

Exactness as Heuristic Structure for Guiding Ant Colony System

Stephen Gilmour and Mark Dras

Department of Computing, Macquarie University,
North Ryde, NSW, Australia 2109
{gilmour, madras}@ics.mq.edu.au

Abstract. For solving combinatorial optimisation problems, exact methods accurately exploit the structure of the problem but are tractable only up to a certain size; approximation or heuristic methods are tractable for very large problems but may possibly be led into a bad solution. A question that arises is, From where can we obtain knowledge of the problem structure via exact methods that can be used on large-scale problems by heuristic methods? We present a framework that allows the exploitation of existing techniques and resources to integrate such structural knowledge into the Ant Colony System metaheuristic, where the structure is determined through the notion of kernelization from the field of parameterized complexity. We give experimental results using vertex cover as the problem instance, and show that knowledge of this type of structure improves performance beyond previously defined ACS algorithms.

1 Introduction

For solving combinatorial optimisation problems, exact methods accurately exploit the structure of the problem but are tractable only up to a certain size; approximation or heuristic methods are tractable for very large problems but may possibly be led into a bad solution. A third approach could be to combine heuristics and exact methods, which would hopefully still run quickly but the quality of solution would be improved over just regular heuristics. Some examples of combining heuristics with exact methods are discussed in [2]. In the work discussed in this paper, we investigate the use of an already well established body of techniques from the field of parameterized complexity [6] for identifying problem structure as part of an exact solution, and the extent to which these techniques can be integrated into heuristics. There are a number of ways such an integration could be defined. For example, we could just use search trees on the problem until a certain point, or we could incorporate the techniques more centrally into the heuristic.

In this paper we compare a number of different approaches to combining the notion of ‘kernelization’ from parameterized complexity with Ant Colony System (ACS) [5, 3]. The paper builds on the work of two papers before it: the first paper [7] is a proposal for our framework, and discusses a part of the framework which is now complete in this paper and gives some preliminary results for our research; the second paper [8] gives an ACS algorithm for the vertex cover problem that is a

modification of that of [9] that we use as a baseline. In section 2 of this paper, we discuss ACS for the vertex cover problem. In section 3 we discuss parameterized complexity and kernelization. In section 4 we discuss the kernelization rule we have chosen to use within our algorithms. In section 5 we present our framework for integrating ACS with kernelization. In section 6 we give experimental results for our new algorithms and in section 7 we conclude.

2 ACS for the Minimum Vertex Cover Problem

In this section we will present an algorithm by Gilmour and Dras [8] for ant colony system on the vertex cover problem (VCP). Given the graph $G = (V, E)$, the minimum VCP is the problem of finding a set of nodes $V' \subseteq V$ such that every edge E is covered by a node from V' and such that the number of nodes in V' is minimised. Our ant colony system algorithm is based upon the ACS algorithm for the TSP [5] and the algorithm by Shyu, Yin, & Lin [9] for the weighted vertex cover problem. In section 2.1 we discuss the problem representation as defined by Shyu *et al.* [9] for ACS on the vertex cover problem. In sections 2.2 and 2.3 we present our adapted random proportional transition rule and pheromone update rules. In section 2.4 we discuss the parameters for this algorithm.

2.1 Problem Representation

Shyu *et al.* note that, unlike the Travelling Salesman Problem (TSP) for which the first ACS was designed, a VCP solution does not constitute a path in the graph. Thus in order to allow our algorithm to find unordered subsets of nodes, we construct a complete graph $G_c = (V, E_c)$. This will “guarantee that there always exists a path in G , a sequence of unrepeatd adjacent vertices, which covers exactly and only the vertices in V' ” [9]. But we want to solve the minimum vertex cover problem for G and not G_c and therefore we need to preserve the details of the original graph within this new representation. Therefore we define for each ant k a binary connectivity function $\psi_k : E_c \rightarrow \{0, 1\}$ as

$$\psi_k(i, j) = \begin{cases} 1 & \text{if edge } (i, j) \in E; \\ 0 & \text{if edge } (i, j) \in E_c - E, \end{cases} \quad (1)$$

This allows ants to construct solutions of unordered nodes of the original graph by allowing ants to move from any node to any other node yet preserving the details of the original graph within the representation.

2.2 Random Proportional Transition Rule

Our rule for deciding which node j ant k should place in its vertex cover construction next is:

$$j = \begin{cases} \arg \max_{u \in J^k} \{[\tau_u(t)] \cdot [\eta_u]^\beta\} & \text{if } q \leq q_0; \\ J & \text{if } q > q_0, \end{cases} \quad (2)$$

where q is randomly selected from the distribution $[0, 1]$; q_0 is a tunable parameter such that $0 \leq q_0 \leq 1$; $\tau_u(t)$ is the amount of pheromone on node u ; $\eta_u = \sum_{z \in N(u)} \psi_k(u, z)$ is the heuristic goodness of node u ; J^k is the set of nodes that ant k may still visit; and $J \in J^k$ is a node that is randomly selected according to the probability:

$$p_{J^k}(t) = \frac{[\tau_J(t)] \cdot [\eta_J]^\beta}{\sum_{l \in J^k} [\tau_l(t)] \cdot [\eta_l]^\beta} \quad (3)$$

After an ant visits a node which it has just placed in its candidate solution, it sets $\psi_k(i, j) = 0$ for every edge (i, j) connected to the node j that it has just visited. This allows the ant to keep track of which edges have been covered and which edges are still to be covered. Ants are interested in constructing the smallest vertex cover possible and therefore continually place nodes into their solution until every edge is covered (that is, $\psi_k = 0$ for all edges within the graph). However, before the next cycle can continue, the connectivity values need to be reset for all edges according to equation (1).

2.3 Pheromone System

On the travelling salesman problem, pheromone is placed on edges since every node is visited within a solution. However, for the vertex cover problem, pheromone is placed on nodes since we are interested in constructing an unordered subset of all the nodes within the graph. Therefore, at the very least, the global and local pheromone update rules need to be updated to work with nodes rather than edges. Our global pheromone update rule, which is executed by one ant at the end of each cycle, is defined as:

$$\tau_i(t) \leftarrow (1 - \rho) \cdot \tau_i(t) + \rho \cdot \Delta\tau_i(t) \quad (4)$$

where i are the nodes belonging to the current best solution T^+ ; $\Delta\tau_i(t) = 1/L^+$ such that L^+ is the size of the solution T^+ ; and ρ is a parameter governing pheromone decay such that $0 < \rho < 1$.

Similarly, our local pheromone update rule, which is executed by every ant on each node as it is placed in the candidate solution, is defined as:

$$\tau_i = (1 - \varphi)\tau_i + \varphi\tau_0 \quad (5)$$

where $\varphi \in (0, 1)$ is a parameter which simulates the evaporation rate of pheromone; and τ_0 is the amount of pheromone every edge is initially set to before this

algorithm starts. We show in [8] that this gives better results than the local pheromone update rule of [9].

Similar to the TSP, our formulation for τ_0 is $\tau_0 = \frac{1}{(n \cdot L_{nn})}$ where L_{nn} is the size of a solution produced by a simple greedy heuristic.

2.4 Parameter Investigation

We generated a set of 160 graphs with number of nodes ranging from 100 to 800 and number of edges ranging from 150 to 4000. We ran ant colony system for each of the 160 graphs for 200 iterations. The optimal parameters for this sample were: number of ants $m = 50$; influence of heuristic information $\beta = 4$; pheromone trail evaporation $\rho = 0.1$; and probability of including the best choice of tour construction $q_0 = 0.9$. By way of comparison, the ideal parameters for the algorithm by Shyu *et al.* were: $m = 10$; $\beta = 1$; $\rho = 0.9$; and $q_0 = 0.9$.

3 Parameterized Complexity

An overview of parameterized complexity is available in [6]; we give a brief outline here. In section 2 we defined what we will now call the general minimum vertex cover problem. A related problem is the k -vertex cover problem: given a graph $G = (V, E)$ and a parameter k , this is the problem of finding a vertex cover of size less than or equal to k . This problem is still NP-complete. A key idea of parameterized complexity is that if some value of a problem is known to be bounded in a certain context, useful algorithms with good complexity properties can be developed. For the k -vertex cover problem, k might be bounded within a particular domain of application; in such a case, there is a best-case algorithm with complexity $O(1.271^k k^2 + kn)$ [4] that is tractable for k up to 400. Here k represents the maximum size of the vertex cover that we are trying to find. Parameterized complexity also allows a more fine-grained analysis of problems classified as NP-complete: some are amenable to this treatment, while others have only brute-force solutions of complexity $O(n^{k+1})$. Those that are amenable to this approach are called fixed-parameter tractable (FPT).

Parameterized complexity contains both a framework of complexity analysis and a corresponding toolkit of algorithm design. One such tool for algorithm design is kernelization. The idea behind kernelization is reducing a problem to its problem kernel; kernelization is usually a form of pre-processing and always completes in polynomial time for FPT algorithms. Ideally the problem kernel after kernelization is small enough that even a brute-force attack is an option; however, usually an approach such as bounded search trees is necessary for difficult problems. Kernelization is the core idea behind the successful algorithms for the vertex cover problem. Although kernelization makes a problem easier to process by reducing it to its problem kernel, it doesn't lose any information necessary to finding an optimal solution.

Many different optimization problems have been analyzed using the notions of parameterized complexity, for example the dominating set problem, travelling salesman problem, and the 3-CNF satisfiability problem, often with several proposed algorithms for each problem. There is thus a wide range of tools available to be used. The aim of this paper is to see whether, and in what way, these can be combined with ACO as a kind of template in the context of the vertex cover problem.

4 Our Kernelization Rules

From all the varieties of exact algorithms described that employ kernelization rules, we initially chose four rules to use as the kernelization rules within our research on the vertex cover problem [7]. We chose these four rules because “in order to integrate kernelization into Ant Colony Optimization, we need to use rules that can be implemented in a distributed system. Therefore, we have identified four rules from different algorithms that all lend themselves to a multi-agent implementation and can validly work together to kernelize a graph into its problem kernel” [7]. The four rules, taken from [11], are:

1. If G has a vertex v of degree greater than k , then replace (G, k) with $(G - v, k - 1)$ and place v in the vertex cover;
2. If G has adjacent vertices u and v such that $N(v) \subseteq N[u]$, then replace (G, k) with $(G - u, k - 1)$ and place u in the vertex cover;
3. If G has a pendant edge uv with u having degree 1, then replace (G, k) with $(G - \{u, v\}, k - 1)$ and place v in the vertex cover;
4. If G has a vertex u of degree 2, with neighbours y and z , and y and z are adjacent, then replace (G, k) with $(G - \{u, y, z\}, k - 2)$ and place y and z in the vertex cover.

However, we have observed that these four rules can be reduced in form to just two rules (rules 1 and 2). The reason for this reduction is that the other two rules can be reduced to the second rule. Take rule three from our original rules list. Rule three is an instance of our second rule where the node v only has one neighbour, u . Therefore we can remove rule three from our rules list since it has been performed already for us by rule two. Similarly, rule four is an instance of our second rule where the node v only has two neighbours, u and z . All the neighbours of v are also neighbours of both u and z and therefore this structure is also solved by rule two. This leaves us with just rules one and two.

However, one more reduction can be made. Our Ant Colony System algorithm is for solving the minimum vertex cover problem and not the k -vertex cover problem; that is the problem of finding a vertex cover of size k . Therefore, we do not have a parameter k to use for rule one. Further, Ant Colony System uses a basic greedy heuristic as its basis; greedy algorithms pick the locally best option at each decision. This makes rule one redundant since its effect is already achieved within Ant Colony System itself.

5 Ant Colony System with Structure

Parameterized complexity has proved useful for the vertex cover problem with these FPT algorithms. However, the current best is only useful for problems with solutions of size up to 400. Therefore for the vertex cover problem there is still reason to use heuristics, not to mention for problems not in the FPT class.

The key idea in this paper is that kernelization can be used to give heuristic methods—here ACO—information about the problem. We will present six variant algorithms for combining these. Within the algorithms that we propose, we will be utilizing just one kernelization rule discussed in section 4.

5.1 Kernelized Ant Colony System

Kernelized ACS performs kernelization within the initialisation stage of the algorithm and then runs standard ACS on the resulting kernel graph. We define a set χ that contains all the nodes that are identified by kernelization as belonging to an exact solution. From the graph we remove all the nodes that belong to χ and all edges connected at either end to nodes in χ . We run standard ACS on the resultant graph.

5.2 PreKernelized Ant Colony System

PreKernelized ACS works similarly to Kernelized ACS except rather than removing nodes from the graph, it sets the pheromone on the selected nodes to be kt_0 . This parameter was picked such that unless the solution to the problem being solved consists of just one node, we can be certain that the pheromone levels on all the other nodes that we are trying to distinguish the kernelization set from contain significantly less pheromone. This will make the nodes in the kernelization set far more attractive to ants than any other nodes in the problem. PreKernelized ACS performs this kernelization in the initialisation phase of ACS.

5.3 CycleKernelized Ant Colony System

CycleKernelized ACS is similar to PreKernelized ACS except that the kernelization information is continually reinforced in pheromone. Within CycleKernelized ACS, we have moved the kernelization component out of the initialisation phase and into the global pheromone update rule. That is, as well as placing pheromone on the current best solution, the ant selected to perform the global pheromone update rule also reinforces the pheromone on the nodes in χ . Formally the new global pheromone update rule can be stated as:

$$\tau_i(t) \leftarrow \begin{cases} kt_0, & \text{if } i \in \chi; \\ (1 - \rho) \cdot \tau_i(t) + \rho \cdot \Delta\tau_i(t), & \text{if } i \in T^+ - \chi. \end{cases} \quad (6)$$

5.4 KernelAnts Ant Colony System

ACS is a population-based metaheuristic that uses m ants to generate solutions to a problem. However, KernelAnts ACS uses k additional ants (called kernelants) to kernelize the graph and place pheromone on the nodes identified through kernelization whilst the original m ants continue to perform regular ant colony system.

Each turn, these kernelants set the pheromone on one node each, identified through kernelization rules, to kt_0 . This occurs in parallel with the regular ants continuing to construct solutions to the problem and execute the local and global pheromone update rules. However, they should be strongly influenced in picking nodes to place in solutions by the pheromone laid out by the kernelants.

This algorithm along with CycleKernelized ACS and PreKernelized ACS are useful because there are many applications where the problem specification can not be altered and therefore these algorithms are interesting to see whether they can perform as well as Kernelized ACS.

5.5 TransKernelized Ant Colony System

Within this algorithm, we have incorporated the kernelization into the ants' random proportional transition rule. When an ant draws a random number between zero and one that is less than the threshold q_0 , the ant will first look in the kernelization set χ to see if there is a node to visit before picking the best of all possible options. Should the random number be above the threshold, the ant assigns to each potential node a probability and decides where to go next probabilistically, as with standard ACS.

Our new random proportional transition rule used within TransKernelized ACS for deciding which node j ant k should place in its vertex cover construction next is:

$$j = \begin{cases} \text{any node } u \text{ such that } u \in (J^k \cap \chi) & \text{if } q \leq q_0 \text{ and } |J^k \cap \chi| > 0; \\ \arg \max_{u \in J^k} \{[\tau_u(t)] \cdot [\eta_u]^\beta\} & \text{if } q \leq q_0 \text{ and } |J^k \cap \chi| = 0; \\ J & \text{if } q > q_0, \end{cases} \quad (7)$$

where q is randomly selected from the distribution $[0, 1]$; q_0 is a tunable parameter such that $0 \leq q_0 \leq 1$; $\tau_u(t)$ is the amount of pheromone on node u ; $\eta_u = \sum_{z \in N(u)} \psi_k(u, z)$ is the heuristic goodness of node u ; J^k is the set of nodes that ant k may still visit; and $J \in J^k$ is a node that is randomly selected probabilistically according to equation (3).

5.6 Neighbourhood TransKernelized Ant Colony System

One problem with TransKernelized ACS is that it can involve a lot of kernelization on the fly. Neighbourhood TransKernelized ACS is an alternative algorithm

	KACS	PKACS	CKACS	KAACS	TKACS	NTKACS
m	40	40	40	40	10	40
k	-	-	-	10	-	-
β	5	5	4	4	5	5
ρ	0.1	0.1	0.1	0.1	0.2	0.1
q_0	0.9	0.9	0.9	0.9	0.999	0.9
kt_0	-	0.5	0.5	0.95	-	-

Table 1. Good parameters found through parameter investigation for our six new algorithms for solving the vertex cover problem. These parameters are: number of ants m ; number of kernelants k ; influence of heuristic information β ; pheromone trail evaporation ρ ; probability of including the best choice in tour construction q_0 ; and quantity of pheromone to drop on kernelized nodes kt_0 .

that picks a node using the standard ACS random proportional transition rule (see equation (2)). However, if $q \leq q_0$, this algorithm then tests all the neighbours of node j to ensure that none of them belong to the kernelization set χ and therefore make a better choice. Since either j is in the vertex cover or all of its neighbours are, it is safe to include j into our vertex cover should none of its neighbours be in the kernelization set.

This algorithm along with TransKernelized ACS is expected to have a larger kernelization set χ compared with all the other kernelization algorithms as, when the random proportional transition rule is functioning similarly to regular ACS, the graph might become sufficiently altered (since once a node is placed into the solution, all its edges are effectively removed from the graph) that the kernelization rule is able to identify more kernelization nodes to be included in the solution

6 Evaluation

6.1 Parameter Investigation

We generated a set of 160 graphs and performed parameter analysis on all six new algorithms for the vertex cover problem. We timed how long ant colony system took to complete 200¹ iterations on each graph and that was the amount of time given to each algorithm during parameter analysis. We then explored one parameter at a time; table 1 contains the parameters found to be good.

6.2 Challenging Benchmarks

Benchmarks with Hidden Optimum Solutions for Graph Problems² is a website with a collection of challenging instances of graph problems constructed by hid-

¹ We chose 200 iterations because the Mann-Whitney statistical test has shown statistical improvement between 50, 100, 150, and 200 iterations but no improvement between 200 and 250 iterations

² <http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm>

	SYL	ACS	KACS	PKACS	CKACS	KAACS	TKACS	NTKACS	Opt
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
frb59-26-3.mis	1496	1493	1490	1493	1489	1490	1490	1491	1475
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
sum	39219	39177	39110	39116	39035	39037	39077	39092	38690
average	980.475	979.425	977.75	977.9	975.875	975.925	976.925	977.3	967.25

Table 2. A sample of results and the sum and average of all results for Shyu *et al.*'s algorithm, ACS, the optimal solution, and our six new algorithms, on benchmark instances.

	SYL	ACS	KACS	PKACS	CKACS	KAACS	TKACS	NTKACS
SYL		0.008	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
ACS			< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
KACS				0.4839	0.0003	< 0.0001	0.0012	0.2225
PKACS					0.0002	< 0.0001	0.0004	0.0819
CKACS						0.8808	0.0385	0.0042
KAACS							0.005	0.0005
TKACS								0.1336

Table 3. p-values for the Mann-Whitney U-test comparing Shyu *et al.*'s algorithm, ACS, and our six new algorithms, on benchmark instances.

ing optimum solutions for a specific problem in hard graphs [13]. This website contains forty instances for the VCP of between 450 and 1534 nodes.

We initially timed how long it took ant colony system to run on each graph for 200 iterations. We then set each algorithm to run on each graph for that period of time. Table 2 contains a sample of the results and the sum and average of the results for each algorithm. We have adopted the approach of Britteri [1], of using a maximum number of instances possible with just one run per instance for all experimentation.

We applied the Mann-Whitney U-test—recommended for use in metaheuristic analysis [12]—to these results and made the following conclusions. Firstly, all algorithms including the ACS with our modification as described in section 2 were statistically significant improvements over the algorithm by Shyu, Yin, & Lin. Secondly, all kernelization algorithms were a statistically significant improvement over the standard ACS. Thirdly, there was no statistical difference between CycleKernelized ACS and KernelAnts ACS but they were statistically better than all other algorithms. Lastly, Kernelized ACS, PreKernelized ACS, TransKernelized ACS, and Neighbourhood TransKernelized ACS all performed roughly the same; there is only a small statistical preference for TKACS. The p-values for these statistical tests are in table 3.

The primary conclusion from these results is that the kernelized algorithms do outperform regular ACS algorithms. The kernelization rule we used to create these six algorithms is useful in understanding a particular structure within instances of the vertex cover problem and guiding ant colony system to solving this structure in a more efficient way. This demonstrates that there are structures that ACS is poor at solving for the VCP and that techniques for approaching

Nodes	Edges	SYL	ACS	KACS	PKACS	CKACS	KAACS	TKACS	NTKACS
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
500	1000	277	264	263	264	263	262	262	262
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	sum	99229	95486	95286	95366	95354	95273	95010	94967
	average	198.458	190.972	190.572	190.732	190.708	190.546	190.02	189.934

Table 4. A sample of results and the sum and average of all results for Shyu *et al.*'s algorithm, ACS, and our six new algorithms, on random graphs with 100 to 500 nodes.

	SYL	ACS	KACS	PKACS	CKACS	KAACS	TKACS	NTKACS
SYL		< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
ACS			0.0001	0.0512	0.0375	< 0.0001	< 0.0001	< 0.0001
KACS				0.0316	0.0658	0.3421	< 0.0001	< 0.0001
PKACS					0.8181	0.0024	< 0.0001	< 0.0001
CKACS						0.0067	< 0.0001	< 0.0001
KAACS							< 0.0001	< 0.0001
TKACS								0.6241

Table 5. p-values for the Mann-Whitney U-test comparing Shyu *et al.*'s algorithm, ACS, and our six new algorithms, on random graphs of 100 to 500 nodes.

these structures such as kernelization are useful for ACS in developing better solutions.

6.3 Random Graphs

We constructed two groups of graphs. The first group of graphs contains graphs with 100 to 500 nodes, the second 600 to 1000 nodes. Altogether, our algorithms were tested on 1000 graphs, running on each graph for the quantity of time required for ACS to perform 200 iterations on that graph. Again only one run per graph was performed. Let us discuss these two groups separately. All random graphs were generated using the algorithm proposed by Skiena [10] and hand-selected to contain a variety of parameters for number of nodes, number of edges, and kernel sizes.

Table 4 contains a sample of the results and the sum and average of the results for each algorithm. We applied the Mann-Whitney U-test to these results and made the following conclusions. Firstly, all algorithms are a statistical improvement over the algorithm by Shyu *et al.*. Secondly, all kernelization algorithms except PreKernelized ACS are a statistical improvement over standard ACS. Thirdly, TransKernelized ACS and Neighbourhood TransKernelized ACS perform statistically speaking roughly the same, and these two algorithms are a statistical improvement over all other algorithms. Table 5 contains the p-values from these tests.

Table 6 contains a sample of the results and the sum and average of the results for each algorithm for the second set of graphs. We applied the Mann-Whitney U-test to these results and got the p-values in table 7. Firstly, all algorithms

Nodes	Edges	SYL	ACS	KACS	PKACS	CKACS	KAACS	TKACS	NTKACS
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
900	1350	446	432	428	431	429	428	428	428
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	sum	264700	255329	254407	254703	254785	254517	253671	253272
	average	529.4	510.658	508.814	509.406	509.57	509.034	507.342	506.544

Table 6. A sample of results and the sum and average of all results for Shyu *et al.*'s algorithm, ACS, and our six new algorithms, on random graphs of 600 to 1000 nodes.

	SYL	ACS	KACS	PKACS	CKACS	KAACS	TKACS	NTKACS
SYL		< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
ACS			< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
KACS				< 0.0001	< 0.0001	0.1118	< 0.0001	< 0.0001
PKACS					0.2041	0.0001	< 0.0001	< 0.0001
CKACS						< 0.0001	< 0.0001	< 0.0001
KAACS							< 0.0001	< 0.0001
TKACS								< 0.0001

Table 7. p-values for the Mann-Whitney U-test comparing results for Shyu *et al.*'s algorithm, ACS, and our six new algorithms, on random graphs of 600 to 1000 nodes.

were a statistical improvement over the algorithm by Shyu *et al.*. Similarly, all kernelization algorithms were a statistical improvement over standard ACS. Secondly, Neighbourhood TransKernelized ACS was an improvement over all other algorithms and TransKernelized ACS a clear second.

7 Conclusion

Our overall conclusion is that kernelization rules from the field of parameterized complexity are a useful and extensive resource for combination with ACO. Specifically, we have found that our six kernelization algorithms are useful for getting better results for both our benchmark problems and random graphs. In the larger, harder benchmark problems, it was found that pheromone based kernelization algorithms performed the best. This is probably because the kernelization within the pheromone based algorithms consumes less CPU time and so more iterations of ACS can be performed which is beneficial for these hard problems. On the other hand, the algorithms with kernelization integrated into the random proportional transition rule seem to work better on the random graphs. Again, this is probably because the random graphs are not quite as hard and so more time can be spent performing kernelization without loss of necessary cycles of ACS.

We have further identified a structure through this work that is common enough in both our benchmark problems and our random graphs to significantly affect quality of solutions, and that ant colony system is poor at solving. Our kernelization algorithms give us some insight into one area in which ant colony system for the vertex cover problem is poor.

Although the ACS algorithm should be capable of solving isolated instances of this structure, it would seem that when they occur in abundance, the random element of ACO is not able to solve them all. By integrating kernelization into ant colony optimization, the kernelization rule helps in processing this structure allowing better solutions to be found.

There are two broad avenues for future work. Firstly, further experimentation of this kind on the vertex cover problem using different kernelization rules would be useful for getting greater insight into what structures ant colony system is weak at solving. Investigation into why this is the case could also prove fruitful. Secondly, we plan to extend the approach to other optimization problems.

References

- [1] M. Birattari. On the Estimation of the Expected Performance of a Metaheuristic on a Class of Instances: How many instances, how many runs? *IRIDIA Technical Report No. TR/IRIDIA/2004-001*, April 2005.
- [2] C. Blum. Ant colony optimization: Introduction and recent trends. *Physics of Life Reviews*, 2:353–373, October 2005.
- [3] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence From Natural to Artificial Systems*. A volume in the Santa Fe Institute studies in the science of complexity. Oxford University Press, 1999.
- [4] J. Chen, I. Kanj, and W. Jia. Vertex cover: Further observations and further improvements. *Journal of Algorithms*, 41:280–301, 2001.
- [5] M. Dorigo and T. Stützle. *Ant Colony Optimization*. A Bradford Book. MIT Press, 2004.
- [6] R. Downey, M. Fellows, and U. Stege. Parameterized complexity: A framework for systematically confronting computational intractability. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 1997.
- [7] S. Gilmour and M. Dras. A Two-Pronged Attack on the Dragon of Intractability. *Proceedings of the 28th Australasian Computer Science Conference (ACSC-2005)*, pages 183–192, 2005.
- [8] S. Gilmour and M. Dras. Understanding the Pheromone System within Ant Colony Optimization. *Australian Conference on Artificial Intelligence*, pages 786–789, 2005.
- [9] S. J. Shyu, P.-Y. Yin, and B. M. T. Lin. An Ant Colony Optimization Algorithm for the Minimum Weight Vertex Cover Problem. *Annals of Operational Research*, 131:283–304, 2004.
- [10] S. S. Skiena. *The algorithm design manual*. Springer-Verlag New York, Inc., New York, NY, USA, 1998.
- [11] U. Stege and M. Fellows. An improved fixed-parameter-tractable algorithm for vertex cover. Tech. report 318, Department of Computer Science, ETH Zurich, April 1999.
- [12] E. D. Taillard. Comparison of non-deterministic iterative methods. *MIC'2001 - 4th Metaheuristic International Conference*, pages 272–276, 2001.
- [13] K. Xu, F. Boussemart, F. Hemery, and C. Lecoutre. A simple model to generate hard satisfiable instances. *Proceedings of 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 337–342, 2005.