A Two-Pronged Attack on the Dragon of Intractability

Stephen Gilmour and Mark Dras
Department of Computing
Macquarie University
{gilmour,madras}@ics.mq.edu.au

Abstract

One approach to tractably finding a solution to an NP-complete optimisation problem is heuristic, where the solution is inexact but quickly found; another approach is to reduce the problem in such a way that the reduction has the same solution as the original but is simpler, and then to solve the reduction, noting that this reduction is still NP-complete. It is possible to combine the two approaches with the goal of taking advantage of both the speed of the heuristic approach and the exactness of the reduction, but this is typically done only in a simple way. The aim of this paper is to begin exploring the range of ways in which these two classes of approach can be combined, using vertex cover as a problem instance.

We take as our reduction method the one used under parameterized complexity, where the problem is reduced through the application of kernelisation rules. For our heuristic we use Ant Colony Optimisation (ACO), where a set of 'ants' chooses a solution via distributed interaction; the search space 'terrain' that these ants traverse can be either flat or, as in a recent proposal, preconfigured by templates. In this paper, we investigate kernelisation rules as a notion of template that is richer than has previously been proposed, show that under three different models of combination the approach outperforms standard ACO for vertex cover, and analyse the solutions generated by the combination models with respect to each other.

Keywords: vertex cover, parameterized complexity, Ant Colony Optimisation

1 Introduction

With combinatorial optimisation problems (COPs), where determining an exact solution is intractable, there are a number of classes of approach to obtaining some solution. Approximation methods involve the design of some polynomial-time algorithm that will give an answer guaranteed to lie within a fixed bound of the optimal solution; for example, for the NP-complete vertex cover problem, the simple algorithm which randomly selects and deletes edges while adding the two endpoints to the cover has a solution at most twice the optimal. Heuristic methods provide a way of navigating the search space that is faster than the enumeration required for the determining of the optimum, but it has no guarantees regarding the goodness of the solution. Reduction methods constitute a third approach, where the problem is reduced in such a way that the reduction is smaller but whose solution can lead to the construction of the exact solution for the original; so even though this reduction will still be NP-complete, determining the solution will take less time.

Combinations of these approaches aim to take advantage of the best characteristics of each. Some are straightforward and quite commonly used: for example, running a heuristic method and an approximation algorithm in parallel, with the former being more likely through its search space navigation to find the optimal solution, and with the latter providing a bound on the result. In this paper we are interested in how the heuristic and reduction methods might be advantageously combined. Concretely, we take Ant Colony Optimisation as our heuristic method, and the parameterized complexity approach for our reduction method.

Ant Colony Optimisation (ACO) is a meta-heuristic for solving COPs (Dorigo, Caro & Gambardella 1999). Ant Colony Optimisation algorithms are biologically inspired and distributed in nature and have been found to be well suited to solving difficult optimisation problems, particularly dynamic optimisation problems (Bonabeau, Dorigo & Theraulaz 1999). Problems that ACO has been applied to include the Travelling Salesman Problem (Dorigo & Gambardella 1997), the Quadratic Assignment Problem (Maniezzo & Colorni 1999), communication networks (Caro & Dorigo 1998), and vehical routing (Gambardella, Taillard & Agazzi 1999). A recent development in ant algorithms—that is, any algorithm that takes its biological inspiration from ants—has been templates (Bonabeau et al. 1999). Templates are a mechanism for preconfiguring the search space of COPs so that ant algorithms are able to find solutions more quickly.

Parameterized complexity (Downey, Fellows & Stege 1997, Fellows 2002, Downey 2003) is a well established field that involves finding tractable algorithms for problems that have traditionally been classified as NP-complete and therefore intractable. Kernelization is a tool within the parameterized complexity framework for reducing COPs by a set of rules to a 'problem kernel'. This problem kernel is smaller than the original problem but is still NP-complete, with the solution to it allowing the optimal solution to the original problem to be reconstructed. Parameterized complexity has been applied to a number of problems, including Maximum Satisfiability (Bansal & Raman 1999), the Vertex Cover Problem (Stege & Fellows 1999), the Planar Dominating Set Problem (Alber, Fellows & Niedermeier 2002), and computational biology (Bodlaender, Downey, Fellows, Hallett & Wareham 1995).

A kernelization rule may for example have the form

If the problem has a component part X of a particular form, then X can be removed and the current optimal solution updated. This direction to remove a component part to simplify the problem could be interpreted as a direction to a heuristic method regarding areas of the search space to focus on or to ignore. This type of kernelization rule, then, could be adapted to derive a type of template. The relationship between templates and kernelization is the key idea of this paper, and will be discussed further on. This paper discusses three ACO algorithms we have implemented that use kernelization as a kind of template for the vertex cover problem. Formally, the vertex cover problem is defined as:

> Given an unweighted graph $G =< V, E >$, the vertex cover problem is to find a subset of vertices V' from the vertex set V such that every edge E in the graph is covered by one of the vertices in V'. That is, we want to find the minimum subset $V' \subseteq V$, such that for each $(x, y) \in E$ either x or y belongs to V'.

Section 2 reviews ant colony optimisation and templates, and presents a standard ACO algorithm for the vertex cover problem. Section 3 contains a discussion of parameterized complexity and looks at a kernelization algorithm for the vertex cover problem. Section 4 details the three algorithms we have produced for using kernelization as an ACO template, section 5 contains an evaluation and discussion of these three algorithms, and section 6 concludes this paper.

2 Ant Colony Optimisation

Ant colony optimisation (ACO) is a biologically inspired meta-heuristic for solving difficult combinatorial optimisation problems (COPs). "Ants are social insects, that is, insects that live in colonies and whose behaviour is directed more to the survival of the colony as a whole than to that of a single individual component of the colony" (Dorigo et al. 1999). Ant colony optimisation is inspired by real ants because it attempts to solve COPs by using distributed, social agents that are directed more to the goals of the colony than to the individual. However, this meta-heuristic does not seek to create a system that is a perfect replication of an ant colony, but to be a biologically inspired "engineering approach to the design and implementation of software systems" (Dorigo et al. 1999). The ACO meta-heuristic uses three different mechanisms that are used by real ant colonies to find the shortest path from their nest to food sources. These three mechanisms are stigmergy, autocatalysis, and emergent behaviour.

Section 2.1 will look at how ACO uses stigmergy, autocatalysis, and emergent behaviour to solve COPs, section 2.2 will present an ACO algorithm from Shyu, Yin, Lin & Hsiao (2001) for solving the vertex cover problem from which our proposed algorithms are derived, and section 2.3 will discuss templates, a new direction in ant algorithms where some structure is given to the search space to allow better solutions to be found.

2.1 Biological Inspiration

Real ants do not communicate directly to each other but communicate indirectly through the laying of pheromone. Pheromone can be recognized as both belonging to a specific ant and to a specific colony, and allows ants to leave very simple messages to other ants such as the direction to food or a warning regarding the approach of a predator. This indirect communication mediated by pheromone laying is called stigmergy. In ACO, we use stigmergy by having ants communicate via the modification of the problem representation through the laying of "digital pheromone". As artificial ants construct solutions to the problem they are working on, they lay digital pheromone indicating the goodness of the solution along the path of the solution they have found (in real ants, the goodness of a solution is implicitly evaluated whereas in many ACO applications it is explicitly evaluated). Then when other artificial ants come across this pheromone, they are probabilistically more likely to follow it than to explore elsewhere in the problem. Digital pheromone is usually represented by a numerical value at each distinct location in the problem representation and can only be placed locally and detected locally.

As the following of pheromone is probabilistic, ants will occasionally try other parts of the search space close to the most successful solutions in the hope of finding better solutions. If successful, a quantity of digital pheromone will be laid to indicate that an even better solution has been found. Since the pheromone only directs the search towards other successful solutions, the pheromone merely guides other ants to a part of the search space that has proven successful. This process is described as "characterized by a positive feedback loop, where the probability with which an ant chooses a path increases with the number of ants that previously chose the same path" (Dorigo, Maniezzo & Colorni 1991). In other words, ACO utilizes autocatalysis: the amount of pheromone in a location dictates probabilistically how many ants will visit that location which in turn will cause more pheromone to be deposited in that location, and so on.

Real ants are blind and have virtually no memory. In a real ant colony, the ants do not plan to find the shortest route from their nest to a food source, but this behaviour emerges from ants dropping pheromone as they walk between food and the nest and other ants being attracted by that pheromone and being partially directed by it. It is only through emergent behaviour (behaviour produced implicitly from the interactions of the components of the system) from interacting ants in the colony that the most direct route from the nest to food is obtainable. Although artificial ants are not simple in the same ways as real ants, they are certainly less complex computationally than would be expected considering the quality of the solutions they produce. Each ant individually is only capable of generating a solution to the problem of highly variable quality and very rarely is anywhere close to optimal. However, through autocatalysis and stigmergy, an artificial ant colony is able to produce a near-optimal (if not optimal) solution to the problem it is being applied to.

2.2 ACO for the Vertex Cover Problem

In the vertex cover problem, we need to find a subset of nodes V' from the graph G such that V' covers all the edges in G, under the definition of section 1. This subset of nodes is found by each ant taking a walk on the graph. Every node that an ant visits goes into the vertex cover that that ant is constructing. Unfortunately this subset of nodes is not guaranteed to form a path or a tree on the underlying graph. Therefore,

we must "construct a complete graph $G_c = (V, E_c)$ involving the vertex set V of G such that every pair of vertices are connected by an edge in E_c. As a result if we apply ACO on G_c, the ants can construct tours corresponding to any possible variations of V'" (Shyu et al. 2001). This complete graph allows our ants to pick any subset of nodes from the graph to form a vertex cover by taking a walk on the graph without the need for backtracking. However, in order to preserve the original graph, the following connectivity function needs to be defined for each edge (r, j) and each ant k:

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

This connectivity function also allows ants to evaluate a preference for putting a node into the vertex cover it is constructing. When a node j is added to the vertex cover by ant k, the connectivity function ψ_k for each of its connected edges (r, j), for all r, is set to 0. The greater the number of edges a node has with $\psi_k = 1$, the more strongly preferred it is. When there are no edges left with $\psi_k = 1$, this means that all edges have been covered and therefore a vertex cover has been constructed for the original graph. Whenever this process is complete, all connectivity values need to be reset according to equation (1) so that the process can begin again.

Ants use a stochastic state transition rule to pick the next node to visit and place in the vertex cover. This rule needs to be stochastic to prevent stagnation occurring in the constructed solutions—stagnation is when all the ants are continally constructing the same solution and not exploring for better solutions. Therefore Shyu et al. (2001) have developed the following rule that describes the probability of an ant k visiting node j and placing it in the vertex cover:

$$p_j^k = \begin{cases} 1, & \text{if } q < q_0 \text{ and j} = \arg\max\limits_{r \in A_k}\{\tau_r \eta_{rk}^\beta\}; \\ 0, & \text{if } q < q_0 \text{ and j} \neq \arg\max\limits_{r \in A_k}\{\tau_r \eta_{rk}^\beta\}; \\ \dfrac{\tau_j \eta_{jk}^\beta}{\sum\limits_{r \in A_k} \tau_r \eta_{rk}^\beta}, & \text{if } q \geq q_0, \end{cases} \quad (2)$$

where A_k denotes the set of accessible vertices for ant k,

η_{jk} is the heuristic goodness of node j,

τ_j is the amount of pheromone on node j,

β is a rational number greater than zero,

q is a random number from the open interval $(0, 1)$ and q_0 is a specified threshold.

The probability of ant k visiting node j is dependent on both the heuristic goodness of node j and the amount of pheromone on node j. This is so that initially the heuristic can guide the search but as better solutions are found through the stochastic element of the search, pheromone starts to drive the search. The relationship between these two measures is balanced by β which determines when this transition occurs. The heuristic goodness of a node is calculated according to the number of edges connected to j with $\psi_k(r, j) = 1$, for all r, and the weight of node j. That is:

$$\eta_{jk} = \frac{\sum\limits_{(r,j) \in E_c} \psi_k(r,j)}{w(j)}.$$

Notice that this heuristic is dynamic. Because each time we place a node j in the vertex cover we set for all r, $\psi_k(r, j) = 0$, the heuristic goodness for all adjacent nodes r changes. This is different from other applications of ACO where the heuristic goodness of choosing a node or edge (depending on the application) is static and never changes. In the vertex cover problem, we are interested in constructing a subset of nodes that covers all edges, and therefore we place pheromone on nodes rather than edges. There are two pheromone update rules used within this algorithm: the Global Update Rule; and the Local Update Rule.

The global update rule occurs once at the end of each cycle. Pheromone is placed on the nodes in the vertex cover of the currently best solution found by any ant. The local update rule is performed by each ant whilst it is constructing solutions to the problem. The local update rule "decreases the pheromone intensity on the vertex just visited by an ant and makes the selected vertices less attractive to other ants" (Shyu et al. 2001). This is to prevent stagnation from occurring in solutions.

The global update rule is:

$$\tau_i = \begin{cases} (1-\rho)\tau_i + \tau_0 \rho \Delta\tau_i, & \text{if } i \in V_c' \\ (1-\rho)\tau_i, & \text{otherwise} \end{cases} \quad (3)$$

where $\Delta\tau_i = \dfrac{1}{\sum\limits_{j \in V_c'} w(j)}$,

V_c' are the nodes in the current best solution.

That is, for all nodes, the pheromone already present is evaporated according to the constant $1 - \rho$. Then the nodes in the current best solution have added to them the inverse of the sum of the weights of the nodes in the vertex cover $\Delta\tau_i$ multiplied by the amount of pheromone initially placed on every node τ_0 and the evaporation constant ρ. To calculate the amount of pheromone initially placed on every node, Shyu et al. (2001) use:

$$\tau_0 = \frac{|V| * (|V| - |V_c'|)}{\sum\limits_{j \in V_c'} w(j)}$$

The local update rule that is applied by all ants on the solution they are constructing as they construct it is:

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

Here φ is the evaporation rate. Both evaporation constants ($\varphi, \rho \in (0, 1)$) simulate the natural evaporation rate of pheromone previously laid on nodes to ensure that positive feedback does not explode and stop better solutions from being found.

2.3 Ants and Templates

Ant algorithms have been found to be quite promising: for some combinatorial optimisation problems they have reached world class performance (Dorigo et al. 1999). A new area of research is giving ant algorithms a template. A template is a pattern used to organise activities. It is said that the ants "self-organise along a template" (Bonabeau et al. 1999). In other words, ants still develop a solution using stigmergy and autocatalysis, but the template guides them so that the final solution is more predictable or more easily found.

As an example of the use of templates, we take a problem in graph partitioning, that involves dividing a graph into c clusters of approximately equal size while minimizing the number of connections between them. This problem is non-parametric and therefore the number of clusters that may form is always unknown. For this problem, the template is a grid that covers the environment (graph). The grid contains for each possible location the probability for items to be placed in that particular location in the graph. One possible template for this problem could have high probability in the corners of the graph and low probability in the centre. This means that the ants are probabilistically more likely to drop items in the four corners of the graph. This template provides parameterisability and predictability to this problem by predefining the number of clusters to be around four. It works very well when the graph does naturally divide into four clusters, and works reasonably well if the number of clusters is close to four (so if it is three, it should just form three clusters in three corners of the graph).

Templates provide a mechanism for reducing problems to make them easier to compute (it is easier to divide a graph into approximately four clusters than it is to divide it into an unknown number of clusters) and provides predictability in solutions (Bonabeau et al. 1999). However, research into templates is still greatly under-developed. Thus far, templates have only been applied to graph partitioning and data analysis problems (Bonabeau et al. 1999), and no attempt has been made to apply them to more prominent NP-complete problems such as the vertex cover problem.

3 Parameterized Complexity

In this section, we outline the notion of Parameterized Complexity as presented in Fellows (2002).

Consider two problems, the vertex cover problem and the dominating set problem. Formally these problems are defined as:

VERTEX COVER
Input: A graph $G = (V, E)$ and a positive integer k.
Question: Does G have a vertex cover of size at most k? (A *vertex cover* is a set of vertices $V' \subseteq V$ such that for every edge $uv \in E$, $u \in V'$ or $v \in V'$ (or both).)

DOMINATING SET
Input: A graph $G = (V, E)$ and a positive integer k.
Question: Does G have a dominating set of size at most k? (A *dominating set* is a set of vertices $V' \subseteq V$ such that for all $u \in V : u \in N[v]$ for some $v \in V'$)

Both of these problems are NP-complete. The best known algorithm for the vertex cover problem runs in time $O(1.271^k + kn)$ (Chen, Kanj & Jia 1999) (an improvement over one of the earlier algorithms for this problem that ran in $O(2^k n)$) yet the best known algorithm for the dominating set problem is a brute force algorithm that runs in time $O(n^{k+1})$ (Fellows 2002). As demonstrated by the frequently referenced table 1 (Downey et al. 1997, Fellows 2002, Downey 2003), although both problems are exponential, clearly the vertex cover algorithm is tractable (in fact it is tractable for all n if $k \leq 60$) yet the dominating set algorithm is not. But both problems are considered NP-complete under classical complexity theory and therefore intractable.

Parameterized complexity is not a replacement for traditional complexity theory but works to solve some of the problems many complexity theorists and applied computing practitioners are having with complexity theory and algorithm design such as the one just demonstrated. Parameterized complexity has two aspects that fulfill these aims: A framework for complexity analysis; and a toolkit of methods for algorithm design.

	$n = 50$	$n = 100$	$n = 150$
$k = 2$	625	2,500	5,625
$k = 3$	15,625	125,000	421,875
$k = 5$	390,625	6,250,000	31,640,625
$k = 10$	1.9×10^{12}	9.8×10^{14}	3.7×10^{16}
$k = 20$	1.8×10^{26}	9.5×10^{31}	2.1×10^{35}

Table 1: The ratio $\frac{n^{k+1}}{2^k n}$ for various values of n and k.

Section 3.1 discusses the parameterized complexity framework and how it improves traditional complexity theory; section 3.2 looks at a particular method for coping with intractability called kernelization; and section 3.3 contains the details of a kernelization algorithm for the vertex cover problem, some parts of which we will use in the algorithms we present in section 4.

3.1 Parameterized Complexity Framework

In order to apply the parameterized complexity framework to problems, they need to be defined in terms of a parameterized language. For example, the definitions of the vertex cover problem and dominating set problem are defined using a parameterized language in section 3 where the parameter is k. Formally, a parameterized language is defined as (Fellows 2002):

DEFINITION 1: *A parameterized language L is a subset $L \subseteq \sum^* \times \sum^*$. If L is a parameterized language and $(x, k) \in L$ then we will refer to x as the main part, and refer to k as the parameter.*

A parameter does not need to be a numerical value such as in the vertex cover problem above. A parameter can also represent "an aggregate of various parts or structural properties of the input" (Fellows 2002). The function of the parameter is to act as a naturally occurring bound on the problem. Many concrete problems in practice are governed by parameters of all kinds that bound the problem and "if we can design algorithms with running times like $2^k n$ for these problems, then we may have something really useful" (Fellows 2002). Such algorithms are called fixed-parameter tractable algorithms.

Within parameterized complexity, we seek fixed-parameter tractable (FPT) algorithms. FPT is in fact a complexity class in parameterized complexity that is somewhat analogous to P in traditional complexity. A problem L is said to be in FPT if the complexity of the best known algorithm for it can be shown to be either multiplicatively or additively fixed-parameter tractable (Fellows 2002):

DEFINITION 2: *A parameterized language L is multiplicatively fixed-parameter tractable if it can be determined in time $f(k)q(n)$ whether $(x, k) \in L$, where $|x| = n$, $q(n)$ is a polynomial in n, and f is a function (unrestricted).*

DEFINITION 3: *A parameterized language L is additively fixed-parameter tractable if it can be determined in time $f(k)+q(n)$ whether $(x, k) \in L$, where $|x| = n$, $q(n)$ is a polynomial in n, and f is a function (unrestricted).*

Clearly the best known algorithm for the dominating

set problem which runs in time $O(n^{k+1})$ is not in FPT whereas the best known algorithm for the vertex cover problem which runs in time $O(1.271^k + kn)$ is additively fixed-parameter tractable and is therefore in the complexity class FPT.

However, before discussing the practical application of parameterized complexity, a definition for a parametric transformation needs to be made. Parametric transformations show us which complexity class a problem belongs to by showing how the problems in each class relate to each other. Formally a parametric transformation is defined as (Fellows 2002):

DEFINITION 4: *A parametric transformation from a parameterized language L to a parameterized language L' is an algorithm that computes from input consisting of a pair (x, k), a pair (x', k') such that:*

1. *$(x, k) \in L$ if and only if $(x', k') \in L'$;*
2. *$k' = g(k)$ is a function only of k;*
3. *the computation is accomplished in time $f(k)n^{\alpha}$, where $n = |x|$, α is a constant independent of both n and k, and f is an arbitrary function.*

The essential property of parametric transformations is that if L transforms to L' and $L' \in FPT$, then $L \in FPT$ (Downey et al. 1997).

3.2 FPT Through Kernelization

Parameterized complexity provides a new framework that is a potential means for coping with classical intractability because it takes into consideration naturally occurring parameters that can bound problems. Within parameterized complexity, we seek FPT algorithms which, although often NP-complete, are tractable for most input. The framework also contains tools for designing algorithms that are in FPT. One such tool is kernelization. Kernelization is a form of pre-processing that can be done on a problem in polynomial time that reduces a problem so that often even a brute force search is tractable on the resulting kernel. However, although kernelization makes a problem easier to process by reducing it to its problem kernel, it doesn't lose any necessary information to finding an optimal solution. Kernelization is a practical tool for designing FPT algorithms. In fact, kernelization is such an important tool for developing FPT algorithms that "a parameterized language L is fixed-parameter tractable if and only if it is kernelizable" (Fellows 2002). This is true to such an extent that the following formal definition of kernelization is an equivalent definition of FPT as well (Fellows 2002):

DEFINITION 6: *A parameterized language L is kernelizable if there is a parametric transformation of L to itself, and a function h (unrestricted) that satisfies:*

1. *the running time of the transformation of (x, k) into (x', k'), where $|x| = n$, is bounded by a polynomial $q(n, k)$ (so that in fact this is a polynomial-time transformation of L to itself, considered classically, although with the additional structure of a parametric reduction);*
2. *$k' \leq k$;*
3. *$|x'| \leq h(k)$, where h is an arbitrary function.*

Kernelization was originally a pre-processing step that occurred before using another algorithm (such as using bounded search trees). However, this is no longer true. Some algorithms do still use kernelization as merely a pre-processing stage (such as (Niedermeier & Rossmanith 1999) for the vertex cover problem) whereas other algorithms apply kernelization after every round of the algorithm (such as (Downey et al. 1997) also for the vertex cover problem). Regardless, kernelization has proved to be a powerful tool for developing FPT algorithms for many NP-complete problems.

3.3 Kernelization for the Vertex Cover Problem

In this section we look at the kernelization rules for a particular algorithm for the vertex cover problem; we use a subset of these later in our own approach. The algorithm these kernelization rules come from is detailed in Downey et al. (1997) and comprises two stages. The first stage reduces the problem to its problem kernel or gives the answer "no". The second stage uses bounded search trees to get a solution to the problem. However, the first stage is reapplied to the graph after every step within the second stage. This algorithm does not use kernelization as merely a pre-processing step but continually reapplies it to the graph as a part of the bounded search tree algorithm. We will only look at the first stage of this algorithm in detail.

The following rules are applied to a graph G until no further applications are possible. All these rules are taken from (Downey et al. 1997).

(0): If G has a vertex v of degree greater than k, then replace (G, k) with $(G - v, k - 1)$.

(1): If G has two nonadjacent vertices u, v such that $|N(u) \cup N(v)| > k$, then replace (G, k) with $(G + uv, k)$.

(2): If G has adjacent vertices u and v such that $N(v) \subseteq N[u]$, then replace (G, k) with $(G - u, k - 1)$.

(3): If G has a pendant edge uv with u having degree 1, then replace (G, k) with $(G - \{u, v\}, k - 1)$.

(4): If G has a vertex x of degree 2, with neighbours a and b, and none of the above cases applies (and thus a and b are not adjacent), then replace (G, k) with (G', k) where G' is obtained from G by:

- Deleting the vertex x
- Adding the edge ab
- Adding all possible edges between $\{a, b\}$ and $N(a) \cup N(b)$.

(5): If G has a vertex x of degree 3, with neighbours a, b, c, and none of the above cases applies, then replace (G, k) with (G', k) according to one of the following cases depending on the number of edges between a, b and c.

(5.1): There are no edges between the vertices a, b, c. In this case G' is obtained from G by:

- Deleting vertex x from G.
- Adding edges from c to all the vertices in $N(a)$.
- Adding edges from a to all the vertices in $N(b)$.
- Adding edges from b to all the vertices in $N(c)$.
- Adding edges ab and bc.

(5.2): There is exactly one edge in G' between the vertices a, b, c which we assume to be the edge ab. In this case G' is obtained from G by

- Deleting vertex x from G.
- Adding edges from c to all the vertices in $N(a) \cup N(b)$.
- Adding edges from a to all the vertices in $N(c)$.
- Adding edges from b to all the vertices in $N(c)$.
- Adding edge bc.
- Adding edge ac.

The problem kernel found from these kernelization rules is an instance of "(G', k') where $|G'| \leq k^2$ and $k' \leq k$ such that G' has a vertex cover of size k' if and only if G has a vertex cover of size k" (Downey et al. 1997). These rules have the effect of removing all nodes of degree three or less from the graph. The job of the bounded search tree algorithm in the second phase deals with the nodes of degree greater than three. Within phase two, if a node v is of degree six or greater, then two children are placed within the search tree. The first child has v placed in the vertex cover and the parameter reduced by 1. The second child has $N(v)$ placed in the vertex cover and the parameter reduced by the degree of v. This rule deals with all nodes of degree six or greater, and since kernelization (phase one) is reapplied to the graph after each step of branching, this just leaves nodes of size four and five to deal with. The bounded search tree algorithm deals with these nodes using four branching rules that are fairly complex and are discussed in Downey et al. (1997).

It can be shown from rule (0) alone that the kernelization rules detailed in this paper perform a polynomial time parametric transformation of vertex cover to itself, so that the resultant graph has a size bounded by a function of the parameter k. In other words, these rules demonstrate that vertex cover is indeed in FPT according to definition 6. Further, if the number of vertices in G' is more than k^2 then we can conclude that there is no vertex cover of size k for graph G. However, if $|G'| \leq k^2$ and $k' \leq k$ and no solution has been found, then we move onto the second phase which involves exhaustively answering the question for (G', k') whilst simultaneously reapplying kernelization after each branching step.

As already discussed in section 3.2, kernelization was traditionally just a pre-processing step that preceded another algorithm. Niedermeier & Rossmanith (1999) discuss a different parameterized complexity algorithm for the vertex cover problem that performs slightly better than Downey et al. (1997) but uses kernelization as a pre-processing step only. Whereas in Downey et al. (1997) the kernelization rules deal with nodes of degree one, two, and three, this algorithm uses "the more 'classical approach' where the search tree deals also with vertices of degree 2 and 3 and reduction to problem kernel is only applied once as a pre-processing phase" (Niedermeier & Rossmanith 1999). This algorithm improves the complexity by Downey et al. (1997) of $O(kn+1.31951^k k^2)$ with an upper bound of $O(kn + 1.29175^k k^2)$. However, since this algorithm contains just one kernelization rule (rule (3) from above), we decided to look at the more interesting algorithm by Downey et al.

4 Combination Algorithms

In section 2.3 we described the use of a template in graph partitioning. The template there effectively assigns prior probabilities through pheromone deposits: this is done in a fairly simplistic manner by considering a grid to be superimposed over the graph, and the pheromone deposited according to this grid: there is no consideration of the structure of the graph itself. Kernelization rules, as enumerated in section 3.3, do precisely that: they identify those vertices in the graph where components of the vertex cover can be determined with certainty. Thus these kernelization rules can act as a template on the graph itself, rather than via a grid superimposed on the graph.

We have identified three general strategies for implementing ACO that makes use of kernelization techniques as a template for the vertex cover problem. The three strategies are:

1. Integrate kernelization into the ant state transition rule;
2. Perform kernelization before running ACO, placing pheromone on the included nodes;
3. Perform kernelization after each cycle of the ACO algorithm, placing pheromone on the included nodes.

We have implemented all three strategies and run tests to see how they perform compared to each other and compared to the standard ACO algorithm. However, in order to integrate kernelization into ACO, we needed to find a way to perform kernelization in a distributed manner. Although some kernelization rules for the vertex cover problem do naturally lend themselves to a multi-agent implementation, some of them do not. For example, in rule (4) from section 3.3, new edges are required to be added, which is not straightforward to do in a distributed manner. Section 4.1 discusses the kernelization rules we used for all three algorithms. Section 4.2 looks at an implementation of the first strategy that is called TransKernelized Vertex Cover ACO. Section 4.3 looks at an implementation of the second strategy that is called PreKernelized Vertex Cover ACO and section 4.4 looks at an implementation of the third strategy that is called CycleKernelized Vertex Cover ACO. Section 5 then compares these three algorithms with an implementation of a standard algorithm as described in section 2.2.

4.1 Kernelization Rules

Throughout the literature concerning kernelization for the vertex cover problem, many different sets of rules have been proposed. Some rules have been included in all algorithms such as the original kernelization rule from Buss' (Buss & Goldsmith 1993) algorithm (see rule 1 below). Other rules have been proposed and then replaced later by another rule that perhaps combined a few different rules into a single, more complex rule. (Stege & Fellows 1999) contains a literature review of kernelization rules that ends with a new algorithm that has not since been superseded. In order to integrate kernelization into ACO, we need to use rules that can be implemented in a distributed system. Therefore, we have identified four rules from many different algorithms that all lend themselves to a multi-agent implementation and can validly work together to kernelize a graph into its problem kernel. The four rules are (Stege & Fellows 1999, Downey et al. 1997):

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 $N(v) \subseteq N[v]$, 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.

These four kernelization rules are justified as follows:

1. Any k-element vertex cover in G must contain v, since otherwise it would be forced to contain $N(v)$, which is impossible.
2. If a vertex cover C did not contain u then it would be forced to contain $N[v]$. But then there would be no harm in exchanging v for u.
3. If G has a k-element vertex cover C that does not contain v, then it must contain u. But then $C - u + v$ is also a k-element vertex cover. Thus G has a k-element vertex cover if and only if it has one that contains v.
4. If G has a k-element vertex cover C that does contain u, then either y or z must also be in C since they are adjacent, say y. But since there is no harm in exchanging the node u with z, then C contains y and z.

Within all the algorithms we have implemented, these are the only four kernelization rules that have been used. This list is by no means a complete list of rules that could be implemented in a distributed way, but these are sufficient to test the usefulness of using parameterized complexity to derive templates for ACO.

4.2 TransKernelized Vertex Cover ACO

Within this algorithm, we have integrated the kernelization rules into the ants' state transition rule, modified from equation (2) in 2.2. The new state transition rule is:

$$p_j^k = \begin{cases} 1, & \text{if } q < q_0 \text{ and } ((j \in \chi_k) \text{ or } (|\chi_k| = 0 \text{ and } j = arg\max_{r \in A_k}\{\tau_r \eta_{rk}^\beta\})) \\ 0, & \text{if } q < q_0 \text{ and } j \notin \chi_k \text{ and } j \neq arg\max_{r \in A_k}\{\tau_r \eta_{rk}^\beta\} \\ \dfrac{\tau_j \eta_{jk}^\beta}{\sum_{r \in A_k} \tau_r \eta_{rk}^\beta}, & \text{if } q \geq q_0, \end{cases}$$

where χ_k denotes the set of vertices that belong in the vertex cover according to the kernelization rules in section 4.1 for ant k.

This new state transition rule is fundamentally the same as in section 2.2 from Shyu et al. (2001) except for the use of a new set χ_k in the cases when $q < q_0$.

4.3 PreKernelized Vertex Cover ACO

Within this algorithm, we perform kernelization before running ACO, placing pheromone on the included nodes. This has been implemented by introducing a new pheromone rule called the initialization pheromone update rule. This rule is run prior to starting the ACO algorithm and lays an initial quantity of pheromone on the nodes determined by the kernelization rules of section 4.1. The initialization update rule for every node i is:

$$\tau_i = \begin{cases} |V|, & \text{if } i \in \chi \\ 0, & \text{otherwise.} \end{cases}$$

where χ denotes the set of vertices that belong in the vertex cover according to the kernelization rules in section 4.1.

Because an initial round of ACO has not been performed, we are not able to drop τ_0 pheromone on the nodes in χ. Therefore, we drop a quantity of pheromone equal to the number of nodes in the graph instead. Otherwise the remainder of this algorithm is identical to the standard ACO algorithm presented in section 2.2 from Shyu et al. (2001).

4.4 CycleKernelized Vertex Cover ACO

Within this algorithm, we perform kernelization after each cycle of ACO, placing pheromone on the included nodes. This has been implemented by modifying the global update rule to place pheromone on the nodes determined by kernelization as well as on the nodes of the current best solution. The new global update rule, modified from equation (3), is:

$$\tau_i = \begin{cases} (1-\rho)\tau_i + \tau_0\rho\Delta\tau_i + \tau_0, & \text{if } i \in V_c' \text{ and } i \in \chi \\ (1-\rho)\tau_i + \tau_0\rho\Delta\tau_i, & \text{if } i \in V_c' \\ (1-\rho)\tau_i + \tau_0, & \text{if } i \in \chi \\ (1-\rho)\tau_i, & \text{otherwise} \end{cases}$$

where χ denotes the set of vertices that belong in the vertex cover according to the kernelization rules in section 4.1.

This new global update rule contains two more cases than the global update rule in section 2.2 but fundamentally just involves placing τ_0 pheromone on the nodes that are in χ after performing the regular global update rule. Otherwise this algorithm is identical to the standard ACO algorithm presented in section 2.2 from Shyu et al. (2001).

5 Evaluation

The test to see if these three algorithms are an improvement over traditional ACO is whether they find better solutions when possible to various vertex cover problems, or whether they find the same solution in less time. Each test has been run twice for each graph, and for each size of graph, two graphs have been generated, giving four tests for each size of graph (this is discussed further in section 5.2). All graphs used are generated from an algorithm outlined in section 5.1. We ran tests on graphs of size 10, 20, 30, 50, 70, 90, 110, 130, 140, 150, 160, 170, 190, 210, 230, 250, 270, 290, 300, 310, 500, and 1000 nodes. For each test, all three algorithms and a traditional ACO implementation were run simultaneously (as much as possible on a single CPU machine). Each algorithm is given 10 minutes running time for itself on the CPU. The output of each test is the size of the smallest vertex cover

Graph ID	Num of Nodes	Solution	Time (sec.)
1	1000	632	814.5
1	1000	635	647.3
2	1000	653	606.4
2	1000	656	642.9
avg	1000	644	677.8

Table 2: Results for traditional ACO on graphs of size 1000

found by each algorithm and the amount of time it took to get to that solution.

Section 5.1 discusses the algorithm we have created to randomly generate graphs for testing the three new ACO algorithms; section 5.2 describes the results of running these tests; and section 5.3 discusses these results.

5.1 Graph Generation Algorithm

Our graph generation algorithm, given a set size n, randomly generates a graph with n nodes. The nodes in the graph are placed spatially on the screen with a minimum bound t on the number of pixels between all nodes. This minimum number of pixels is calculated according to the formula $t = \frac{20000}{n+400}$.

Initially the nodes are created and given a random location spatially on the screen. If the Euclidean distance between a node and any other is greater than the threshold t, then the node is placed at its current location on screen. Otherwise a new location is randomly generated and compared again to see if it fulfills this requirement.

Then, for each node the algorithm picks the minimum guaranteed number of edges connecting to it, generated randomly from a uniform distribution on the interval [0,4]. These n edges are then created, connecting that node with its n closest neighbours (as determined by Euclidean distance). Note that since this takes place for every node, it is possible to have a maximum of $n-1$ edges connected to any node in the generated graph.

5.2 Results

As discussed above, for each size of graph, two runs were done each on two different graphs of that size. For example, the results for the standard ACO algorithm on graphs of size 1000 were as in table 2. The Solution column gives the number of nodes in the vertex cover that the algorithm has chosen as optimal; the Time column gives the number of seconds to first find a solution of this size. Table 3 presents summarised data for all four algorithms, that is, the three new ACO algorithms plus the standard ACO algorithm.

The columns labelled (A) are the results for TransKernelized Vertex Cover ACO. The columns labelled (B) are the results for CycleKernelized Vertex Cover ACO. The columns labelled (C) are the results for PreKernelized Vertex Cover ACO. The columns labelled (D) are the results for traditional Vertex Cover ACO.

The Wilcoxon rank sum test can be used to test the null hypothesis that two populations X and Y have the same continuous distribution (by comparing n samples from each population). If the two populations have the same approximate mean, then W is close to zero. The larger W is, the further apart the two populations are in distribution. If the p value is small, this tells us that this difference in distributions is not a coincidence.

Table 5 contains the results from applying the Wilcoxon rank-sum test to the raw data generated from the three algorithms and a standard ACO algorithm. In table 5, the columns labelled Sol are the results of comparing the size of the vertex cover found by an algorithm against the size of the vertex cover found by traditional ACO within the time limit. The columns labelled Time are the results of comparing the time it takes to find a solution by an algorithm against traditional ACO in the cases where they both find the same solution. In the case of these tests, we want a negative W with an absolute value as large as possible. The larger the absolute value of W is, the better the algorithm performs overall compared to traditional ACO.

Tables 6 and 7 contain the results of applying the Wilcoxon rank-sum test to the processed data that is sampled in section 5.2 for the three algorithms and a standard ACO. A negative value for W shows that the first algorithm being compared is better whereas a positive value shows that the second algorithm being compared is better. For example, in comparing TransKernelized Vertex Cover ACO vs PreKernelized Vertex Cover ACO the W value for solutions is -91, so we can say that the TransKernelized Vertex Cover ACO algorithm performs significantly better than the PreKernelized Vertex Cover ACO algorithm in generating solutions.

In table 6, the columns show how each algorithm performs in terms of solutions compared to each other. In table 7, the columns show how each algorithm performs in terms of time compared to each other for the cases where both algorithms find the same solution.

5.3 Discussion

In this section we will draw a few conclusions from the results discussed in section 5.2.

Firstly, all three combination algorithms perform significantly better than the standard algorithm in solution quality and in time to solution with p-value 0.0001, as indicated in table 5.

Secondly, TransKernelized Vertex Cover ACO performs significantly better than all other algorithms. This is indicated in table 5 where its W value is much greater than the W values for either of the other two algorithms, and confirmed the pairwise results of table 6, where TransKernelized outperforms the other three, with p-values significantly less than 0.01.

Thirdly, CycleKernelized Vertex Cover ACO and PreKernelized Vertex Cover ACO perform roughly the same. Table 5 gives these algorithms W values of -703 and -535 respectively. The p-values in the pairwise comparison of the two in table 6 mean that we cannot claim that the performance of the algorithms differs.

Fourthly, CycleKernelized Vertex Cover ACO and PreKernelized Vertex Cover ACO appear to find a solution most quickly on average. However, there is not sufficient data, as indicated in table 7, to make a strong claim about this.

Further, to investigate the scalability of the algorithms, we performed a linear regression on the averaged results for the size of the vertex cover. The steeper the slope produced, the worse the solutions generated by that algorithm are likely to be as the algorithm scales up. Table 4 contains the slopes, intersects, and r^2 values for the three new algorithms

Size	Sol. (A)	Time (A)	Sol. (B)	Time (B)	Sol. (C)	Time (C)	Sol. (D)	Time (D)
10	7	0.34	7	0.07	7	0.08	7	0.08
50	31	1.30	31	1.69	31	1.039	31.5	6.12
90	59	4.04	59	4.93	59	3.47	59	7.79
130	80	8.49	80	47.54	80.5	5.507	80.25	165.86
170	106.5	47.53	108	117.02	107.5	22.04	109.25	91.70
210	127.5	31.86	127.5	27.25	127.5	32.43	127.75	132.01
250	155.5	66.52	159.25	196.38	158.75	32.84	160.25	156.94
290	180	170.04	181	171.56	182	50.85	182	295.61
500	306.25	302.10	311.75	415.07	312.5	195.18	314.25	470.77
1000	631.5	527.34	636.5	615.11	633.25	431.31	644	677.78

Table 3: Extract of results from averaged TransKernelized Vertex Cover ACO (A) data, CycleKernelized Vertex Cover ACO (B) data, PreKernelized Vertex Cover ACO (C) data, and traditional Vertex Cover ACO (D) data. Times measured in seconds.

Algorithm	Slope	Intersect	r^2
TransKernelized Vertex Cover ACO	0.6271	-1.1040	0.9996
CycleKernelized Vertex Cover ACO	0.6334	-1.4241	0.9997
PreKernelized Vertex Cover ACO	0.6311	-0.9157	0.9998
Traditional Vertex Cover ACO	0.6408	-2.0130	0.9996

Table 4: Slope, intersect, and r^2 values for curves fitted to solutions of three new ACO algorithms and a standard ACO algorithm.

	Sol. (A)	Time (A) (sec.)	Sol. (B)	Time (B) (sec.)	Sol. (C)	Time (C) (sec.)
W	-1275	-531	-703	-979	-535	-1065
n	50	38	37	50	34	50
z	-6.15	-3.85	-5.3	-4.72	-4.57	-5.14
p (1-tail)	<.0001	0.0001	<.0001	<.0001	<.0001	<.0001
p (2-tail)	<.0001	0.0001	<.0001	<.0001	<.0001	<.0001

Table 5: Results from the Wilcoxon rank-sum test on the raw data

	A vs D	B vs D	C vs D	A vs C	A vs B	B vs C
W	-136	-120	-100	-91	-78	-21
n	16	15	14	13	12	10
z	-3.5	-3.39	-3.12	-3.16	-3.04	-1.04
p (1-tail)	0.0002	0.0003	0.0009	0.0008	0.0012	0.1492
p (2-tail)	0.0005	0.0007	0.0018	0.0016	0.0024	0.2983

Table 6: Results from the Wilcoxon rank-sum test to the averaged solutions

	A vs D	B vs D	C vs D	A vs C	A vs B	B vs C
W	-9	-22	-34	31	-21	40
n	6	7	8	9	10	12
z	-	-	-	-	-1.04	1.55
p (1-tail)	-	-	-	-	0.1492	0.0606
p (2-tail)	-	-	-	-	0.2983	0.1211

Table 7: Results from applying the Wilcoxon rank-sum test to the averaged times

and the standard ACO algorithm. All algorithms appear to scale up linearly (r^2 values very close to 1), with no major differences between slopes.

6 Conclusion

As has been discussed in section 5.3, all three algorithms provide a definite improvement over standard ACO. More precisely, TransKernelized Vertex Cover ACO has been found to find significantly better solutions over the other three algorithms, but PreKernelized Vertex Cover ACO and CycleKernelized Vertex Cover ACO still find better solutions over standard ACO. Clearly using kernelization rules as a kind of template under all three strategies provides a definite improvement over not using any template at all.

However, both PreKernelized Vertex Cover ACO and CycleKernelized Vertex Cover ACO appeared to be able to find solutions quicker than TransKernelized Vertex Cover ACO (which could find solutions only marginally quicker than traditional ACO), although more data needs to be collected to establish this definitively.

Since TransKernelized Vertex Cover ACO signicantly outperforms all algorithms in terms of solution whereas CycleKernelized Vertex Cover ACO and PreKernelized Vertex Cover ACO appear to perform better in terms of time, this suggests the trade-off between ant complexity and quality of solutions. Since TransKernelized Vertex Cover ACO uses more complex ants, it is able to find better solutions at the trade-off of spending more time generating solutions. However, since all three algorithms outperform standard ACO in both solution and time, there is no tradeoff between whether to use kernelization as a kind of template or not. It is always beneficial to use kernelization over not.

Although TransKernelized Vertex Cover ACO is definitely best for this particular application, as a metaheuristic it might not be applicable to all combinatorial optimisation problems (COPs) or all applications of these COPs. It may turn out that CycleKernelized Vertex Cover ACO or PreKernelized Vertex Cover ACO are the most useful out of all three algorithms because they are better suited to different problems and applications. It is also interesting to note than PreKernelized Vertex Cover ACO did contain significant gains in both solution quality and time through a simply kernelization pre-processing step before running ACO. This algorithm conforms most closely to the original ants with templates idea. The pheromone laid by the kernelization step at the beginning forms a template that acts as a guide along which the ants can perform stigmergy and autocatalysis to find solutions.

This paper represents the beginning of an exploration of this type of combination of heuristic and reduction approaches to COPs. The most immediate area for future work is a more fine-grained analysis of the vertex cover experimental data. We are interested in determining, beyond our informal intuitions, why each method performed as it did. In addition, there is recent work on the notion of phase transition for vertex cover (Hartmann & Weigt 2003) which may be useful in classifying problem instances in a way that will allow a more detailed understanding of the methods' performance.

Further avenues for future work include investigating the application of our work to other COPs and trying to integrate more kernelization rules into the algorithms to see if this results in further improved solutions. More interestingly, this might also involve the development of quasi-kernelization rules: that is, kernelization rules that reduce a problem in complexity but only guarantee with high probability that an optimal solution can be found (as opposed to the absolute guarantee that our current rules provide).

References

Alber, J., Fellows, M. R. & Niedermeier, R. (2002), 'Efficient data reduction for dominating set: A linear problem kernel for the planar case', *Proceeeings of the 8th Scandinavian Workshop on Algorithm Theory* pp. 150–159.

Bansal, N. & Raman, V. (1999), Upper bounds for maxsat: Further improved, *in* 'Proceedings of the 10th International Symposium on Algorithms and Computation', Springer-Verlag, pp. 247–258.

Bodlaender, H. L., Downey, R. G., Fellows, M. R., Hallett, M. T. & Wareham, H. T. (1995), 'Parameterized complexity analysis in computational biology', *Computer Applications in the Biosciences (CABIOS)* **11**(1), 49–57.

Bonabeau, E., Dorigo, M. & Theraulaz, G. (1999), *Swarm Intelligence From Natural to Artificial Systems*, A volume in the Santa Fe Institute studies in the science of complexity., Oxford University Press.

Buss, J. F. & Goldsmith, J. (1993), 'Nondeterminism within P', *SIAM J. Comput.* **22**(3), 560–572.

Caro, G. D. & Dorigo, M. (1998), 'Antnet: Distributed stigmergetic control for communications networks', *Journal of Artificial Intelligence Research* **9**, 317–365.

Chen, J., Kanj, I. & Jia, W. (1999), 'Vertex cover: Further observations and further improvements', *Proceedings of the Workshop on Graph-Theoretic Concepts in Computer Science* pp. 313–324.

Dorigo, M., Caro, G. D. & Gambardella, L. M. (1999), 'Ant algorithms for discrete optimization', *Proceedings of Artificial Life 5* pp. 137–172.

Dorigo, M. & Gambardella, L. M. (1997), 'Ant colonies for the traveling salesman problem', *BioSystems* **43**, 73–81.

Dorigo, M., Maniezzo, V. & Colorni, A. (1991), Positive feedback as a search strategy, Technical report 91-016, Dipartimento Di Elettronic, Politecnico Di Milano.

Downey, R. (2003), 'Parameterized complexity for the skeptic', *Proceedings of the 18th IEEE Annual Conference on Computational Complexity* pp. 147–169.

Downey, R., Fellows, M. & Stege, U. (1997), 'Parameterized complexity: A framework for systematically confronting computational intractability', *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* .

Fellows, M. (2002), 'Parameterized complexity: The main ideas and connections to practical computing', *Electronic Notes in Theoretical Computer Science* **61**.

Gambardella, L. M., Taillard, E. & Agazzi, G. (1999), MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows, Technical report, Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale.

Hartmann, A. K. & Weigt, M. (2003), 'Statistical mechanics of the vertex-cover problem', *Journal of Physics A: Mathematical and General* **36**(43), 11069–11093.

Maniezzo, V. & Colorni, A. (1999), 'The ant system applied to the quadratic assignment problem', *Knowledge and Data Engineering* **11**(5), 769–778.

Niedermeier, R. & Rossmanith, P. (1999), 'Upper bounds for vertex cover further improved', *Lecture Notes in Computer Science* **1563**, 561–570.

Shyu, S. J., Yin, P.-Y., Lin, B. M. T. & Hsiao, T. S. (2001), 'An ant colony optimization algorithm for the minimum weight vertex cover problem', *Annals of Operational Research* **131**, 283–304.

Stege, U. & Fellows, M. (1999), An improved fixed-parameter-tractable algorithm for vertex cover, Tech. report 318, Department of Computer Science, ETH Zurich.