Using Search in Knowledge-Based Engineering

Andreas Junghanns[1] and **Rüdiger Klein**[2]

Abstract. Search has been instrumental in many well known successes of AI problem solving. These successes are currently mostly restricted to well structured domains like games or scheduling. The resulting search spaces can be searched using special purpose search heuristics and strategies. In this position paper we formulate the problems and challenges encountered when these successes are to be extended to more complex domains. Engineering provides a good test bed for these kinds of efforts: though quite complex and diverse, engineering applications have a clear structure and are (generally) well understood. Many different problem solving methods are typically used in combination. The resulting search spaces tend to be huge, they are typically dynamic, and include continuous as well as discrete dimensions. We analyze in which way current search techniques can be extended and modified in order to be applicable under these circumstances.

1 INTRODUCTION

Knowledge about real world domains and about problem solving in such domains tends to be quite complex. Building up knowledge bases and problem solvers in such domains will easily result in huge efforts - especially if we want to overcome the restricted scope and the isolation of current knowledge-based engineering (KBS). Consequently, reusing domain knowledge as well as problem solving methods is an essential issue towards a broader and more efficient application of knowledge-based technology. For this purpose, we have to find ways allowing us to formulate the domain knowledge and to describe problem solving methods and their relation to the domain knowledge in an application-independent way.

Search, as one of the fundamental artificial intelligence (AI) techniques, was instrumental in many, if not most, well known successes of AI problem solving. These successes are currently mostly restricted to well structured and easily comprehensible domains like games or scheduling. They use special-purpose solvers with "built-in" knowledge about the search space to achieve high performance. This knowledge is available because the search spaces are well structured, known beforehand and analyzed by humans, and the objective of the problem solving process is well defined and fixed (definitions of goals or function to be optimized).

1 Freightliner Corporation, Enterprise Product Documentation, Portland/Oregon, USA, AndreasJunghanns@freightliner.com

2 Daimler-Chrysler AG, Research and Technology Department, Berlin, Germany, Ruediger.Klein@daimlerchrysler.com

Knowledge-Based Engineering (KBE) offers new challenges to AI, because its domain knowledge has a considerable diversity, search spaces are huge, and search objectives are changing constantly. In fact, KBE is most often iterating through synthesis and analysis steps with the goal of modifying the problem definition, and thus both, the search spaces and objectives. Engineering applications tend to be relatively well defined with more or less clear rules, constraints, and criteria - at least if compared with domains like common sense reasoning or natural language understanding.

This paper outlines the resulting challenges for adapting and extending traditional AI search methods to the applications in KBE. These challenges consist mainly in finding a general enough framework for problem and knowledge descriptions that still allows for efficient treatment by a generic problem solving method, here search. These and other problems are outlined in this paper and possible plans of attack are proposed.

This paper is organized as follows: In Chapter 2 we outline how human problem solving can achieve its power and efficiency. In Chapter 3, we circumscribe what this means to knowledge-based technologies, with special emphasis on reusability of domain knowledge and problem solving methods. Chapter 4 discusses special properties of the knowledge in engineering. Chapter 5 characterizes traditional problem solving methods in the engineering domains. Chapter 6 surveys knowledge modeling and reasoning for knowledge-based engineering. Chapter 7 discusses strengths and weaknesses of traditional search methods, and Chapter 8 describes the consequences and challenges, as well as possible solutions when traditional search methods are to be used for engineering problems. Chapter 9 contains a real world example as an illustration: a truck chassis design problem. We conclude in Chapter 10.

2 HUMAN PROBLEM SOLVING IN GENERAL

Where does human expertise come from (in any area)? What is the secret of human intelligence, and what can it tell us about how to improve current methods? A hypothetical answer focussing on our topic "search in knowledge-based engineering" may be described as follows:

Common sense and domain-specific knowledge allows us to (re-)formulate a given problem in a way that it can be solved. Problem descriptions are completed using "common sense" or domain knowledge including assumptions and other kinds of meta knowledge:

A) the relevant aspects are collected and analyzed in their interdependencies,
B) the irrelevant points are dropped; and
C) disambiguities of any kind (anaphors, metaphors, or homonyms in natural text, etc.) are resolved.

This kind of problem solving is done by humans quite often unconsciously and apparently without any effort. Nevertheless, it can be the result of a complex reasoning process. The knowledge analysis process may include meta-knowledge and meta-level reasoning. This allows us to have a view on our own knowledge and on our problem solving process: how precisely do we know what we belief to know; how important is it to know this precisely; how are the dependencies between belief A and belief B; etc.

Human pattern-matching capabilities allow us to deal with very complex information very effectively and efficiently. Heuristic knowledge results from problem analysis and is used efficiently to control the problem solving process. Humans seem to be able to detect the "natural fault-lines" in problems to break them up into sub-problems. These are the "weak" dependency links among clusters of objects and objectives and their relations. Identifying clusters allows humans to try and identify fatal conflicts early on where they are most likely, without wasting valuable time.

If there are no patterns available human problem solving "falls back" into a kind of puzzling and search. However, this search is not just goal oriented, but remains alert to the possible detection of any kind of (meta-)knowledge that might help to solve the problem more efficiently (see A).

3 KNOWLEDGE MODELING AND KNOWLEDGE-BASED PROBLEM SOLVING

If we adopt this view on intelligent (human) problem solving, what are the consequences for effective and efficient problem solving by knowledge-based systems? In the following, we will concentrate on three aspects: Knowledge modeling, problem solving methods (i.e., search), and control.

3.1 Knowledge Modeling

Today, applications of knowledge-based technologies are based on special purpose knowledge models. Typically, knowledge engineers (in cooperation with domain experts) go through a knowledge acquisition process. They analyze all relevant aspects in the domain by hand and formulate a complete model of the necessary generic and case-specific knowledge. Then, the knowledge represented in this way will be "mapped" onto the selected problem solving methods which can use it directly. The necessary control information is coded quite often implicitly (for instance, as sequence information), or explicitly as probabilities, weights, etc. This approach works well in relatively well structured domains with manageable (decision) complexities [1]. Typical examples are game playing, scheduling, symptom-based or model-based technical diagnosis, and (some sub-areas of) configuration. Here, the knowledge engineer can easily identify the relevant "pieces of knowledge", and represent them in a way which corresponds to the problem solving techniques to be applied.

What are the limitations of this approach? The knowledge modeling process itself as well as the usage of the problem solving methods is based on assumptions. Normally, these assumptions are left implicit - at best the knowledge engineer will keep them in mind. The knowledge engineer is responsible for that the knowledge modeling process and the usage of the modeled knowledge in the problem solving methods will be done in a way which respects these assumptions. Of course, this provides limitations to the reusability of the modeled knowledge as well as of the applied problem solving methods.

Another limiting point to be mentioned is the scope of the modeled knowledge. If the knowledge is modeled with a concrete application in mind one can focus the modeling on the relevant aspects in this application. Much more complicated is knowledge modeling in general - without a specific application. Then one has to consider all possible aspects, all interdependencies of this knowledge with other "pieces" of knowledge, and all possible usage of this knowledge - including the applied assumptions. The advantage of this approach is its result: generic, application-independent knowledge. This knowledge can be reused in many different applications, by different problem solving methods. Currently, it is an open research issue if this is even possible at all in its full extent [11,6].

The price to pay for the improved generality of the modeled knowledge is the need for additional knowledge processing. Given a concrete problem description in terms of generic knowledge, this problem description has to be "pre-processed" using the generic knowledge. All relevant issues have to be identified - and all irrelevant aspects have to stripped away. What currently is done by a human knowledge engineer will then be part of the automatic problem solving process: to re-formulate the problem in a way that it can be solved more efficiently by the selected problem solving method. This may include complex reasoning activities: common sense, default, context reasoning, etc.

3.2 Problem Solving Methods

As outlined in Chapter 2, the power of human problem solving depends (among other things) on our capability to analyze and to structure a problem into smaller, only weakly coupled sub-problems. In knowledge modeling, tasks and task structures are used to describe this process (a la CommonKADS [12]). The resulting task structure reflects the main properties of the search space.

In knowledge-based systems, problem solving is done (to a large extent) by searching. From this viewpoint, a problem is sufficiently defined when a description of the world is given and the goal criteria are sufficiently well defined. Traditionally, worlds are implicitly defined as states of the world and rules about how to transform one state into another. Goals are states of the world for which a certain condition is true. Such a

problem description is sufficient to find a solution, if one exists. However, without further information, a solver (human or machine) would have to resolve to simply and blindly traversing the search space until a solution is encountered. Of course, this is not very efficient, and in many real world problems simply intractable. For a more "intelligent" approach to finding a solution, more information, i.e., domain and control knowledge and an algorithm that uses this knowledge is required to find solutions more efficiently than blind traversal.

Search in AI problem solving has two aspects:

First, the decomposition of a problem into (weakly coupled) sub-problems implies that the overall solution is a composition of the partial solutions. The sub-problems are only weakly coupled - but they ARE coupled. Therefore, one has to arrange the composed solution in a way that everything fits together. Searching through the diverse alternatives accomplishes this, because quite often it can not be seen in advance if the various partial solutions are compatible.

The second search aspect comes into play as follows: Human problem solving is done largely by a kind of pattern matching, associative reasoning, and the like (take for example game playing, natural language understanding, or vision). In AI, this is currently "simulated" by search. In other words: human problem solving can avoid search wherever the right pattern-matching capabilities are available (or search is done here *as* pattern matching).

Whereas the first point shows a striking similarity between human intelligence and AI (what can be used in knowledge modeling and search control), the second point results in an essential difference between human problem solving and KBS. E.g. computer chess uses rather different methods than human chess players. It seems to be an interesting research issue whether human-like pattern-matching capabilities can be gained in KBS as kind of "compiled" results of search, possibly somehow similar to the training phase of neural networks. But with some of the successes that AI has produced recently we are becoming increasingly aware of the fact that human problem solving is often far from optimal, neither in methods, nor in results achieved. Especially decomposition can lead to globally sub-optimal solutions that are hard to recognize as such. That leads to interesting problems.

3.3 Control of problem solving

The most well known successes in AI have heavily relied on speed; fast traversal of the search space to compensate for poor domain and control knowledge. Hard coding of the knowledge into the search algorithm has produced application-dependent programs with astonishing capabilities in many different application domains such as games and scheduling. Some of these programs, running on fast, even special-purpose, hardware, are capable of visiting and evaluating billions of states per second. An impressive array of general search algorithms and search enhancements has been proposed that boost the efficiency of these algorithms even further by many orders of magnitude. The resulting overall performance is surpassing even peak human performance convincingly in domains generally admitted to require intelligence to solve.

However, hard coding knowledge into algorithms does not allow for reuse of neither knowledge nor algorithms. And this approach is only applicable in domains with well-defined and clear structures and with fixed, well-defined objectives/goals. It is not applicable to typical engineering problems.

4 KNOWLEDGE IN ENGINEERING

4.1 Diverse Knowledge Structures

One of the most fundamental obstacles to algorithmic treatment of knowledge-based engineering applications is the vast variability of the knowledge. The following list is by no means exhaustive but serves to illustrate the diversity we have to deal with:

1) Objects are grouped in classes, taxonomies and instances to describe relationships of various kinds;
2) Relations among objects are given and attributes describe properties of objects;
3) Different kinds of constraints have to be considered: arithmetic and geometric ones – dealing with real number domains, finite discrete domain constraints, and various logical expressions;
4) Inheritance via class hierarchies and taxonomies can supply default values; knowledge can be given for different abstractions where the different levels of abstractions are connected with relationships (part-of etc.);
5) Different (static) knowledge categories have to be dealt with: functions, structures, behaviors, states, constraints, statistics etc. and their relationships;
6) Various dynamic (problem solving related) knowledge categories exist: requirements, design descriptions, unsolved sub-goals, conflicting decisions, ...

4.2 Engineering Problem Solving Is Highly Dynamic

Whereas other application domains can rely on the problem being fixed, engineering problem solving exhibits many variables, constraints and values that are generated and changed during problem solving. Not all relations and constraints are known in advance: they are the result of intermediate analysis steps during the problem solving process. Conflicts may occur for which no solution can be found because there is no solution (or because the incomplete problem solving method can't find any) and therefore those constraints involved in this conflict will have to be withdrawn or relaxed (depending on their nature and other aspects).

There are mainly two types of (engineering) problems w.r.t. constraints:

1) The under-constrained ones: there is a lot of freedom which has to be used in order to find a good or near-optimal solution. Some of these optimality criteria can be formulated as constraints; others as global parameters (costs etc.) of the total solution which results in a comparability of different solutions.

2) The over-constrained ones: there is no simple solution, and one has to find those constraints which can best (with a minimum of costs) be withdrawn or relaxed.

Just as documented in so many other domains, the hard problems are in the middle of the two main types. In this region it is hard to proof that no solution exists and also hard to find a solution that can be found. Especially when engineers are trying to push the limits of current technology, this is the kind of problem they will attempt to find solutions for.

5 HUMAN PROBLEM SOLVING IN ENGINEERING

These aspects are also relevant for human problem solving in engineering. They are not easy to solve. Engineers developed sophisticated techniques to deal with such complex tasks:

5.1 Human Engineers Structure the Entire Problem Solving Process

From conceptual to detailed problem solving: A rough problem description is iteratively developed into a more detailed solution. This method tries to analyze the problem space to understand in which way a sub-problem or its solutions/conflicts are related to other sub-problems and their solutions/conflicts. Only the main characteristics of the problem and the main acting principles are considered. The restriction of human problem solving capabilities to relatively simple situations is an important reason for this approach. In detailed design the principled solution obtained in the earlier design phases are fully elaborated: more and more of the detailed constraints are taken into account.

Hierarchical refinements: This approach attempts to first solve the design problem on an abstract level, possibly with some approximations. This will result in some boundary conditions or requirements to the next lower abstraction level (example: total car design -> power train design -> gear design -> ...).

5.2 Humans Use Generic Knowledge as well as Knowledge from Former Cases

Problems like similarity and retrieval still pose considerable challenges to main-stream AI.

5.3 Priorities

Humans are able to discriminate (prioritize) between relevant and less relevant (sub-) problems (sometimes with dramatic mistakes!), and between problems which are easy and more complicated to solve. A “feeling” for which (sub-) problems might adversely interact is also helping to guide the problem solver in her task. These forms of experience allow engineers to develop problem solving strategies, such as “prove to be impossible” or the opposite “prove to be possible”.

5.4 Analysis and Synthesis

Engineering problem solving switches between analysis phases (or steps) and synthesis (real) problem solving steps. The latter *modifies* the solution by *adding new* information to it. Analysis results in new requirements (new sub-problems) and in control knowledge about how to proceed with problem solving (which sub-problem to attack next and in which way to attack it). Especially analysis steps may rely again on domain and common sense knowledge - thus providing an interaction between problem solving (search) and knowledge modeling [9].

6 KNOWLEDGE MODELING AND REASONING IN KNOWLEDGE-BASED ENGINEERING

In Chapter 2 we outlined three typical aspects of human intelligence. This scheme is also applicable to engineering problem solving:

A) Engineering knowledge has to be used to (re-)formulate a problem in such a way that it can be solved. Due to the well-defined meaning of engineering knowledge this seems to be simpler than in many other domains. The problem will be formulated in a unique, consistent, and complete manner.

B) Meta-knowledge is especially needed in order to analyze the correlations between sub-problems, partial solutions, alternatives, conflicts, etc. (as discussed above and in the MOKA framework paper [9]). This seems to be less demanding than in many common sense areas where modal aspects like beliefs and probabilities have to be taken into account.

C) Patterns as used by humans seem to be a problem in KBS technology up to now. Pattern matching is an intrinsically parallel process that does not lend itself well to our sequential computing architectures and leads to high computational costs. This is complicated by the fact that exact matching is rarely called for, but similar patterns are needed → yet another problem that is not well understood.

The whole process may run in iterations on different abstraction levels, due to inconsistencies between partial solutions, etc. The interplay between synthesis and analysis which is typical for engineering problem solving will also result in multiple iterations between these three problem solving activities. Given a (sub-) problem, human experts will typically only generate one or a couple of solutions. Quite often, these solutions are those which correspond best to their past experience (also related to the famous problem of professional blindness). Only if there is an explicit need or demand they will generate more solutions. In many cases, there is no guarantee that all existing solutions will be found.

In order to circumvent these and other problems and in their own right, search techniques are applied in engineering problem solving. With the right problem formulation (A) and the meta-level capabilities (B) their usage has a realistic chance of being successful (as outlined in Chapter 3).

7 TRADITIONAL SEARCH METHODS

AI text books traditionally depict search as a question of search strategy: Given a certain problem, what is the best algorithm to traverse the search space, e.g. depth vs. breadth first vs. A* vs. Minimax/Alpha-Beta. However, this choice is often very easy given a fixed search space (what do problem states look like and how to change from one to the other) and its semantics (what each state means and what a goal state is). For example, it is trivial to chose between Minimax-like algorithms and say depth/breadth-first-type algorithms knowing that the semantics of the search space is an AND/OR graph. The availability of domain knowledge allows the use A*-like algorithms, without such knowledge blind searches have to be used. The goal distribution and search-space size will lead to a simple decision between depth-first and breadth-first algorithms. Of course, the multitude of sub-variants makes that choice somewhat more complicated than described here, but the important point is that given a fixed search space, these choices are relatively easy.

What makes high-performance hard to accomplish is the difficulty to find and exploit more than the initially obvious search-space properties. These properties follow from domain knowledge because they are only applicable to specific problems, trees, or subtrees or even just individual nodes. Most of the effort in developing high-performance AI-search applications is devoted to exactly this part of the problem solving process: gathering domain knowledge. This effort is well spent because the search-space properties and objective remain relatively constant from execution to execution of the problem solver.

During the last 2 decades domain-independent enhancements to generic search algorithms have been proposed in the literature that can boost performance by several orders of magnitude [5,8,10,…]. Recently, some efforts have been made to describe these enhancements in terms of search-space properties required to allow a specific search enhancement to result in a significant performance gain. For example, transposition tables are beneficial if the search space is represented as a graph, but the search algorithm assumes a tree. Transposition tables as a generic search enhancement, can be used in any of these cases and often result in speedups of several orders of magnitude because the search avoids revisiting large parts of the search space.

Understanding the value of a search enhancement as a function of search-space properties and search algorithm leads eventually to the definition of enabler functions. The enabler function of an enhancement contains the meta-knowledge about the search-space properties required for the enhancement to work, possibly even adjusting it by supplying it with performance-critical parameters, and a way to determine, if these properties are present. However, this is still a relatively new approach that has to prove its value beyond isolated application domains.

Domain knowledge can have two distinct, but related uses in search: state (or solution) evaluation and search strategy (often called “search control”). State evaluation is often used to derive search strategy. When doing so, assumptions are made about the search space. A classic example are direct search methods [7]: by evaluating the “neighbors” of a current state in the search space, the search “moves” the new state towards a local optimum.

Two underlying assumptions about the search space are made:

First, that there is a monotonic increase of the values between the previous current state and the new current state. In discrete domains this can often be guaranteed by visiting every state, at the expense of many states being visited. In continuous domains, a maximum step size should be chosen that minimizes the risk of violating this assumption, to improve efficiency. Second, moving in the direction indicated by any measure assumes that this measure is an overall trustworthy indicator as to where in the search space a global optimum can be found. Methods like simulated annealing are due to the fact that this assumption is not generally true because of the existence of local optima.

Usually, search spaces are too large to be traversed exhaustively. The overall performance of search depends on simply visiting the most relevant parts of the search space within the allotted time. In many domains, such as computer chess, hundreds of thousands of nodes are visited every second to ensure that nothing relevant has been missed. Often certain pieces of knowledge are consciously removed from the programs because the cost of computing it slows the program down by too much resulting in an overall performance loss. This tradeoff shows that quality and cost of knowledge are two important factors when considering the overall performance of a program using mainly search. Most importantly, it shows that search can substitute knowledge, or put more accurately: search is knowledge - even though only containing it implicitly in its results. The last leads to an even more surprising conclusion: The poor quality of the knowledge used by our programs makes search necessary and successful.

8 SEARCH IN ENGINEERING PROBLEM SOLVING

8.1 The Problem

Typically search-space descriptions in engineering domains are given in an implicit manner, usually with states and state transition rules. Engineers will setup such “generic” search spaces for application domains and dynamically add constraints to restrict the search or solution space or the objective function (in case of optimizations).

Often, searches are only sub-solvers, driven by a master-solver (possibly a human) that is continuously refining the problem definition and/or the objective function. For example, a problem description might be over-constrained and the solver fails to produce a solution. By relaxing certain constraints, the search space changes and the solver can be called again.

Once a solution is found, it is often subjected to an analysis that might produce new objective functions and/or search-space constraints. Several such iterations between synthesis and analysis are common. This dynamic adjustment of the search space and objective (function) is a major challenge for traditional search methods. With the search space changing, its properties are changing as well. Without fixed, or at least

reasonably stable, search-space properties, it is impossible to use specific, hard-coded search enhancements or knowledge in the search.

The result is a substantial loss in speed: fewer nodes of the search space can be visited within a given time limit. To make up for this loss, the quality of the knowledge has to improve. However, since strategic knowledge depends on the problems and these are changing, this knowledge has to be found, generated or at least adapted every time. Compromising in using only knowledge general enough to be applicable to the entire problem class is restricting the knowledge and thus will lead to substantial loss in performance. Generating the knowledge on the fly costs time as well and the tradeoff between knowledge gathering and knowledge use have to be considered to optimize the overall performance.

Traditionally, search domains, such as games, are discrete or at least easily discretisable domains. None of the problem parameters are inherently continuous. In engineering domains, this is not quite that simple. Often one can restrict certain values to a finite subset of then discrete values, but this "technicality" can prove to be a tricky problem. We will return to this problem below.

In engineering domains one more difficulty has to be faced: the cost of evaluating a state (or configuration). For example, to evaluate a vehicle layout, it might be necessary to include vibration analysis, a crash test analysis using FE methods, 3D geometric overlap testing, accessibility analysis, drag coefficient calculations and/or many other possible calculations of complex performance metrics. The cost and complexity of such an optimization function results in at least two major challenges:

1) Instead of visiting thousands or even millions of states in the search space only few can be analyzed. This is not only true for optimizations, but also for complicated consistency checks in CSP-like formulations. Note that this problem refers to the speed with which the domain knowledge can be analyzed. We call this the "SPEED PROBLEM".
2) In such complex search spaces, search strategy is hard to come by, even humans are often at a loss in such cases. We call this the "STRATEGY LACK".

Of course, these two problems in combination seriously inhibit the performance of the traditional approaches that rely on speed (fast evaluation) and good search control knowledge (for the search enhancements).

8.2 The Tackle

8.2.1 Overcoming the Speed Problem

A) To speed up the evaluation of one of the data points, approximations hold the most promise for dramatically reducing the computational cost. Starting with a rough but cheap initial value, the search can, with increased need for precision, increase the computational cost. One could increase the number of elements in a FE analysis, or the depth of the octree for geometrical overlap calculations. This requires algorithms that can be controlled in their computational cost. Such algorithms are known from 3D geometric interference testing [4], but many other engineering calculations can be executed with varying degree of accuracy/speed, such as FE analysis and vibration calculations.
B) A slightly different approximation approach by surrogate functions was proposed by [3]. Instead of decreasing the cost of each of the function calls, only few calls are made and results are used to construct a surrogate function that predicts where good exploration points can be found. At these carefully chosen points the expensive evaluation function is used, albeit hopefully less often.
C) It will be a matter of intelligently controlling the tradeoff between speed and accuracy that will ultimately decide how much system performance can be boosted without loosing solution quality. Especially comparing tradeoffs among different (sub-) calculations will be complicated, since every calculation should return results of similar accuracy, otherwise valuable cycles are wasted. The different sub-calculations are combined to form a total score of a state. This combination is usually a weighted sum, where a human expert gives the weights. If these weights allow for a subset of dominant features, then a technique called "lazy evaluation" can be used: Instead of executing all sub-calculations, first only the most dominant are called. With this preliminary value a test is performed to verify if the score is already so bad that adding the rest of the minor features, even under the most favorable conditions, cannot beat the currently best design (giving an upper or lower bounds to the solution quality).

8.2.2 Overcoming the Strategy Lack

As mentioned before, search strategy is knowledge about where good solutions are expected. Assuming we have only domain knowledge, meaning we can only evaluate states in the search space, we will have to use them to infer where good solutions lie. Information theoretically this is impossible without certain assumptions. The approach of surrogate function makes these assumptions obvious.

The most important assumption we will make is about the "smoothness" of the search space. We understand smoothness of the search space as the property that the gradient of the evaluations changes only "slowly". That is, the second (partial) "derivatives" of the evaluations are "small". This is not a rigorous definition, simply because engineering domains use non-derivable functions, such as discrete parameters or output values (such as thresholds). The assumption of derivability would violate the basic idea of this paper, to be true to the practical circumstances of engineering problems - and these are inherently non-derivable. Here is where we pay for discretising certain variables, as discussed above.

While it was necessary to keep the number of possible variable values to a manageable number, it also means that it is now difficult to find reliable gradient information, if needed. The absence of this information usually favors direct search methods.

A search strategy similar to hill climbing can be applied only if an assumption of smoothness is made. The smoother the search space is believed to be, the larger the step size to the neighbors can be chosen. If no knowledge about such smoothness is supplied, the algorithm has to gather information from which such knowledge can be derived.

Such a search algorithm could start out at random locations with small steps into each "direction" (variable changes) of the search space and gather statistics about the changes of the evaluation for each variable or combinations of variables. More sophisticated approaches might consider arbitrary change vectors (to be added to the current variable setting). It is important to avoid "overshooting", that is, step sizes too large for the smoothness of the search space that would lead to missing "topological" features in the search space, such as local extrema.

By maximizing the step size of the algorithm when traversing the search space, large numbers of evaluations can be saved. However crude this approach might appear to be, it is robust and can be applied to any domain with a minimum amount of prior knowledge.

9 EXAMPLE

As an example to briefly illustrate some of the problems discussed here, we introduce the truck chassis layout problem. For a given problem specification (wheel base, engine and transmission characteristics and a list of parts to be mounted to the frame) a legal, inexpensive and reliable design/layout with good vibration characteristics has to be found.

The quality of a chassis layout is a combination of a multitude of performance measures. These are: vibration analysis of the driveline and potentially the frame, physical interference of parts according to their location on the frame, adherence to legally enforced distances between parts with safety concerns, costs of auxiliary structures to connect parts, accessibility and serviceability of the individual components and rules of thumb representing best practices derived and used by engineers over the years.

The current solution is a static, modular rule-based system that is a nightmare to maintain, because of the complex interactions between the modules and the (redundant) distribution of the knowledge in the many different parts of the calculations. To derive legal and geometrically (physically) possible positions of parts, 2D layout approximations and a crude interval arithmetic are used. The reduction to 2D forces a conservative treatment, potentially eliminating some legal placements of parts from considerations. Also, the modular treatment cannot generally achieve global optima, since subsystems are considered pseudo independently. The current production system routinely fails to produce valid layouts and a human has to work out (part of) the solution by hand.

This optimization problem can be treated as a search problem where an optimal design is the goal. There are continuous parameters (location/lengths of parts), discrete parameters (parts with certain mount points and/or restricted lengths selection) and enumerated parameters (part selection from different venders or materials) that describe a design. The evaluation function is not differentiable, thus only direct search methods are of interest. No search strategy is available. Part geometries and characteristics change and thus cannot be hard coded. The vibration analysis of the driveline can be accomplished with different methods (simple 2D calculation or a complex 3D vector-mechanical analysis). The interference tests can be accomplished with different levels of accuracy/cost.

The problem is translated into an objective function representing the design objectives (or soft constraints) and physical laws (or hard constraints). The objective function is (currently) a (linearly) weighted sum of sub-terms which represent individual considerations:

$$F = W_{Overlap}(P_{Overlap}) + W_{Driveline}(P_{Driveline}) + \cdots + W_{Battery}(P_{Battery}) + W_{Component\,Cost}(P_{Component\,Cost})$$

The weight-functions W_i represent the relative importance of the individual terms *and* the adherence of those performance parameters P_i to engineering thresholds. $P_{Overlap}$ represents the volume of physical overlap of components. $W_{Overlap}$ induces a large, super-linearly increasing penalty to force this term to 0 to achieve a (physically) feasible design. Currently, it is the most expensive of our sub-functions, using roughly 55% of the total runtime of the system. $P_{Driveline}$ represents the overall penalty of the current driveline layout. It is internally composed of a weighted combination of different driveline-specific performance parameters (torsional resultant acceleration, inertial acceleration (coast and drive) and dynamic bearing loads), length-restriction penalties, inter-segment angle limits and relative driveline-segment length guidelines:

$$\begin{aligned} P_{Driveline} &= W_{Torsion}(P_{Torsion}) + W_{InertiaC}(P_{InertiaC}) \\ &+ W_{InertiaD}(P_{InertiaD}) + \sum_{i=1}^{n-1} W_{DBL}(P_{DBL}(i)) \\ &+ \sum_{i=1}^{n} W_{Length}(Segment(i)) + \sum_{i=1}^{n-1} W_{Angle}(Angle(i)) \\ &+ \sum_{i=1}^{n-1} W_{LenRatio}(Segment(i), Segment(i+1)) \end{aligned}$$

Since the engineers impose limits on all these driveline performance numbers, the functions W_i represent these restrictions. As an example, $W_{Torsion}$ could be implemented as follows: $W_{Torsion}(t) = w_{torsion} * \left(\frac{t}{tors\_upper}\right)^2$, where $w_{torsion}$ is a weight giving the relative importance of the torsional resultant acceleration penalty with respect to the other driveline performance parameters, and *tors_upper* is the maximal allowed torsional resultant acceleration. Similar functions can be created for lower and upper bound restrictions, or even more complex constraints.

Currently there exists an implementation to a simplified version of the problem using pattern search and a different objective function [16,19]. It is still too slow to solve the problem under reasonable assumptions for time resources: it

takes about 2 to 3 minutes to produce a good design. We are currently working on a solution where the engineering knowledge used is strictly separated from the optimization routine and the optimization routine can use different search strategies to find optima. A meta-optimizer is observing the behavior and performance of the optimization routine and changes it's input to improve its performance and ensure adherence to hard constraints (which is not guaranteed for the optimization alone). This is ongoing research.

10 CONCLUSIONS

Search can be used as a fundamental problem solving technique in Knowledge-Based Engineering. Generic search engines can be developed that use domain-specific knowledge to find solutions to complex engineering problems. The current challenge is to achieve efficiency close to the performance achieved in domain-dependent search applications where hand-tuned objective functions, search strategies and enhancements can be used. This performance gap can only be narrowed when search algorithms are empowered to derive search strategies automatically.

ACKNOWLEDGMENT

We would like to thank our colleagues in the DaimlerChrysler KBE Research Group in Berlin and the Freightliner Enterprise Product Documentation Department in Portland for their interest and stimulating discussion. One of us (RK) was partially funded by the European Commission within the MOKA Esprit project (No. 25 418).

REFERENCES

[1] V. Allis, 1988, "A knowledge-based Approach to Connect-Four. The Game is Solved: White Wins.", Master's thesis, Free University, Amsterdam, The Netherlands

[2] V. Allis, 1994, "Searching for Solutions in Games and Artificial Intelligence", PhD thesis, University of Limburg, The Netherlands

[3] A.J. Brooker, J.E. Dennis, Jr, P.D. Frank, D.B. Serafini, V. Torczon, M. W. Trosset, 1998, "A Rigorous Framework for Optimization of Expensive Functions by Surrogates", To appear

[4] J. Cagan, D. Degentesh and S. Yin, 1998, "A Simulated Annealing-Based Algorithm Using Hierarchical Models for General Three-Dimensional Component Layout", Computer Aided Design, 30(10), 781-790

[5] J. Culbertson and J. Schaeffer, 1998, "Pattern Databases", in Computational Intelligence, 14(4):318-334

[6] N. Guarino (ed.), 1998, Formal Ontology in information systems, IOS Press.

[7] R. Hooke and T.A.Jeeves, 1961, "Direct Search Solution of Numerical and Statistical Problems", Journal of the Association for Computing Machinery, 8(2):212-229

[8] A. Junghanns, 1999, "Pushing the Limits: New Developments in Single-Agent Search", PhD thesis, University of Alberta, Edmonton, Canada

[9] R. Klein, 2000, Knowledge modelling in design – the MOKA framework, in J.S. Gero (ed.), proceedings of the International AI in Design conference, Kluwer, to appear.

[10] R. Korf, 1985, "Macro-Operators: A Weak Method for Learning", Artificial Intelligence, 26(1):33-77

[11] D.B.Lenat and R.V. Guha, 1990, Building large knowledge-based systems, Representation and inference in the CYC project, Addison Wesley, 1990.

[12] G. Schreiber et al., 1999, Knowledge engineering and management – the CommonKADS methodology, MIT Press, Boston.

[13] V. Torczon, 1997, "On the Convergence of Pattern Search Algorithms", SIAM J. Optimization, 7(1), 1 - 25

[14] V. Torczon and M.W. Trosset, 1997, "From Evolutionary Operation to Parallel Direct Search: Pattern Search Algorithms for Numerical Optimization", In Computing Science and Statistics, Vol. 29

[15] V. Torczon and M.W. Trosset, 1998, "Using Approximations to Accelerate Engineering Design Optimizations", Proceedings of the 7th AIAA

[16] S. Yin, P. Hodges, J. Cagan and X. Li, 1999, "Layout of an Automobile Transmission Using Three-Dimensional Shapeable Components", Proceedings of DETC 99

[17] S. Yin and J. Cagan, 1997, "A Pattern Search-based Algorithm for Three-Dimensional Component Layout", Proceedings of DETC 98