

Solver for vehicle routing problems

DOCUMENTATION V. 1.3

INFO@SHOBB.SITE

Contents

Description of the task for which this program is intended:	2
The algorithm used:	3
The data you need to solve the problem:.....	4
Features of technical implementation:.....	4
Point data structure:	5
Distance matrix element data structure:	5
Vehicle data structure:	6
Skills string format:	6
Transfer order data structure:	7
Solution structure description:	8
Summary structure:	8
Error structure:	8
Route structure:	8
Segment structure:	10

Description of the task for which this program is intended:

- You have a fleet of vehicles each of which has its own properties (cost, opening time, payload, start-to-end route, etc).
- There is a set of orders for transportation, each of which has its own characteristics - loading place, destination, temporary windows in which can be loading or unloading, weight of cargo, etc.
- You need to form a plan of fulfillment of orders for each vehicle, which takes into account all the restrictions and has the lowest possible cost.
- Orders and vehicles have options that the optimizer can be used for tasks such as PDPTW, VRPPD, VRPTW, DARP (with many extensions) - and even scheduling.

The algorithm used:

- The optimizer uses a highly advanced meta heuristic algorithm to solve the problem, the effectiveness of which is confirmed by several world records and some achievements in solving reference tests.
- Its speed, with very good quality of the solution is quite high. All operations that are possible are performed in parallel. Thus, you can increase the speed of the solution by increasing the number of processor cores.

The data you need to solve the problem:

- A set of starting and end points for vehicles and points of departure and destination. Some of these points may be the same.
- The time/distance matrix between these points. The elements of the matrix may depend on time. In addition, up to four different matrixes can be set (for each vehicle, you can specify which matrix to use if the characteristics of the vehicles are different).
- Description of shipping orders.
- Vehicle descriptions.

The result of the program's work is a file with a detailed task execution plan in JSON format for ease of use in other systems. There is also an example of converting this file to Excel.

Features of technical implementation:

- The program is written in C # and consists of several assembly modules.
.Net Framework 4.5.2 or higher required.
- To supply data to the input of the solver, it is necessary to implement the interface (described in a separate module):

```
interface IDataIntrf
{
    // A method that returns a list of orders as appropriate data structures.
    IEnumerable<SOrderData> getOrders();

    // A method that returns a list of vehicles as appropriate data structures.
    IEnumerable<SVehData> getVehicles();

    // A method that returns a list of all distinct loading / unloading points and vehicle start
    / end points as appropriate data structures.
    IEnumerable<SPtInfo> getPoints();

    // A method that returns one element of a specified matrix of distances between two
    points, as a corresponding data structure.
    SVDistInfo getDistance(int matrix_index,int point_from_id , int point_to_id );
}
```

This allows you to take input data from various sources.

All data structures will be described below.

Point data structure:

```
struct SPtInfo {  
    int Id          // Unique point ID  
    string Name     // Name  
    string Info     // optional: Additional information  
    string Address  // optional: Address  
    string LatLng   // optional: GPS coordinates  
}
```

Distance matrix element data structure:

```
struct SVDistInfo {  
    int Point_from_id // Identifier of the point of the beginning of the section  
    int Point_to_id   // Identifier of the end point of the section  
    STimeInfo [] ATime // Array of time/distance description structures (at least 1 element)  
    // Time / distance array element:  
    struct STimeInfo {  
        double StartTime // Start time of this element  
        double TravelTime // Travel time  
        double Distance   // Travel distance  
    }  
    // ATime elements must be sorted by StartTime and StartTime of all elements must be different  
}
```

Vehicle data structure:

```
struct SVehData
{
    int Id                // Unique identifier of vehicle.
    string Name           // Name
    int Start_Point_Id    // Identifier of the route start point
    int End_Point_Id      // Identifier of the route end point (can be the same as Start_Point_Id)
    double Capacity       // load capacity
    double Volume         // optional: maximum cargo volume
    int Matrix_Index      // Index of the time/distance matrix (0 - 3)
    double Max_run        // maximum allowed path length
    int Max_Order_Count   // maximum allowed number of orders to be executed
    int Attr              // 0 - if the vehicle ends the path at the destination of the last order,
                        // otherwise 1
    int LoadConstr        // Restrictions on the order of loading / unloading: 0 - no, 1 - LIFO, 2 -
                        // FIFO
    double TimeStart      // Start time
    double TimeEnd        // End time of work
    double delay_at_pt    // Delay time when for loading / unloading
    double own_weight     // Own weight of vehicle
    double cost_vehicle    // Fixed cost for using vehicle
    double cost_distance   // Cost of distance unit
    double cost_time       // Cost of a unit of time
    double cost_job        // Cost of a unit of transport work
    double kcost_order     // Coefficient of the cost of an order for this vehicle
    double coefficient_of_delay // Coefficient of the cost of the delay in the start of the order
    double ktime_speed     // Speed coefficient - the time of distance matrix is multiplied by this
                        // coefficient
    string Skills         // - optional: list of additional properties of this vehicle
}
```

Skills string format:

<NAME OF SKILL1: string>:<count:int>:<type:int>;<NAME OF SKILL2:string>:<count:int>:<type:int>

Where

NAME OF SKILL – some abbreviation for skill

Count – number of such skills

Type of skill: 0, 1 or 2.

If type of skill is 1 – then it is released after using, is 0 – vehicle simply “has” such property, type 2 – if skill has limit for whole route.

At example skills for taxi: front seat, two rear seats and lux car – “FRS:1:1;REAR:2:1;LUX:1:0”

For Order it does not contain type, only name of skill and count.

At example:

“REAR:1;LUX:1” – Order needs 1 rear seat in LUX car.

If Skills for order – empty string – order does not need any special resources.

Transfer order data structure:

```
public struct SOrderData {  
    public int Id                // Unique order identifier  
    public string Name           // Name  
    public int Point_from_id     // loading point identifier  
    public int Point_to_id       // destination point identifier  
    public double Cargo_weight   // Cargo weight  
    public double Cargo_volume   // optional: cargo volume  
    public double TimeToLoad     // Time to Load  
    public double [] TimeStartLoad // Array of start time of time windows to load  
    public double [] TimeEndLoad  // Array of the end time of time windows for loading  
    // Arrays TimeStartLoad and TimeEndLoad must have equal length  
  
    public double TimeToUnload    // Array of start times of time windows to unload  
    public double [] TimeStartUnload // Array of start times of time windows for unloading  
    public double [] TimeEndUnload  // Array of the end time of time windows for unloading  
    // Arrays TimeStartUnload and TimeEndUnload must have equal length  
  
    public bool adjust_UnloadTw   // true - if the time window for unloading is set relative to the  
                                // load time, and in fact it represents the time limit for order  
                                // fulfillment.  
    public double CostOfDelay     // optional: the cost of delaying the start of the order  
    public string Skills          // optional: Requires additional vehicle properties to complete  
                                // the order.  
}
```


Solution structure description:

```
solution
{
    solution_summary summary    // Total values across all routes
    List<route> routes          // list of all routes
}
```

Summary structure:

```
solution_summary
{
    double duration            // Total duration of all routes
    double distance            // Total length of all routes
    double job                  // Total transport job
    double cost                 // Total cost
    double orders_cost         // Total orders cost
    double cargo_weight         // Total cargo weight
    int orders_count            // Total number of orders
    double wait_time           // Total waiting time
    int number_of_routes        // Total routes
    List<error> errors          // List of errors, if any
}
```

Error structure:

```
error
{
    string description          // Error description
    double value                // Value
}
```

Route structure:

```
route
{
    int route_nr                // Route number
    vehicle vehicle              // Vehicle description
    summary summary              // Summary for this route
    List<segment> segments       // List of route segments
}
```

Vehicle structure:

```
vehicle
{
    int id // Unique identifier of vehicle.
    string name // Name
    string loading_constraints // Restrictions on the order of loading / unloading
    double delay_at_point // Delay time when for loading / unloading
    double ktime_speed // Speed coefficient
    List<skill> skills // List of additional properties of this vehicle
    point depot // Depot point
    point destination // Destination point
    int Matrix_Index // Index of the time/distance matrix
}
```

Skill structure:

```
skill
{
    string name // Skill name
    short as_resources // Skill type
    int count // Skill count
}
```

Point structure:

```
point
{
    int id // Unique point ID
    string name // Name
    string info // Additional information
    string address // Address
    string latlng // GPS coordinates
    List<operation> operations // List of operations performed at this point, if any
}
```

Operation structure:

```
operation
{
    string type                // Type of operation: Load / Unload
    double start_time          // Start time of this operation
    double end_time            // End time of this operation
    double late                 // Delay, if any
    double wait                 // Wait time before operation, if any

    // Information about related order:
    int order_id                // Unique order identifier
    string order_name           // Name
    double order_weight         // Cargo weight
    double order_volume         // Cargo volume
    List<skill> order_skills     // Additional vehicle properties, required to complete the order,
                                // if any
}
```

Segment structure:

```
segment
{
    point from_point           // Segment start point
    point at_point             // Segment end point
    double distance            // Travel distance
    double duration            // Travel duration
    double start_time          // Start time
    List<onboard> onboard;      // Items on board
}
```

Onboard structure:

```
onboard
{
    // Information about order:
    int order_id                // Unique order identifier
    string order_name           // Name
    double order_weight         // Cargo weight
    double order_volume         // Cargo volume
    List<skill> order_skills     // Additional vehicle properties, required to complete the order,
                                // if any
}
```