Dynamic programming and greedy heuristic in Camera Trap Traveling Salesman Problem

Yaroslav V. Salii
yvs314@gmail.com

Evgenii E. Ivanko
viatore@ya.ru

Krasovskii Institute of Mathematics and Mechanics (Yekaterinburg, Russia)
Ural Federal University (Yekaterinburg, Russia)

Abstract

We consider a very weak version of 1-PDTSP related to one specific problem of ordering a sequence of single-type objects, which stems from the problem of rearranging camera traps to increase statistic significance in problems of assessing animal population. Compared with our previous version of dynamic programming, the dimension of problem instances solved went up to 12 camera traps (25 cities including the base) due to a leaner C++ implementation and the lack of treatment of infeasible DP states. Feasibility of a dynamic programming solution was studied analytically in view of the memory constraints, i.e., we implemented an algorithm for calculating the number of states for this problem and compared it with the corresponding traveling salesman problem. For solving the problems of dimension greater than 12, we implemented a simple greedy heuristic, the quality of which was compared with the exact algorithm for one hundred randomly generated 12-camera trap problem instances.

1 Introduction

The problem under consideration consists of relocating several pre-installed single-type objects to designated new installation sites, starting from some point of origin that does not coincide with either pre-installed objects or new object positions, and returning there after the work is finished, all of this as fast as possible. It is almost the same as the traveling salesman problem (TSP), its only difference being the obvious constraint of never visiting a new installation site without at least one object "in the backpack." The "backpack" is assumed to have infinite capacity.

This problem was first formulated in [6] in view of a real-life problem that biologists who study animal population dynamics face when trying to improve the statistical significance of animal population assessments made with the aid of camera traps. A camera trap is sufficiently light to carry any reasonable amount in a backpack, yet good camera traps are also too expensive to obtain in such an amount so as to remove the need to relocate those already placed. Hence the need to consider exactly the problem we describe below; throughout the paper, we will refer to our objects as camera traps, although it is of course applicable to other objects in similar circumstances. In fact, this problem is a very weak version of 1-PDTSP (see, for example, [5]), with infinite capacity and binary supply and demand.

In: A.A. Makhnev, S.F. Pravdin (eds.): Proceedings of the 47th International Youth School-conference "Modern Problems in Mathematics and its Applications", Yekaterinburg, Russia, 02-Feb-2016, published at http://ceur-ws.org

We propose an exact method of solving this problem, via dynamic programming (DP), and a simple greedy heuristic. These methods are chosen over the more traditional and developed mixed integer-linear programming because a) there is no actual need to solve really large problems optimally—relocating 12 camera traps is deemed sufficient and b) they do not require expensive license fees of good linear programming frameworks.

2 Problem statement

Here and below, the symbol $\triangleq$ denotes *equality by definition*; an *ordered pair* of two objects a and b is written in parentheses, (a, b); and the *power set* of a set S is written $\mathcal{P}(S)$.

Denote the dimension of the problem, i.e., the number of cameras, by a natural N. The *ground set* is $X \triangleq \overline{0, 2N} = \{0\} \cup \mathcal{R} \cup \mathcal{B}$, where the symbol 0 denotes the *base* (the point of origin and return), the set $\mathcal{R} \triangleq \overline{1, N}$, $|\mathcal{R}| = N$, contains the indices of extant camera trap locations, and the set $\mathcal{B} \triangleq \overline{N+1, 2N}$, $|\mathcal{B}| = N$, indexes the new sites for the camera traps. For the sake of convenience, "extant camera trap locations" will be referred to as *red* points, and "new camera trap locations" will be referred to as *blue* points. We will also avoid distinguishing a point from its index where it will not lead to confusion.

The cost of moving between two points is defined through a function $\mathbb{c} : X \times X \to \mathbb{R}$. Each problem instance is completely determined by the set X and the corresponding cost function $\mathbb{c}$. A *solution* to a problem $(X, \mathbb{c})$ is a permutation of the indices $\overline{1, 2N}$, or a *route*; the set of all solutions is denoted by $\mathbb{P}$. We will denote the routes by lowercase Greek letters, and, in an expression of the form $\alpha(i) = j$, the natural number j is an index of the point that occupies the i-th position of the route α.

Since the agent may only visit a *blue* point with at least one camera trap "in his pack," some routes will inevitably be infeasible, e.g., the route that visits all blue points first. We must offer a means of distinguishing between *feasible* and infeasible routes. For a DP solution, it is convenient to use a "feasible exit point operator" $\mathbb{I} : \mathcal{P}(X) \to \mathcal{P}(X)$ defined as follows:

$$\mathbb{I}(K) \triangleq \begin{cases} K, & |K \cap \mathcal{R}| > |K \cap \mathcal{B}|; \\ K \cap \mathcal{B}, & |K \cap \mathcal{R}| = |K \cap \mathcal{B}|. \end{cases}$$

Informally, it says that the *last* point of a route through K starting at 0 MAY be any color if there are more *red* than *blue* points in K, but it MUST be a *blue* point if K has equal number of red and blue points. There may never be more *blue* than *red* points in K because in this case the number of new camera trap sites is greater than the number of camera traps collected.

With the aid of the operator $\mathbb{I}$, the feasibility of a route is expressed by a gradual application of this operator to the route's "prefixes" (more formally, the set "supports" of these prefixes), and the *set of all feasible routes* is defined as follows:

$$\mathbb{A} \triangleq \Big\{ \alpha \in \mathbb{P} \,\Big|\, \forall s \in \overline{1, 2N}\ \alpha(s) \in \mathbb{I}\Big(\big\{\alpha(t) : t \in \overline{1, s}\big\}\Big) \Big\}.$$

Note that this is actually the same as a Dyck language, and the number of such routes is given by the N-th Catalan number (see the references in [7]).

Finally, the quality of a *feasible route* $\alpha \in \mathbb{A}$ is measured by the following additive *quality criterion*:

$$\mathfrak{C}[\alpha] \triangleq \Big\{ \mathbb{c}\big(0, \alpha(1)\big) + \sum_{i=1}^{2N-1} \mathbb{c}(i, i+1) + \mathbb{c}\big(\alpha(2N), 0\big) \Big\},$$

and the problem is to *minimize* it by choosing the best feasible route.

3 Dynamic programming

We employ a forward DP procedure not unlike [4] (and contrasting with the backward DP of [1]); this choice is only a matter of preference—we find the forward procedure more straightforward for our particular problem. Our notation is largely that of [3], with certain inconsequential tweaks to adapt it to the forward DP. For our problem, this method was proposed in [6], with its validity proved by induction.

The principal idea of DP is to decompose a complex, intractable problem into a family of less complex *subproblems*, from optimal solutions of which it is easy to obtain an optimal solution of the complete problem; such subproblems are called the DP *states*.

In our case, a *state* is an ordered pair (K, x), where $K \subset \overline{1, 2N}$ is called a *task list*, and $x \in X \setminus K$ is the *terminal point*. It describes the (sub)problem of visiting the points from K starting from the point of origin 0 and finishing at the point x. In contrast with the ordinary TSP, not all such states are *feasible*, i.e., some subproblems may not actually be solved without violating the condition of never visiting a *blue* point without a camera trap in the pack, e.g., the state $(\mathcal{B}, 1)$.

In [6], *infeasible* states were assigned a cost of "infinity" (larger than any real cost possible). It is actually possible to save both CPU cycles and storage space if, instead of making the cost of infeasible states "infinite," we could omit the treatment of these altogether. To this end, let us elaborate on a formal feasibility condition for the states and a generation procedure that only produces the feasible ones.

A task list $K \subset \overline{1, 2N}$ is called *feasible* if the number of blue points in it does not exceed the number of red points; let us gather all such task lists into the set

$$\mathcal{G} \triangleq \Big\{ K \subset \mathbb{X} \,\Big|\, |K \cap \mathcal{R}| \geq |K \cap \mathcal{B}| \Big\}.$$

We will find a use for a cardinality-wise partition of this set, $\mathcal{G}_i \triangleq \big\{ K \in \mathcal{G} \,\big|\, |K| = i \big\}$, $i \in \overline{0, 2N}$, with the empty task list $\{\varnothing\}$ also deemed trivially feasible.

If a task list is feasible, it is still possible to "break" the state it would be a part of by choosing an inappropriate *terminal point*. Obviously, if a task list $K \in \mathcal{G}$ has more red points than blue, i.e., after we finish it, we will still have a camera trap to spare, we can choose any remaining point as the terminal. However, if all camera traps we pick during our tour of K end being "used up" as we finish this tour (i.e., $|K \cap \mathcal{R}| = |K \cap \mathcal{B}|$), we must continue to a *red* point to restock (with the obvious exception of $K = \mathcal{R} \cup \mathcal{B}$, after which we only have to return to the base). This is formally expressed through a *feasible expansion operator* $\mathcal{J} : \mathcal{G} \to X$ defined as follows:

$$\mathcal{J}(K) \triangleq \begin{cases} \mathbb{X} \setminus K, & |K \cap \mathcal{R}| > |K \cap \mathcal{B}|; \\ \big(\mathbb{X} \setminus K\big) \cap \mathcal{R}, & |K \cap \mathcal{R}| = |K \cap \mathcal{B}|; \end{cases}$$

we find it convenient to treat the case of $K = \mathcal{R} \cup \mathcal{B}$ separately, without resorting to the expansion operator. The feasible expansion operator may also be defined through the *feasible exit point operator* $\mathbb{I}$ in the form

$$\mathcal{J}(K) = \Big\{ x \in (X \setminus K) \,\Big|\, x \in \mathbb{I}\big(K \cup \{x\}\big) \Big\};$$

these are clearly equivalent, however, the direct definition helps to understand how exactly are feasible expansions generated.

We are finally ready to formally define a feasible state. A state (K, x) is considered *feasible* if $K \in \mathcal{G}$ and $x \in \mathcal{J}(K)$. Let us collect all the feasible states into the set

$$D \triangleq \big\{ (K, x) : K \in \mathcal{G}, x \in \mathcal{J}(K) \big\},$$

which we partition by the cardinality of the task sets contained therein, $\forall i \in \overline{0, 2N-1}$, $D_i \triangleq \big\{ (K, x) : K \in \mathcal{G}_i, x \in \mathcal{J}(K) \big\}$; we call the elements of this partition the *state space layers*, or just *layers*. Since we avoided dabbling with the base to prevent the clutter in our definition of $\mathcal{J}$, we have to define the final layer separately, $D_{2N} \triangleq \big\{ (\overline{1, 2N}, 0) \big\}$. This one corresponds to the complete problem. Note also the initial layer $D_0 = \big\{ (\varnothing, x) : x \in \mathcal{J}(\varnothing) \big\}$, which contains the trivial subproblems of starting at the base 0 and finishing at some *red* x.

For each state $(K, x) \in D$, its *value*—the cost of optimally visiting K, with start at 0 and finish at x,— is defined by the following recursive *Bellman equation*:

$$\begin{aligned} v(K, x) &= \min_{y \in \mathbb{I}(K)} \Big\{ v\big(K \setminus \{y\}, y\big) + \mathbb{c}(y, x) \Big\}; \\ v(\varnothing, x) &= \mathbb{c}(0, x) \;\forall x \in \mathcal{R}, \end{aligned}$$

where the value of trivial states from D_0 serves as an initial condition. The value of the complete problem is rendered as $v\big(\overline{1, 2N}, 0\big)$. The validity of this equation clearly follows from the proof of a similar equation from [6, Sect. 2.1].

The procedure of generating the feasible states D bottom-up and calculating their values is as follows:

1. Prime the algorithm with $\mathcal{G}_0 = \{\varnothing\}$, $\mathcal{G}_1 = \mathcal{R}$, and D_0. Note that the values $v[D_0]$ are already known.
2. For each $l \in \overline{1, 2N-1}$

   For each $K \in \mathcal{G}_l$, for each $x \in \mathcal{J}(K)$,

   - Add $K \cup \{x\}$ to $\mathcal{G}_{l+1}$.
   - Add (K, x) to D_l. Calculate $v(K, x)$ through the Bellman equation.
3. Calculate the final value $v\big(\{\overline{1, 2N}\}, 0\big)$ through the Bellman equation.
4. Recover an optimal route based on the values of v on D.

We will not elaborate on the recovery procedure used at stage 4 of this algorithm; its nothing new and was discussed, for example, in [6, Sect 2.2].

4 Experiment

We analyzed 100 randomly generated problem instances with $N = 12$ camera traps (25 points including the base) and compared the performance of the exact DP and greedy heuristic. The particular value of N was chosen to reflect the problem of the greatest dimension the DP solution of which would not exceed 16GB of RAM at the author's PC. All data sets are available from the authors on request.

The exact DP algorithm took from 85 to 94 seconds to solve for each instance, while the time required by the greedy heuristic was negligible. The performance gap (i.e, the ratio of the result of the greedy heuristic and the result of the exact DP minus 1) was, on the average, 28%, with the minimum being 1%, maximum 81%, and median 26%.

5 Space complexity of exact DP

Since our implementation of exact DP could not fit more than a 12-camera trap problem instance into the 16GB available (a 12-camera trap instance consumed circa 12GB of RAM), we decided to try and calculate how many states are actually feasible and compare this number with that of the corresponding TSP (i.e., we compared the states count of our N-camera problem with $2N$-city TSP).

It is not hard to calculate how many are there states in a TSP. In TSP, all task sets are feasible, and all expansions are feasible too. Fix a number n. Then, the number of states is

$$C_{\mathrm{TSP}}(n) = \sum_{i=0}^{n-1} \binom{n}{i}(n-i) + \binom{n}{n}1.$$

Indeed, for $i \in \overline{0, n-1}$, $|\mathcal{G}_i| = \binom{n}{i}$, and the number of terminal points is $n - i$; there is only one state in D_n, namely, $(\overline{1, n}, 0)$ is unique.

For our problem, this calculation is a bit more complex since a)some task lists are infeasible and b)some expansions are not feasible (for certain feasible task lists of *even* cardinality, we can only expand into *red* points). We find it impractical to provide a closed formula for our problem, but we will sketch its idea for a problem with 3 camera traps.

Consider the feasible task lists $\mathcal{G}_0, \ldots, \mathcal{G}_6$. For $\mathcal{G}_0$, we may choose neither a red nor a blue point, $|\mathcal{G}_0| = \binom{3}{0}\binom{3}{0}$; obviously, it may be expanded by any red point, thus, $|D_0| = |\mathcal{R}| = 3$. Next, $\mathcal{G}_1$ has odd cardinality. Since we only choose one point, it has to be red, thus, $|\mathcal{G}_1| = \binom{3}{1}\binom{3}{0}$; in all task lists of $\mathcal{G}_1$, there are more red than blue points, hence, any of the remaining $2n-1$ points is a feasible expansion, $|D_1| = \binom{3}{1}\binom{3}{0} \cdot 5$. Let us pass to $\mathcal{G}_2$. We may either pick two red points, or one red and one blue, $|\mathcal{G}_2| = \binom{3}{2}\binom{3}{0} + \binom{3}{1}\binom{3}{1}$. In the former case, we may expand to any of the remaining $2n-2$ points, however, in the latter, we have to go to a red point, of which there remains $n-1$: $|D_2| = \binom{3}{2}\binom{3}{0} \cdot 4 + \binom{3}{1}\binom{3}{1} \cdot 2$. For $\mathcal{G}_3$, we have $|\mathcal{G}_3| = \binom{3}{3}\binom{3}{0} + \binom{3}{2}\binom{3}{1}$ and $|D_3| = \binom{3}{3}\binom{3}{0} \cdot 3 + \binom{3}{2}\binom{3}{1} \cdot 3$. Passing to $\mathcal{G}_4$, we see $|D_4| = \binom{3}{3}\binom{3}{1} \cdot 2 + \binom{3}{2}\binom{3}{2} \cdot 2$; for D_5, we have $|D_5| = \binom{3}{3}\binom{3}{2} \cdot 1$, and D_6 is a singleton, $|D_6| = \binom{3}{3}\binom{3}{3} \cdot 1$.

The procedure for calculating the number of states for our problem was coded in `Haskell`. We also calculated the ratio between $C_{\mathrm{TSP}}(2n)$ and $|D|$ for n-camera trap problem. It starts at 1.666 for 1 camera trap (2 TSP cities, not counting the base), goes up to 1.941 for 2 camera traps (4 TSP cities, not counting the base), and has six nines after the point as early as for 9 camera traps (18-city TSP).

6 Conclusions

We studied a rather weak variety of 1-PDTSP and proposed two palatable methods of solving it; we also found out it is about twice as easy as a comparable TSP in terms of DP state count. The dimension of problems solved was doubled in comparison with our previous attempt [6].

Our future work on similar problems has several objectives. The first one is to develop a software for planning multiple camera trap relocations through a GIS so as to integrate this function into a GPS navigation device. The second one is to deal with more constraints that arise in the field—typically, one hopes to complete the relocations during the day's sunlight hours. However, in case of a great number of camera traps or rough terrain it may be impossible. To automate the planning procedure, we intend to adopt certain features of Orienteering Problem or Team Orienteering Problem (see, for example, [2]).

References

[1] R. Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM (JACM)*, 9(1):61–63, 1962

[2] I. M. Chao, B. L. Golden, E. A. Wasil. The team orienteering problem. *European journal of operational research*, 88(3):464–474, 1996

[3] A. G. Chentsov. *Extremal problems of routing and scheduling: a theoretical approach.* Regular and Chaotic Dynamics, 2008 (in Russian). = А. Г. Ченцов. *Экстремальные задачи маршрутизации и распределения заданий. Вопросы теории.* «Регулярная и хаотическая динамика», 2008.

[4] M. Held, R. M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial & Applied Mathematics*, 10(1):196–210, 1962

[5] H. Hernández-Pérez, J. J. Salazar-González. A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. *Discrete Applied Mathematics*, 145(1):126–139, 2004.

[6] E. Ivanko. Dynamic programming in a problem of rearranging single-type objects. *Trudy Inst. Mat. i Mekh. UrO RAN*, 19(4):125–130, 2013 (in Russian). = Е. Иванко. Динамическое программирование в задаче перестановки однотипных объектов. *Тр. ИММ УрО РАН*, 19(4):125–130, 2013.

[7] J. Labelle, Y. N. Yeh. Generalized Dyck paths. *Discrete mathematics*, 82(1):1–6, 1990.