Dynamic Case Creation and Expansion for Analogical Reasoning

Thomas A. Mostek, Kenneth D. Forbus, and Cara Meverden

Qualitative Reasoning Group, Northwestern University

Abstract

Most CBR systems rely on a fixed library of cases, where each case consists of a set of facts specified in advance. This paper describes techniques for *dynamically extracting* cases for analogical reasoning from general-purpose knowledge bases, and *dynamically expanding* them during the course of analogical reasoning. These techniques have several advantages: (1) Knowledge authoring is simplified, since facts can be added without regard to which case(s) they will be used in. (2) Reasoning is more efficient, since task constraints can be used during case extraction to focus on facts likely to be relevant. (3) Larger problems can be tackled, since cases can be dynamically expanded with more details during the matching process itself, rather than starting with completely detailed cases. We describe algorithms for case extraction and case expansion, including how a version of the Structure-Mapping Engine (SME) has been modified to incorporate this new matching technique. The utility of this technique is illustrated by results obtained with two large knowledge bases, created by other groups, and used to answer questions in the DARPA High-Performance Knowledge Base Crisis Management Challenge Problem.

Introduction

Analogical reasoning operates by comparing cases. In most case-based reasoning systems, cases are stored as named collections of facts in a memory (c.f. [15, 16]). Most CBR systems are designed for a specific range of problems, and this strategy can be effective for such tasks. However, it becomes problematic for creating systems that can tackle multiple types of problems, involving very large amounts of knowledge. An International Crisis Management Assistant, for example, would require substantial knowledge of the nations of the world and their history. How should this knowledge be organized into cases? For example, facts about Great Britain presumably appear in cases describing WWI, WWII, and Great Britain, as well as cases describing interactions of Great Britain with other countries and describing events that occur inside it. Knowing what to store where becomes a complex issue, leading to potential missed inferences and storage redundancies. Worse, different tasks demand different types of information. Reasoning about Great Britain's military options in response to a hypothetical threat is, for instance, unlikely to require knowledge of its livestock feeding practices, although such practices are very relevant in reasoning about its economic relationships with the rest of the European Union. The static organization of knowledge into cases, whose contents are crafted by human designers in advance, is unlikely to scale to this level of application system, let alone the human-like flexibility of common sense reasoning.

Considering what would be involved in scaling up to human-sized knowledge bases raises a second problem with case organization: Controlling level of detail. For example, a representation of the Persian Gulf War might be broken into three major phases: the invasion of Kuwait, Operation Desert Shield, and Operation Desert Storm. Each of these phases consists in turn of various events, which are often further decomposable, and so on. Similar examples abound in medicine, engineering, science, and business. A rich case library should describe complex systems and events at multiple levels of detail. Unfortunately, larger cases are more expensive to match: directly comparing two full cases containing thousands of propositions can easily blow out even today's large memories and render a system too slow to be usable. The ability to modulate the level of detail during matching seems essential to scaling up.

This paper proposes a new method for organizing and using case libraries in analogical reasoning. The idea is to store the facts of all cases in a general-purpose knowledge base, and automatically extract relevant subsets of knowledge for reasoning, based on task constraints. This leads to two techniques:

1. *Dynamic case extraction* extracts case contents from a knowledge base, given a *target entity* and a query about that entity.
2. *Dynamic case elaboration* expands a case during the matching process, adding more information to help the matcher decide between competing submatches.

These techniques have three advantages. First, they simplify knowledge authoring: Concrete, specific facts can be added without regard to which case (or cases) they are part of, since that decision will be made automatically. Second, reasoning with cases can be made more efficient: The contents of a case can be partially determined by the current task, thus eliminating irrelevant material from consideration. Third, larger cases can be handled. In a fixed-contents case memory, finding the right level of detail is a difficult tradeoff. Too little detail, and useful inferences will be missed. Too much detail, and the reasoning system bogs down. We show that dynamic case expansion enables us to handle detailed cases that, on the same system, lead to memory blowouts if matched directly.

The next section begins with a brief review of the relevant aspects of structure-mapping theory, SME, and MAC/FAC, the analogical reasoning approach and tools

we are using. Then we describe the methods we use for structuring cases in a knowledge base and extracting relevant aspects of them via KB queries. Dynamic case expansion during matching is discussed next. Empirical results obtained as part of the DARPA High Performance Knowledge Bases Crisis Management Challenge Problem follow, showing that these techniques work well with two different knowledge bases and case libraries, neither authored by us. Finally, we discuss related work and future plans.

Prelude: Cases and analogical matching

According to structure-mapping theory [11], an analogy match takes as input two structured representations (*base* and *target*) and produces as output a set of *mappings*. Each mapping consists of a set of *correspondences* that align items in the base with items in the target and a set of *candidate inferences*, which are surmises about the target made on the basis of the base representation plus the correspondences. The constraints that govern mappings, while originally motivated by psychological concerns [11], turn out to be equally important for the use of analogy in case-based reasoning, since they ensure that candidate inferences are well defined and that stronger arguments are preferred [12].

Two simulations based on structure-mapping are particularly relevant to this paper. The Structure-Mapping Engine (SME) [1,5,7] is a cognitive simulation of analogical matching. Given base and target descriptions, SME finds globally consistent interpretations via a local-to-global match process. SME begins by proposing correspondences, called *match hypotheses,* in parallel between statements in the base and target. Then, SME filters out structurally inconsistent match hypotheses. Mutually consistent collections of match hypotheses are gathered into global mappings using a greedy merge algorithm. An evaluation procedure based on the systematicity principle is used to compute the *structural evaluation* for each match hypothesis and mapping. These numerical estimates are used both to guide the merge process and as one component in the evaluation of an analogy. SME operates in polynomial time, and its results can be incrementally extended as new information arrives.

MAC/FAC is a two-stage model of similarity-based retrieval that is consistent with psychological constraints [6] and has been used in a fielded application [6]. The key insight of MAC/FAC is that memory contents should be filtered by an extremely cheap match that filters a potentially huge set of candidates, followed by a structural match (i.e., SME) to select the best from the handful of candidates found by the first stage. The extremely cheap match is based on *content vectors*, a representation computed from structured descriptions. Each dimension of a content vector represents the number of occurrences of a particular predicate in a description. For example, if a (tiny) description had three `BEFORE` statements and one `IMPLIES` statement, its content vector would be `((BEFORE 0.75)(IMPLIES 0.25))`. Content vectors are normalized to avoid size biases. Content vectors are useful cheap matchers because their dot product provides an estimate of the largest structural match that could be obtained between the two original structured descriptions. During the construction of the match hypothesis forest in SME, base and target items with identical predicates are hypothesized to match, which may in turn suggest other matches (e.g., entity matches, non-identical function matches). Thus the size of a match hypothesis forest for two structured descriptions is roughly correlated with the dot product of their corresponding content vectors. In this paper, we use content vectors as a cheap similarity metric. Every entity (and indeed every predicate) in the knowledge base has an associated content vector, derived from the set of statements in the KB that mentions that entity.

Dynamic case construction

Cases are about something. That something can be a specific entity (e.g., the United States) or an event (e.g., WWII). Depending on task, even abstract concepts can be the subject of comparison, e.g., comparing notions of justice across cultures. We assume that one can always identify a *seed* for a case, the entity that the case is about. Given a task `T`, the case for a seed is a subset of facts from the KB about seed that are relevant for `T`. The two issues that must be addressed are

1. What facts about a seed `S` are *relevant* for a given task `T`?
2. What *bounds* the subset of the KB to be included?

The set of facts about `S` that are relevant to `T` can be divided into two sets: Those that explicitly mention `S` and those that do not. (If the term `S` appears in fact `F`, we denote this via `Mentions(F,S)`.) Not every fact that mentions `S` is relevant: In reasoning about possible US responses to an economic crisis, it is very unlikely that the fact that George Washington was the first US president will be relevant. We assume that for each task `T`, a set of predicates `RP(T)` can be identified such that statements whose predicates are in `RP(T)` and mention `S` are relevant. For example, in the case of reasoning about economic interests, predicates such as `has-economic-interest` and `economic-action` are included in `RP(economic-interests)`. `RP` can be defined very broadly, excluding only predicates used for internal, bookkeeping statements, or very sharply, including only predicates relevant to a particular aspect of knowledge about a domain (e.g., economic versus political versus military). `RP` can also be defined via inference rather than via explicit enumeration.

Unfortunately, task constraints are often more complex than can be expressed in terms of simply filtering via categories of predicates. For example, when reasoning about options a country might have had in a situation based on a historical precedent, one wants to extract the relevant facts of the situation up to, but not after, the key event. By

doing this, the match against the historical precedent will yield candidate inferences that represent potential options for the new situation that are analogous to what occurred in the historical precedent. Such task constraints can be expressed in terms of filtering out facts that match some specific criterion. Since the criterion depends on the details of the task and the representations, we must settle for describing it abstractly. Consequently, we assume the existence of a procedure, `Filter?`, that takes two arguments, a fact and a task, and returns true if the given fact should be ignored.

We can use `RP` and `Filter?` to narrow in on the facts relevant to a task. Let the set `RM(S,T)` be the set of facts that mention `S`, whose predicate is in `RP(T)`, and which do not satisfy `Filter?` (i.e., they are relevant to the task). `RM(S,T)` constitutes the relevant facts for `T` that mention `S` explicitly. These are not necessarily all of the relevant facts, of course, since the background for these facts in turn may need to be considered.

The set of relevant facts that do not mention `S` explicitly is found by recursive expansion, based on the entities mentioned in `RM(S,T)`. Let `GT(`*`<expressions>`*`)` refer to the set of ground terms occurring in the statements `<expressions>`. The terms (i.e., entities, events, processes, etc.) in `GT(RM(S,T))` are the conceptual entities for which additional facts should be included, since constraints on these terms can affect conclusions drawn with `RM(S,T)`. Let the *basic relevant facts* `RB(S,T)` be defined as follows:

```
RB(S,T) = {f ∈ KB | f ∈RM(S,T)
           ∨ [predicate(f) ∈ RP(T)
                ∧ GT({f}) ⊆ GT(RM(S,T))] }
```

That is, `RB(S,T)` is the set of facts in `RM(S,T)`, plus the facts that mention only entities in `RM(S,T)`. The basic relevant set of facts can be expanded by recursively computing `RB(e,T)`, for every `e`∈`GT(RM(S,T))`, and taking their union.. Obviously, the scope of this expansion has to be limited, otherwise in a highly interconnected knowledge base, all the facts will be included in every case. We scope the expansion by having `Filter?` be more constrained on facts that don't mention S. Table 1 shows the different `Filter?` methods for the case denoting functions from [9], which range from no expansion (`minimal-case-fn`) to expanding all the sub-parts of the original case (`recursive-case-fn`). The appropriate definitions for what are internal, bookkeeping predicates (`book-keeping?`), causal relationships (`causal?`), part/whole relations (`subparts?`) and attributes (`attributes?`) will be specific to the particular KB. For example, in Cyc `isa` statements constitute attributes.

Input:
- An entity or expression `S` which the case will be about.
- A knowledge base `KB` and task T
- A procedure `Filter?` that encodes task-specific constraints (see text).

Procedure GenerateCase(S,T)

1. `RM(S,T)` ← {}
2. For all `f` ∈ `KB` s.t. `Mentions(f,S)`,
   2.1 If `predicate(f)`∈`RP(T)` ∧ ¬`Filter?(f,T)` then
   `RM(S,T)` ← `RM(S,T)` ∪ {`f`}
3. `RB(S,T)` ← `RM(S,T)`
4. For each `E` in `GT(RM(S,T))`
   4.1 `RB(S,T)` ← `RB(S,T)`∪`GenerateCase(E,T)`

Figure 1: GenerateCase algorithm

Function	Method
Minimal-case-fn(S)	Book-keeping?(f) or ¬mentions(f,S)
Case-fn(S)	Book-keeping?(f) or (¬mentions(f,S) and ¬Attribute?(f))
Event-case-fn(S)	Book-keeping?(f) or (¬mentions(f,S) and ¬(Attribute?(f) or causal?(f)))
Agent-case-fn(S)	Book-keeping?(f)
In-context-case-fn(S,C)	Book-keeping?(f) or GT(f) subparts(C) = ϕ or (¬mentions(f,S) and ¬Attribute?(f))
Recursive-case-fn(S)	Book-keeping?(f) or(GT(f) ⊄ GT(RM(S,T) and ¬mentions(f, {subparts(S)})

Table 1 – The semantics of `filter?` for the analogy ontology case functions

Dynamic case expansion

Complex cases typically have a hierarchical structure. Complex events have subevents, complex objects have parts, complex systems have subsystems, and complex devices have components. This hierarchical structure typically manifests itself in representations by things that are conceptual entities at one level being expanded into a collection of facts and entities when viewed at a finer level of detail. Matching can be made more efficient by exploiting this hierarchical structure. All matchers require time proportional to the size of the input descriptions. Starting with a high-level description, then incrementally refining the match by further exploring potentially corresponding parts, avoids considering many fruitless matches. For example, in comparing a person to a chimpanzee the rough match between their overall form invites a closer look at comparing their heads but not, say, the human's head to the chimpanzee's foot. Matching then becomes an incremental, iterative process, with the results of one stage of matching helping to guide the next. We call this process *dynamic case expansion*.

The ideas of the previous section provide the framework needed for dynamic case expansion. Given a comparison, the seed is chosen to be at an appropriate level of detail (i.e., Persian Gulf War versus Operation Desert Storm versus a particular sortie), corresponding to the highest level of abstraction required. Base and target cases are created, without recursing, and SME is used to create the forest of match hypotheses that describes how statements in these two cases might be aligned. As noted above, expansion takes place at entities, so it is potential matches (MHs) between entities that form the candidates for expansion. There are three criteria used for deciding

whether to expand an MH:
1. There must be at least one other competing MH for either the base or target entity in the original MH. Only those MH's whose score is close to the top scoring MH (currently 60%) are considered.
2. The content vector overlap between the two entities paired by the MH must be over some threshold (currently 0.4). This heuristic makes it more likely that expansion will give rise to new overlapping structure.
3. A task-specific procedure, `Expandable?`, which takes as arguments a candidate for expansion and the depth, and returns true if the candidate is worth expanding. For example, in reasoning about international crises it is typically only appropriate to expand events one level, whereas actors and goals are worth expanding deeper.

An MH is expanded by treating the entities it pairs as seeds for case extraction, as described in the previous section. The new facts for the entities are added to the appropriate descriptions (i.e., facts about the base entity are added to the base, and similarly for the target). Normally, when new statements are added to the base or target, SME's incremental match process extends the set of match hypotheses by considering the new base items against all of the target, and the new target items against all of the base. Our *focused match* algorithm modifies this by considering the new base items only against the new target items, thus avoiding hypothesizing local matches that are likely to be irrelevant. This process continues recursively, up to some depth bound (currently 2). The algorithm is described in detail in Figure 2.

Inputs:
- The base `B` and target `T` being compared
- The knowledge base `KB` from which `B` & `T` were drawn
- Procedures `Filter?` and `Expandable?` that encodes task-specific constraints (see text).
- An integer `MaxDepth` which limits expansion by depth
- A threshold `CVOverlap` that specifies the minimal content vector overlap (currently 0.4).
- The match hypothesis forest `MHS` created by the standard SME algorithm given initial `B`, `T`.

Context: This algorithm is executed immediately after the usual match hypothesis forest step in the incremental SME algorithm, and when finished, the rest of the SME algorithm proceeds as usual.

Procedure `CaseExpansion`
For each `MH` in `MHS`, `ExpandMHS(MHS, 0)`

Procedure `ExpandMHS(theMHS, depth)`
1. When `depth = MaxDepth`, return.
2. For each `MH` in `theMHS`
   a. Unless `Entity?(BaseItem(MH))`, skip.
   2.2 Unless `InCompetition?(MH, newMHS)`, skip.
   2.3 Unless `Expandable?(MH)`, skip.
   2.4 Unless `CVDotProduct(MH) > CVOverlap`, skip.
   2.5 `NewBase` ← `GenerateCase(BaseItem(MH),KB)`
   2.6 `NewTarget` ← `GenerateCase(TargetItem(MH),KB)`
   2.7 `NewMHS` ← `CreateMHS(NewBase,NewTarget)`
   2.8 `MHS` ← `MHS` ∪ `NewMHS`
   2.9 `ExpandMHS(NewMHS, depth+1)`

Procedure `CVDotProduct(MH)`
`ContentVector(BaseItem(MH))`
- `ContentVector(TargetItem(MH))`

Procedure `InCompetition? (MH1, MHS)`
1. For each `MH2` ∈`MHS`, such that `MH1` ≠ `MH2`
   1.1 Unless `BaseItem(MH1) = BaseItem(MH2)` or `TargetItem(MH1) = TargetItem(MH2)`, skip.
   1.2 If `CVDotProduct(MH2) > CVOverlap`, then return `True` from `InCompetition?`
2. Return `False` from `InCompetition?`

Figure 2: Dynamic case expansion algorithm

Importantly, this process is different from recursively calling SME on the cases created from the entities because global mappings are not created during expansion at any level. Structure-mapping theory tells us that large, systematic matches are preferred [11]. Global mappings between lower-level matches would not be sensitive to relations that occur at higher levels. By keeping all of the match hypotheses in the same forest, SME's constraint satisfaction mechanisms can combine evidence from all levels in creating its interpretations, which improves accuracy and enables interpretations to include all relevant levels of detail.

The worst-case complexity of the focused match algorithm is polynomial, assuming a fixed maximum depth limit for recursive expansion and assuming that the computation of RM and RP is polynomial. The latter are polynomial if implemented as lookup operations rather than inference steps; if they require inference, then the complexity of the inference machinery becomes a factor. The savings over uniform preexpansion come from two sources: (1) Many fewer match hypotheses are generated, saving storage and time, and (2) fewer match hypotheses means fewer things to consider when constructing global interpretations. These savings can be significant in practice, as the next section illustrates.

Empirical Results

In the DARPA High Performance Knowledge Bases program, the Crisis Management Challenge Problems focused on building knowledge bases and systems that could answer the kinds of queries that an analysts' assistant might provide. When reasoning about international crises, analysts commonly rely on analogy to analyze the present in terms of history (c.f. [17,18]). Consequently, a number of analogy queries were included in the tests, and we used the algorithms described here in providing analogical processing services for both teams in the evaluation.

TABLE 2: Crisis Management queries		Base			Target			Dynamic Creation +expansion		No Dynamic Expansion		No Dynamic Creation	
Team	Query	Start	Final	Max	Start	Final	Max	MHs	Seconds	MHs	Seconds	MHs	Seconds
SAIC	SQM226	777	883	2062	632	822	1312	2579	27	10634	346	36602	3000*
SAIC	TQE225	437	503	1057	777	863	2062	2030	32	11437	310	33114	2400*
SAIC	TQE226	777	1117	2062	239	507	507	3976	144	11699	259	18824	1493
Cyc	TQE225	299	721	721	192	1891	1891	9146	141	11614	821	11614	821
Cyc	SQM226	192	1891	1891	108	592	592	6319	112	9299	307	9299	307
SAIC	TQM226	240	324	493	632	729	1312	1470	12	3619	13	8933	123
Cyc	TQE226	192	1842	1891	34	168	168	1857	47	2704	46	2704	46
SAIC	TQF225a	184	274	274	234	486	486	1499	14	2122	5	2122	5
Cyc	TQM226	119	205	205	108	592	592	1318	8	1583	3	1583	3
Cyc	TQF225a	79	239	239	120	457	457	780	7	869	1	869	1
Cyc	TQF225b	118	129	129	423	440	440	665	4	665	1	665	1
SAIC	TQF225b	91	120	120	99	144	302	302	6	333	1	333	1

Examples of the analogy questions include

TQE225: How is the UN's mediation of the dispute between Iran and the Taliban in the 1998 Iranian-Taliban Crisis similar to the UN's mediation of the dispute between Iran and the GCC in the Y2 Scenario?

SQM226: Who/what is IRAN in Y2-SCENARIO-CONFLICT similar to in PERSIAN-GULF-WAR? How so, and how are they different?

Examining how the case creation and case expansion algorithms work on these problems is a good test for two reasons. First, the cases involved in these problems were often substantial, two orders of magnitude larger than many examples used in the analogy literature, and an order of magnitude larger than anything we had tackled previously. Second, our algorithms had to work with two independently developed knowledge bases, created by other research groups. (Since the evaluation was competitive, we were allowed to consult with both teams about how to improve their knowledge bases, but were not allowed to make extensions ourselves.)

Table 2 describes data from an experiment using queries from the Crisis Management challenge problem. The first two columns indicate what team's KB was used and the specific query. The next six columns show the initial size of the base or target, as found via `GenerateCase` (Start), how large it reached due to dynamic expansion (Final), and the size of the full case (Max). The results of dynamic case creation and expansion are shown in the next two columns, which indicate the size of the match hypothesis forest generated in SME (MHs) and the total run time (seconds). The final four columns provide data that help tease apart the relative contributions of dynamic case expansion versus creation. The No Dynamic Expansion column shows the amount of work done when the focused match algorithm is not used. In this condition, SME is being run on the largest cases found dynamically, but interactions between different aspects of the cases matches are considered, as opposed to only attempting matches between expansions of corresponding parts. The final pair of columns indicate the amount of work done if the full cases were compared. Runs marked with "*" indicate that the program hadn't completed by the time recorded. The task-specific settings of the algorithms used in these runs were as follows: For `RP`, only internal, bookkeeping predicates were excluded. For `Filter?`, depending on the query, either nothing was filtered out, or causal consequences of the seed were filtered out. `Expandable?` always expanded interests and was set so events were expanded first, and objects expanded at the last iteration. This reduced complexity in the scoping algorithm used in `Filter?`. The same parameter settings were used for both team's KB's; only the particular lists of predicates (e.g., what constituted a bookkeeping predicate, interest, or event) varied.

The table contents are ordered by the worst-case match hypothesis count. Several interesting properties can be seen in this table[1]. First, smaller cases are faster, and when sufficiently small, the complete case tends to be retrieved. Looking at the KB, the cases used in these queries are without substantial substructure, so this makes sense. Even on these smaller cases, some space savings occurs due to the focused nature of the matches used during case expansion, but the overhead of dynamic expansion makes the runtime slower. However, on larger cases, both significant time and space savings are found: up to an order of magnitude reduction in storage, and finishing in a reasonable time versus not finishing at all in the largest cases. The average storage savings over all examples is 75%, and the speedup over the entire set of queries is 4.6. The combination of significant speedups plus the ability to do examples that were impossible before is strong evidence for the utility of our techniques.

Related Work

In some systems cases are automatically generated by performance systems (c.f. , [1]) but most often cases are created manually, with the help of software tools (e.g., [15]). Although cases can be added to or modified by

[1] It may seem surprising that run time is not always a monotonic function of the number of MHs, but this falls directly out of the structure of the SME algorithm [7]

human authors, from the perspective of the reasoning system such case memories are static, since they are not being evolved during the course of reasoning. The practice of storing results of a problem solving session as new cases, while helping to expand a system's performance across multiple reasoning sessions, does not affect the structure of cases within a single reasoning session itself.

The closest previous work is Progressive Sapper [19], which combines spreading activation with progressive deepening to provide an anytime algorithm for retrieval. Unfortunately, like Sapper, it accumulates match hypotheses in long-term memory, which leads to an exponential growth over time. Thus it seems unlikely that this model would scale up to the size of knowledge bases that our system handles. Progressive Sapper also does not exploit the semantics of the domain and task in the way that we do, nor does it exploit the ongoing match in deciding how to expand a case.

Discussion

The traditional reliance of CBR on a libraries of fixed-structure cases has been useful in practice, but it is unclear that such techniques will scale to human-scale memories. The ability to dynamically extract cases from large-scale knowledge bases, combined with the ability to dynamically expand them during matching, supports the use of analogical reasoning with rich, relational representations drawn from large-scale, general-purpose knowledge bases. In addition to providing a fundamentally new capability, dynamic case expansion also provides more efficient matching on large descriptions, facilitating scale-up. The fact that these techniques succeed on multiple large-scale knowledge bases constructed by other research groups is strong evidence that these techniques are generally useful

A number of issues remain to be explored. As the structure of large knowledge bases becomes understood, it may be possible to have a stronger theory of what our algorithm currently uses as procedural parameters. Techniques from compositional modeling [6] might be generalized to automatically handle selection of initial perspective and level of detail. Finally, new possibilities for dynamic expansion open up when considering larger-scale systems: Suppose SME were run to completion with the most abstract level of match, with expansion taking place when a downstream system needed more detail about a particular aspect of a comparison. This could provide a useful generalization to Falkenhainer's map/analyze cycle [3].

Acknowledgements

This research was supported by the DARPA High Performance Knowledge Bases program and by the AI Program of the Office of Naval Research.

References

1. Blythe, J. & Veloso, M. (1997) Analogical replay for efficient conditional planning, *Proceedings of AAAI-97*, pages 668-673.
2. Cohen, P., Schrag, R., Jones, E., Pease, A., Lin, A., Starr, B., Gunning, D., & Burke, M. 1998. The DARPA High Performance Knowledge Bases Project. *AI Magazine*, Winter, 1998.
3. Falkenhainer, B. (1987). An examination of the third stage in the analogy process: Verification-based analogical learning. *Proceedings of IJCAI-87*, 260-263.
4. Falkenhainer, B., Forbus, K., & Gentner, D. (1986, August) The Structure-Mapping Engine. Proceedings of AAAI-86, Philadelphia, PA
5. Falkenhainer, B., Forbus, K., & Gentner, D. (1989) The Structure-Mapping Engine: Algorithm and examples. Artificial Intelligence, 41, pp 1-63.
6. Falkenhainer, B. & Forbus, K. "Compositional Modeling: Finding the Right Model for the Job", Artificial Intelligence, **51** (1-3), October, 1991.
7. Forbus, K., Ferguson, R. & Gentner, D. (1994) Incremental structure-mapping. *Proceedings of the Cognitive Science Society,* August.
8. Forbus, K., Gentner, D. & Law, K. (1995) MAC/FAC: A model of Similarity-based Retrieval. *Cognitive Science*, 19(2), April-June, pp 141-205.
9. *Anonymous authors*. An analogy ontology for integrating analogical processing and first-principles reasoning. *Submitted to AAAI-2000.*
10. Forbus, K.D., Whalley, P., Everett, J., Ureel, L., Brokowski, M., Baher, J. & Kuehne, S. (1999) CyclePad: An articulate virtual laboratory for engineering thermodynamics. *Artificial Intelligence*.
11. Gentner, D. (1983). Structure-mapping: A theoretical framework for analogy. Cognitive Science, 7, 155-170.
12. Gentner, D. (1989). The mechanisms of analogical learning. In S. Vosniadou & A. Ortony (Eds.), *Similarity and analogical reasoning* (pp. 199-241). London: Cambridge University Press. (Reprinted in *Knowledge acquisition and learning*, 1993, 673-694.)
13. Gentner, D., & Holyoak, K. J. (1997). Reasoning and learning by analogy: Introduction. *American Psychologist, 52,* 32-34.
14. Gentner, D., & Markman, A. B. (1997). Structure mapping in analogy and similarity. *American Psychologist, 52,* 45-56. (To be reprinted in *Mind readings: Introductory selections on cognitive science,* by P. Thagard, Ed., MIT Press)
15. Kolodner, J. L. (1994). Case-based reasoning. San Mateo, CA: Morgan Kaufmann Publishers.
16. Leake, D. (Ed.) 1996. *Case-Based Reasoning: Experiences, Lessons, and Future Directions*, MIT Press.
17. Neustad, R. & May, E. 1988. *Thinking in time: The uses of History for Decision Makers*. Free Press.
18. IET, Inc. and PSR Corp. 1999. HPKB Year 2 Crisis Management End-to-end Challenge Problem Specification. http://www.iet.com/Projects/HPKB/Y2/Y2-CM-CP.doc
19. Veale, T., & Keane, M. T. 1998. 'Just in Time' Analogical Mapping, An Iterative-Deepening Approach to Structure-Mapping. *Proceedings of ECAI'98, the Thirteenth European Conference on Artificial Intelligence*