Distributed Coaching for an Intelligent Learning Environment

Kenneth D. Forbus[1], John O. Everett[2], Leo Ureel[1],
Mike Brokowski[3], Julie Baher[1], Sven E. Kuehne[1]

Abstract

Several barriers hinder the widespread application of AI-based educational software. School and student machines are often underpowered, keeping software and case libraries updated can be difficult, and customization typically requires AI expertise. The widespread growth of Internet access, combined with appropriate AI technologies, enables the creation of *distributed coaches* that can help overcome these barriers. We describe a distributed coaching system for a deployed intelligent learning environment in engineering thermodynamics. Part of the coach resides on the student's computer, with the rest residing in a server accessed via email. The on-board coach handles common kinds of contradictions in student's assumptions and makes suggestions about parameter values based on its understanding of the teleology of the student's design, derived via Bayesian reasoning. The email coach provides additional analysis help and uses analogy for design coaching, providing step-by-step advice on how principles in a web-based library can be applied to a student's particular design. The distributed coach is currently undergoing field testing.

1. Introduction

Artificial Intelligence techniques have already proven themselves valuable in various types of educational software [1,2]. In education and training, there are never enough instructors to go around. Students often work at odd hours, on highly variable schedules, and, with the growing importance of distance learning, increasingly at remote sites. Computer-based coaches typically are not as good as the best human instructors, but they can be surprisingly valuable (c.f. [3]), and the ability to make them widely available at low cost makes them attractive. Typically such coaches are hard-wired into interactive learning environments that sit on a student's desktop. This leads to problems typical of updating and maintaining software. Worse, sophisticated coaches tend to require more memory and computational resources than many student and school computers can provide. For researchers trying to evaluate new kinds of educational software, the blessing of widespread computers becomes the curse of being unable to find out what students are actually doing with the software. Data gathering, even with cooperative instructors, can be quite difficult, and nearly impossible when a program is distributed freely via the Web.

Fortunately, the growth of widespread Internet access, combined with the right AI technologies, supports novel, distributed educational software systems that can overcome these problems. This paper describes such a system we have built for CyclePad, an intelligent learning environment for engineering thermodynamics. We begin by briefly reviewing CyclePad, focusing on the mix of AI technologies that make it work. We then outline the problems encountered in deploying it, and the distributed coaching architecture we developed to overcome these problems. We describe the on-board coaching next. We then discuss the email-based coaching, and highlight its use of cognitively-motivated analogical processing to provide case-based advice. Finally, we discuss experience with the system to date and plans for future work.

2. CyclePad

CyclePad is an *articulate virtual laboratory* (AVL) for learning engineering thermodynamics by design. Design tasks are highly motivating, and tie classroom learning to real-world concerns. Students using CyclePad can design power plants, refrigerators, engines, cryogenic systems, and other types of single-substance thermodynamic cycles. CyclePad's *conceptual CAD system* supports carrying out necessary calculations and making modeling assumptions, including both thermodynamic and economic analyses of a student's design. AVLs also include *coaches* that scaffold students, providing guidance in analysis and design.

CyclePad itself relies on several AI technologies:

- *Constraint propagation* is used to derive the consequences of student assumptions. Whereas conventional analysis software can make it hard for students to see how their assumptions are combined with the laws of thermodynamics to yield results, CyclePad's constraint propagator is organized to provide explanations and to reflect expert preferences in solutions. For example, CyclePad prefers values from equations over those from property table lookups when both are available because of the relative loss of accuracy with each property table calculation.
- *Logic-based truth maintenance* provides explanations of how consequences are derived from student assumptions. CyclePad provides a dynamically constructed hypertext explanation system based on the dependency network that highlights critical factors and suppresses uninformative details.

[1] Institute for the Learning Sciences, Northwestern University {forbus ureel jbaher skuehne } @ils.nwu.edu
[2] Xerox Palo Alto Research Center jeverett@parc.xerox.com
[3] Department of Mechanical Engineering, Northwestern University brokowski@nwu.edu

To appear in *Proceedings of QR98*, May 1998, Cape Cod.

These explanations help students gain insight into the application of thermodynamic laws, and are essential in tracking down physically inconsistent assumptions.

- *Qualitative representations* provide common sense "reality checks" of student assumptions. For example, substances cannot experience a drop in temperature across a heater. Simple ordinal constraints produce contradictions when student assumptions violate device models.
- *Compositional modeling* provides explicit representations of modeling assumptions. Instructors find an understanding of which modeling assumptions are necessary and correct is a key hurdle in learning thermodynamics. By explicitly representing modeling assumptions and their consequences, CyclePad helps students understand what assumptions make sense for particular components and what consequences they entail.

How these technologies are combined in CyclePad is described in [4].

3. Why Distributed Coaching?

CyclePad has been deployed experimentally in several universities for three years, and is currently used by over 180 students per year. Instructor and student feedback has been sufficiently positive that it is now publicly available for download on the Web. However, due to our web-based distribution mechanism, we only have partial knowledge of who is using it for what[4]. We would like more interaction with CyclePad end users to better gauge its educational impact.

Another difficulty arises because we have further development plans for CyclePad. Feedback from our users indicates that they would like more coaching facilities. However, they do not want to see memory requirements rise, and instructors who rely on it daily are adamant about keeping it stable.

For our part, we would like to gather more data about what students and instructors are doing with CyclePad for formative evaluation. We currently work with instructors from several remote sites to ensure robustness, but expanding that pool is difficult. For instance, 1997-98 classroom adoptions that we know about include a power plant course at Rutgers (New Jersey, USA) and a thermodynamics course in University of Queensland (Brisbane, Australia). As the number of sites continues to grow, travel budgets and time constraints preclude on-site data gathering.

[4] Our usage estimates are based on figures from our collaborators and random sampling. From September 1997 to January 1998, for example, we had 568 distinct downloads from 40 countries, making extensive follow-up impractical.

Our solution to these dilemmas is to make the coaching in CyclePad distributed. Although some lightweight coaching facilities are bundled with the software, the computationally-intensive coaching facilities are accessed via email. Through a simple dialog menu, a student uses CyclePad's integrated email facility to request help. The student's request and the associated design are sent to a server at our site running a coach, the *CyclePad Guru*. The Guru processes the student's request, sending a reply via email. The obvious disadvantages of this approach are that it requires students to have access to email, and that responses take more time than from an on-board coach. However, it does solve the problems our users raise: we can add complex new coaching facilities without increasing the memory footprint on their machines, or indeed, requiring any changes to their software at all. As a particular piece of coaching technology becomes solid enough, it can be migrated to the onboard coach as appropriate. It also facilitates our data collection: students get help with their work, in exchange for letting us examine their designs.

4. The Onboard Coach

Quick response to some common problems students encounter is important. Therefore we have added lightweight coaching facilities onboard which give advice for handling contradictions and for adjusting parameters.

4.1 Help with Contradictions

Students often have difficulty understanding why a set of assumptions is contradictory. CyclePad's truth-maintenance system includes a stack of contradiction handlers [5]. Each handler responds to a class of incorrect student assumptions. For example, the most common source of problems is choosing parameter values outside the property tables. The handler for this case presents a table-boundary diagram and a dot showing the location of the out-of-bounds value. The handler of last resort simply provides a hypertext dialog that enables students to explore the assumptions underlying the contradiction.

4.2 Teleology for Coaching

Many problems with cycle design are not apparent to students because their knowledge of cycles is so limited. For example, an experienced designer will note that low quality in the working fluid exiting a heat engine's turbine is likely to cause damage to the turbine blades and therefore attempt to adjust the system's parameters to increase the exit quality, or failing that, make a structural alteration to the cycle. To spot problems like this and understand how to fix them requires knowledge of how function relates to structure. For example, low exit quality is only a problem if the cycle is intended as a heat engine. A Carnot cycle

deliberately disregards the engineering challenges of expanding a saturated fluid through its turbine in order to provide a theoretical benchmark for ideal performance, and so by intention has low exit quality. In contrast, a turbine may also be used in a cryogenic cycle to cool the working fluid sufficiently to cause precipitation, because a resisted expansion results in a greater drop in the working fluid temperature than a throttled expansion, so in this situation we might be trying to achieve low quality. CyclePad now incorporates Everett's CARNOT teleological recognition system [6] to provide advice about values of system parameters based on its understanding of the intended function of the cycle.

CARNOT originally used dependency-directed search to infer function from structure. This was far too slow to be deployed. Also, it often produced a plethora of very similar solutions that varied only in minor details, making the principled choice of an interpretation to use for generating advice problematic. CARNOT now uses evidential rules and Bayesian inference to suggest plausible functional roles for each component in a student's cycle [7]. CARNOT's algorithm is quadratic in the size of the cycle, analyzing a 49 component cycle (far larger than any student has attempted, to our knowledge) in about two minutes on a midrange Pentium. CARNOT achieves broad coverage of the domain of single-substance, closed thermodynamic systems with 107 evidential rules.

The notion of *role* is crucial in CARNOT's construal of function. Each type of component in thermodynamic cycles can play between one and five functional roles. For example, a mixer may act as a simple way to join flows, as a heat-exchanger, or as a jet-ejector, in which a high-velocity jet of fluid entrains and compresses another inlet stream. The evidential rules provide evidence either for or against a particular role. The ability to suppress the likelihood of a role greatly enhances the expressive power of our representation. Each piece of evidence has a subjectively assigned likelihood[5], which is used to update the prior probability of each role for each component. The evidential reasoning is included in CyclePad's explanation system, so that students can find out why (and with what certainty) a particular role is believed and get an explanation of why other potential roles were rejected.

CyclePad's onboard Analysis Coach combines CARNOT's teleological inferences with *norms* to generate advice for adjusting parameters. A norm is a range for a component's parameter that is appropriate based on the component's functional role. For example, the temperature of the steam leaving a Rankine cycle boiler typically falls in the range of 300-600ºC. Lower temperatures result in inadequate efficiency whereas higher temperatures require uneconomically expensive materials in the downstream components. Likewise there is a normal range of pressures. Our knowledge base currently contains eighteen norms, between two and six per component depending on the number of potential roles for that component. When the Analysis Coach is invoked, CARNOT infers the teleology of the cycle. The functional roles assigned to each component are used to retrieve applicable norms, which are checked against known parameter values. Any violations or suggestions are noted using CyclePad's explanation system, providing explanatory text associated with each norm. In addition to being used to provide onboard advice, CARNOT's teleological representations also play a critical role in design coaching (see Section 5.3).

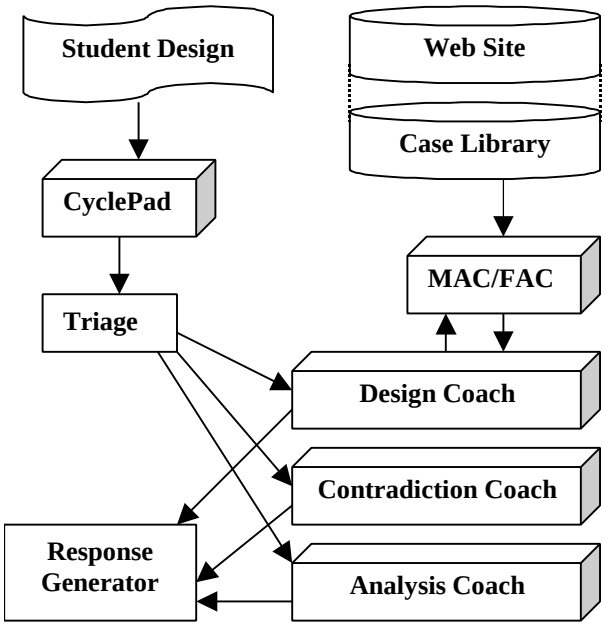

Figure 1: Information flow in the CyclePad Guru

5. The CyclePad Guru

The CyclePad Guru is part of a RoboTA agent colony [8], a TA agent that provides help for CyclePad users via email. The email dialog in CyclePad offers five choices: Students can turn in an assignment, ask for help with their analysis, ask for help figuring out a contradiction, ask for help in improving their design, or report a bug. When requesting design improvement help, the extra information is filled out via pull-down menus whose contents are based on the student's design (i.e., *<increase/decrease>*

[5] Subjective assignment of these values turns out to be straightforward for a domain expert, and the introduction of significant amounts of noise into these estimates does not materially affect the outcome.

the <*parameter*> of <*device/cycle*>). There is also the ability to add free-form comments, but these are not used by the Guru. Bug reports are passed on to the developers and turned-in assignments are currently ignored (but see Section 7). For all other requests, the Guru first does triage by using CyclePad to analyze the student's design. For example, if the student requests design help but the analysis is incomplete, the Guru provides help on finishing the analysis instead. Any discrepancy between the help requested and that provided is noted in the reply. The information flow within the CyclePad Guru is illustrated in Figure 1.

5.1 Analysis Help

Students analyze their designs by making modeling assumptions and exploring choices for parameter values. The complexity of even medium-scale cycles often makes it hard to know what to do next.

The onboard Analysis Coach is the student's first resort, providing advice linked to the intended purpose of the cycle. The Guru provides complementary assistance with strategies for analyzing cycles. The Guru has a domain-specific expert model of how to nudge students, based on observing our instructor-collaborators. First, it checks to see if a working fluid has been chosen, and if not, advises that as a good first step. Second, it examines the design to see what aspects remain unknown. It then presents a list of questions that the student should consider in thinking about the design (i.e., if not all modeling assumptions have been made, it suggests doing so). Since users often miss useful features in software, the Guru also runs the Analysis Coach on the student's design to see if it can provide advice, and if so includes instructions for using it. If none of these strategies is applicable, it responds with general canned text[6].

5.2 Contradiction Help

Since we have already built into CyclePad handlers for all of the common causes of contradictions that we know about, the Guru currently responds with a canned text about strategies for finding contradictions. Our distributed coaching approach will help us gather data about other contradictions arising in student use, and we will prototype handlers for new contradictions by incorporating them into the Guru first. Once they prove their worth and stability, such handlers will be moved onboard as appropriate.

5.3 Design Help

For now, we are limiting design help to the specific case of improving some quantitative aspect of the system. Of course, engineering design is complex, since it involves tradeoffs in both thermodynamics and economics that may require structural as well as numerical decisions. Our goal in giving design advice is to nudge students in useful directions so that they will learn, rather than to solve the problem for them. Consequently, we provide plausible specific suggestions, but do not attempt to validate these suggestions in the students' context. (Understanding why a suggestion will or will not work in a particular circumstance is an important learning experience.) Design coaching is described in the next section.

5.4 Coaching via Analogy

Advice for design requests is generated by a case-based coach, using cognitively-motivated analogical processing techniques [9]. Case-based coaching (c.f. [9,10]) is useful in educational software because it helps students tie their work to real-world examples. For that reason, case libraries for education tend to be media-heavy. Such systems have relied almost exclusively on hand-generated representations of cases. Cases often consist solely of user-interpretable media (e.g., videos) with the only formal representations being feature-based descriptors used for indexing. Cases are typically encoded and woven into a case library by hand. This lack of rich formal representations (e.g., proofs or causal arguments) limits the ability of a coach to show just how the principles explained in the case could be applied to a student's situation.

In the CyclePad Guru, we overcome these limitations in two ways. First, our cases are generated automatically from instructor input by a *case compiler* that uses CyclePad to build the necessary representations. Second, we use analogical processing techniques, drawn from Cognitive Science research, that can handle rich, structured representations. In particular, we use MAC/FAC, a model of similarity-based reminding [11] to retrieve cases relevant to a student's design. We start by describing how we retrieve cases and generate advice from them, then summarize how the case compiler works, and discuss some properties of our case library.

5.4.1 Retrieving cases and generating advice

MAC/FAC produces remindings in a two-stage process. The first stage (MAC) is a computationally efficient filter that selects from a large case memory a handful of cases for further processing. MAC uses a specialized feature vector that is automatically constructed from the structured representations in a case memory. The dot product of these vectors is an estimate of the size of match that the second stage will produce. The second stage uses the Structure-Mapping Engine (SME) [12], an analogical matcher based on Gentner's structure-mapping theory [13]. SME compares each case produced by MAC to the student's design and returns the best structural match,

[6] Hence the name "Guru."

plus one or two others, if close, as remindings. When SME compares two descriptions, it produces one or two mappings that consist of *correspondences* linking particular items in the student's design to the case, and *candidate inferences* that are statements in the case that may be transferable to the student's design. These inferences are used to generate specific advice about how the case can be applied to the student's design. While SME and MAC/FAC have been tested in a variety of cognitive simulation studies (c.f. [14,15]) to our knowledge this is their first application in a system used routinely by a large community.

```
Suggestions for <Desc WM of Rankine Cycle>:
Suggestion <Use INCREASE-RANKINE-BOILER-T>:
   1 step
   support=.085 extrapolation = 0.66
   normalized = 0.45 overlap = .408
   combined = .944
<Mapping 153 Candidate Inferences>
(BOILER htr1)
(CONDENSER clr1)
(IMPLIES (AND (TURBINE tur1 s2 s3)
              (HEATER htr1 s1 s2))
   (APPLICABLE (:SKOLEM :dsn-tr)))
(TRANSFORMATION-OF (:SKOLEM :dsn-tr)
      (STEPS (ASSIGN (T s2) (:SKOLEM :+))))
Suggestion <Use REHEAT-RANKINE-CYCLE>:
   16 steps
   support=0.03 extrapolation = .846
   normalized = .404 overlap = .134
   combined = .567
<Mapping 172 Candidate Inferences>
(BOILER htr1)
(CONDENSER clr1)
(IMPLIES (AND (TURBINE tur1 s2 s3)
              (COOLER clr1 s3 s4))
   (APPLICABLE (:SKOLEM :dsn-tr)))
(TRANSFORMATION-OF (:SKOLEM :dsn-tr)
   (STEPS (DISCONNECT (OUT tur1) (IN clr1) s3)
      (INSERT-DEVICE (:SKOLEM heater)
                     (:SKOLEM htr2))
      (CONNECT (OUT tur1) (IN (:SKOLEM htr2))
               (:SKOLEM s5))
      (INSERT-DEVICE (:SKOLEM turbine)
                     (:SKOLEM tur2))
      (CONNECT (OUT (:SKOLEM htr2))
               (IN (:SKOLEM tur2))
               (:SKOLEM s6))
      (CONNECT (OUT (:SKOLEM tur2)) (IN clr1)
               (:SKOLEM s7))
      (INVOKE-ASN (SATURATED (:SKOLEM s5)))
      (ASSIGN (DRYNESS (:SKOLEM s5))
              (:SKOLEM 1.0))
      (INVOKE-ASN (REHEATER (:SKOLEM htr2)))
      (INVOKE-ASN (ISOBARIC (:SKOLEM htr2)))
      (INVOKE-ASN (MATERIAL-OF (:SKOLEM htr2)
                  (:SKOLEM molybdenum)))
      (INVOKE-ASN (FUEL-OF (:SKOLEM htr2)
                  (:SKOLEM natural-gas)))
      (INVOKE-ASN (ISENTROPIC (:SKOLEM tur2)))
      (INVOKE-ASN (MATERIAL-OF (:SKOLEM tur2)
                  (:SKOLEM molybdenum)))
      (INVOKE-ASN (SATURATED (:SKOLEM s7)))
      (ASSIGN (DRYNESS (:SKOLEM s7))
                       (:SKOLEM 1))))
```

Figure 2: Candidate inferences for the cases retrieved via MAC/FAC that are turned into suggestions

When the Guru determines that the student's request for design advice is reasonable (i.e., the analysis has been completed and the design assumptions are non-contradictory), it uses the structural and teleological aspects of its representation of the student's design as a probe to MAC/FAC to generate candidate remindings. The numerical aspects of the description are not used for retrieval because we found that level of information to be basically irrelevant for this task. A case includes a description of a design, a problem with that design, and a transformation that modifies the original design in a way that solves the original problem. Each case that MAC/FAC is reminded of has, as part of that reminding, an analogical match between the student's design and that case. Since SME can generate multiple construals of a comparison (e.g., a plan to improve efficiency by increasing turbine inlet temperature is applicable in three different ways to a design that has three turbines), each reminding can generate several suggestions. Recall that a candidate inference of a mapping is a statement in the base (here, the case) that is suggested by the correspondences of the mappings as possibly holding in the target (here, the student's design). Candidate inferences are the source of advice. Figure 2 shows the candidate inferences when the Guru is reminded of reheat given a Rankine Cycle.

Suggestions are filtered for relevance in two ways. First, the candidate inferences must include the case's design transformation – otherwise, there is no advice to give. Second, the candidate inferences must include a statement of the form

```
(implies <structural/functional
          properties of cycle>
         (applicable <plan of case>))
```

Each case is guaranteed to include a statement of this form (see below), and the antecedents are exactly those things that must be true for the case's transformation to make sense. For example, neither of the cases retrieved in Figure 1 would be relevant for cycles lacking turbines. Therefore, a suggestion that does not include a candidate inference of this form, and correspondences for each of the antecedents of this inference, cannot be applied to the student's situation.

Next, the suggestions are prioritized according to the complexity of the transformation they suggest (with simpler transformations being preferred) and the structural quality of the candidate inference [16]. Finally, at most two suggestions are selected to serve as the basis for design advice. Limiting the advice to two suggestions prevents students from being overloaded with advice.

Figure 3 shows the advice generated from the candidate inferences in Figure 2. The advice generator splits out

structural transformations from other suggestions, keeping other assumptions separate as advice that may or may not be relevant to the student's particular situation. (Thinking about which of these suggestions is relevant is good exercise for the student. For instance, Figure 2 includes a suggestion to specify that certain devices be made of molybdenum, which a student should recognize as unusual and expensive.) If the advice is purely in terms of parameter changes, qualitative descriptions of relative change are used in the plan and to generate advice. The Guru's advice includes a URL describing the general principles behind the design transformation, in addition to the specific instructions on how to apply it to their situation.

```
I have 2 suggestions.
==========================================
Suggestion #1
Your problem reminds me of a method: increasing
boiler temperature in a Rankine cycle.  Increas-
ing the boiler temperature increases the effi-
ciency of the cycle.
You can find out more about this at
http://www.qrg.ils.nwu.edu/thermo/design-
library/turank.htm.
Here is how you might apply this to your design:
1. Increase T(S2).
==========================================
Suggestion #2
Your problem reminds me of a method: reheat in a
Rankine cycle.  Reheat adds another heater and
another turbine.  The second heater, a reheater,
heats up the working fluid at the turbine
outlet, and the second turbine extracts yet more
work from that.  This increases efficiency
because more heat is being added when the steam
is still at a reasonably high temperature.
You can find out more about this at
http://www.qrg.ils.nwu.edu/thermo/design-
library/reheat.htm.

Here is how you might do this with your design:

1. Disconnect the outlet of TUR1 from the inlet
of CLR1.
2. Create a new heater, which we'll call HTR2.
3. Connect the outlet of TUR1 to the inlet of
HTR2.  Let's refer to the properties of the
working fluid there as S5.
4. Create a new turbine, which we'll call TUR2.
5. Connect the outlet of HTR2 to the inlet of
TUR2.  Let's refer to the properties of the
working fluid there as S6.
6. Connect the outlet of TUR2 to the inlet of
CLR1.  Let's refer to the properties of the
working fluid there as S7.

You might find the following assumptions relevant
or useful:

1. Assume that the working fluid at S5 is
saturated.
2. Assume quality(S5) = 1.0000[0-1]
3. Assume that HTR2 is a reheater.
4. Assume that HTR2 works isobarically.
5. Assume that HTR2 is made of molybdenum.
6. Assume that HTR2 burns natural-gas.
7. Assume that TUR2 works isentropically.
8. Assume that TUR2 is made of molybdenum.
9. Assume that the working fluid at S7 is
saturated.
10. Assume quality(S7) = 1.0000[0-1]
==========================================
```

Figure 3: Design advice from the Guru

5.4.2 Automatic compilation of cases

New cases in the Guru's design library are automatically generated by a *case compiler*. To add a case, instructors provide two snapshots of a CyclePad design, one before and one after their transformation. They also specify the goals of the transformation, in terms of changes in parameter values (i.e., what parameters must have increased or decreased), some strings to be used in templates, and a URL pointing to a detailed rationale for that case. While we insist that the web page for the case include an explanation of the case, this explanation is in natural language: case authors only need to be thermodynamics experts, not AI experts. The case compiler uses CyclePad to analyze the before and after design snapshot. It uses a record of user actions stored internally with each dumped design to construct the description of the transformation that leads from one to the other, and augments the case description with this plan, the problems it is intended to solve, and the applicability condition described above. (It also checks to ensure that the transformation actually achieves the claimed goals, since even experts can make mistakes.) Adding the new case to the MAC/FAC memory is trivial, since no indexing is required: The structured representations needed to support reasoning also suffice for retrieval.

5.4.3 The case library

Our case library currently consists of 14 cases, averaging 74 expressions involving 23 entities each. Retrieval and advice generation is very quick: less than five seconds on the average, with no more than six seconds at most, on a 200 MHz Pentium Pro. This performance comes from two factors. First, the MAC stage provides significant filtering, with only two or three cases proposed for processing by SME each time. Second, SME now uses a polynomial-time greedy algorithm in its merge step, making its overall complexity quadratic in the size of descriptions compared [12].

The potential value of a distributed coach becomes especially apparent when considering the issue of extending and maintaining a case library. A large, rich case library with lots of associated media (e.g., pictures of the real physical systems corresponding to the CyclePad design) is probably best treated as a network resource, rather than installed on each student machine. Instructors and thermodynamics experts can author new cases with nothing more than CyclePad plus an HTML editor, since CyclePad and our case compiler take care of generating the formal representations, and MAC/FAC handles retrieval. We are

forming an editorial board for the web-based design library, to ensure quality control, and encouraging submissions from CyclePad experts worldwide, much in the manner of the Eureka community-maintained database of service tips [17] developed at Xerox PARC.

6. Deployment

At this writing (April 1998) RoboTA and the CyclePad Guru have undergone in-house testing for several weeks (e.g., bombardment with large numbers of email requests, odd requests, etc.) and have been on-line almost continuously for the last four months. The contents of the case library are evolving and expanding, as we gain experience with the system, and we expect this process to continue for some time. However, the system is already robust enough that we have now made the distributed coaching version of CyclePad publicly available via the Web on an experimental basis. The CyclePad Guru is fast enough that we believe our current configuration (200MHz Pentium Pro for the Guru) will handle the volume of requests from our collaborators' students and others. If server swamping becomes a problem, the RoboTA architecture is designed to support adding additional agents on extra CPUs to handle the load.

7. Discussion

Successful AI applications in education have typically used on-board tutors and coaches to supply advice and guidance. We believe that the widespread availability of Internet access facilitates the deployment of even more sophisticated coaching, by allowing the use of distributed coaches. We have described the distributed coach we have built for CyclePad, adding Bayesian inference and analogical processing to the mix of constraint propagation, qualitative physics, logic-based truth maintenance, and compositional modeling that CyclePad already uses. The use of cognitively-motivated analogical techniques in educational software systems, rather than the usual feature-based techniques typically employed in case-based coaching, enables us to directly employ structured representations. This means that cases can be automatically compiled from CyclePad descriptions and other instructor-supplied materials, automatically retrieved as appropriate, and used to generate step-by-step advice on how the principle embodied in a case can be applied to the student's problem. The distributed coach enables us to extend and experiment with new coaching strategies and methods, without increasing memory requirements or creating instability for our users. It also enables us to gather the data we need for the educational component of the research and formative evaluations of the software

We are also working on an additional incentive for instructors to send us data: helping them with grading. We are nearly ready to deploy a grading support system to the CyclePad/RoboTA system, designed with extensive input from our instructor-collaborators. It works like this: instructors use CyclePad to author assignments, using an additional wizard-style interface that enables them to express constraints on the assignment. Examples of constraints include requiring that particular substances be used as working fluids, establishing minimum and maximum criteria for cycle parameters, and restricting the kinds of parameters about which legitimate assumptions can be made (i.e., you can't create a cycle of a particular efficiency just by assuming that it has that efficiency). Students will then hand in their assignments by emailing them to RoboTA, which will use the assignment's constraints to check the student's work. The results will be provided to instructors by email or via a private web site, as they choose. The grading support system will not assign grades – that is the province of the instructor – but it will ensure that a student's design is correctly analyzed and determine how well it meets the specification of the assignment. This should give instructors more time to focus on providing students with the higher-level feedback usually made impractical by the time-consuming nature of checking numerical accuracy in assignments (e.g., how elegant and creative are students designs? Do students repeat the same mistakes or make more interesting ones?).

We believe this form of *learner support system* will become a common pattern for educational practice. By opening up the architecture, instructors can contribute cases, assignments, and other materials customized to meet their needs. Distributed coaches could provide extra encouragement for the formation of learning communities, by providing opportunities for participants to author materials that are automatically woven into the advice given to students. Instead of just browsing, AI techniques could enable software to help bring participants together, based on shared interests.

Acknowledgements

This research is supported by the Applications of Advanced Technology program of the National Science Foundation and by the Cognitive Science and Artificial Intelligence Programs of the Office of Naval Research. We thank Peter Whalley (University of Oxford), Bob Wu and Sheila Palmer (United States Naval Academy), and David Mintzer and Siavash Sohrab (Northwestern University), for providing thermodynamics expertise, feedback on our software, and for letting us try things out with them and their students.

References

1 Woolf, B. 1991. Representing, acquiring, and reasoning about tutoring knowledge. In J. W. P. C. L. R. H. Burns (Ed.), *Intelligent Tutoring Systems* . Hillsdale, NJ: Erlbaum.

2 Lesgold, L., Bunzo & Eggan. 1992. SHERLOCK: A Coached Practice Environment. In R. W. C. Jill H. Larkin (Ed.), *Computer-Assisted Instruction and Intelligent Tutoring Systems: Shared Goals and Complementary Approaches*. Hillsdale, N.J.: Lawrence Erlbaum Associates.

3 Koedinger, K. R., Anderson, J.R., Hadley, W.H., & Mark, M. A. 1997. Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education*, 8, 30-43.

4 Forbus, K. and Whalley, P. 1994 Using qualitative physics to build articulate software for thermodynamics education. *Proceedings of AAAI-94*, Seattle.

5 Forbus, K. and de Kleer, J., 1993. *Building Problem Solvers*, MIT Press.

6 Everett, J. O. 1995. A Theory of Mapping from Structure to Function Applied to Engineering Domains. 14th International Joint Conference on Artificial Intelligence, Montreal, Morgan Kaufmann.

7 Everett, J.O. 1997. *Topological Inference of Teleology: Deriving Function from Structure via Evidential Reasoning* Doctoral Dissertation, Computer Science Department, Northwestern University.

8 Forbus, K. and Kuehne, S. RoboTA: An agent colony architecture for supporting education. 1998. *Proceedings of Agents 98*.

9 Schank, R. and Cleary, C. 1994. *Engines for Education.* Erlbaum.

10 Leake, D. (Ed.) 1996. *Case-Based Reasoning: Experiences, Lessons, and Future Directions*, MIT Press.

11 Forbus, K. D., Gentner, D., & Law. 1995. MAC/FAC: A model of similarity-based retrieval. Cognitive Science, 19(2), 141-205.

12 Forbus, K. D., Ferguson, R. W., and Gentner, D. 1994. Incremental Structure Mapping. In Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society. Hillsdale, NJ: Erlbaum.

13 Gentner, D. 1983. Structure-mapping: a theoretical framework for analogy. Cognitive Science, 23, 155-170.

14 Gentner, D., Falkenhainer, B., & Skorstad, J. 1987. Metaphor: The good, the bad and the ugly. In Proceedings of the Third Conference on Theoretical Issues in Natural Language Processing, Las Cruces, New Mexico.

15 Gentner, D., Rattermann, M.J., Markman, A.B., & Kotovsky, L. 1995. Two forces in the development of relational structure. In T. Simon & G. Halford (Eds) Developing cognitive competence: New approaches to process modeling. Hillsdale, NJ: Erlbaum.

16 Forbus, K., Everett, J., Gentner, D., and Wu, M. 1997. Towards a computational model of evaluating and using analogical inference. *Proceedings of CogSci97*, Erlbaum.

17 Bell, D.G., Bobrow, D.G., Raiman, O., and Shirley, M.H., 1996. "Dynamic Documents and Situated Processes: Building on local knowledge in field service," IPIC'96, The International Working Conference on Integration of Enterprise Information and Processes, "Rethinking Documents", Cambridge, MA.

To appear in *Proceedings of QR98*, May 1998, Cape Cod.