

Towards a Model and Algorithm Management System for Vehicle Routing and Scheduling Problems

M. Desrochers †

GERAD, Montreal

C.V. Jones

Aspen Technology

J.K. Lenstra

Eindhoven University of Technology; CWI, Amsterdam

M.W.P. Savelsbergh

Georgia Institute of Technology

L. Stougie

Eindhoven University of Technology

Dedicated to the memory of Martin Desrochers by his co-authors

November 4, 1998

Towards a Model and Algorithm Management System for Vehicle Routing and Scheduling Problems

M. Desrochers †

GERAD, Montreal

C.V. Jones

Aspen Technology

J.K. Lenstra

Eindhoven University of Technology; CWI, Amsterdam

M.W.P. Savelsbergh

Georgia Institute of Technology

L. Stougie

Eindhoven University of Technology

Dedicated to the memory of Martin Desrochers by his co-authors

We outline a system for supporting the modeling of physical distribution situations and the selection or construction of algorithms for their solution. The system will incorporate the current knowledge and expertise in vehicle routing. Its main contribution will be in the design and implementation of techniques for representing and manipulating this knowledge.

Keywords: vehicle routing, model and algorithm management, knowledge base, inference techniques.

1. Introduction

Since the papers by Dantzig and Ramser [9] and Clarke and Wright [7], the development of optimization and approximation algorithms for vehicle routing and scheduling has been a central activity in operations research. The interest in the area is motivated by the importance of effective and efficient methods for handling physical distribution problems and by the intriguing nature of the underlying combinatorial optimization models. This is not to say, however, that all issues have been dealt with satisfactorily. In particular, the application of theoretical achievements in practical situations is still very much an *ad hoc* affair. Due to the great diversity of vehicle routing and scheduling problems encountered in practice and the large number of available algorithms, one has to be an experienced distribution manager as well as an operations researcher in order to be able to select a method that is well suited for a specific situation. To facilitate this decision process, we propose to develop a model and algorithm management system that provides support in modeling problem situations and in suggesting algorithms that might be applicable to the resulting models.

The system will represent and manipulate information at three different levels. At the first level, there is the *real-life problem situation*. It may contain many aspects that are not relevant for the selection of a solution method. At the second level, there is the *abstract problem type*. It is obtained from the real-life problem situation by determining and modeling the relevant entities that describe this situation in terms of decisions, objectives and constraints. At the third level, there are the *algorithms*. One that appears to be suitable in the situation at hand is selected or constructed.

The knowledge and expertise that must be built into the system concern two different issues. On the one hand, there is the knowledge and expertise that is applied to obtain an abstract representation of the problem situation. On the other hand, there is the knowledge and expertise that is applied to choose from among the multitude of available algorithms one that is appropriate for this model.

There is a vast literature on vehicle routing and scheduling that contains the knowledge and expertise of either type [6, 12, 14, 18]. An interesting aspect of the project is that the knowledge exists and the question is how to formalize its use. In order to meet this challenge, we have to create a vocabulary for representing the knowledge and to design inference algorithms for manipulating it.

Desrochers et al. [11] propose a language to define abstract problem types in vehicle routing and scheduling. In the problems considered a number of vehicles, stationed at one or more depots, have to serve a collection of customers in such a way that given constraints are respected and a given objective function is minimized. To define one such problem type in a formal way, the language uses four fields. The first field describes the characteristics and constraints that are relevant only to single addresses (customers and depots). The term 'address' is used instead of 'customer' because of the great variety of customer types: apart from the usual single-address customer, there is also the customer corresponding to an origin-destination pair or to all the addresses located on a street segment. The second field specifies the characteristics relevant only to single vehicles. The third field contains all problem characteristics that cannot be identified with single addresses or single vehicles. The fourth field defines one or more objective functions. The classification scheme is given in Appendix A.

A fifth field may be added to describe additional information about a specific class of problem instances. Although such information does not belong to the proper model as defined in the first four fields, it can still be useful for the selection of a suitable algorithm. For example, it might be helpful to know the average number of addresses that are to be assigned to one vehicle.

In this paper we discuss the components of the system that select or construct a suitable algorithm for an abstract problem type. Section 2 gives a general outline of the system, Section 3 deals with the representation of knowledge, formulations, and algorithms, Section 4 discusses their manipulation, and Section 5 contains some preliminary observations regarding an initial implementation of a system of this type.

Who should read this paper? Anyone who is interested in putting more science into the art of modelling. In particular, the paper is oriented towards researchers in model management, as it focuses on some of the fundamental issues in that area: model representation, model integration, meta-knowledge (i.e., knowledge about reasoning about models), and meta-knowledge representation. It may also be of interest to researchers in artificial intelligence that investigate those issues. Furthermore, it should appeal to the decision support and expert systems community, since we propose an inductive case-based reasoning approach.

And why should they read it? Not because it will report on the design and implementation of an operational system; our work is by no means completed. The reader will find a challenging problem. The automatic or semi-automatic construction of a good algorithm for a given vehicle routing problem is a research task of evident scientific interest and practical relevance, and the literature on the subject is scarce. The reader will also find the outline of a path toward a solution, and a few steps along that path: the development of a framework for a knowledge base for vehicle routing problems, models and algorithms, the start of an implementation, and the identification of the remaining difficult challenges. Finally, we observe that, although vehicle routing and scheduling is chosen as primary problem domain, the methodology presented is more general and could be applied to other areas as well.

Several research endeavors are in some sense related to our proposed model and algorithm management system. Two of those deal with a class of deterministic machine scheduling problems. The MSPCLASS program of Lageweg et al. [20, 21] uses a classification scheme similar to ours, and binary relations between the problems in the class so as to keep track of results on their computational complexity. The input of the program consists of two listings of known easy and known hard problems. The program uses the structure of the problem class to determine the implications of these results. The output provides a listing of essential results in the form of maximal easy and minimal hard problems as well as listings of minimal and maximal open problems. Rubach [31] developed a library of Pascal programs that solve machine scheduling problems and an environment to access these programs. The environment includes a component that uses the structure of the problem class to determine whether the library contains an algorithm for the problem itself, for a problem to which it can be reduced, or for a problem that is in some other sense close to it.

Blanning [4] proposed a framework for automatically combining existing models in order to solve a particular problem. Liang and Jones [26] developed a different framework, though having similar goals. Liang [25] proposed the use of analogical reasoning for the automatic construction of models. These papers did not discuss the details of how to generate a high-quality combination of models; the level of model complexity was rather limited. We attempt to address that concern by focusing on a specific application area.

The work of Lee et al. [23, 24] is also relevant to our paper. It considers the automatic identification of a problem given its mathematical formulation, which is in a sense the inverse problem of the one considered in the present paper. These authors pursued their research as a step towards the automatic development of decomposition algorithms.

The DecisionNet project of Bhargava and Krishnan (see [1]) is also related. Their goal is to publish working models on the Web. Visitors to DecisionNet will input the characteristics of their problem, and get a relevant model in return.

2. Model and algorithm management

Model management and model management systems are relatively new concepts, which have emerged from the interest in decision support systems, expert systems, and artificial intelligence. The primary objective of a model management system can be viewed as the counterpart of that of a database management system. It provides an environment for storing, retrieving, and manipulating models. A model management system serves as a bridge linking the decision maker's environment with the appropriate models. The current design paradigm for these systems stresses the need for expert knowledge in the system along with associated knowledge-handling facilities.

The ultimate goal of the system discussed in this paper is to provide its user with a suitable algorithm for the problem situation he is facing. In order to achieve this goal, the system will maintain models and algorithms, and will contain inference mechanisms to manipulate them. Because the system will be used as a consultant, it should be able to provide explanations of its line of reasoning, i.e., it should explain why it is asking particular questions and how it has reached a conclusion. It should also provide means to perform sensitivity analysis, for instance by allowing the user to attach confidence factors to his responses to the system's queries.

The system will go through two phases. In the first phase, it asks a set of questions in a man-machine dialogue in order to decide upon the problem type, the characteristics of the expected problem instances, and the algorithmic requirements. The classification scheme mentioned in the introduction is used to represent problem types. The characteristics of the expected problem instances, such as the average number of customers in a route and the average load factor of the vehicles, supplement the information embodied in the problem type. The algorithmic requirements reflect the

type of algorithm the user wants. For example, a user might be interested only in very fast algorithms, or in algorithms that produce an optimal solution. The characteristics of the expected problem instances and the algorithmic requirements have a large impact on the inference mechanisms the system will employ in the second phase.

In the second phase, the system tries to select or construct a suitable algorithm based on the knowledge residing in the system. Two classes of algorithms are distinguished. The first contains algorithms that are based on a mathematical programming formulation, for instance the generalized assignment heuristic [15], the second class contains all others, for instance the savings method. This differentiation is motivated by the fact that a mathematical programming formulation often reveals structural information about a problem that may be used in determining which algorithm should be applied. In the future, a third class may be added: dynamic programming algorithms. Here too, the underlying state space and the recursion equations may provide useful information.

The system's distinction between classes of algorithms is reflected in the knowledge organization. There are four different knowledge bases: a problem knowledge base (PKB), a formulation knowledge base (FKB), an algorithm knowledge base (AKB), and a general knowledge base (GKB). The PKB is a set of well-known and well-investigated problem types, such as the traveling salesman problem and the standard vehicle routing problem. The FKB and AKB contain mathematical programming formulations and algorithms for these problems, respectively, and the GKB contains knowledge on formulation and algorithm construction. Together they represent the expertise of researchers in the field as well as stacks of literature.

The problems as defined by the classification language must be viewed as abstractions that belong to the PKB, but the mathematical formalism in which they are described is not made explicit. The FKB, however, contains problem representations in terms of a mathematical formalism.

The selection or construction of a suitable algorithm for a given problem type proceeds as follows. First, if the problem type is present in the PKB, we are done; in that case, there is at least one associated algorithm in the AKB. Second, if the problem type is not present in the PKB, we try to identify 'related' problems in the PKB and use their associated formulations, if any, and associated algorithms to piece together one or more promising algorithms for the problem type at hand; this is called 'analogical reasoning' in the artificial intelligence literature. Finally, if the problem type is not present in the PKB and there are no related problems in the PKB, we cannot handle it.

The second step is, of course, by far the most intriguing and difficult one. There are a number of issues that need further consideration. We indicate them briefly here and elaborate on them in the following sections.

Related problems. If a problem type is not present in the PKB, the PKB is searched for related problem types, which will then be used to construct an algorithm for the problem type at hand. In order to relate problem types to each other, we need to define a metric on the set of abstract problem types.

Model integration and validation. Model integration, i.e., building composite models and decomposing models into their constituent parts, is used to create a mathematical programming formulation for a problem that does not belong to the PKB. Some form of validation has to be performed on the obtained formulation to ensure that it deals with all the constraints of the problem.

Algorithm integration and validation. Alongside model integration, there is algorithm integration, i.e., building composite algorithms and decomposing algorithms into their constituent parts, so as to construct an algorithm for a problem not present in the PKB. Validation, in this case, should also test whether the algorithm obtained meets the algorithmic requirements.

3. Representation

A major question pertaining to the system is that of representation. How do we represent knowledge, formulations, and algorithms? This is a crucial issue since all manipulations, i.e., the kinds of inferences that can be made, depend directly upon this structure.

There are several approaches to knowledge representation [29, 30]. With regard to modeling and decision support, most knowledge representation approaches can be divided into three categories, which are based on first-order predicate logic, networks, and frames, respectively.

Logical representation schemes employ the notions of constant, variable, function, predicate, logical connective, and quantifier to represent facts as logical formulas in some logic. A knowledge base, according to this view, is a collection of logical formulas which provides a partial description of reality. Modifications to the knowledge base occur with the introduction and deletion of logical formulas. In these schemes, logical formulas are the atomic units for knowledge base manipulation.

Network representation schemes, often called *semantic networks*, attempt to describe reality in terms of objects (nodes) and binary associations (labelled edges), the former denoting individual entities and the latter binary relationships between them. According to the network representational view, a knowledge base is a collection of objects and associations, or a labelled directed graph, and modifications to the knowledge base occur through the insertion and deletion of objects and the manipulation of associations.

Frame-based representation schemes view a knowledge base as a frame system. A frame is a complex data structure for representing a stereotypical situation. The frame has slots for the objects that play a role in this situation as well as for the relations between these objects. Attached to each frame are different kinds of information, such as how to use the frame and default values for its slots. A further feature of this approach is the concept of a frame system, which is a collection of frames linked together by an information retrieval network. The main function of the frame system is to provide a retrieval capability for matching frames with reality or some part thereof. Frames were conceived originally for visual applications [28], but have been used for scheduling [19] and as a major component of the 'model abstraction' concept [13].

In the current design, the model and algorithm management system uses a network representation scheme to represent knowledge about related problems, a logical representation scheme to represent knowledge about formulations, and a frame-based representation scheme to represent knowledge about algorithms.

3.1. Formulations

Most mathematical formulations for vehicle routing and scheduling problems are mixed integer programming models. The system should therefore be able to represent a mixed integer programming formulation in a manageable form. The research done on *modeling languages* for mathematical programming might prove useful in this context. Practical large-scale mathematical programming involves more than just the application of an algorithm to minimize or maximize an objective function subject to constraints. Before any optimization routine can be invoked, considerable effort must be expended on formulating the underlying model and generating the requisite computational data structures. Modeling languages are designed to make these steps easier and less error-prone. Examples are GAMS [3, 5], MP-MODEL [10], MPL [27], AMPL [16] and AIMMS [2].

An alternative to these might be *structured modeling* [17]. The overall objectives of structured modeling are to provide a formal mathematical framework, language, and computer environment for conceiving, representing, and manipulating a wide variety of models. Structured modeling has benefited significantly from ideas from modeling languages, spreadsheet modeling, and database theory. It purports to be more widely applicable than a modeling language, because it is not

restricted to mathematical programming models.

At this point, the best choice for our purposes seems to be AMPL [16]. An AMPL model consists of five sections: sets, parameters, variables, objective, and constraints. The sets and parameters sections define the data of the problem, the variables section defines the decisions that have to be taken, and the objective and constraints sections define the problem characteristics.

AMPL is intentionally rather close to the mathematical form while still being easy to type on an ordinary keyboard and to process by a computer program. There are constructions for each of the five sections listed above and ways to write arithmetic expressions, sums over sets, and so on. Sets are a fundamental component of an AMPL model. Almost all of the parameters, variables, and constraints in a typical model are indexed over sets, and many expressions contain operations over sets. Set indexing is the feature that permits a concise model to describe a large mathematical program. A large optimization model invariably uses many numerical values. In AMPL a single named numerical value is called a parameter. Parameters and other numerical values are the building blocks of the expressions that make up a model's objective and constraints. The variables of a linear or integer program have much in common with the numerical parameters. Both are symbols that stand for numbers and that may be used in arithmetic expressions. Parameters values are supplied by the modeler, while the values of variables are determined by the optimization algorithm. The objective and constraints are formed by linear expressions in previously defined sets, parameters, and variables, and define the problem characteristics.

We believe that for vehicle routing and scheduling models, a limited number of decision variables, listed in Figure 1, suffices, and that the many problem characteristics can all be covered by a small set of generalized characteristics, listed in Figure 2.

arc assignment	$x_{ij}^k = \begin{cases} 1 & \text{if arc } (i, j) \text{ is traversed by vehicle } k \\ 0 & \text{otherwise} \end{cases}$
node assignment	$y_i^k = \begin{cases} 1 & \text{if node } i \text{ is visited by vehicle } k \\ 0 & \text{otherwise} \end{cases}$
flow	$z_{ij}^k =$ amount of commodity k (such as type of good or type of vehicle) traversing arc (i, j)
resource utilization	$D_i =$ amount of scarce resources (such as time) utilized at node i

Figure 1. Types of decision variables.

arc assignment	forces the selection of one or more arcs
node assignment	forces the association of a vehicle with a node
position assignment	forces the selection of a node for a given position in a tour
flow conservation	relates the inflow of a node to its outflow
no subtours	forces a depot to be on all feasible tours
(<classification element>)	forces feasibility with respect to the characteristic modeled by a given classification element

Figure 2. Types of generalized problem characteristics.

However, in the model and algorithm management system for vehicle routing and scheduling problems, we not only want to represent mathematical programming formulations, we also want to manipulate them. AMPL needs to be extended to enable model integration. To accomplish this, AMPL comments will no longer be just comments, but will serve as a source of information for the inference engine. The information provided will be divided into three categories: first, information that links parameters, variables, and constraint sets to the corresponding elements in the

classification; second, information on the interpretation of parameters, variables, and constraints sets; and, finally, information on formulations for related problems. All this information will be expressed in our *AMPL extension language*, defined in Backus-Naur form in Figure 3.

```

<CLASS> ::= AMPL parameter, variable, or constraint;
          # (CLASS: <classification element>)
<ATTRIB> ::= AMPL parameter;
          # (ATTRIB: <attribute>)
<attribute> ::= travel time | demand | time window | capacity | arrival time | waiting time |
              departure time
<DEFINES> ::= AMPL variable;
          # (DEFINES: <decision>)
<decision> ::= arc assignment | node assignment | flow | resource utilization
<CONDUCEES> ::= AMPL constraint;
          # (CONDUCEES: <characteristic> {and <characteristic>}*)
<characteristic> ::= arc assignment | node assignment | position assignment | flow conservation |
                   no subtours | (<classification element>)
<MOD> ::= AMPL parameter, variable, or constraint;
          # (MOD: (<classification element> REPLACES <classification element>)
          {[and | or] (<classification element> REPLACES <classification element>)}* ->
          (<empty symbol>|<CLASS>|<ATTRIB>|<DEFINES>|<CONDUCEES>
          REPLACES <empty symbol>|<CLASS>|<ATTRIB>|<DEFINES>|<CONDUCEES>)
          {and (<empty symbol>|<CLASS>|<ATTRIB>|<DEFINES>|<CONDUCEES>
          REPLACES <empty symbol>|<CLASS>|<ATTRIB>|<DEFINES>|<CONDUCEES>)}*)

```

Figure 3. Syntax of the AMPL extension language.

More specifically, the system views a formulation as composed of two parts: the parameter and variable definitions and the constraint definitions. The AMPL extension language enables us to provide additional information. In case of a parameter definition, we add information to relate it to the attribute it is representing, in case of a variable definition, we add information on the type of decision that is being modeled, and in case of a constraint definition, we add information on the problem characteristic that is being modeled. The objective function is considered to be a constraint.

The AMPL extension language also allows a concise description of similar formulations for related types. Similar formulations are stored as one base formulation and a series of modifications. This results in great savings in space over retaining all of them separately.

Figure 4 shows an extended AMPL formulation for the standard vehicle routing problem, $1 | = m, cap | \sum T_i$. A closer examination reveals that the extended AMPL formulation in fact contains four AMPL formulations instead of just one. The base formulation for the vehicle routing problem ($1 | m, cap | \sum T_i$) and modifications to obtain the vehicle routing problem with a fixed number of vehicles ($1, = m, cap | \sum T_i$), the vehicle routing problem with time windows ($1, tw_j | m, cap | \sum T_i$), and the vehicle routing problem with time windows and a fixed number of vehicles ($1, tw_j | = m, cap | \sum T_i$). The sentence from the AMPL extension language

```

# (MOD: (= m REPLACES m) ->
  (param m >= 0) REPLACES (var m >= 0))

```

in the variables section describes how the base formulation should be modified in case m in the classification is replaced by $= m$. The sentence from the AMPL extension language

```

# (MOD: (tw_j REPLACES <empty symbol>) ->
  (param e {customers} >= 0; # (ATTRIB: earliest service time) REPLACES () and
  param l {customers} >= 0; # (ATTRIB: latest service time) REPLACES ()))

```

in the parameters section partly describes how the base formulation should be modified in case <empty symbol> in the classification is replaced by tw_j .

3.2. Algorithms

The system views an algorithm as a sequence of operations performed on a set of objects. There are two types of objects: first, objects associated with the classification language, such as an arc, a set of arcs, an address, a set of addresses, a cluster, a set of clusters, a route, a set of routes, a vehicle, and a set of vehicles. Second, there are objects associated with mathematical programming formulations, such as a parameter, a set of parameters, a variable, a set of variables, a constraint, a set of constraints, and a linear programming formulation. Each of these objects may have one or more attributes associated with it, such as a travel time t_{ij} for an arc (i, j) , a demand q_j and a time window $[e_j, l_j]$ for an address j , and a capacity Q_i for a vehicle i . Operations on these objects include finding the best element in a set, deleting an address from a route, adding an address to a route, evaluating an exchange of addresses in a route, and solving a linear programming formulation.

The objects can be used to describe the input and output of an algorithm. The overall goal of a vehicle routing algorithm is to produce a set of routes (the output) on the basis of a set of addresses and vehicles with associated attributes and restrictions as implied by the problem type (the input). It can be broken into subgoals. The ‘cluster first-route second’ algorithms, for instance, have two subgoals: to produce a set of clusters using the initial sets of addresses and vehicles, and to produce a set of routes using the set of clusters. Therefore, an algorithm, consisting of a sequence of smaller building blocks, is valid as long as the input of a building block is the output of the previous one.

An important notion in the description of algorithms is that of a *technique*. A technique is a basic methodology that can be used in the construction of algorithms, such as Lagrangean relaxation and branch and bound.

The model and algorithm management system for vehicle routing and scheduling problems will use what we will call *templates* to represent techniques and algorithms. A template is a data structure that provides a description of a technique or an algorithm in terms of their constituent components and the interrelations between these. A template consists of a set of *slots*, which store the components and relations. There are *variable* slots and *permanent* slots. The fundamental operation performed on a template is *instantiation*, i.e., the creation a specific instance of a template by filling the variable slots. As a consequence we can distinguish two types of templates: generic and instantiated. A generic template represents a class of techniques or algorithms by describing the characteristics of the prototypical member of the class, i.e., by providing a skeleton for describing any possible instance in the class. An instantiated template represents a specific technique or algorithm by instantiation of a generic template. The model and algorithm management system will have one generic algorithm template and several generic technique templates.

An algorithm or technique template has to provide three types of information. First, there should be general information: the problem that is being solved, a brief procedural description of the method, and results on its performance. Second, there should be information that is to be used by the inference mechanisms: if the algorithm is based on a formulation, there should be a reference to it, it should be stated what parts of the method deal with which constraints, and also whether there are known modifications to make it applicable to other problems. Finally, there should be information for the user: a natural language description and references to relevant papers in the literature.

More specifically, a generic algorithm template will have no permanent slots and the following variable slots: **classification**, **definition**, **performance**, **formulation**, **description**, **literature**, and one or more algorithm specific slots. Although the names of the slots are chosen to be self-explanatory, we will discuss them briefly.

```
### SETS ###
set addresses;
set customers within addresses;
set arcs within addresses cross addresses;
### PARAMETERS ###
param t {arcs} >= 0;
    # (ATTRIB: travel time)
param q {customers} >= 0;
    # (ATTRIB: demand)
param Q > 0;
    # (CLASS: cap)
    # (ATTRIB: capacity)
    # (MOD: ( $tw_j$  REPLACES <empty symbol>) ->
        (param e {customers} >= 0; # (ATTRIB: earliest service time) REPLACES ( ) and
        param l {customers} >= 0; # (ATTRIB: latest service time) REPLACES ( )))
### VARIABLES ###
var m >= 0;
    # (CLASS: m)
    # (MOD: (=  $m$  REPLACES  $m$ ) -> (param m >= 0) REPLACES (var m >= 0))
var x {arcs} ∈ {0,1};
    # (DEFINES: arc assignment)
var 0 <= D {addresses} <= Q;
    # (DEFINES: resource utilization ( $cap$ ))
    # (MOD: ( $tw_j$  REPLACES <empty symbol>) ->
        (var T {addresses} >= 0; # (DEFINES: resource utilization ( $tw_j$ )) REPLACES ( )))
### OBJECTIVE ###
minimize total-travel-time:
    sum {(i,j) in arcs} t[i,j] × x[i,j];
    # (CLASS: sum  $T_j$ )
### CONSTRAINTS ###
subject to in-assign {i in addresses}:
    if {i in customers}
        then sum {(j,i) in arcs} x[j,i] = 1
        else sum {(j,i) in arcs} x[j,i] - m = 0;
    # (CONDUCTES: arc assignment)
subject to out-assign {i in addresses}:
    if {i in customers}
        then sum {(i,j) in arcs} x[i,j] = 1
        else sum {(i,j) in arcs} x[i,j] - m = 0;
    # (CONDUCTES: arc assignment)
subject to capacity-constraints {(i,j) in arcs}:
    D[i] + q[i] - D[j] <= (1 - x[i,j]) × M;
    # (CONDUCTES: ( $cap$ ) and no subtours)
    # (MOD: ( $tw_j$  REPLACES <empty symbol>) ->
        (time-constraints REPLACES ( ) and
        address-time-window REPLACES ( )))
### REPLACEMENT CONSTRAINTS ###
subject to time-constraints {(i,j) in arcs}:
    T[i] + t[i,j] - T[j] <= (1 - x[i,j]) × M;
    # (CONDUCTES: ( $tw_j$ ) and no subtours)
subject to address-time-window {j in addresses}:
    e[j] <= T[j] <= l[j];
    # (CONDUCTES: ( $tw_j$ ))
```

Figure 4. Extended AMPL formulation for the VRP.

- (1) The **classification** slot gives the classification of the problem the algorithm is solving.
- (2) The **definition** slot contains a listing of the other algorithms and techniques used by the algorithm and a procedural description of their interaction. These algorithms and techniques are defined in the algorithm specific slots.
- (3) The **performance** slot contains information on the efficiency and effectiveness of the algorithm. It will indicate the time and storage requirements, whether it is an optimization or approximation algorithm, and results on its empirical, worst-case and maybe even average-case behavior.
- (4) The **formulation** slot records, if the algorithm is based on a formulation, where this formulation can be found.
- (5) The **description** slot provides a short natural language description of the algorithm.
- (6) The **literature** slot gives a number of relevant references.
- (7) The algorithm specific slots contain instantiated algorithm or technique templates.

A generic technique template will have two permanent slots, **name** and **definition**, and the following variable slots: **input**, **output**, **performance**, **description**, and one or more technique specific slots.

- (1) The **name** slot gives the name of the technique.
- (2) The **input** slot lists the objects the technique expects to be available.
- (3) The **output** slot lists the objects that will be produced.
- (4) The **definition** slot contains a listing of the other algorithms and techniques used by the technique and a procedural description of their interaction. These algorithms and techniques are defined in the technique specific slots.
- (5) The **performance** slot contains information on the efficiency and effectiveness of the technique. It will indicate the time and storage requirements, whether it is an optimization or approximation technique, and results on its empirical, worst-case and maybe even average-case behavior.
- (6) The **description** slot provides a short natural language description of the technique.
- (7) The technique specific slots contain instantiated algorithm or technique templates.

As was the case with formulations, we need more information to be able to perform algorithm integration. A *template extension language*, which is defined in Figure 5, is introduced for that purpose and will associate problem characteristics with parts of the algorithm or technique and provide information on relevant modifications.

The most important information in a technique template is how the technique specific slots interact. This can best be explained by means of examples.

Consider the technique template for *Lagrangean relaxation*. Its **input**, **output**, and **definition** slots are given in Figure 6. The GKB contains several construction rules about the application of Lagrangean relaxation: the constraints of the subproblem and the relaxed constraints form a partition of the original constraint set; a multiplier has to be defined for each relaxed constraint; if a candidate subproblem has the integrality property, then the Lagrangean relaxation is not interesting; the multiplier adjustment method is to be chosen from among subgradient optimization (the default), a dual ascent method tailored for the model, or column generation; it may be possible to obtain a feasible solution from a solution to the subproblem and the values of the multipliers. Such supplementary information is useful when this technique has to be instantiated.

Another example is the technique template for the well-known 2-exchange iterative improvement procedure. Its **input**, **output**, and **definition** slots are given in Figure 7. The complete extended technique template can be found in Appendix B.

```
<CLASS> ::= slot instantiation;
          # (CLASS: <classification element>)
<ATTRIB> ::= slot instantiation;
          # (ATTRIB: <attribute>)
<attribute> ::= travel time | demand | time window | capacity | arrival time | waiting time |
              departure time
<DEFINES> ::= slot instantiation;
          # (DEFINES: <decision>)
<decision> ::= arc assignment | node assignment | flow | resource utilization
<CONDUCEES> ::= slot instantiation;
          # (CONDUCEES: <characteristic> {and <characteristic>}*)
<characteristic> ::= arc assignment | node assignment | position assignment | flow conservation |
                   no subtours | (<classification element>)
<MOD> ::= slot instantiation element;
          # (MOD: (<classification element> REPLACES <classification element>)
            {[and | or] (<classification element> REPLACES <classification element>)}* ->
            (<empty symbol>|<CLASS>|<ATTRIB>|<DEFINES>|<CONDUCEES>
            REPLACES <empty symbol>|<CLASS>|<ATTRIB>|<DEFINES>|<CONDUCEES>)
            {and (<empty symbol>|<CLASS>|<ATTRIB>|<DEFINES>|<CONDUCEES>
            REPLACES <empty symbol>|<CLASS>|<ATTRIB>|<DEFINES>|<CONDUCEES>)}*)
```

Figure 5. Syntax of the template extension language.

4. Manipulation

A substantial part of the knowledge in the GKB is related to the reasoning and manipulation done by the system. We will distinguish several types of general knowledge.

Conceptual knowledge. This refers to all the concepts that are stored as facts in the system and that form the basis of all manipulation done by the system, such as a constraint, a relaxation, and a mixed integer programming formulation.

Operations research knowledge. This refers to the knowledge that may be useful when trying to decide if and how to apply a certain technique. As an example, the following knowledge may be available: the linear relaxation of a mixed integer programming formulation provides a, sometimes bad, lower bound; in fact, any relaxation of a mixed integer programming formulation provides a lower bound. There is knowledge on how to construct various relaxations from a formulation, such as relaxing one or more constraints.

Basic problems. There are descriptions of basic problems, such as the knapsack problem and the linear assignment problem, in terms of constraint sets plus information on the available algorithms for these problems. These basic problems are useful when decomposition techniques are applied to a formulation. For example, if the system considers applying Lagrangean relaxation to a formulation, it will scan the set of basic problems for problems that only have constraints that appear in the formulation as well, since these are suitable candidate subproblems for Lagrangean relaxation.

Modification knowledge. Because a number of formulations (algorithms) are stored only as modifications to other formulations (algorithms), there is knowledge to guide the construction of formulations (algorithms) not explicitly available.

Search knowledge. A considerable part of the knowledge will be devoted to issues regarding the search of the various data bases. Efficient search strategies are of crucial importance for the system since the amount of data in the system is enormous.

```
INPUT
  OriginalProblem
  RelaxedProblem

OUTPUT
  LowerBoundValue
  UpperBoundValue
  UpperBoundSolution

DEFINITION
  UPPERBOUND( )
  INITMULTIPLIERS( )
  ADJUSTMULTIPLIERS( )
  SOLVE( )
  FEASIBLE( )
  UPDATE( )

UpperBoundValue & UpperBoundSolution ← UPPERBOUND(OriginalProblem)
INITMULTIPLIERS(Multipliers)
iter ← 0
repeat
  iter ← iter + 1
  LowerBoundValue & LowerBoundSolution ← SOLVE(RelaxedProblem)
  if FEASIBLE(LowerBoundSolution)
    then
      if LowerBoundValue = UpperBoundValue
        then
          stop
        else
          UPDATE(LowerBoundSolution, UpperBoundValue)
      ADJUSTMULTIPLIERS(LowerBoundSolution, Multipliers)
until 'no improvement' or iter > itermax
```

Figure 6. The input, output, and definition slots for the Lagrangean relaxation template.

The short descriptions above are only meant to give an impression of the types of knowledge available. In the following sections, more elaborate examples will be given, especially for knowledge regarding the application of techniques.

4.1. Related problems

The system uses a semantic network to relate problem types. Each node of the network represents a problem type, and each weighted arc of the network represents the fact that, with a certain confidence factor, one of the algorithms (formulations) for the problem type associated with the node at the tail of the arc can be used to construct an algorithm (a formulation) for the problem type associated with the node at the head of the arc. The confidence factors, which are all in the half open interval $(0, 1]$, represent a substantial part of the expertise that is built into the system.

A path in the above described network tells us that, with a certain confidence factor, starting from an algorithm (a formulation) for the problem type associated with the node at the beginning of the path, it is possible to construct an algorithm (a formulation) for the problem type associated with the node at the end of the path. The confidence factor for the path is obtained by multiplying the

```

INPUT
  tour

OUTPUT
  tour

DEFINITION
  FEASIBLE()
  EVALUATE()

  CurrentTour ← InputTour
  CurrentCost ← EVALUATE (CurrentTour)
  while ( {(i, i + 1), (j, j + 1)} → {(i, j), (i + 1, j + 1)} not examined in CurrentTour ) do
  begin
    ExchangeTour ← perform {(i, i + 1), (j, j + 1)} → {(i, j), (i + 1, j + 1)}
    if FEASIBLE (ExchangeTour) then
    begin
      ExchangeCost ← EVALUATE (ExchangeTour)
      if (ExchangeCost < CurrentCost) then
      begin
        CurrentTour ← ExchangeTour
        CurrentCost ← ExchangeCost
      end
    end
  end
end
OutputTour ← CurrentTour

```

Figure 7. The input, output, and definition slots for the 2-opt template.

confidence factors of all of the arcs on the path. It will also be in the open interval $(0, 1]$, and many arcs on a path imply a high chance of a low confidence factor.

The key concept in the definition of the initial network is that of a *single subfield change*. Recall that the representation defined in Appendix A uses 26 subfields to describe a problem type, and each subfield has a limited number of values. In a single subfield change, the value of a certain subfield k is changed from i to j and the values for the other subfields remain unchanged. The confidence factor for such a single subfield change will be denoted by w_{ij}^k . Based on our knowledge of, and experience in, the area of vehicle routing and scheduling area, we have defined a confidence factor for every single possible subfield change. Observe that in general $w_{ij}^k \neq w_{ji}^k$. The initial network contains arcs for all single subfield changes that have a positive confidence factor.

The initial network is augmented with arcs corresponding to *multiple subfield changes*, also called *shortcuts*. There are two types of shortcuts. The first type corresponds to a model transformation and has a confidence factor 1, for example a pickup and delivery problem with full truck loads can be modeled as a multisa salesman problem. The second type of shortcut corresponds to a known algorithm transformation and has a confidence factor in the interval $(0, 1)$, for example the modification of 2-exchange iterative improvement methods to incorporate time windows, precedence constraints, and mixed collections and deliveries.

Some of the nodes in the network are marked *known* to indicate that the problem type associated with this node is well known and well investigated and that formulations and algorithms for this problem type exist.

Given an arbitrary problem type and the above defined network, it is possible to construct a list of related known problem types with associated confidence factors by computing all shortest paths originating from the given problem type.

In the shortest path computations, we enforce certain restrictions on the allowable paths. Only direct subfield changes are allowed in a path from one problem type to another, unless the path contains a shortcut. For example, when the address scheduling constraints subfield is changed from ‘none’ to ‘single time windows’, it is not allowed to go through ‘fixed schedule’.

In the final system, at least two of the above described networks will exist, because the confidence factors may differ considerably, depending on whether we are constructing an optimization or an approximation algorithm.

The following examples illustrate the concept of single subfield changes and the corresponding interpretation. For the notation used, we refer to Appendix A.

$1|1||T \rightarrow 1, tw_j|1||T$. The former problem has one depot, one vehicle, and the minimization of route duration as its objective; this is the traveling salesman problem. In the latter problem the cities get time windows, which make the problem more difficult.

$1, TASK|m, cap||sumT_i \rightarrow 1, TASK|m, cap|full||sumT_i$. The former problem has a single depot and multiple capacitated vehicles that have to perform pickup and delivery tasks. In the latter problem only full truck loads are allowed, which simplifies the problem to a multisalesman problem.

$1|m, cap||sumT_i \rightarrow 1, \sim|m, cap||sumT_i$. The former problem is the standard single-depot multi-vehicle problem with deterministic demands. In the latter problem the demands become stochastic, which completely changes the problem structure.

4.2. Model integration

There are two ways to come up with a formulation for a problem type P not explicitly stored in the PKB.

The first way is to see whether P is implicitly stored in the PKB. This is done by searching the FKB for appropriate modification statements from the AMPL extension language. If such statements are found, they provide all the information needed to obtain a valid model for P .

The second way is to carefully merge two formulations associated with problems similar in structure to P . In this case, we really construct a new formulation. Let P' and P'' be two problems similar in structure to P . If the union of their characteristics completely covers the characteristics of P and if their formulations have compatible variable and constraint definitions, we can attempt to merge the two formulations. First, the new variable set is defined as the union of the two variable sets. Second, complementary constraint sets are extracted from the formulations, expressed in the new variables, and merged to form the new formulation. Finally, the new formulation is validated, i.e., it is checked whether the formulation is syntactically correct and deals with all the constraints of problem P . Knowledge about the syntactic structure of a mathematical programming formulation could be in the form of rules concerning the relationships between coefficient, variable and right-hand side indices and their use in summations.

As an example, consider a PKB that contains, among others, the known problems $1, tw_j|1||T$ and $1| = m, cap||sumT_i$, and an FKB that contains the associated formulations. For presentational convenience, we use the standard mathematical notation, but we have added the relevant statements from the AMPL extension language. The formulation of $1, tw_j|1||T$ is

minimize $\sum_{ij} c_{ij} x_{ij}$

subject to

$$\begin{aligned} \sum_j x_{ij} = \sum_j x_{ji} = 1 & \quad \text{for } i \in N, & \quad \text{[#CONDUCEs: arc assignment]} \\ D_i + t_{ij} - D_j \leq C(1 - x_{ij}) & \quad \text{for } (i, j) \in A, & \quad \text{[#CONDUCEs: } (tw_j) \text{ and no subtours]} \\ e_i \leq D_i \leq l_i & \quad \text{for } i \in N, & \quad \text{[#CONDUCEs: } (tw_j)] \\ x_{ij} \in \{0, 1\} & \quad \text{for } (i, j) \in A. & \quad \text{[#DEFINES: arc assignment]} \end{aligned}$$

The formulation of $1 \mid m, cap \mid \sum T_i$ is

minimize $\sum_{ij} c_{ij} \sum_k x_{ij}^k$

subject to

$$\begin{aligned} \sum_k y_i^k &= \begin{cases} |M| & \text{for } i = 0, \\ 1 & \text{for } i \in N, \end{cases} & \quad \text{[#CONDUCEs: node assignment]} \\ \sum_i q_i y_i^k \leq Q & \quad \text{for } k \in M, & \quad \text{[#CONDUCEs: node assignment and } (cap)] \\ \sum_j x_{ij}^k = \sum_j x_{ji}^k = y_i^k & \quad \text{for } i \in N, k \in M, & \quad \text{[#CONDUCEs: arc assignment]} \\ \sum_{i \in S, j \notin S, k} x_{ij}^k \geq 1 & \quad \text{for } \emptyset \neq S \subset N, S \neq N, & \quad \text{[#CONDUCEs: no subtours]} \\ y_i^k \in \{0, 1\} & \quad \text{for } i \in N, k \in M, & \quad \text{[#DEFINES: node assignment]} \\ x_{ij}^k \in \{0, 1\} & \quad \text{for } (i, j) \in A, k \in M. & \quad \text{[#DEFINES: arc assignment]} \end{aligned}$$

Now suppose we are interested in a formulation for the problem $1, tw_j \mid m, cap \mid \sum T_i$, which is not in the PKB. The system recognizes that the characteristics of this problem are completely covered by the union of the characteristics of the two problems mentioned above and that their associated formulations have compatible variable and constraint definitions. In addition, the system detects that if the two formulations are merged some redundancy arises: there will be two constraint sets to prevent subtours. It will therefore delete the one that has become obsolete to obtain

minimize $\sum_{ij} c_{ij} \sum_k x_{ij}^k$

subject to

$$\begin{aligned} \sum_k y_i^k &= \begin{cases} |M| & \text{for } i = 0, \\ 1 & \text{for } i \in N, \end{cases} & \quad \text{[#CONDUCEs: node assignment]} \\ \sum_i q_i y_i^k \leq Q & \quad \text{for } k \in M, & \quad \text{[#CONDUCEs: node assignment and } (cap)] \\ \sum_j x_{ij}^k = \sum_j x_{ji}^k = y_i^k & \quad \text{for } i \in N, k \in M, & \quad \text{[#CONDUCEs: arc assignment]} \\ D_i + t_{ij} - D_j \leq C(1 - x_{ij}^k) & \quad \text{for } (i, j) \in A, k \in M, & \quad \text{[#CONDUCEs: } (tw_j) \text{ and no subtours]} \\ e_i \leq D_i \leq l_i & \quad \text{for } i \in N, & \quad \text{[#CONDUCEs: } (tw_j)] \\ y_i^k \in \{0, 1\} & \quad \text{for } i \in N, k \in M, & \quad \text{[#DEFINES: node assignment]} \\ x_{ij}^k \in \{0, 1\} & \quad \text{for } (i, j) \in A. & \quad \text{[#DEFINES: arc assignment]} \end{aligned}$$

4.3. Algorithm integration

An algorithm for a problem type P not explicitly stored in the PKB can be constructed in several ways.

First, the AKB should be scanned for appropriate modification statements from the template extension language, to see if P is implicitly stored in the PKB. If so, these provide information indicating how the algorithm should be modified to obtain a valid algorithm for P .

Second, two algorithms associated with problems that are similar in structure to P might be merged. This closely resembles the merging of two formulations as described above. However, there is a distinction between merging algorithms based on a formulation and merging algorithms not based on a formulation. In the first case, there exists a formulation F that is obtained by merging two formulations F' and F'' associated with problem types P' and P'' similar in structure to P . Let A' and A'' be two algorithms associated with the formulations F' and F'' respectively. The system tries to adapt one of them using parts of the other. Suppose the system tries to adapt A' . To start, the system identifies the characteristics or constraints of P that are not dealt with by A' . Then, it establishes how these characteristics or constraints are dealt with by A'' and, if possible, modifies A' according to the techniques used in A'' . Knowledge about the structure of the associated formulations might guide this process. In the second case, the system performs the same steps without the additional knowledge from the formulations.

Consider the example given in the previous subsection, i.e., a PKB that contains, among others, the known problems $1, tw_j|1||T$ (TSPTW) and $1| = m, cap|sumT_i$ (VRP), and a user interested in the problem $1, tw_j| = m, cap|sumT_i$ (VRPTW). Furthermore, assume that one of the algorithms associated with the VRP is of the type 'cluster first-route second'. The GKB contains the fact that in 'cluster first-route second' algorithms the second phase consists of the solution of a number of TSPs. Based on this knowledge one of the system's suggested algorithms for the VRPTW will be an adapted 'cluster first-route second' method in which the original algorithm for the solution of the TSPs in the second phase is replaced by one of the algorithms associated with the TSPTW.

Finally, techniques might be applied to construct an algorithm. Suppose the system obtains a formulation for some problem by merging formulations for problems that have a similar structure. Instead of trying to merge the associated algorithms, it might try to apply one or more techniques to this formulation. For instance, it could try to apply Lagrangean relaxation. Consider the formulation for the problem $1|||T$, the symmetric TSP:

$$\text{minimize } \sum_{ij} c_{ij}x_{ij}$$

subject to

$$\sum_j x_{ij} = 1 \quad \text{for } i \in N, \quad (1)$$

$$\sum_j x_{ji} = 1 \quad \text{for } i \in N, \quad (2)$$

$$\sum_{i \in S, j \notin S} x_{ij} \geq 1 \quad \text{for } \emptyset \neq S \subset N, S \neq N, \quad (3)$$

$$x_{ij} \in \{0, 1\} \quad \text{for } (i, j) \in E. \quad (4)$$

The system could successively try to move one or more of the constraint sets (1)-(3) into the objective function and evaluate the resulting formulations. Note that in order to be able to perform this evaluation, the system has to be able to recognize the resulting subproblems as being the minimum spanning 1-tree problem, in case (1) or (2) is moved into the objective function, and the linear assignment problem, in case (3) is moved into the objective function. Formulations should therefore be stored such that these structural properties are included. Lee [22] addresses the question of how to manipulate and store formulations in such a way that structural information is included.

5. Towards implementation

An implementation was begun using LPA Prolog for Windows version 2.6. We chose Prolog [8] since it is well suited to representing symbolic information and deriving inferences. Lisp would have been another good choice.

We implemented the semantic network described in Section 4.1 to relate problem types. We did not implement model and algorithm integration as discussed in Sections 4.2 and 4.3.

Recall that the nodes of the semantic network are the problem types from our classification scheme, either known or unknown. There are two types of (directed) arcs: field change arcs and shortcut arcs. The weights of the arcs are confidence factors. The weight of a least-weight path from one problem to another is a measure of similarity.

Our Prolog implementation only stores the nodes that represent known problems; other nodes are generated when needed during the construction of paths. Relying on standard Prolog syntax, we store a node in the form of the `problem/2` predicate. It contains two arguments: the name of the problem, and a list of attribute value pairs. For example, the TSP is represented as

```
problem('TSP', [['#depots', 1], [beta2, 1], [objective, 'Ti']]).      (5)
```

This means that the TSP has one depot and one vehicle (`beta2`), and that the objective is to minimize route duration.

Similarly, the confidence factors w_{ij}^k are represented by the `weight/4` predicate, which lists field, original value, new value, and weight. For example, the predicate

```
weight('#depots', 1, 1, 0.6)
```

indicates that when the number of depots changes from one (1) to an arbitrary number (1), the resulting problem has similarity measure 0.6 when compared to the original.

Shortcuts, which identify similar problems that differ by several fields, are implemented by the `shortcut/2` predicate. The first argument is a list of triples, each triple consisting of a field name and two of its allowed values. The second argument is the similarity weight between problems that have fields with values equal to the middle value of each triple and problems with field values equal to the last value of each triple. Consider the following example:

```
shortcut([[vsched, o, twi], [rdur, duri, o]], 1).
```

Here, a problem without time windows but with different upper bounds on route duration is equivalent (weight 1) to a problem with time windows but without route duration constraints.

Given a set of known problems, a set of confidence factors and a set of shortcuts, users enter an unknown problem in the style shown in (5). The system then produces a sorted list of similar problems along with the path taken from the unknown to the known problem. For example, the request

```
?- mostsimilar(['#depots', 1], [beta2, m], [rdur, duri],
               [capacity, cap], [zeta1, 'DV'], [objective, 'Ti'])
```

(where the unknown problem has a single depot, multiple vehicles, bounds on route duration, vehicles with identical capacities and depot-vehicle restrictions, and the objective is to minimize route duration) yields the following output:

```
Problems in order of decreasing similarity
Problem:  test
Weight:   1
1|m, cap, duri|DV|Ti      Weight:  1
```

Problem: VRPvtw

Weight: 1

1 m, cap, twi Ti	Weight: 1
1 m, cap, twi DV Ti	Weight: 1
1 m, cap, duri DV Ti	Weight: 1

Problem: VRPdur

Weight: 1

1 m, cap, duri Ti	Weight: 1
1 m, cap, twi Ti	Weight: 1
1 m, cap, twi DV Ti	Weight: 1
1 m, cap, duri DV Ti	Weight: 1

Problem: VRP

Weight: 1

1 m, cap Ti	Weight: 1
1 m, cap DV Ti	Weight: 1
1 m, cap, twi DV Ti	Weight: 1
1 m, cap, duri DV Ti	Weight: 1

Problem: TSP

Weight: 0.900000

1 1 Ti	Weight: 1
1 1 DV Ti	Weight: 1
1 1, twi DV Ti	Weight: 1
1 1, duri DV Ti	Weight: 1
1 1, cap, duri DV Ti	Weight: 0.900000
1 m, cap, duri DV Ti	Weight: 1

Problem: VRPtw

Weight: 0.400000

1, twj m, cap Ti	Weight: 1
1, twj m, cap DV Ti	Weight: 1
1, twj m, cap, twi DV Ti	Weight: 1
1, twj m, cap, duri DV Ti	Weight: 0.400000
1 m, cap, duri DV Ti	Weight: 1

Problem: TSPtw

Weight: 0.360000

1, twj 1 Ti	Weight: 1
1, twj 1 DV Ti	Weight: 1
1, twj 1, twi DV Ti	Weight: 1
1, twj 1, duri DV Ti	Weight: 1
1, twj 1, cap, duri DV Ti	Weight: 0.900000
1, twj m, cap, duri DV Ti	Weight: 0.400000
1 m, cap, duri DV Ti	Weight: 1

```

Problem:  SVRPtw
Weight:   0.0
1, ~, twj | m, cap | | Ti      Weight:  1
1, ~, twj | m, cap | DV | Ti   Weight:  1
1, ~, twj | m, cap, twi | DV | Ti Weight:  1
1, ~, twj | m, cap, duri | DV | Ti Weight:  0.400000
1, ~ | m, cap, duri | DV | Ti  Weight:  0
1 | m, cap, duri | DV | Ti     Weight:  1

```

```

Problem:  SVRP
Weight:   0
1, ~ | m, cap | | Ti          Weight:  1
1, ~ | m, cap | DV | Ti       Weight:  1
1, ~ | m, cap, twi | DV | Ti  Weight:  1
1, ~ | m, cap, duri | DV | Ti Weight:  0
1 | m, cap, duri | DV | Ti    Weight:  1

```

The algorithm used by the system calculates the product of the weights along a shortest path (in terms of the number of edges traversed) from a node representing an unknown problem to a node representing a known problem. It performs a depth-first search from each known problem K to the unknown problem U . Each iteration considers a particular problem and estimates the similarity of adjacent problems. The algorithm is given in more detail below:

- Step 1. Set $T \leftarrow \{K\}$ and $P \leftarrow \emptyset$. Set $e_n \leftarrow 1$ for all nodes n .
- Step 2. If $T = \emptyset$, then stop: there is no viable path from K to U .
- Step 3. Choose $X \in T$ such that e_X is maximum. If $X = U$, stop and output the shortest path from K to U .
- Step 4. Set $T \leftarrow T \setminus \{X\}$ and $P \leftarrow P \cup \{X\}$.
- Step 5. For all $Y \notin P$ that differ from X by a single field change or shortcut of weight w , update $e_Y \leftarrow we_Y$. If $Y \notin T$, then $T \leftarrow T \cup Y$.
- Step 6. Go to Step 2.

The current implementation only considers optimization algorithms. Adding the capability to provide a different network for approximation algorithms will be trivial. Less trivial will be the calibration of the weights for single field changes and shortcuts. One would like the ratings to reflect the actual similarity. This will require much iteration involving experts in vehicle routing to test a variety of known and unknown problems.

Acknowledgements

The authors gratefully acknowledge the support of the CWI in Amsterdam, where they were employed or visiting when performing the research reported here. This work was also supported by the National Science Foundation, through the Center for Research on Parallel Computation, Rice University, under Cooperative Agreement No. CCR-9120008.

References

- [1] H.K. Bhargava, A.S. King, D.S. McQuay (1995). DecisionNet: modeling and decision support over the world wide web. *Proceedings of the Third ISDSS Conference, Hong Kong, June 1995*. URL for DecisionNet: <http://dnet.sm.nps.navy.mil/>
- [2] J. Bisschop, R. Entriken (1993). *AIMMS The Modeling System*, Paragon Decision Technology, Haarlem.
- [3] J. Bisschop, A. Meeraus (1982). On the development of a General Algebraic Modeling System in a strategic planning environment. *Mathematical Programming Study* 20, 1-29.
- [4] R.W. Blanning (1989). Model management systems. R.H. Sprague, Jr., H.J. Watson (eds.). *Decision Support Systems: Putting Theory into Practice*, Prentice Hall, 156-159.
- [5] A. Brooke, D. Kendrick, A. Meeraus (1988). *GAMS: A User's Guide*, The Scientific Press, Redwood City.
- [6] N. Christofides (1985). Vehicle routing. E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys (eds.). *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, Chichester, 431-448.
- [7] G. Clarke, J.W. Wright (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research* 12, 568-581.
- [8] W.F. Clocksin, C.S. Mellish (1987). *Programming in Prolog*, Springer, Berlin.
- [9] G.B. Dantzig, J.H. Ramser (1959). The truck dispatching problem. *Management Science* 6, 80-91.
- [10] Dash Associates (1997). *XPRESS-MP User Manual*, Blisworth.
- [11] M. Desrochers, J.K. Lenstra, M.W.P. Savelsbergh (1990). A classification scheme for vehicle routing and scheduling problems. *European Journal of Operational Research* 46, 322-332.
- [12] J. Desrosiers, Y. Dumas, M.M. Solomon, F. Soumis (1995). Time constrained routing and scheduling. M.O. Ball, T.L. Magnanti, C.L. Monma, G.L. Nemhauser (eds.). *Handbooks in Operations Research and Management Science, Volume 8: Network Routing*, North-Holland, Amsterdam, 35-140.
- [13] D.R. Dolk, B.R. Konsynski (1984). Knowledge representation for model management systems. *IEEE Transactions on Software Engineering SE-10*, 619-628.
- [14] M.L. Fisher (1995). Vehicle routing. M.O. Ball, T.L. Magnanti, C.L. Monma, G.L. Nemhauser (eds.). *Handbooks in Operations Research and Management Science, Volume 8: Network Routing*, North-Holland, Amsterdam, 1-34.
- [15] M.L. Fisher, R. Jaikumar (1981). A generalized assignment heuristic for vehicle routing. *Networks* 11, 109-124.
- [16] R. Fourer, D.M. Gay, B.W. Kernighan (1993). *AMPL: A Modeling Language for Mathematical Programming*, The Scientific Press, San Francisco.
- [17] A.M. Geoffrion (1987). An introduction to structured modeling. *Management Science* 33, 547-587.
- [18] B.L. Golden, A.A. Assad (eds.) (1988). *Vehicle Routing: Methods and Studies*, North-Holland, Amsterdam.
- [19] I.P. Goldstein, B. Roberts (1979). Using frames in scheduling. P.H. Winston, R.H. Brown (eds.). *Artificial Intelligence: An MIT Perspective*, MIT Press, Cambridge.
- [20] B.J. Lageweg, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan (1981). *Computer-Aided Complexity Classification of Deterministic Scheduling Problems*, Report BW138, Mathematisch Centrum, Amsterdam.
- [21] B.J. Lageweg, J.K. Lenstra, E.L. Lawler, A.H.G. Rinnooy Kan (1982). Computer-aided complexity classification of combinatorial problems. *Communications of the ACM* 25, 817-822.

- [22] J.S. Lee (1986). *A Model Base for Identifying Mathematical Programming Structures*, Report 86-06-05, The Wharton School, University of Pennsylvania, Philadelphia.
- [23] J.S. Lee, M. Guignard, C.V. Jones (1991). MAPNOS: Mathematical programming formulation normalization system. *Expert Systems with Applications 1*, 367-381.
- [24] J.S. Lee, M. Guignard, C.V. Jones (1992). Variations in model formulations. *Annals of Operations Research 38*, 325-335.
- [25] T.-P. Liang (1993). Analogical reasoning and case-based learning in model management systems. *Decision Support Systems 10*, 137-160.
- [26] T.-P. Liang, C.V. Jones (1988). Meta-design considerations in developing model management systems. *Decision Sciences 19*, 72-92.
- [27] Maximal Software (1991). *MPL Modeling System*, Reykjavik.
- [28] M. Minsky (1975). A framework for representing knowledge. P.H. Winston (ed.). *The Psychology of Computer Vision*, McGraw-Hill, New York.
- [29] J. Mylopoulos (1980). An overview of knowledge representation. M.L. Brodie, S.N. Ziles (eds.). *Proceeding of the ACM Workshop on Data Abstraction, Databases, Conceptual Modeling*, 5-12.
- [30] J. Mylopoulos, H.J. Levesque (1984). An overview of knowledge representation. M.L. Brodie, J. Mylopoulos, J.W. Schmidt (eds.). *On Conceptual Modeling*, Springer, New York, 3-17.
- [31] T. Rubach (1985). *Programmbibliothek und Expertensystem zur Maschinenbelegungsplanung: Algorithmische Beschreibung - Dokumentation*, Report WIOR-241, Universitaet Karlsruhe.

Appendix A. Classification scheme for vehicle routing and scheduling problems

◦ indicates the empty symbol.

<classification> ::=

<addresses>
<vehicles>
<problem characteristics>
<objectives>

<addresses> ::=

<number of depots>
<type of demand>
<address scheduling constraints>
<address selection constraints>

<number of depots> ::= $1 \vee l$

1 [one depot]
 l [specified as part of the problem instance]

<type of demand> ::= $\langle \alpha_1 \rangle \langle \alpha_2 \rangle \langle \alpha_3 \rangle$

$\langle \alpha_1 \rangle$::= $\circ \vee \text{EDGE} \vee \text{MIXED} \vee \text{TASK}$

◦ [node routing]
EDGE [edge routing]
MIXED [mixed routing (nodes and edges)]
TASK [task routing]

$\langle \alpha_2 \rangle$::= $\circ \vee \pm$

◦ [either all deliveries or all collections]
 \pm [mixed deliveries and collections]

$\langle \alpha_3 \rangle$::= $\circ \vee \sim$

◦ [deterministic demand]
 \sim [stochastic demand]

<address scheduling constraints> ::= $\circ \vee fs_j \vee tw_j \vee mw_j$

◦ [no scheduling constraints]
 fs_j [fixed schedule]
 tw_j [single time windows]
 mw_j [multiple time windows]

$\langle \text{address selection constraints} \rangle ::= \circ \vee \textit{subset} \vee \textit{choice} \vee \textit{period}$

- \circ [single plan; all addresses must be visited]
- subset* [single plan; a given subset of addresses must be visited]
- choice* [single plan; at least one address in each subset of a given partition must be visited]
- period* [a number of plans over a given time period is to be made]

$\langle \text{vehicles} \rangle ::=$

- $\langle \text{number of vehicles} \rangle$
- $\langle \text{capacity constraints} \rangle$
- $\langle \text{commodity constraints} \rangle$
- $\langle \text{vehicle scheduling constraints} \rangle$
- $\langle \text{route duration constraints} \rangle$

$\langle \text{number of vehicles} \rangle ::= \langle \beta_1 \rangle \langle \beta_2 \rangle$

$\langle \beta_1 \rangle ::= \circ \vee =$

- \circ [at most β_2 vehicles can be used]
- $=$ [all β_2 vehicles must be used]

$\langle \beta_2 \rangle ::= c \vee m$

- $c (c \in \mathbf{N})$ [c vehicles]
- m [specified as part of the problem instance]

$\langle \text{capacity constraints} \rangle ::= \circ \vee \textit{cap} \vee \textit{cap}_i$

- \circ [no capacity constraints]
- cap* [vehicles with identical capacities]
- cap_i* [vehicles with different capacities]

$\langle \text{commodity constraints} \rangle ::= \circ \vee \textit{sep} \vee \textit{ded}$

- \circ [no compartments]
- sep* [vehicles have interchangeable compartments]
- ded* [vehicles have dedicated compartments]

$\langle \text{vehicle scheduling constraints} \rangle ::= \circ \vee \textit{tw} \vee \textit{tw}_i$

- \circ [no scheduling constraints]
- tw* [identical time windows for vehicles]
- tw_i* [different time windows for vehicles]

<route duration constraints> ::= $\circ \vee dur \vee dur_i$

- \circ [no route duration constraints]
- dur* [identical upper bounds on route duration]
- dur_i* [different upper bounds on route duration]

<problem characteristics> ::=

- <type of network>
- <type of strategy>
- <address-address restrictions>
- <address-vehicle restrictions>
- <vehicle-vehicle restrictions>

<type of network> ::= $\langle \gamma_1 \rangle \langle \gamma_2 \rangle$

$\langle \gamma_1 \rangle ::= \circ \vee \Delta$

- \circ [general costs]
- Δ [the costs satisfy the triangle inequality]

$\langle \gamma_2 \rangle ::= \circ \vee dir \vee mix$

- \circ [undirected network]
- dir* [directed network]
- mix* [mixed network]

<type of strategy> ::= $\langle \delta_1 \rangle \langle \delta_2 \rangle \langle \delta_3 \rangle \langle \delta_4 \rangle$

$\langle \delta_1 \rangle ::= \circ \vee / \vee \div$

- \circ [splitting of demand not allowed]
- / [a priori splitting of demand allowed]
- \div [a posteriori splitting of demand allowed]

$\langle \delta_2 \rangle ::= \circ \vee back \vee full$

- \circ [no backhauling or full loads required]
- back* [backhauling, in case of node routing]
- full* [full loads, in case of task routing]

$\langle \delta_3 \rangle ::= \circ \vee \geq 1R/V$

- \circ [at most one route per vehicle]
- $\geq 1R/V$ [more than one route per vehicle allowed]

$\langle \delta_4 \rangle ::= \circ \vee \geq 1D/R$

- \circ [a route starts and finishes at the same depot]
- $\geq 1D/R$ [multi-depot routes allowed]

$\langle \text{address-address restrictions} \rangle ::= \langle \varepsilon_1 \rangle \langle \varepsilon_2 \rangle \langle \varepsilon_3 \rangle$

$\langle \varepsilon_1 \rangle ::= \circ \vee prec$

- [no precedence constraints]
- prec* [precedence constraints]

$\langle \varepsilon_2 \rangle ::= \circ \vee DA$

- [no depot-address restrictions]
- DA* [depot-address restrictions]

$\langle \varepsilon_3 \rangle ::= \circ \vee AA$

- [no address-address restrictions]
- AA* [address-address restrictions]

$\langle \text{address-vehicle restrictions} \rangle ::= \langle \zeta_1 \rangle \langle \zeta_2 \rangle$

$\langle \zeta_1 \rangle ::= \circ \vee DV$

- [no depot-vehicle restrictions]
- DV* [depot-vehicle restrictions]

$\langle \zeta_2 \rangle ::= \circ \vee AV$

- [no address-vehicle restrictions]
- AV* [address-vehicle restrictions]

$\langle \text{vehicle-vehicle restrictions} \rangle ::= \circ \vee VV$

- [no vehicle-vehicle restrictions]
- VV* [synchronization between vehicles needed]

$\langle \text{objectives} \rangle ::= \langle \text{objective} \rangle \vee \langle \text{objective} \rangle \langle \text{objectives} \rangle$

$\langle \text{objective} \rangle ::= \circ \vee \langle \text{operator} \rangle \langle \text{function} \rangle$

$\langle \text{operator} \rangle ::= \text{sum} \vee \text{max}$

- sum* [minimize the sum of the cost function values]
- max* [minimize the maximum cost function value]

$\langle \text{function} \rangle ::= T_i \vee C_i \vee P_i(\langle \text{vehicle constraints} \rangle) \vee$
 $C_j \vee P_j(\langle \text{address constraints} \rangle)$

T_i	[route duration]
C_i	[vehicle costs]
$P_i(\langle \text{vehicle constraints} \rangle)$	[vehicle penalty]
C_j	[address costs]
$P_j(\langle \text{address constraints} \rangle)$	[address penalty]

$\langle \text{vehicle constraints} \rangle ::= \langle \text{vehicle constraint} \rangle \vee$
 $\langle \text{vehicle constraint} \rangle \langle \text{vehicle constraints} \rangle$

$\langle \text{vehicle constraint} \rangle ::= cap \vee cap_i \vee tw \vee tw_i \vee dur \vee dur_i$

$\langle \text{address constraints} \rangle ::= tw_j \vee mw_j$

In the case of a single vehicle, the operator sum or max and the subscript i are dropped in the objectives related to routes and vehicles.

Appendix B. Extended technique template for 2-opt

NAME
2-OPT

INPUT
tour

OUTPUT
tour

DEFINITION
FEASIBLE()
EVALUATE()

```
CurrentTour ← InputTour
CurrentCost ← EVALUATE (CurrentTour)
while (  $\{(i, i + 1), (j, j + 1)\} \rightarrow \{(i, j), (i + 1, j + 1)\}$  not examined in CurrentTour ) do
begin
  ExchangeTour ← perform  $\{(i, i + 1), (j, j + 1)\} \rightarrow \{(i, j), (i + 1, j + 1)\}$ 
  if FEASIBLE (ExchangeTour) then
    begin
      ExchangeCost ← EVALUATE (ExchangeTour)
      if (ExchangeCost < CurrentCost) then
        begin
          CurrentTour ← ExchangeTour
          CurrentCost ← ExchangeCost
        end
      end
    end
  end
OutputTour ← CurrentTour
```

PERFORMANCE
 $O(n^2)$ time for verification of local optimality

DESCRIPTION
2-Opt tries to improve a tour by repeatedly exchanging two arcs with two other arcs
#(REF: S. Lin (1965). Computer solutions to the traveling salesman problem. Bell System Tech. J. 44, 2245-2269.)
#(REF: M.W.P. Savelsbergh (1990). An efficient implementation of local search algorithms for constrained routing problems. European J. Oper. Res. 47, 75-85.)

FEASIBLE(t : tour):boolean

```
TRUE
#(MOD: ( $tw_j$  REPLACES <empty symbol>) ->
( $0 \leq k \leq n: t_k \leq l_k$ ; #(TYPE: FORMULA); #(CLASS: ( $tw_j$ )) REPLACES (TRUE)))
#(MOD: ( $\pm$  REPLACES <empty symbol>) ->
( $0 \leq k \leq n: 0 \leq \sum_{0 \leq i \leq k, i \in C} q_i - \sum_{0 \leq i \leq k, i \in D} q_i \leq Q$ ; #(TYPE: FORMULA); #(CLASS: ( $\pm$ )) REPLACES (TRUE)))
```

EVALUATE(t : tour):real

```
 $\sum_{0 \leq k \leq n-1} t_{k,k+1}$ 
#(MOD: (max  $T$  REPLACES sum $T$ ) ->
( $t_n$  REPLACES  $\sum_{0 \leq k \leq n-1} t_{k,k+1}$ ))
```

