Analysis of a Random Cut Test Instance Generator for the TSP

Ronald L. Rardin
School of Industrial Engineering, Purdue University, West Lafayette, IN 47907 USA

Craig A. Tovey
Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332-0205 USA

Martha G. Pilcher
School of Business, University of Washington, Seattle, WA 98105 USA

Abstract

Test Instance Generators (TIG's) are important to evaluate heuristic procedures for NP-hard problems. We analyze a TIG in use for the TSP. This TIG, due to Pilcher and Rardin, is based on a *random cut* method. We show that it generates a class of instances of *intermediate* complexity: not as hard as the entire TSP class unless $NP = co(NP)$; not as easy as P unless $NP = P$. Since the upper bound on complexity must hold for any efficient TIG, our analysis verifies that this random cut TIG is, in a sense, as good as possible a TIG for the TSP. This suggests that the random cut method may be a good basis for constructing TIG's for other problems.

Keywords:
traveling salesman problem, complexity, computational test, test case, problem generation

1 Introduction

A tremendous amount of effort in the past two decades has been directed towards developing good heuristics for NP-hard problems. A good heuristic, classically, has performance ratio v^h/v^* close to 1, where v^h and v^* denote the value of the heuristic and optimal solutions, respectively. Since we have no practical way to find v^* for an arbitrary instance (if we did we wouldn't be resorting to heuristics) we cannot empirically determine v^h/v^* by testing instances generated purely at random. Hence we need a Test Instance Generator (TIG) that supplies instances with known optimal solutions.

Pilcher and Rardin [11, 12] developed a TIG in use for the TSP based on a *random cut* method. This TIG was later extended by Rais and Rardin [13]. We analyze the complexity of the class of instances generated by this TIG. The result is that the TIG essentially does as well as one could hope for. In particular, while it fails to generate a class as difficult as the TSP in general, this is a generic failing of efficient TIG's. We verify that the random cut TIG does generate a class as difficult as possible (for efficient TIG's).

Let us for a moment discuss TIG's in general. These have been considered by Sanchis [14]. Sanchis observes that any TIG designer faces a problem analogous to the cryptographers — how to simultaneously satisfy three conditions:

1. The TIG should generate instances *efficiently*, i.e., it should run in time polynomial in the length of the output.

2. The TIG should generate instances with *known optimal solutions.*

3. The TIG should generate instances that are *hard to solve.*

If one did not enforce condition (2), one could just generate an instance at random. If one did not enforce condition (1), one could just generate an instance at random and solve it by brute force.

Returning to the TSP, let us call the random cut generator the RC TIG. Ideally, to satisfy condition (3), we would like to be able to generate instances as hard as the hardest TSP instances. It is very unlikely that RC accomplishes this; for unless $NP = co(NP)$, its instances are not at the same complexity level as the general TSP. This turns out not to be a specific failing of the RC TIG, but rather an inevitable consequence of satisfying conditions (1) and (2). This is because the language generated by anything satisfying (1) and (2) must be in NP; the "solver" could nondeterministically replicate the generative process and thereby have a succinct proof of optimality. That is, there must be a short computation that solves the instance, namely the computation path of the TIG itself. Therefore, the full complexity of an optimization problem cannot be captured by a TIG satisfying (1) and (2). [See [14] for more details.]

Does this mean that the RC TIG must produce *easy* problems? Surprisingly, the answer is "no". We show that unless $P = NP$, the instances cannot be solved in (deterministic) polynomial time. This is true even if the instance comes with a "promise" that it was created by the generator. At the heart of the analysis is the issue of recognizing valid input.

Taken together, these results imply that the class of instances so generated is of *intermediate* complexity. This depends on distinguishing different questions associated with an instance. The *exact value* question, which asks whether a particular v is the optimal solution value to a given instance, is D^P-complete for general TSP instances (harder than NP-Complete unless $NP = co(NP)$), but only in NP for the instances generated by RC TIG. On the other hand, the search problem, which asks for an optimal solution given an instance, turns out to be NP-hard for the RC TIG instances. Hence, the RC TIG generates instances as hard as can be expected from an efficient TIG.

TIG's with similar properties could be constructed for many other optimization problems for which integer programming formulations are known. We believe this suggests the random cut method is a good basis for effective TIG design, at least according to criteria 1,2,and 3.

2 Versions of the General TSP and Class D^P

The Traveling Salesman Problem is the problem of finding a minimum total weight hamiltonian (vertex-spanning) cycle of a graph. Our interest will always be in the symmetric (undirected) case on a completed graph with $n \triangleq |V|$ vertices and rational edge weights. A formal definition is as follows:

Traveling Salesman Optimization (TSP)

Instance: a complete graph G with rational weights on the edges.

Solution: a minimum total weight hamiltonian cycle of G.

Several different language recognition or decision problems can be derived from this optimization form. The most familiar is the *threshold* decision problem

Traveling Salesman Threshold ($TSP^{\leq}$)

Instance: same as *TSP* plus a rational threshold v.

Question: Does there exist a hamiltonian cycle of G with total weight less than or equal to v?

A related decision problem important to our development is the *exact value* version

Traveling Salesman Exact Value ($TSP^{=}$)

Instance: same as $TSP^{\leq}$.

Question: Does a minimum total weight hamiltonian cycle of G have weight v?

The rest of this section contains elementary background material related to these three versions of the TSP.

We have already alluded to the well-known facts that $TSP^{\leq} \in NP\text{-}Complete$, $TSP^{\leq} \propto TSP$, and thus $TSP \in NP - Hard$ (here and throughout $\propto$ denotes polynomial reduction).

Also notice that the optimization version of the TSP is qualitatively different from the two decision problems in the type of "solution" it demands. The optimization form requires a full optimal solution. Contrast with decision problems $TSP^{\leq}$ and $TSP^{=}$ that call for only a *yes* or *no* response. An algorithm "solves" the latter problems if it can recognize all *yes* cases, i.e. accepts an input if and only if its is a well formed instance for which the corresponding question is properly answered *yes*.

Still, these decision questions are not equivalent. The former is certainly in NP. The latter is probably not because it is complete for complexity class D^P, introduced by Papadimitriou and Yannakakis [8].

A problem in D^P is formed as the intersection of the set of instances of a member of NP and the set of instances of a member of $co(NP)$ (the collections of complements of problems in NP). One example is $TSP^{=}$. An instance of $TSP^{=}$ with $v = \bar{v}$ has proper answer *yes* if and only if the corresponding instance of $TSP^{\leq}$ can be answered *yes*, and the instance with $v = \bar{v} - \delta$ has answer *no*, where δ is the least common denominator of its edge weight denominators. Informally, $TSP^{=}$ is the intersection of $TSP^{\leq}$ with a "translation" of its own complement.

It is also easy to see that $NP \subseteq D^p$ and $co(NP) \subseteq D^p$. To show any problem in NP(respectively $co(NP)$) belongs to D^p, we need only appeand a vacuous $co(NP)$question (respectively NP question).

Papadmitriou and Yannakakis [8] also showed there are D^p-complete problems, i.e. members of D^p to which all problems in D^p reduce in polynomial time. Among these "hardest" members of D^p are the exact value versions of many hard discrete optimization problems, including $TSP^{=}$ on which we are focusing.

We may summarize these facts about the three TSP versions we have so far introduced:

Lemma 2.1 *$TSP^{\leq} \in NP - Complete$, $TSP^{=} \in D^p - Complete$, $TSP \in NP - Hard$, and $TSP^{\leq} \propto TSP^{=} \propto TSP$.*

The following is implicit in [8]:

Lemma 2.2 *If any problem in $D^p - Complete$ also belongs to NP or $co(NP)$, then $NP = co(NP) = D^p$.*

We see from Lemma 2.2 that unless $NP= co(NP)$, $D^p - Complete$ is a higher complexity class that either NP or $co(NP)$. Since $TSP^{=} \in D^p - Complete$, that exact value decision problem is apparently materially different from the threshold version $TSP^{\leq} \in NP$. (Search for TSP is harder still [6].) Figure 1 summarizes in a Venn diagram the containments generally conjectured, and the standing of our three versions of the Traveling Salesman Problem.

Figure 1: Generally Conjectured Complexity Class Containments

3 Polyhedral Relaxations and Random Cut Generators

In this section we present the RC TIG for the TSP. It is based on polyhedral methods, one of the earliest approaches to combinatorial optimization, now experiencing renewed interest (see for example [1]). Given a combinatorial optimization problem OPT, the approach introduces a polynomial-dimension vector of binary decision variables, x, and encodes OPT as the binary linear program:

$$\begin{array}{ll} \text{minimize} & cx \\ \text{subject to} & A_0 x \le b_0 \\ & 1 \ge x \ge 0 \\ & x \text{ integer} \end{array}$$

Here $A_0 x \le b_0$ denotes a system of linear inequalities on solution vectors x satisfied by exactly the binary x feasible for OPT. We make no assumption about the size of the $A_0 x \le b_0$ system relative to the size of OPT.

The *linear programming* (LP) *relaxation* of this formulation, which we denote $ROPT_0$, is formed by deleting the requirement "x integer". If an optimal solution to this relaxation happens also to be integer, it obviously produces an optimum in OPT. If the relaxation optimum is not integer, we may sharpen the formulation by adding new constraint sets

$$\begin{array}{ccc} A_1 x & \le & b_1 \\ \vdots & & \\ A_t x & \le & b_t \end{array}$$

Each system $A_i x \le b_i$ contains new inequalities or "cuts" valid for all binary solutions, but violated by some x feasible in linear programming relaxations. If any of the sharper relaxations

$$\begin{array}{llll} & \text{minimize} & cx & \\ ROPT_k & \text{subject to} & A_i x \le b_i & \text{for all } 0 \le i \le k \\ & & 1 \ge x \ge 0 & \end{array}$$

formed over the new systems ($k \le t$) has an integer optimum, that solution also yields an optimum for OPT.

In the specific case of Traveling Salesman Problems on vertices in V, one integer linear programming formulation is

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i \in V} \sum_{j>i} c_{ij} x_{ij} & \\
\text{subject to} \quad & \sum_{j<i} x_{ji} + \sum_{j>i} x_{ij} \leq 2 \quad \text{for all } i \in V & (1) \\
& -\sum_{i \in V} \sum_{j>i} x_{ij} \leq -|V| & (2) \\
& \sum_{i<j \in S} x_{ij} \leq |S| - 1 \quad \text{for all } S \subset V & (3) \\
& 1 \geq x_{ij} \geq 0 \qquad \text{for all } i < j \in V & (4) \\
& x_{ij} \text{ integer} \qquad \text{for all } i < j \in V &
\end{aligned}
$$

The $O(|V|^2)$ variables x_{ij} indicate whether edges (i, j) are part of the tour. Constraints (1) - (2) require the solution to have degree two at each vertex. Inequalities (3) are the famous *subtour elimination* constraints preventing non-spanning cycles. Together (1) - (3) constitute system $A_0 x \leq b_0$ for TSP.

Many other inequalities are known for the TSP that can form the systems $A_i x \leq b_i$, $i \geq 1$. Among these are the *comb inequalities* [3], and the more general *clique tree inequalities* [4].

These polyhedral considerations led us to propose a random cut generation scheme [10, 11] based on creating instances drawn from the large subset that could, in principle, be solved over an appropriate linear programming relaxation. To be more specific, standard linear programming optimality conditions establish that a vector x^* is optimal in $ROPT_k$ above if there exist dual vectors $u_0, u_1 \ldots, u_k, z, w$ satisfying

$$
\begin{aligned}
& u_i \leq 0 \qquad \text{for all } 0 \leq i \leq k & (5) \\
& u_i(A_i x^* - b_i) = 0 \qquad \text{for all } 0 \leq i \leq k & (6) \\
& z \leq 0, w \geq 0 & (7) \\
& z(1 - x^*) = 0, w x^* = 0 & (8) \\
& c = \sum_{i=0}^{k} u_i A_i + z + w & (9)
\end{aligned}
$$

The procedure for generating instances of OPT with known solution x^* exploits (5)–(9) as follows:

Procedure Random Cut [RC]

1. Randomly chose the feasible set for an instance of OPT, and generate any binary vector x^* satisfying the corresponding $A_0 x \leq b_0$ constraints;

2. Randomly select a sample of main and cut inequalities $A_i^* x^* \leq b_i^*, i = 0, 1, \ldots, k$, for the instance such that all are satisfied as equality at x^* (i.e. $A_i^* x^* = b_i^*$);

3. Randomly generate vectors $\{u_i^* < 0 \quad : i = 0, 1, \ldots, k\}$ of dimension consistent with $\{A_i^*\}$;

4. Randomly generate vector $z^* \leq 0$ with $z_j^* = 0$ for all j with $x_j^* \neq 1$, and vector $w^* \geq 0$ with $w_j^* = 0$ for all j with $x_j^* \neq 0$;

5. Compute $c \leftarrow \sum_{i=0}^{k} u_i^* A_i^* + z^* + w^*$;

6. Return an instance of OPT with feasible set as in Step 1 and objective function vector c;

Lemma 3.1 *The solution x^* chosen at Step 1 of procedure [RC] is optimal for the resulting OPT instance.*

Proof: An instance generated by [RC] will have (binary) x^* optimal because the construction clearly assures it satisfies LP optimality conditions (5)–(9) when all dual u_i components not set at Step 3 are taken as zero. ■

Pilcher [10] employed subtour and comb inequalities in an implementation of [RC] for generating TSP's. Later Rais and Rardin [13] extended the approach to include clique tree inequalities.

The technical challenge in these implementations of the random cut strategy arises in sampling Step 2. The RC method must only use inequalities that are tight for the prospective optimal solution x^*, because complementary slackness constraints (6) forbid nonzero weighting of nontight constraints at Step 3. Implementation of [RC] on any particular problem thus requires a constructive characterization of the tight set at any optimal solution. Subtour elimination constraints provide an easy illustration.

Lemma 3.2 *Subtour elimination constraint (3) for vertex subset S is satisfied as an equality at the tour indexed by x^* if and only if vertices of S are adjacent on that tour.*

Proof: Consider the subgraph of a tour induced by vertex subset S. This subgraph contains no cycles because S is a proper subset of V. Thus well known results in graph theory show its number of edges $\sum_{i<j} x_{ij}$ will total $|S|$ less the number of its connected components. It follows that the corresponding subtour elimination constraint will be active exactly when the subgraph has only one component, i.e. when vertices of S are adjacent on the tour. ■

With the characterization of Lemma 3.2, it is straightforward to implement RC over subtour constraints. Sampling in Step 2 is done simply by choosing intervals off the fixed tour as the S's of tight inequalities (3).

4 Exposed Instances

To analyze RC TIG's, we begin by establishing a simple geometric characterization of the instances that it generates.

Definition. *An instance of a combinatorial optimization problem OPT is exposed for partial polyhedral description k if the corresponding LP relaxation $ROPT_k$ for that instance has an integer optimal solution.*

Informally, an instance is exposed for k it it can be solved by linear programming over $ROPT_k$. Figure 2 illustrates the notion. It shows the complete polytope of a combinatorial optimization problem, along with one LP relaxation $ROPT_0$. The extreme point marked x^* belongs to the underlying OPT feasible set. Every instance of such an OPT consists of a description of its feasible set and a cost vector c. Two different instances are illustrated in Figure 2. The one in part (a) is exposed because c supports the partial polyhedral description at integer solution x^*. Relaxation $ROPT_0$ will yield x^* as an LP optimum. The instance in part (b) is not exposed. Vector x^* is still the integer optimal solution, but a more complete polyhedral description is required before that solution can be found by linear programming.

Although exposed instances can in principle be solved by linear programming, nothing in the definition guarantees it is particularly easy to do so. The number of constraints in $ROPT_k$ need not be polynomial in the size of OPT, even for $k = 0$. Thus, exposed instances are polynomially solvable by linear programming only if suitable separation routines exist for constraints of $ROPT_k$ (see for example [9, chapter 4]). Furthermore, (as we will see), if the $ROPT_k$ optimal solution is not unique, it may still be a difficult task to find an integer optimum among the alternative LP solutions.

Collecting all instances of problem OPT exposed for partial polytope k, we define exposed instance subsets

$$\mathrm{OPT}_k \stackrel{\triangle}{=} \{\text{instances of } OPT : \text{ LP relaxation } ROPT_k \text{ has an integer optimum}\}$$

These exposed instances are exactly the once that can be produced by our RC TIG.

Theorem 4.1 *Each exposed optimization subset OPT_k is exactly the collection of instances of OPT generatable by procedure [RC] over the corresponding $ROPT_k$ polytope.*

Proof: Optimality conditions (5)-(9) are necessary and sufficient for x^* to solve $ROPT_k$. Thus, since procedure [RC] can generate any instance satisfying those conditions, it generates exactly the members of OPT_k. ■

To go further we need to be a bit more precise about the nature of constraints $A_i x \leq b_i$. Such constraint systems are termed *nondeterministically recognizable* if the problem of deciding whether a string constitutes one of the constraints belongs to NP. It is easy to check that most well-known constraints for TSP and other discrete

Figure 2: Exposed Instances of Optimization Problems

problems have polynomial-length derivations, which implies they are nondeterministically recognizable. However, some cases are known, notably the hypohamiltonian constraints of Grötschel and Wakabayashi [1987], that are not believed to possess this property.

Parallelling our TSP notation, we can define threshold and exact value decision problems associated with instance subsets OPT_k:

> **Combinatorial Optimization Threshold ($OPT_k^{\leq}$)**
>
> Instance: an instance in OPT_k plus a rational threshold v.
>
> Question: Does there exist a feasible solution with objective value less than or equal to v?
>
> **Combinatorial Optimization Exact Value ($OPT_k^{=}$)**
>
> Instance: same as $OPT_k^{\leq}$.
>
> Question: Is the optimal objective function value v?

Then we are ready for a main result.

Theorem 4.2 *Given any combinatorial optimization problem OPT for which feasible solutions can be nondeterministically verified, let OPT_k be the exposed subset corresponding to nondeterministically recognizable $ROPT_k$ constraints $A_i x \leq b_i$, $0 \leq i \leq k$. Then the associated threshold and exact value decision problems $OPT_k^{\leq}$ and $OPT_k^{=}$ belong to NP.*

Proof: We must exhibit a nondeterministic polynomial algorithm accepting precisely the language of OPT_k instances and v's for which the relevant questions is properly answered *yes*. That is the algorithm should accept an input exactly when both the optimization instance belongs to exposed subset OPT_k, and its optimal value is $\leq v$ (respectively $= v$).

Given any instance of OPT, whether or not a exposed for k, we can emulate generator [RC] to nondeterministically compute and verify its $ROPT_k$ optimal value, say v_k. Standard LP theory establishes that there must exist a corresponding optimal basis for the dual of LP relaxation $ROPT_k$, consisting of polynomially many $(O(|x|))$ active constraints of the primal. To compute value v_k, we need only guess the nondeterministic derivation/recognition of such a polynomial-size collection of binding constraints, compute nonzero parts of the associated dual basic solution, and verify that it is dual feasible with value v_k.

An OPT instance is exposed for k if and only if its integer solution x^* is an extreme-point of the feasible set for $ROPT_k$, i.e. if and only if the complementary primal solution corresponding to a dual nondeterministically solved in this [RC]-like way is primal feasible. Thus our $NDTM$ for $OPT_k{}^{\leq}$ (respectively $OPT_k{}^{=}$) proceeds by guessing an integer optimum x^* for the given instance of OPT, applying the hypothesized algorithm for nondeterministically verifying its feasibility, computing its objective function value v^*, and nondeterministically solving its $ROPT_k$ dual as just outlined for v_k. We accept exactly when the derived dual solution is complementary with x^* and $v_k \leq v^*$ (respectively $v_k = v^*$). ■

5 Intermediate TSP's

Paralleling the above notation, define

$$\begin{array}{lcl} TSP_{subtour} & \triangleq & \{TSP \text{ instances exposed for subtour constraints}\} \\ TSP_{comb} & \triangleq & \{TSP \text{ instances exposed for comb inequalities}\} \\ TSP_{clique} & \triangleq & \{TSP \text{ instances exposed for clique tree inequalities}\} \end{array}$$

Similarly, let $TSP^{\leq}_{-}$ and $TSP^{=}_{-}$ be the corresponding threshold and exact value decision problems. Then, since all three of the defining constraint forms are nondeterministically recognizable, and feasibility of a TSP solution x can be deterministically verified, we have an immediate corollary to Theorem 4.2.

Corollary 5.1 *Exposed exact value decision problems $TSP^{=}_{subtour}$, $TSP^{=}_{comb}$, $TSP^{=}_{clique}$ all belong to NP.*

A subset of instances of an optimization problem can appropriately be termed "intermediate" if it is plausibly neither as general as the full problem nor polynomially solvable. We are now ready to establish the "upper bound" half of the argument that subsets $TSP_{subtour}$, TSP_{comb}, and TSP_{clique} fulfill this definition by distinguishing their exact value forms from that of the full TSP.

Theorem 5.2 *$TSP^{=}$ does not polynomially reduce to $TSP^{=}_{subtour}$, $TSP^{=}_{comb}$, or $TSP^{=}_{clique}$ unless $NP = co(NP) = D^p$.*

Proof. From Corollary 5.1, exposed decision problems $TSP^{=}_{subtour}$, $TSP^{=}_{comb}$, and $TSP^{=}_{clique}$ all belong to NP. If the D^p-complete problem $TSP^{=}$ polynomially reduced to any of the three, the latter would also be D^p-complete. We know from Lemma 2.2 that a D^p-complete problem can belong to NP only if $NP = co(NP) = D^p$. ■

Perhaps the more surprising half of the argument for intermediate status is the "lower bound" fact that all instances of $TSP_{subtour}$, TSP_{comb}, and TSP_{clique} are polynomially solvable only if $P = NP$. Toward that end define the cost vector of graph G, denoted c_G, to be indicator vector of edges in G (1 if the edge belongs to the graph and 0 otherwise).

Lemma 5.3 *For every hamiltonian graph G, the corresponding instance of TSP with weight vector $-c_G$ belongs to $TSP_{subtour}$, TSP_{comb}, and TSP_{clique}.*

Proof: It is sufficient to prove the result only for $TSP_{subtour}$ because subtour constraints are special cases of both comb and clique tree inequalities. Summing degree-two constraints (1) with weights of $-1/2$ and combining with (2) shows that $-\sum_{i<j} x_{ij} = -|V|$ for every x satisfying (1)–(2). Thus for any graph G

$$-|V| \leq \min\{-c_G x : \ x \text{ satisfying (1)-(4)}\} \leq \min\{-c_G x : \ \text{integer } x \text{ satisfying (1)-(4)}\}$$

When G is hamiltonian, the incidence vector of the implied hamiltonian cycle exactly achieves this lower bound. That is, the instance of TSP with weights $-c_G$ has an integer optimal solution over the subtour constraint polytope, exactly what is required for membership in $TSP_{subtour}$. ■

Theorem 4.1 tells us any member of $TSP_{subtour}$ should be generatable by random cut procedure [RC]. Thus, under Lemma 5.3 there must exist an [RC] generation sequence based on the subtour LP relaxation that yields the (negative) cost vector of any hamiltonian graph. The actual construction begins with the incidence vector of a hamiltonian cycle/tour and places dual multiplier $-\alpha$ ($\alpha > 0$) on all tight subtour inequalities. Summing as in [RC] Step 5, an interim c will have some integer number of copies of $-\alpha$, say $-q\alpha$, on all tour edges, and another integer number of copies, say $-q'\alpha$, $q' < q$, on nontour edges. Now choosing dual multipliers $q'\alpha/2$ on all degree-two constraints (1), yields the desired cost sum of 0 for nontour edges $(-q'\alpha + q'\alpha/2 + q'\alpha/2)$. Corresponding costs on tour edges are $(q' + q')\alpha$. Setting all unmentioned dual variables 0 and fixing $\alpha = 1/(q - q') > 0$ completes the recovery of $-c_G$.

It is interesting to consider what happens for a non-hamiltonian G. The corresponding TSP instance with cost $-c_G$ is still well-defined, but there are two possibilities. If the instance is exposed for subtours, i.e. it belongs to $TSP_{subtour}$, then the LP relaxation optimum over (1)–(4) will be integer, but its value must be strictly worse that $-|V|$. This is the case, for example, if we try non-hamiltonian bipartite graph $K_{2,3}$ as G. The other possibility is that the instance does not belong to $TSP_{subtour}$, i.e. every LP relaxation optimum is fractional. The famous Petersen graph (see for example [7, Figure 11.8]) provides an instance. Its subtour LP relaxation achieves objective value $-|V|$, but the only optimal solution uses $x_{ij} = 2/3$ on all edges.

These ideas lead directly to a reduction from the NP-complete hamiltonian graph problem, HAM (is a given graph hamiltonian?), that proves our exposed subsets are hard.

Theorem 5.4 *Exposed decision problems* $TSP^{\leq}_{subtour}$, $TSP^{\leq}_{comb}$, $TSP^{\leq}_{clique}$, $TSP^{=}_{subtour}$, $TSP^{=}_{comb}$, *and* $TSP^{=}_{clique}$ *belong to* $NP - Complete$.

Proof: We proceed by showing HAM reduces to both $TSP^{\leq}_{subtour}$ and $TSP^{=}_{subtour}$. This is sufficient for all claims because subtour constraints are special cases of both comb and clique tree inequalities, and because Theorem 4.2 already demonstrates all the problems belong to NP.

Consider a graph G and the associated instance of TSP, say I_G, with weight vector $-c_G$. If G is hamiltonian, Lemma 5.3 establishes that I_G is exposed for subtours. It follows that the corresponding threshold and exact value instances with threshold $v = |V|$ are acceptable because the optimal value in I_G will be exactly $-|V|$. If G is not hamiltonian, then (from the above remarks) either I_G is not exposed for subtours, or $I_G \in TSP_{subtour}$, but $\min\{-c_G x : x$ satisfies (1)–(4)$\} > -|V|$. Either way input pair $(I_G, |V|)$ would not be accepted in $TSP^{\leq}_{subtour}$ or $TSP^{=}_{subtour}$. ■

Theorem 5.5 *If every instance of TSP that belongs to $TSP_{subtour}$ or TSP_{comb} or TSP_{clique} can be solved in polynomial time, then $P = NP$.*

Proof: We proceed as in the proof of Theorem 5.4 to show that an algorithm [A] solving every instance in $TSP_{subtour}$ in time bounded by polynomial $p(n)$ would provide a polynomial algorithm for HAM. Since subtours are special cases of both comb and clique tree inequalities, this will prove all claims.

Given any G, we form the corresponding $-c_G$ and submit to [A]. If G is hamiltonian, [A] would halt with a feasible tour. If G is not hamiltonian, the result is less predictable, but [A] certainly will not halt with a tour; there are none. Thus we will know G is not hamiltonian when [A] either halts with some other outcome or exceeds time limit $p(n)$. ■

Readers might be puzzled by the fact that this last result says set $TSP_{subtour}$ is NP-hard, even though well known separation techniques can solve its LP relaxation of in polynomial time (see [7]). By definition instances of $TSP_{subtour}$ have an integer optimal solution over the subtour relaxation. How can they be NP-hard?

The answer hinges on whether the LP optimum over the subtour relaxation is unique. If so, then for any instance of $TSP_{subtour}$ the LP solution will index an optimal tour. But when an instance has alternative relaxation optima, there is no guarantee any LP solver will yield an integer one. Relaxation (1)–(4) does have fractional extreme-points. Thus, even with the optimal value in hand, there remains an NP-hard "rounding" task to find and prove an integer optimum.

6 Well Formed Instances and Promises

It is usual in complexity proofs to take as trivial the issue of recognizing a well formed instance of a problem. For example, define

Traveling Salesman Recognition ($TSP^{\in}$)

Instance: any string

Question: Does the string encode an instance of TSP?

All that is required to answer this question is to decide whether the input string can be viewed as the weight vector of a complete graph.

A corollary of Theorem 5.4 shows the case is quite different for at least the subtour-exposed case:

Subtour-Exposed Traveling Salesman Recognition ($TSP^{\in}_{subtour}$)

Instance: any string

Question: Does the string encode an instance of $TSP_{subtour}$?

Corollary 6.1 *It is NP-hard to determine whether a given instance of TSP is exposed for the subtour polytope, i.e. $TSP^{\in}_{subtour}$ is NP-complete.*

Proof: We will show that $TSP^{\in}_{subtour} \in NP$ and that NP-complete $TSP^{=}_{subtour} \propto NP^{\in}_{subtour}$. The proof of the first is essentially that of Theorem 4.2. When an instance is exposed for the subtour polytope, the implied integer optimum for the LP relaxation must be a tour. By guessing the tour, and then guessing the construction of a corresponding dual-optimal basis, optimality of the tour can be verified in polynomial time.

To show $TSP^{=}_{subtour} \propto TSP^{\in}_{subtour}$ observe that an input is accepted for $TSP^{=}_{subtour}$ if and only if it encodes a subtour-exposed instance of the TSP together with its optimal value. Given any string, our reduction algorithm first invokes any polynomial procedure for $TSP^{\in}$. If the string proves to be a well formed TSP instance, followed by a rational v, we then solve (via separation) the subtour LP relaxation for that instance to obtain its optimal value $\bar{v}$. Strings rejected by $TSP^{\in}$ or having subtour relaxation value $\bar{v}$ equal to the prospective exact value v are submitted directly to an oracle for $TSP^{\in}_{subtour}$. In all other cases, which must have $\bar{v} \neq v$, we submit instead an instance on 10 vertices with weights $-c_G$, where G is the Petersen graph.

We have already remarked that this Petersen graph instance cannot be exposed for subtours. Thus the $TSP^{\in}_{subtour}$ oracle will accept the submitted string exactly when it yields a well formed instance of $TSP_{subtour}$ with optimal value v. ■

The proof of Corollary 6.1 depends strongly on the existence of polynomial-time separation procedures for LP optimization over the subtour relaxation (1)-(4). Since separation schemes are not known for comb and clique inequalities, we do not know whether recognition of instances exposed over those polytopes is also NP-hard. Still, we could use Sanchis's reference [14, Proposition 4.4] to conclude that if $TSP^{\in}_{comb}$ or $TSP^{\in}_{clique}$ belongs to P, then $NP = co(NP)$.

Her approach also raises our next issue. We know that threshold versions $TSP^{\leq}_{subtour}$, $TSP^{\leq}_{comb}$ and $TSP^{\leq}_{clique}$ are all NP-complete (Theorem 5.4). What about their complements $co(TSP^{\leq}_{subtour})$, $co(TSP^{\leq}_{comb})$ and $co(TSP^{\leq}_{clique})$? Since the exact optimal value v^* can be nondeterministically computed (proof of Theorem 4.2) for an instance in any of these languages, *no* instances with unattainable threshold $v < v^*$ can also be identified by a $NDTM$. It might seem that say $co(TSP^{\leq}_{subtour}) \in NP$. It would follow that $NP = co(NP)$ because the complement of an NP-complete problem belongs to NP.

One simply may not ignore the "technicality" of recognition of well formed instances. In particular, the complement of $TSP^{\leq}_{subtour}$ is the union of two parts

$$\begin{aligned} co(TSP^{\leq}_{subtour}) &= \{\text{strings acceptable in } TSP^{\in}_{subtour} \text{ with } v \text{ unattainable}\} \quad (10) \\ &\quad \cup \{\text{strings unacceptable in } TSP^{\in}_{subtour}\} \end{aligned}$$

Nondeterministic polynomial recognizability of the first part is not enough to place $co(TSP^{\leq}_{subtour})$ in NP. (Sanchis's proof of Proposition 4.4 simply observes that if any of the instance recognition cases $TSP^{\in}_{-}$ were in P, both halves of (10) could be checked, implying $NP = co(NP)$).

Imagine now receiving some files of test instances (and solutions) from the authors, or generating them yourself. You would know that the instances of say $TSP_{subtour}$ were produced by the RC TIG. They were constructed to be acceptable for $TSP^{\in}_{subtour}$, and the only remaining way a derived threshold instance can be a *no* case is if the threshold is unattainable.

This scenario provides a natural example for that part of complexity theory dealing with *promises* [15, 5]. A promise is an extra bit in the input of a language guaranteeing the instance possesses some mathematical property. The Turing machine is allowed to rely on this promise in deciding whether an input belongs to the language.

What occurs when problems $TSP^{\leq}_{subtour}$, $TSP^{\leq}_{comb}$ and $TSP^{\leq}_{clique}$ come with a promise that they were constructed by [RC]? In a sense, they become easier. Define

$$\begin{aligned} RCTSP_{subtour} &\triangleq \{TSP_{subtour} \text{ instances known generated by [RC]}\} \\ RCTSP_{comb} &\triangleq \{TSP_{comb} \text{ instances known generated by [RC]}\} \\ RCTSP_{clique} &\triangleq \{TSP_{clique} \text{ instances known generated by [RC]}\} \end{aligned}$$

Also let threshold ($\leq$), exact value ($=$) decision problems be defined analogously.

Theorem 6.2 *Generated threshold problems $RCTSP^{\leq}_{subtour}$, $RCTSP^{\leq}_{comb}$ and $RCTSP^{\leq}_{clique}$ belong to $promise-(NP \cap co(NP))$.*

Proof: The proof is the same for all three forms. It is easy to see that $RCTSP^{\leq}_{subtour} \in promise-NP$; we merely ignore the promise bit and use the $NDTM$ for $TSP^{\leq}_{subtour}$. To show the problem is also in *promise-co*(NP), we must establish that it is the complement of a member of *promise-NP*, i.e. that its complement $co(RCTSP^{\leq}_{subtour})$ is nondeterministically recognizable with the aid of a reliable promise bit. As in (10), the complement $co(RCTSP^{\leq}_{subtour})$ has two parts: those that have the promise bit *off* and those generatable by [RC] that have unattainable thresholds. A mere scan of the bit will settle the first, and we know from discussion above that valid inputs with invalid thresholds can be nondeterministically verified. ■

We have already observed several times that subtour form $RCTSP^{\leq}_{subtour}$ is unique among our three classes of exposed instances in that its linear programming relaxation, (1)-(4), is known to be polynomially solvable. This leads to another promise-based result.

Theorem 6.3 *Subtour-generated value problem $RCTSP^{=}_{subtour}$ belongs to promise-P.*

Proof: Given an instance of $RCTSP^{=}_{subtour}$ we first apply the known polynomial-time algorithm to compute $\bar{v} \overset{\triangle}{=}$ the optimal value of its LP relaxation. If the promise bit is *off*, or the input threshold v differs from $\bar{v}$, we reject. Otherwise, since the optimal value of subtour-generated TSP's is known to equal $\bar{v}$, we can accept. ■

So in one sense, a promise or guarantee that the instances were generated by the RC TIG does reduce their complexity. We conclude by showing that nonetheless they remain formally hard to solve. Return for a moment to the problem of hamiltonian cycles in graphs. One promise-based question in this context is

Hamiltonian Cycle Exhibition (HCE)

Instance: a hamiltonian graph G

Solution: the incidence vector x_G of a hamiltonian cycle in G

As pointed out in [2], the promise of an input's hamiltonicity does not change the fact that the HCE form is hard. For if there were a polynomial time algorithm for HCE, it could be modified to recognize hamiltonian graphs, by incorporating its polynomial time bound as a time limit.

Similarly, full optimization on RC TIG instances remains hard, even with a promise of [RC] generatability.

Theorem 6.4 *Subtour-generated instances $RCTSP_{subtour}$ form an NP-hard set.*

Proof: Reduction from HCE. Given an instance of HCE, we proceed as above to construct one for $RCTSP_{subtour}$ by using graph cost vector $-c_G$. The promise bit can correctly be marked *on* for such an instance because Lemma 5.3 shows that [RC] can generate $-c_G$ so long as the input graph is hamiltonian. The $RCTSP_{subtour}$ output, which will be an integer solution to (4)-(6), then yields the tour required in HCE. ■

Even if test instances from the RC TIG come with an absolute seal of authenticity, producing their optimal tours is, in a formal sense, still difficult.

7 Acknowledgement

The authors are grateful to colleagues Eric Allender, Vijay Chandru, Collette Coullard, Andrew Odlyzko, R. Gary Parker and Don Wagner for helpful discussions on the material of this paper, and to the referee of an earlier version for some very useful insights.

References

[1] Crowder, H., E.L. Johnson, and M.S. Padberg [1983], "Solving Large-Scale Zero-One Linear Programming Problems to Optimality," *Operations Research*, 31, 803-834.

[2] Garey, M.R. and D.S. Johnson [1979], *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, San Francisco, California.

[3] Grötschel, M. and M.F. Padberg [1979], "On the Symmetric Travelling Salesman Problem I: Inequalities," *Math Programming* 16, 265-280.

[4] Grötschel, M. and W.R. Pulleyblank [1986], "Clique Tree Inequalities and the Symmetric Travelling Salesman Problem," *Math of Operations Research*, 11.

[5] Johnson, D. [1985], "The NP-Completeness Column: An Ongoing Guide," *Journal of Algorithms*, 6, 291-305.

[6] Krentel, M. [1986] "The complexity of optimization problems" Proc. 18th ACM Symp. Th. Comp, 69–76, ACM, New York, 1986.

[7] Lawler, E.L., J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys [1985], editors, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, John Wiley and Sons Ltd., London.

[8] Papadimitriou, C.H. and M. Yannakakis [1984], "The Complexity of Facets (and Some Facets of Complexity)", *J. of Computer and System Sciences*, 28, 244-259.

[9] Parker, R.G. and R.L. Rardin [1988], *Discrete Optimization*, Academic Press, Boston, Massachusetts.

[10] Pilcher, M.G. [1985], "Development and Validation of Random Cut Test Problem Generator", Ph.D. dissertation., School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia.

[11] Pilcher, M.G. and R.L. Rardin [1986], "Partial Polyhedral Description and Generation of Discrete Optimization Problems with Known Optima," report CC-87-4, University Research Initiative in Computational Combinatorics, School of Industrial Engineering, Purdue University, West Lafayette, Indiana, to appear in *Naval Research Logistics.*

[12] Pilcher, M.G. and R.L. Rardin [1986], "Invariant Problem Statistics and Generated Data Validation: Symmetric Traveling Salesman Problems," report CC-87-16, University Research Initiative in Computation Combinatorics, School of Industrial Engineering, Purdue University, West Lafayette, Indiana.

[13] Rais, A., and R. L. Rardin [1988], "Random Generation of Travelling Salesman Problems Using Clique Tree Inequalities, report CC-88-21, University Research Initiative in Computational Combinatorics, School of Industrial Engineering, Purdue University, West Lafayette, Indiana.

[14] Sanchis, L.A., [1990], "On the Complexity of Test Case Generation for NP-Hard Problems," *Information Processing Letters*, 36, 135–140.

[15] Valiant, L. and V. Vazirani, "NP is As Easy As Detecting Unique Solutions," *Proceedings of 17th ACM Symposium on Theory of Computing*, ACM, New York.