of the performance of two current maximum clique heuristics is presented and conclusions drawn as to the design of an effective parallel maximum clique hyper-heuristic. This is followed, in Sect. 3, by a description of CLS, an implementation of such a hyper-heuristic. In Sect. 4 the CLS performance is assessed by means of an empirical study, confirming the intuitions of Sect. 2 while, in Sect. 5, the correlation between the performances of the low level heuristics and some properties of selected instances and the benefit that arise from the cooperation between the low level heuristics is assessed. Finally, a summary of the main contributions of this work, insights gained from this study, and an outline of some directions for future research are described in Sect. 6.

2 Background to CLS

The so-called 'No Free Lunch' theorem (Wolpert and Macready 1997), generally stated as '...for any algorithm, any elevated performance over one class of problems is exactly paid for in performance over another class.' Wolpert and Macready (1997), is substantiated by the results presented in Pullan and Hoos (2006), Battiti and Protasi (2001), Pullan (2006) and Grosso et al. (2008) which show that, to efficiently solve all the DIMACS benchmark instances, heuristics with conflicting characteristics are required. Specific examples of this are RLS (Battiti and Protasi 2001) and PLS (Pullan 2006), where both heuristics perform well on the majority of DIMACS instances but have poor performance on two, non-overlapping subsets of instances. For RLS this poor performance subset consists of the brock family of instances and, for PLS, the poor performance subset consists of the keller and MANN instances. While RLS and PLS have a number of similarities, the differences between the two heuristics is highlighted by these two subsets of instances and neither RLS nor PLS can claim to be fully effective heuristics with regard to the complete DIMACS benchmark. In fact, all existing maximum clique heuristics appear to have a poor performance subset of the DIMACS instances which is understandable given the range of instance types in the DIMACS benchmark and the quotation cited above. The first direct attempt at addressing this consistency issue for the maximum clique problem appears to be PLS (Pullan 2006) which repeatedly cycled through three different low level heuristics (Greedy, Random and Penalty), having in this way a broader set of heuristics suitable for a wider class of instances.

From the RLS and PLS results, two of the major instance characteristics that appear to directly influence heuristic performance are, firstly, the proportion of plateau area near the optimal cliques on the instance search-space being explored. If this is relatively high then the heuristic must be able to effectively search plateau areas without cycling. However this ability is wasteful when the instance search-space has a low proportion of plateaus near the optimal cliques. The second instance characteristic that appears to be important is the relationship between the distribution of the vertex degrees in the maximum clique and the distribution of the vertex degrees in the graph. As the vertex degree is the only characteristic directly associated with a vertex, most greedy maximum clique heuristics search solely on the basis of this and will easily find maximum cliques that consist only of the higher degree nodes but have a high probability of being defeated by maximum cliques that contain lower degree vertices. Using vertex penalties to bias the search towards the lower degree vertices can be useful but requires a careful balance so that some vertices are not completely locked out from the search. The more difficult maximum clique problems would appear to be those where the range of vertex degrees in the maximum cliques is similar to the vertex degree range of the graph.

With regard to these two instance characteristics, an analysis of the DIMACS and BHOSLIB benchmarks shows that the instances can be broadly be grouped into the following four categories:

- A large number of the instances (*e.g.* DIMACS C family) have maximum cliques that consist only of higher degree vertices and can be effectively solved by a greedy heuristic such as RLS or the greedy phase of PLS.

- A small number of DIMACS instances (*e.g.* DIMACS brock family) have maximum cliques that consist of medium to lower degree vertices and are effectively solved by the penalty phase of PLS.
- A small number of DIMACS instances (*e.g.* DIMACS MANN family) have a large proportion of plateaus in the instance search-space and can be effectively solved by a heuristic, such as RLS, that devotes a large proportion of time to searching plateaus and also has a mechanism to prevent cycling.
- The BHOSLIB instances have maximum cliques consisting of vertices whose distribution of vertex degree closely matches that for the complete graph. These are difficult instances for both greedy and penalty based heuristics.

From these groupings it can be reasonably concluded that, for a parallel maximum clique hyper-heuristic to be effective on both the DIMACS and BHOSLIB benchmarks, it should be composed of low level heuristics that focus on different ranges with regard to the distribution of the vertex degrees in the cliques constructed and also on being efficient for instances with different proportions of plateau area in their search-space.

The most straight-forward implementation of a parallel maximum clique hyper-heuristic would be, as both RLS and PLS are stochastic local search heuristics, to use either RLS or PLS as heuristics performing concurrent independent runs. However this technique will not produce a hyper-heuristic that materially improves on the RLS/PLS performance for those DIMACS instance subsets where they perform poorly. The next step in this evolution of parallelism would be a hyper-heuristic that runs some number of PLS heuristics in parallel with some number of RLS heuristics. This clearly would give a hyper-heuristic that would be effective for all DIMACS instances but is still less than optimal, primarily because there is no gain in running a heuristic which is not suitable for the current instance (for example, PLS on a DIMACS MANN instance or RLS on a DIMACS brock instance) and these cores could be more effectively utilised by a heuristic that was more suited to the current instance. This simple hyper-heuristic also ignores any possibility of improved performance resulting from communication between heuristics so that, for example, redundant searching is not performed (the performance of such a parallel maximum clique hyper-heuristic is investigated further in Sect. 4).

3 The CLS hyper-heuristic

CLS is a parallel maximum clique hyper-heuristic that incorporates four low level heuristics, namely: Greedy Search (*GREEDY*) which uses random selection within vertex degree, is biased towards higher degree vertices and performs limited plateau search; Level Search (*LEVEL*) which uses random selection within vertex degree, is biased towards higher degree vertices and performs extensive plateau search; Focus Search (*FOCUS*) which identifies a vertex degree as being the focus of the current iteration and selects vertices with a goal of obtaining an average vertex degree for the clique as close as possible to this focus vertex degree. This allows the bias of the search to be manipulated during the search; and, Penalty Search (*PENALTY*), which

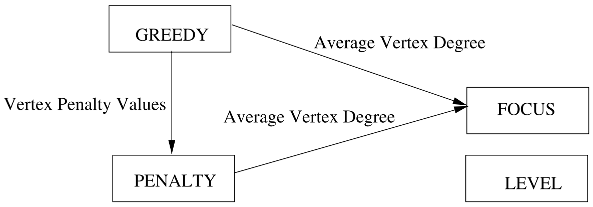

Fig. 1 Sharing of information between the CLS heuristics

performs random selection within minimum vertex penalties and is biased towards lower degree nodes.

The allocation of CLS heuristics to computer cores is, at the commencement of the search, an initial configuration that depends on the number of cores available. After a short period of time, CLS reconfigures the allocation of heuristics to cores into either the Plateau configuration or the Non-Plateau configuration based on feedback from *GREEDY* during the CLS start-up period.

The sharing of information between the CLS heuristics (Fig. 1) is firstly by using vertex penalties accumulated by *GREEDY* as input into *PENALTY* and, secondly, by using the average vertex degree of the cliques constructed by *GREEDY* and *PENALTY* to provide information when determining a new vertex degree focus for *FOCUS*.

The basic structure of all four CLS heuristics is common and is now described using the following notation: $G(V, E)$—an undirected graph with $V = \{1, 2, \ldots, n\}$, $E \subseteq \{\{i, j\} : i, j \in V\}$; $N(i) = \{j \in V : \{i, j\} \in E\}$—the vertices adjacent to i; K—current clique of G; and $C_0 = \{i \in V : N(i) \supseteq K\}$ while in general $C_p(K) = \{i \in V : |K \setminus N(i)| = p\}$. $p = 0, 1, \ldots$—the set of all vertices not adjacent to exactly p vertices in K. That is, C_0 is the increasing set of vertices while C_1 is the level set of vertices.

```
Function CLS-Heuristic(tcs, max-time)
        Input: integer tcs (target clique size); max-time;
        Output: Clique of cardinality tcs or ‘failed’;
 1.     ⟨Randomly select a vertex $v \in V$, $K := \{v\}$⟩;
 2.     do
 3.         do
 4.             while $C_0(K) \neq \emptyset$ do
 5.                 $v := Select(C_0(K))$;
 6.                 $K := K \cup \{v\}$;
 7.                 if $|K| = tcs$ then return $K$;
 8.             end while
 9.             if $C_1(K) \neq \emptyset$ then
10.                 $v := Select(C_1(K))$;
11.                 $K := [K \cup \{v\}] \setminus \{i\}$, where $\{i\} = K \setminus N(v)$;
12.             end if;
13.         while Continue;
14.         Restart;
15.     while (time < max-time);
16.     return ‘failed’;
```

This *CLS-Heuristic* framework alternates between an iterative improvement phase (lines 4–8), during which vertices from the increasing set $C_0(K)$ are added to the current clique K, and a plateau search phase (lines 9–12), in which vertices from the level set $C_1(K)$ are swapped with the vertex in K with which they do not share an edge. The search phase terminates for *GREEDY*, *FOCUS* and *PENALTY* when $C_0(K) = \emptyset$ and either $C_1(K) = \emptyset$ or all vertices that are in $C_1(K)$ have already been an element of K during the current iteration. For *LEVEL*, the search phase terminates after an extensive plateau search using a tabu mechanism to prevent search cycles. As the final step of the iteration, a perturbation of K is performed to generate a new starting point for the next iteration. Iterations are repeated until either the maximum clique is found or the allowed time is exceeded.

Within the *CLS-Heuristic* framework, the individual heuristics are implemented by the use of the following functions:

- *Select*: With regard to selection of vertices from C_0:
  - *GREEDY* and *LEVEL* are uniform random from the highest degree sub-set of C_0.
  - *FOCUS* is uniform random from the vertices in C_0 whose degree is closest to the current vertex focus for *FOCUS*.
  - *PENALTY* is uniform random from the minimum vertex penalty sub-set of C_0.

  With regard to selection of vertices from C_1:
  - *GREEDY* is uniform random from the highest degree sub-set of C_1.
  - *LEVEL* is uniform random from the subset of K which causes the maximum increase in $|C_0|$.
  - *FOCUS* is uniform random from the sub-set of vertices of C_1 whose degree is closest to the current vertex degree focus for *FOCUS*.
  - *PENALTY* is uniform random from the lowest penalty sub-set of C_1.
- *Continue*: For *GREEDY*, *FOCUS* and *PENALTY* iterations will continue while either C_0 is not empty or C_1 contains vertices that have not been in K since the first use of a vertex from C_1. For *LEVEL*, the search phase terminates after an extensive plateau search using a tabu mechanism to prevent search cycles. The version of *LEVEL* used within CLS is based on the RLS (Battiti and Protasi 2001) heuristic.
- *Restart*: Performs a perturbation in the search by uniform randomly select a vertex v, adding this to K and removing all vertices from K that are not connected to v. This perturbation mechanism provides for some continuity in the search and also maintains K as relatively large at all times.

The CLS low level heuristics were designed to be non-overlapping in their roles during the search and do this firstly by the heuristic differences noted above and also by sharing information that basically describes where they have been searching. In more detail, the rationale for each heuristic is as follows:

- *GREEDY*: The role of *GREEDY* is two-fold. Firstly it provides a mechanism for generating vertex penalties to be input to *PENALTY* so that it is guided towards vertices that tend to be ignored by a greedy search. Secondly, it is an appropriate search mechanism for problems where the search-space being explored is ‘rugged’ in the sense that it contains a relatively small proportion of plateau areas.

- *LEVEL*: This heuristic uses a greedy selection method but differs from *GREEDY* in that it also performs extensive plateau search using an adaptive tabu technique to prevent search cycles.
- *PENALTY*: The selection method within *PENALTY* is based on the vertex penalties created by *GREEDY*. The purpose of vertex penalties is to provide additional diversification to the search process, which otherwise could easily stagnate in situations where the current clique has few or no vertices in common with an optimal solution for a given maximum clique instance. Perhaps the most obvious approach for avoiding this kind of search stagnation is to simply restart the constructive search process from a different initial vertex. However, even if there is random (or systematic) variation in the choice of this initial vertex, there is still a risk that the heuristic guidance built into the greedy construction mechanism causes a bias towards a limited set of suboptimal cliques. Therefore, integer penalties are associated with the vertices that modulate the heuristic selection function used in the greedy construction procedure in such a way that vertices that repeatedly occur in the cliques obtained from the greedy constructive search process are discouraged from being used in future penalty based constructions.
- *FOCUS*: The role of *FOCUS* is to target areas of the search domain that the three other heuristics tend to ignore. This is done by using the average vertex degree of the cliques located by *GREEDY* and *PENALTY* to guide *FOCUS* so that it focuses on constructing cliques whose average vertex degree are different from those normally found by *GREEDY* and *PENALTY*. *FOCUS* determines a new target average vertex degree at each *Restart* and monitors the difference between this target clique average vertex degree and that achieved so that it can adaptively offset the target clique average vertex degree for subsequent iterations.

The overall CLS hyper-heuristic operates by, if the user has not specified the number of cores available, determining this from the system and initially allocating the heuristics to the cores as shown in the Initial Configuration in Table 1. All heuristics are then activated until either the nominated optimal clique size is located or the maximum allowed selections has been exceeded. If after 1,000,000 selections *GREEDY* determines that there is a large amount of plateau area then the heuristics are reallocated to the cores as shown for the Plateau Configuration in Table 1 otherwise they are reallocated as shown for the Non-Plateau Configuration. These configurations remain until the trial is completed. With the exception of the single core case, CLS allocates no more than two heuristics to each available core. As the best suited heuristic for an instance is never known exactly, CLS, for any number of cores and for all three configurations, keeps at least one copy of the each of the four CLS heuristics active. While this can reduce the efficiency of CLS on some specific instances, it does improve the consistency of CLS over all benchmark instances. It was determined empirically that the allocation of heuristics to cores shown in Table 1 provided the best results, in terms of consistency and execution time, for CLS over both benchmarks evaluated in Sect. 4.

CLS is implemented in C/C++ and operates in one of two modes. In the first mode, the LAM/MPI (Burns et al. 1994; Squyres and Lumsdaine 2003) software is required which enables CLS to execute on both Microsoft Windows and Unix computers using either a single computer or a cluster of networked computers. In this mode, any

Table 1 Mappings of CLS heuristics to cores (numbered from 1), as the number of available cores changes, for each of the three possible CLS heuristic to core configurations. As described in the text, CLS always starts in the Initial Configuration and then, after determining the nature of the problem, reconfigures to either the Plateau or Non-Plateau Configuration. Heuristics sharing a core do so with equal priority

Number of	Core	CLS configuration		
available cores	number	Initial	Plateau	Non-Plateau
1	1	*GREEDY*, *PENALTY* *LEVEL*, *FOCUS*	*GREEDY*, *PENALTY* *LEVEL*, *FOCUS*	*GREEDY*, *PENALTY* *LEVEL*, *FOCUS*
2	1	*GREEDY*, *PENALTY*	*GREEDY*, *PENALTY*	*GREEDY*, *PENALTY*
	2	*LEVEL*, *FOCUS*	*LEVEL*, *FOCUS*	*LEVEL*, *FOCUS*
3	1	*GREEDY*, *PENALTY*	*GREEDY*, *PENALTY*	*GREEDY*, *PENALTY*
	2	*LEVEL*	*LEVEL*, *FOCUS*	*LEVEL*, *FOCUS*
	3	*FOCUS*	*LEVEL*	*FOCUS*
4	1	*GREEDY*	*GREEDY*, *PENALTY*	*GREEDY*, *PENALTY*
	2	*PENALTY*	*LEVEL*, *FOCUS*	*LEVEL*, *FOCUS*
	3	*LEVEL*	*LEVEL*	*FOCUS*
	4	*FOCUS*	*LEVEL*	*FOCUS*
>4	1	*GREEDY*	*GREEDY*, *PENALTY*	*GREEDY*, *PENALTY*
	2	*PENALTY*	*LEVEL*, *FOCUS*	*LEVEL*, *FOCUS*
	3	*LEVEL*	*LEVEL*	*FOCUS*
	4	*FOCUS*	*LEVEL*	*FOCUS*
	>4	*FOCUS*	*LEVEL*	*FOCUS*

of the computers may contain any number of cores and all will be utilised. All CLS heuristics execute asynchronously with messages passed between them via the MPI queue facility. The second mode is a thread based implementation which operates on a single Unix computer containing any number of cores. Again, all CLS heuristics execute asynchronously with messages passed between them using locked array data structures. Both these variants of CLS have similar performance. The results presented below are for a LAM/MPI based implementation of CLS,[2] executing on a dedicated single computer with eight cores, and can be taken as representative for both modes of CLS operation. Unless otherwise noted, all results presented below are for a four core version of CLS (that is, unless noted, only four of the eight cores available were utilised).

4 Empirical performance results

In order to evaluate the performance and behaviour of CLS, extensive computational experiments were first performed on all maximum clique instances from the Second

[2]The C++ code for the MPI version of CLS is available at www.cit.gu.edu.au/~s994853/CLS.

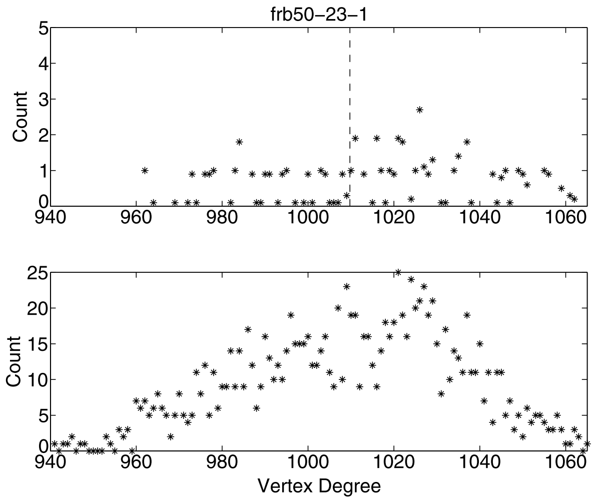

Fig. 2 For frb50-23-1, the *top graph* shows, averaged over 10 trials, the number of times a vertex of a particular degree appeared in the optimal maximum cliques (the *dashed line* is the average vertex degree for the optimal maximum cliques). The minimum vertex degree for frb50-23-1 is 941 while the maximum vertex degree is 1065 and the distribution of vertex degree over the 1150 vertices of frb50-23-1 is shown in the *lower graph*

DIMACS Implementation Challenge (1992–1993)[3] were performed. These instances have also been used extensively for benchmarking purposes in the recent literature on maximum clique heuristics. The 80 DIMACS maximum clique instances were generated from problems in coding theory, fault diagnosis problems, Keller's conjecture on tilings using hypercubes, and the Steiner triple problem, in addition to randomly generated graphs and graphs where the maximum clique has been "hidden" by incorporating low-degree vertices. These problem instances range in size from less than 50 vertices and 1000 edges to greater than 3300 vertices and 5,000,000 edges.

In a second series of experiments, the Benchmarks with Hidden Optimum Solutions for Graph Problems (Maximum Clique, Maximum Independent Set, Minimum Vertex Cover and Vertex Colouring)[4] (BHOSLIB) were used. These maximum clique instances are transformed from forced satisfiable SAT benchmarks of Model RB, with the set of vertices and the set of edges respectively corresponding to the set of variables and the set of binary clauses in SAT instances. In effect, this allows an exact solution to be hidden within a random graph and, as shown in Fig. 2, in contrast to randomly generated graphs used for maximum clique problems, the maximum clique can contain a wide range of low and high degree vertices.

All experiments for this study were performed on a computer that, when executing the DIMACS Maximum Clique Machine Benchmark[5] required 0.24 CPU seconds for r300.5, 1.49 CPU seconds for r400.5 and 5.65 CPU seconds for r500.5. In the following, unless explicitly stated otherwise, all CLS run-times refer to the reference machine and were generated using a four core version of CLS.

[3]http://dimacs.rutgers.edu/Challenges/.

[4]http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm.

[5]*dmclique*, ftp://dimacs.rutgers.edu in directory /pub/dsj/clique.

Table 2 CLS performance results, averaged over 100 consecutive successful independent trials, for the 27 DIMACS benchmark instances where the average run time was greater than 0.01 seconds. For all the remaining 53 DIMACS benchmark instances, CLS achieved a 100% success rate in less than 0.01 seconds. The target clique size, for each instance, is shown in the 'ω' column; 'SCPU' is the PLS CPU time in seconds from Pullan (2006) scaled to the reference computer while 'RT' is the CLS run-time in seconds, averaged over all successful trials, for each instance. The results for those instance names flagged with '*' in the 'SCPU' column are for RLS run on the reference computer

Instance	ω	PLS		CLS		Instance	ω	PLS/RLS		CLS	
		Success	SCPU	Success	RT			Success	SCPU	Success	RT
brock200_2	12	100	0.01	100	0.01	c500.9	57	100	0.07	100	0.04
brock200_4	17	100	0.03	100	0.05	dsjc1000.5	15	100	0.18	100	0.05
brock400_1	27	100	0.42	100	0.44	gen400_p0.9_55	55	100	0.10	100	0.04
brock400_2	29	100	0.15	100	0.11	keller6*	59	100	7.16	100	1.19
brock400_3	31	100	0.07	100	0.05	MANN_a27	126	100	0.01	100	0.03
brock400_4	33	100	0.04	100	0.03	MANN_a45*	344	100	35.24	100	20.03
brock800_1	23	100	11.73	100	2.63	MANN_a81*	1098	100	64.64	100	27.04
brock800_2	24	100	9.51	100	2.25	p_hat1500-1	12	100	1.28	100	0.48
brock800_3	25	100	5.88	100	1.64	san1000	15	100	1.84	100	0.53
brock800_4	26	100	2.55	100	0.76	san400_0.5_1	13	100	0.02	100	0.01
c1000.9	68	100	0.73	100	0.18	san400_0.7_1	40	100	0.02	100	0.01
c2000.5	16	100	0.28	100	0.09	san400_0.7_2	30	100	0.04	100	0.01
c2000.9	78	100	43.96	100	9.47	san400_0.7_3	22	100	0.07	100	0.06
c4000.5	18	100	58.31	100	17.73						

4.1 CLS performance

To evaluate the performance of CLS on the DIMACS and BHOSLIB benchmark instances, 100 independent trials were performed for each instance using target clique sizes (tcs) corresponding to the respective provably optimal clique sizes or, in cases where such provably optimal solutions are unknown, with three exceptions, largest known clique sizes. Each trial was allowed a maximum number of vertex selections before a failure occurred. To provide a better basis for comparison, where CLS was not 100% successful for an instance, the run time corresponding to CLS exceeding the maximum allowed selections has been documented because, even with the same maximum number of vertex selections, the run time is instance dependent and is a more comparable and representative metric for the overall computation effort performed by CLS.

The CLS performance results (averaged over 100 independent trials) are shown in Table 2 for the DIMACS benchmark instances that required more than 0.01 seconds of run time to achieve a 100% success rate. CLS finds the target clique size with a success rate of 100% over all 100 trials per instance for all the 80 DIMACS instances. The average run-time for CLS to reach the target clique size is less than 1 second for 72 of the 80 instances, and an average run-time of more than 10 seconds is only required for 3 of the 8 remaining instances, all of which have at least 800 vertices.

The instances where the target clique size was not the optimal (best known) were C2000.9, where all 100 trials achieved 78 as compared to 80 (Grosso et al. 2008);

Table 3 CLS performance results, averaged over 100 consecutive independent trials, for the 40 BHOSLIB benchmark instances. The optimal maximum clique size, for each instance, is shown in the 'ω' column; 'Success' gives the number of successful trials (from a total of 100) in which the optimal maximum clique size was located; 'SCPU' is the scaled CPU time for PLS (averaged over successful trials only) from Pullan (2008) while 'RT' is the CLS run time averaged over all successful runs. The 'FT' column contains the maximum CLS run time allowed before a trial was deemed to be unsuccessful. All times are in seconds

Instance	ω	PLS		CLS		Instance	ω	PLS		CLS		
		Success	SCPU	Success	RT			Success	SCPU	Success	RT	FT
30-15-1	30	100	0.05	100	0.01	50-23-1	50	72	803.75	100	114.51	
30-15-2	30	100	0.07	100	0.01	50-23-2	50	45	900.27	99	270.07	1510
30-15-3	30	100	0.53	100	0.17	50-23-3	50	16	800.53	36	589.38	1420
30-15-4	30	100	0.03	100	0.00	50-23-4	50	100	97.20	100	11.49	
30-15-5	30	100	0.25	100	0.12	50-23-5	50	99	335.38	100	36.19	
35-17-1	35	100	3.20	100	1.81	53-24-1	53	6	1312.48	19	683.61	1550
35-17-2	35	100	0.95	100	0.30	53-24-2	53	23	1190.64	100	291.86	
35-17-3	35	100	0.20	100	0.04	53-24-3	53	66	911.44	100	211.66	
35-17-4	35	100	4.34	100	1.01	53-24-4	53	46	1094.08	90	469.25	1650
35-17-5	35	100	0.61	100	0.19	53-24-5	53	85	753.22	100	55.19	
40-19-1	40	100	2.55	100	0.55	56-25-1	56	12	953.34	94	613.35	1800
40-19-2	40	100	41.59	100	11.20	56-25-2	56	6	1308.64	87	562.36	1700
40-19-3	40	100	3.71	100	0.83	56-25-3	56	8	1135.08	97	478.29	1800
40-19-4	40	100	17.72	100	8.69	56-25-4	56	68	1002.42	100	97.60	
40-19-5	40	100	76.67	100	12.49	56-25-5	56	81	837.18	100	246.55	
45-21-1	45	100	31.79	100	7.28	59-26-1	59	0	–	29	751.56	1600
45-21-2	45	100	63.50	100	12.74	59-26-2	59	0	–	6	774.57	1900
45-21-3	45	100	318.27	100	17.04	59-26-3	59	6	1482.88	46	804.35	1850
45-21-4	45	100	45.57	100	8.42	59-26-4	59	5	1571.94	33	780.22	1900
45-21-5	45	100	83.70	100	10.98	59-26-5	59	78	917.24	100	153.65	

MANN_a45 where all 100 trials achieved 344 as compared to 345 (Hansen et al. 2004; Battiti and Protasi 2001) and MANN_a81 where all 100 trials achieved 1098 as compared to 1099 (Katayama et al. 2004). CLS was also able to locate the 79 vertex clique for C2000.9 in 10 out of 10 trials with an average run time of 273.71 seconds, the 80 vertex clique for C2000.9 in 1 out of 10 trials (2150 seconds maximum run time allowed for each trial), the 345 vertex clique for MANN_a45 in 1 out of 10 trials (2000 seconds maximum run time allowed for each trial), and the 1099 clique for MANN_a81 in 1 out of 10 trials (5800 seconds maximum run time allowed for each trial).

Table 3 summarises the results obtained, over 100 consecutive trials, for CLS on the BHOSLIB benchmark instances. Also shown in Table 3 is the results obtained for PLS (Pullan 2006), to date the only other maximum clique heuristic to be applied to the BHOSLIB benchmark. For the more difficult BHOSLIB instances that both PLS and CLS were 100% successful on, even if the SCPU for PLS is divided by four (= the number of cores utilised by CLS), the results obtained by CLS are an improvement on those obtained by PLS. In addition, there is a marked improvement in the number

Table 4 CLS performance as compared to SAT Solvers in the SAT Competition 2004 for the BHOSLIB benchmark. For each instance, the number of SAT solvers (from a total of 55 SAT solvers) in the SAT Competition 2004 that were able to solve the corresponding SAT problem and also the number of successful trials (from a total of 100) in which CLS located the optimal maximum clique are shown

Instance	SAT 2004 results	CLS success	Instance	SAT 2004 results	CLS success
frb40-19-1	Solved by 28 solvers	100	frb53-24-1	Unsolved	19
frb40-19-2	Solved by 27 solvers	100	frb53-24-2	Unsolved	100
frb45-21-1	Solved by 8 solvers	100	frb56-25-1	Unsolved	94
frb45-21-2	Solved by 5 solvers	100	frb56-25-2	Unsolved	87
frb50-23-1	Solved by 1 solver	100	frb59-26-1	Unsolved	29
frb50-23-2	Solved by 1 solver	99	frb59-26-2	Unsolved	6

of instances which were solved with a 100% success rate. For the frb100-40 instance, CLS was able to locate the 97 vertex clique as compared to the optimal clique size of 100. As an alternative means of comparison, some corresponding SAT instances of the BHOSLIB maximum clique instances were used for the SAT Competition 2004 (55 SAT solvers)[6] with the results shown in Table 4. As can be seen, the results for CLS are an improvement on these results.

These results demonstrate that CLS achieves excellent and robust performance on the DIMACS and BHOSLIB benchmark instances. With regard to comparative performance, in Pullan and Hoos (2006), extensive comparisons are presented between DLS-MC and DAGS (Grosso et al. 2004), GRASP (Resende et al. 1998) (using the results contained in Grosso et al. (2004)), *k-opt* (Katayama et al. 2004), RLS (Battiti and Protasi 2001), GENE (Marchiori 2002), ITER (Marchiori 2002) and QUALEX-MS (Busygin 2006). In Pullan (2006), DLS-MC is directly compared with PLS.

Clearly the results presented above must be interpreted in the context of comparing single core heuristics (PLS/RLS) with a four core hyper-heuristic (CLS). Table 5 addresses this issue by comparing CLS with five, four core hyper-heuristics constructing using all possible combinations of RLS and PLS, with different random initialisations, and terminating when the target maximum clique is first located on the more difficult DIMACS and BHOSLIB instances. As shown in Table 5, all of the PLS/RLS parallel combinations were unable to successfully complete the 100 trials for all the selected instances. This is in contrast to CLS which did complete all 100 trials for each instance successfully although it was not the most efficient hyper-heuristic for all instances.

Some of the results in Table 5 can be explained in terms of the capabilities of PLS, RLS and the CLS heuristics. For example, for MANN_a81, in the RLS/PLS combinations this is always solved by RLS so the performance is totally dependent on the number of copies of RLS active. For CLS, MANN_a81 is almost always (Table 5) solved by the *LEVEL* heuristic and, as CLS reconfigures to the Plateau configuration for this instance, 2.5 cores will be used for the *LEVEL* heuristic which implies the

[6]http://www.nlsde.buaa.edu.cn/kexu/benchmarks/graph-benchmarks.htm.

Table 5 Average run-times, over 100 successful trials for CLS and all combinations of zero to four copies of RLS/PLS running in parallel (for example, RRPP signifies 2 copies of RLS and 2 copies of PLS with different random initialisations). An entry of—signifies that the RLS/PLS combination was unable to successfully complete all 100 trials in the allowed run-time and is an ineffective combination of RLS and PLS for that DIMACS and BHOSLIB benchmark instance. Also shown is the number of times, from 100 trials on all instances, each CLS heuristic was the successful heuristic in locating the target clique size

Instance	RRRR	RRRP	RRPP	RPPP	PPPP	CLS	GREEDY	LEVEL	FOCUS	PENALTY
C2000.9	9.55	11.40	12.12	23.49	49.30	9.47	**58**	24	18	0
C4000.5	9.38	9.49	10.03	16.53	20.54	17.73	26	**53**	16	5
MANN_a45	18.22	14.16	18.71	18.45	22.78	20.03	0	**85**	0	15
MANN_a81	15.07	19.88	28.58	54.49	216.75	27.04	0	**95**	0	5
keller6	0.43	0.40	0.63	1.18	413.15	1.19	0	**97**	3	0
brock800_1	–	8.94	5.18	4.01	2.81	2.63	0	0	1	**99**
brock800_2	–	5.71	3.30	2.23	1.64	2.25	0	0	2	**98**
frb53-24-2	–	–	–	–	–	291.86	2	2	**96**	0
frb56-25-4	–	–	–	–	–	97.60	1	11	**88**	0

result for CLS should be between that obtained for RRRP and that obtained for RRPP. The fact that it is closer to the result for RRPP is caused by the delay before CLS reconfigures to the Plateau configuration during which only one core is used by the *LEVEL* heuristic. The failure of all combinations of RLS/PLS on the two BHOSLIB instances is caused by RLS/PLS lacking any component that performs the role of the CLS heuristic *FOCUS* (which is almost always the CLS heuristic that is successful on the larger BHOSLIB instances (Table 5)).

5 Discussion

To gain a deeper understanding of the run-time behaviour of CLS and the efficacy of its underlying mechanisms, additional empirical analyses were performed. These studies were an investigation of the performance of the individual heuristics and the relationship between the heuristics and instance characteristics, the importance of the information passed between heuristics and the scalability of CLS.

5.1 CLS heuristic performance

Table 5 shows the frequency of each heuristic identifying the optimal solution for the more difficult DIMACS instances and two representative BHOSLIB instances. From these results it is clear that *PENALTY*, in conjunction with the vertex penalties supplied by *GREEDY*, is the effective heuristic for the DIMACS brock family of instances. This is to be as expected because, as shown in Pullan (2006), for brock800_1, the vertices in the optimal maximum clique are biased towards the lower than average vertex degree (note that the DIMACS brock instances were created in an attempt to defeat greedy heuristics that used vertex degree for selecting vertices to be added to the current clique (Brockington and Culberson 1996)). Also, as shown in Pullan (2006) for C1000.9, the maximal clique vertices are biased towards the higher degree

Table 6 Average $|C_0|$ and $|C_1|$ as the current clique size ($|K|$) increases near the target clique size for C2000.9, MANN_a45, keller6 and frb53-23-1. Also shown is R, the ratio $|C_1|/(N - |K|)$ used to identify instances whose search-space contains a relatively large proportion of plateaus

C2000.9				MANN_a45				keller6				frb53-23-1			
$\lvert K\rvert$	$\lvert C_0\rvert$	$\lvert C_1\rvert$	R	$\lvert K\rvert$	$\lvert C_0\rvert$	$\lvert C_1\rvert$	R	$\lvert K\rvert$	$\lvert C_0\rvert$	$\lvert C_1\rvert$	R	$\lvert K\rvert$	$\lvert C_0\rvert$	$\lvert C_1\rvert$	R
69	0.33	4.54	0.00	335	0.02	387.36	0.55	50	0.05	6.63	0.00	41	0.13	9.54	0.01
70	0.22	3.80	0.00	336	0.02	384.45	0.55	51	0.05	6.00	0.00	42	0.09	6.57	0.01
71	0.14	3.27	0.00	337	0.01	381.79	0.55	52	0.05	5.70	0.00	43	0.06	4.76	0.00
72	0.10	2.88	0.00	338	0.01	369.34	0.53	53	0.06	4.84	0.00	44	0.04	3.70	0.00
73	0.05	2.47	0.00	339	0.00	349.35	0.50	54	0.02	4.07	0.00	45	0.03	3.01	0.00
74	0.04	2.27	0.00	340	0.00	327.68	0.47	55	0.00	2.80	0.00	46	0.02	2.52	0.00
75	0.00	2.33	0.00	341	0.00	306.75	0.44	56	0.00	0.00	0.00	47	0.01	2.13	0.00
76	0.00	1.57	0.00	342	0.00	282.98	0.41	57	0.00	0.00	0.00	48	0.01	1.87	0.00
77	0.00	0.00	0.00	343	0.00	193.90	0.28	58	0.00	0.00	0.00	49	0.00	1.84	0.00

vertices and this is also the case for C2000.9 and C4000.5 so it is to be expected that *GREEDY* and *LEVEL* would be the dominant heuristics for these instances.

For the MANN instances, as shown in Table 6, they have a high proportion of plateau area. For example, for MANN_a45, with a clique size one less than the target clique size, 28% of vertices that were not currently in the clique could be swapped with a vertex that is in the clique and the clique size would not change. *LEVEL* is the dominant heuristic for these instances because of its extensive plateau searching capability. For the keller6 instance, both $|C_0|$ and $|C_1|$ are very low (<0.005) when $|K| > 55$ ($\omega = 59$) which tends to indicate that the maximum clique region could be visualised as being surrounded by a large number of lower peaks, arising from lower plateaus with only a few paths rising to the optimal maximum clique. This structure would certainly favour the plateau searching capabilities of *LEVEL*.

For the BHOSLIB instances, as shown in Fig. 2 for frb50-23-1, there is a mix of lower and higher degree vertices in the optimal maximum cliques which tend to defeat the *GREEDY*, *LEVEL* and *PENALTY* heuristics. However, as shown in Fig. 3, *FOCUS* generates cliques whose average vertex degree is somewhat between those generated by *GREEDY* and *PENALTY* and, from Table 5, is, by far, the most successful CLS heuristic for the more difficult of the BHOSLIB instances.

The overall totals from Table 5 for each heuristic was 87 for *GREEDY*, 367 for *LEVEL*, 224 for *FOCUS* and 222 for *PENALTY*. For 100 trials on each of the 80 instances in the complete DIMACS benchmark, the overall totals for each heuristic were 1780 for *GREEDY*, 1659 for *LEVEL*, 1993 for *FOCUS* and 2568 for *PENALTY*. While these results are skewed by the CLS reconfigurations, they are basically in agreement with the success rates from a two core CLS optimisation (where all heuristics have equal processor time) which were, in *GREEDY*, *LEVEL*, *FOCUS* and *PENALTY* order, (0, 86, 0, 14) for MANN_a81 and (2, 26, 72, 0) for frb56-25-4.

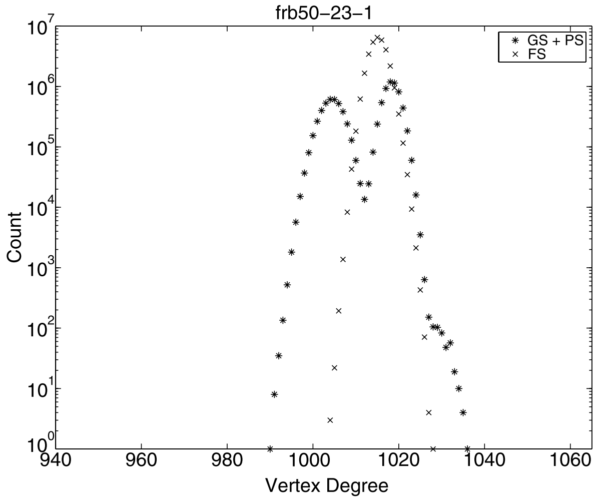

Fig. 3 For frb50-23-1, the average vertex degree of the clique at the point at which C_0 first became empty for *GREEDY* + *PENALTY* ($GS + PS$) and *FOCUS*. Of note is that the range of the average vertex degree of reasonable size cliques is considerably smaller than the range of vertex degrees in the frb50-23-1 (941–1065)

5.2 CLS heuristic interactions

The basic data flow between CLS heuristics is: vertex penalty values from *GREEDY* to *PENALTY*, distribution of average vertex degree of cliques from *GREEDY* to *FOCUS* and distribution of average vertex degree of cliques from *PENALTY* to *FOCUS*. These values are accumulated by the sending heuristic and requested by the receiving heuristic every 100,000 selections. The impact of this data on the behaviour of both *PENALTY* and *FOCUS* can be seen in Fig. 3 which shows the combined distribution of the average vertex degree for cliques generated by *GREEDY* and *PENALTY* (*GS* + *PS* curve) and the average vertex degree for cliques generated by *FOCUS*. Significant features on the $GS + PS$ curve are the peak on the left, which is from the *PENALTY* cliques, while the peak on the right of the $GS + PS$ curve is from the *GREEDY* cliques and show the influence of the greedy selection based on vertex degree. The $GS + PS$ curve also shows that, even though the vertex degrees in the frb50-23-1 graph range from 941 to 1065, it is basically impossible to generate any reasonable size clique whose average vertex degree is outside the range 990 to 1035. Also shown is how *FOCUS*, because of the information received from *GREEDY* and *PENALTY*, tends to visit cliques less frequented by *GREEDY* and *PENALTY*.

5.3 Scalability of CLS

Table 7 shows the scalability of CLS as the number of cores increases from one to eight for the MANN_a81 and frb56-25-4 instances. These two instances were chosen because one is addressed by the CLS Plateau configuration (MANN_a81) while the other (frb56-25-4) is addressed by the Non-Plateau configuration. The important factor that must be kept in mind when interpreting the data in Table 7 is that the configurations that CLS adopts during the search have an impact. For example, MANN_a81 is almost always solved by the *LEVEL* heuristic but, in the one core CLS, *LEVEL* only utilises one quarter of a core, in the two core case, *LEVEL* will

Table 7 CLS average run time (in seconds, over 100 successful trials) for MANN_a81 and frb50-23-4 as the number of cores is increased from one to eight

Instance	Cores							
	1	2	3	4	5	6	7	8
MANN_a81	141.35	78.81	31.30	27.04	19.09	17.56	11.31	10.88
frb56-25-4	881.00	406.19	158.61	97.60	76.45	60.75	52.25	48.89

always utilise one half of a core. When N, the number of cores available increases above two, *LEVEL* uses a single core during CLS start-up and $(N-1.5)$ cores after CLS reconfiguration to the Plateau configuration. The identical situation exists with regard to *FOCUS* for frb56-25-4.

From Table 7 it can also be concluded that, for the four core case, if all cores were devoted to *LEVEL* for MANN_a81 then the expected run time for CLS would be approximately 18.33 (as compared to 27.04) and, for frb56-25-4, 68.60 (as compared to 97.60). This basically reflects the trade-off incurred for CLS consistency over all 120 instances of the DIMACS and BHOSLIB benchmarks.

6 Conclusions and future work

This study has demonstrated that, by applying, in parallel, variants of the general paradigm of dynamic local search to the maximum clique problem, using multi core computers, the state-of-the-art in maximum clique solving can be improved. CLS builds on previous maximum clique heuristics, in particular the PLS and RLS heuristics, in addition to adding a new concept, focused search. All of these heuristics follow the schema of iteratively clique building and then applying a perturbation. CLS combines four heuristics which are effective for different instance types. The first heuristic, *GREEDY*, uses vertex degrees to bias the search towards cliques containing higher degree vertices. The second heuristic, *LEVEL*, is also a greedy heuristic using vertex degree but has extensive plateau searching capabilities. The third heuristic, *FOCUS*, is driven by input from *GREEDY* and *PENALTY* and focuses on areas of the search domain that those two heuristics tend to ignore. The final CLS heuristic, *PENALTY*, uses vertex penalties generated by *GREEDY* to bias the search towards cliques containing lower degree vertices. The vertex penalties are increased when the vertex is in the current clique when a perturbation occurs and are subject to occasional decrease, which effectively allows the heuristic to 'forget' vertex penalties over time. The frequency with which these decrease is adaptively modified to obtain near optimal performance. To optimise the performance of CLS, guidance information is passed between heuristics directing them to particular areas of the search domain. In addition, CLS dynamically reconfigures the allocation of heuristics to cores, based on information obtained during a trial, to ensure that the mix of heuristics is appropriate for the instance being optimised.

The fact that CLS has comparable, and sometimes improved, performance over heuristics that are optimised to a particular instance type in addition to being effective over all the DIMACS and BHOSLIB instances clearly demonstrates the value of the underlying paradigm of combining variants of a basic local search heuristic. The overall performance of CLS on standard maximum clique instances reported here suggests that the underlying method has substantial potential to provide the basis for high-performance hyper-heuristics for other combinatorial optimisation problems, particularly weighted versions of maximum clique and conceptually related clustering problems.

Acknowledgements We would like to thank the anonymous reviewers for their valuable comments, which helped to improve the quality of this article.

References

Balus, E., Yu, C.: Finding a maximum clique in an arbitrary graph. SIAM J. Comput. **15**(4), 1054–1068 (1986)

Battiti, R., Protasi, M.: Reactive local search for the maximum clique problem. Algorithmica **29**, 610–637 (2001)

Bomze, I., Budinich, M., Pardalos, P., Pelillo, M.: The maximum clique problem. In: Du, D.Z., Pardalos, P. (eds.) Handbook of Combinatorial Optimization, vol. A, pp. 1–74. Kluwer Academic, Norwell (1999)

Boppana, R., Halldórsson, M.: Approximating maximum independent sets by excluding subgraphs. BIT **32**, 180–196 (1992)

Brockington, M., Culberson, J.: Camouflaging independent sets in quasi-random graphs. In: Johnson, D.S., Trick, M. (eds.) Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge. DIMACS Series, vol. 26. American Mathematical Society, Providence (1996)

Burke, E., Hart, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S.: Hyper-heuristics: An emerging direction in modern search technology. In: Glover, F. (ed.) Handbook of Meta-heuristics, pp. 457–474. Kluwer Academic, Norwell (2003)

Burns, G., Daoud, R., Vaigl, J.: LAM: An open cluster environment for MPI. In: Proceedings of Supercomputing Symposium, pp. 379–386. ACM, New York (1994)

Busygin, S.: A new trust region technique for the maximum weight clique problem. Discrete Appl. Math. **154**(15), 2080–2096 (2006)

Chakhlevitch, K., Cowling, P.: Hyper-heuristics: Recent developments. In: Cotta, C., Sevaux, M., Sörensen, K. (eds.): Adaptive and Multilevel Metaheuristics. Studies in Computational Intelligence, vol. 136, pp. 3–29. Springer, Berlin (2008)

Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness. Freeman, San Francisco (1979)

Grosso, A., Locatelli, M., Croce, F.D.: Combining swaps and node weights in an adaptive greedy approach for the maximum clique problem. J. Heuristics **10**, 135–152 (2004)

Grosso, A., Locatelli, M., Pullan, W.: Randomness, plateau search, penalties, restart rules: simple ingredients leading to very efficient heuristics for the maximum clique problem. J. Heuristics **14**, 587–612 (2008)

Hansen, P., Mladenović, N., Urosević, D.: Variable neighborhood search for the maximum clique. Discrete Appl. Math. **145**, 117–125 (2004)

Håstad, J.: Clique is hard to approximate within $n^{1-\varepsilon}$. Acta Math. **182**, 105–142 (1999)

Ji, Y., Xu, X., Stormo, G.D.: A graph theoretical approach for predicting common RNA secondary structure motifs including pseudoknots in unaligned sequences. Bioinformatics **20**(10), 1591–1602 (2004)

Johnson, D., Trick, M. (eds.): Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge. DIMACS Series, vol. 26. American Mathematical Society, Providence (1996)

Katayama, K., Hamamoto, A., Narihisa, H.: Solving the maximum clique problem by k-opt local search. In: Proceedings of the 2004 ACM Symposium on Applied Computing, pp. 1021–1025. ACM, New York (2004)

Marchiori, E.: Genetic, iterated and multistart local search for the maximum clique problem. In: Applications of Evolutionary Computing. Lecture Notes in Computer Science, vol. 2279, pp. 112–121. Springer, Berlin (2002)

Pevzner, P.A., Sze, S.H.: Combinatorial approaches to finding subtle signals in DNA sequences. In: Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology, pp. 269–278. AAAI Press, Menlo Park (2000)

Pullan, W.: Phased local search for the maximum clique problem. J. Combin. Optim. **12**, 303–323 (2006)

Pullan, W.: Approximating the maximum vertex/edge weighted clique using local search. J. Heuristics **14**, 117–134 (2008)

Pullan, W., Hoos, H.: Dynamic local search for the maximum clique problem. J. Artif. Intell. Res. **25**, 159–185 (2006)

Resende, M., Feo, T., Smith, S.: Algorithm 786: FORTRAN subroutine for approximate solution of the maximum independent set problem using GRASP. ACM Trans. Math. Softw. **24**, 386–394 (1998)

Squyres, J.M., Lumsdaine, A.: A component architecture for LAM/MPI. In: Proceedings, 10th European PVM/MPI Users' Group Meeting, Venice, Italy, September/October 2003. Lecture Notes in Computer Science, vol. 2840, pp. 379–387. Springer, Berlin (2003)

Wolpert, D., Macready, G.: No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **1**, 67–82 (1997)