Kernelization as Heuristic Structure for the Vertex Cover Problem

Stephen Gilmour and Mark Dras

Department of Computing, Macquarie University, Sydney, Australia
{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 exploited 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 [1]. 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 [2,3] for identifying problem structure as part of an exact solution, the extent to which these techniques can be integrated into heuristics, and what advantage this gives over a standard heuristic approach.

In section 2 of this paper, we discuss ant colony system (ACS) for the vertex cover problem. In section 3 we discuss parameterized complexity and kernelization. In section 4 we present our framework for integrating ACS with kernelization. In section 5 we give experimental results for our new algorithms and in section 6 we conclude.

2 ACS for the Minimum Vertex Cover Problem

In this section we will present an algorithm by Gilmour and Dras [4] for ant colony system on the vertex cover problem (VCP). Given the graph $G = (V, E)$,

M. Dorigo et al. (Eds.): ANTS 2006, LNCS 4150, pp. 452–459, 2006.

the minimum VCP is the problem of finding a set of nodes $V' \in V$ 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 [6] for the weighted vertex cover problem. In section 2.1 we discuss the problem representation as defined by Shyu *et al.* [6] 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.

2.1 Problem Representation

Shyu *et al.* note that, unlike the Traveling Salesman Problem (TSP) for which the first ACS algorithm 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 unrepeated adjacent vertices, which covers exactly and only the vertices in V'" [6]. 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

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

2.2 Random Proportional Transition Rule

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

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

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:

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

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. A solution is constructed when all edges are covered. But, before the next cycle can continue, the connectivity values need to be reset according to equation (1).

2.3 Pheromone System

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, the global and local pheromone update rules need to be updated to work with nodes. Our global pheromone update rule, which is executed by one ant at the end of each cycle, is defined as:

$$\tau_i(t) \leftarrow (1-\rho) \cdot \tau_i(t) + \rho \cdot \Delta\tau_i(t) \tag{4}$$

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:

$$\tau_i = (1-\varphi)\tau_i + \varphi\tau_0 \tag{5}$$

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.

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.

3 Parameterized Complexity

An overview of parameterized complexity is available in [2,3]; 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, the k-vertex cover problem 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 by the maximum size of the vertex cover that we are trying to find; in such a case, there is a best-case algorithm with complexity $O(kn + 1.2852^k)$ [7] that is tractable for k up to 400. 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 in polynomial time to its problem kernel such that ideally, 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.

Many different optimization problems [3] have been analyzed using the notions of parameterized complexity, for example the dominating set problem, traveling 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 Ant Colony System with Structure

The key idea in this paper is that kernelization can be used to give ACS 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. The rule we have chosen to use is: "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" [2]; here $N(v)$ denotes the set of vertices that form the neighbourhood of v and $N[v]$ denotes $N(v) \cup \{v\}$. See [8] for a more detailed discussion on how we came to use this rule.

Kernelized Ant Colony System. This algorithm performs kernelization within the initialisation stage of the algorithm and then runs regular 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. Within the kernelization phase we remove from the graph all the nodes that belong to χ and all edges connected to nodes in χ.

PreKernelized Ant Colony System. This algorithm works similarly to Kernelized ACS except rather than removing nodes from the graph, it sets the pheromone on the selected nodes to be τ_{kern}. This will initially make the nodes in the kernelization set more attractive to ants than any other nodes in the problem.

CycleKernelized Ant Colony System. This algorithm is similar to PreKernelized ACS except that the kernelization information is continually reinforced in pheromone. Therefore we have moved the kernelization component out of the initialisation phase and into the global pheromone update rule. 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.

KernelAnts Ant Colony System. ACS uses m ants to generate solutions to a problem. However, KernelAnts ACS uses k additional ants 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 to τ_{kern}; this node is selected by choosing the node in χ with the smallest pheromone value. This occurs in

parallel with the regular ants continuing to construct solutions to the problem influenced by the pheromone on the graph.

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 as with regular ACS. 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 regular ACS.

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 that picks a node using the regular 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.

5 Evaluation

5.1 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 and performed parameter analysis on ant colony system and our six new algorithms for the vertex cover problem. We timed how long ant colony system took to complete 200[1] 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.

5.2 Challenging Benchmarks

Benchmarks with Hidden Optimum Solutions for Graph Problems[2] is a website with a collection of challenging instances of graph problems constructed by hiding optimum solutions for a specific problem in hard graphs [9]. 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

[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

Table 1. Good parameters for ant colony system and our six new algorithms. 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 τ_{kern}.

	ACS	KACS	PKACS	CKACS	KAACS	TKACS	NTKACS
m	50	40	40	40	40	10	40
k	-	-	-	-	10	-	-
β	4	5	5	4	4	5	5
ρ	0.1	0.1	0.1	0.1	0.1	0.2	0.1
q_0	0.9	0.9	0.9	0.9	0.9	0.999	0.9
τ_{kern}	-	-	0.5	0.5	0.95	-	-

Table 2. 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	Opt
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

period of time. Table 2 contains the sum and average of the results for each algorithm. We have adopted the approach of Birattari [10], of using a maximum number of instances possible with just one run per instance for all experimentation. See [8] for a more detailed discussion of all experiments and results.

We applied the Mann-Whitney U-test—recommended for use in metaheuristic analysis [11]—to these results and made the following conclusions. Firstly, all algorithms including our ACS algorithm were statistically significant improvements over the algorithm by Shyu *et al.*. Secondly, all kernelization algorithms were a statistically significant improvement over regular 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 TransKernelized ACS. The primary conclusion from these results is that the kernelized algorithms do outperform regular ACS algorithms.

5.3 Random Graphs

We constructed two groups of graphs of 500 nodes each. The first group of graphs contains graphs with 100 to 500 nodes, the second 600 to 1000 nodes. Each algorithm ran on each graph for the quantity of time required for ACS to perform 200 iterations on that graph. All random graphs were generated using the algorithm proposed by Skiena [12] and selected to contain a variety of parameters for number of nodes, number of edges, and kernel sizes. Again only one run per graph was performed.

Table 3. 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
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. 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
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 3 contains the sum and average of the results for each algorithm for the first group of graphs. 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 regular 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 4 contains 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 made the following conclusions. Firstly, all algorithms were a statistical improvement over the algorithm by Shyu *et al.*. Similarly, all kernelization algorithms were a statistical improvement over regular ACS. Secondly, Neighbourhood TransKernelized ACS was an improvement over all other algorithms and TransKernelized ACS a clear second.

6 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 pheromone based algorithms consume less CPU time and so more iterations of ACS can be performed which is benefical for these hard problems. However, the algorithms with kernelization integrated into the random proportional transition rule work better on the random graphs; probably because the random graphs are not quite as hard and so more time can be used performing kernelization. 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.

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. Blum, C.: Ant colony optimization: Introduction and recent trends. Physics of Life Reviews **2** (2005) 353–373
2. Downey, R., Fellows, M., Stege, U.: Parameterized complexity: A framework for systematically confronting computational intractability. DIMACS Series in Discrete Mathematics and Theoretical Computer Science (1997)
3. Downey, R., Fellows, M.: Parameterized Complexity. Monographs in Computer Science. Springer-Verlag (1998)
4. Gilmour, S., Dras, M.: Understanding the Pheromone System within Ant Colony Optimization. Australian Conference on Artificial Intelligence (2005) 786–789
5. Dorigo, M., Stützle, T.: Ant Colony Optimization. A Bradford Book. MIT Press (2004)
6. Shyu, S.J., Yin, P.Y., Lin, B.M.T.: An Ant Colony Optimization Algorithm for the Minimum Weight Vertex Cover Problem. Annals of Operational Research **131** (2004) 283–304
7. Chen, J., Kanj, I., Jia, W.: Vertex cover: Further observations and further improvements. Journal of Algorithms **41** (2001) 280–301
8. Gilmour, S., Dras, M.: Exactness as Heuristic Structure for Guiding Ant Colony System. Technical report, Department of Computing, Macquarie University, Australia (2006)
9. Xu, K., Boussemart, F., Hemery, F., Lecoutre, C.: A simple model to generate hard satisfiable instances. Proceedings of 19th International Joint Conference on Artificial Intelligence (IJCAI) (2005) 337–342
10. Birattari, M.: 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 (2005)
11. Taillard, E.D.: Comparison of non-deterministic iterative methods. MIC'2001 - 4th Metaheuristic International Conference (2001) 272–276
12. Skiena, S.S.: The algorithm design manual. Springer-Verlag New York, Inc., New York, NY, USA (1998)