Приложение к диплому. Код программы с комментариями.

16 июня 2018 г.

1 Вступление.

Далее приводится код использованной программы оптимизации. Программа не является образчиком «хорошего кода» и ее дальнейшее использование не предполагалось, код приводится в ознакомительных целях.

Программа написана на языке Lua. Lua (луа, с порт.- «луна») - скриптовый язык программирования, разработанный в подразделении Tecgraf (Computer Graphics Technology Group) Католического университета Риоде-Жанейро (Бразилия). Интерпретатор языка является свободно распространяемым, с открытыми исходными текстами на языке Си.

По идеологии и реализации язык Lua ближе всего к JavaScript, в частности, он также реализует прототипную модель ООП, но отличается Паскалеподобным синтаксисом и более мощными и гибкими конструкциями. Характерной особенностью Lua является реализация большого числа программных сущностей минимумом синтаксических средств. Так, все составные пользовательские типы данных (массивы, структуры, множества, очереди, списки) реализуются через механизм таблиц, а механизмы объектно-ориентированного программирования, включая множественное наследование - с использованием метатаблиц, которые также отвечают за перегрузку операций и ряд других возможностей.

Язык широко используется для создания тиражируемого программного обеспечения (например, на нем написан графический интерфейс пакета Adobe Lightroom). Также получил известность как язык программирования уровней и расширений во многих играх (в том числе World of Warcraft) из-за удобства встраивания, скорости исполнения кода и легкости обучения. За дальнейшей информацией можно обратиться в источники. **ССЫЛКИ**

2 Данные и переменные.

В программе используются следующие глобальные переменные (рис. 1):

- число посадочных площадок, а также 2*N- мерный вектор их координат
- **size_y, size_x, step**- размер, а также шаг сетки в км
- **vtol_velocity, car_velocity**- скорости движения VTOL и автомобиля
- **vtol_service_time** - суммарное время, требуемое для взлета, посадки, загрузки и выгрузки пассажиров
- **preference**- пользователь предпочитает VTOL если выигрыш времени превосходит эту величину (в данной модели- 40%)

```
--Число ВПП
N = 10

current_point = {
24, 33,
6,  34,
13, 17,
17, 7,
25, 13,
5,  23,
9,  9,
15, 27,
15, 35,
23, 19
}

--Глобальные данные о карте
size_y = 28
size_x = 40
step   = 2.5

--Координаты аэропортов
vko_y = 19
vko_x = 19
svo_y = 3
svo_x = 23
dme_y = 28
dme_x = 35


--Веса аэропортов (пропорциональны транспортному потоку)
vko_p = 0.22
svo_p = 0.41
dme_p = 0.37

--Данные модели
vtol_velocity = 272
car_velocity  = 38.5
vtol_service_time = 7.25
preference = 0.4
time_difference = 30
```

Рис. 1: Список глобальных переменных.

- **time_difference**- дополнительный параметр- минимальное значение выигранного времени, при котором пользователь предпочтет VTOL (в данной модели не использовался)

Карты транспортной доступности, а также матрица плотности населения из формата CSV переводятся в двумерные таблицы (размером 28*40), которые сохраняются в виде отдельных файлов. (рис. 2) Которые далее подключаются к основному файлу программы. (рис. 3)

```
times = {}          -- create the matrix
    for i=1,size_y do
      times[i] = {}     -- create a new row
      for j=1,size_x do
        times[i][j] = 0
      end
    end
```

```
times = {
{0,0,0,0,74,71,69,67,65,63,63,62,63,63,65,67,69,71,74,0},
{0,0,0,74,70,67,64,62,60,59,58,57,58,59,60,62,64,67,70,74},
{0,78,74,70,67,63,60,58,55,54,53,52,53,54,55,58,60,63,67,70},
{0,76,71,67,63,60,56,53,51,49,48,47,48,49,51,53,56,60,63,67},
{0,73,69,64,60,56,53,49,46,44,43,42,43,44,46,49,53,56,60,64},
{76,71,67,62,58,53,49,46,42,40,38,37,38,40,42,46,49,53,58,62},
{74,69,65,60,55,51,46,42,38,35,33,32,33,35,38,42,46,51,55,60},
{73,68,63,59,54,49,44,40,35,31,28,27,28,31,35,40,44,49,54,59},
{0,68,63,58,53,48,43,38,33,28,24,22,24,28,33,38,43,48,53,58},
{0,67,62,57,52,47,42,37,32,27,22,17,22,27,32,37,42,47,52,57},
{0,68,63,58,53,48,43,38,33,28,24,22,24,28,33,34,43,48,53,58},
{0,68,63,59,54,49,44,40,35,31,28,27,22,18,23,24,44,43,54,59},
{0,0,65,60,55,51,46,42,38,35,27,22,7,11,15,42,35,51,49,57},
{0,0,0,0,58,53,49,46,39,32,32,20,6,0,15,29,31,40,48,50}
}
return times
```

Рис. 2: Пример таблицы Lua (здесь приводится уменьшенная версия).

```
--Подключение файлов
filepath = [[/Users/igorpanin/Desktop/VTOL/Optimization solution/Lua tables/?.lua]]
outfile = [[/Users/igorpanin/Desktop/Programming/MyLua/Results.csv]]
io.output(outfile)
package.path = package.path..';'..filepath

svo = require 'SVO_model'
dme = require 'DME_model'
vko = require 'VKO_model'

svo_ground = require 'SVO_ground'
dme_ground = require 'DME_ground'
vko_ground = require 'VKO_ground'

density    = require 'Density'
```

Рис. 3: Подключение таблиц с данными.

3 Вспомогательные функции.

Запись таблицы в файл. Следующая функция (рис. 4) записывает таблицу с названием **new_times** в файл с округлением до одной минуты.

```
--Вывод таблицы в файл
for i = 1, size_y do
    for j = 1, size_x do
        if new_times[i][j] ~= 0 then
            io.write(math.floor(new_times[i][j]+1))
        else io.write(0)
        end
        if j == size_x then
            io.write('\n')
        else
            io.write(',')
        end
    end
end
```

Рис. 4: Базовые функции 1.

На следующем рисунке (рис. 5) содержатся три функции. Первая принимает значения координат двух точек и возвращает дистанцию между ними в км. Вторая- возвращает время транзита на VTOL с учетом временных затрат на взлет и посадку. Третья функция устроена сложнее. В функцию передаются параметры:

- **y, x**- координаты начала маршрута
- **s_y, s_x**- координаты посадочной площадки
- **a_y, a_x** - координаты аэропорта, до которого осуществляется трансфер
- **times**- карта транспортной доступности
- **base_times**- транспортная доступность аэропорта без учета VTOL

Эта функция возвращает минимально возможное время транзита из заданной точки в аэропорт, при этом выполняется сравнение времени транзита с использованием посадочной площадки с координатами **s_y, s_x** и текущая транспортная доступность из таблицы **times**. Также время транзита должно быть не менее чем на 40% меньше, чем поездка с использованием существующего транспорта (данные из таблицы **base_times**). Функция используется для вычисления времени поездки с учетом N+1 посадочных площадок, если в таблице **times** время транзита для N площадок.

Еще две функции (рис. 6). Первая записывает в таблицу **new_times** карту транспортной доступности указанного аэропорта (**a_y, a_x**). Значения времени, содержащиеся в таблице **times** обновляются с учетом посадочной площадки в точке с координатами **s_y, s_x**. Фактически эта

```
--Вычисляет дистанцию между двумя точками
function linear_distance (y1, x1, y2, x2)
    local dist = step * ( (y1 - y2)^2 + (x1 - x2)^2 )^0.5
    return dist
end

--Вычисляет время перелета использует параметры   vtol_velocity  vtol_service_time
function vtol_time (y1, x1, y2, x2)
    local distance = 1.42 * linear_distance (y1, x1, y2, x2)
    local time = 60 * distance / vtol_velocity + vtol_service_time
    return time
end

--Возвращает минимальное время с учетом передаваемой таблицы времен
function calculate_time (y, x, s_y, s_x, a_y, a_x, times, base_times)
    local pref = preference
    local time_0 = times[y][x]
    -- Если доставка из этой точки осуществляется с помощью VTOL в условии нет необходимос
    if time_0 < base_times[y][x] then
        pref = 0
    end
    local time_1 = 0
    local site_distance = linear_distance (y, x, s_y, s_x)
    time_1 = time_1 + 60 * site_distance / car_velocity
    time_1 = time_1 + vtol_time ( s_y, s_x, a_y, a_x)
    if time_1 < time_0 * (1 - pref) then
        return time_1
    else
        return time_0
    end
end
```

Рис. 5: Базовые функции 2.

функция применяет **calculate_time** для каждой точки сетки. Функция **update_situation** возвращает карту транспортной доступности с учетом N посадочных площадок, координаты которых задаются вектором **vector**. Следующая функция вычисляет эффективнсть. (рис. 7) И еще одна сервисная функция осуществляет проверку точки на выход за границы сетки. (рис. 8)

```
--Возвращает таблицу значений времен для заданных координат ВПП
function update_times (s_y, s_x, a_y, a_x, new_times, times, base_times)
for y= 1, size_y do
    for x = 1, size_x do
        new_times[y][x] = calculate_time (y, x, s_y, s_x, a_y, a_x, times, base_times)
    end
end
end

--Возвращает таблицу значений времен для N ВПП с известными координатами
function update_situation (vector, a_y, a_x, times, base_times)
    local updated_times ={}
        for i=1,size_y do
            updated_times[i] = {}
            for j=1,size_x do
                updated_times[i][j] = times[i][j]
      end
    end
    for i = 1, N do
        s_y = vector[2*i-1]
        s_x = vector[2*i]
        update_times (s_y, s_x, a_y, a_x, updated_times, updated_times, base_times)
    end
    return updated_times
end
```

Рис. 6: Базовые функции 3.

```
--Вычисляем эффективность с учетом плотности населения
function efficiency (times, new_times)
    local population = 0
    local delta = 0
    for y = 1, size_y do
        for x = 1, size_x do
            if times[y][x] ~= 0 then
                population = population + density[y][x] / 1000
                delta = delta + ( times[y][x] - new_times[y][x] ) * density[y][x] / 1000
            end
        end
    end
    return delta / population
end
```

Рис. 7: Базовые функции 4.

```
--Проверка правильности точки на выход за границы сетки или зоны измерений
function check (vector)
    local validity = true
    for j = 1, 2*N do
    --Проверяем выход за границы сетки
        if vector[j] <= 0 then
            validity = false
        end
        if j%2 == 1 and vector[j] > size_y then
            validity = false
        end
        if j%2 == 0 and vector[j] > size_x then
            validity = false
        end
    end
    --Выход за границы зоны измерений
    for j = 1, 2*N-1, 2 do
        local y = vector[j]
        local x = vector[j+1]
        if vko[y][x] == 0 or svo[y][x] == 0 or dme[y][x] == 0 then
            validity = false
        end
    end
    return validity
end
```

Рис. 8: Базовые функции 5.

4 Вычисление градиента.

Аргументом функции является точка 2*N мерного пространства- конфигурация посадочных площадок. Возвращается 2*N мерный вектор градиента эффективности (вектор приращений).

```
--Вычисление градиента
function calculate_gradient(current_point)
    local gradient = {}
    for i =1, 2*N do
    gradient[i] = 0
end

    for i = 1, 2*N do
    local inc_point = {}
    local dec_point = {}
    for k = 1, 2*N do
        inc_point[k] = current_point[k]
        dec_point[k] = current_point[k]
    end
    inc_point[i] = inc_point[i] + 1
    dec_point[i] = dec_point[i] - 1

    local inc_check = check (inc_point)
    local dec_check = check (dec_point)
    local inc_efficiency = 0
    local dec_efficiency = 0
    local delta = 2

    if not(inc_check) then
        delta = delta - 1
        inc_point[i] = inc_point[i] - 1
    end
    if not(dec_check) then
        delta = delta - 1
        dec_point[i] = dec_point[i] + 1
    end
    if delta == 0 then
        gradient[i] = 0
        break
    end

    local inc_times_vko = update_situation (inc_point, vko_y, vko_x, vko, vko_ground)
    local inc_times_dme = update_situation (inc_point, dme_y, dme_x, dme, dme_ground)
    local inc_times_svo = update_situation (inc_point, svo_y, svo_x, svo, svo_ground)

    local dec_times_vko = update_situation (dec_point, vko_y, vko_x, vko, vko_ground)
    local dec_times_dme = update_situation (dec_point, dme_y, dme_x, dme, dme_ground)
    local dec_times_svo = update_situation (dec_point, svo_y, svo_x, svo, svo_ground)

    inc_efficiency = efficiency(vko, inc_times_vko) * vko_p +
    efficiency(dme, inc_times_dme) * dme_p + efficiency(svo, inc_times_svo) * svo_p
    dec_efficiency = efficiency(vko, dec_times_vko) * vko_p +
    efficiency(dme, dec_times_dme) * dme_p + efficiency(svo, dec_times_svo) * svo_p
    gradient[i] = (inc_efficiency - dec_efficiency ) / delta
end

    return gradient
end
```

Рис. 9: Вычисление градиента.

5 Остальной код.

Для лучшего результата рекомендуется последовательно запускать код, увеличивая число посадочных площадок на 1 в каждой итерации. В противном случае возможно попадание в локальный экстремум, когда несколько посадочных площадок совпадают. В каждой итерации выбираются 6 стартовых позиций с которых осуществляется градиентный спуск, варьирование стартовой точки осуществляется по последней посадочной площадке (две последние координаты вектора **current_point**). Из 6 экстремумов выбирается максимум, который и определяется как результат оптимизации для N посадочных площадок. (рис. 10, 11)

```
best_point = {}
for i = 1, 2*N do
    best_point[i] = 0
end

best_eff = 0
for j = 1, 2 do
    for k = 1, 3 do
            current_point[2*N - 1] = 10 * j
            current_point[2*N]     = 10 * k


--Реализация градиентного метода
new_point = {}
for i = 1, 2*N do
    new_point[i] = 0
end



new_times_vko = update_situation (current_point, vko_y, vko_x, vko, vko_ground)
new_times_dme = update_situation (current_point, dme_y, dme_x, dme, dme_ground)
new_times_svo = update_situation (current_point, svo_y, svo_x, svo, svo_ground)

vko_efficiency = efficiency (vko, new_times_vko)
dme_efficiency = efficiency (dme, new_times_dme)
svo_efficiency = efficiency (svo, new_times_svo)
current_efficiency = vko_efficiency * vko_p + dme_efficiency * dme_p + svo_efficiency * svo_p

gradient = {}
for i =1, 2*N do
    gradient[i] = 0
end

flag = 'run'

repeat

gradient = calculate_gradient(current_point)
```

Рис. 10: Остальной код 1.

```
--Переходим к новой точке
for j =1, 2*N do
    if gradient[j] > 0 then
        new_point[j] = current_point[j] + 1
    elseif gradient[j] < 0 then
        new_point[j] = current_point[j] - 1
    else
        new_point[j] = current_point[j]
    end
end

new_times_vko = update_situation (new_point, vko_y, vko_x, vko, vko_ground)
new_times_dme = update_situation (new_point, dme_y, dme_x, dme, dme_ground)
new_times_svo = update_situation (new_point, svo_y, svo_x, svo, svo_ground)

vko_efficiency = efficiency (vko, new_times_vko)
dme_efficiency = efficiency (dme, new_times_dme)
svo_efficiency = efficiency (svo, new_times_svo)
new_efficiency = vko_efficiency * vko_p + dme_efficiency * dme_p + svo_efficiency * svo_p
if new_efficiency > current_efficiency then
    for k = 1, 2*N do
        current_point[k] = new_point[k]
    end
    current_efficiency = new_efficiency
else
    flag = 'end'
end
until flag == 'end'
print(3*(j-1)+k, current_efficiency)
if current_efficiency > best_eff then
    best_eff = current_efficiency
    for f = 1, 2*N do
        best_point[f] = current_point[f]
    end
end
end
end
print(best_eff)
for i = 1, 2*N-1, 2 do
    local y = best_point[i]
    local x = best_point[i+1]
    print(y, x, density[y][x])
end
```

Рис. 11: Остальной код 2.