A proposed meta-model for combinatorial optimisation problems within transport logistics

Philip G. Welch, Anikó Ekárt, Christopher Buckingham

Aston University, Aston Triangle, Birmingham B4 7ET, UK
p.g.welch@aston.ac.uk, a.ekart@aston.ac.uk, c.d.buckingham@aston.ac.uk

Abstract

This paper proposes a meta-model that describes a variety of combinatorial optimisation problems within transport logistics. The aim of the meta-model is to represent real-world transport logistics problems in a manner that is (a) easy to use, (b) concise and (c) can be optimised. Suitable configuration of the meta-model allows the definition of many different known problems (e.g. Travelling Salesman Problem (TSP), Vehicle Routing Problem (VRP), arc-routing problems, pick-up delivery problems, periodic problems, etc...) as well as rich variants which may not be described in the literature. This approach aims to allow the definition and optimisation of different complex, real-world problems without having to implement a specialised algorithm for each problem.

Meta-model problem instances should be solvable for a broad class of problems using suitable variants of modern meta-heuristic techniques combined with discrete event simulation, providing modifications to a solution can be evaluated with sufficient speed. Suggestions are made as to how to perform these evaluations more efficiently.

1 Introduction

1.1 Commercial reality & real-world problems

Routing and scheduling problems within transport logistics have been the subject of many academic papers over the last few decades and various commercial systems are used within industry to optimise them. Typical commercial systems can optimise the ordering of collections and deliveries on vehicle route(s), the allocation of driver resources and many other key quantities. Although the published academic work does consider a wide variety of different problems, real-world problems often have many combinations of complex features which are rarely considered together and sometimes not considered at all in academic work (see Caric et al. [2] and Sörensen et al. [16]). Examples of this complexity from Sörensen's list include time windows, pick-up/delivery, special requirements for driver/vehicle, product types, work-break rules, heterogeneous fleets, separation of vehicle/trailer and various others. Additionally real-world problems can correspond to node or arc routing (or both), may have periodic constraints, can have loading order restrictions, involve multi-level distribution, multiple forms of transport and various other complexities not included in Sörensen's extensive list.

Less concrete information is available on the true capabilities of commercial systems (as their implementation details are often closely guarded commercial secrets [16]), although the SPIDER solver (see [7]) is a notable exception[1]. Although SPIDER does model many rich variants of the vehicle routing problem (VRP), its authors still state that it is easy to find interesting VRP variants which are not covered [7]. Furthermore, from the regular survey on these commercial systems published by the Institute for Operations Research and the Management Sciences (OR/MS) [1, 11] it is clear that although various rich features are included, routing installations still tend to require a large degree of customization for individual clients' problems. Similarly, Kleijn states that in the 1999 survey of the Dutch market, most companies' vehicle routing software was at least partially tailor-made to their individual needs [8].

Although commercial transport scheduling systems may tackle more rich and complex problems than their academic counterparts [16], in general they are not advanced enough to be able to tackle commonplace problems off-the-shelf, without additional and expensive bespoke software development.[2]

[1] SPIDER is based on academic research projects from 1996-99, for details see http://www.spidersolutions.no.

[2] P. G. Welch can also confirm from personal experience of working within the routing and scheduling software industry.

1.2 Academic work on rich problems

In recent years the need for rich models of routing problems has become more widely recognised by researchers [3, 6, 12]. These models tend to focus on defining a single very general, rich model of VRPs and have made an important contribution to the field.[3]

These general rich-model approaches use local improvement heuristics combined with evolutionary algorithms to provide sufficiently good, but generally not exact solutions. These heuristics are favoured over linear programming (LP) – which can provide optimal solutions – as although LP can be applied to problems with many different constraints, the size of real-world problems generally makes its use problematic, as noted by Goel and Gruhn [6]. Similar optimisation approaches relying on heuristics rather than LP have also been reported for commercial solutions by Sörensen [16] (although the commercial ILOG engine does use LP). The commercial SPIDER solver also uses a single, general and rich model approach with heuristic-based optimisation [7].

An alternative approach to rich problems, examined by Caric et al. [2], is developing an interactive programming environment and language for the solution of VRPs and providing many heuristics implemented in this language. Users can develop new programs and modify these heuristics for their particular VRP; allowing different problems to be tackled without defining a single 'general' model. The user would however need to be skilled in software development, as their language uses programmatic elements such as while-loops, SQL-like select statements, intermediary variables and subroutines. Their system is focused exclusively on VRPs and it is unclear if it is extensible to arc routing or wider scheduling problems.

1.2.1 Motivation behind this work

Although substantial progress has been made by authors considering general VRPs, these techniques are still not general enough to describe and solve many of the features of real-world problems. Moreover, it seems unlikely that a single general model could ever be general enough to consider all of these problem facets; the quest to find a single 'perfect' general model may be fundamentally flawed.

Of the alternatives to general models, linear programming can be very flexible but requires a complex mathematical formulation of the problem and has difficulties handling real-world sized instances. An alternative is to provide heuristic-based programmatic libraries and languages, but this still leaves much of the software development to be done when implementing a solution for a specific problem.

We therefore seek a different way to describe and optimise real-world problems, one that (a) sits between the extremes of a generalised single model and a programmatic framework and (b) exhibits better scaling than LP. This 'fourth way' is a meta-model that should be easy to use, with a concise problem description, to minimise the amount of learning and work (particularly programming work) required of the end-user. It should explicitly avoid the trap of attempting to formulate a single general model by providing an easy way to define and optimise different models.

Given the scaling problems exhibited by LP methods, we exclude them from the discussion in the rest of this work and instead focus on heuristic methods.

2 Proposed meta-model

2.1 The inevitable discrete event simulation

Evolutionary algorithms (EAs) are meta-heuristics which work by repeatedly making incremental changes to a solution or a population of solutions and evaluating those changes. Many different control mechanisms exist to decide upon, manage and accept/reject these changes (greedy search, genetic algorithms, simulated annealing, variable neighbourhood search etc.). EAs' popularity for logistics problems can be explained by the fact that they will always produce a solution. Defining the best suited EA for a problem

[3] The adaptive large neighbourhood search model of Pisinger and Ropke [12] has even been implemented in a commercial system - http://logvrp.com.

requires careful planning and considerable skill. Once a suitable EA has been defined, the solution's quality – or closeness to optimality – will mainly depend on the allocated time.

Given a solution for a VRP instance (i.e. an ordered set of stops for each route), a heuristic embedded within an EA can modify it by, for example, inserting a stop between two existing stops on a single route. This insertion will then need to be checked against the feasibility constraints for all subsequent stops in the route. Potvin and Rousseau [14] discuss this and note that for problems with simple capacity constraints said checks are straightforward and do not require a recalculation at each subsequent stop. However for problems with time windows, in the worst case scenario a recalculation can be required at each subsequent stop (although techniques are available to reduce the number of calculations needed, through the use of 'slack time'). The introduction of time-of-day dependent travel times between stops (to account for rush hours), will similarly force more recalculations when a route is modified.

It is therefore apparent that as a model becomes richer, in particular if the model's constraints are user-defined and hence 'unknown' to the optimiser algorithm, it will become difficult or impossible to evaluate the effect of a modification to a solution without doing a full recalculation for each stop (in VRP terminology) following the modification. In the very worst case scenario, different vehicle routes might interact (for example, through a single item which is transported through a large distribution network on several different consecutive vehicles), meaning that a modification to a single route may require the full recalculation for subsequent stops on multiple routes.

If we consider a stop or other task that needs to be scheduled to be an 'event', then as a routing or scheduling problem becomes richer, the evaluation function will in many cases tend to a discrete event simulation (DES). DES is a mathematical or logical model of a physical system that portrays state changes at precise points in simulated time (see Nance [10]). Events are simulated in chronological order and subsequent events can only be evaluated by first evaluating the preceding events - equivalent to a modification on a route requiring the recalculation of subsequent stops.

The need to perform full recalculations makes it far more expensive to evaluate the effect of a modification and hence at least $O(n)$ slower to optimise a given problem using DES as the evaluation function, for problems which do not normally require full-evaluation of stops after a modification. However it is our conjecture that given the transformation to a DES is likely to be an unavoidable side-effect of richer and user-defined problems, rather than trying to avoid it we should focus research efforts on how best to optimise problems subject to the limitations imposed. Provided effective ways can be found to optimise under these limitations, many rich problems can be defined and optimised with relative ease. This is because the logic required to update a 'system' (e.g. set of routes) with complex constraints is far easier to handle if we only consider the effect of a single modified event at a specific point in-time. The simulation itself should take care of the evaluation of the effect on subsequent events.

The insight that optimisation of schedules using DES as the evaluation function makes it far easier to implement very complex constraints has already been noted by a few authors [4, 18]. Deroussi et al. examined the use of DES to optimise schedules but within manufacturing systems, not transport logistics [4]. Zhongyue and Zhongliang describe using DES to simulate many complex constraints within the VRP, however they do not optimise the VRP or offer guidance on this [18]. Neither of these authors considered user-defined problems; the work presented here may therefore be unique in its in-depth examination of the considerable potential of using DES to optimise user-defined scheduling problems.

It should be noted that many works have been published concerning the combination of optimisation and simulation, including DES (e.g. Kurkin and Šimon [9]; Fu, Glover and April [5]). However, the overwhelming majority of these works examine the optimisation of designs or policies - for example, the layout of a manufacturing plant. Unlike this work, their decision variables are not based upon the ordering of events in the simulation and hence they address a very different type of problem.

2.2 Meta-model description

We propose a meta-model to describe various problems within transport logistics, where a single DES implementation can serve as the evaluation function for any problem definition, model instance and solution triplet. Figure 1 shows the high-level working of the meta-model. Within our meta-model the

problem definition is explicitly described by a data structure (i.e. no longer hardcoded). This definition is object-oriented and comprised of the following entities, together with user-defined properties, constraints and functions to update the properties:

- **Physical entities**
  - **Actor:** An entity that performs actions; typically a vehicle or a person. [4]
  - **Passive entities:** A physical entity that is affected by actions but does not perform them.
    - **Location:** An actor visits a location.
    - **Item:** An actor collects or delivers an item.
- **Non-physical entities**
  - **Event:** An event is a change in the state of the system (i.e. the entity properties) occurring at a single instant in-time (events therefore have zero duration).
  - **Action:** A series of at least two events; as an action is assumed to extend over a period of time it must have a start event and a stop event. Whole actions rather than events are assigned to actors.
  - **Objective:** The objective vector is an ordered real-valued vector, where each element is to be minimised (such as travel cost) or maximised (such as number of stops). Lower index elements are prioritised above higher index elements.
  - **Schedule and section:** The series of actions performed by an actor. A schedule is split into **sections** where each **section** has a minimum and maximum allowed number of actions, together with restrictions on which types of actions are allowed.
  - **Request:** A series of actions that should be performed in-order but not necessarily consecutively, by the same actor. An example could be **pick-up-item**, **deliver-item**.

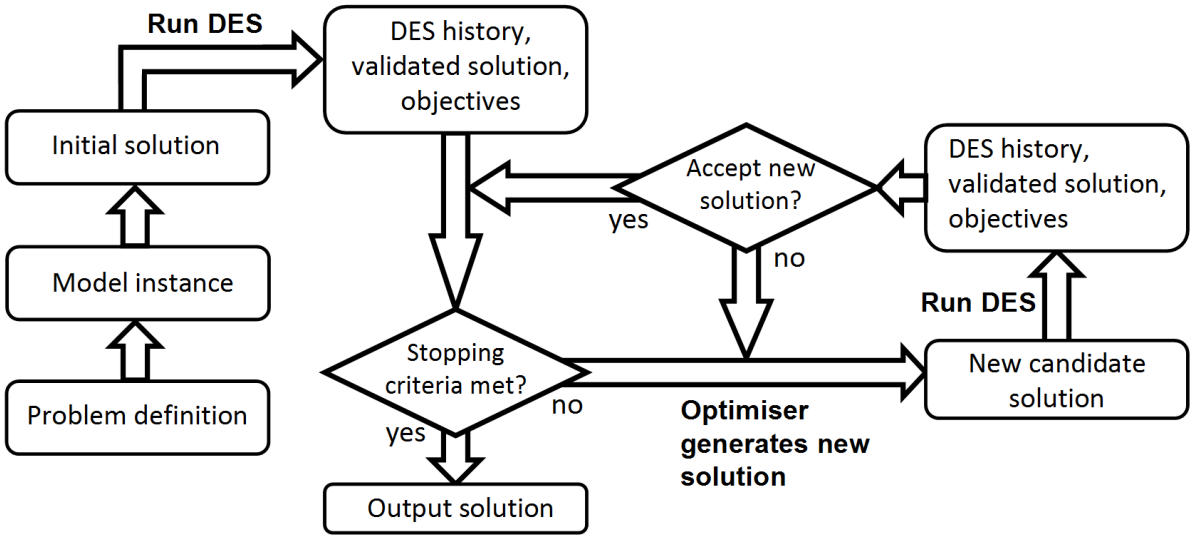

Figure 1: The proposed meta-model, embedded within a greedy algorithm as a simple example. The optimiser generates a new solution in the neighbourhood of the current solution, guided by the current solution's DES history.

Each entity has an associated user-defined set of properties which we term the state vector. State vectors are initialised with default values prior to the running of the DES and are updated during it. A

[4] Not to be confused with the term agent of multi-agent systems. Agents are defined to have beliefs, intentions and desires [17], our actors do not possess any of these.

shared state can also exist for a pair or set of entities. This would be required if, for example, we wanted to penalise multiple visits of a single actor to a single location in a pick-up and delivery problem; a shared state could exist for the actor-location combination that recorded how many times the location was visited by the actor. A penalty function could then be implemented based on this number.

User-defined constraints (UDCs) can be placed upon actions/events and upon physical entities. The UDCs on an event pertain to the state of the system at the present time, which allows the event to be executed. The UDCs on the actors, items, locations etc. pertain to states that the system cannot be allowed to enter. UDCs take the relevant state vectors as input and return true or false.

User-defined functions (UDFs) are employed to update the relevant state vectors when an event is executed. UDCs can either ADD or SET properties within the state vectors. An event is therefore primarily a series of calls to UDCs and UDFs. Specifically an event does the following:

1. Check the event's UDCs to see if the system is in a valid state to allow its execution.
2. Call the event's UDFs to calculate the new states of all the entities it needs to modify.
3. Check the UDCs of all modified entities to ensure no constraints have been violated.
4. Save the new states back onto the system state if no violations are found.

New types of entities are defined using specialisation of the base entity types. This allows, for example, a vehicle specialisation to be derived from the base actor type. A particular instance of a VRP would then use several instances of the vehicle specialisation (one per vehicle).

Part of the specialisation of an action involves defining which specialisations of actor and other physical entities are involved in the action. Thus given an actor specialisation **salesman** and a location specialisation **city**, we could define an action specialisation **visit-city** which involves **salesman** and **city**.

2.2.1 Implementation of UDCs and UDFs

The UDCs and UDFs are simple tree-based functions with a form similar to s-expressions in the Lisp programming language [15] and the tree representation used within genetic programming [13]. They are composed of parent and child nodes where a node can be a function (min, max, sum, and, or, ...) or a terminal (a constant value or in our case, a property from a state vector). By deliberate design the UDCs and UDFs cannot include advanced programmatic elements; we are not seeking to create a programming language and hence there are no loops, intermediary variables or subroutines. The lexicon of available functions and terminals is kept simple and more importantly, deterministic. For many problems the tree structure of the individual UDCs and UDFs is small and simple; often a UDF is a single terminal node storing a constant value.

The UDCs and UDFs can read the state vectors of entities involved in the event, with the exception of the objective. We do not allow them to read the objective state as this would encourage the structuring of problems where an actor is more likely to be dependent on another actor's events. For the same reason, UDCs and UDFs can only read from and write to a single actor - the actor performing the event.

3 Meta-model definition of the travelling salesman problem

As a simple example, we present a meta-model definition of the travelling salesman problem (TSP). Table 1 details the specialised entities which are used within our TSP definition and table 2 lists the specialised events and their UDFs. There are no UDCs within the definition as the TSP is unconstrained. All specialised actions act upon the **salesman** and **city** entities.

Moving to a different **city** is performed by the **move-to-city** action, which comprises of the single event **start-move-to-city**. The UDFs in the **start-move-to-city** event set the actor's location property (**current-city**), increment the actor's and the objective's **number-of-visits** properties and add the travel cost (typically distance or time) to the objective vector's **travel-cost** property using a built-in function which provides travel cost between locations.

Table 1: Specialised entities within the TSP

Specialisation	**Base entity**	**Properties**	**No.**
Salesman	Actor	Current-city, origin-city, number-of-visits	1
City	Location		n
Move-to-city	Action	City	n
Return-to-origin	Action		1
Objective	Objective	Number-of-visits (maximise), travel-cost (minimise)	1

Table 2: Specialised events within the TSP

Property	**Change**	**User-defined function**
Event Start move-to-city		
Objective(Number-of-visits)	Add	+1
Objective(Travel-cost)	Add	+ TravelCost{ Salesman(Current-location), Action(City)}
Salesman(Current-city)	Set	Move-to-city(City)
Salesman(Number-of-visits)	Add	+1
Salesman(Origin-city)	Set	**if** Salesman(Number-of-visits) = 0 **then return** Move-to-city(City)
Event Start return-to-origin		
Salesman(Current-city)	Set	Salesman(Origin-city)
Objective(Travel-cost)	Add	+ TravelCost{ Salesman(Current-location), Salesman(Origin-city) }

To allow the route to loop back to its beginning, we introduce a special **return-to-origin** action which returns to the first **city** visited by the actor. The actor's schedule is then constrained to two sections {any-number-of **move-to-city**}, {$1 \times$ **return-to-origin**}, forcing it to perform **return-to-origin** as its last move. The actor's first **city** is recorded on its **origin-city** property, during the first **move-to-city**.

An optimisation algorithm using this definition would seek to (a) maximise the objective's **number-of-visits** property, by adding visits to every city and then (b) minimise the objective's **travel-cost** property by making the route more efficient.

Other meta-model definitions of the TSP are possible. If the origin city was pre-chosen, we could define a new city specialisation **origin-city**. The model would then instantiate $(n-1)\times$ **city** objects and the schedule would be: {$1 \times$ **move-to-origin-city**}, {any-number-of **move-to-city**}, {$1 \times$ **move-to-origin-city**}. This would remove the need to explicitly store the **origin-city** as an actor property.

In another variant, the built-in **TravelCost** function could be replaced by a shared state for a pair of locations which held the travel cost. The matrix of travel costs would then be held by $n \times (n-1)$ shared states for each pair of locations.

Although the basic TSP does not require any end-of-action events, if we introduced time windows they would be needed. The actor would need a **time** property which would be incremented by the **start-move** event and checked against a time window UDC on the **end-move** event.

4 Extension to other problems

Vehicle routing problem The definition of the VRP is similar to the TSP, except that it has multiple actors, it makes use of a built-in actor property **time** and for each capacity constraint, it has a property storing the used (or free) capacity and another one storing the minimum or maximum limit. The actor's **time** property is used to define which event happens next during the simulation. Each constrained value is incremented / decremented by the specialised event(s) and has a UDC placed upon the state of the actor. If the actor was constrained by a maximum quantity, where **quantity** is a property of the **city**, then the **end move-to-city** event would add + **city(quantity)** to **actor(quantity)** and the UDC would be

actor(quantity) $\leq$ **actor(max-quantity)**.

We would also define an additional location type **depot** and **move-to-depot** actions. The actor's schedule would be $\{1 \times$ **move-to-depot**$\}$, {any-number-of **move-to-city**$\}$, $\{1 \times$ **move-to-depot**$\}$. Additionally as it is usual to prioritise the minimisation of the number of vehicles used above the travel cost, the objective vector would have a **number-of-vehicles** property which has a lower index than **travel-cost** and is incremented by one on the first **move-to-city** event (i.e. when the actor's **number-of-visits** property equals zero).

Arc-routing problems For arc-routing problems, the action to serve arc $A \rightarrow B$ between nodes A and B could be modelled comprising of four events: **start-move-to-A**, **end-move-to-A**, **start-move-to-B**, **end-move-to-B**. In the case where an edge A, B can be served in either direction by arc $A \rightarrow B$ or $B \rightarrow A$, a state vector would have to exist shared by nodes A and B with the property **served**. **Served** would be set to true on both actions $A \rightarrow B$ and $B \rightarrow A$. The **no. arcs served** property to be maximised on the objective vector would only be incremented if **served** equaled false. Additionally the two actions would need to be explicitly linked in some manner, to prompt an optimiser algorithm to frequently try swapping one action for the other.

Pickup-delivery problems These would require the use of a specialised item entity, together with specialised **pick-up** and **deliver** actions and a specialised request to link these two actions. These actions would decrement and increment the actor's **free-capacity** (or alternatively **used-capacity**) property.

Periodic problems Problems with repeat frequencies can be defined in various ways - for example, each section within the actor's schedule could be a different day with a property **day-number**. If an action to be repeated was part of a request, the request could hold a **last-performed-day** property, which is set equal to the schedule's **day-number** when an action in the request is performed. The action could then have a constraint which considered the difference between **day-number** and **last-performed-day**.

Restrictions on assignments of actors to actions Restrictions on assignments can be dealt with by simple constraint(s) on the action or first event in an action that would read properties from the actor, item, location, etc. as needed. If an actor needed a certain skill level to perform an action, they could have a property **skill-level** and the action could have a UDC **actor(skill-level)** $\geq$ **action(minimum-level)**. If the actor was a vehicle rather than a person, similar logic could be applied to only allow certain vehicles to transport certain types of items or even define a multi-depot VRP by only making certain depot locations available to certain vehicles.

Restricted access problems Restricted access problems can occur when, for example, a depot can only load or unload a limited number of vehicles at once. This could be described by a depot location having (a) a property **number-of-vehicles**, (b) a property **max-number-of-vehicles** and (c) a constraint **depot(number-of-vehicles)** $\leq$ **depot(max-number-of-vehicles)**. The event of a vehicle arriving at depot would increment **depot(number-of-vehicles)** and similarly the vehicle leaving would decrement this. The constraint would also need to be flagged as one that should cause the actor to wait, rather than abandon the entire action, if executing an event would break this constraint.

Restricted access problems are an example of the class of problems where the actors are not independent - i.e. the actions of one actor can affect the actions of another.

Unloading order restriction problems Some VRPs have last-in, first-out constraints; the last item to be loaded onto a truck must be the first to be unloaded. To model this, the vehicle could have a property **number-of-items** which is incremented and decremented every time an item is loaded / unloaded. When an item is loaded onto the vehicle, its **load-order** property is set equal to the vehicle's new **number-of-items**. The **deliver-item** action then has the constraint **vehicle(number-of-items)** $=$ **item(load-order)**.

Other scheduling problems Although the meta-model will be initially targeted towards routing problems, the existence of a 'location' is the only element of the meta-model that may be specific to routing problems. It it therefore likely that the meta-model can be applied to other scheduling problems such as people scheduling (e.g. service personnel or nurse rostering), machine scheduling and exam timetabling.

5 Adapting meta-heuristics to use the meta-model

5.1 Overview

Although the meta-model can be used to describe a wide range of scheduling problems, its usage would be severely limited without any means to provide optimised solutions for the problems it defines. Work on adapting suitable optimisation algorithms for the meta-model and DES is still in an early stage, although some helpful techniques have already been developed.

In a naive approach, moves (such as switching the positions of two actions within their respective schedules) could be generated randomly and then evaluated with a complete run of the DES (i.e. for all actors; not just the ones modified). This type of approach was examined using a simple genetic algorithm combined with the DES as the evaluation function, but as expected, proved to be far too slow to be useful (primarily as genetic algorithms do not make use of domain-specific knowledge other than what is embedded in the representation). Adapting evolutionary algorithms which make use of domain specific knowledge (e.g. variable neighbourhood search), requires efficient methods to (a) generate likely high-quality moves and (b) evaluate the effects of a move.

5.2 Proven techniques

The following techniques have been examined with a partial implementation of the meta-model, simulated using a DES and embedded within a simple greedy local search algorithm. The meta-model was configured for several basic test problems within transport logistics. Although this is work-in-progress and much work still needs to be done, these techniques have been shown to significantly improve the speed of the optimisation engine and hence the quality of the resulting solutions by (a) reducing the time needed to evaluate modifications to a solution and (b) restricting the number of evaluated modifications.

Key-framed simulation history Recording the history of the simulation state over the course of an entire run allows us to examine the system state at any point in time before/after an event occurs and can form the basis of improvement heuristics. This history can be recorded in a memory-efficient manner, with $\approx O(n)$ storage requirement, by storing key-frames for individual state vectors when they change.

As the recording of this history can create an overhead, in the current prototype we do not record the history when evaluating a move, although if the move is accepted, we re-run the DES to generate it.

Estimated identification of high-quality moves Given this key-framed simulation history, we can estimate the 'cost' (in terms of change to the objective vector) of inserting an action at a given time by (a) taking the simulation state at that time and then (b) executing only the events in that action and in some cases, the events in the action directly afterwards. This is a relatively quick operation. Although it does not give the effects of the inserted events on subsequent events and will not hold true in all circumstances, it does identify moves which are likely to be high-quality (provided they do not break constraints on later events). Likely high-quality moves can then be properly evaluated by trying the estimated best moves first. The move is properly evaluated by performing a DES run from the point in time where the modification occurred and is then accepted or rejected as appropriate. This technique uses domain-specific knowledge and is an approximate equivalent of performing the cheapest possible move (cheapest possible move is a well-known local search heuristic).

Caching the inputs and outputs of events The efficiency of the DES can be improved significantly by caching of per-event results. This relates to the first limitation imposed by the DES. Although the UDCs and UDFs cannot easily be understood by an optimisation engine, it is trivial to record which properties (and their values) are read from which state vectors, and how properties are set or incremented (i.e. record the event's inputs and outputs). As the UDCs and UDFs are deterministic, if all the inputs to the execution of an event are the same, then the outputs will also be the same. In the general case, actors can interact with each other (e.g. in restricted access problems via a location state). A full evaluation of a move modifying actor A's schedule must therefore also include the simulation of all other actors. However given this caching technique, in practice for many problems actor A would be simulated whilst the other actors would just be updated via the event cache. In some simple unconstrained problems (such as the TSP), or multi-shift problems where constraints may only apply on a per-shift basis, this could even allow for the event cache to be used for actor A events that occur *after* the modification point in actor A's schedule.

This highlights the possibility of another speed-up tactic. The effect of a modification to an existing solution may-or-may not propagate to other actors and may 'close' at some point in the future for the modified actor(s). It may therefore be possible to only simulate the 'change region' (in time and subset of actors), where the modification will alter the inputs and behaviour of subsequent events. The event cache already provides this functionality to some extent, but it has a significant overhead of checking that the inputs match against the recorded values for all events on all actors subsequent to the modification.

6 Conclusions and future work

The volume of research on tackling routing and scheduling problems is indicative of the difficulty in solving them and also the variety of their particular instantiations. Organisations with significant routing or scheduling operations will often have their own idiosyncratic processes and constraints that are difficult to fit into standardised approaches that can just be taken "off the peg" and applied. Scheduling systems that are more flexible need considerable programming expertise to configure and significant expenditure of time and money on consultants. This approach also makes it harder to adapt the systems to the organisations' operations as they inevitably evolve. The organisations will have to spend more money on programmers and consultants or have a continuous support contract.

Some middle ground is clearly needed between ready-made but inflexible software and software that can be adapted to an organisation only by experienced programmers. This paper describes an approach that attempts to reach this ground by modelling routing and scheduling problems in a flexible and user-configurable manner, using intuitively understood entities, operations, and constraints.

We have demonstrated that this proposed meta-model can describe a variety of transport scheduling problems and preliminary results give favourable indications that optimisation of a meta-model under DES is feasible. However many research questions remain, such as:

1. What is the most efficient method to evaluate modified solutions?
2. How broad is the range of problems which can be optimised using this approach?
3. How will the speed and scaling with problem size of the optimisation algorithm(s) compare to other heuristic techniques and linear programming?
4. How will the quality of solutions compare to traditional techniques?

The size and breadth of problems that can be tackled by this approach will only become known with a full implementation of the meta-model and significantly more work on the accompanying optimisation engine. Meta-model based optimisation using discrete event simulation is expected to be significantly slower than other heuristic techniques as it needs to do many more calculations and can make less use of domain-specific knowledge. However, it has high potential to provide very flexible models for scheduling-based combinatorial optimisation problems and there is a very clear need for this.

Acknowledgement

Work for this paper was supported by the European Commission through the 7th FP project ADVANCE (http://www.advance-logistics.eu) under grant No. 257398.

References

[1] OR/MS vehicle routing software survey. `www.lionhrtpub.com/orms/surveys/Vehicle_Routing/vrss.html`, 2010.

[2] T. Caric, A. Galic, J. Fosin, H. Gold, and A. Reinholz. *A Modelling and Optimization Framework for Real-World Vehicle Routing Problems*. 2008.

[3] G. Confessore, G. Galiano, and G. Stecca. An evolutionary algorithm for vehicle routing problem with real life constraints. In *Manufacturing Systems and Technologies for the New Frontier*. 2008.

[4] L. Deroussi, M. Gourgand, and N. Tchernev. Combining optimization methods and discrete event simulation: A case study in flexible manufacturing systems. In *Service Systems and Service Management, 2006 International Conference on*, volume 1, 2006.

[5] M.C. Fu, F.W. Glover, and J. April. Simulation optimization: a review, new developments, and applications. In *Simulation Conference, 2005 Proc. of the Winter*, 2005.

[6] A. Goel and V. Gruhn. A general vehicle routing problem. *European Journal of Operational Research*, 191(3), 2008.

[7] G. Hasle and O. Kloster. Industrial vehicle routing. In *Geometric Modelling, Numerical Simulation, and Optimization*, 2007.

[8] M.J. Kleijn. Vehicle routing software - a comparison for the Dutch market. *NEA Transport research and training*, 1999.

[9] O. Kurkin and M. Šimon. Optimization of layout using discrete event simulation. *IBIMA Business Review*, 2011.

[10] R.E. Nance. A history of discrete event simulation programming languages. In *The second ACM SIGPLAN conference on History of programming languages*, 1993.

[11] J. Partyka and R. Hall. On the road to connectivity, April 2010.

[12] D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Comput. Oper. Res.*, 34, August 2007.

[13] R. Poli, W.B. Langdon, and N.F McPhee. *A Field Guide to Genetic Programming*. 2008.

[14] J.Y Potvin and Rousseau J.M. An exchange heuristic for routeing problems with time windows. *Journal of the Operational Research Society*, 1995.

[15] P. Seibel. *Practical Common Lisp*. 2005.

[16] K. Sörensen, M. Sevaux, and P. Schittekat. multiple neighbourhood search in commercial vrp packages: Evolving towards self-adaptive methods. In *Adaptive and Multilevel Metaheuristics*, Studies in Computational Intelligence. 2008.

[17] M. Wooldridge. *An Introduction to MultiAgent Systems*. 2009.

[18] S. Zhongyue and G. Zhongliang. Vehicle routing problem based on object-oriented discrete event simulation. In *2nd International Conference on Advanced Computer Control*, volume 5, 2010.