A FRAMEWORK FOR DISCRETE-EVENT SIMULATION OF PICK-UP-AND-DELIVERY PROBLEMS

Anna Syberfeldt
University of Skövde
Virtual Systems Research Center
Skövde, Sweden
anna.syberfeldt@his.se

KEYWORDS
Discrete-event simulation, pick-up-and delivery, waste collection.

ABSTRACT

So called pick-up-and-delivery problems are encountered in everyday life – for example parcel transportation, dial-a-ride services, and collection of mail from mail boxes. For modeling these problems, discrete-event simulation is a powerful technique that is able to handle complexties in a transportation system. In this paper, a new framework for discrete-event simulation of pick-up-and-deliviery problems are proposed. The framework includes a general and extensible library of simulation modeling components applicable to different types of pick-up-and-delivery problems. For evaluation, the proposed framework is used in the simulation-optimization on a real-world pick-up-and-delivery problem of waste collection.

INTRODUCTION

A common problem within the area of transportation systems is the pickup-and-delivery problem (PDP). The PDP basically involves a set of transportation requests and a fleet of vehicles. Generally, each request specifies the size of the goods to be transported and the pickup and delivery points. Each vehicle in the fleet has a given capacity, a start location, and an end location. The objective is to satisfy all transportation requests while at the same time minimize the total cost, which may include carbon dioxid emission or transportation times, amongst others. There are plenty of examples of PDPs encountered in everyday life – for example parcel transportation, dial-a-ride services, and collection of mail from mail boxes.

Real-world PDPs are often highly complex as a consequence of combinatorial relationships, uncertainty factors, and non-linearities in the transportation system. When traditional, analytical methods are used to model these problems, simplifications must usually be done since these methods do not provide means for modeling complex behaviour (Al-Nory & Brodsky, 2008). Furthermore, the concepts of order and timing of events are usually not explicit in analytical models, which makes it hard to achieve a correct representation of the system (Al-Nory & Brodsky, 2008). For a complex PDPs, discrete-event simulation is a powerful alternative to analytical methods for modeling the transportation system. Discrete-event simulation provides means for incorporating complex relations using easier methodologies and tools compared to mathematical programming (Law, 2007). A simulation model essentially consists of a set of system states and transitions between these states, which correspond to elements and event sequences in the real system.

The next two sections continue with describing the PDP and discrete-event simulation in further detail. Thereafter, a new framework for simulating PDPs are described, followed by an evaluation of the proposed framework on a real-world PDP problem of garbage collection. The paper ends with a discussion of results and conclusions.

THE PICK-UP-AND-DELIVERY PROBLEM

The general pick-up and delivery problem (GPDP) is about creating routes for carrying out a number of transportation requests using a fleet of vehicles (Savelsbergh & Sol, 1995). Each vehicle has a given maximum capacity, a starting point, and an endpoint. A transport request defines the quantity of goods to be transported and the locations where it is to be picked up and delivered. Each request is served by a single vehicle without intermediate transhipment. The pickup and delivery problem (PDP) is a specialization of GPDP in which all vehicles start and stop at a central depot. Furthermore, for each transport request only one location is defined for pick up and delivery, respectively.

Based on GPDP and PDP further problem specializations can be defined. These problems can be divided into to categories; static and dynamic. A problem is static if all transport requests are known when the routes are constructed (Savelsbergh & Sol, 1995). In a situation where the transport requests may be added while running routes, meaning that routes have to be changed on-the-fly, the problem is instead dynamic. Besides static and dynamic problems, it is also meaningful to distinguish between problems with one operating vehicle (single vehicle) and several operating vehicles (multi-vehicle). Furthermore, problems can also be categorized according to if they involve time constraints or not. As an example, vehicles are often only available during the drivers' working hours, or if transporting people their pickup must be performed at a specific time. Such restrictions are called time windows. Different problem specializations can also be combined into problem formulations such as ”static multi-vehicle pickup and delivery problem with time windows”.

DISCRETE-EVENT SIMULATION

Simulations in event-driven simulation models are driven by a set of events that take place at pre-defined, fixed time

points (Banks, 1998). The events are either defined from the start, or generated as a result of other events during the simulation. The events are stored in an event list.

A simulation model contains a set of variables that holds the state of the system, the so-called state variables (Banks, 1998). Which state variables that are needed vary depending on what the simulation will be used for. Different applications require different output parameters, and hence different state variables to calculate the output parameters. Which state variables that are needed in the simulation model to simulate a particular system may vary depending on what the simulation should be used for. In an event-driven simulation model the values of the state variables may be changed only at the times when an event occurs. The opposite of this is so-called continuous models, where the state variables values are defined by mathematical equations and may change continuously over time (Banks, 1998).

The execution of an event-driven simulation model is controlled by the times points of the scheduled event. After an initialisation phase where the simulation clock is reset and the state variables, the statistical variables and the event list are initiated, the simulation cycle is started, that is the simulation main loop (Law & Kelton, 2000). In each iteration, the type of the next event in the event list is determined (ie the event which has the lowest start time). Subsequently, the simulation clock is advanced to the start time for this event and a routine for handling the current event type is run. An event routine first updates the state variables, then the statistical variables and, finally, future events are generated and added to the event list. The last thing that happens in the simulation cycle is to test whether the condition for when the simulation is considered to be completed is satisfied. If it is not, a new simulation cycle is run. If the condition is met, the simulation is terminated, and output parameters are calculated and reported.

In this study, a new framework for discrete-event simulation of PDPs is being prosed. This framework is described in the next section.

A DISCRETE-EVENT SIMULATION FRAMEWORK

This section describes the general and extensible library of simulation modeling components that is proposed for simulating PDPs. The library aims to be generally applicable to different types of PDPs. From an overall perspective, the library is divided into three subsystems: a) simulator, b) simulation data, and c) events. These subsystems are described below.

Simulator

The simulator subsystem (Figure 1) defines two main classes: `Simulator` and `Graph`. Simulator is an abstract class providing a general interface for PDPs. When simulating a specific variant of the general PDP (for example PDP with time windows), a subclass to Simulator is created by the user which defines all functionality needed for the specific implementation at hand. Two common operations are provided through the Simulator interface: Simulate and BuildRoute.

`Simulate` performs a simulation according to the given input parameters and returns the results from it. The input parameters are an ordered list of locations to be visited, the vehicle that is to be used, the vehicle operation hours, vehicle starting and ending points, and whether the simulation should be stopped prematurely if it is found to be invalid (this functionality can be used to save time). Note that in the optimization of multi-vehicle problems, the user must create a list of places to visit for each vehicle and call the simulation function once per vehicle.

`BuildRoute` perform a simulation under the given parameters while building a route object which is then returned. The route object contains a list of locations visited and the corresponding time of the visit. This object is built during the simulation based on subsequent locations visited in the simulation. The object also contains all the results returned from the simulator.

The motivation behind the use of two different functions (Simulate and BuildRoute) is that it is likely that the end user is interested in information about what time the vehicle will arrive at each location, while this information is not interesting for the optimization algorithm. Simulate is therefore intended to be used during the optimization, while BuildRoute is intended to be used afterwards to present detailed information to the user about which time different vehicles are estimated to arrive at each location.

The class Simulator also defines the enumeration type State, which is used during a simulation to determine what state the vehicle in the simulation currently are in. The state ON_THE_ROAD means that the vehicle left the previous location but not yet on the next spot. The state WAITING_TO_BEGIN_SERVICING_LOCATION means that the vehicle has arrived at a location but not yet started the service at the location. The state SERVICING_LOCATION means that the vehicle is currently undertaking the service.

The Simulator class also contains a graph object. The graph stores all the information about locations, distances, and travel times in the form of a neighboring array containing objects of the class Edge. The graph also holds an object of type EdgeDataReader. EdgeDataReader is an interface class used to load the edge data. An EdgeDataReader object is sent to the constructor for the simulator and then used to create the object graph used in Simulator.

Events

The subsystem Events (figure 2) provides functionality for event handling and is used internally in simulations. It includes two classes, `EventList` and `Event`, and the enumeration `EventType`. EventType is used to define an event type. The events are modeled as Event objects and stored in a list of type EventList.

EventType contains five different types of events. LOCATION_ARRIVAL occurs when a vehicle arrives at a

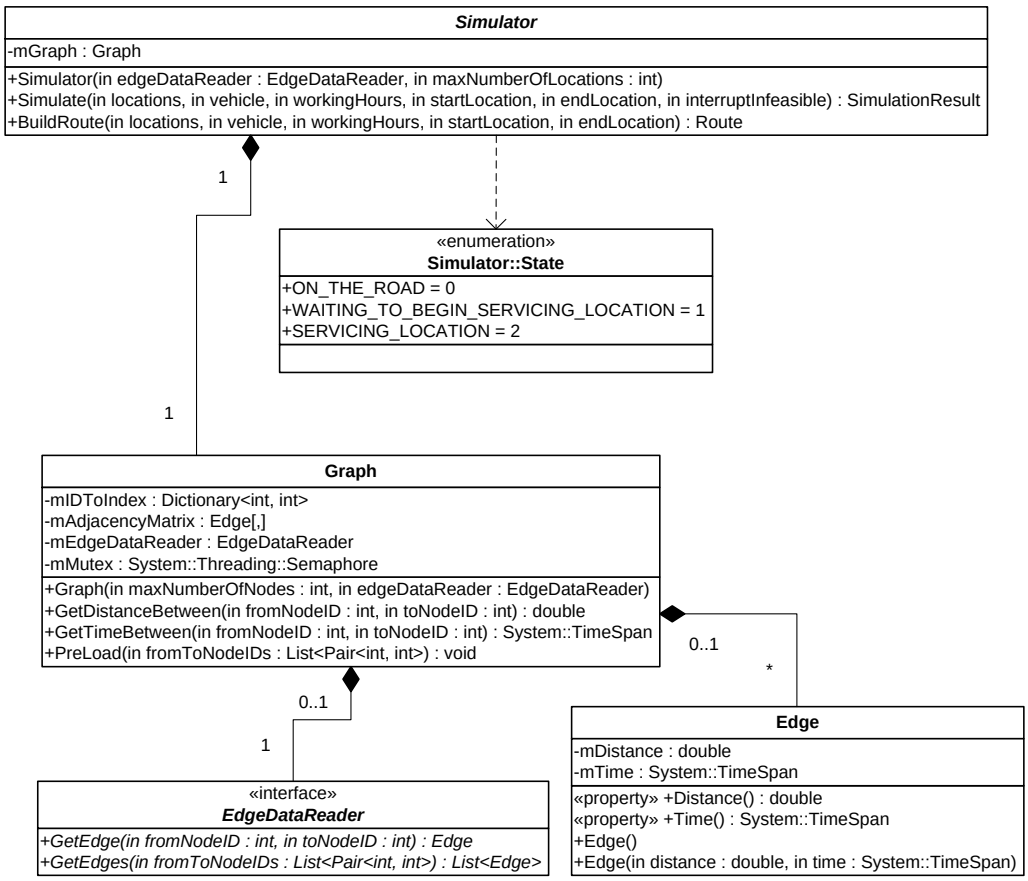

Figure 1: Components of subsystem Simulation

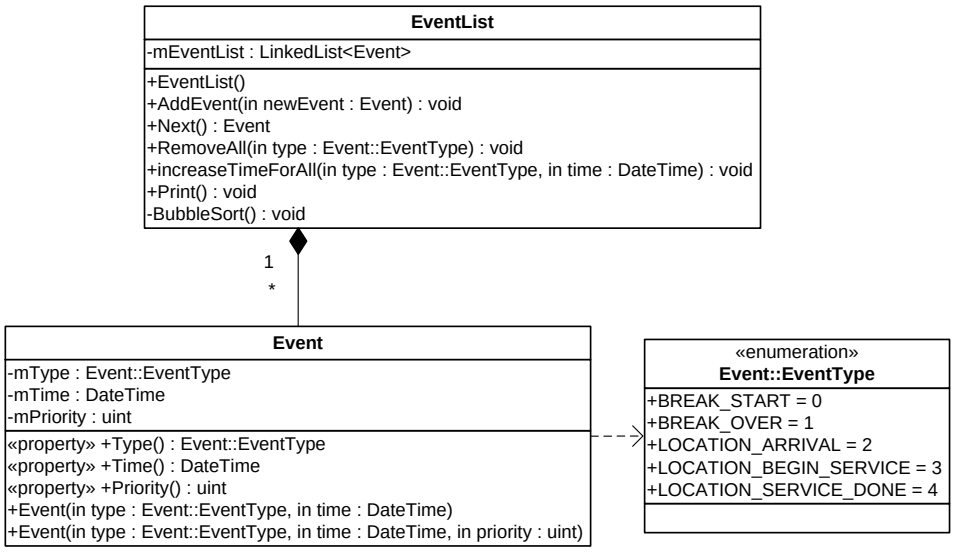

Figure 2: Components of subsystem Events

location. LOCATION_BEGIN_SERVICE occurs when a vehicle has arrived to a location and service is about to be started. LOCATION_SERVICE_DONE occurs when the servicing at a location is complete and the vehicle is ready to go on. BREAK_START occurs at the end of each session except the last. BREAK_OVER occurs at the start of each session except the first.

Each event has three attributes: type, time, and priority. Type is used by the simulator to determine what action should be taken when the event occurs. Time determines when the event occurs, and is used by the EventList in order to keep the events sorted, and also by the simulator in order to manage the simulation clock. Priority is used by the EventList to determine the order of events occurring at the same time.

The events are stored in a linked list in EventList. The list is kept sorted in ascending order according to time of occurrence, and in case two events have the same time of occurrence, according to priority. If two events both have the same time of occurrence and the same priority, the order is undefined. Four operations are possible on the event list: `AddEvent` (receives an event and adds it at the right place), `Next` (removes and returns the first event), `RemoveAll` (removes all events of a specific type), and `IncreaseTimeForAll` (increases the time with a given value for all events of a specific type).

Simulation Data

The subsystem Simulation Data (figure 3) provides classes for all non-common data that is provided for each simulation, and data to be returned from the simulator. This includes classes for representing vehicles, locations, time windows, routes and simulation results.

The class Location is a base class for places to be visited during a simulation. Its attributes are a list of time windows, a unique ID, and time consumption at the location. This is the minimal amount of information required for the simulator to be able to visit a location (the list of time windows may be blank, however). Two subclasses to Location are defined: `PickUpLocation and DeliveryLocation`. These subclasses both holds a Goods object. For a simulator to be able to perform more specialised activities at a location, further subclasses to Location need to be created.

The classes Route and SimulationResults are used to keep all data returned from the simulator's BuildRoute and Simulate functions. Route holds a list of pairs of locations and time points, which represent the locations to be visited, the order among them, and at what times. Route also contains a SimulationResult object. SimulationResult holds all information about performance of the system, including total time consumption, distance, travel time, end time, total transported weight, total transported volume, the total time all goods spend in the system, how much total maximum weight is exceeded by, how much total maximum volume has been exceeded by, the number of times a deliviery location has been visited before the corresponding pickup location, number of missed time windows, total delay with respect to time window, the total time deviation from the time window (in both early as the late arrival), and whether the simulation was interrupted prematurely.

The class Vehicle contains all relevant information about the vehicle used. Attributes of a vehicle includes maximum weight, volume, and velocity coefficient. The velocity coefficient is multiplied by the travel time between two locations that is returned from the graph. The purpose of having such coefficient is to allow vehicles of different velocities (for example, bicycles and trucks) to use the same underlying data for travel times between locations.

REAL-WORLD CASE STUDY

To evaluate the proposed framework, it is implemented on a real-world PDP concerning waste collection in a middle-sized municipality in Sweden. In general terms, the overall problem is to pick up waste on a number of predefined sites, and then leave it on landfill sites. The focus of the study is on larger waste containers located at companies and schools, rather than smaller household trash cans. Within the municipality, there are 182 such containers that the waste management organization is responsible for taking care of. Each of these containers hold either combustible waste, compost, or corrugated cardboard.

There is currently one truck dedicated only for collecting garbage from containers and the focus of the study is on this truck. The truck has a limited capacity in terms of both weight and volume, and when this capacity is reached must be emptying. The emptying is made either at a recycling central (compost and corrugated cardboard), or at a heat plant (combustible waste). Since different waste should be deposit at different places, different types of waste cannot be mixed in the truck.

Currently, each container is emptied once or several times during a 14-day period, according to a predefined emptying schedule. The laws specify that 14 days is a maximum limit on how long waste may be stored before it must be taken care of due to hygienic reasons. The 14-days' time limit means that time windows are present in the problem.

Today, the emptying schedule is defined by a transportation planner through a manual process based on domain knowledge. When the schedule is defined, the transportation planner must take into account the driver's working hours and breaks. Defining an efficient schedule is very hard to carry out manually, not at least since it is difficult to estimate how often or when the container needs to be emptied, which means that containers often only half full when the truck will or that they are overcrowded. Every day, the transportation planner provides the driver with a list of containers to be emptied for the specific day. The transportation planner does not specify which routes to take, but this must be decided by the driver itself. Performing a route optimization is very difficult to do manually due to the high complexity of the transportation network and lack of supporting tools. Even greater difficulty is created by temporal constraints that must

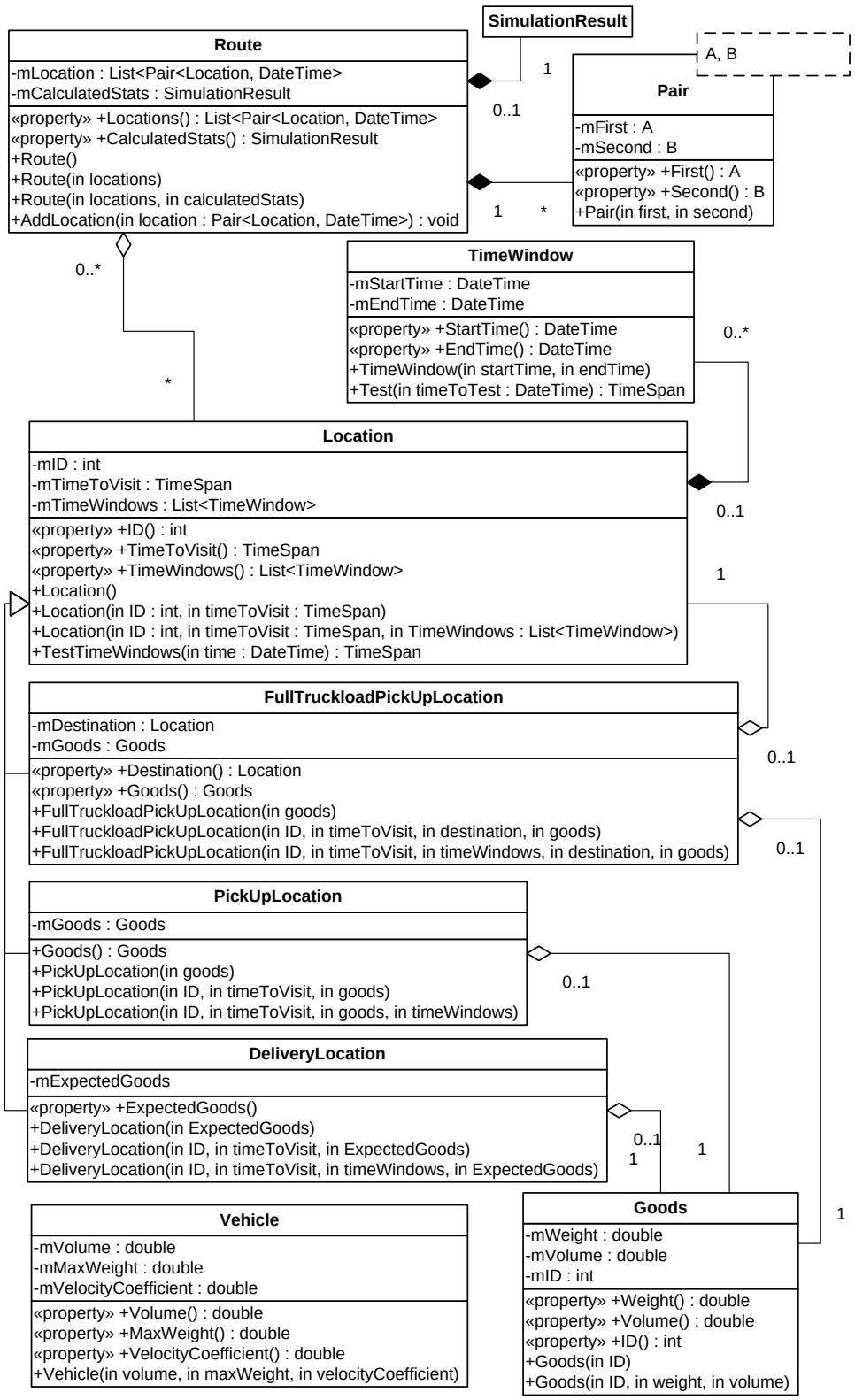

Figure 3: Components of subsystem Simulation Data

be taken into account, which means that some containers preferably should, or should not, be emptied at certain times (for example, containers at day care centers should be avoided when the kids are out playing).

Due to the high complexity of planning the waste collection and optimizing route, the waste management organization has expressed a need to improve the process by using simulation and optimization. The next section presents how this has been realized.

SIMULATION AND OPTIMIZATION

A simulation model of the waste collection procedure was built using the proposed simulation framework. Data for the model was collected during two months in corporation with the waste management organization. Validations of the simulation model indicates that it represents reality very well (average deviation from reality is 7%). For optimizing the problem, an optimization module has been developed and connected to the simulation. Two objectives are considered in the optimization, as specified by the waste management organization: 1) minimal carbon dioxide emissions (basically driving distance), and 2) minimal violations against temporal constraints. These two objectives are conflicting with each other, as considering temporal constraints negatively affects the driving distance.

The difficulty with conflicting optimization objectives is that there is usually no single optimal solution with respect to both objectives. Instead of a single optimum, there is a set of optimal trade-offs between the conflicting objectives, called Pareto-optimal solutions (Deb, 2004). In order to manage multiple objectives, specific optimization algorithms have been suggested. Instead of only seeking a single optimum, these algorithms maintain a set of Pareto-optimal solutions. One of the most efficient algorithms for Pareto optimization is the elitist non-dominated sorting genetic algorithm (NSGA-II) (Deb et. al, 2000). Due to the algorithm's proven efficiency, this is the optimization algorithm selected for the study.

NSGA-II is basically a genetic algorithm extended with features for handling multiple trade-off solutions. In this study, a solution consists of a collection of routes (a specification of which containers to visit a specific day). Like an ordinary genetic algorithm, NSGA-II maintains a population of solutions and refines these solutions through generations. In the refinement process, recombination (crossover) and mutation are being used to create offspring solutions that take part of the next generation. The basic steps involved in evolution are presented below (a complete description of genetic algorithms can be found in Deb 2004).

```
Initialize population
Evaluate the fitness of solutions in the
population
repeat
  Select solutions to reproduce
  Form a new generation of population through
    crossover and mutation
  Evaluate the new solutions
until terminating condition
```

Contrary to an ordinary genetic algorithm, however, NSGA-II selects solutions for the next generation of the population based on Pareto ranks. Rank 1 includes the Pareto-optimal solutions in the complete population, and rank 2 the Pareto-optimal solutions identified when temporarily discarding all solutions of rank 1, and so on. More specifically, the selection of solutions for the next generation is done from the set R, which is the union of the parent population and the offspring population (both of size N). Pareto-based sorting (also called non-dominted sorting) is applied to R and the next generation of the population is formed by selecting solutions from one of the Pareto fronts at a time. The selection starts with solutions in the best Pareto front, then continues with solutions in the second best front, and so on, until N solutions have been selected. If there are more solutions in the last front than there are remaining to be selected, niching is applied to determine which solutions should be chosen. All the remaining solutions are discarded. The selection procedure is illustrated in Figure 4 (adopted from Deb, 2004).

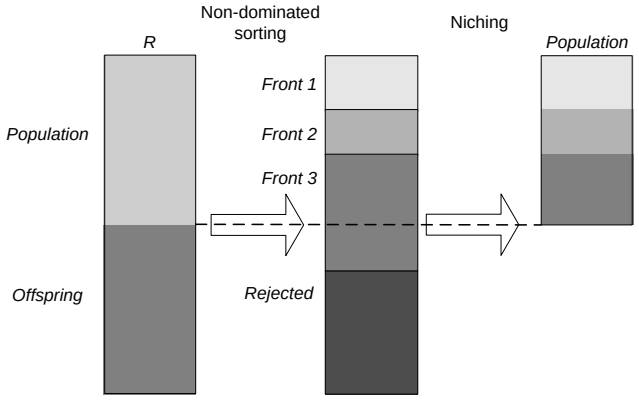

Figure 4: Selection procedure of NSGA-II.

The NSGA-II algorithm has been implemented in an optimization module that is connected to the simulation. The simulation-optimization process is an iterative loop in which the NSGA-II algorithm generates solutions that are evaluated by the simulation (figure 5).

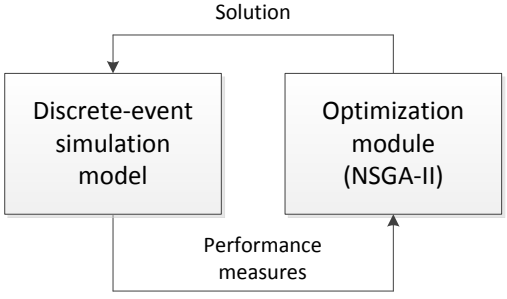

Figure 5: Simulation-optimization loop.

RESULTS

The simulation-optimization was evaluated using different real-world scenarios and the test results were carefully evaluated by the waste management organization. In comparison with solutions created manually by the transportation plannering, the simulation-optimization is able to find solutions with 13-28% lower carbon dioxide emissions (depending on scenario) while maintaining all temporal contraints. About 10 000 simulation runs are needed to reach these results, corresponding to a runtime of approximately 5 hours.

Compared to the manual approach of creating schedules used by the transportation planner, the simulation-optimization implemented has advantages not only with respect to the results, but also with respect to efficiency. Since no manual control or intervention in the simulation-optimization process is necessary, a lot of time and effort are saved for the transportation planner.

CONCLUSIONS

This study describes a new framework for simulating pick-up-and-delivery problems based on the discrete-event simulation paradigm. Compared to traditional, analytical methods, discrete-event simulation is powerful in modeling complex behavior and achieve a close representation of reality. The proposed framework includes a general and extensible library of simulation modeling components applicable to different types of pick-up-and-delivery problems. For evaluation, the proposed framework are used to model a real-world pick-up-and-delivery problem of waste collection. Validations of the simulation model indicates that it represents reality closely with an average error of only 7%.

To improve the waste collection process, the simulation model is connected to an optimization module. In the optimization, two objectives are considered: 1) minimal carbon dioxide emissions, and 2) minimal violations against temporal constraints. The two objectives conflict with each other, as improving performance on one of them deteriorates performance of the other. For handling the conflict, an optimization algorithm utilizing the Pareto concept is being used, called NSGA-II. Results from the study shows that the simulation-optimization is successful in improving the waste collection process, being able to find solutions with 13-28% lower carbon dioxide emissions while maintaining all temporal contraints.

To improve the optimization results further, future work includes studying how domain expert knowledge can be captured and incorporated in the optimization process. A human expert may have extensive knowledge valuable for the optimization, and incorporating this knowledge in the optimization strategy may be a way to obtain faster and more accurate optimization results. Combining the simulation with heuristics based on human knowledge is therefore an interesting topic for the future.

Another aspect that should be considered in future work is to reduce the time consumption of the simulation-optimization procedure. In the real-world case study, the simulation-optimization took about 5 hours to complete on a single PC. If more trucks and pick-up-locations are added to the problem, the runtime will be even longer. It can be noticed that the long runtime is not due to the simulation framework as such, but due to the enormous search space (e.g. a large number optimization iterations is needed), in combination with a heavy road database. Using the road database, which is provided by the Swedish Transport Administration, is necessary in order to obtain information about distances, speed limitations, traffic regulations, etc. A runtime of several hours is too long if the procedure is to be used on a regular basis (e.g. daily). Reducing the time consumption is, however, no easy task as real-world pick-up-and-delivery problems are highly complex. One way to address the problem might be to include a computationally efficient simulation metamodel in the proposed framework. A metamodel is essentially an approximation of the simulation and the underlying data. It has previously been shown that by adopting metamodels, the computational burden of the optimization process can be greatly reduced since the computational cost associated with running a metamodel is negligible compared to a simulation run (Jin et. al 2002). A metamodel could, however, never fully replace a simulation model as it is only an approximation.

REFERENCES

M. Al-Nory and A. Brodsky. “Unifying simulation and optimization of strategic sourcing and transportation”. *Proceedings of the 2008 Winter Simulation Conference*, Miami, Florida, pp. 2616-2624. 2008.

J. Banks. “Handbook of Simulation”. New York: John Wiley & Sons, Inc. 1998.

K. Deb, A. Pratap., S. Agarwal, and T. Meyarivan. “A fast and elitistmulti-objective genetic algorithm NSGA-II”. KanGAL Report 2000001, Indian Institute of Technology Kanpur, India. 2000.

K. Deb. “Multi-objective optimization using evolutionary algorithms”. Chichester: John Wiley & sons, Ltd. 2004.

A.M. Law “Simulation Modeling and Analysis”, Fourth Edition, McGraw-Hill, 2007.

Y. Jin, M. Olhofer, and B. Sendhoof, “A Framework for Evolutionary Optimization With Approximate Fitness Functions”, IEEE Transactions on Evolutionary Computation, Vol.6 No.5 (Oct), 2002, pp. 481-494.

M.P.W Savelsbergh, M. Sol. The general pickup and delivery problem. *Transportation Science*, 29(1), pp. 17-29. 1995.

BIOGRAPHY

ANNA SYBERFELDT is a senior researcher at the University of Skövde, Sweden. She holds a PhD in Computer Science from the De Montfort University, UK and a Master's degree in Computer Science from the University of Skövde, Sweden. Her research interests include simulation-based optimization, artificial intelligence, metaheuristics, and advanced information technology with applications in logistics and manufacturing. Her email address is anna.syberfeldt@his.se.

ACKNOWLEDGEMENTS

This work has been carried out within the SOL project which is partially financed by the Knowledge Foundation (KK Stiftelsen), Sweden. The author gratefully acknowledge the Knowledge Foundation for the provision of research funding and also the waste management organization Avfallshantering Östra Skaraborg for their support in this study. A special thanks also to the reviewers of the paper for providing helpful feedback.