A data-structure covering a large scope of the taxonomy for routing applications

Atoptima

An open standard

Routing applications in logistics come in a rich panel of variations. Where possible, we consider the most general case that can be accommodated in our generic data structure; for otherwise, we make restrictive assumptions on the problem variant.

1 Restrictive Assumptions

So, we make the following restrictions, we consider:

- a **static and deterministic** problem as opposed to stochastic or with dynamically incoming data a multi-period horizon;
- **pure routing** decisions as opposed to integration with driver timetabling, inventory-management, production scheduling; in particular the framework does not implement restrictions due to driver regulation or packing placement issues, such as a 3D positioning in the vehicle (such issue are not trivial); in both suc case only surrogate relaxation of such constraint can be incorporated;
- a **single route** within the time horizon (the extension to the multiple route case is not straightforward).

2 A large scope of problem Variant

But our framework includes the following extentions of the basic problem:

- **multiple depots** (single depot being a special case);

- **heterogenous vehicle fleet** (a homogenous fleet being a special case);
- **bounded number of vehicles** (the unbounded case being a special case where the bounds are large enough to induce no restrictions on the problem);
- **time lag on shipments:** this is modeled by a max duration for a shipment as in dial a ride;
- **time-windows:** can be defined on pickups, deliveries, vehicle availability (f.i. for driver working day);
- **sequencing:** our framework recognize the backhauls special case; as well as prohibited sequence;
- **conflicts:** our framework recognize forbidding products to share a route;
- **split-quantity-option:** it is on if requests can be fulfilled by different routes;
- **request-cover-mode:** a request can be mandatory, or optional with a price reward for covering it;
- **mixing single-commodity and multi-commodity mode:** In the single commodity, the goods that are transported are undistinguishable. So a delivery can be done from any pickup point. When the requests are only of one type: either all are pickups or all are deliveries; then the problem should be understood as a single commodity problem, are there is no need to distinguish the goods. A multi-commodity model is required in the presence of "Pickup and Delivery" requests. The latter is associated with a specific pickup point and to a specific delivery point. When some goods are replaceable by others, then the associated delivery (resp. pickup) can have a choice of pickup (resp. delivery) points. So a complex request may be defined that involves several alternative pickup-point and/or several alternative delivery points. This complexity can make specific sense along with the split quantity option. In particular, in the multi depot case, goods can typically be delivered from (resp. delivered to) several of them. Our shipment-request model allows us to mix single-commodity and multi-commodity types.
- **service-times:** we distinguish access time and pickup or delivery points, pickup time and delivery time, the sum making up the service time.

- **multiple compartments** in the vehicle (single compartment being a special case), with loading mode option. For instance, one can model a restriction of one request per compartment for liquid transportation; then, the request can be accommodated only if the compartment capacity is larger than the request capacity requirement. Or a vehicle can have a trailer; then both the vehicle and the trailer have a maximum weight capacity, while each request has its own weight.
- **Rechargeable vehicle:** Recharging electric battery or any kind of fuel is possible at recharging stations. We assume a pice-wise linear concave recharging rate where the entry level depends on the remaining vehicle charge. To modle it, each vehicle has vector of charging doses (primary, secondary , etc); while the charging station as a vector of charging times to achieve each of these doses; the primary charging being at a higher rate than the secondary etc.

A Data Structure

A.1 Inputs

```
struct Coord
    x::Float64
    y::Float64
end

mutable struct Location
    id::String
    index::Int # Not given in JSON
    coord::Coord # optional
end

struct TimeWindow
    begin_time::Float64
    end_time::Float64
end

mutable struct PickupPoint
    id::String # If its part of a shipment, it has is own id anyway
    index::Int # Not given in JSON.
    location::Location
    opening_time_windows::Vector{TimeWindow} # optional
    access_time::Float64 # optional
end

mutable struct DeliveryPoint
    id::String # If it is part of a shipment,it has is own id anyway
    index::Int # Not given in JSON.
    location::Location
    opening_time_windows::Vector{TimeWindow} # optional
    access_time::Float64 # optional
end

mutable struct DepotPoint # location to start or end a route; a depot can also a
    id::String
    index::Int # Not given in JSON
    location::Location
    opening_time_windows::Vector{TimeWindow} # optional
    access_time::Float64 # optional
end

mutable struct RechargingPoint
    id::String # If it is part of a shipment,it has is own id anyway
    index::Int # Not given in JSON.
    location::Location
    energy_recharging_times::Vector{Float64} # to model a piecewise linear recha
    opening_time_windows::Vector{TimeWindow} # optional
    access_time::Float64 # optional
end

mutable struct ProductCategory
    id::String
    index::Int # Not given in JSON.
    conflicting_product_ids::Vector{String} # if any
    prohibited_predecessor_product_ids::Vector{String}  # if any
end

mutable struct SpecificProduct # an entity to understand as a commodity in a mul
    id::String
    index::Int # Not given in JSON.
    product_category_id::String
    pickup_availabitilies_at_point_ids::Dict{String,Float64} # pickups can be ei
    delivery_capacities_at_point_ids::Dict{String,Float64} # deliveries can be e
end

mutable struct Request # can be
    # a specific product shipment from a depot to a DeliveryPoint, or
    # a specific product shipment from a PickupPoint to a depot, or
    # a delivery to a DeliveryPoint of a product that is not specific to this de
    # a pickup from a PickupPoint of a product that is not specific to this pick
    # a specific product shipment from a specific PickupPoint to a specific Deli
    # a specific product shipment from one of several PickupPoints to a specific
    # a specific product shipment from a specific PickupPoints to one of several
    # a specific product shipment from one of several PickupPoints to one of sev
    id::String
    index::Int # Not given in JSON
    product_id::String
    is_optional::Bool  # default is false
    price_reward::Float64 # if is_optional
    product_quantity::Float64 # of the shipment
    compartment_capacity_conso::Float64 # portion of the vehicle compartment cap
    split_fulfillment::Bool  # true if split delivery/pickup is allowed, default
    precedence_restriction::Int # default is 0 = no restriction; 1 after all pic
    alternative_pickup_point_ids::Vector{String} # defined only if it is a subse
    alternative_delivery_point_ids::Vector{String} # defined only if it is a sub
    pickup_service_time::Float64 # optional; on top of PickupPoint service_time;
    delivery_service_time::Float64 # optional; on top of DeliveryPoint service_t
    max_duration::Float64 # used for the dail-a-ride model or similar applicatio
end

struct UnitPrices # cost per unit
    travel_distance_price::Float64
    travel_time_price::Float64
    service_time_price::Float64
    waiting_time_price::Float64
    recharging_time_price::Float64
end

mutable struct VehicleCategory
    id::String
    index::Int # Not given in JSON
    fixed_cost::Float64
    unit_pricing::UnitPrices
    compartment_capacities::Vector{Float64} # the stantard case is to have a sin
    energy_recharges::Vector{Float64} # to model a piecewise linear recharging c
    loading_option::Int # 0 = no restriction (=default), 1 = one request per com
    prohibited_product_category_ids::Vector{String}  # if any
end

mutable struct HomogeneousVehicleSet # vehicle type in optimization instance.
    id::String
    index::Int # Not given in JSON
    departure_depot_id::String # "": mentionned vehicle start from first action
    arrival_depot_ids::Vector{String}
    departure_depot_index::Int # -1: mentionned vehicle start from first action
    arrival_depot_indices::Vector{Int}
    vehicle_category::VehicleCategory
    working_time_window::TimeWindow
    initial_energy_charge::Float64
    min_nb_of_vehicles::Int
    max_nb_of_vehicles::Int
    max_working_time::Float64
    max_travel_distance::Float64
    allow_ongoing::Bool # true if these vehicles routes are open, and the vehicl
end

struct RvrpInstance
    id::String
    travel_distance_matrix::Array{Float64,2}
    travel_time_matrix::Array{Float64,2}
    energy_consumption_matrix::Array{Float64,2}
    pickup_points::Vector{PickupPoint}
    delivery_points::Vector{DeliveryPoint}
    depot_points::Vector{DepotPoint}
    recharging_points::Vector{RechargingPoint}
    product_categories::Vector{ProductCategory}
    products::Vector{SpecificProduct}
    requests::Vector{Request}
    vehicle_categories::Vector{VehicleCategory}
    vehicle_sets::Vector{HomogeneousVehicleSet}
end
```

A.2 Computed auxiliary data

```
mutable struct RvrpComputedData
    instance_id::String
    pickup_id2Index::Dict{String, Int}
    delivery_id2Index::Dict{String, Int}
    depot_id2Index::Dict{String, Int}
    recharging_id2Index::Dict{String, Int}
    product_category_id2Index::Dict{String, Int}
    product_id2Index::Dict{String, Int}
    request_id2Index::Dict{String, Int}
    vehicle_category_id2Index::Dict{String, Int}
    vehicle_set_id2Index::Dict{String, Int}
end
```

A.3 output

```
# To use the solution, the input data is required
struct Action
    id::String
    action_type::Int # 0- Depot, 1 - Pickup, 2 - Delivery, 3 - otherOperation
    operation_id::String #  associated request or depot id
    scheduled_start_time::Float64
end

struct Route
    id::String
    vehicle_set_id::String
    sequence::Vector{Action}
    end_status::Int # 0 returnToStartDepot, 1 returnToOtherDepot, 2 ongoing
    # path::OSRMpath     # TODO: add OSRM path here
end

struct RvrpSolution
    id::String
    instance_id::String
    cost::Float64
    routes::Vector{Route}
    unassigned_request_ids::Vector{String}
end
```