The Backbone of the Travelling Salesperson

Philip Kilby
ANU
Canberra, Australia
Philip.Kilby@anu.edu.au

John Slaney
NICTA and ANU
Canberra, Australia
John.Slaney@anu.edu.au

Toby Walsh
NICTA and UNSW
Sydney, Australia
tw@cse.unsw.edu.au

Abstract

We study the backbone of the travelling salesperson optimization problem. We prove that it is intractable to approximate the backbone with any performance guarantee, assuming that P≠NP and there is a limit on the number of edges falsely returned. Nevertheless, in practice, it appears that much of the backbone is present in close to optimal solutions. We can therefore often find much of the backbone using approximation methods based on good heuristics. We demonstrate that such backbone information can be used to guide the search for an optimal solution. However, the variance in runtimes when using a backbone guided heuristic is large. This suggests that we may need to combine such heuristics with randomization and restarts. In addition, though backbone guided heuristics are useful for *finding* optimal solutions, they are less help in *proving* optimality.

1 Introduction

In recent years, there has been much research into what makes a search problem hard. In decision problems, we now have a rich picture based upon rapid transitions in solubility and a corresponding thrashing in search algorithms for the critical constrained problems close to such “phase transitions” [Cheeseman *et al.*, 1991; Mitchell *et al.*, 1992]. The picture is much less clear for optimization. Optimization problems do not have a transition in solubility, since optimal solutions always exist. A strong candidate to replace the transition in solubility is a transition in the backbone size [Monasson *et al.*, 1998]. In this paper, we study the complexity of computing and of approximating the backbone. We also look at using such backbone information to help find optimal solutions, and to prove optimality. Throughout this paper, we focus on the symmetric travelling salesperson (TSP) optimization problem. This is the problem of computing the shortest tour which visits every city, where the inter-city distance matrix is symmetric. However, most of our results easily generalise to the asymmetric case.

2 Backbones

In decision problems, the concept of the *backbone* has proven to be very useful in understanding problem hardness. For example, the backbone of a satisfiability (SAT) problem is the set of literals which are true in every model [Monasson *et al.*, 1998]. Backbone size appears closely correlated to problem hardness [Parkes, 1997; Monasson *et al.*, 1998]. If a SAT problem has a large backbone, there are many opportunities to assign variables incorrectly. Such problems tend to be hard therefore for systematic methods like Davis-Putnam. A large backbone also means that solutions are clustered. Such problems therefore can be hard to solve with local search methods like WalkSAT.

Backbones have been less well studied in the context of optimization. In [Slaney and Walsh, 2001], the backbone of an optimization problem is defined to be the *frozen decisions*: those with fixed outcomes for all optimal solutions. For example, the backbone of a TSP problem is the set of edges which occur in all tours of minimal cost. Such a concept again seems useful in understanding problem hardness. For instance, the cost of finding optimal (or near optimal) solutions is positively correlated with backbone size [Slaney and Walsh, 2001].

3 Computing the Backbone

It is not difficult to see that it is NP-hard to find the backbone of a TSP problem. If the optimal tour is unique, then the backbone is complete. A procedure to compute the backbone then trivially gives the optimal tour length. Complications arise when the optimal tour is not unique and the backbone is incomplete. For example, suppose there are two disjoint but optimal tours. The backbone is then empty. Nevertheless, we can find the optimal tour length using a polynomial number of calls to a procedure that determines backbone edges.

Theorem 1 *The* TSP BACKBONE *problem (the problem of deciding if an edge is in the backbone of a* TSP *problem) is NP-hard.*

Proof: We rescale the TSP problem so that only one optimal tour remains. Without loss of generality, we assume that inter-city distances are integers. First, for a problem with n cities, we multiply each inter-city distance by $2^{n(n+1)}$. This gives $n(n+1)$ new bits in the least significant digits of each number. Note that this only requires polynomial space. We divide these new bits into n sections, one for each city, of length $n+1$ bits. For the inter-city distance between city i and city j, we set the jth bit in the ith section and the ith bit in the jth section. Each tour length now encodes the list of edges visited. In particular, the ith section of the tour length has two bits, say, j and k set. This represents the fact that the tour includes edges from j to i and then to k. Even if the old distance matrix had several optimal tours, the new distance matrix only has one of them. Let c_i be the two cities on a tour connected to the ith city. Then the optimal and unique tour of the rescaled problem is the optimal tour of the unrescaled problem in which $\langle \max(c_1), \min(c_1), \ldots, \max(c_n), \min(c_n)\rangle$ is lexicographically least. As the optimal tour is now unique, the backbone is complete. We can compute this by a polynomial number of calls to a procedure for deciding if an edge is in the backbone. $\Box$

It remains open whether the TSP BACKBONE problem is NP-complete or not. Completeness would seem to require a short witness that a tour was optimal. We can, however, say it is both NP-hard and NP-easy. It is NP-easy as deciding if an edge is in the backbone can be reduced to a polynomial number of calls to a TSP decision procedure. Garey and Johnson suggest that problems which are both NP-hard and NP-easy might be called NP-equivalent [Garey and Johnson, 1979]. Although this class contains problems which do not belong to NP, the class has the property of NP-complete decision problems that: unless P=NP, no problem in the class can be solved in polynomial time, and if P=NP then all problems in the class can be solved in polynomial time. In other words, the TSP BACKBONE problem is polynomial if and only if P=NP.

4 Approximating the Backbone

Even though computing the backbone is intractable in general, might we be able to approximate it?

4.1 Sound approximation

Suppose that we have an approximation procedure that returns some subset of the backbone edges. It is easy to see that, assuming P $\neq$ NP, no such procedure can be guaranteed to return a fixed fraction of the backbone in polynomial time: First, we rescale the distance matrix as before so that the optimal tour is unique and the backbone complete. The sound approximation procedure could be called to return at least one backbone edge - say (a, b). Create a new TSP by collapsing a and b into a single node a'. The cost $c(a', x)$ for node x would be set to MIN $(c(a, x), c(b, x))$; all other costs as before. The procedure could be repeatedly called to construct the optimal tour edge by edge in polynomial time.

A similar argument shows that no sound approximation procedure can be guaranteed to return at least one backbone edge if the backbone is non-empty in polynomial time.

4.2 Unsound approximation

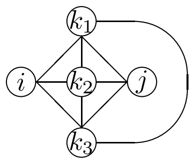
Suppose that edges returned by an approximation procedure are not guaranteed to be in the backbone. If we do not limit the number of edges incorrectly returned, then there exists a polynomial time approximation that meets any approximation ratio (that is, returns any given fraction of the backbone). For example, the approximation procedure that returns all $O(n^2)$ possible edges returns all the backbone. We therefore consider approximation procedures which limit the number of edges falsely assigned to the backbone. An approximation procedure is a "majority-approximation" iff, when the backbone is non-empty, more edges are returned that come from the backbone than do not come from the backbone. If the backbone is empty, any number of edges can be falsely returned.

Theorem 2 *If P $\neq$ NP then no majority-approximation procedure can be guaranteed to return a fixed fraction α or greater of the backbone edges in polynomial time.*

Proof: We show how such a procedure could determine if a graph (V, E) has a Hamiltonian path with a designated starting and ending vertex in polynomial time, contradicting P $\neq$ NP. We construct a TSP problem which can have two sorts of shortest tours. For every Hamiltonian path between the starting and ending vertex, there is a corresponding tour of length $n-1$ where $|V| = n$. These tours have a non-empty set of backbone edges taken from some set S. On the other hand, if there is no Hamiltonian path between the starting and ending vertex, the shortest tour is of length n, and has a non-empty set of backbone edges (denoted T) disjoint with S.

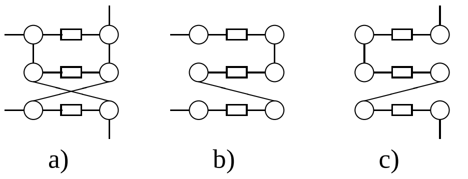
Since the approximation procedure returns at least a fixed fraction of the backbone, it returns at least one correct backbone edge. As it is a majority-approximation procedure, the number of edges in the backbone correctly returned is more than the number incorrectly returned. By computing the ratio of the number of edges of the two types returned, we can determine if the backbone is drawn from edges in S or from those in T. That is, we can determine if there is a Hamiltonian path or not in polynomial time.

We need two special gadgets. The first is a "backbone free" connector. This connects together two nodes without introducing any backbone edges. For example, to connect node i to node j, we use the following small circuit which introduces 3 new intermediate nodes: k_1, k_2 and k_3. All marked edges are of cost 0, and all unmarked edges are of cost n^2:



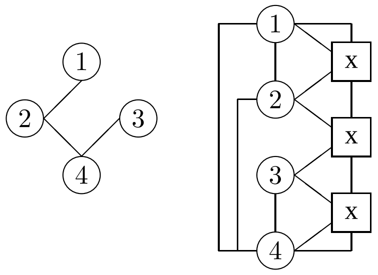
A tour from i to j goes through some permutation of k_1, k_2, k_3. No edge is in all possible tours, and thus no edge appears in the backbone. Note that we can modify this (and the other circuits) to have non-zero edges costs by increasing all other edge costs appropriately. We will draw this gadget as a rectangular box between nodes i and j.

The second gadget is a "switch" circuit. This uses three of the backbone free connector gadgets. Four edges go into this circuit, two horizontally and two vertically. The switch has two modes. In the first mode, the tour enters and exits by the vertical edges. In the other mode, the tour enters and exits by the horizontal edges. The switch circuit contains 15 nodes, all of which are visited in both modes. Note that no backbone edges are common between the two modes. The gadget is show below in a). As before, all marked edges are of cost 0, and all unmarked edges are of cost n^2. We give the two different modes of the circuit in b) and c). We will draw this switch gadget as a square box containing an "x".



We use these gadget to construct a TSP problem with the required shortest tours. The problem has $16(n-1)+1$ nodes, n of which correspond to vertices in the graph (V, E) and the rest are in $n-1$ switch gadgets. If the graph (V, E) has an edge between vertex i and j then the TSP problem has an edge of cost 1 between node i and j. We shall assume that the starting and ending vertices/nodes are 1 and n respectively. To finish the tour, we put a zero cost edge between node 1 and n. There are also $n-1$ switch circuits. The horizontal wires in the ith switch circuit are connected to node i and $i+1$ $(1 \leq i \leq n-1)$. The vertical wires in the jth switch circuit are connected to the $j-1$th and $j+1$th switch circuit $(1 < j < n-1)$. The vertical wires in the 1st switch circuit are connected to node 1 and the 2nd switch circuit. The vertical wires in the $n-1$th switch circuit are connected to the $n-2$th switch circuit and to node n. All edge costs between switch circuits and nodes, and between switch circuits are zero except for the edge between node 1 and the horizontal input to the 1st switch circuit which has cost n. The cost between any two unmarked nodes is n^2 as before.

We give an example for $n = 4$ in which the graph (V, E) has edges between nodes 1 and 2, 3 and 4, and 2 and 4:



If there is a Hamiltonian path in the graph, we follow the corresponding path in the TSP starting with node 1 and ending at node n. We then exit from node n, and enter the vertical wire in the $n-1$th switch, and continue through the vertical wires to switch 1 where we exit and take the zero cost edge back to node 1. This completes a tour of length $n-1$. On the other hand, if there is not a Hamiltonian path, the shortest tour visits nodes 1 to n by alternating with the $n-1$ switch circuits. It then takes the zero cost edge from node n back to node 1. This completes a tour of length n. $\square$

Similar arguments show that if P $\neq$ NP then no majority-approximation procedure can be guaranteed to return at least one backbone edge when the backbone is non-empty in polynomial time

5 Epsilon Backbone

Despite these negative complexity results, backbones may still be easy to compute in practice. We study here the possibility of using approximation procedures. We define the ϵ-backbone of a TSP problem as the set of edges which occur in all tours within a factor $(1+\epsilon)$ of the optimal. In Figure 1, we plot the fractional size of the ϵ-backbone (that is, the size of the ϵ-backbone normalized by n) against $1+\epsilon$ for random Euclidean TSP problems.

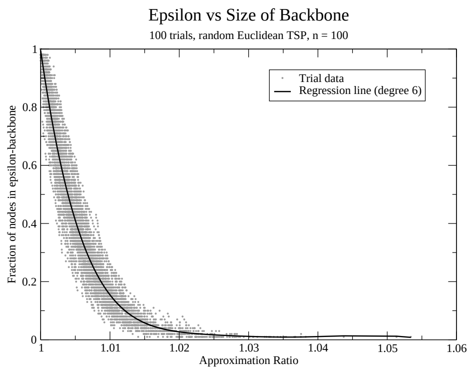


Figure 1: Fractional size of the ϵ-backbone (y-axis) plotted against approximation ratio (= $1+\epsilon$) for 100 random Euclidean TSP problems with 100 nodes. Degree 6 regression line fitted.

We see that tours within 5% of optimal have approximately 40% of the backbone of optimal tours. We observe similar results with non-random instances taken from TSPLib. It appears that much of the backbone of a TSP problem is present when we are near to the optimal solution. As heuristics can often find near optimal solutions in a short time, it may be easy to compute a large part of the backbone in practice.

6 Approximation Methods

We now see if TSP heuristics can be used as the basis of an approximation method for computing the backbone. To compute if an edge is in the backbone or not, we can commit to the edge and compute the optimal tour, and then throw the edge out and re-compute the optimal tour. The edge is in the backbone iff the first tour is shorter in length than the second. Suppose now that instead of computing the two optimal tour lengths, we run some good heuristic method like [Lin and Kernighan, 1973] for a polynomially bounded time. If it is true that the heuristic is good, it will frequently find close to optimal length tours (though we have no guarantee that it does). This yields a simple approximation method for computing edges that are likely to be in the backbone.

Tests were run on random Euclidean TSP problems of size 100, 250 and 500 nodes. Most problems have unique solutions, and hence complete backbones. Exceptions were: 6 problems of size 100, 16 of size 250 and 19 of size 500 had backbones of size $< n$. Table 1 shows the percentage of the backbone correctly identified using the heuristic proceedure, as well as the percentage of "false positive" results. The Lin-Kernighan heuristic is very accurate at these problem sizes, so

		n = 100	n = 250	n = 500
Ave Approx Ratio		1.0000	1.0002	1.0008
Correct	10%	99.0	69.4	28.8
(%)	Median	100.0	100.0	75.8
	90%	100.0	100.0	100.0
False	10%	0.0	0.0	0.0
Positive	Median	0.0	0.0	0.0
(%)	90%	1.0	0.4	0.6

Table 1: Estimation of backbone using the Lin-Kernighan heuristic. 100 trials of random Euclidean TSP problems at each size. Approx Ratio is mean (heuristic-solution)/(optimal-solution) of original problem.

approximation ratios were small. The heuristic is very good at identifying backbone edges and gives few false-positives. Its main weakness is missing backbone edges.

7 Backbone Guided Heuristics

One motivation for identifying backbone is to try to reduce search. For example, Climer and Zhang use backbone variables in asymmetric TSP problems to preprocess and simplify [Climer and Zhang, 2002]. As a second example, Dubois and Dequen use a backbone guided heuristic to solve 700 variable hard random 3 SAT problems [Dubois and Dequen, 2001]. In the following section, we show that whilst backbone guided heuristics can be helpful in finding optimal solutions, we must use them with care. Median runtime are good, but there are a few long runs (suggesting that a randomization and restarts strategy may be useful). In addition, we show both experimentally and theoretically, that backbone guided heuristics can be very poor when it comes to proving optimality. To make the demonstration very direct, we use a method for solving TSP problems that relies heavily on the branching heuristic. The method does not use the "cuts" that characterise state-of-the-art solvers. However, it lets us compare directly the efficacy of various branching decisions.

We use a branch and bound solver with lower bounds provided by Lagrangean relation with 1-trees [Reinelt, 1994]. Upper bounding is by a single, deterministic run of Or-opt [Or, 1976] testing all forward and reverse moves of blocks of size $n/2$ down to 1. The MST bound is enhanced by eliminating all other edges into a node if two incident edges are already forced into the solution. Depth-first search is used, with the node having the least lower bound explored. The algorithm branches on an edge currently in the tour representing the upper bound. The edge can be chosen in one of several ways:

bb We exclude in turn every possible edge from the solution. A comparison is then made between the new objective and the current upper bound. If the new objective is reduced, then an improvement to the upper bound has been found. If such an improving move is found, the edge giving the largest improvement is chosen. If no such improvement is found, then the edge giving the largest *increase* in objective is most likely to be in the backbone, so that edge is chosen.

freqbb A frequency table is kept during search. As each new upper-bound tour is created, the frequency of appearence of each edge is updated. Edges with higher frequency are more likely to be backbone edges, so the branch choice is the edge with the greatest frequency.

next The next unused edge in the tour is chosen

long The longest unused edge in the tour is chosen

short The shortest unused edge in the tour is chosen

longish A method suggested by the results of this test. Use *long* for the first n branches, then use *next*.

rand A random edge is chosen. As this is non-deterministic, the heuristic was run five times on each problem.

	Mean		Median	
Method	Times/s	Nodes	Time/s	Nodes
bb	60.9	685.8	3.0	5.0
freqbb	98.5	1618.1	5.0	174.0
long	34.0	436.8	0.0	4.0
longish	21.8	332.0	0.0	5.0
next	38.0	642.8	1.0	19.0
rand	58.1	980.5	2.0	40.0
short	61.1	909.8	7.0	113.0

Table 2: Branching heuristics – finding the optimal solution

	Mean		Median		limit
Method	Time/s	Nodes	Time/s	Nodes	exceeded
bb	199.1	2559.8	35.0	244.0	12
freqbb	206.5	3284.9	28.5	422.0	14
long	360.3	4816.7	51.6	755.0	26
longish	87.7	1946.1	12.5	130.0	4
next	93.7	1775.3	10.1	108.0	4
rand	179.1	2859.3	24.4	348.0	10
short	199.8	2921.9	35.8	413.0	11

Table 3: Branching heuristics – proving optimality

A branch-and-bound solver has two phases: finding the optimal solution, and proving optimality. Different heuristics do better in different phases, so we report the results separately in Table 2, and Table 3. We used 50 node random Euclidean TSP problems with a time limit of 1000 seconds on each run. Table 3 shows the number of times this limit was exceeded. The time and node results *include* problems for which the time limit was exceeded.

These results show a "heavy-tailed" distribution [Gomes *et al.*, 2000] with many problems being solved quickly (hence a low median) but a few taking a very long time (hence a high average). Randomised algorithms and restarts have been shown to reduce the average time in NP-hard problems [Meisels and Kaplansky, 2004; Selman *et al.*, 1994]. However, we wished to look at the "pure" algorithms for comparison.

These results show that the bb heuristic is quite effective at finding the optimal in terms of the median number of nodes visited. However, the average is quite large, indicating the method could benefit from multiple restarts. This reduced number of nodes comes at a price in terms of runtime. Other methods, such as *long* outperform it in runtime. In terms of proving optimality, the *bb* heuristic performed relatively poorly, being little better than random.

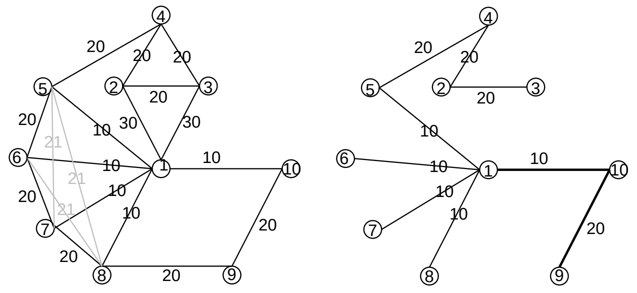
8 Pathological Example

As a further caution to using backbone guided branching heuristics, we present a pathological instance of the TSP problem. If we branch on a non-backbone edge, this instance can be solved in a *single* branch. However, if we branch first on backbone edges, we visit an *exponential* number of branches before proving optimality. Since a search space is defined by a particular algorithm, we must define a solution method. The following algorithm is sensible, but simple enough that we can predict its behavior theoretically. It lacks the refined LP-based cuts that allow modern TSP codes to solve huge problems, but is a fairly standard basic algorithm ([Reinelt, 1994]). It has the following features:

1. Uses branch and bound.
2. Stops when the lower bound equals the upper bound.
3. Upper bound provided by current best tour.
4. Lower bound provided by 1-tree relaxation. A 1-tree is simply a minimum spanning tree (MST) with one extra edge added from a leaf to another node already in the tree. The 1-tree contains exactly one cycle, which may be a complete tour.
5. If two edges are forced to be incident to a single node, then the lower-bounding procedure does not consider any other edges incident to that node.
6. Preprocessing identifies nodes which have only two "useful" incident edges. These two are forced into the solution. A "useful" edge here is defined to be an edge with cost less than any available upper bound (an edge with cost greater than an estimate for the entire tour will never be used in an optimal solution). An initial upper bound can be calculated using the tour (1, 2, 3, ...).

The TSP problem is given in Figure 2(a). Node 6 can be "cloned" arbitrarily many times, and has been cloned at nodes 7 and 8. The clones connect to each neighbour at cost 20, node 1 at cost 10, and all other clones at cost 21. Arcs not shown have cost 1000 + $20n$, and hence will never be used in an optimal solution. The upper bound for this graph is given by the tour 1, 2, .. 10, 1 with cost 200. The minimum spanning tree is shown in Figure 2(b) and has cost 130. The bold edges (10,1) and (10,9) are fixed in due to the preprocessing. The one-tree adds, for example (3,4) at cost 20. The lower bound is therefore 150.

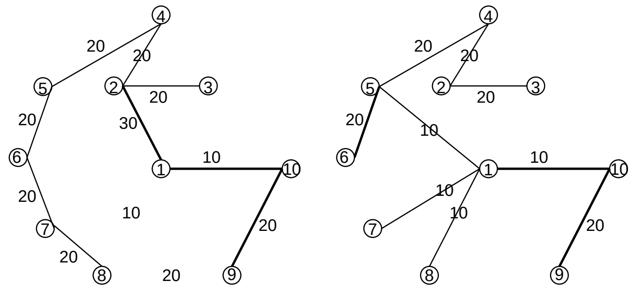
Forcing the non-backbone edge (1,2) excludes the "spoke" edges at nodes 1, as node 1 now has 2 edges incident. The MST (shown in Figure 3(a)) now has cost 180, and the 1-tree bound (using edge (8,9) or (3,4) to connect) is 200. As the lower bound is the same as the upper bound, the branch and bound algorithm completes. So if (1,2) (or, by a symmetric argument (1,3)) were chosen as the branch node, the procedure terminates immediately. On the other hand, branching on backbone edges would force in edges like (5,6). If 6 were cloned as described above, there could be arbitrarily many of these edges. Figure 3(b) shows the effect on the lower bound



(a) TSP problem. Optimal tour is 1,2,...,10 – cost 200

(b) The Minimum Spanning Tree – cost 130

Figure 2: The TSP problem, and its MST



(a) Force (1,2) – MST cost 180, 1-tree bound 200

(b) Force (5,6) – MST cost 120, 1-tree bound 140

Figure 3: Forcing edges. Forced edges in bold.

of forcing edgec (5,6) into the solution. Forcing (5,6) gives an MST with cost 140. The 1-tree is connected using an edge like (6,7) or (3,4) with cost 20, giving a lower bound of 160. If there were other clones of 6, when they were forced into the solution, then the lower bound would not approach the upper bound until edges between *all* clones have been forced. Forcing out (5,6) leaves a problem that looks very similar to the original. It still has backbone edges round the perimeter, but an MST/1-tree which uses the "spoke"-type edges around node 1 and hence gives an unachievable lower bound.

On backtracking, forcing out (5,6) gives the same lower bound as the parent problem. A new upper bound can be found using the cost 21 edges. As the lower-bound is still less than the upper, we can not stop. A recursive argument shows that the right branch must be completely explored before it can be excluded. It is not until all backbone edges are forced in, or forced out, and the algorithm moves on to branch on edge (1,2) or (1,3) that the optimal is deterimend. Hence the backbone heursitic would need to visit an exponential number of branches to solve the problem.

9 Related Work

Slaney and Walsh demonstrated that the cost of finding optimal (or near optimal) solutions is postively correlated with backbone size [Slaney and Walsh, 2001]. However, they also showed that the cost of proving optimality is negatively correlated with backbone size. If we have a small backbone, then there are many optimal and near-optimal tours. An algorithm like branch and bound has to do a lot of work to ensure there are no shorter tours.

Zhang has shown that asymmetric random TSP problems undergo a phase transition in the tour cost and backbone size as the precision of the distance matrix is varied [Zhang, 2004]. He argues that similar results hold for the symmetric TSP problem, and for structured TSP problems provided distances are drawn from a common distribution. The search cost of his branch-and-bound solver also changes from easy to hard as the precision of the distance matrix is increased and the backbone size increases.

Climer and Zhang used an approximation method they call “limit-crossing” to identify backbone variables in asymmetric TSP problems (as well as their dual, “fat” variables which are not part of any optimal solution) [Climer and Zhang, 2002]. By committing to backbone variables and eliminating fat, we can reduce the size of the problem and thereby reduce search.

Beacham has considered the complexity of computing the backbone for a range of decision problems like the satisfiability and Hamiltonian path problem [Beacham, 2000]. He considers a slightly modified definition of backbone: the set of decisions whose negation give an unsatisfiable subproblem. This definition is equivalent to the usual one for satisfiable problems only. He shows that recognising when the backbone is empty is NP-complete.

Zhang has demonstrated experimentally that there is a sharp transition in the size of the backbone of random MAX 3SAT problems [Zhang, 2001]. This appears to be correlated with the transition in the random 3SAT decision problem.

10 Conclusion

We have looked at the backbone of the travelling salesperson problem. It is not hard to see that computing the backbone is NP-hard. However, we have also shown that it is hard to approximate the backbone with any performance guarantee (assuming P$\neq$NP and there is a limit on the number of edges falsely returned). Nevertheless, in practice, it appears that much of the backbone is present in close to optimal solutions. Approximation methods based on good heuristics can often therefore find much of the backbone. Such backbone information can be used to guide the search for an optimal solution. However, it should be used with care as the variance in runtimes when using a backbone guided heuristic is large. In addition, backbone guided heuristics are less good at proving optimality.

What general lessons can be taken from this study? First, these intractability results are likely to generalize to other problems domains. We conjecture that it will also be hard to compute or approximate the backbone in other optimization problems (both fundamental theoretical problems like MAX-SAT, and more practical problems like job shop scheduling). Second, we predict that much of the backbone will also be present in close to optimal solutions in these other domains. Approximation based methods may also work well in these cases. Third, whilst backbones are a useful concept in explaining some aspects of problem hardness in optimization, it is clear that other concepts are still needed. A number of candidate measures have been proposed for decision problems (e.g. backdoor variables) that have yet to be explored for optimization.

References

[Gomes *et al.*, 2000] Carla Gomes, Bart Selman, Nuno Crato, and Henry Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24(1/2):67–100, 2000.

[Lin and Kernighan, 1973] S. Lin and B. Kernighan. An effective heuristic for the traveling salesman problem. *Operations Research*, 21:498–516, 1973.

[Meisels and Kaplansky, 2004] Amnon Meisels and Eliezer Kaplansky. Iterative restart technique for solving timetabling problems. *European Journal of Operational Research*, 153(1):41–50, 2004.

[Or, 1976] I. Or. *Travelling Salesman-Type Combinatorial Problems and Their Relation to the Logistics of Blood-Banking.* PhD thesis, Dept of Industrial Eng. and Management Sciences, Northwest University, Evanston, IL., 1976.

[Reinelt, 1994] Gerhard Reinelt. *The Travelling Salesman. Computational Solutions for TSP Applications, Lecture Notes in Computer Science* vol. 840 Springer-Verlag, 1994.

[Beacham, 2000] A.J. Beacham. The complexity of problems without backbones. Master’s thesis, Dept. of Computing Science, University of Alberta, 2000.

[Cheeseman *et al.*, 1991] P. Cheeseman, B. Kanefsky, and W.M. Taylor. Where the really hard problems are. In *Proc. of 12th IJCAI*, pages 331–337. 1991.

[Climer and Zhang, 2002] S. Climer and W. Zhang. Searching for backbones and fat: A limit-crossing approach with applications. In *Proc. of 18th National Conf. on AI*. 2002.

[Dubois and Dequen, 2001] O. Dubois and G. Dequen. A backbone-search heuristic for efficient solving of hardn 3-SAT formulae. In *Proc. of 17th IJCAI*, pages 248–253. 2001.

[Garey and Johnson, 1979] M.R. Garey and D.S. Johnson. *Computers and intractability : a guide to the theory of NP-completeness*. W.H. Freeman, 1979.

[Mitchell *et al.*, 1992] D. Mitchell, B. Selman, and H. Levesque. Hard and Easy Distributions of SAT Problems. In *Proc. of 10th National Conf. on AI*, pages 459–465. 1992.

[Monasson *et al.*, 1998] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity for characteristic ‘phase transitions’. *Nature*, 400:133–137, 1998.

[Parkes, 1997] A. Parkes. Clustering at the phase transition. In *Proc, of the 14th National Conf. on AI*, pages 340–345. 1997.

[Selman *et al.*, 1994] Bart Selman, Henry A. Kautz, and Bram Cohen. Noise strategies for improving local search. In *Proceedings of AAAI’94)*, pages 337–343, 1994.

[Slaney and Walsh, 2001] J. Slaney and T. Walsh. Backbones in optimization and approximation. In *Proceedings of 17th IJCAI*. 2001.

[Zhang, 2001] W. Zhang. Phase transitions and backbones of 3-SAT and Maximum 3-SAT. In *Proc. of 7th Int. Conf. on Principles and Practice of Constraint Programming (CP2001)*. Springer, 2001.

[Zhang, 2004] W. Zhang. Phase transitions and backbones of the asymmetric traveling salesman problem. *Journal of Artificial Intelligence Research*, 21:471–497, 2004.