Division of Informatics, University of Edinburgh

Centre for Intelligent Systems and their Applications

Knowledge Acquisition and Knowledge Modelling for Operational Research

by

John Kingston, Andrew Tuson

Informatics Research Report EDI-INF-RR-0051

Division of Informatics **March 2000**
http://www.informatics.ed.ac.uk/

Knowledge Acquisition and Knowledge Modelling for Operational Research

John Kingston, Andrew Tuson

Informatics Research Report EDI-INF-RR-0051

DIVISION *of* INFORMATICS
Centre for Intelligent Systems and their Applications

March 2000

Proceedings of the Young Operations Researcher conference (YOR 2000), Cambridge, March 30, 2000

Abstract :

This paper provides the OR practitioner with an introduction and discussion of the areas of knowledge acquisition and knowledge modelling and their utility to OR. To do this, recent knowledge-based approaches to a technique of interest to the OR community, neighbourhood search, will be used as a case study. References for further reading are provided.

Keywords : Knowledge acquisition, knowledge modelling, operational research

Knowledge Acquisition and Knowledge Modelling for Operational Research

[1]John Kingston and [2]Andrew Tuson

[1]Artificial Intelligence Applications Institute
University of Edinburgh, 80 South Bridge, Edinburgh EH1 1HN
[2]Department of Computing, City University
Northampton Square, London EC1V 0HB
Email: jkk@aiai.ed.ac.uk, andrewt@soi.city.ac.uk

Abstract

This paper provides the OR practitioner with an introduction and discussion of the areas of knowledge acquisition and knowledge modelling and their utility to OR. To do this, recent knowledge-based approaches to a technique of interest to the OR community, neighbourhood search, will be used as a case study. References for further reading are provided.

Introduction

The fields of artificial intelligence (AI) and operational research (OR) have much in common, with their interest on applying computation and rational behaviour to real-world problems. For example, neighbourhood search optimisers [**?**, **?**], a class of meta-heuristics which use an analogy with search by trial-and-error to solve difficult optimisation problems, are an active subject of research in both communities. But while OR concentrates on optimisation of search algorithms, the focus of much AI work is on using domain knowledge to guide the search process in an intelligent manner. The need for domain knowledge was recognised by Tuson [**?**], who asserted that "for an optimisation technique to be effective, it *must* contain domain knowledge, even implicitly".

The above assertion is based on both practical experience [**?**] and theoretical results. These results, known as the "No Free Lunch" (NFL) theorems [**?**], show that all optimisers give equivalent average performance over the space of all possible problems.

What does this have to do with knowledge? The argument is as follows. Though the NFL results state that it is not possible to say that one optimiser is better than another when all possible problems are being considered, it is still possible to make comparisons with respect to *subsets* of the problem space. Of course, for one to *predict* which optimiser will be better for a given subset it is necessary to make use of the structure of that subset, i.e. one needs to have *knowledge* about that problem subset. Given that optimiser design can be thought of as a process of selecting an optimiser out of a set of alternatives, then an effective optimiser design must make reference to the designer's knowledge of the problem to be solved (otherwise you would just be guessing).

Of course, this does not inform the important issue of *how* to go about this in practice. To resolve this issue, Tuson [**?**] goes on to propose a **knowledge-based systems (KBS)** approach to optimisation, that organises and classifies the types of knowledge that are needed by a system that performs optimisation using neighbourhood search. He identifies three types of knowledge: *problem solving* knowledge that decomposes the problem and excludes/focuses on areas of the search space; *problem specification* knowledge that specifies desirable solutions (i.e. the evaluation function); and *search control* knowledge that examines how a search space should be tackled.

Knowledge engineering – the science of building knowledge based systems – has several stages that must be tackled to transfer knowledge successfully from a knowledge source (typically a human expert) to a working system. This paper will focus on *acquisition* and *analysis* of knowledge, by briefly discussing key characteristics of expert knowledge; then discussing how that knowledge can be accessed by various techniques; and finally, how a template "knowledge model" for planning tasks can be used to identify and capture the knowledge needed for neighbourhood search problems.

Expert Knowledge

If knowledge belongs to an expert, it must be stored in memory. The goal of knowledge acquisition is to extract that knowledge from memory. Since knowledge consists both of items of information and of dependencies and relationships between items of information, it is also important to preserve the relationships; in short, it is necessary to understand the *structure* in which knowledge exists as well as the "knowledge items".

The building blocks of human memory structure are associations: links between two or more items. There are (at least) four ways in which associations are formed: chunking (extending the capacity of short memory by remembering "chunks" of several items), lexical memory (associating concepts by the shape of their words e.g. alphabetical order), phonetic memory (associating concepts by the sound of their words e.g. nursery rhymes) and semantic memory. Semantic memory associates words (or rather concepts) on the basis of meaning. There are many number of ways that se-

mantic associations can be created; for example:

- Associations based on taxonomic classification, e.g. A car is a vehicle.
- Associations based on similar roles/capabilities, e.g. doctor and dentist.
- Associations based on "closeness", e.g. Big Ben and Tower Bridge.

A good knowledge acquisition technique will not only capture all relevant associations, but will actually mimic the structure of that knowledge in the outputs produced.

Knowledge Acquisition

Knowledge acquisition (KA) is the term used to describe the process of collecting and systematising the knowledge required to build a knowledge-based system. The available techniques for acquiring knowledge from experts are divided such that each one specialises in acquiring one of three types of knowledge:

- **Declarative Knowledge**: knowledge of concepts, objects, properties, and relationships, including categorisations.
- **Procedural Knowledge**: knowledge of processes, sequences, steps and tasks.
- **Constraints**: knowledge of restrictions on procedures or properties.

Some specific techniques are suggested below.

Techniques for Acquiring any Type of Knowledge

Interviewing

Interviews have been the most widely used technique for eliciting knowledge in the history of A.I. They are familiar to both experts and knowledge engineers, they are a well-known technique for acquiring information, and there is literature available on how to conduct a good interview. However, interviews have one big drawback: transcripts. Transcribing a 1 hour interview takes between 5 and 10 hours, and produces a transcript which can be up to 10,000 words long (though a couple of thousand is more typical). Analysing the transcript to find relevant knowledge takes even longer, since it is normally done by combing the transcript for knowledge items and for sentence structures (such as "if A then B" or "the X of Y") that indicate relationships or dependence between these knowledge items. Structured interviews (where everything the expert says is in answer to a prepared question) alleviate the problems but do not remove them.

As a result of the problems inherent in acquiring knowledge through interviews, other techniques have been developed, and are described below.

Rapid Prototyping

In the early days of knowledge based systems, a common technique for system development was “rapid prototyping”. This required a knowledge engineer to acquire a small amount of knowledge from the expert and then to build a small “prototype” knowledge based system that encoded and reasoned with this knowledge. This prototype was then extended and refined with further knowledge until was considered to be a complete system.

While rapid prototyping had its problems (notably in documentation and maintenance of systems), it did prove to be an effective knowledge acquisition technique; showing the prototype to an expert was a good way of eliciting criticism of past understanding as well as identification of incomplete knowledge. In large projects, or where a good prototyping environment is available, it might be considered as an aid to knowledge acquisition, despite the extra effort over and above paper- or diagram-based knowledge acquisition.

Techniques for Acquiring Declarative Knowledge

Laddering

The laddering technique is used to elicit a taxonomic hierarchy from an expert. It’s a useful technique to use early in knowledge elicitation, because it only requires the knowledge engineer to know one domain term.

The technique works as follows: Take a large piece of paper, or a whiteboard, and write the domain term on it. Then use a series of standard questions to acquire terms which are down, up, or across the hierarchy. For example, to move down the hierarchy (i.e. to find some subclasses or instances of the term), you might ask “What are examples of <ITEM>?”; to move up (i.e. to find superclasses), you might ask ”What do <SAME LEVEL ITEMS> have in common?”, and to move across (i.e. find other classes at the same level) you might ask “What other examples of <SUPERCLASS> are there apart from <ITEM>?”. There are also questions for defining intermediate superclasses and for eliciting identifying attributes of a class or instance.

The laddering is a good technique for acquiring a lot of knowledge quite quickly, and the end result can be treated as an expert knowledge structure with very little further analysis.

Card Sorting

Once a suitable set of domain terms have been collected (using the laddering or by other means), the card sorting technique can be applied. The knowledge engineer is required to write the domain terms on index cards (one for each term), which are then handed to the expert. The expert is asked to sort the cards into piles, according to any criterion which seems sensible to him. The knowledge engineer then asks which criterion was used and what each pile indicates, notes which pile each card is in, and

shuffles the cards. He then hands them back to the expert and asks the expert to sort the cards again according to a different criterion. This procedure continues until the expert can think of no more ways to sort the cards.

Card sorting is a simple but surprisingly effective technique. Every criterion that is chosen is an attribute of the domain terms, and every pile represents a value for that attribute. The knowledge which is obtained therefore consists of a set of attributes, and a complete set of values for those attributes. Card sorting can also be used to elicit further domain terms by asking the expert if any cards are ”missing” from any piles of cards which he creates; this is very effective because it accesses associative memory (i.e. it’s easier to remember something if you’re reminded of something else associated with it).

Repertory Grid

The repertory grid technique is derived from George Kelly’s “personal construct” psychology [**?**], which proposes that each person creates a number of dimensions on which he/she evaluates events and states; for example, a fall in Far Eastern currency values will have different implications to an investment banker and a backpacker. A repertory grid is a technique designed for eliciting constructs and measuring our opinions of particular events using the constructs. It is “a two-way classification of data in which events are interlaced with abstractions in such a way as to express part of a person’s system of cross-reference between his personal observations or experience of the world (*elements*) and his personal *constructs* or classifications of that experience”. In practice, the knowledge engineer draws a 2-dimensional table, labels each column with knowledge items (corresponding to Kelly’s ’elements’), and then labels rows with constructs as they are elicited. The cells of the grid are then filled in with values for each construct.

What has been described so far is similar to the card sorting technique. However, the repertory grid has important differences from card sorting. First and foremost, the values which are used to fill in the grid are numbers, on a 1-5 (or 1-7) scale. Secondly, the technique for eliciting constructs is different. The repertory grid technique uses “triadic elicitation”, in which three data elements are picked at random, and the expert is asked to name one way in which one of them differs from the other two. This is then used as a construct to rate all the elements. Finally, and most obviously, the use of numbers as the values in the grid permits statistical cluster analysis to be performed on the knowledge items or the constructs. Cluster analysis compares two or more ordered lists of numbers against each other, and calculates how different the lists are; for example, (3 4 2 5 1) only differs by 1 point from (3 4 2 5 2), but it differs by 3 points from (3 4 2 3 2). This allows the repertory grid to “deduce” associations and interrelationships between knowledge items.

Techniques for Acquiring Procedural Knowledge

Protocol Analysis

Protocol analysis is a technique which requires asking the expert to perform a knowledge-based task, and to explain why he's doing each action as he performs it. The expert's words are then transcribed, and the result is known as a protocol. The technique is best carried out while the expert is solving a real problem, although it can be done by retracing a previous scenario. If there are good reasons why the expert should not be distracted by a knowledge engineer while performing the task (as in a case where knowledge engineers elicited knowledge from air traffic controllers), then "shadowing" is possible, in which one expert performs the task and another expert describes what is being done.

This technique is good for capturing what an expert really does, as opposed to what he claims he does. It's good for working with experts who are not used to discussing their work in abstract terms; asking about concrete actions nearly always elicits a response. It's also good for finding out what interactions take place with people or information sources, and whether there are any real-time restrictions on these interactions.

Drawbacks of this technique include the need for a transcript (although analysis is no more onerous than for a structured interview), the need for video recording and/or very careful note taking, and the problems of interfering with an expert's problem solving process by asking about it. However, none of these problems are serious enough to prevent it being a good weapon in the knowledge engineer's armoury.

20 Questions

For those who rarely listen to BBC Radio 4, "20 Questions" is a verbal game in which one contestant thinks of an object, and the other contestants are allowed to ask up to 20 closed questions (questions which can be answered Yes or No) in order to determine what that object is. The knowledge engineering version is similar:

- The knowledge engineer is required to think of a hypothetical problem which needs to be solved.
- The knowledge engineer then meets with the expert, who must ask the knowledge engineer *verbally* for all the information needed to solve the problem. Any questions are allowed (not just closed questions), and any number of questions are allowed, although it's very unusual to require more than 20.
- The knowledge engineer then examines "what if" alternate answers had been given to earlier questions.

This is another technique which, although simple, is remarkably effective. By forcing the expert to ask for all the necessary information verbally, it quickly becomes

clear if any perceptual information is being used, and which information is considered particularly important. It also becomes clear that certain questions are always asked first, usually because they narrow down the search space significantly, or because they confirm or deny common solutions. The main drawback of the 20 questions technique is that it requires the knowledge engineer to know enough about the domain to answer the expert's questions. This can be worked around, however, by having a second expert collaborate with the knowledge engineer to answer the questions.

Techniques for Acquiring Constraint Knowledge

There are no widely recognised knowledge acquisition techniques that focus specifically on constraint knowledge. In the context of the knowledge needed for operational research, it is perhaps unfortunate that the knowledge needed for search problems is in inverse proportion to the available techniques.

While it is often possible to acquire constraints by adapting any of the techniques described above, some suggestions have been made for techniques that rely on associative memory to elicit constraints (see [**?**]). These suggestions are:

- **Conflict** - convert a statement into another statement that names interested parties who might be viewed as in conflict. For example, "Town planners see a need for change and renewal which is not necessarily appreciated by the aged who have lived in the area all their lives";
- **Contradiction** - trying to contradict an earlier statement by taking an opposing viewpoint (e.g. "slum clearance disturbs old people – old people need safety & hygiene");
- **Complication** - identify any factors which should really have been considered when making a previous statement e.g. "Old folk need modern housing because they need safety & hygiene" is subject to the complication "But modernisation usually means increased rent charges";
- **Similarity** - try to think of, and then think through, an analogous situation (e.g. slum clearance is to housing as a plough is to a field; the process of slum clearance destroys previous street patterns, but opens up the area for new growth);
- **Chance** - pick a word from a dictionary and see if it sparks any new ideas. For example, the word "softly" might suggest soft music, which in turn could lead to a consideration of the difficulties of moving grand pianos and other accumulated furniture into modern housing.

While Lawson's suggestions are not as systematic as the other techniques described above, they provide the components for a more comprehensive future technique for acquiring knowledge of constraints.

Knowledge Modelling Guides Knowledge Acquisition

Having a suite of knowledge acquisition techniques such as those described above is all very well. But it still does not assist in determining what types of knowledge need to be acquired for a neighbourhood search problem. As noted earlier Tuson [**?**] characterised the necessary knowledge as problem-solving knowledge, problem specification knowledge, and search control knowledge – but how do these categories map to declarative, procedural and constraint knowledge?

It seems fairly clear that Tuson's "search control" knowledge maps to procedural and constraint knowledge. Similarly, the evaluation function must be in terms of items, objects, and relationships within the domain – declarative knowledge. But "problem solving" knowledge needs further analysis.

What exactly is meant by problem-solving knowledge? Tuson provides a suitable working definition thus: "problem-solving knowledge is knowledge from the problem domain, that is not used in specifying the quality of a given solution, but instead is used to guide the optimiser's decisions so that an effective search behaviour is produced." TEXT DELETED HERE. Tuson [**?**] proposes that the types of problem-solving knowledge that neighbourhood search optimisers fall under the following categories:

1. **Features**. Which problem features correlate with solution quality?
2. **Linkage**. What is the structure and strength of interaction between certain problem features?
3. **Search Space Reduction**. Which solutions can be excluded from the search?
4. **Initialisation**. In which areas of the search space do good solutions lie?
5. **Move Selection**. Given a set of possible moves, which is most likely to result in a better quality solution?

Tuson then goes on to show how such a decomposition can be reconstructed formally to provide specifications for neighbourhood operators that captures such domain knowledge.

TUSON'S KNOWLEDGE BASED APPROACH IS REMINISCENT OF THE WAY THAT KNOWLEDGE IS MODELLED by a major methodology for developing knowledge based systems, known as CommonKADS. In the remainder of this paper, CommonKADS will be described briefly, and then the application of this modelling approach to knowledge based planning (which requires neighbourhood search) will be described and considered for its applicability to general neighbourhood search problems in OR.

The CommonKADS Methodology

The CommonKADS methodology [**?**] is a collection of structured methods for building knowledge based systems. CommonKADS views the development of a knowledge

based system as a modelling activity, and so the heart of these methods is the construction of a number of models which represent different views on problem solving behaviour. The CommonKADS methods for developing knowledge-based systems have proved their usefulness repeatedly over a range of different tasks.

The key element in the success of CommonKADS is the library of generic **inference models** which can be applied to tasks of specified types. These models suggest the inference steps which take place in a typical task of that type, and the roles which are played by domain knowledge in the problem solving process.

For example, the generic model for a systematic diagnostic task (e.g. [**?**]) includes inference steps such as **decomposing** a set of possible faults, and **matching** observed values against expected values. This model also shows that the set of possible faults plays two roles in the diagnostic process; firstly as a part of a model of the behaviour of a faulty system, and secondly as hypothesised causes of the symptom(s) currently being observed.

These generic models can either be used in a top-down manner, as frameworks for knowledge acquisition (e.g. [**?**]), or they can be used to verify the completeness of models developed bottom-up by analysis of the domain (e.g. [**?**]). In this paper, we will take the former approach; we will examine CommonKADS inference structures relevant to neighbourhood search problems in order to identify what types of knowledge need to be acquired.

CommonKADS Model for Planning: Top Level

Given that neighbourhood search problems often occur in the context of a planning task of some kind, it seems sensible to start our investigations with a generic inference structure for knowledge based planning (Figure 1). This inference structure is derived from the operation of a well-known AI planning system, O-Plan [**?**]. O-Plan provides a generic domain independent computational architecture suitable for command, planning and execution applications. It makes use of a variety of AI planning techniques, including a hierarchical planning system which can produce plans as partial orders on actions; an agenda-based control architecture; incremental development of "plan states"; and temporal and resource constraint handling.

- *Condition satisfaction*: Adding a new activity, or further constraints on currently planned activities, in order to resolve the issue;
- *Achieving*: adding new issues to the plan which, if resolved, will allow the current issue to be resolved;
- *Decomposing*: Expanding the issue into a number of sub-issues.

A typical "run" through the inference structure, in which rectangles represent declarative knowledge and ellipses represent inference steps, would see the following operations taking place:

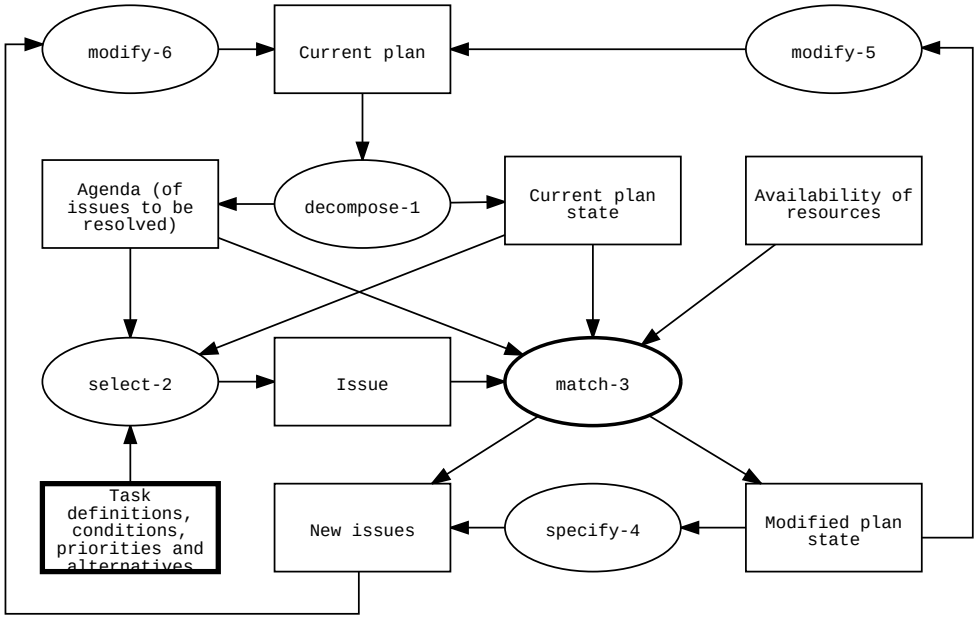

Figure 1: Inference Structure for Knowledge Based Planning, Derived from O-Plan

- The **current plan state** is notionally decomposed into three components (**decompose-1**): the **agenda** of issues which are to be resolved, the **current plan entities** and the **constraints**. This decomposition does not alter any of these structures; it simply makes explicit the role which each component of the plan state plays in the problem solving process. These roles are described in [**?**].

- From the agenda of issues, at least one **issue** is selected for resolution (**select-2**). The choice of an issue depends on a number of factors monitored by the **controller**, such as the available processing capabilities, the knock-on effect on other issues, etc.

- Pattern matching between issues and possible activities is used to find an activity which is capable of resolving the current issue, perhaps by adding entities to the plan, or by creating new issues (**match-3**, which is expanded into more detail in other diagrams). Issues may be resolved in one of three ways:

  Each of these options corresponds to a single O-Plan “knowledge source”.

- The resulting agenda of issues, plan entities and constraints are assembled, and used to update the current plan (**assemble-4**).

From this top level model, it can be deduced that the knowledge to be acquired includes the initial plan state, the resources available, and the desired goal state. This

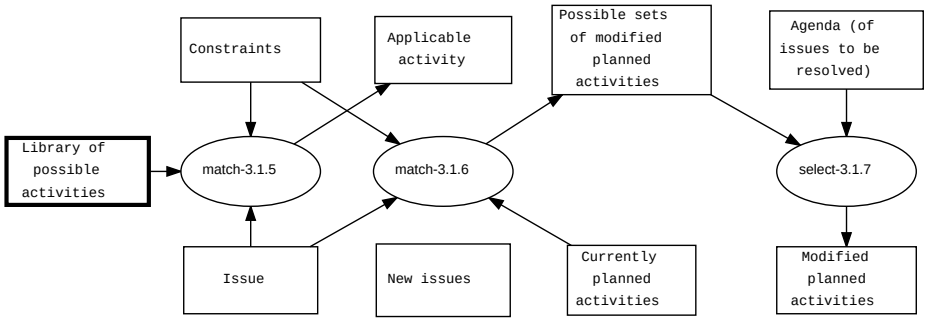

Figure 2: Inference Structure for Resolving an Issue by Introducing New Activities or Constraints into the Plan

information is hardly earth-shattering, but it is worth noting that all of this knowledge is declarative knowledge, according to CommonKADS; so techniques such as laddering, card sorting and the repertory grid may be used to acquire this information.

It is also worth noting that the three “knowledge sources” that are suggested represent three strategies for approaching all or part of a neighbourhood search problem. Condition satisfaction represents the ”forward” approach to a search problem – deciding which search space operator to apply next, based on the current plan state. Achieving represents a “backward” approach to a search problem – determining what states must be achieved in order to fulfil the conditions of the goal state, and setting up these states as “sub-goals”. Decomposing represents the setting up of intermediate sub-goals between the current state and the goal state. Since certain search problems are much better tackled “backwards” rather than “forwards”, it is wise to consider the knowledge available for a neighbourhood search problem to determine if the knowledge for “achieving” or “decomposing” is available – and, if not, to make efforts to collect it.

CommonKADS Model for Planning: Finding Applicable Activities

The next stage of CommonKADS is to create more detailed inference structures that describe one of the inference steps in the top level model in more detail. Since there are three ways of performing **match-3** in O-Plan, three sub-inference structures are required to represent each of the three “knowledge sources”. In this paper, one will be described – the “condition satisfaction” knowledge source (Figure 2).

The *condition satisfaction* knowledge source can be applied when the conditions for an activity which fulfils an outstanding issue are found to be matched. Conditions typically consist of one or more resources being available. For example, if an issue in the plan was to arrange transport for a mountain rescue team from A to B, then one possible activity (**match-3.1.5**) might be to transport the team by helicopter. The conditions of this activity might be that the mountain rescue team is present at a helicopter landing site, and a helicopter is also present at that site; the expected plan

state produced by the partial plan developed to date will determine if these conditions can be fulfilled. (**match-3.1.6**). If the conditions of an issue are fulfilled, and that issue is selected as the best method of transporting the team (**select-3.1.7**), then that issue is removed from the agenda. The plan itself is also modified, in any or all of the following ways:

- New planning entities may be created (e.g. "helicopter X must land at location A");
- New variable restrictions may be enforced (e.g. "the helicopter must use the backup landing site at A");
- New temporal orderings may be introduced (e.g. "the helicopter has to refuel; this must be done before flying to A").

The other two "knowledge sources" can be modelled similarly.

For the purposes of knowledge acquisition for neighbourhood search, we learn the following from this inference structure:

- Applicable activities (i.e. search operators) are considered as **declarative** knowledge, with conditions and actions described in a declarative fashion. This is an important insight for knowledge acquisition, for it allows us to use techniques for acquiring declarative knowledge to acquire the conditions and actions of search operators and to build an exhaustive set of search operators, as well as techniques for acquiring procedural or constraint knowledge.
- Constraints must be expressed in the same terms that the conditions and actions of activities are expressed, which requires that the knowledge acquisition of constraints parallels the acquisition of search operators.

This concludes our description of the CommonKADS model that is derived from O-Plan. However, O-Plan itself has a controller that not captured by the CommonKADS model. This controller has various features that may be of use in neighbourhood search, including a (fairly coarse) method of search space pruning. This pruning is achieved by placing restrictions of varying strength on the circumstances under which conditions of activities can be considered to be fulfilled. For more on this concept, see [**?**].

A CommonKADS Model for Activity Selection

While we have reached the most detailed level of the the generic inference structure for planning, we have not finished our exploration of the knowledge that needs to be acquired for neighbourhood search problems. The problem solving knowledge identified in [**?**] requires not only identification of search operators, but knowledge that

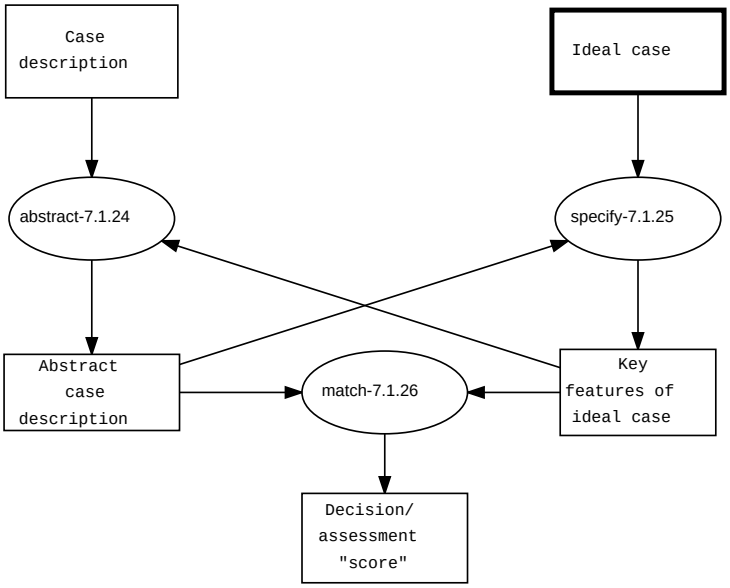

Figure 3: Inference Structure for Assessment Tasks

enables the selection of good search operators. For this, we must expand the inference step *select-3.1.7* into another level of inference structure, drawing on CommonKADS' generic inference structure for selection tasks, rather than the generic inference structure for planning tasks.

In fact, CommonKADS offers a generic model for **assessment** tasks rather than selection tasks; selection consists of repeated assessment of items, followed by selection of whichever receives the highest assessment "score". The simplest version of the model is shown in Figure 3; more complex versions are available, and problem-specific versions can be generated by following the configuration questions outlined in [**?**].

The best way to describe this inference structure is through an example. If an organisation is interviewing candidates for a job, then the "ideal case" will consist of the ideal candidate for the job. The key features of the ideal candidate will be the skills expected; as an example, let us say that these skills include a good degree in computing or a related subject, 3 years' experience in C++ programming, and the ability to converse in French and German.

Each candidate's CV can be considered to be an abstract case description, for it abstracts from the candidate's life history those details that are relevant to this job. The CVs are then matched against the desired skills list, and judgements are made concerning the relative importance of features; for example, should a candidate with a degree in computer science, 3 years' experience in C and 6 months' experience in C++ be rated higher or lower than a candidate with a First in electronic engineering, 5 years' experience in C++, but who can barely understand German?

It might be thought that much of this functionality is subsumed in neighbourhood

search by the evaluation function. But what this inference structure shows us is that the knowledge that must be acquired to represent Tuson's "problem solving" knowledge ought to sustain *several* evaluation functions, one for each "key feature" of the ideal case. Taking a wargame as an example of a neighbourhood search problem, there should enough knowledge for at least two evaluation functions: one to determine the value of destroying enemy resources, and another to evaluate the risk of losing one's own resources. The relative weightings given to achievement and risk will determine whether some parts of the search space can be ruled out because they are "suicide missions", or whether missions that open the path to attack dangerous enemy weapons are considered to be "good" areas of the search space. Search options that accomplish the loss or anticipated loss of a friendly resource without commensurate reduction in enemy resources (by moving an aeroplane to a point where it cannot refuel before crashing, for example) should be ruled out almost immediately. Of course, this requires some lookahead through possible future states, which must be accomplished efficiently; but that is not CommonKADS' concern.

In summary: while CommonKADS' generic inference structures do not specify the exact knowledge that needs to be acquired for a neighbourhood search problem (their genericity prevents this), they can provide a useful guide to what knowledge must be acquired for a wide range of problems, including neighbourhood search problems, and can also help determine which knowledge acquisition techniques might be helpful.

Relating CommonKADS to Neighbourhood Search

How do the CommonKADS inference structures above relate to the framework for neighbourhood search problem solving knowledge proposed by Tuson? Though they may initially look quite different, they are not incompatible. The differences arise for a number of reasons which are described below:

- **Bottom-up vs top down** - CommonKADS, as it is applied here, works in a top-down fashion, collecting and modelling the domain knowledge, and using this to outline the structure of inference required. Tuson's framework is constructed bottom-up by a consideration of the forms of knowledge that a neighbourhood search optimiser can exploit. It should therefore not be surprising that they are organised differently.

- **Application vs algorithm centred** - COMMONKADS CONCEBNTRATES ON THE STRUCTURE OF AN APPLICATION RATHER THAN ITS EFFICIENT IMPLEMENTATION. IN FACT, COMMONKADS PROVIDES HIGHER LEVEL MODELS THAT ALLOW THE ORGANISATION THAT MIGHT USE THE APPLICATION TO BE CONSTRUCTED. This helps to ensure that the optimisation system produce is effective in the situation where the system is deployed. Tuson's framework does not address this issue.

- **Granularity of analysis** - in the planning example above, each of the modification operators (activities to resolve an issue) are considered as KNOWLEDGE CONCEPTS[1] IN THEIR OWN RIGHT. This is in common with other attempts to knowledge-model neighbourhood search optimisers [**?**]. Tuson's framework sees operators as being a *composition* of knowledge concepts. This finer-grained level of analysis has the advantage of being able to show commonalities between superficially quite different operators. For instance, the recombination operator in evolutionary algorithms is in fact derived from the same knowledge concepts as the conventional unary operator (mutation).

- **Strategic knowledge** - as noted earlier, the knowledge sources in Tuson's framework correspond to strategic knowledge that is used to guide the procedural choices made by the optimisation algorithm. This is one aspect of domain knowledge that COMMONKADS' GENERIC INFERENCE STRUCTURES DO NOT EXPLICITLY MODEL; COMMONKADS DOES HAVE A FRAMEOWRK FOR MODELLING INDIVIDUAL STRATEGIES AS "PROBLEM SOLVING METHODS", BUT FEW SUCH METHODS HAVE BEEN PUBLISHED TO DATE.

So in effect the two approaches are complementary. The open research issue now becomes one of integrating the two together. This would be desirable for a number of reasons.

- Tuson's framework allows for the formal derivation of neighbourhood operators.

- Due to its application-centric approach, CommonKADS situates the system in its application domain, and aspect of system design that Tuson's framework does not address.

- CommonKADS better assists with knowledge acquisition, providing explicit guidance.

- A number of heuristic search frameworks have been proposed in the OR literature [**?**, **?**], which in principle map well onto Tuson's framework.

- Following from this, assuming that CommonKADS and Tuson's work can be reconciled, this should prepare the way for extending CommonKADS to deal with other OR heuristic optimisation methods.

Two factors are in our favour in this regard. First, CommonKADS is a modelling-based methodology and thus makes no strong commitment to either the search method being used, or the granularity of analysis. On the other hand, the two approaches are orthogonal and therefore they should be best seen as alternative knowledge-level

[1]These "knowledge concepts" are usually called "knowledge sources" within the OR literature. They are referred to as "knowledge concepts" here to emphasise CommonKADS, view of them, and to avoid confusion with O-Plan's "knowledge sources" (described earlier).

viewpoints on an optimisation system. Similar parallels can be drawn with software engineering where it is perfectly acceptable to view a system in multiple ways and then reconcile their strengths and weaknesses in the design of a system.

Conclusion

The aim of this paper was to introduce some current work in knowledge acquisition and modelling to OR practitioners. The reason for this is that, in both AI and OR, optimisation problems are of great interest and that from whichever field the optimisation practitioner is from, they need to exploit domain knowledge.

This paper described a number of knowledge acquisition strategies and noted that knowledge modelling can assist greatly in this process. To this end, two KBS frameworks for modelling neighbourhood search were described and contrasted.

The challenge now is to strengthen and reconcile these approaches and broaden their appeal and applicability to the wider OR community.

References