A Stochastic Local Search Approach to Vertex Cover

Silvia Richter[1,3], Malte Helmert[2], and Charles Gretton[1]

[1] NICTA, 300 Adelaide St, Brisbane QLD 4000, Australia
{silvia.richter,charles.gretton}@nicta.com.au
[2] Albert-Ludwigs-Universität Freiburg, Georges-Köhler-Allee 052, 79110 Freiburg, Germany
helmert@informatik.uni-freiburg.de
[3] Institute for Integrated and Intelligent Systems, Griffith University, 170 Kessels Road, Nathan QLD 4111, Australia

Abstract. We introduce a novel stochastic local search algorithm for the vertex cover problem. Compared to current exhaustive search techniques, our algorithm achieves excellent performance on a suite of problems drawn from the field of biology. We also evaluate our performance on the commonly used DIMACS benchmarks for the related clique problem, finding that our approach is competitive with the current best stochastic local search algorithm for finding cliques. On three very large problem instances, our algorithm establishes new records in solution quality.

1 Introduction

Finding a *minimum vertex cover* of a graph is a well-known NP-hard problem [1]. Given an undirected graph $G = (V, E)$, a vertex cover is defined as a subset of the vertices $C \subseteq V$, such that every edge of G has an endpoint in C, i.e. for all $(u, v) \in E : u \in C$ or $v \in C$. The task then is to find a vertex cover of minimum size, or, for the corresponding NP-complete decision problem *k-vertex cover*, to decide whether a vertex cover of size k exists.

Applications of the vertex cover problem arise in network security, scheduling and VLSI design [2]. For example, finding a minimum vertex cover in a network corresponds to locating an optimal set of nodes on which to strategically place controllers such that they can monitor the data going through every link in the network. Algorithms for minimum vertex cover can also be used to solve the closely related problem of finding a *maximum clique*, which has a range of applications in biology, such as identifying related protein sequences [3].

Over the past decade, numerous algorithms have been proposed for solving the vertex cover problem, including evolutionary algorithms [4], ant colony system approaches [5] and complete search [6]. A recent approach of the latter kind that has proven useful in biological applications is the work of Abu-Khzam et al. [3].

In this work, we introduce a stochastic local search algorithm for vertex cover, dubbed COVER (*Cover Edges Randomly*), and show that it achieves excellent performance on a large variety of benchmarks. For the protein sequencing problems used by Abu-Khzam et al. [3], COVER is several orders of magnitude faster at finding the optimal solution than their approach. On a suite of "hard" benchmarks with hidden optimal solutions [7], COVER performs very well and establishes a new record on the largest instance. Furthermore, we compare the performance of COVER against state-of-the-art solvers for the related independent set and clique problems, showing that we achieve competitive results on a commonly used benchmark set. We also explore the importance of knowing the target solution size k for our algorithm.

The remainder of this article is organized as follows. We first give an overview of related work, including algorithms for the clique and independent set problems. We then describe the general idea of stochastic local search, and the specific details of our algorithm. In Sec. 5, we describe the empirical evaluation of our algorithm conducted on a wide range of benchmarks, after which we conclude.

2 Background & Related work

Various complete algorithms for k-vertex cover have been derived from the theory of fixed-parameter tractability (FPT) [6, 8, 9, 3]. The characterizing feature of FPT algorithms is that their run-time is bounded by $f(k) \cdot p(n)$, where the dependence $f(k)$ on the *parameter* k may be arbitrary, but the dependence $p(n)$ on the *graph size* n is polynomial. FPT algorithms generally work by first reducing the problem at hand in what is called a *kernelization* phase (transforming the problem to an equivalent problem with smaller parameter k), and then performing a bounded tree search on the remaining problem kernel. Recently, FPT algorithms have been implemented in a parallel fashion [9, 3].

Besides FPT methods, there are a number of heuristic approaches to the vertex cover problem. Evans describes an evolutionary approach, and also reviews previous evolutionary algorithms for the problem [4]. Ant colony systems have been employed by Shyu et al. [10] and Gilmour and Dras [5]. Looking beyond work on vertex cover, a wide range of heuristic algorithms have been proposed for related problems, which we review in the following.

2.1 Maximum Clique and Independent Set

A clique of a graph $G = (V, E)$ is a subset of the vertices $K \subseteq V$ such that all vertices in K are pairwise connected. An independent set of a graph $G = (V, E)$ is a subset of the vertices $S \subseteq V$ such that no two vertices in S are connected. A *maximum clique* (*maximum independent set*) is a clique (independent set) of maximum cardinality for a given graph.

Cliques and independent sets are closely related to vertex covers. In particular, a vertex set S is an independent set of G iff $V \setminus S$ is a vertex cover of G, and a vertex set K is a clique of G iff K is an independent set of the complementary graph $\overline{G}$, in which two vertices are connected iff they are unconnected in G. Hence, the problems of computing maximum cliques or maximum independent sets can be reduced to the computation of minimum vertex covers. In particular, all these problems are NP-hard, and the associated decision problems are NP-complete.

Many of the algorithms that have been proposed for computing maximum cliques or maximum independent sets in the past have been evaluated on the set of benchmark problems from the Second DIMACS Implementation Challenge in 1992–1993 [11]. The instances in this benchmark set are taken from a variety of application domains and also include examples that are specifically engineered to be “hard”. They can be considered the standard benchmark for algorithms that compute cliques or independent sets.

For the maximum independent set, a recently proposed heuristic approach is the *Widest Acyclic Orientation* algorithm by Barbosa and Campos, which is competitive with the algorithms used in the original DIMACS challenge [12]. Even better results are obtained with the recent QSH algorithm by Busygin, Batenko and Pardalos [13], which outperforms the maximum clique algorithms used during the DIMACS challenge. However, QSH fares badly on two classes of the DIMACS benchmarks.

For maximum clique, the recently published DLS-MC algorithm by Pullan and Hoos seems to deliver the best results, clearly dominating previously published algorithms on the DIMACS benchmark set [14]. DLS-MC stands for *Dynamic Local Search – Maximum Clique* and works by iteratively growing a candidate solution (initially one vertex) and conducting a plateau search where no further improvement is possible. When neither improvement nor new plateau steps are possible, search restarts from a single vertex. The DLS-MC algorithm has since evolved into the *Phased Local Search* algorithm [15], which eliminates the need for tuning a parameter while producing similar results.

3 Stochastic Local Search

Stochastic Local Search (SLS) methods are a popular means of solving notoriously hard combinatorial problems. They can be very effective while usually being conceptually simple [16]. For example, SLS algorithms are state of the art for solving Boolean Satisfiability problems [17]. In the following, we give a short description of SLS using the terminology of the textbook by Hoos and Stützle [16].

An SLS algorithm operates by searching in a space of *candidate solutions* for a problem π, where a candidate solution may not satisfy all of the constraints required by a solution.

Starting from an initial candidate solution, an SLS algorithm iteratively performs a small step to a *neighbouring* candidate solution by perturbing its current candidate solution. These steps, as well as the initialization of the search, may be randomized. The perturbation steps are only based on *local* information and on some *memory state* m of the algorithm (for example a taboo list). Pseudo code for a general SLS algorithm for a decision problem is shown in Alg. 1.

Algorithm 1 SLS(π)

```
1: initialize $(s, m)$
2: while not terminate($\pi, s, m$) do
3:     $(s, m) = $ step($\pi, s, m$)
4: end while
5: if $s \in S^*(\pi)$ then
6:     return $s$
7: else
8:     return failure
9: end if
```

In line 1 of Alg. 1, a candidate solution s and an initial *memory state* m of the algorithm are computed. In line 2, a termination criterion is used to determine whether the search should be terminated. In line 3, the step function replaces the current candidate solution s by a new candidate solution from the neighbourhood of s, while replacing the memory state m with a corresponding new memory state. If after terminating the search the candidate solution s is in the set $S^*(\pi)$ of solutions to π, then s is returned; otherwise, the algorithm fails.

In the following section, we describe the COVER algorithm as a specific instance of an SLS algorithm by elucidating the exact choices we made for the nature of candidate solutions as well as the initialization, termination and step functions.

4 The COVER algorithm

COVER is an SLS algorithm for k-vertex cover, i. e. it takes as input a graph $G = (V, E)$ and a parameter k, and searches for a vertex cover of size k of G. Its candidate solutions are subsets of the vertices V of size k (which are not necessarily vertex covers). The step to a neighbouring candidate solution consists of exchanging two vertices: a vertex u that is in the current candidate solution C is taken out of C, and a vertex v which is not currently in C is put into C.

The initial candidate solution is constructed greedily. In detail, COVER builds C by iteratively adding vertices that have a maximum number of incident edges which are not *covered* by C, i. e. they have no endpoint in C, until the cardinality of C is k. When several vertices satisfy the criterion for inclusion in C, COVER selects one of them randomly, with uniform probabilities. Favouring vertices of high degree is a common heuristic for vertex cover algorithms. In fact, we find that some benchmark problems are even solved by this initialization step alone, e. g. the *p-hat* class of the DIMACS benchmark set.

The termination criterion COVER uses is straightforward: at each step, it tests whether its current candidate solution is a vertex cover of G. The algorithm terminates when either a vertex cover is found, or when a maximum number of steps, denoted by MAX_ITERATIONS, has been reached.

The most influential part of an SLS algorithm is the definition of its step function. COVER uses several heuristic criteria to choose which two vertices to exchange in C, but also utilizes a substantial element of randomness, thus striking a balance between guided search and the diversity that is necessary to escape local optima. This balance is achieved with a simple division of responsibilities: the vertex to be taken out of C is chosen mainly according to heuristics, while the vertex to be put into C is chosen almost randomly.

When choosing possible candidates for inclusion in C, COVER selects uniformly at random an edge e that is not covered (following the strategy popularized by Selman and Kautz for SAT [18]). The vertex added to C is then chosen from one of the endpoints of e, ensuring that e will be covered in the successive candidate solution. When choosing which one of the two endpoints of e to include and which vertex to take out of the current candidate solution, COVER uses a heuristic based on an *edge weighting* scheme: with each edge of G, we associate a positive real number. Intuitively, these weights indicate for each edge how "difficult" it is to cover it – i. e. how difficult it is to find a candidate solution that contains one of the endpoints of that edge.

In the beginning, all edge weights are initialized to a small constant (0.05). In each of the following iterations, COVER adds 1 to the weights of all edges that are not covered. We then derive *vertex* weights from the weights of edges incident to the vertex. We say that a vertex v *potentially covers* an incident edge (v, u) if u is not in the current candidate solution C. Let the weight of a vertex v, $weight(v)$, be the weighted sum of all edges that vertex v potentially covers. An exchange of two vertices a and b, where a is taken out of C and b is put into C then results in a *gain* defined as $weight(b) - weight(a) + \delta$, where δ is the weight of the edge between a and b if they are connected, and zero otherwise. COVER tries to maximize this gain, thereby covering the more "difficult" edges of higher weight with greater priority. Using weights to direct the search of stochastic local search algorithms is popular practice, and a similar approach has led to very good results for the Boolean Satisfiability problem [19].

COVER also employs a taboo list of size 2, keeping track of the vertices last inserted into C and last removed from C. This prevents it from immediately reversing a decision made in the last iteration. Moreover, COVER remembers for each vertex the iteration count at which it was last removed from C. When inserting a vertex into C, COVER favours vertices that have not been in the cover recently. In particular, this "time stamp" criterion is used to break ties between all insertion candidates that result in maximum gain and are not taboo. When removing vertices from the candidate solution, COVER chooses randomly with uniform probability between all removal candidates.

Pseudo code for the algorithm is given in Alg. 2.

5 Empirical Performance Results

We evaluate the performance of COVER on an extensive set of benchmarks from three different sources. In Sec. 5.1, we use graphs representing biological real-world problems, which were kindly provided to us by Abu-Khzam et al. [3]. In Sec. 5.2, we report results on the BHOSLIB benchmark suite [7], a set of

Algorithm 2 COVER(G, k, MAX_ITERATIONS)

```
1:  initialize $C$ greedily with $|C| = k$
2:  initialize weights
3:  $iteration\_number = 1$
4:  while exists uncovered edge and $iteration\_number <$ MAX_ITERATIONS do
5:      choose uncovered edge $e = (u_1, u_2)$ randomly
6:      choose vertices $u \in \{u_1, u_2\}$ and $v \in C$ according to max gain criterion
7:      $C = C \backslash \{v\}$
8:      $C = C \cup \{u\}$
9:      $taboo\_list = \{v, u\}$
10:     $u.time\_stamp = iteration\_number$
11:     update weights
12:     increase $iteration\_number$ by 1
13: end while
```

graphs with "hidden optimal solutions" that are specifically designed to be hard to solve. In Sec. 5.3, we evaluate our performance on the (complement) graphs of the Second DIMACS Implementation Challenge for the maximum clique problem [11]. For each benchmark set, we compare against the best results we could find in literature. Unfortunately, we have to compare against a different system for each benchmark set, as we could not obtain the respective programs to run them on the other benchmark sets.

Our experiments were run on a machine with a 2.13 GHz CPU with 2 GB RAM. For comparing different algorithms on the DIMACS benchmark suite, three machine benchmarks are available from the DIMACS web site. When run on our machine, we obtained run-times of 0.51 seconds for r300.5, 3.01 seconds for r400. 5, and 11.31 seconds for r500. 5.

Due to the random elements of COVER, we ran the algorithm 100 times, with different random seeds, on each instance of each experiment described in this section. In all cases, the MAX_ITERATIONS parameter was set to 100,000,000. For each instance, we report the following information:

– *solution quality*: This is denoted as a triple *a*-*b*-*c*, where a is the number of runs (out of 100) in which the algorithm found a vertex cover with the minimal (or lowest known) cardinality k^*; b is the number of runs in which a vertex cover of size k^* was not found, but a vertex cover of size $k^* + 1$ was found; and c is the number of runs where only vertex covers of cardinality $k^* + 2$ or worse could be found.
– *median run-time* and *1st quartile run-time*: We ordered the outcomes of the 100 runs by the cardinality of the solution found (the lower, the better) and, in case of vertex covers of equal cardinality, by search time. *Median run-time* is the run-time for the median element in this sequence (i. e., the 50th-best result), and *1st quartile run-time* is the run-time for the element at the 1st quartile (i. e., the 25th-best result). Thus, median run-time is indicative of a "typical run" of the algorithm, and 1st quartile run-time is indicative of the typical performance one might obtain by running the algorithm repeatedly,

with four restarts. If the median (or 1st quartile) run did not achieve an optimal solution, the run-time result is reported in parentheses.

5.1 Biological Data

The graphs used in this section originate from phylogeny and correspond to protein sequencing data [3]. The task here is to find maximum sets of closely correlated protein sequences, which can be directly cast as a (weighted) maximum clique problem. Our input graphs are obtained from the biological problems as follows: first a weighted graph is constructed with vertices corresponding to protein sequences, and weighted edges between vertices corresponding to the extent of correlation between two sequences. Then, for a chosen threshold all edges with weights below the threshold are removed. The complement of this graph is the input for a vertex cover algorithm. The problems we use here were obtained from Abu-Khzam et al., who use them in their paper on parallel FPT algorithms for vertex cover [3].

Our run-time results, as well as the results obtained by the algorithm of Abu-Khzam et al. which we will denote by P-FPT (*parallel FPT*) in the following, are shown in Tab. 1. COVER found optimal solutions in all runs on all graphs except for the *globin-15* instance, where 98 of the 100 runs found optimal solutions, with the other two finding solutions of size $k^* + 1$ and $k^* + 3$, respectively.

Graph				COVER			P-FPT
				Quality	Runtime		
Instance	$\|V\|$	$\|E\|$	k^*	Hist.	1st Qu.	Median	Runtime
globin3	972	3898	165	100–0–0	0.01	0.01	23
globin7	972	38557	350	100–0–0	0.01	0.01	47
globin9	972	62525	378	100–0–0	0.01	0.01	227
globin15	972	149473	427	98–1–1	0.01	0.01	14
sh2-3	839	5860	246	100–0–0	0.01	0.01	22
sh2-4	839	13799	337	100–0–0	0.01	0.01	2593
sh2-5	839	26612	399	100–0–0	0.01	0.01	7
sh2-10	839	129697	547	100–0–0	0.01	0.01	332
sh3-10	2466	1508850	2044	100–0–0	0.47	0.80	8400

Table 1. Results on biological problems. $|V|$ and $|E|$ denote the number of vertices and edges of the input graph, k^* the minimum vertex cover size, as determined by Abu-Khzam et al. The performance metrics for COVER (solution quality, 1st quartile run-time, median run-time) are explained in detail at the start of this section. The last column denotes the run-time of the P-FPT algorithm. All run-times are in seconds.

The difference in run-time between the two approaches is compelling. A problem that took P-FPT more than two hours to solve is solved by COVER in less than one second. Of course, our local search algorithm may fail to find an optimal solution, while P-FPT searches exhaustively, guaranteeing optimality. However, as the results show, COVER reliably finds optimal vertex covers.

The approach used by Abu-Khzam et al. has reportedly delivered valuable results in collaboration projects with biologists: they report that, based on the cliques they derived from microarray data, "neurobiologists have identified what appear to be both network structures and gene roles in intra-cellular transport that were previously unrecognized". The vast difference in run-time between the two approaches prompts the question of what could be gained by using a local search algorithm like COVER in practice.

It is noteworthy that for many application domains of vertex cover, a near-optimal solution may be sufficient. In the protein sequencing domain, for example, choosing a certain correlation threshold has a somewhat arbitrary influence on the size of the greatest clique, which has more impact on the resulting solutions than the fact that the algorithm might be slightly suboptimal. Our algorithm is likely to provide solutions that are very close to the optimal, even for problems where the optimal solution is difficult to find. Hence, for solving large real-world problems, an incomplete algorithm like ours might prove to be more fruitful than a complete, but prohibitively slow algorithm.

5.2 The BHOSLIB problems

The BHOSLIB problems ("Benchmarks with Hidden Optimal Solutions") [7] result from translating binary Boolean Satisfiability problems that were generated randomly according to the model *RB* [20]. The satisfiability versions of these benchmarks are guaranteed to be satisfiable, and the model parameters were set to such values that the instances are in the phase transition area of model RB. They have been proven to be hard both theoretically and in practice [20]. The full BHOSLIB set of instances we use here is available on the Internet [7]. Some of these instances were also used in the 2004 SAT competition [21].

The problem instances are grouped by size into 8 groups, with 5 graphs per group, where all graphs of a group have the same number of vertices and edges. In addition to the 40 instances that form the actual benchmark suite, there is a single "challenge problem", a very large graph with 4,000 vertices and 572,774 edges. The minimum vertex cover for this instance has size 3,900.

The first two instances from each of the groups 3–8 (the *frb40*–*frb59* graphs) were used in the 2004 SAT competition. From the 6th group (*frb53*) onwards, none of the 55 solvers in the 2004 SAT competition was able to solve either of the two instances within a time limit of 10 minutes. For the very large instance, the best solution found up to now was a vertex cover of size 3,904, using a run-time of 3,743 seconds on a Pentium IV 3.4GHz/512MB machine [7]. This solution was obtained by translating the problem to a propositional logic formula extended with cardinality atoms, and using a dedicated solver [22].

The results obtained by COVER on these benchmarks are displayed in Tab. 2. The increasing difficulty of the instances is apparent both in the increasing run-times, as the graph sizes grow, and in the fact that COVER does not find optimal solutions consistently, i.e. in all runs. However, COVER does find optimal solutions for each graph at least once, and the sizes of the vertex covers it finds never exceed the minimum by more than 1.

For comparison, Gilmour and Dras recently developed a series of ant colony system algorithms for the vertex cover problem, evaluated on the BHOSLIB benchmarks [5]. They do not report run-times or solution results for individual

				COVER			CKACS	
Graph				Quality	Runtime		Quality	
Instance	$\lvert V\rvert$	$\lvert E\rvert$	k^*	Hist.	1st Qu.	Median	Hist.	Avg.
frb30-15-1	450	17827	420	100–0–0	0.06	0.08	0–1–9	424.0
frb30-15-2	450	17874	420	100–0–0	0.07	0.10	0–0–10	424.5
frb30-15-3	450	17809	420	100–0–0	0.21	0.40	0–0–10	424.6
frb30-15-4	450	17831	420	100–0–0	0.05	0.08	0–0–10	424.0
frb30-15-5	450	17794	420	100–0–0	0.12	0.17	0–0–10	423.6
frb35-17-1	595	27856	560	100–0–0	0.45	0.90	0–0–10	565.5
frb35-17-2	595	27847	560	100–0–0	0.40	0.84	0–0–10	566.5
frb35-17-3	595	27931	560	100–0–0	0.15	0.27	0–0–10	564.4
frb35-17-4	595	27842	560	100–0–0	0.62	1.12	0–0–10	565.5
frb35-17-5	595	28143	560	100–0–0	0.34	0.49	0–0–10	564.1
frb40-19-1	760	41314	720	100–0–0	0.33	0.62	0–0–10	725.6
frb40-19-2	760	41263	720	100–0–0	4.52	10.21	0–0–10	726.8
frb40-19-3	760	41095	720	100–0–0	1.37	3.17	0–0–10	727.6
frb40-19-4	760	41605	720	100–0–0	3.37	8.81	0–0–10	726.1
frb40-19-5	760	41619	720	96–4–0	21.80	63.47	0–0–10	725.3
frb45-21-1	945	59186	900	100–0–0	3.54	8.48	0–0–10	908.2
frb45-21-2	945	58624	900	100–0–0	11.67	28.46	0–0–10	908.5
frb45-21-3	945	58245	900	99–1–0	28.91	70.13	0–0–10	908.3
frb45-21-4	945	58549	900	100–0–0	4.90	12.28	0–0–10	908.4
frb45-21-5	945	58579	900	99–1–0	22.14	66.53	0–0–10	909.1
frb50-23-1	1150	80072	1100	89–11–0	58.32	171.92	0–0–10	1110.4
frb50-23-2	1150	80851	1100	30–70–0	543.56	(1.72)	0–0–10	1109.7
frb50-23-3	1150	81068	1100	24–76–0	(0.70)	(2.61)	0–0–10	1108.3
frb50-23-4	1150	80258	1100	100–0–0	8.45	16.94	0–0–10	1109.6
frb50-23-5	1150	80035	1100	98–2–0	24.43	88.94	0–0–10	1110.3
frb53-24-1	1272	94227	1219	9–91–0	(5.17)	(11.31)	0–0–10	1229.9
frb53-24-2	1272	94289	1219	34–66–0	403.98	(4.24)	0–0–10	1229.3
frb53-24-3	1272	94127	1219	91–9–0	65.21	157.80	0–0–10	1231.6
frb53-24-4	1272	94308	1219	24–76–0	(1.26)	(10.74)	0–0–10	1230.5
frb53-24-5	1272	94226	1219	84–16–0	109.36	253.05	0–0–10	1231.8
frb56-25-1	1400	109676	1344	15–85–0	(8.48)	(20.73)	0–0–10	1356.8
frb56-25-2	1400	109401	1344	12–88–0	(10.06)	(30.33)	0–0–10	1355.7
frb56-25-3	1400	109379	1344	76–24–0	130.11	435.30	0–0–10	1355.6
frb56-25-4	1400	110038	1344	84–16–0	85.60	291.11	0–0–10	1354.8
frb56-25-5	1400	109601	1344	98–2–0	30.45	89.58	0–0–10	1354.6
frb59-26-1	1534	126555	1475	11–89–0	(14.18)	(30.76)	0–0–10	1486.8
frb59-26-2	1534	126163	1475	6–94–0	(18.11)	(40.86)	0–0–10	1486.4
frb59-26-3	1534	126082	1475	12–88–0	(23.08)	(65.04)	0–0–10	1487.8
frb59-26-4	1534	127011	1475	1–99–0	(31.47)	(73.92)	0–0–10	1487.3
frb59-26-5	1534	125982	1475	89–11–0	90.18	292.60	0–0–10	1487.3

Table 2. Results on the BHOSLIB benchmark suite.

graphs, but only the *average* vertex cover size found over all BHOSLIB graphs. (We obtained the detailed results shown in Tab. 2 from personal communications.)

The best result they achieve, using the CKACS algorithm, is an average vertex cover size of 975.875, while 967.25 is the optimal value. This means that the vertex covers found by CKACS, on average, have 8.625 more vertices than an optimal solution. In comparison, COVER achieves an average vertex cover size of 967.50 on the BHOSLIB suite, i. e. the vertex covers it finds are only off by 0.25 on average.

On the challenge problem, COVER does not find an optimal solution. Indeed, the designer of the BHOSLIB benchmark set conjectures that this problem will not be solved on a PC in less than a day within the next two decades [7]. However, the COVER algorithm finds a solution of size 3,903 within 71 seconds, surpassing the best solution known so far in terms of both quality and run-time.

5.3 The DIMACS benchmark suite

The DIMACS benchmark set is taken from the Second DIMACS Implementation Challenge (1992-1993) [11], a competition targeting the maximum clique, graph colouring, and satisfiability problems. The maximum clique benchmarks from this competition have since been used in many publications as a reference point for new algorithms [4, 12–15]. The benchmark set comprises 80 problems from a variety of applications. For example, the *C-fat* family is motivated by fault diagnosis, the *johnson* and *hamming* graphs by coding theory, the *keller* group is based on Keller's conjecture on tilings using hypercubes, and the *MANN* graphs derive from the Steiner Triple Problem [11]. In addition, there are graphs generated randomly according to various models. For example, the *brock* family is generated by explicitly incorporating low-degree vertices into the cover, in order to defeat algorithms that search greedily with respect to vertex degrees [23]. The sizes of the graphs range from less than 30 vertices and ∼200 edges to more than 3000 vertices and ∼5,000,000 edges.

The results for COVER are shown in Tab. 3 and 4. For comparison, the tables also contain the results obtained by Pullan and Hoos with the DLS-MC algorithm [14]. DLS-MC was also run 100 times with the same limit on iterations as COVER. The times reported are the ones published by Pullan and Hoos [14], and refer to a 2. 2GHz Pentium IV machine with 512 MB RAM, which executed the DIMACS machine benchmarks r300. 5 (r400. 5, r500. 5) in 0.72 (4.47, 17.44) seconds. The run-times are thus roughly comparable, our machine being 30–35% faster according to this measure. The "avg." column shows the *mean* run-time for COVER across all runs where optimal solutions were found, for graphs where both algorithms found optimal solutions. This allows a direct comparison with the corresponding column for DLS-MC, taken from the article by Pullan and Hoos and determined by the same method. Note that this only compares the run-time for the cases *where an optimal solution was found*, and thus ignores runs where the found vertex cover was sub-optimal. Unfortunately, a direct comparison of our median run-time criterion (which we consider more indicative of actual performance because it is also influenced by sub-optimal runs) is not possible with the published results on DLS-MC.

In 75 of the 80 benchmarks, COVER finds a vertex cover of the putative minimum size for that instance. Note that it is only for some graphs of the *brock* family that COVER never finds optimal results. For the *brock* graphs, the cardinality of the vertex covers found by COVER can become as large as $k^* + 5$ in the worst case. This is not surprising, as COVER favours vertices of high degree, which generally is a helpful heuristic for finding minimum vertex covers. The *brock* graphs, however, were explicitly designed to counteract this approach.

Of the 75 instances where COVER finds an optimal solution, in 69 cases it does so consistently, i. e. in all 100 runs. For the remaining instances, there are occasional sub-optimal runs, but COVER always finds vertex covers of cardinality $k^* + 2$ or less. For *MANN_a81*, the putatively hardest problem in this benchmark set, COVER finds an optimal solution in 4 runs, is off by 1 in 3 runs, and off by 2 in the remaining 93 runs.

Graph				COVER				DLS-MC	
				Quality	Runtime			Quality	Runtime
Instance	$\|V\|$	$\|E\|$	k^*	Hist.	1st Qu.	Median	Avg.	Hist.	Avg.
brock200_1	200	5066	179	100–0–0	0.01	0.01	0.01	100–0–0	0.02
brock200_2	200	10024	188	100–0–0	0.15	0.23	0.43	100–0–0	0.02
brock200_3	200	7852	185	100–0–0	2.32	5.54	7.62	100–0–0	0.04
brock200_4	200	6811	183	100–0–0	2.04	6.52	7.90	100–0–0	0.05
brock400_1	400	20077	373	0–0–100	(0.04)	(0.06)	n/a	**100–0–0**	n/a
brock400_2	400	20014	371	0–1–99	(0.04)	(0.05)	n/a	**100–0–0**	n/a
brock400_3	400	20119	369	60–30–10	71.36	247.87	135.26	**100–0–0**	0.18
brock400_4	400	20035	367	76–18–6	44.16	137.68	112.98	**100–0–0**	0.07
brock800_1	800	112095	777	0–0–100	(0.77)	(1.06)	n/a	**100–0–0**	n/a
brock800_2	800	111434	776	0–0–100	(0.60)	(0.98)	n/a	**100–0–0**	n/a
brock800_3	800	112267	775	0–0–100	(1.43)	(2.31)	n/a	**100–0–0**	n/a
brock800_4	800	111957	774	0–0–100	(0.99)	(1.35)	n/a	**100–0–0**	n/a
C125.9	125	787	91	100–0–0	0.01	0.01	0.01	100–0–0	0.01
C250.9	250	3141	206	100–0–0	0.01	0.01	0.01	100–0–0	0.01
C500.9	500	12418	443	100–0–0	0.08	0.24	0.31	100–0–0	0.13
C1000.9	1000	49421	932	100–0–0	1.32	3.27	5.82	100–0–0	4.44
C2000.5	2000	999164	1984	100–0–0	0.82	1.84	3.78	100–0–0	0.97
C2000.9	2000	199468	1922	84–16–0	124.03	323.11	369.33	**93–7–0**	193.22
C4000.5	4000	3997732	3982	100–0–0	423.08	621.38	689.74	100–0–0	181.23
c-fat200-1	200	18366	188	100–0–0	0.01	0.01	0.01	100–0–0	0.01
c-fat200-2	200	16665	176	100–0–0	0.01	0.01	0.01	100–0–0	0.01
c-fat200-5	200	11427	142	100–0–0	0.01	0.01	0.01	100–0–0	0.01
c-fat500-1	500	120291	486	100–0–0	0.01	0.01	0.01	100–0–0	0.01
c-fat500-2	500	115611	474	100–0–0	0.01	0.01	0.01	100–0–0	0.01
c-fat500-5	500	101559	436	100–0–0	0.01	0.01	0.01	100–0–0	0.01
c-fat500-10	500	78123	374	100–0–0	0.01	0.01	0.01	100–0–0	0.01
DSJC500.5	500	62126	487	100–0–0	0.01	0.01	0.01	100–0–0	0.01
DSJC1000.5	1000	249674	985	100–0–0	0.28	0.95	2.17	100–0–0	0.80
gen200_p0.9_44	200	1990	156	100–0–0	0.01	0.01	0.01	100–0–0	0.01
gen200_p0.9_55	200	1990	145	100–0–0	0.01	0.01	0.01	100–0–0	0.01
gen400_p0.9_55	400	7980	345	100–0–0	0.04	0.06	0.08	100–0–0	0.03
gen400_p0.9_65	400	7980	335	100–0–0	0.01	0.01	0.01	100–0–0	0.01
gen400_p0.9_75	400	7980	325	100–0–0	0.01	0.01	0.01	100–0–0	0.01
hamming6-2	64	192	32	100–0–0	0.01	0.01	0.01	100–0–0	0.01
hamming6-4	64	1312	60	100–0–0	0.01	0.01	0.01	100–0–0	0.01
hamming8-2	256	1024	128	0–0–100	(0.01)	(0.01)	n/a	**100–0–0**	n/a
hamming8-4	256	11776	240	100–0–0	0.01	0.01	0.01	100–0–0	0.01
hamming10-2	1024	5120	512	100–0–0	0.01	0.01	0.01	100–0–0	0.01
hamming10-4	1024	89600	984	100–0–0	0.01	0.01	0.11	100–0–0	0.01
johnson8-2-4	28	168	24	100–0–0	0.01	0.01	0.01	100–0–0	0.01
johnson8-4-4	70	560	56	100–0–0	0.01	0.01	0.01	100–0–0	0.01
johnson16-2-4	120	1680	112	100–0–0	0.01	0.01	0.01	100–0–0	0.01
johnson32-2-4	496	14880	480	100–0–0	0.01	0.01	0.01	100–0–0	0.01
keller4	171	5100	160	100–0–0	0.01	0.01	0.01	100–0–0	0.01
keller5	776	74710	749	100–0–0	0.01	0.03	0.07	100–0–0	0.02
keller6	3361	1026582	3302	100–0–0	12.35	15.18	15.63	100–0–0	170.48
MANN_a9	45	72	29	100–0–0	0.01	0.01	0.01	100–0–0	0.01
MANN_a27	378	702	252	100–0–0	0.01	0.01	0.01	100–0–0	0.05
MANN_a45	1035	1980	690	**41–59–0**	246.92	(0.28)	n/a	0–100–0	n/a
MANN_a81	3321	6480	2221	**4–3–93**	(3.36)	(30.89)	n/a	0–0–100	n/a
p_hat300-1	300	33917	292	100–0–0	0.01	0.01	0.01	100–0–0	0.01
p_hat300-2	300	22922	275	100–0–0	0.01	0.01	0.01	100–0–0	0.01
p_hat300-3	300	11460	264	100–0–0	0.01	0.01	0.01	100–0–0	0.01
p_hat500-1	500	93181	491	100–0–0	0.01	0.01	0.01	100–0–0	0.01
p_hat500-2	500	61804	464	100–0–0	0.01	0.01	0.01	100–0–0	0.01
p_hat500-3	500	30950	450	100–0–0	0.01	0.02	0.02	100–0–0	0.01
p_hat700-1	700	183651	689	100–0–0	0.01	0.01	0.04	100–0–0	0.02
p_hat700-2	700	122922	656	100–0–0	0.01	0.02	0.01	100–0–0	0.01
p_hat700-3	700	61640	638	100–0–0	0.01	0.01	0.01	100–0–0	0.01
p_hat1000-1	1000	377247	990	100–0–0	0.01	0.01	0.01	100–0–0	0.01
p_hat1000-2	1000	254701	954	100–0–0	0.01	0.04	0.04	100–0–0	0.01

Table 3. Results on the DIMACS benchmark suite (continued in Tab. 4).

Graph				COVER				DLS-MC	
				Quality	Runtime			Quality	Runtime
Instance	$\|V\|$	$\|E\|$	k^*	Hist.	1st Qu.	Median	Avg.	Hist.	Avg.
p_hat1000-3	1000	127754	932	100–0–0	0.06	0.10	0.11	100–0–0	0.01
p_hat1500-1	1500	839327	1488	100–0–0	13.80	18.25	21.27	100–0–0	2.71
p_hat1500-2	1500	555290	1435	100–0–0	0.09	0.12	0.12	100–0–0	0.01
p_hat1500-3	1500	277006	1406	100–0–0	0.07	0.10	0.11	100–0–0	0.01
san200_0.7_1	200	5970	170	100–0–0	0.01	0.01	0.01	100–0–0	0.01
san200_0.7_2	200	5970	182	100–0–0	0.01	0.01	0.01	100–0–0	0.07
san200_0.9_1	200	1990	130	100–0–0	0.01	0.01	0.01	100–0–0	0.01
san200_0.9_2	200	1990	140	100–0–0	0.01	0.01	0.01	100–0–0	0.01
san200_0.9_3	200	1990	156	100–0–0	0.01	0.01	0.01	100–0–0	0.01
san400_0.5_1	400	39900	387	100–0–0	0.05	0.14	0.12	100–0–0	0.16
san400_0.7_1	400	23940	360	100–0–0	0.05	0.06	0.06	100–0–0	0.11
san400_0.7_2	400	23940	370	100–0–0	0.06	0.07	0.08	100–0–0	0.21
san400_0.7_3	400	23940	378	100–0–0	0.08	0.12	0.13	100–0–0	0.42
san400_0.9_1	400	7980	300	100–0–0	0.01	0.01	0.01	100–0–0	0.01
san1000	1000	249000	985	100–0–0	0.98	3.88	3.91	100–0–0	8.36
sanr200_0.7	200	6032	182	100–0–0	0.01	0.01	0.01	100–0–0	0.01
sanr200_0.9	200	2037	158	100–0–0	0.01	0.01	0.01	100–0–0	0.01
sanr400_0.5	400	39816	387	100–0–0	0.01	0.02	0.06	100–0–0	0.04
sanr400_0.7	400	23931	379	100–0–0	0.01	0.01	0.03	100–0–0	0.02

Table 4. Results on the DIMACS benchmark suite (continued from Tab. 3).

Comparing against the results of the DLS-MC algorithm, we find that the two algorithms are largely competitive. In fact, many of the benchmarks seem to be too easy for a state-of-the-art solver nowadays. On the graph families *c-fat*, *DSJC*, *gen*, *hamming*, *johnson*, *p_hat*, *san* and *sanr*, both DLS-MC and COVER consistently find optimal solutions within extremely short run-times. However, on *MANN_a45* and *MANN_a81*, COVER significantly outperforms the DLS-MC algorithm. On these graphs, DLS-MC does not find an optimal solution. On the hard *MANN_a81* instance, DLS-MC indeed only finds solutions that are of distance 2 or more from the optimum, while COVER finds optimal solutions for both graphs. In fact, to our knowledge COVER is the first algorithm to find covers of this quality for the two *MANN* graphs.

On the other hand, on the *brock* family DLS-MC shows far better results than COVER. This can be explained by the fact that DLS-MC uses a parameter called *penalty-delay*, which Pullan and Hoos hand-tuned for each graph to achieve the best possible performance. While for almost all other graphs this parameter was set to a value between 1 and 5, it was set to 15 and 45 for the larger *brock* graphs, encouraging DLS-MC to quite drastically change its usual behaviour in these cases [14]. We conclude that, despite the fact that COVER is designed for vertex cover problems and DLS-MC is designed for clique problems, COVER is competitive with DLS-MC on clique benchmarks. COVER furthermore has the advantage of requiring no parameters, while achieving excellent results but for one special class of artificial graphs.

5.4 Search without Parameter

To further understand the run-time complexity of COVER, we conduct a set of experiments aimed at determining the importance of knowing k, the target cover size. Most state-of-the-art solvers, including the ones we compared against

Graph	Run-time(s)	k^*	k^*+1	k^*+2	k^*+3	$> k^*+3$
Globin7	0.54	48.15%	48.15%	3.70%		
Sh2-5	0.70	37.14%	37.14%	25.71%		
johnson32-2-4	0.92	28.26%	28.26%	26.09%	11.96%	5.43%
brock200_1	1.52	21.71%	17.11%	17.11%	16.45%	27.63%
p_hat700-1	2.16	43.06%	24.54%	11.57%	10.65%	10.19%
keller5	5.12	40.43%	7.23%	5.08%	5.08%	42.19%
frb30-15-1	7.15	41.68%	24.20%	7.27%	3.64%	23.22%
hamming10-4	10.73	47.62%	25.63%	2.42%	2.42%	21.90%
san400_0.5_1	15.30	34.77%	22.35%	15.82%	19.28%	7.78%
DSJC1000.5	88.10	97.63%	1.16%	0.30%	0.30%	0.62%
brock200_3	166.01	99.18%	0.19%	0.16%	0.16%	0.32%
san1000	645.26	28.25%	20.13%	18.73%	18.72%	14.16%
frb50-23-4	1344.88	85.85%	9.78%	2.42%	1.42%	0.53%
frb59-26-5	17611.90	84.57%	13.84%	0.89%	0.35%	0.35%

Table 5. Run-time distribution for various parameters.

in this paper, are, like COVER, solving the k-vertex cover problem (or k-clique, respectively). In practice, however, we do not usually know the optimal value for k. Instead, we want to find a minimum (or close to minimum) vertex cover of unknown size.

The question thus arises whether COVER can be used efficiently for finding a minimum vertex cover by iteratively searching for various decreasing values of k. Specifically, we are interested in determining how much run-time is spent searching for several values of k as opposed to just searching with a known optimal value k^*. We expect that it is much easier to find solutions that are suboptimal than ones that are optimal, and that indeed only the last few runs where k is close to k^* substantially influence run-time.

To test this hypothesis, we extend COVER to an iterative version COVER-I, which runs without a parameter k, as follows. First, COVER-I greedily computes a vertex cover for the input graph. This is done much in the same way as COVER computes an initial candidate solution. Instead of stopping when the size of the candidate solution reaches a prespecified parameter, however, COVER-I keeps adding vertices until the candidate solution is indeed a vertex cover. The size k of this vertex cover is thus an upper bound for the optimal value k^*. COVER-I then iteratively calls COVER as a subroutine, decreasing k each time COVER succeeds in finding a solution within the usual limit of 100,000,000 iterations. When COVER fails to find a solution, COVER-I stops and returns the last solution found.

For our experiment, we select a representative set of graphs containing instances from all three benchmark suites in varying sizes. The results are displayed in Tab. 5. The *run-time* column shows total run-time for COVER-I for the given graphs, summed up for 25 different random seeds, to give an impression of the relative difficulty of these instances. The column k^* shows the percentage of total run-time spent on the final iteration (producing the optimal solution), with columns k^*+1, k^*+2 etc. referring to the previous iterations.

The results largely confirm our expectations. For small graphs, where the search times for the optimal value k^* are already short, run-time is sometimes spread out fairly evenly across iterations; but for the larger graphs, the amount of time spent in the ultimate iteration dominates the total run-time of COVER-I.

6 Conclusion & Outlook

We have presented a stochastic local search algorithm for the vertex cover problem, COVER, and evaluated its performance on a wide variety of benchmarks. COVER is surprisingly effective while being conceptually simple and not requiring any instance-dependent parameters. For biological real-world problems, COVER finds optimal solutions in just a fraction of the time needed by a complete search, which leads us to believe that COVER is a valuable approach for practical problems. On the hard BHOSLIB benchmark set, COVER vastly improves on existing results and sets a new record for the 20-year challenge problem.

Compared to the state-of-the-art solver DLS-MC for the maximum clique problem, COVER shows competitive results on the DIMACS suite. We emphasize the fact that unlike DLS-MC and many algorithms proposed in the literature previously, COVER has not been tuned in any way to the benchmark sets we evaluated it on. The excellent performance of COVER is further underlined by the fact that it sets a new record in solution quality on two large benchmark instances of the DIMACS set. However, COVER did not perform well on the *brock* family of graphs from the DIMACS test set. An obvious direction of future work is therefore to develop further techniques to more reliably escape local minima during search.

Acknowledgements

NICTA is funded by the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council. We thank Abdul Sattar for his input on the direction of this project and Duc Nghia Pham for helpful discussions. We also thank Stephen Gilmour, Wayne Pullan and Michael Langston for providing problem graphs, results, and helpful suggestions.

References

1. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Company (1979)
2. Gomes, F.C., Meneses, C.N., Pardalos, P.M., Viana, G.V.R.: Experimental analysis of approximation algorithms for the vertex cover and set covering problems. Computers and Operations Research **33**(12) (2006) 3520–3534
3. Abu-Khzam, F.N., Langston, M.A., Shanbhag, P., Symons, C.T.: Scalable parallel algorithms for FPT problems. Algorithmica **45** (2006) 269–284
4. Evans, I.K.: Evolutionary algorithms for vertex cover. In: Evolutionary Programming VII, Proceedings of the Seventh International Conference on Evolutionary Programming (EP98). (1998) 377–386
5. Gilmour, S., Dras, M.: Kernelization as heuristic structure for the vertex cover problem. In: Proceedings of the Third Workshop on Ant Colony Optimization and Swarm Intelligence (ANTS 2006), Brussels, Belgium. (2006)

6. Downey, R.G., Fellows, M.R., Stege, U.: Parameterized complexity: A framework for systematically confronting computational intractability. In: Contemporary Trends in Discrete Mathematics: From DIMACS and DIMATIA to the Future, Vol. 49 of DIMACS Series. (1999) 49–99
7. Xu, K.: BHOSLIB: Benchmarks with hidden optimum solutions for graph problems (maximum clique, maximum independent set, minimum vertex cover and vertex coloring) – hiding exact solutions in random graphs. Web site, http://www.nlsde.buaa.edu.cn/∼kexu/benchmarks/graph-benchmarks.htm
8. Niedermeier, R., Rossmanith, P.: Upper bounds for vertex cover further improved. In: Proceedings of the 16th Symposium on Theoretical Aspects in Computer Science (STACS'99). (1999) 561–570
9. Cheetham, J., Dehne, F., Rau-Chaplin, A., Stege, U., Taillon, P.J.: Solving large FPT problems on coarse grained parallel machines. Journal of Computer and System Sciences **67** (2003) 691–706
10. Shyu, S.J., Yin, P.Y., Lin, B.M.T.: An ant colony optimization algorithm for the minimum weight vertex cover problem. Annals of Operations Research **131**(1–4) (2004) 283–304
11. Johnson, D.S., Trick, M.A., eds.: Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge. Volume 26 of DIMACS Series. American Mathematical Society (1996)
12. Barbosa, V.C., Campos, L.C.D.: A novel evolutionary formulation of the maximum independent set problem. Journal of Combinatorial Optimization **8** (2004) 419–437
13. Busygin, S., Butenko, S., Pardalos, P.M.: A heuristic for the maximum independent set problem based on optimization of a quadratic over a sphere. Journal of Combinatorial Optimization **6** (2002) 287–297
14. Pullan, W., Hoos, H.H.: Dynamic local search for the maximum clique problem. Journal of Artificial Intelligence Research **25** (2006) 159–185
15. Pullan, W.: Phased local search for the maximum clique problem. Journal of Combinatorial Optimization **12** (2006) 303–323
16. Hoos, H.H., Stützle, T.: Stochastic Local Search: Foundations and Applications. Morgan Kaufmann (2004)
17. Kullmann, O.: The SAT 2005 solver competition on random instances. Journal on Satisfiability, Boolean Modeling and Computation **2** (2006) 61–102
18. Selman, B., Kautz, H.A.: Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In: Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93). (1993) 290–295
19. Thornton, J., Pham, D.N., Bain, S., Ferreira Jr., V.: Additive versus multiplicative clause weighting for SAT. In: Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI 2004). (2004) 191–196
20. Xu, K., Boussemart, F., Hemery, F., Lecoutre, C.: A simple model to generate hard satisfiable instances. In: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05). (2005) 337–342
21. Le Berre, D., Simon, L.: The SAT'04 competition. Web site, http://www.lri.fr/∼simon/contest04/results/
22. Liu, L., Truszczynski, M.: Local-search techniques for propositional logic extended with cardinality constraints. In: Principles and Practice of Constraint Programming, Proceedings of the 9th International Conference on Constraint Programming (CP 2003). (2003) 495–509
23. Brockington, M., Culberson, J.C.: Camouflaging independent sets in quasi-random graphs. [11] 75–88