Algorithm in Empirical Modelling: Dynamic Programming with Cadence

1035196

Abstract

This paper looks at a scenario which involves route planning and optimization problem. Travelling Salesperson Problem (TSP) is the model to be discussed. It aims at piloting the usage of dynamic programming (DP) in Empirical Modelling (EM) in terms of the deep dependency of its nature. The investigation will align with pure definitive notation to maintain a clean structure of the model and to avoid side effect of other approaches, though different programming approaches are embraced by EM. The subject investigated remains a pilot purpose because of the novel environment.

1 Introduction

Conventional programming relies heavily on data structures and algorithms to produce fast and accurate result. While EM introduces programming with definitive notation, it is possible that the use of data structures and algorithms would assist in the understanding and development process in EM. Hopefully, the support of algorithm and data structure can not only provide fast and responsive result, but also allow interaction with user and programmer.

TSP is a NP-hard problem in combinatorial optimization. Given a set of cities that a travelling salesperson has to visit, and the distance between any two cities, TSP is the task to route the shortest possible path that goes to each city once and back to its starting point so as to give an optimal result. TSP is particularly useful in planning, logistics.

Various solutions have been developed to solve TSP, such as heuristics, approximation, ant colony optimization, etc. TSP can actually also be solved by greedy algorithm, it is easy but it involves a lot of procedurals process. However, the model uses the simplest approach: Dynamic Programming.

2 TSP

2.1 Dynamic Programming

DP is a approach that solving the complex problem by solving its smaller, simpler sub-problems.

The DP approach is similar to EM underlying meaning: observation as programming. In DP, based on the observation of the problem, it can be formulated to generalize the complex problem. Based on what the observation is, formulate the observation, and finally construct the formulation.

2.1.1 Observation

The problem is to route a path that go through all the cities once and get back home with the lowest cost.

Given a set of cities and the pairwise distance, it is observed that the lowest cost path from city A to a set of cities S depends on the lowest cost path from any city B insides S to S\{B}.

2.1.2 Formulation

Let S be a set of cities the travelling salesperson has to travel to.
Let Z be a set of cities unvisited.
Let H be the home city.

D(A, B) = Distance between city A and city B
f(A, Z) = lowest cost path from city A to set Z

Then, we can try to translate the observation into formulation.

$$f(A, Z) = \min(D(A,B) + f(B, Z\backslash\{B\}))$$

Hence, obviously our goal is f(H, S).

As noticed, this formulation is a recursive function, which means it will generate a new node when it goes deeper. Therefore, the running time and

memory used is very likely to be increased exponentially with the number of cities. For example, a 10 cities TSP would generate more than 3millions nodes. Therefore, DP for TSP is probably not a feasible solution but as a pilot purpose, the paper will use a model with only 4 cities TSP. Figure 1 shows the exponential property of 4 cities TSP.

The deeper the DP grows, the more nodes and more definitions are needed, and the more powerful comparing to other paradigm it is.

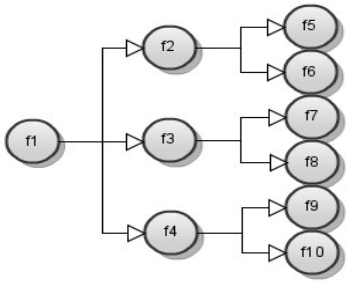

Figure 1: 4 Cities TSP

2.1.3 Construction

The construction is easy to do by making use of abstract definition. The messy states of conventional procedural programming is therefore omitted. By giving only definition of observables, the logic is easily readable and bug fixing is much more easier without the need of studying what are those symbols or states for.

However, due to the limitation of the Cadence environment, it is difficult to dynamically generate new nodes. Using agents (definer_is/ definer_willbe) would be helpful, but at this point the 'definition' part of the definer_is agent cannot be used to determine dynamic definition. For example, it has no way to use definer_is to give definition which dynamically determine its dependency, ie. '*definition is {.(.determiner)}*'.

Therefore, the definition in the model is given explicitly. By doing so, we can all embrace the power of definitive notation, which gives responsive result on the fly on matter how its dependencies change. The advantages of explicit definition are:

- easy-to-understand clean-structure and definition
- easy-to-debug
- fast and immediate response

Actually explicit definition simply means instantiating all the possible instances of the recursive function. And therefore gives the following disadvantages as well:

- duplicating lengthy code
- lose of dynamic
- time consuming to produce large amount of instances
- maybe slow in generating many nodes (but would be reduced over time since required nodes are previously generated)

Figure 2: Interface of the Model

3 Environment

3.1 Eden v.s. Cadence

Eden is a hybrid environment which on one side composes of different scripting notation like Scout, Donald, on the other side it encompasses different programming paradigm like procedural programming, functional programming, declarative programming. It is a powerful tool that can handle not only computation but also drawing.

However, this hybrid environment gives Eden a chance to be suffered from the side effects of different programming paradigm, for example the problem of state, and may potentially ruin the philosophy of definitive notation which contains no state though it can maintain its definitive power. It maybe a confusing environment for learning EM.

For Cadence, it is a novel yet powerful pure definitive environment. Although it presents simple procedural like 'if-then-else' and 'select', it still maintaining definitive position by allowing stateless environment.

3.1.1 Limitation

Using Cadence for modelling would experience following limitations:

- Lack of functions

  Only small number of mathematical functions are supported, some other useful function like max, min, power, factorial, etc.

- Difficulty in writing parametric function

  If parametric function is needed, most probably have to new observables in the context to store parameter and result, and also need mechanism to reset parameter

- Confusing syntax

  Braces and parenthesis are confusing, and the situation where parenthesis is needed or not is difficult to notice

3.1.2 Possible resolution

In Cadence, using Agent would give much help. The use of multi-agents can simulate lots of functions. Automated agents can also give a new perspective to EM.

- Agent can be used to monitor unexpected situation, and save a lot of code then 'if-then-else'. For example,

```
.agents boundX = (new union (@prototypes assigner)

        condition is {@man x < 0 or (@man x > 500)}

        object = @man

        property = x

        value = (@city city1 x)

)
```

whenever the condition is true, the Agent is triggered.

- Agent can also be used to simulate procedural process, for example 'for-loop':

```
.agents for_loop = (new union (@prototypes assigner)

        condition is {@root cond}

        object = @root

        property = counter

        value is {@root counter + 1}

);
```

The segment of codes increments the counter (can also be done with *.counter := .counter + 1*). To simulate the assignment inside for-loop, create more agents sharing the same *condition*, so all the agents within the for-loop will be triggered at the same time when the *condition* is true.

- Agent can be button listener/event listener as well instead of having lots of 'if-then-else' statement:

```
.agents addSpeedBtnListener = (new union (@prototypes
assigner)

        condition is {@screen addSpeedBtn click}

        object = @man

        property = speed

        value is {@man speed + 10}

);
```

4 Conclusion

To conclude, this paper as well as the model remains a pilot position to the field of algorithm of Empirical Modelling. However, the concept of functions and algorithm is worth researching deeper. Here, only some very basic concept and experience are discussed, and more further discussions are also needed.

In terms of modelling environment, Cadence needs to be improved so as to be capable of more complex computation. Eden is a powerful tools but its position is confused in EM.

References

0515575 Reactive Search and Rescue Planning, [Online]:http://www2.warwick.ac.uk/fac/sci/dcs/research/em/publications/web-em/05/search-and-rescue.pdf

Greedy Algorithms and the Making Change Problem, [Online]:http://www2.warwick.ac.uk/fac/sci/dcs/research/em/publications/web-em/01/greedy.pdf

Meurig Beynon, Antony Harfield, Empirical Modelling in support of constructionism:
a case study, [Online]:http://www.dcs.warwick-.ac.uk/report/pdfs/cs-rr-412.pdf