

SHEARER'S ENTROPY LEMMA AND THE TSP

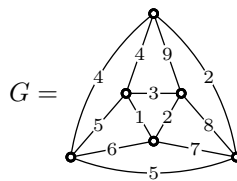
THORE HUSFELDT

ABSTRACT. This note aims to give an application of entropy to the analysis of algorithms, using Shearer's entropy lemma to bound the running time of Bellman's dynamic programming solution to the traveling salesman problem.

The original venue for this material was my course *Randomised algorithms* (2008) at Lund University.

1. The Traveling Salesman Problem

Given a weighted graph like



with n vertices $V = \{v_1, \dots, v_n\}$ (sometimes called “cities”) the *traveling salesman problem* is to find a shortest Hamiltonian cycle, i.e., a tour that starts and ends at the same vertex, includes every other vertex exactly once, and travels along edges whose total weight is minimal.

This is a hard problem, in fact, NP-hard, so we have to resign ourselves to algorithms whose running time is super-polynomial.

The straightforward way to solve this problem takes time $O(n!)$, see exercise 1. In the early 1960s, one of the earliest applications of dynamic programming was to improve this running time to $O(2^n n^2)$, a bound that is still the best known.

Here's how that works: Select an arbitrary reference vertex $s \in V$; we choose the topmost vertex in our example graph G . For $T \subseteq V$ and $v \in T$, denote by $D(T, v)$ the minimum weight of a path from s to v that consists of exactly the vertices in T . The minimum weight of a tour is then found by computing

$$\min_{v \in V} D(V, v) + d(v, s) .$$

To construct $D(T, v)$ for all $s \in T \subseteq V$ and all $v \in T$, the algorithm starts with $D(\{s\}, s) = 0$, and evaluates the recurrence

$$(1) \quad D(T, v) = \min_{u \in T \setminus \{v\}} D(T \setminus \{v\}, u) + d(u, v) .$$

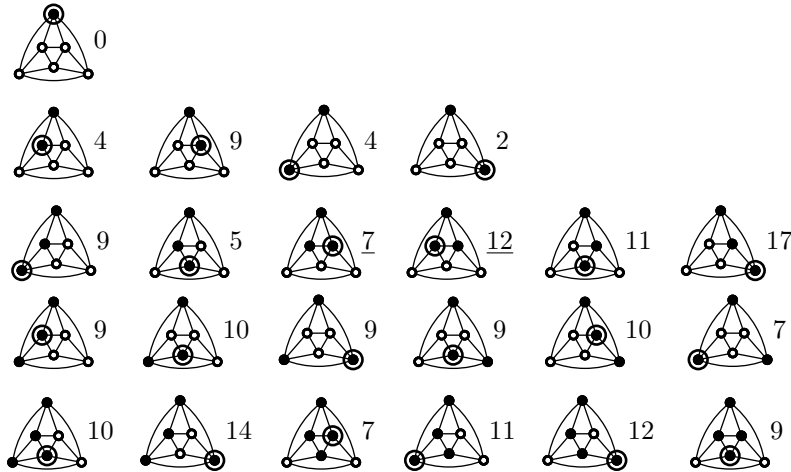


FIGURE 1. The first few steps of filling out a table for $D(T, v)$ for the example graph. The starting vertex s is at the top, v is circled, and T consists of the black vertices. At this stage, the values of $D(T, v)$ have been computed for all $|T| \leq 3$, and we just computed the value 9 at the bottom right by inspecting the two underlined cases. The “new” black vertex has been reached either via a weight 2 edge, for a total weight of $2 + 7$, or via a weight 1 edge for a total weight of $12 + 1$. The optimum value for this subproblem is 9.

The values $D(T, v)$ are stored a table when they are computed to avoid redundant recomputation, an idea sometimes called *memoisation*. The space and time requirements are within a polynomial factor of 2^n , the number of subsets $T \subseteq V$. Figure 1 shows the first few steps.

Exercises.

1. Give a simple algorithm for TSP with running time $O(n!)$.
2. Implement the algorithm from the previous exercise.
3. Give a pseudocode implementation of the dynamic programming algorithm. There are two approaches to implementing dynamic programming; you should write out both. (i) Build the tables “bottom-up” by starting with $D(\{s\}, s)$ and then constructing, for $i = 1, \dots, n$, the tables $D(T, v)$ for all $v \in T$ and T of size $|T| = i$. (ii) Write the program as a recursive function that looks up its parameters in a table. If the answer has already been computed it simply returns the table value, otherwise it proceeds to compute the value (presumably incurring further recursive invocations) and stores the result.
4. Finish the example in figure 1.

2. Connected sets

Our idea to expedite this will restrict the family of subsets for which (1) is ever evaluated. To this end, consider any prefix (v_1, v_2, \dots, v_k) of a finite-weight tour with $v_1 = s$. The central observation is that set of vertices $T = \{v_1, v_2, \dots, v_k\}$ is *connected*. Put otherwise, $D(T, v) = \infty$ unless T is a connected set. Thus, it suffices

to evaluate (1) not over all subsets of V , but only over the family of connected sets \mathcal{C} ; details of the implementation are considered in exercise 5. Roughly speaking, the running time is dominated by filling in the table entries, so finding the asymptotic running time of this algorithm boils down to bounding the number of connected sets of a given graph.

In general, that's not much help. If you look at the example in figure 1 the only case that we can avoid is the disconnected set



all other choices of T that include the topmost vertex are connected. More generally, if G is a clique of size n then *every* vertex subset is connected. And exercise 6 shows that some very sparse graphs can have a lot of connected subsets, not significantly smaller than 2^n .

The rest of this note shows that for regular graphs of small degree, we *can* bound the number of connected sets. To be precise, we will show in the rest of this note that if every vertex in G has exactly $k - 1$ neighbours then

$$(2) \quad |\mathcal{C}| \leq \left[(2^k - 1)^{1/k} \right]^n + n$$

The good news is that the quantity in the square brackets is strictly less than 2 for every k , so once we establish (2) can triumphantly conclude that the dynamic programming algorithm runs in time $O((2 - \epsilon)^n)$ on bounded degree regular graphs.

Exercises.

5. Give the details of the above algorithm in pseudocode. *Hint:* Whether $T \in \mathcal{C}$ can be tested in polynomial time by, e.g., depth-first search; furthermore, for every $T \in \mathcal{C}$ with $|T| > 1$ there exists at least one $v \in T$ with $T \setminus \{v\} \in \mathcal{C}$ – consider the leaves of a spanning tree of $G[T]$ – which enables T to be discovered from $T \setminus \{v\}$.

6. Find a family of graphs with average degree at most 2 and $2^{n-1} - 1$ connected sets.

3. Entropy

I repeatedly roll an eight-sided and want to tell you the outcomes; I can do that with $\log 8 = 3$ bits per die roll. But assume I've changed the faces to 1, 1, 1, 1, 2, 2, 3, 4 as in Fig. 2. Now, there are only four different outcomes, with probabilities $p(1) = \frac{1}{2}$, $p(2) = \frac{1}{4}$, and $p(3) = p(4) = \frac{1}{8}$. Since there are only four outcomes, I could communicate the die roll in $\log 4 = 2$ bits. But if you and I agreed to encode the outcomes like this,

outcome	1	2	3	4
message	1	01	001	000

I'd need only $\frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \frac{1}{8} \cdot 3 = 1.75$ bits. No such clever encoding is possible for a standard 4-sided die; I can't do better than 2 bits on the average. In that sense, the modified 8-sided die contains less information, or is "less random" than the 4-sided die. At the extreme, if I had replaced *all* faces of my die with "1", I would need zero bits to communicate the outcome, because you already knew it.

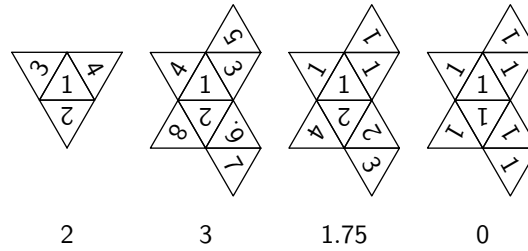


FIGURE 2. Eight dice and their entropies

This value, the “average information content” of X , is called the entropy, and we’ll formalise it now.

Consider a random variable X with outcomes x_1, \dots, x_k such that x_i happens with probability $p(x_i)$. The value

$$(3) \quad I(x_i) = \log \frac{1}{p(x_i)} = -\log p(x_i)$$

measures the *information content* (or *surprisal*) of outcome x_i . This value is high for outcomes of low probability (which motivates the reciprocal) and additive, see ex. ?? (which motivates the logarithm).

The expected value of (3) is called the *entropy* of X , given by

$$\mathbf{H}[X] = E(I(X)) = -\sum_{i=1}^k p(x_i) \log p(x_i);$$

for this to make sense we use the convention $0 \log 0 = 0$. From this perspective, $\mathbf{H}[X]$ measures the expected number of bits needed to describe an outcome of X in an optimal encoding. Note that, unlike the variance and expectation of a random variable, the entropy depends only on the *probabilities* $p(x_i)$ of the outcomes, not on their *values* x_i ; the introductory example would have been exactly the same if I’d decorated the die with ♣, ♥, ♠, ♦ instead of 1, 2, 3, 4.

We first observe the bounds

$$0 \leq \mathbf{H}[X] \leq \log |K|.$$

The lower bound is clear, since $p(\cdot) \leq 1$, so all the terms in the definition are positive. It is attained when X is constant, say $p(x_1) = 1$:

$$\mathbf{H}[X] = -1 \log 1 - \sum_{i=2}^k 0 \log 0 = 0.$$

The upper bound is proved in ex. ?.?. It is attained when X is uniformly distributed:

$$\mathbf{H}[X] = -\sum_{i=1}^k \frac{1}{|K|} (\log 1 - \log |K|) = \log |K|.$$

For two random variables X, Y , we define the *conditional entropy of Y given X*

$$\mathbf{H}[Y | X] = -\sum_{x \in X} \sum_{y \in Y} \Pr(X = x, Y = y) \log \Pr(Y = y | X = x).$$

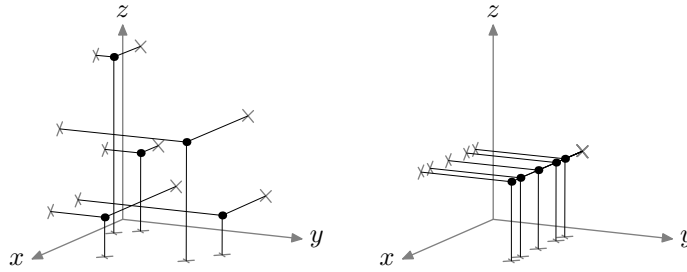


FIGURE 3. The formula $n^2 \leq n_{x=0}n_{y=0}n_{z=0}$ for $n = 5$ points. Left: the general case with $25 = n^2 < n_{x=0}n_{y=0}n_{z=0} = 125$. Right: equality is attained in the extremal case with $n_{x=0} = 1$.

This describes the average amount of information I'd need to communicate to you about Y if you already knew the outcome of X .

By simple manipulation of the definition it can be seen that the joint entropy of two (not necessarily independent) random variables satisfies a chain rule,

$$(4) \quad \mathbf{H}[X, Y] = \mathbf{H}[X] + \mathbf{H}[Y | X].$$

If X and Y are independent, $\mathbf{H}[Y | X] = \mathbf{H}[Y]$ and we have recovered the formula $\mathbf{H}[X, Y] = \mathbf{H}[X] + \mathbf{H}[Y]$.

We also have

$$(5) \quad \mathbf{H}[Y] \geq \mathbf{H}[Y | X],$$

with the intuitively satisfying interpretation that conditioning reduces information. That's a bit harder to prove.

Exercises.

7. Let $D \in \{1, \dots, 8\}$ denote the outcome of throwing a fair 8-sided die, and let E denote the indicator random variable that is 1 if and only if the die roll was even. Compute (from the definition) the values $\mathbf{H}[D]$, $\mathbf{H}[E]$ and $\mathbf{H}[D | E]$. Repeat for all the dice in Fig. 2.

8. True or false: a random variable takes on at least $2^{\mathbf{H}[X]}$ values. What about "exactly $2^{\mathbf{H}[X]}$ values"?

9. Prove (4).

10. Prove $\mathbf{H}[X_1, \dots, X_n] \leq \mathbf{H}[X_1] + \dots + \mathbf{H}[X_n]$.

11. Define $\mathbf{H}[Y | X = x] = -\sum_{y \in Y} \Pr(Y = y, X = x) \log \Pr(Y = y | X = x)$. True or false: $\mathbf{H}[Y] \geq \mathbf{H}[Y | X = x]$?

4. A geometry puzzle

Place n points in space and let $n_{x=0}$, $n_{y=0}$, and $n_{z=0}$ denote the number of projected points on the (y, z) -, (x, z) -, and (x, y) -planes, respectively. Then I claim

$$(6) \quad n^2 \leq n_{x=0}n_{y=0}n_{z=0}.$$

Pick a point (X, Y, Z) at random among the n points; we have

$$\mathbf{H}[X, Y, Z] = \log n.$$

Applying (4) twice, we also have

$$(7) \quad \log n = \mathbf{H}[X, Y, Z] = \mathbf{H}[X] + \mathbf{H}[Y | X] + \mathbf{H}[Z | X, Y].$$

How does the projection (X, Y) behave? It's picked among the $n_{z=0}$ points, so $\mathbf{H}[X, Y] \leq \log n_{z=0}$. (This would hold with equality if (X, Y) were picked *uniformly* among the $n_{z=0}$ projected points, but it's not.) We also have $\mathbf{H}[X, Y] = \mathbf{H}[X] + \mathbf{H}[Y | X]$. We have three such equations, which we can set up suggestively like this:

$$\begin{aligned} \log n_{z=0} &\geq \mathbf{H}[X, Y] = \mathbf{H}[X] + \mathbf{H}[Y | X] &&= \mathbf{H}[X] + \mathbf{H}[Y | X] \\ \log n_{y=0} &\geq \mathbf{H}[X, Z] = \mathbf{H}[X] + \mathbf{H}[Z | X] &\geq \mathbf{H}[X] &+ \mathbf{H}[Z | X, Y] \\ \log n_{x=0} &\geq \mathbf{H}[Y, Z] = \mathbf{H}[Y] + \mathbf{H}[Z | Y] &\geq \mathbf{H}[Y | X] &+ \mathbf{H}[Z | X, Y] \end{aligned}$$

The last inequalities in the second and third line are just applications of (5). Adding these three lines we arrive at

$$\log n_{z=0} + \log n_{y=0} + \log n_{x=0} \geq 2\mathbf{H}[X] + 2\mathbf{H}[Y | X] + 2\mathbf{H}[Z | X, Y],$$

and finally combining with (7) we have

$$(8) \quad \log n_{z=0} + \log n_{y=0} + \log n_{x=0} \geq 2 \log n,$$

which yields (6).

5. Shearer's Lemma

This trick works not only for three variables. In general, the claim is the following:

Lemma 1. *Let \mathcal{F} be a family of subsets $F \subseteq \{1, \dots, n\}$ such that each $1 \leq i \leq n$ appears in k sets of \mathcal{F} . Let (X_1, \dots, X_n) be a random variable and write X_F for $(X_i : i \in F)$. Then*

$$\mathbf{H}[X_1, \dots, X_n] \leq \frac{1}{k} \sum_{F \in \mathcal{F}} \mathbf{H}[X_F].$$

Before the proof, let's connect this to the previous example. There, we had $\mathcal{F} = \{\{1, 2\}, \{2, 3\}, \{1, 3\}\}$, $k = 2$, and the lemma's bound was just (8). The case where $\mathcal{F} = \{\{1\}, \dots, \{n\}\}$, $k = 1$ was considered in exercise 10.

Proof. For given F we can repeatedly apply (4) for

$$\mathbf{H}[X_F] = \sum_{j \in F} \mathbf{H}[X_i | (X_j)_{j < i, j \in F}].$$

Summing these for all $F \in \mathcal{F}$ arrives at

$$\sum_{F \in \mathcal{F}} \mathbf{H}[X_F] = \sum_{F \in \mathcal{F}} \sum_{j \in F} \mathbf{H}[X_i | (X_j)_{j < i, j \in F}].$$

Since each i appears in k sets $F \in \mathcal{F}$, there are k terms of the form $\mathbf{H}[X_i | \dots]$ on the right hand side. Moreover, we can lower bound each of these terms by $\mathbf{H}[X_i | \dots] \geq \mathbf{H}[X_i | (X_j)_{j < i}]$. Thus,

$$\sum_{F \in \mathcal{F}} \mathbf{H}[X_F] \geq k \sum_{i=1}^n \mathbf{H}[X_i | (X_j)_{j < i}] = k \mathbf{H}[X],$$

where the last equality is another application of (4). \square

6. The Connected Induced Subsets

Let \mathcal{F} be the family of closed neighbourhoods

$$F(v) = \{u \in V : uv \in E\} \cup \{v\}$$

of a graph G (read *friends* for F). Then $|\mathcal{F}| = n$ and every v appears in exactly k subsets of \mathcal{F} . We want to bound the size of \mathcal{C} . For a moment, remove the singletons from \mathcal{C} , we'll put them back later.

Consider a random connected set $C \in \mathcal{C}$ of size $|C| \geq 2$ and let (X_1, \dots, X_n) denote the corresponding indicator random variables

$$X_i = \begin{cases} 1 & \text{if } i \in C \\ 0 & \text{if } i \notin C. \end{cases}$$

Then we have

$$\mathbf{H}[X_1, \dots, X_n] = \log |\mathcal{C}|.$$

To apply Shearer's lemma we also need an upper bound on $\mathbf{H}[X_F]$ for every $F = F(v) \in \mathcal{F}$. To make notation easy look at the case where vertex 1 has neighbours $2, \dots, k$, so $F = F(1) = \{1, 2, \dots, k\}$. We're interested in the joint random variable $X_F = (X_1, X_2, \dots, X_k)$. How many different values can it take? Naïvely, these are k different 0/1 variables, so there are 2^k different choices. But wait! They can never have the value $(1, 0, \dots, 0)$, which would correspond to vertex 1 but none of its neighbours belonging to C . So in effect there are only $2^k - 1$ possible choices for X_F . Thus

$$\mathbf{H}[X_F] \leq \log(2^k - 1).$$

Plugging these bounds into Shearer's lemma we find

$$\log |\mathcal{C}| \leq \frac{1}{k} \sum_{F \in \mathcal{F}} \log(2^k - 1) = \frac{1}{k} \log \left(\prod_{F \in \mathcal{F}} (2^k - 1) \right) = \log((2^k - 1)^{n/k}).$$

Removing the logarithms and putting the singletons back into \mathcal{C} we finally arrive at

$$|\mathcal{C}| \leq (2^k - 1)^{n/k} + n,$$

which is what we promised in (2).

Notes

Shearer's lemma is from [Chung, F., Frank, P., Graham, R., and Shearer, J.: *Some intersection theorems for ordered sets and graphs*. Journal of Combinatorial Theory (A) 43 (1986), 23–37.]. The dynamic programming solution for the TSP is from [Bellman, R.: *Combinatorial Processes and Dynamic Programming*. In: Bellman, R., Hall, M., Jr. (eds.) *Combinatorial Analysis, Proceedings of Symposia in Applied Mathematics* 10, pp. 217–249. American Mathematical Society (1960)], [Bellman, R.: *Dynamic Programming Treatment of the Travelling Salesman Problem*. J. Assoc. Comput. Mach. 9, 61–63 (1962)] and [Held, M., Karp, R.M.: *A Dynamic Programming Approach to Sequencing Problems*. J. Soc. Indust. Appl. Math. 10, 196–210 (1962)]. The exposition of Shearer's lemma, including the gentle introduction via the three-dimensional point set, is entirely from [Radakrishnan,

J.: *Entropy and counting.*, In: Mishra, J.C. (ed.) IIT Kharagpur Golden Jubilee Volume on Computational Mathematics, Modelling and Algorithms, Narosa Publishers, New Delhi, (2001)]; the standard proof is an induction argument, see for example [Jukna, S.: *Extremal Combinatorics*, Springer-Verlag (2001)]. The application to the TSP is from [Björklund, A., Husfeldt, T., Kaski, P., and Koivisto, M.: *The Travelling Salesman Problem in bounded degree graphs*, ICALP 2008]; the bound in the result can be improved and actually holds for bounded degree graphs (not only *regular* such graphs).

Finding an algorithm with running time $O(1.9999^n)$ for the TSP for general graph remains an open problem.