Artificial Intelligence 114 (1999) 297–347

Artificial
Intelligence

www.elsevier.com/locate/artint

CyclePad:
An articulate virtual laboratory for engineering thermodynamics

Kenneth D. Forbus [a,*], Peter B. Whalley [b], John O. Everett [c], Leo Ureel [d],
Mike Brokowski [e], Julie Baher [f], Sven E. Kuehne [a]

[a] *Qualitative Reasoning Group and Computer Science Department, Northwestern University*
[b] *Department of Engineering Sciences, University of Oxford*
[c] *Palo Alto Research Center, Xerox Cooperation, 3333 Coyote Hill Road, Palo Alto, CA 94304, USA*
[d] *Qualitative Reasoning Group and Institute for the Learning Sciences, Northwestern University*
[e] *Qualitative Reasoning Group and Department of Mechanical Engineering, Northwestern University*
[f] *Qualitative Reasoning Group and School of Education and Social Policy, Northwestern University*

Received 26 January 1999

Abstract

One of the original motivations for research in qualitative physics was the development of intelligent tutoring systems and learning environments for physical domains and complex systems. This article demonstrates how a synergistic combination of qualitative reasoning and other AI techniques can be used to create an intelligent learning environment for students learning to analyze and design thermodynamic cycles. Pedagogically this problem is important because thermodynamic cycles express the key properties of systems which interconvert work and heat, such as power plants, propulsion systems, refrigerators, and heat pumps, and the study of thermodynamic cycles occupies a major portion of an engineering student's training in thermodynamics. This article describes CyclePad, a fully implemented *articulate virtual laboratory* that captures a substantial fraction of the knowledge in an introductory thermodynamics textbook and provides explanations of calculations and coaching support for students who are learning the principles of such cycles. CyclePad employs a *distributed coaching model*, where a combination of on-board facilities and a server-based coach accessed via email provide help for students, using a combination of teleological and case-based reasoning. CyclePad is a fielded system, in routine use in classrooms scattered all over the world. We analyze the combination of ideas that made CyclePad possible and comment on some lessons learned about the utility of various AI techniques based on our experience in fielding CyclePad.

---

[*] Corresponding author. Email: forbus@ils.nwu.edu.

PII: S0004-3702(99)00080-6

Keywords: Qualitative reasoning; AI and education; Engineering thermodynamics; Compositional modeling; Case-based coaching; Functional reasoning; Intelligent learning environments; Virtual laboratories

1. Introduction

One of the central motivations for research into qualitative physics has been its potential for the construction of intelligent learning environments. Qualitative reasoning provides computational accounts of human reasoning about the physical world, ranging from what the person on the street knows to the extensive expertise of scientists and engineers. Thus qualitative reasoning should provide representation languages and reasoning techniques that can be applied to helping people make the transition from novice to expert reasoning about physical systems. Indeed, some of the earliest work in the field was directly aimed at instructional problems (e.g., [9,25]). Over the last decade there have been several important efforts aimed at using qualitative physics to help teach diagnosis, troubleshooting, and operation of complex physical systems (e.g., [29,35,44]). Unfortunately, little effort has been focused on using qualitative physics in classroom settings, to help undergraduates learn principles of a domain (one rare exception is [40]).

In this paper, we describe CyclePad, an *articulate virtual laboratory* for engineering thermodynamics that is currently being used regularly by college engineering students and others. We start in Section 2 by describing the idea of articulate virtual laboratories, provide a brief overview of engineering thermodynamics, and describe the software from a user's perspective. Section 3 summarizes the architecture of CyclePad. Section 4 describes CyclePad's knowledge base, highlighting the kinds of knowledge in it and how it is represented. Section 5 discusses the analysis techniques used in CyclePad, including how it extends constraint propagation to include table lookups and on-the-fly reformulation of system equations. Section 6 describes CyclePad's explanation system, crucial to the program's success, showing how a *structured explanation system* can provide understandable, incrementally derived hypertext explanations of a student's work. Section 7 describes the *distributed coaching* model CyclePad uses, including coaching students through contradictions, teleological reasoning to help them understand the meaning of parameters, and case-based design help using analogical reasoning on a web-based design library that can be extended by domain experts. Finally, Section 8 discusses how CyclePad has been deployed and reflects on the implications of this work.

2. Motivation

2.1. Articulate virtual laboratories

Engineering is about creating useful artifacts and systems. Design is at the heart of engineering, but unfortunately, typically not at the heart of engineering education. Engineering education in the 1990s is oriented towards analysis rather than design.

Students who arrive at college eager to create new things are instead put through many courses of analysis. Design, if they get exposure to it at all, is reserved for “capstone” design courses. The perceived irrelevance of what they are learning to what they want to do is a major factor in the small incoming enrollments in (and large transfer rates out of) engineering education. Design tasks are highly motivating, and tie classroom learning to real-world concerns. Increased use of design tasks in education is now widely perceived as a way to improve engineering education [1].

Unfortunately, there are many reasons why design does not permeate engineering education. Design exercises historically consume tremendous resources (especially time) to create, supervise, and evaluate. In many domains cost and/or safety concerns preclude tackling many design tasks that would be illuminating (e.g., designing power plants or jet engines). The sheer amount of detail work needed to produce a working artifact can be important for integrative exercises, but it detracts from the value of design exercises aimed at teaching just one principle. These factors are leading to an increasing role for software in *virtual laboratories* that enable students to “build” designs and try them out without expense or danger. Examples of commercial software that can be used this way include MultiSim^{TM} [13] for electronic circuits and Working Model^{TM} [43] for mechanics. When used properly, such programs provide valuable experiences for students. However, they still do not solve many of the resource constraints involved in using design in teaching. First, they do not provide explanations, relying on human instructors and lab assistants to provide the scaffolding necessary for students. For example, such programs do not help students understand exactly where and how their assumptions lead to problems. Second, they do not providing coaching. For example, they do not help students tie their results back to the phenomena of the domain, nor do they provide advice on how to improve a student’s design. Third, they do not provide support for assessment—instructors still have to grade every aspect of a student’s work by hand, a process which is often less convenient for work submitted on-line than on paper.

These limitations can be overcome by using artificial intelligence techniques to create *articulate virtual laboratories*, software that has a conceptual understanding of the domain being taught and uses that understanding to scaffold students in design tasks. An articulate virtual laboratory (hereafter, “AVL”) includes several components:

- A *conceptual CAD system* that enables students to construct designs from a collection of parts.
- *Analysis tools* that enable students to understand the properties of their design and how they arise from the consequences of their assumptions. These tools relieve students of the burden of routine calculations, in much the same manner that calculators are used in mathematics education. However, these design tools go beyond calculators in a number of ways.
- *Coaching* that scaffolds students in creating, analyzing, and improving their design.

CyclePad is an articulate virtual laboratory for learning engineering thermodynamics by designing cycles. Students using CyclePad can design power plants, refrigerators, engines, cryogenic systems, and other types of thermodynamic cycles. For readers not familiar with this domain, we briefly review thermodynamic cycles in the next section.

2.2. Engineering thermodynamics

A thermodynamic cycle is a system within which a working fluid (or fluids) undergoes a series of transformations in order to process energy. Every power plant and every engine is a thermodynamic cycle. Refrigerators and heat pumps are also examples of thermodynamic cycles. Cycles play much the same role in engineering thermodynamics as electronic circuits do in electrical engineering; a small library of parts (in this case, compressors, turbines, pumps, heat exchangers, and so forth) are combined into networks, thus potentially generating an unlimited set of designs for any given problem. Practically, cycles range from four components, in the simplest cases, to networks consisting of dozens of components.

The analysis and design of thermodynamic cycles is the major task that drives engineering thermodynamics, aside from applications to chemistry. In thermodynamics education for engineers, cycle analysis and design generally appears towards the end of their first semester, or is even delayed to a second course, since understanding cycles requires a broad and deep understanding of the fundamentals of thermodynamics. Introductory engineering thermodynamics textbooks typically devote several chapters to cycle analysis, and in more advanced books the fraction devoted to cycles rises sharply. Indeed, some textbooks focus exclusively on cycle analysis (e.g., [32]). Aside from their intrinsic interest, the conceptual design of thermodynamic cycles provides a highly motivating context for students to learn fundamental principles more deeply than they would otherwise.

One source of the complexity of cycle analysis stems from the complex nature of liquids and gases, and the fact that subtle interactions between their properties must be harnessed in order to improve designs. The purpose of cycle analysis is to determine the overall efficiency of a system, how much heat or work is consumed or produced, and what operating parameters (e.g., temperatures and pressures) are required of its components. An important activity in designing cycles (or indeed in many engineering design problems) is performing sensitivity analyses, to understand how choices for properties of the components and operating points of a cycle affect its global properties.

To illustrate, consider a sequence of power plant designs. Fig. 1 shows a simple Rankine cycle, which pumps a working fluid (e.g., liquid water), into a boiler to produce steam. In the turbine the high-pressure steam expands, thus performing work. Heat is extracted from the steam in the condenser so that the working fluid is again liquid. Finally, this liquid water is pumped into the boiler (which requires work to be done in the pump) thus beginning the whole cycle again. These processes happen continuously in steady flow around the cycle.

As it happens, the simple Rankine cycle trades efficiency for practicality. Fully condensing the working fluid exiting the turbine enables the use of a pump, but at the cost of discarding thermal energy that must be injected anew in the boiler. Operating the boiler at higher temperatures and pressures improves this situation, but only to a point, beyond which common materials cannot withstand the operating conditions.

Fig. 2 shows a more efficient design, which uses a second turbine to extract more energy from the steam. The purpose of the reheater is to ensure that the steam does not begin to condense in the latter stages of the second turbine, because water droplets moving at high speed may damage the blades of the turbine. The extra energy required to reheat the steam

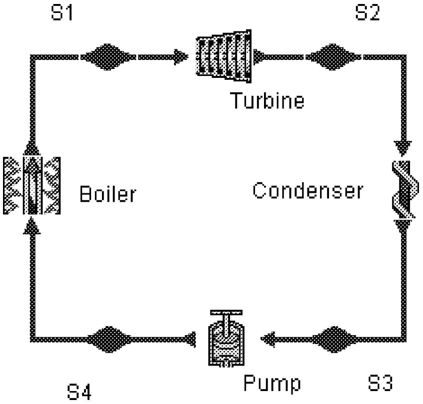


Fig. 1. A Rankine cycle.

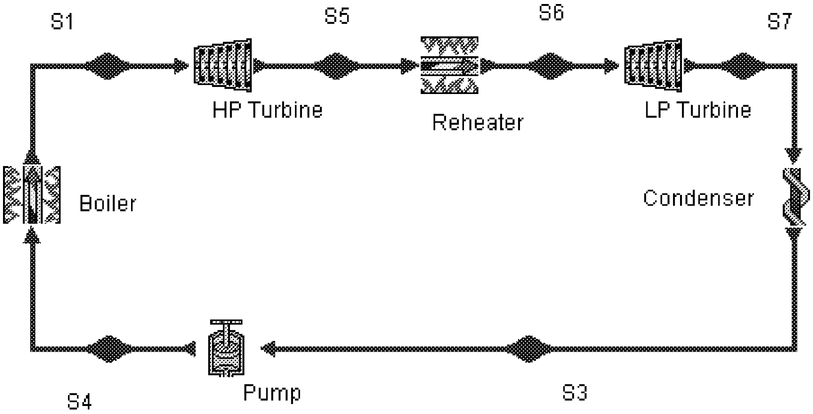


Fig. 2. Rankine cycle with reheat.

is more than balanced by the additional work gained from the second turbine. One can do even better, however. Fig. 3 shows a *regenerative feedwater cycle* where some of the steam from the outlet of the high-pressure turbine is routed back to the water feeding the boiler. The heat of the steam tapped from the turbines and added to the boiler feedwater more than makes up for the loss of work from that steam, thus increasing efficiency. Many power plants use several stages of reheat and regeneration, although the increased efficiency must be carefully balanced by the increased cost of introducing more parts.

2.2.1. Steady-flow versus closed cycles

The kinds of cycles described above are called *steady flow* cycles. Steady-flow systems, such as Rankine cycles, involve changes of properties in a working fluid that is continuously moving within a collection of components. Startup, shutdown, and transient phenomena are not modeled and the system is presumed to be at steady-state, so that the

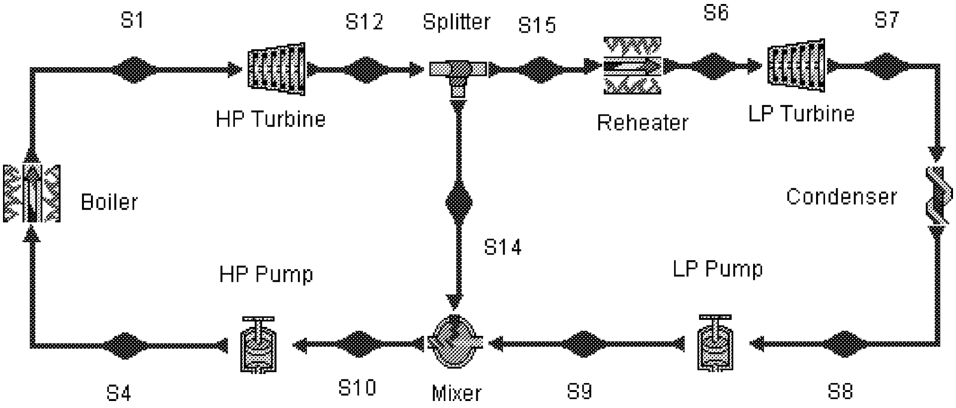


Fig. 3. Regenerative feedwater cycle.

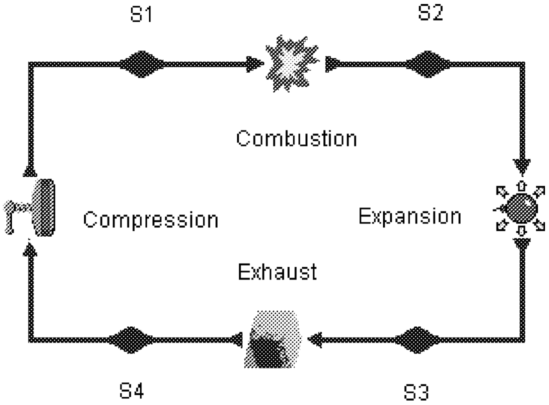


Fig. 4. A closed cycle example.

physical properties at each physical location within the cycle are constant over the time of interest.

Another category of cycles consists of *closed cycles,* where a sequence of processes (perhaps repeating) occurs in the same place. In a steady-flow cycle, the four essential thermodynamic processes—compression, heating, expansion, and cooling—occur simultaneously but at different locations (i.e., the pump, the boiler, the turbine, and the condenser) within the cycle. In a closed cycle, such as an automobile engine, these same processes occur in the same place (i.e., the cylinder) but at different times. Fig. 4 illustrates the Otto cycle commonly used in automobile engines.

The ability to describe closed cycles also provides the ability to handle many introductory types of problems involving single processes or a sequence of processes. These assignments are given to students as a way for them to explore the fundamental properties of fluids and processes.

It is worth noting that, like most engineering domains (but unlike electronics), the mapping from the detailed, physical artifact to the abstractions used to design and analyze it is often complex. This is particularly apparent in closed cycles, where all the mechanisms needed to achieve the physical processes (e.g., pistons, crankshafts, ignition system, fuel system, etc.) are all abstracted away. This level of abstraction is standard and entirely appropriate; if the fundamental thermodynamic properties of the cycle do not have the required performance parameters, other perspectives are irrelevant.

2.3. *What is hard about engineering thermodynamics*

The design of CyclePad was driven by addressing the needs of instructors in teaching engineering thermodynamics. A variety of problems arise when teaching students how to design and analyze thermodynamic cycles: [1]

(1) Students tend to get bogged down in the mechanics of solving equations and carrying out routine calculations. They avoid exploring multiple design alternatives and avoid carrying out trade-off studies (e.g., seeing how overall cycle efficiency varies as a function of turbine efficiency versus how it varies as a function of boiler outlet temperature). Yet without making such comparative studies, many opportunities for learning are lost.

(2) Students often have trouble thinking about what modeling assumptions they need to make, such as assuming that a heater operates isobarically (i.e., no pressure drop across it), leading them to get stuck when analyzing a design.

(3) Students typically don't challenge their choices of parameters to see if their design is physically possible (e.g., that their design does not require a pump to produce rather than consume work).

(4) Students typically have no basis for relating the values they calculate to the physical world and their everyday experience. The units of thermodynamic quantities, such as kilowatts, are not as accessible as pounds or feet. This lack of intuition about, for instance, whether 10,000 kilowatts is enough to light a room or a city causes students to treat thermodynamics problems as abstractions divorced from practical application.

These considerations drove the design of CyclePad. CyclePad handles routine calculations and is set up to facilitate sensitivity analyses, using truth-maintenance techniques to provide explanations for its results. CyclePad helps students keep track of modeling assumptions and their consequences. Finally, CyclePad detects physically impossible designs, using a combination of qualitative constraints and numerical reasoning.

2.4. *CyclePad in action*

When a user starts up CyclePad, they have the choice of designing a *steady flow* system or a *closed cycle* system. Steady-flow systems are specified by combining components

---

[1] These observations are based on the experience of the second author, who teaches engineering thermodynamics to undergraduates, and have been verified repeatedly by our instructor-collaborators at a number of universities.

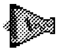
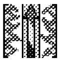
Component	Description	Graphical Symbol
Turbine	Gaseous working fluid expanding through this component causes a shaft to rotate, producing work	
Compressor	Work input to this component causes a compression of gaseous working fluid, resulting in a significant reduction in working fluid volume	
Pump	Work input to this component causes a increases the pressure of liquid working fluid, without an appreciable change in volume	
Heater	Increases the thermal energy of the working fluid. If the fluid is saturated, this may not result in a change in temperature	
Cooler	Decreases the thermal energy of the working fluid. If the fluid is saturated, this may not result in a change in temperature	
Heat-exchanger	Causes a flow of thermal energy from one working fluid to another. In a co-current heat exchanger, the fluids flow in parallel, whereas in a counter-current exchanger they flow in opposite directions.	
Mixer	Mixes two flows into a single outlet stream.	
Splitter	Splits a single flow into two outlet streams.	
Throttle	Causes an unresisted expansion of the working fluid, resulting in a drop in the fluid's temperature, and possibly a change of phase from liquid to gas.	
Source	Originates a flow of working fluid	
Sink	Terminates a flow of working fluid	

Fig. 5. Types of components used to model steady-flow cycles.

from a library (e.g., turbine, compressor, pump, heater, cooler, heat exchanger, throttle, splitter, mixer) illustrated in Fig. 5. Closed cycle systems, such as Otto and Diesel cycles, involve a sequence of physical processes operating on a piece of working fluid in the same physical location. Closed cycle systems are specified by combining physical processes from a library (illustrated in Fig. 6) to specify what happens to the working fluid over the cycle. Closed-cycle designs can also be used to express many of the introductory, non-design problems that occur in thermodynamics courses that explore the behaviors of fluids under different processes.

Let us start by considering a steady-flow design. In a steady-flow system, components are connected together by *stuffs*, which represent the properties of the working fluid at that point in the system. (Stuffs are analogous to nodes in electronic circuits). The interface helps the user put together a design by highlighting what parts remain unconnected and providing simple critiques of the structure. Once the structural description of the cycle is finished (e.g., there are no dangling connections or stuffs), CyclePad allows the user to enter an analysis mode. In analysis mode, the particular properties of the system, such as

Physical Process/event	Description	Graphical Symbol
Combustion Heating	The process of heating the working substance.	
Cooling	The process of cooling the working substance.	
Polytropic Expansion	The process of expanding the working substance.	
Polytropic Compression	The process of compressing the working substance.	
Begin	Represents beginning of a sequence of occurrences of physical processes. Not used when representing cycles of processes.	
End	Represents ending of a sequence of occurrences of physical processes. Not used when representing cycles of processes.	

Fig. 6. Types of components used to model closed cycles.

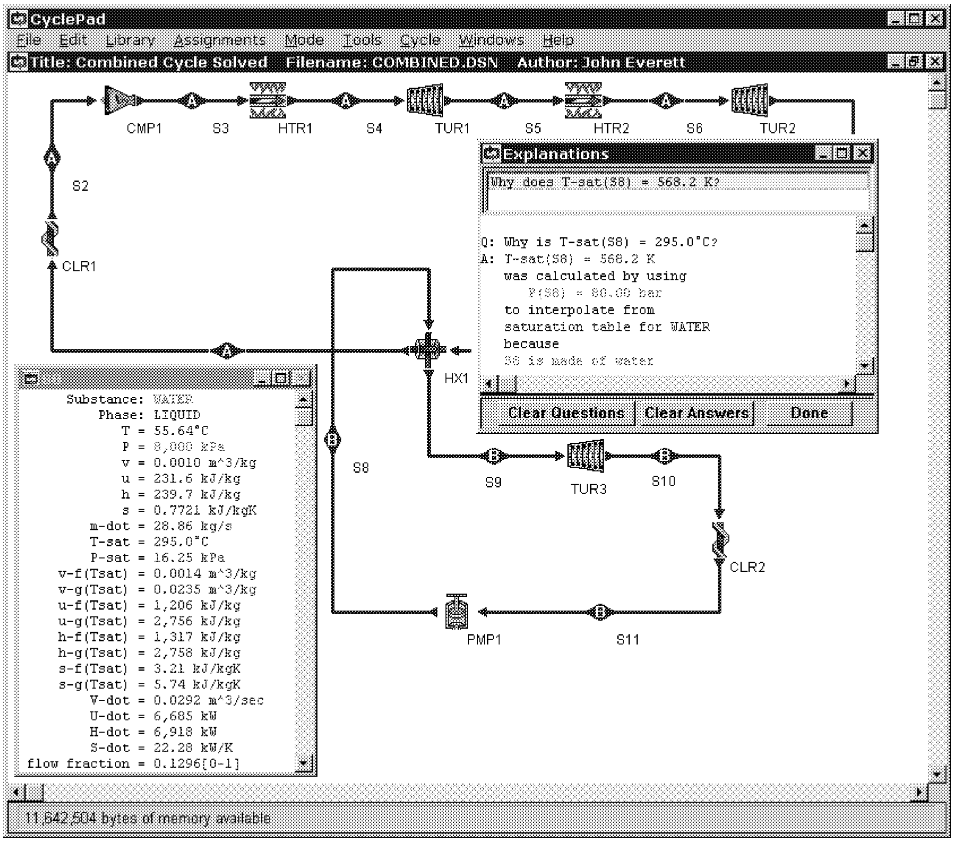
the choice of working fluid, the values of specific numerical parameters, and modeling assumptions can be entered and explored.

CyclePad accepts information incrementally, deriving from each user assumption as many consequences as it can. At any point questions can be asked by clicking on a displayed item to show the set of questions (or commands) that make sense for it. In addition to numerical parameters and structural information, all modeling assumptions made about a component are displayed with it, and clicking on a component shows the modeling assumptions that can legitimately be made about that component, given what is known about the system so far. The questions and answers are displayed in English, and include links back into the explanation system, thus providing an incrementally generated hypertext. Fig. 7 illustrates.

In addition to numerical assumptions, selecting a component provides commands for making or retracting modeling assumptions concerning that component. For example, clicking on a turbine yields a menu of commands which offers the options of assuming the turbine is adiabatic or isentropic. Such modeling assumptions can introduce new constraints which may help carry an analysis further and new parameters (e.g., the efficiency of the turbine) that must be set.

When CyclePad uncovers a contradiction, it changes the interface to provide tools to resolve the problem by presenting the source of the contradiction (e.g., an impossible fact becoming believed, or conflicting values for a numerical parameter) and the set of assumptions underlying that contradiction. The hypertext dialog facilities can be used with this display to figure out which assumption(s) are dubious and change them accordingly.

Sensitivity analyses are an important tool for building a student's intuitions about how physical principles contribute to the way an artifact works. CyclePad performs sensitivity analyses by asking a student to select a dependent parameter first. It then offers a choice of potentially relevant independent parameters, based on its analysis of the cycle. Given an independent and dependent parameter, CyclePad derives an equation (including table



```
Why does T-sat(S8) = 568.2 K?

Q: Why is T-sat(S8) = 295.0°C?
A: T-sat(S8) = 568.2 K
   was calculated by using
      P(S8) = 80.00 bar
   to interpolate from
   saturation table for WATER
   because
   S8 is made of water
```

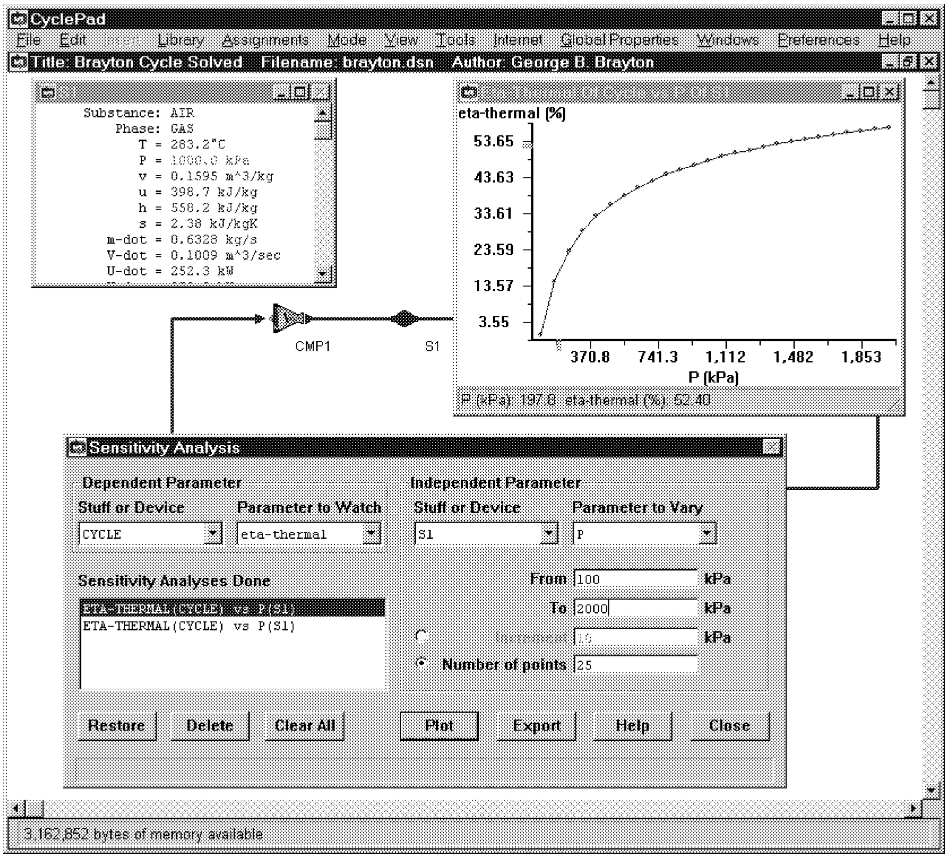
```
     Substance: WATER
         Phase: LIQUID
             T = 55.64°C
             P = 8,000 kPa
             v = 0.0010 m^3/kg
             u = 231.6 kJ/kg
             h = 239.7 kJ/kg
             s = 0.7721 kJ/kgK
         m-dot = 28.86 kg/s
         T-sat = 295.0°C
         P-sat = 16.25 kPa
   v-f(Tsat) = 0.0014 m^3/kg
   v-g(Tsat) = 0.0235 m^3/kg
   u-f(Tsat) = 1,206 kJ/kg
   u-g(Tsat) = 2,756 kJ/kg
   h-f(Tsat) = 1,317 kJ/kg
   h-g(Tsat) = 2,758 kJ/kg
   s-f(Tsat) = 3.21 kJ/kgK
   s-g(Tsat) = 5.74 kJ/kgK
         V-dot = 0.0292 m^3/sec
         U-dot = 6,685 kW
         H-dot = 6,918 kW
         S-dot = 22.28 kW/K
flow fraction = 0.1296[0-1]
```

Fig. 7. The CyclePad interface.

references, if needed) that describes the dependent parameter in terms of the independent parameter, and plots the results. Fig. 8 illustrates.

Coaching is available on student request. The on-board analysis coach uses a teleological analysis of the student's design to identify the likely functional roles of each component, and point out potential problems in choices of parametric values based on these assumed roles. Explanations for functional roles can also be provided, as Fig. 9 illustrates. Students can also access the CyclePad Guru, a server operating at Northwestern, for additional assistance. For example, students can ask for help completing the analysis of their design and for help in improving their design, as described in Section 7.

The design and analysis of closed cycles proceeds in the same manner, except that the ontology of connection differs. The analog of state points for closed cycles is therefore the *time-slice*, which, for simplicity, we term a *slice*. Reifying slices (a concept Hayes introduced in [31]) enables us to portray a time-varying cycle in two-dimensional space, as in Fig. 4.



```
      Substance: AIR
          Phase: GAS
              T = 283.2°C
              P = 1000.0 kPa
              v = 0.1595 m^3/kg
              u = 398.7 kJ/kg
              h = 558.2 kJ/kg
              s = 2.38 kJ/kgK
          m-dot = 0.6328 kg/s
          V-dot = 0.1009 m^3/sec
          U-dot = 252.3 kW
```

P (kPa): 197.8 eta-thermal (%): 52.40

Dependent Parameter
Stuff or Device: CYCLE
Parameter to Watch: eta-thermal

Independent Parameter
Stuff or Device: S1
Parameter to Vary: P

Sensitivity Analyses Done:
ETA-THERMAL(CYCLE) vs P(S1)
ETA-THERMAL(CYCLE) vs P(S1)

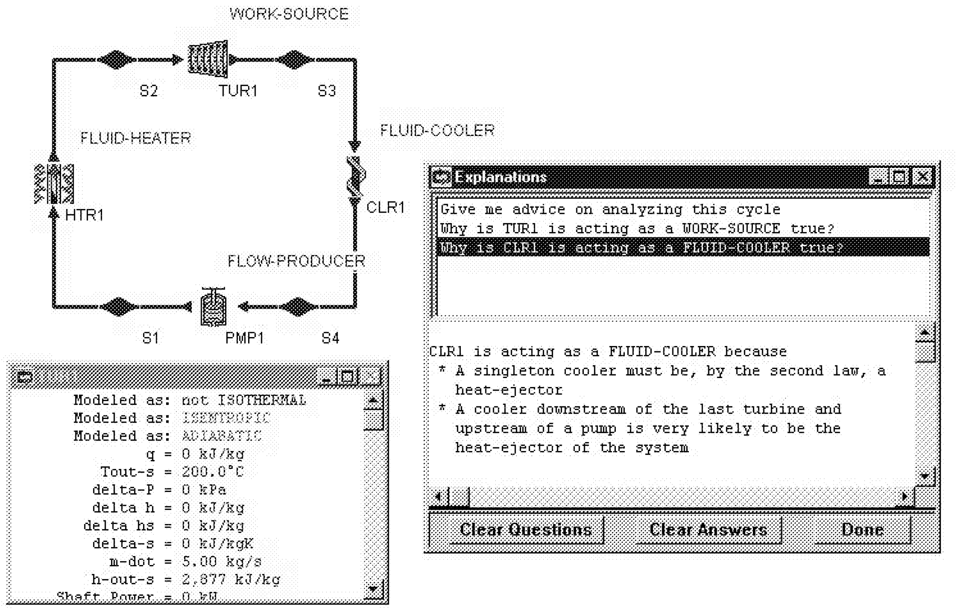
From 100 kPa
To 2000 kPa
○ Increment 10 kPa
● Number of points 25

3,162,852 bytes of memory available

Fig. 8. CyclePad sensitivity analysis.

Even though the vocabulary of steady-flow cycles seems more closely tied to physical components (pumps, turbines, etc.), here, too, significant abstractions are being made. Turbines in high-capacity power plants, for instance, often have taps for bleeding off steam between stages to be used in regeneration. Such turbines can be modeled in CyclePad by combinations of turbines and splitters. Similarly, mixers in CyclePad are used to represent certain kinds of heat exchangers, jet ejectors, and places in the cycle where flows are joined. Since there is no standard graphical symbol vocabulary for cycle diagrams (unlike, say, circuit schematics in electronics), we designed a vocabulary that maximizes expressive power with a minimal number of symbols. This vocabulary is designed to fit naturally with existing thermodynamics education and practice, while forcing students to confront modeling issues in their choice of components and processes.

Although the level of design and analysis in CyclePad is abstract, it provides a setting for design exercises that fits with thermodynamic practice. Conceptual design of thermodynamic cycles emphasizes steady-state behavior and overall performance, rather than the dynamics of the system, startup and shutdown procedures, plant geometry, or any



```
     Modeled as: not ISOTHERMAL
     Modeled as: ISENTROPIC
     Modeled as: ADIABATIC
              q = 0 kJ/kg
         Tout-s = 200.0°C
        delta-P = 0 kPa
        delta h = 0 kJ/kg
       delta hs = 0 kJ/kg
        delta-s = 0 kJ/kgK
          m-dot = 5.00 kg/s
        h-out-s = 2,877 kJ/kg
    Shaft Power = 0 kW
```

```
Give me advice on analyzing this cycle
Why is TUR1 is acting as a WORK-SOURCE true?
Why is CLR1 is acting as a FLUID-COOLER true?
```

```
CLR1 is acting as a FLUID-COOLER because
 * A singleton cooler must be, by the second law, a
   heat-ejector
 * A cooler downstream of the last turbine and
   upstream of a pump is very likely to be the
   heat-ejector of the system
```

Fig. 9. CyclePad uses teleological reasoning to identify intended functions of components.

of the more detailed design decisions that are required to design a physical artifact to the level of detail that enables its construction. This abstraction does not remove the analysis from real-world concerns. In engineering, issues like efficiency provide bounds on the performance of the actual plant, and cost estimation procedures are used to determine the feasibility of a project. CyclePad supports these analyses by including an economic model, so that concerns such as capital costs and fuel costs can be considered when evaluating a design, in addition to standard thermodynamic properties.

3. CyclePad's architecture

CyclePad, as an articulate virtual laboratory, is designed to be more open-ended than typical ITS' (cf. [3]) but provide more scaffolding than traditional virtual laboratory software. An open question in the design of educational software is the relative value of student modeling (cf. [33]). The AVL architecture is agnostic on this question. In creating CyclePad we have not found student modeling to be necessary, but this does not rule out student modeling in AVLs in general.

The architecture of CyclePad is shown in Fig. 10. The CAD-like schematic editor used for creating the structure of designs and the other parts of the user interface are similar to other interactive engineering programs, so we lump all of those facilities into the Interface Manager for this paper, to concentrate on the AI techniques used. The analysis facilities are provided by the constraint propagator and sensitivity analysis, both of which rely on property table interpolation. Scaffolding and coaching is provided by the structured

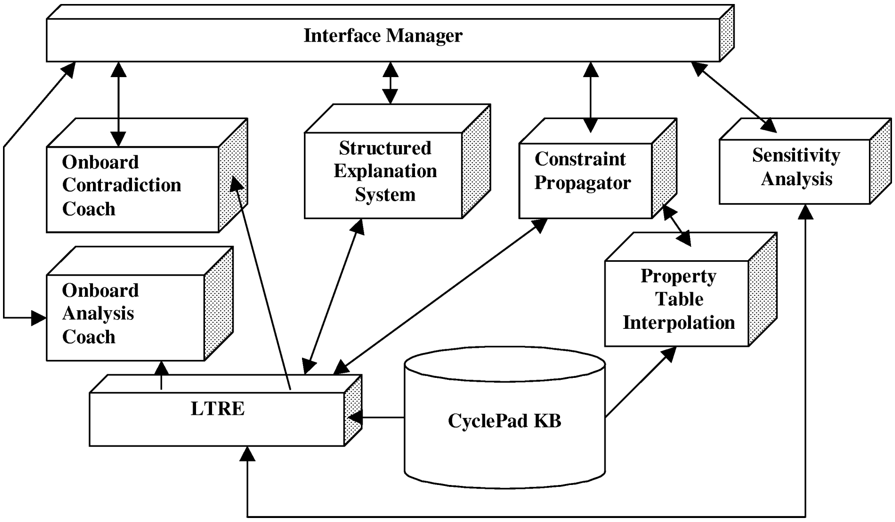


Fig. 10. CyclePad’s architecture. Arrows indicate information flow.

explanation system and onboard contradiction and analysis coaches. The LTRE module [23] is a *l*ogic-based *t*ruth maintenance system (TMS) coupled to a forward-chaining *r*ule *e*ngine, and serves as a blackboard, storing a design and the results of analyzing it.

The other modules are described in subsequent sections; here we comment on the design of the LTRE, since it has been critical to the project’s success. The dependency network in the TMS is used to provide explanations, guide inference, detect contradictory design assumptions, and facilitate sensitivity analyses, as detailed below. We used a logic-based TMS (LTMS) [36] in CyclePad because it offered the best tradeoff between inferential power and economy. We ruled out a justification-based TMS (JTMS) because Horn clauses are too clumsy for many of CyclePad’s inferential needs, including biconditionals (used in definitional consequences of modeling assumptions, e.g., a compressor is operating isentropically exactly when its isentropic efficiency is 1.0) and TAXONOMY constraints [31] (used in implementing assumption classes). The ability of an assumption-based TMS (ATMS) to provide rapid switching between very different contexts was not required because, while frequent additions and retractions of assumptions are made in carrying out an analysis, these changes are typically a small fraction of the working set of assumptions in force.

The LTMS used is a variant of that described in [23], because we found that the TMS dependency structures quickly consumed all available memory in caching the derivations of numerical values. Initially, this is not a problem, and indeed is essential for CyclePad to provide hypertext explanations of derived numerical values. However, when a CyclePad user changes a numerical value, the conventional LTMS would preserve the dependency structures for the old value and generate new (and virtually identical) structures for the new value. As each assumed value may have many consequences, these structures can be quite large.

Observations of CyclePad in typical conditions indicated that, while the amount of memory consumed rose linearly with the number of assume/retract cycles, the total time taken for each cycle rose non-linearly. We identified two sources for the non-linear time increase:

(1) there are more clauses to process on each iteration, because the dependency network has grown, and
(2) as storage space begins to run out, the Lisp garbage collector and the virtual memory system require a much larger fraction of the total time.

Our solution to this problem was to modify the LTMS to enable fact-level garbage collection [16]. This enables us to designate certain classes of facts as eligible for deletion when they become unknown. We preserve the integrity of the dependency network for producing explanations of current beliefs at the cost of greater reliance on the inference engine to rederive facts. This solution makes sense when one can identify classes of facts in which

(1) it is unlikely that a particular fact will be believed again once it is retracted, and
(2) the cost of rederivation is small.

Numerical assumptions in CyclePad fit both of these criteria. Using a fact garbage collector and designating statements of numerical values as collectible results in a significant improvement in CyclePad's performance. Empirically, we have assumed and retracted 1000 consecutive values at an average time per assumption/retraction cycle of 3.1 seconds on a Pentium 133 MHz machine equipped with 32 MB of RAM. In contrast, CyclePad running on the same computer with the conventional LTRE on the same example exhausted the memory space at 48 changes, at which point the average retraction was requiring 46 seconds.

4. Domain theory

CyclePad's domain theory was developed over a period of six years, with the bulk of the work occurring in the first four years. It has been a collaborative effort involving both mechanical engineering faculty and students and AI faculty and students, including significant feedback and suggestions from other instructors and CyclePad users. The development was guided entirely by the needs of engineering thermodynamics instruction. Consequently, the validation consists of handling the kinds of problems that appear in textbooks and the kinds of design exercises that students tackle in both introductory and advanced courses.

The compositional modeling methodology [17,18,38] is used to organize the domain theory. Compositional modeling provides formal representation and reasoning techniques for formulating and reasoning about models. Briefly, compositional modeling works as follows. Knowledge about an area is organized into a *domain theory*. A principle component of domain theories are *model fragments*, which express some piece of knowledge about the domain. Model fragments include the conditions under which they are applicable, the types of individuals that they can be instantiated for, and what must be true for them to be valid. Formulating a model for a specific problem consists of instantiating fragments from the domain theory, taking into account the kinds of tasks the model is to be used for. The link between task information and the structure of the domain theory is expressed in terms of *modeling assumptions*. Modeling assumptions encode control information about what perspectives (including ontologies), granularity, and/or phenomena make sense to consider or ignore. The construction of a model for a particular task is generally automated, using a *model formulation* algorithm. Model formulation algorithms use information about the physical situation and the task to select a consistent and coherent set of modeling assumptions, and based on these assumptions, instantiate a collection of model fragments from the domain theory to compose a model of that situation for the intended purpose.

CyclePad's domain theory encodes knowledge of thermodynamics relevant to reasoning that goes on during conceptual design of thermodynamic cycles. Modeling assumptions play a crucial role in organizing this domain theory. For instance, the choice of whether to model a system as a steady-flow or open-cycle system is fundamental, leading to a completely different set of primitives being available for a design. Indeed, learning to make appropriate modeling assumptions is a key problem for students learning the domain. On the other hand, the range of types of analyses used in conceptual design of thermodynamic cycles is narrower than some situations where compositional modeling has been used previously. As noted earlier, CyclePad does not cover dynamic phenomena, since such phenomena are ignored in such analyses. Similarly, it does not cover fault models, chemical effects (corrosion, working fluids consisting of multiple substances), and other phenomena that lie outside what is needed to support engineering thermodynamics education.

The modeling language used in CyclePad is similar to other implementations of compositional modeling. CyclePad's knowledge base currently consists of 52 conceptual entities, 6 physical processes, 38 assumption classes, 195 equations, 184 pattern-directed rules and 156 background facts about thermodynamics. The rest of this section outlines the types of entities defined in CyclePad's domain theory, and what CyclePad knows about modeling assumptions, equations, and property tables.

Issue	Steady-flow	Closed-cycle
Role of components	Explicit in structure of design; Provides the notion of part.	Unspecified; any components that achieve the desired process will do.
Role of physical processes	Implied by the choice of components.	Explicit in the structure of the design; provides the notion of part.
Connection between parts	Stuffs (aka state points) describe properties of working fluid between components.	Slices describe properties of working fluid before and after physical processes.
Time	Implicit episodes of indefinite duration during which all operations are viewed as steady.	Sequence of physical processes, with repeating structure in case of cycles.

Fig. 11. Comparison between steady flow and closed cycle ontologies.

4.1. Types of entities

As with most kinds of engineering schematic notations, CyclePad uses a notion of part and connections between parts to organize the structure of a design. However, as noted previously, CyclePad supports both steady-flow and closed-cycle models. There are deep conceptual differences between these ontological choices, but there are also substantial overlaps between them. The differences are summarized in Fig. 11.

Process descriptions in closed-cycle representations always have a single before and after slice. Component descriptions in steady-flow representations specify their inlets and outlets. These participants are always expressed when the component or physical process is instantiated. Equations for both types of entities are expressed in terms of the parameters of that type of entity and the parameters of the stuffs or slices that participate in it. Qualitative constraints on the participants are included in the entity descriptions. Figs. 12 and 13 show examples of definitions for steady-flow components and closed-cycle processes, respectively.

```
(defentity (turbine ?self ?in ?out)
  (work-converter ?self ?in ?out) ;;
  (ss-thermodynamic-stuff ?in) ;; steady-state stuff
  (ss-thermodynamic-stuff ?out) ;; ditto
  (total-fluid-flow ?in ?out) ;; Process occurring inside
  (expansion ?in ?out ?self) ;; Process occurring inside
  (parameter (spec-shaft-work ?self)) ;; J/kg
  (parameter (shaft-power ?self))
  (parameter (mass-flow ?self))
  (== (mass-flow ?in) (mass-flow ?self))
  (parameter (delta-p ?self))
  (parameter (delta-spec-h ?self)) ;; enthalpy, in J
  (parameter (delta-spec-h-isentropic ?self)) ;; J
  (parameter (delta-spec-s ?self))
  (parameter (pr ?self)) ;; Pressure ratio
  (parameter (Q-dot ?self))
  (parameter (spec-q ?self))
  (parameter (spec-shaft-work-isentropic ?self))
  (parameter (Tout-isentropic ?self))
  (parameter (spec-h-out-isentropic ?self))
  ((parts:cycle) has-member ?self) ;; Structural relationships
  (?self instance-of turbine) ;; Curried for LTRE efficiency
  (?self part-names (in out))
  (?self IN ?in) (?in IN-OF ?self)
  (?self OUT ?out) (?out OUT-OF ?self)
    ;; Contributions to global sets, which must be closed later
  ((possible-heat-flow-contributors:cycle) has-member
    (Q-dot ?self))
  ((work-flows-out:cycle) has-member (shaft-power ?self)))
```

Fig. 12. Example of a steady-flow component definition (partial).

```
(defentity (closed-system-process ?self ?in ?out)
  (thermodynamic-stuff-slice ?in)
  (thermodynamic-stuff-slice ?out)
  (mass-transfer ?in ?out)
  ;; Structural relationships
  ((parts:cycle) has-member ?self)
  (?self part-names (in out))
  (?self in ?in) (?in IN-OF ?self)
  (?self out ?out) (?out OUT-OF ?self)
  (parameter (mass ?self))
  (parameter (spec-Q ?self))
  (parameter (spec-work ?self))
  (parameter (Q ?self))
  (parameter (work ?self)))

(defentity (polytropic-process ?self ?in ?out)
  (closed-system-process ?self ?in ?out)
  (parameter (n ?self)))

(defentity (polytropic-expansion ?self ?in ?out)
  (polytropic-process ?self ?in ?out)
  (expansion ?in ?out ?self) ;; process occurrence
   ((work-flows-out:cycle) has-member (work ?self))
  ;; Structural relationships
  (?self instance-of polytropic-expansion))

(defAssumptionClass ((polytropic-expansion ?self ?in ?out))
  (adiabatic ?self)
  (:not (adiabatic ?self)))

(defAssumptionClass ((polytropic-expansion ?self ?in ?out))
  (isentropic ?self)
  (:not (isentropic ?self)))
  (defAssumptionClass ((polytropic-expansion ?self ?in ?out))
  (isothermal ?self)
  (:not (isothermal ?self)))
```

Fig. 13. Example of a closed-cycle process definition (partial).

The explicit representation of physical processes occurring in components is a crucial use of qualitative physics in CyclePad. Representing such processes explicitly simplifies the knowledge base (i.e., the same definition of compression is used for pumps and compressors, and the same definition of heat flow is used for heaters, coolers, and heat exchangers), and enables explanations to be organized in terms of meaningful mechanisms occurring inside components. As with other types of entities, qualitative physics provides the medium for representing constraints on what is physically possible. Fig. 14 illustrates.

```
(defProcessEpisode (heat-flow ?src-start ?src-end
     ?dst-start ?dst-end)
  (> (T ?src-start) (T ?dst-start))
  (:not (< (T ?src-start) (T ?dst-end)))
  (:not (> (T ?dst-end) (T ?src-end))))

(defProcessEpisode (compression ?in ?out ?worker)
  (> (P ?out) (P ?in))
  (< (spec-shaft-work?worker) 0)) ;; Eats work

(defProcessEpisode (expansion ?in ?out ?receiver)
  (< (P ?out) (P ?in))
  (> (spec-shaft-work?receiver) 0)) ;; Does work
```

Fig. 14. Part of CyclePad's representation of steady-flow physical processes.

4.2. Modeling assumptions

CyclePad's focus on thermodynamics for helping students learn about cycles simplifies its knowledge base, but the need to represent and reason with modeling assumptions is still critically important. Model formulation is in fact one of the central skills being taught in using CyclePad, since students are not used to the level of thinking about modeling assumptions that thermodynamics requires. A boiler, for instance, is typically approximated as a heater for the purposes of cycle analysis. A flash chamber is modeled as a splitter whose working fluid is saturated and with particular assumptions about the dryness of the outlets. A multi-stage turbine is modeled as a sequence of turbines and splitters. In most cases, detailed models are simply unavailable, and simplifying assumptions are needed to make progress in an analysis. Assuming that heaters and coolers are isobaric, or that turbines are isentropic, are examples of simplifying assumptions. The modeling assumptions used in CyclePad are shown Fig. 15.

As usual in compositional modeling, CyclePad's modeling assumptions are organized into *assumption classes* [2,17]. Assumption classes are always associated with particular classes of entities. The relevance of one assumption class can depend on the particular choices made for another assumption class. For example, it only makes sense to consider whether a compressor is isentropic if it is already known (or assumed) adiabatic. CyclePad's focus on thermodynamics for helping students learn about cycles simplifies its knowledge base, but the need to represent and reason with modeling assumptions is still critically important.

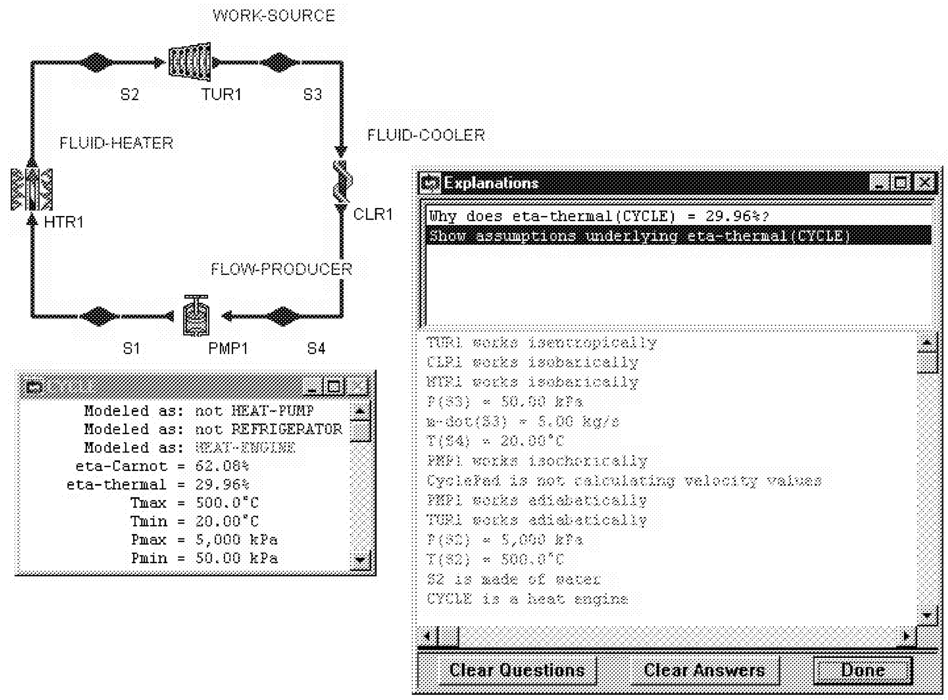
CyclePad's interface is designed to scaffold students in working with modeling assumptions. This scaffolding takes three forms. First, when starting a new design, students must choose whether they are creating a steady-flow or an open-cycle model. This choice must be made first since it determines what set of primitives is available for their design. Second, modeling assumptions associated with each component and process are explicitly shown along with other information about that entity. This is illustrated in Fig. 16. As choices for modeling assumptions are made, other logically dependent choices may become possible or become ruled out, and the display is updated automatically with this

Component	Modeling Assumptions	Process	Modeling Assumptions
Turbine	Isentropic Adiabatic Isothermal	Combustion heating	Isobaric Isochoric Isothermal
Compressor	Isentropic Adiabatic Isothermal	Cooling	Isobaric Isochoric Isothermal
Pump	Isentropic Adiabatic Isothermal	Polytropic expansion	Adiabatic Isentropic Isothermal
Heater	Isobaric Isochoric	Polytropic compression	Adiabatic Isentropic Isothermal
Cooler	Isobaric Isochoric		
Heat-exchanger	Isobaric Isochoric Co-current Counter-current		
Mixer	None		
Splitter	At-saturation Isoparametric		
Throttle	None		
Source	None		
Sink	None		

Fig. 15. Modeling assumptions that can be used in CyclePad.

information. We have found this explicit reminder of the need for modeling assumptions to be very powerful in keeping students on-track. Third, the analysis tools in CyclePad help students figure out if their choices of idealizations make sense. Modeling assumptions appear in the explanations CyclePad generates, which enables students to easily see the consequences of their assumptions. The mechanisms of assumption classes and logical constraints between modeling assumptions helps students understand and appreciate the consequences of their assumptions. For instance, students quickly discover that they cannot assume a turbine is isentropic if different entropies have already been computed for its inlet and outlet.

CyclePad does not provide direct assistance with formulating an idealized model from an informal specification (e.g., natural language descriptions or sketches). Consequently, automatic model formulation, which typically has been the focus of previous compositional modeling work, has not been relevant for this application. We have deliberately not provided support for this aspect of the task because it is sufficiently constrained in educational settings that students will learn more from doing it themselves, using CyclePad as an arbiter of whether or not their choices can be consistently realized.



```
     Modeled as: not HEAT-PUMP
     Modeled as: not REFRIGERATOR
     Modeled as: HEAT-ENGINE
    eta-Carnot = 62.08%
  eta-thermal = 29.96%
          Tmax = 500.0°C
          Tmin = 20.00°C
          Pmax = 5,000 kPa
          Pmin = 50.00 kPa
```

```
Why does eta-thermal(CYCLE) = 29.96%?
Show assumptions underlying eta-thermal(CYCLE)
```

```
TUR1 works isentropically
CLR1 works isobarically
HTR1 works isobarically
P(S3) = 50.00 kPa
m-dot(S3) = 5.00 kg/s
T(S4) = 20.00°C
PMP1 works isochorically
CyclePad is not calculating velocity values
PMP1 works adiabatically
TUR1 works adiabatically
P(S2) = 5,000 kPa
T(S2) = 500.0°C
S2 is made of water
CYCLE is a heat engine
```

Fig. 16. Modeling assumptions are made explicit in CyclePad's interface and explanations.

4.3. Equations

There are two kinds of equations in CyclePad:

- *Local equations* constrain the parameters of a particular entity, component, or physical process.
- *Global equations* are based on properties of multiple components in the cycle.

Two examples of local equations are shown in Fig. 17. The potential validity of local equations is predicated by modeling assumptions. Local equations have the same temporal extent as their subject; as long as it is part of the design, and the modeling assumptions for the equation are valid, the equation is valid. Cycles themselves can have local equations. For example, the net work of a cycle is the difference between the sum of the work done by the cycle and the work done to the cycle, both of which are parameters of the entire cycle.

Global equations depend on the structure of the design. There are a number of global equations for every cycle. For instance, four global equations determine the heat and work flows into and out of the cycle. These inflows and outflows are then used to determine the net heat flow and net work flow for each cycle in the design. They are formulated based on closed-world assumptions about sets satisfying particular criteria. Returning to the example of net work, the sum of work flows out of the cycle is a parameter of the cycle defined by a sum of work over components that perform work, e.g., turbines in steady-flow systems. Similarly, the sum of work flows out of the cycle is defined by the sum of work

```
(defEquation Pump-P-ratio
    ((pump ?p ?in ?out))
    (:= (PR ?p) (/ (P ?out) (P ?in))))

(defEquation Reversible-pump-spec-work-1
  ((pump ?p ?in ?out)
    (liquid ?in)
    (isentropic ?p))
  (:= (spec-shaft-work ?p) (- (* (spec-v ?in) (delta-P ?p)))))
```

Fig. 17. Examples of local equations in CyclePad.

over components that absorb work from outside the cycle, e.g., pumps and compressors in steady-flow systems. Other global equations are used to find the maximum and minimum temperatures in the cycle.

An additional complication is that determining the participants in certain equations can depend on the numerical values derived for certain parameters. An example in CyclePad is considering the heat flows into and out of a cycle. Turbines, for example, can either reject heat to their surroundings or absorb heat from them, depending on how they are operating. Assuming that no heat exchange with its surroundings occurs is tantamount to assuming that the turbine is adiabatic. Turbines are in fact commonly assumed adiabatic in order to ignore such potential heat flows, but there are cases where it is important to consider such imperfections (e.g., when a non-ideal isentropic thermal efficiency is given for a component). When such heat flows are considered, they must be taken into account in formulating global equations.

4.4. Tables

One source of complexity in engineering thermodynamics is that simple analytic equations are inadequate to express the constraints that hold between properties of real substances. In some cases, approximations such as ideal gases are valid. Air, for example, can be treated as an ideal gas for many purposes. However, even air cannot be so treated for cryogenic cycles, where the purpose is to liquefy and separate air into its constituent substances. The more typical case, for working substances such as water, ammonia, and refrigerant-12 (better known by its trade name Freon), is that extensive tables of property data are needed to reliably calculate properties of working fluids.

For thermodynamics purposes, pieces of real substances exist in one of three phases:

- *Saturated* fluids are in the process of becoming either a subcooled liquid or a superheated gas. This is predicated on them being at their saturation temperature (equivalently, saturation pressure), which in turn depends on the current properties of the fluid. Fluids at saturation absorb and emit thermal energy with no change in temperature (given a constant pressure).
- *Superheated* fluids are on the gas side of the saturation boundary, i.e., they contain more heat than they would have if they were saturated under their current conditions of pressure or temperature.

- *Subcooled* fluids are on the liquid side of the saturation boundary, i.e., they contain less heat than they would if they were saturated under their current conditions of pressure or temperature.

Each phase for every real working substance requires a property table. Since in principle any two parameters completely constrain any thermodynamic substance, property tables describe a (complex) 2D surface in the space of thermodynamic parameters. Since rich data provide better accuracy, property tables can be quite large. CyclePad has property tables for seven substances: water, nitrogen, ammonia, methane, and Refrigerants 12, 22, and 134a. The largest of these, the one for superheated water, is 122 KB in size and contains over 1900 lines of data (each line containing numerical values for six parameters.

4.5. Economic model

Economic constraints typically play an important role in design. In particular, the thermal efficiency of a design must be weighed against its capital and operating costs. CyclePad's knowledge base includes an economic model whose ballpark calculations are useful in evaluating the relative quality of designs. The total cost of a thermodynamic cycle is a function of capital and operating costs. Capital costs represent the acquisition and installation of the actual machinery, whereas operating costs represent the materials and labor required during the lifetime of the system.

CyclePad incorporates standard engineering cost estimating functions (cf., [12]) that extrapolate capital costs from a known cycle based on the size of the component, generally estimated by mass-flow. Fig. 18 shows the cost function used for medium-pressure boilers (the resulting curve fails to account for the capital cost of high-pressure boiler installations, so we have a separate equation for boilers operating at high pressures). In this equation, ModuleF is a module factor that estimates the cost of installation (e.g., pouring a concrete mounting pad), PressureF is a pressure factor estimating the cost based on the pressure at which the boiler is operating, and MaterialF is a material factor that depends on the material used to construct the device.

CyclePad contains information about several different materials, including stainless steel, nickel alloy, titanium, molybdenum, and unobtainium. Aside from unobtainium, each material has limits on the temperatures (high and low, the latter for cryogenic applications) that it can endure. Unobtainium is useful for suspending the economic constraints on a particular device or subset of devices (albeit at extraordinary cost). CyclePad also estimates the resulting mass of the cycle as a function of the materials employed, which may be a critical constraint, for example, in the design of an aircraft engine.

$$\mathit{CapCost} = \mathit{RefCostMedPrBoiler} \times \left[\frac{\mathit{BoilerMassflow}}{\mathit{RefMassflow}}\right]^{\mathit{SizeExpt}} \times \mathit{ModuleF} \times \mathit{PressureF} \times \mathit{MaterialF}$$

Fig. 18. Medium pressure boiler capital cost function.

Operating costs include an allowance for maintenance, expressed as a percentage of initial capital cost, and fuel cost. CyclePad contains information about the energy content and cost of several different fuels, including natural gas, coal, coal gas, hydrogen, and oil.

For electrical power generating facilities, CyclePad estimates revenue as a function of kilowatt hours produced, which enables the calculation of cycle profitability and net present value based on assumed utilization and an assumed interest rate

Although quite basic, this model introduces real-world constraints that act in opposition to purely performance concerns. The intention here is to make the student aware of the context in which most engineering design occurs, and of the trade-offs in design that must happen in practice.

5. Design and analysis

The process of designing a thermodynamic cycle, like most engineering design problems, starts with a specification of what the cycle is supposed to achieve. Specifications usually include some performance parameters that must be met (e.g., a minimum output of work, a maximum cost, maximum heat flow transferred to the environment, weight, etc.) An initial structure of the cycle is chosen, i.e., components selected and connected together appropriately in the case of steady-flow designs, or the sequence of physical processes in the case of closed-cycle designs. Assumptions and parametric choices are made, analyzed, and revised. These revisions sometimes lead to changes being made in the basic structure of the cycle itself. This process of elaborating and revising a design continues until the result is satisfactory.

Although homework problems are simpler than real world design problems, within CyclePad they require that the student approach them in the manner of an engineer. Homework assignments use a variety of simplifications to scaffold students. For example, a problem might focus on the parameters of a single component or process, the cycle diagram (i.e., the "blueprint" for the cycle) might be specified as part of the problem, and economic considerations are often ignored in introductory work. Creating CyclePad as a design-oriented system has enabled its use both in introductory classes, where instructors provide the necessary scaffolding by partially specified design files, and in advanced classes, where students tackle problems that occasionally lead to published papers (cf. [45,46]).

This section describes how AI techniques are used in CyclePad to support design and analysis. We start with reasoning about the topological structure of cycles, then describe the equation formulation and constraint propagation algorithm used. Finally, the sensitivity analysis and domain-specific graphing techniques are described.

5.1. Building and modifying the structure of cycles

While much of the reasoning about thermodynamic cycles is local, there are several critical aspects of the reasoning that require global information:

- *Topological analysis*: The structural description language of steady-flow cycles permits the creation of impossible cycles. Consider the cycle in Fig. 19. The working fluid can flow into the loop formed by HTR2 → S20 → MXR4 → S19 → HTR2,

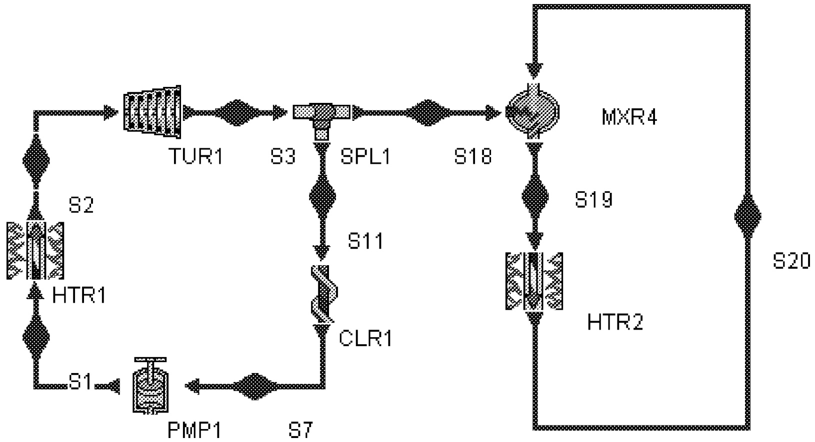


Fig. 19. An impossible cycle.

but cannot leave it, because all flows are unidirectional (note the arrowheads on the diagram). Such accumulations violate the presupposition of steady-flow analysis. A student who inadvertently creates such a cycle must be informed about it.

- *Formulation of global equations*: As noted in Section 4.3, some of the equations governing thermodynamic cycles are global, requiring knowing the structure of the cycle and the sign of particular parameter values before they can be created.

To support these global analyses, CyclePad uses two distinct modes:

- In *Build mode* the student modifies the structure of the cycle, adding or removing components or processes, changing connections, etc.
- In *Analyze mode* the student makes modeling assumptions, chooses parameter values, etc.

Global operations on the structure of the cycle are carried out during the Build → Analyze transition. For steady-flow cycles, a topological analysis is carried out to identify fluid loops. Any loops which do not contain both an inflow and an outflow are flagged as problematic and the student is so informed. The closed-world assumptions needed to formulate the global equations are also made at this time.

5.2. Constraint propagation

A design is not finished until numerical values have been chosen for its parameters. This is one reason why the overwhelming majority of thermodynamics textbook problems require numerical answers.[2] Combined with the desire to make a highly interactive system, this suggested using constraint propagation as the primary analysis technique in CyclePad. The overall structure of CyclePad was inspired in part by EL [42], an experimental system for DC analysis of analog electronic circuits. EL was one of the first systems to use constraint propagation and dependency networks to organize its reasoning,

---

[2] In a typical textbook we surveyed, 90% of the exercises required numerical answers.

and introduced the idea of dependency-directed backtracking. CyclePad's domain is more complex than EL's, because thermodynamics involves greater use of modeling assumptions than simple analog circuits, property tables must be used appropriately, and the number and variety of equations that must be dealt with is larger. Nevertheless, the constraint propagation techniques described here have been sufficient.

5.2.1. Formulating and using equations

When an equation is instantiated, it is automatically compiled into a set of constraint rules. Each constraint rule corresponds to how that equation can be solved for one of its parameters, given the rest. Every constraint rule includes [3]

- `Equation:` The equation it was derived from.
- `Sets:` The parameter it can be used to derive a value for.
- `Uses:` The parameters that must be known for the rule to be used in deriving a value.
- `Test:` A condition which must be true for the equation to be used. This is used to avoid numerical errors, e.g., division by zero.

Constraint rules are used in two ways. First, they are used for antecedent reasoning, to propagate the results of knowing the `Uses` of a rule to calculate the value for the `Sets` of a rule, assuming that the `Test` is valid. Second, they are used to generate advice about what a student might want to assume in order to derive a desired value.

Recall from Section 4.3 that equations are either local or global. The constraint rules for local equations are precomputed and compiled into procedures for efficiency, whereas the constraint rules for global equations are computed dynamically. Some global equations can be computed during the Build → Analyze transition, i.e., the equations for net work flows into and out the cycle as a whole and its subcycles. Other global equations need to be derived during constraint propagation, as the values of parameters become known. Consider for example a turbine which is not adiabatic, i.e., it can exchange heat with its surroundings. This heat flow must be taken into account when computing the heat balance of the system—if it were negligible, the turbine could be assumed adiabatic. But until a specific value is calculated for the heat flow, we cannot tell if it corresponds to a heat flow into or out of the system. Thus we cannot formulate the equations for net heat flow into or out of a cycle until such parameters are known.

5.2.2. Property table interpolation

Property tables comprise a critical source of information for CyclePad. Property tables are woven into the constraint propagator via pattern-directed rules, operating under the same protocol as the rules compiled for equations.

Due to the inherent loss of accuracy in interpolation, it is important, unlike equations, to *avoid* using tables in every logically possible fashion. Given a superheated vapor, for instance, knowing the pressure and temperature suffice to determine everything else (e.g., the specific enthalpy, specific entropy, etc.). However, CyclePad propagates all possible values, so it could in principle redundantly compute, say, the pressure, from the specific enthalpy and the temperature, and it is quite possible that the newly estimated pressure

---

[3] This is basically the organization of rules used for antecedent constraint systems [23], although implemented on top of a pattern-directed inference system for flexibility.

value will be slightly different (given the accumulated inaccuracies in the interpolation process), triggering a contradiction. Very clever interpolation techniques can minimize, but not totally vanquish, this possibility. Consequently, redundant interpolations are avoided.

Minimizing the number of interpolations also improves the accuracy of the analysis. This has two benefits. First, in complex cycles, contradictions can result from accumulated inaccuracies, even when the cycle would be consistent when analyzed differently. Second, the results of constraint propagation look more like the work a student would do, and thus the explanations provided are a better model for the student. Consider two stuffs, S1 and S2, on either side of a turbine which is assumed to be isentropic. Suppose we know the specific entropy at S1. The turbine being isentropic implies $s(S1) = s(S2)$, therefore we can derive a value for the specific entropy at S2. However, we might also know the temperature and pressure at S2, and could derive its specific entropy via table lookup. Unless there is a problem with the cycle these values will be consistent, but it may cost the user extra effort to determine this; CyclePad may display 6.60 kJ/kg at the inlet and 6.60001 kJ/kg at the outlet, for instance. We found that if CyclePad did not do the "obvious" propagation in preference to interpolation, students trusted it less.

5.2.3. *CyclePad's constraint propagator*

Constraint propagators work antecedently; given a numerical value, equations are used to derive as many other values as possible, which in turn may support the derivation of yet more values, and the process continues until quiescence. The nature of engineering thermodynamics combined with the needs of students requires a more complex algorithm. Recall from the previous two sections that

(a) some equations must be formulated dynamically,
(b) interpolations should be minimized, and
(c) derivations should serve as reasonable models for student work.

The structure of the propagation process addresses these requirements.

The propagation process services the following queues:

- `NewQuantitativeQueue:` Parameters that have received a numerical value.
- `ConstraintRuleQueue:` Constraint rules whose uses are known.
- `NewUnknownQueue:` Parameters that have lost their numerical value.
- `TableLookupQueue:` Opportunities for looking up property values via tables.

Whenever a numerical value is assumed or retracted, the parameter is added to NewQuantitativeQueue or NewUnknownQueue as appropriate, and the constraint propagator is invoked. The algorithm used is described in Fig. 20. Step 1 prevents the system from wasting time deriving information from inconsistent assumptions. The ordering of Steps 2, 3, and 4 help provide coherent problem-solving behavior and good explanations, by deriving as much as possible from a given parameter (or by working as much as possible on a newly unknown parameter) before moving to another.

Testing global equations to see if they can be used to derive values takes time linear in the number of parameters mentioned in the global equations, unlike local equations, whose constraint rules are only tested when there is some reason to believe that they have changed. Consequently it makes sense to check global equations after as much as possible has been derived from local equations. Reformulating balance equations, if necessary, is postponed until after all other analytic derivations have been exhausted, to maximize the

```
Algorithm CyclePadPropagate
 1. If design is contradictory, halt
 2. If ConstraintRuleQueue is not empty,
    2.1 Let rule = pop(ConstraintRuleQueue)
    2.2 If ¬Test(rule) go to 1
    2.3 Let P = Sets(rule) and v = result of executing rule
    2.4 ProposeNvalue(P,v,rule)
    2.5 Go to 1
 3. If NewQuantitativeQueue is not empty,
    3.1 Let P = pop(NewQuantitativeQueue)
    3.2 For each equation E using P,
        3.2.1 For each constraint rule R for E such that Uses(R) are
              all known,
        3.2.2 Push(R,ConstraintRuleQueue)
    3.3 Go to 1
 4. If NewUnknownQueue is not empty,
    4.1 Enqueue possible table lookup operations for the stuff/slice
    4.2 For each equation E using P,
        4.2.1 For each constraint rule R for E such that Uses(R) are
              all known,
        4.2.2 Push(R,ConstraintRuleQueue)
    4.3 Go to 1
 5. Check runtime equations
    5.1 For each global equation E,
        5.1.1 If no parameter mentioned in E has changed, return
              to 5.1
        5.1.2 If E can be used to derive some parameter P, ProposeN-
              value(P,v,E)
    5.2 If any values were proposed, go to 1
 6. Reformulate balance equations
    6.1 For each balance equation (heat flows in/out) in each cycle,
        6.1.1 If signs for all heat flows in C are known
              6.1.1.1 Formulate appropriate balance equation
              6.1.1.2 Add to set of global equations
    6.2 If any balance equations formulated, go to 1
 7. If TableLookupQueue is not empty,
    7.1 Let S = pop(TableLookupQueue)
    7.2 Attempt property table lookup on S
    7.3 Go to 1
```

Fig. 20. CyclePad's constraint propagation algorithm.

amount of available information and thus the likelihood at succeeding. Table lookups are relegated to Step 7 to implement the preference for analytic solutions over interpolation.

The procedure `ProposeNvalue` is described in Fig. 21. The justification installed in the LTMS is computed from the antecedent given, including both an expression for the type of domain knowledge and method used to derive it (e.g., an equation in the case of a constraint rule or global equation, or the property table used when interpolating) and

```
Algorithm ProposeNvalue(p,v,ante)
1. If Known(p)
   1.1 If CloseEnough?(Value(p),v), return
   1.2 Signal contradiction based on ante and antecedents of
       Value(p)
2. Let Value(p) = v; Reason(p) = ante
```

Fig. 21. Algorithm ProposeNvalue.

the antecedent values it relied upon. Since the antecedent values have similar justifications (or are themselves assumptions), and instantiated equations are justified in terms of the properties of the cycle that justify them, the dependency structure that is built up by the constraint propagator provides well-founded support for all numerical values in terms of the structure of the cycle and the students' assumptions about it.

To illustrate this procedure, suppose that a student is developing a model of a Rankine cycle consisting of a heater, a turbine, a cooler, and a pump (cf. Fig. 1). From the given information, the student knows the pressure at the outlet of the pump. The student assumes this value (say, 100 bar), and then assumes that the heater is isobaric. Because the outlet of the pump is connected to the inlet of the heater, this isobaric assumption causes the outlet pump pressure to propagate to the outlet of the heater (i.e., there is no change in pressure across the heater). Now the student would like to fix the state of the stuff exiting the heater, so she assumes that the exit temperature is 500 °C. At this point, CyclePad has the values of two state point variables, which enable the calculation of values for all other parameters at this state point (in this case, via a lookup operation in the steam tables).

Let's consider how `ProposeNvalue` operates during the calculation of entropy for this state point. First, a rule triggers on the existence of numeric value propositions for pressure and temperature, and installs a property-table lookup on the `TableLookupQueue`. At some point in the propagation, this queue will be examined, and this object will be passed to the appropriate table lookup function. This function will use the numerical values of pressure and temperature contained in this object to execute a table lookup, and it will then call `ProposeNvalue` on the resulting values, passing along the numerical value propositions as the antecedents.

`ProposeNvalue` will first look for known numerical value propositions. If any exist, their values are compared to the newly computed value. So long as these values lie within an established tolerance (to allow for floating point rounding errors), ProposeNvalue does nothing; the numerical value is already known by other means. If the values are not close enough, `ProposeNvalue asserts` the new proposed numerical value as an implication of its antecedents (in this case, the values for Pressure and Temperature). This new proposition will trigger a contradiction-checking rule that disallows any parameter to have more than one distinct numerical value, and the system will at that point halt and signal a contradiction to the user. The interface will display the two contradictory statements and enable the user to trace back from these statements to their root assumptions. This is possible because the antecedents are part of the LTMS dependency structure.

In many cases, there will be no already-existing values, in which case `ProposeN-value` will assert the new value as an implication of its antecedents. In this case, the two antecedents are the numerical value propositions for the Pressure and the Temperature of the stuff exiting the heater. Since they become part of the LTMS dependency structure, the user may at any time investigate the reasons for the value of entropy at the exit of the heater. In this case, the immediate reasons will be that (a) the user assumed that the Temperature at the heater exit is 500°C, and that the Pressure at the heater exit is 100 bar. This value in turn rests on the assumed pump outlet pressure and the modeling assumption that the heater is isobaric. Although many such chains of justification will be longer than in this example, it is possible via this mechanism for the user to trace the derivation of any calculated value in the system back to user assumptions.

The intricacies of CyclePad's propagator do not change its computational complexity; it remains linear in the number of parameters in the system. The number of new equations that can be formulated is fixed as a function of the number of topological cycles in the system (e.g., one equation for heat flows in and one for heat flows out for each cycle).

5.3. Sensitivity analysis

Exploring the sensitivity of a derived value, such as the cycle's thermal efficiency, to a particular numerical assumption is an important activity for developing intuitions about thermodynamics. Although sensitivity analyses (often called "trade-off studies") play a central role in engineering design and their educational value is unquestioned, they are rarely assigned because they are so tedious to perform. Consequently, CyclePad provides tools for carrying out such analyses.

There are two subtleties with sensitivity analysis. First, there is ascertaining just how the dependent parameter actually depends on the independent parameter. When working with algebraic models, this is typically done for small examples by deriving an expression for the dependent parameter in terms of the independent parameter, but in large examples the complexity of the algebra can make this approach intractable. For complex systems, numerical sensitivity analyses are now routine in engineering practice (the engineering equivalent of "What if" analyses in financial spreadsheets).

The use of numerical techniques leads to the second subtlety; what happens if the range of an analysis steps outside the bounds over which the model is applicable? The principle way this happens in thermodynamics is when phase boundaries are inadvertently crossed. CyclePad addresses these subtleties by providing two sensitivity analysis tools.

Slow but sure. This tool incrementally retracts and reassumes the independent parameter, mapping it over the user-specified range. This method captures phase transitions accurately. It also detects if a choice for the independent value leads to a contradiction, and does not report the value of the dependent parameter for that choice. Unfortunately, this is simply too slow for most sensitivity analyses, as it requires the recalculation of all derived values for the cycle each time the independent value is changed. Although fact garbage collection makes it possible, the overhead entailed in deleting and reestablishing clauses is high enough that this option is rarely used.

Quick but heuristic. This tool uses the dependency network to construct an algebraic expression for the independent parameter in terms of the dependent parameter. The

Inputs: Dependent parameter `D`, Independent parameter `I`.
Output: Algebraic expression for `D` in terms of `I`
Assumes:

- `DerivingExpression(P)` = algebraic expression used to derive value for P, calculated using the dependency network.
- `ParametersOf(E)` = set of parameters occurring in `E`.
- Results of table interpolations are expressed as a special kind of algebraic equation.
- `Subst(x,y,z)` = traditional substitution operator.
- `DependsOnParameter(E,P)` is true when `E` depends on the value of `P`. This is calculated using the dependency network.
- `EquationToExpression(E,P)` rewrites equation `E` into an algebraic expression which computes `P`. `P` must be mentioned in `E`.

```
Algorithm DeriveExpressionIn(D,I)
(1) If D = I, return I
(2) Let E = EquationToExpression(DerivingExpression(I),I)
(3) For each P in ParametersOf(E)
    3.1 If DependsOnParameter(Nvalue(P),D) then
        Subst(DeriveExpressionIn(P,D),P,E)
    3.2 Otherwise, Subst(Nvalue(P),P,E)
(4) Return E
```

Fig. 22. Constructing expressions for sensitivity analysis.

algorithm for constructing the algebraic expression is illustrated in Fig. 22. It exploits the insight that, although the algebraic expression for any parameter is potentially huge, those subexpressions which do not depend on the independent parameter can be treated as numerical constants. This minimizes the complexity of the expression generated, making sensitivity analysis very efficient even for complex cycles. The drawback, of course, is that its results can be misleading if the chosen parameter range causes phase changes or leads to contradictions. Our solution to this problem is pedagogical, not technological; we encourage students to use sensitivity analysis as a rough guide, but to try out particularly interesting values themselves to ensure that they are consistent. That way, students have access to the usual analysis and explanation tools in CyclePad to explore their hypotheses, rather than replicating this functionality in the sensitivity analysis system.

5.4. Domain-specific graphing

Graphical representations are an important means of describing, as well as analyzing phenomena, and the field of thermodynamics has evolved several commonly used representations to describe cycles, including the enthalpy-entropy, or Mollier, diagram, the temperature-entropy diagram, and the pressure-volume diagram. Of these, the temperature-entropy, or TS, diagram, is the most commonly used, so CyclePad has the capability to generate such graphs corresponding to a student's design.

A TS diagram for a simple Rankine cycle is shown in Fig. 23. The bell-shaped curve is called the saturation line. At all points within it the working fluid will be saturated. To the left and above the line, the working fluid is liquid, while to the right and above it the

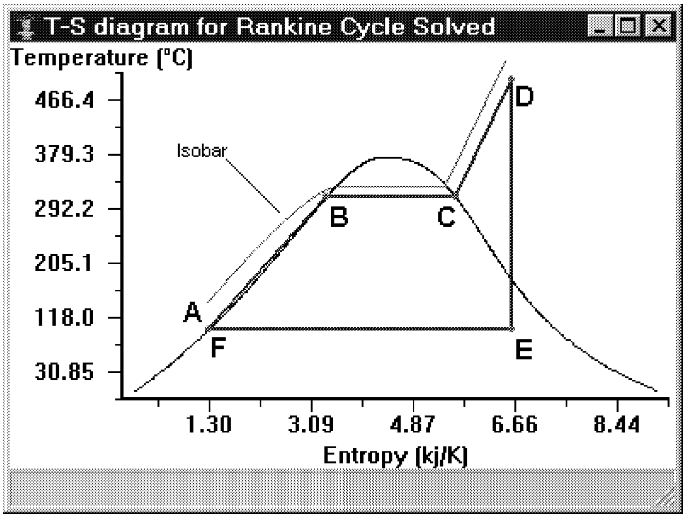


Fig. 23. An automatically generated temperature/entropy diagram.

working fluid is gas. The apex of the curve is called the critical point; at pressures higher than this, the working fluid does not go through a phase transition between liquid and gas, but simply behaves as a fluid. Contours of constant pressure, called isobars, are of particular interest, as heating and cooling processes tend to move along them.

The lines in the diagram connecting points A through F illustrate the operation of the Rankine cycle. The working fluid exits the pump and enters the boiler with relatively low temperature and entropy, at point A in the lower left of the diagram. The fluid then rises in temperature as heat is applied in the boiler, until it reaches the saturation point for the pressure of the boiler (in other words, it moves along the isobaric segment AB that closely tracks the left flank of the saturation line). At the saturation point, however, the working fluid starts to change phase from liquid to gas, and no longer increases in temperature. This process is represented on the diagram by the upper horizontal line that traverses the region under the saturation line from points B to C. When this line intersects the right side of the saturation curve the working fluid has been complete vaporized, and its temperature once again starts to rise, to point D in the upper right.

At this point the working fluid enters the turbine, where it expands and its temperature drops precipitously, as indicated by the vertical line running from D to E. Ideally, this expansion is accomplished with no change in entropy (hence the vertical line segment DE), but in practice there would be some rise in entropy, causing point E to lie to the right of point D.

Notice that the vertical descent ends within the saturation region, indicating that the fluid leaving the turbine has started to condense back into liquid. Too much condensation in the back stages of a turbine can cause damage, a situation that would be readily apparent from this diagram.

Finally, the water is condensed back into a liquid, moving from point E to point F. Despite appearances, this is not quite where we started from, as we have yet to account for the state transformation that occurs within the pump. In fact, points F and A, although they appear to be a single point, are separated vertically by about 0.5°C, which is the rise in temperature attributable to the action of the pump. The pump moves the working fluid across isobars, and at this point in the diagram the isobaric contours are quite close together. If the working fluid were a gas, we would see a large increase in temperature as the pressure on the fluid increased.

Movement from state to state on this graph is constrained by the physics of the domain. Compression and expansion processes cross isobars (i.e., the pressure changes during these processes), enabling upward and downward movement, whereas heating and cooling processes typically move along isobars (although there may be some change in pressure as well). Traversal of the saturation region is isothermal (i.e., horizontal) unless the pressure changes substantially, which is quite hard to do in practice.

Generating this graph therefore requires knowledge of the domain. We can extract the temperature and entropy values for points A (boiler inlet), D (turbine inlet), E (condenser inlet), and F (pump inlet) directly from the LTRE's database of propositions, but simply graphing these points would produce the triangle ADE. We also need to check for changes in pressure and phase changes between each pair of points and insert new points (e.g., B and C) where we find such changes.

Our algorithm checks for all six pairwise combinations of the three phases (liquid, saturated, and gas) that could occur between the pair of points. To illustrate, suppose we are processing points A and D, which correspond to the entry and exit points of the cycle's boiler. In this case, the working fluid is being boiled, then superheated. We use the pressure at point A to determine, via a thermodynamic property table lookup, the temperature at which the liquid will boil. This provides the y-coordinate of the point at which the liquid will cross the left flank of the saturation line. The x-coordinate of this point is the entropy of the working fluid given its pressure and the fact that we know it is saturated liquid (i.e., the point lies on the left flank of the saturation line). Together, these coordinates specify point B in Fig. 23. We insert this point between the original pair of points. Because we have now entered the saturation region, we have to insert a second point on the right flank of the saturation line before we can move directly to D. This point, C in Fig. 23, will have the same temperature as point B, since we are traversing the saturated region.

6. Explanations

Explanations are the heart of articulate software. An innovation developed for CyclePad is the idea of *structured explanation systems*, an abstract layer between the reasoning system and interface that mediates explanation generation. We first describe the idea of structured explanation systems, and then outline how CyclePad's implementation of this idea works.

6.1. Structured explanation systems

Explanation generation is a very complex task (cf. [30,37]). A major source of complexity is that it requires strong interactions between two systems that operate under very different constraints:

- Reasoning systems need to be optimized for efficient inference. This often means leaving implicit information that is not changing over the course of the program's operation, and minimizing information about antecedents in a dependency system to the bare minimum needed for correct operation.
- Interfaces need to be optimized for communication with the user. This often means including background information that isn't needed by the reasoning system, and providing explanations that are conceptually clear and easy to explore.

Obviously these constraints are in conflict. Our solution is to introduce a mediating layer, a *structured explanation system*, which reifies the information in the reasoning system in a manner that simplifies creating good explanations in the interface. Structured explanation systems use two kinds of techniques in this mediation:

- *Suppression techniques* hide information found in the dependency network that is either implementation-specific or is irrelevant to users.
- *Reintroduction techniques* introduce explicit items and dependencies representing information that is needed to understand the system's results but that are not found in the dependency network.

The datastructures in a structured explanation system consist of *e-propositions* and *e-arguments*. E-propositions correspond to the statements in the domain of discussion, and are identified with elements of the underlying reasoning system. Each e-proposition is justified by exactly one e-argument. E-arguments can refer to other e-propositions, aside from the distinguished e-proposition they support. This structure is analogous to the node/definite clause structure of a JTMS [23], but differs from it in two important ways. First, unlike TMS structures, e-propositions and e-arguments in structured explanation systems do not persist over time. Second, e-propositions and e-arguments are typed objects, based on the semantics of the domain being explained. This typing enables the definition of the explanation system in terms of generic operations (e.g., finding antecedents, generating hypertext or field values) that behave appropriately for the entities in a domain and the types of arguments used in it. This provides a natural way to implement presentation-based interactions [10]. Specifically, they provide the basis for hypertext generation protocols, enabling the dynamic creation of content from the reasoning system's operations.

Conceptually, the structured explanation system can be thought of as managing a dynamically generated rational reconstruction of the knowledge content of the dependency network in the underlying reasoning system. They provide a natural foundation for creating automatically generated hypertexts. To illustrate, we describe CyclePad's structured explanation system next.

6.2. *CyclePad's structured explanation system*

The basic service provided by CyclePad's structured explanation system is the generation of hypertext in response to user queries. The starting points for user queries are GUI elements in the cycle design, i.e., clicking on components or on parameters in meters.

The types of e-propositions used in CyclePad are illustrated in Fig. 24. For each type of e-proposition, the following two methods are defined:

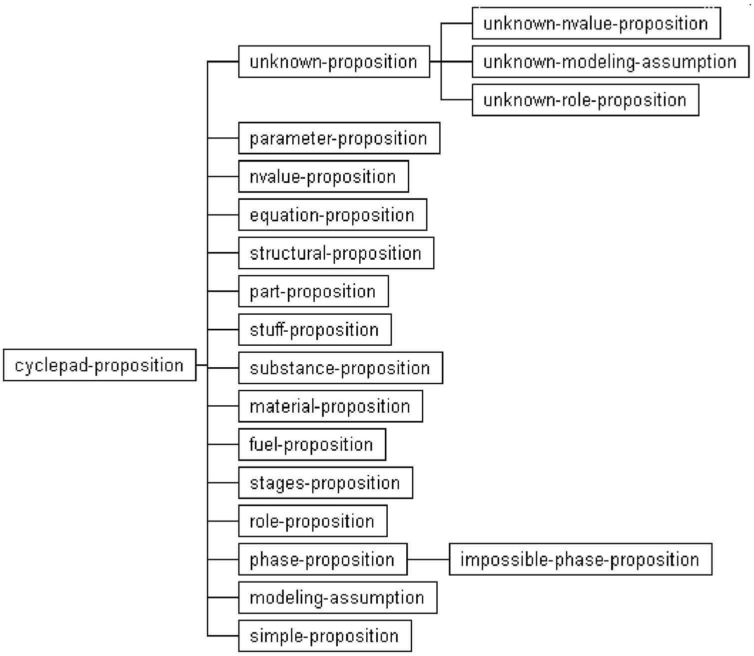


Fig. 24. E-propositions used in CyclePad's structured explanation system.

- `QuestionsFor(`<*e-proposition*>,<*design*>`)` returns a list of questions that are appropriate for <*e-proposition*> given the state of <*design*>. Each entry in this list contains a string suitable for inclusion in a menu or web page describing the English content of the question (e.g., "Why is S1 superheated?") and a symbol indicating the category of question (e.g.,:WHY).
- `FindAnswer(`<*e-proposition*>,<*question*>,<*design*>`)` creates an e-argument that corresponds to asking <*question*> about <*e-proposition*> in the context of <*design*>. For example, one might ask how a parameter was derived, which will cause the generation of a `numerical-value-via-equation, numerical-value-via-table`, or `numerical-value-via-propagation` e-argument, depending on the information recorded in the LTMS about how it was derived.

The inclusion of the design as an argument provides full access to the LTRE for that design. This means that the list of questions can include commands, e.g., for assuming or retracting numerical values.

For each type of e-argument, there are methods for producing text, hypertext, or values suitable for displaying fields (selectable or not) that produce on the appropriate interface medium the contents of the e-argument. The types of e-arguments used in CyclePad's structured explanation system are illustrated in Fig. 25.

The importance of the abstraction layer provided by structured explanations can be seen by considering an example. The argument type `numerical-value-via-table` explains how a numerical value is derived via property-table interpolation. The

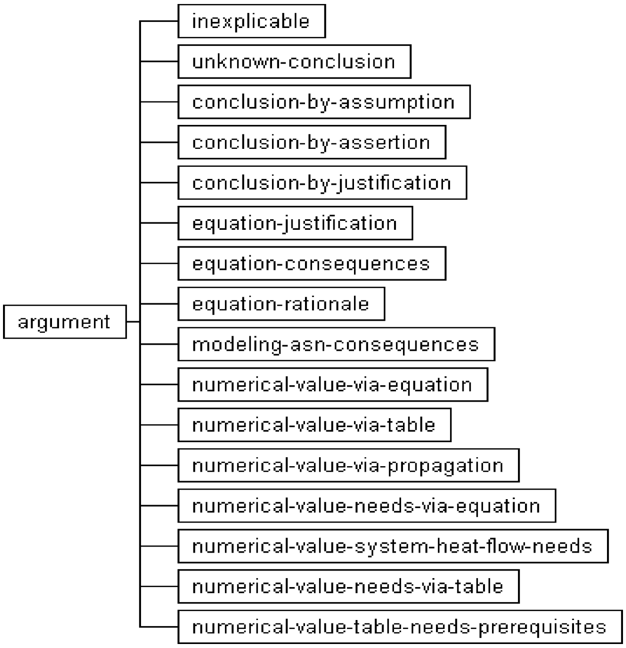


Fig. 25. E-arguments used in CyclePad's structured explanation system.

logical dependency for a numerical value derived from a table in the LTMS includes an assertion that the state point is either a thermodynamic stuff or slice. This detail of CyclePad's representation is irrelevant to the student, and is therefore hidden. This hiding is accomplished by a *"can't say, don't tell"* policy. Given any proposition in the database, CyclePad can determine whether or not that proposition can be stated in English for the user. Any antecedent that cannot be so articulated is not included in the antecedents for the argument. When appropriate, the antecedents for the silent antecedent are themselves promoted for potential inclusion. (Premises that are inarticulate are simply left out.) This recursive process ensures that no aspect of the dependency structure is left out of the explanation, if there is some way to state it.

We found the structured explanation system to be extremely helpful in developing CyclePad, since it allowed us to optimize the content of explanations separately from the means of displaying them. For example, early in the development process only textual displays were available, whereas ultimately the software ended up tightly integrated into the graphical Windows environment.[4] However, most of the explanation code remained unchanged through the user interface evolution, which went from Macintosh Common Lisp to Lucid Common Lisp under UNIX before settling into the Windows world.

---

[4] This was made possible by Franz, Inc.'s commercial Lisp environment, which provides a civilized interface building environment in Lisp that makes extensive use of the native Microsoft Windows environment.

7. Coaching

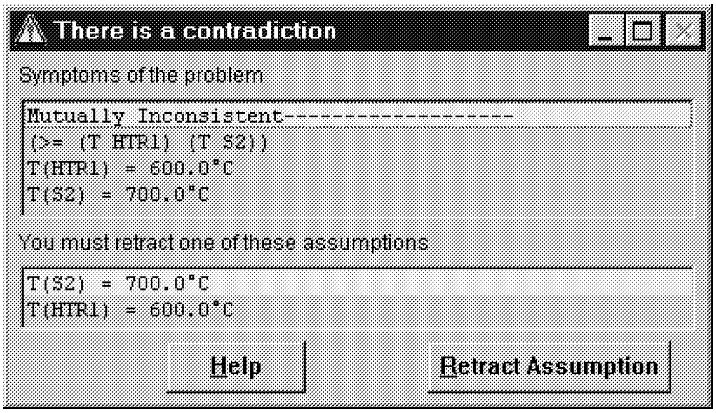
Coaching helps students understand how to operate in a domain, both in terms of problem-solving skills and in relating their specific situation to the broader context of thermodynamics. By virtue of its increased conceptual understanding of the domain, articulate software can provide novel types of coaching. This section describes CyclePad's coaching facilities, including how it helps students resolve contradictions and makes suggestions about parameter values based on an understanding of the intended teleology of the system. We also describe a distributed coaching facility that provides more sophisticated help with analysis and design, using an email-based agent and a web-based design library.

7.1. Helping students understand contradictions

As soon as CyclePad detects a contradiction (e.g., the set of propositions in memory lead to two different values for a given state point parameter), it halts and informs the student that a contradiction has occurred. The set of contradictory facts is displayed, along with all assumptions underlying these facts, as in Fig. 26. The student must resolve the contradiction by retracting one of the implicated assumptions.

However, students often have difficulty understanding why a set of assumptions is contradictory. The generative hypertext created by the structured explanation system provides a tool for browsing the chains of reasoning that lead to the contradiction. It also is useful in examining the consequences of the contradictory assumptions, to figure out which of them might be better to retract.

CyclePad's truth-maintenance system uses a stack of contradiction handlers [23]. This enables additional handlers to be created to handle particular classes of contradictions. For example, a common source of problems is choosing parameter values outside property table bounds. The handler for this case presents a table-boundary diagram and a dot



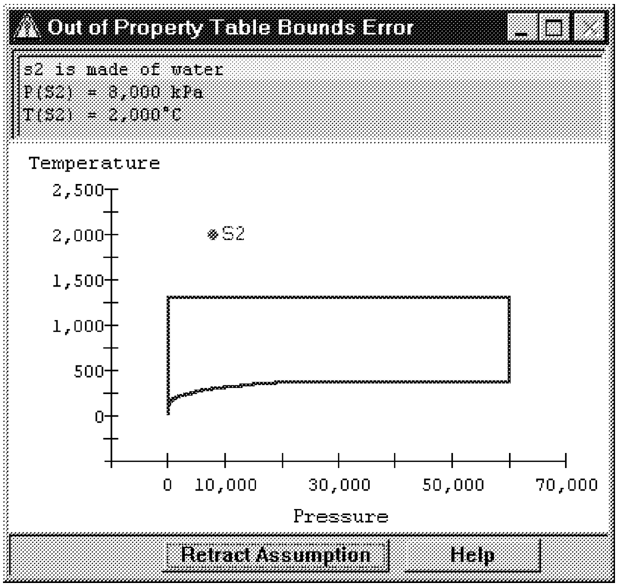
Symptoms of the problem

```
Mutually Inconsistent------------------
(>= (T HTR1) (T S2))
T(HTR1) = 600.0°C
T(S2) = 700.0°C
```

You must retract one of these assumptions

```
T(S2) = 700.0°C
T(HTR1) = 600.0°C
```

Fig. 26. Contradictions must be resolved as soon as they occur.



s2 is made of water
P(S2) = 8,000 kPa
T(S2) = 2,000°C

Fig. 27. Special cases of contradictions can be recognized.

showing the location of the out-of-bounds value, in addition to the usual information in the contradiction dialog, as Fig. 27 illustrates.

7.2. The role of teleology

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. 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 hoping for low quality. CyclePad incorporates Everett's CARNOT teleological recognition system [14] 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 [15]. This was far too slow to be deployed. Also, it often produced a plethora of 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. 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 low-end 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] that 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 also get an explanation of why other potential roles were rejected.

7.3. Understanding the meaning of parameters

CyclePad 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 300–600 °C. 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. In contrast, the range of temperatures for the refrigerant leaving the coils of a refrigerator (which are modeled as a heater) is quite different, typically in the range of 5–15 °C. Inferring the role a component is playing is therefore essential to providing relevant advice to the student. 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 on-board advice, CARNOT'S teleological representations also play an important role in design coaching (see Section 7.5).

7.4. Distributed coaching

CyclePad is publicly available via the web. This has led to a geographically distributed user base of uncertain size. Counting classrooms where we have direct knowledge (i.e., classes being taught by collaborators or that we physically visit), we know we have at least

---

[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.

80 students using CyclePad per year. Based on informal samples, there are at least (for 1998) 600 more students. Systematic follow-up is difficult, since there have been 2593 distinct downloads[6] from 63 countries between April 1997 and September 1999.

This highly distributed user base leads to several problems. Adding new coaching facilities to the software requires distributing upgrades. For Northwestern students, we can easily install upgraded versions in the local labs during an academic quarter. However, instructors at remote sites are typically unwilling to download and install upgrades in labs and local distribution sites once their class starts. To assess CyclePad's educational impact, we would like to gather more data about what students and instructors are doing with CyclePad. We currently work with instructors from several remote sites to ensure robustness, but expanding that pool is difficult. For instance, classroom adoptions include a power plant course at Rutgers (NJ, 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. This poses a dilemma for gathering the data needed by the educational aspect of this research.

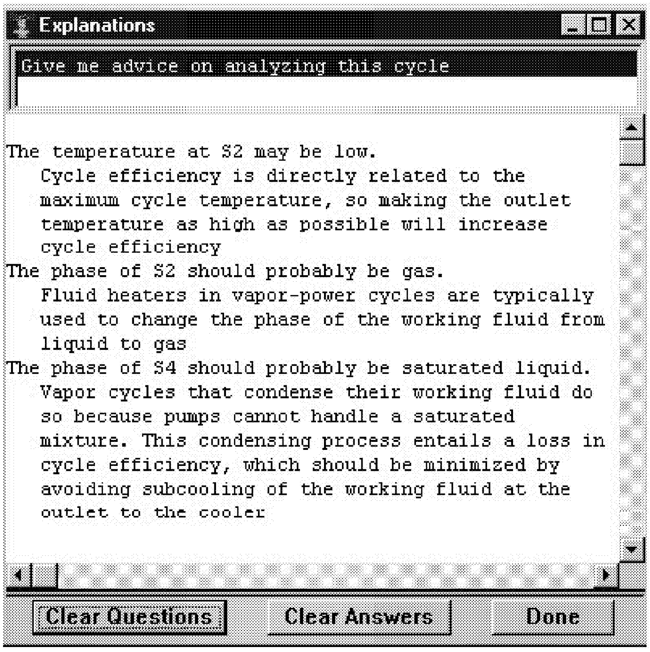
As our user base expanded and instructors started relying on CyclePad more, another dilemma arose. Feedback from our users indicates that they would like more coaching facilities. However, they do not want to see memory requirements rise, and instructors who rely on it daily are adamant about keeping it stable.

Our solution to these dilemmas was to make the CyclePad coaching architecture distributed. The facilities described above remain embedded in the software. All new coaching facilities are accessed via email, sent to a software coaching agent, the *CyclePad Guru*. The email is sent via CyclePad, so that both the student's request and the design that is the subject of the request are included. (The requests the student can make are limited, so as to be understandable by the Guru, using a form-based interface. Students can include natural language comments on the side, which the developers may read, but the Guru does not.) The email goes to the RoboTA Agent Colony [22] at Northwestern, which passes all CyclePad-related queries on to the CyclePad Guru. The Guru processes the student's request, and uses the Post Office facilities in RoboTA to send a response to the student 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.

---

[6] By distinct, we mean that we count each email address once, even though the same users download the system multiple times, as upgrades become available. This only counts downloads from our web site; we know that mirror sites exist, and most faculty we know download it once and redistribute it to their class locally. The breakdown of users, based on self-reports when downloading, works out to be 1,200 students, 426 professors, 598 engineers, 19 corporate trainers, and 349 others. (Based on informal sampling, the others include students interested in science fair projects, hobbyists, and those with a broad curiosity streak.)



```
Give me advice on analyzing this cycle

The temperature at S2 may be low.
   Cycle efficiency is directly related to the
   maximum cycle temperature, so making the outlet
   temperature as high as possible will increase
   cycle efficiency
The phase of S2 should probably be gas.
   Fluid heaters in vapor-power cycles are typically
   used to change the phase of the working fluid from
   liquid to gas
The phase of S4 should probably be saturated liquid.
   Vapor cycles that condense their working fluid do
   so because pumps cannot handle a saturated
   mixture. This condensing process entails a loss in
   cycle efficiency, which should be minimized by
   avoiding subcooling of the working fluid at the
   outlet to the cooler
```

Fig. 28. CARNOT's teleological reasoning is used to provide advice about parameter values.

The rest of this section outlines the kinds of help provided in the CyclePad Guru. The case-based coaching is sufficiently novel that it merits discussion in a separate section, which follows.

7.4.1. 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. CyclePad provides on-demand coaching facilities that students may invoke at their discretion.

The onboard CARNOT-based Analysis Coach is the student's first resort, providing advice linked to the intended purpose of the cycle. For example, students can have CyclePad display the roles inferred for each device, which can provide useful feedback, particularly if the system believes a mixer is acting as a flow-join rather than the open heat-exchanger the student intends it to be. Norms can help students focus on problematic parts of their cycle models.

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 choosing a working fluid 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 canned text that espouses general principles, but does not make reference to the specifics of the student's problem.[7]

7.4.2. Design help

The Guru also accepts questions about design optimization, i.e., "how can I *increase/decrease* the <*parameter name*> of <*part*>?". Examples include increasing the efficiency of the whole cycle, decreasing the operating cost of the cycle, or increasing the quality of steam exiting a particular turbine. We have two goals in giving design advice. First, we want to nudge students in useful directions, rather than solving problems for them. Consequently, the Guru provides plausible specific suggestions, but does not attempt to validate those suggestions in the students' context. Understanding why a suggestion will or will not work in a particular circumstance is an important learning experience that we want students to have. Second, we want to motivate students to dig more deeply into the nature of thermodynamics—ideally, to immerse themselves in the culture of engineering thermodynamics by studying real-world systems and how they are connected to the assignments they are grappling with. Consequently, the Guru uses case-based coaching to generate design advice, motivating the student to explore the case more deeply by showing exactly how it might be relevant to the improvement they are trying to make.

7.5. Coaching via analogy

Advice for design requests is generated by a case-based coach, using cognitively-motivated analogical processing techniques [43]. Case-based coaching (cf. [34,41]) 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 [21] to retrieve cases relevant to a student's design.

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

---

[7] Hence the name "Guru".

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) [20], an analogical matcher based on Gentner's structure-mapping theory [26]. SME compares each case produced by MAC to the student's design and returns the best structural match, 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 (cf. [27,28]) to our knowledge this is their first application in a fielded system.

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 (as computed by CARNOT) 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. 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. Fig. 29 shows the candidate inferences when the Guru, given a Rankine Cycle, is reminded of a plan to increase the boiler temperature and to add reheat.

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 Fig. 29 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 [24]. Finally, at most 2 suggestions are selected to serve as the basis for design advice, so that students are not deluged with advice. Sometimes the

```
Suggestions for <Desc WM of Rankine Cycle>:
Suggestion <Use INCREASE-RANKINE-BOILER-T>:
  1 step
  support=.085 extrapolation = 0.66 ;; Support scores provide estimate
  normalized = 0.45 overlap =.408 ;; of quality of advice.
  combined =.944
<Mapping 153 Candidate Inferences>
(BOILER htr1)
(CONDENSER clr1)
(IMPLIES (AND (TURBINE tur1 s2 s3) ;; Applicability condition for this
              (HEATER htr1 s1 s2)) ;; suggestion
  (APPLICABLE (:SKOLEM:dsn-tr)))
(TRANSFORMATION-OF (:SKOLEM:dsn-tr)
  (STEPS (ASSIGN (T s2) (:SKOLEM:+)))) ;; Qualitative value increase
Suggestion <Use REHEAT-RANKINE-CYCLE>:
  16 steps
  support=0.03 extrapolation =.846 ;; Shorter plans are preferred over
  normalized =.404 overlap =.134 ;; longer plans
  combined =.567
<Mapping 172 Candidate Inferences>
(BOILER htr1)
(CONDENSER clr1)
(IMPLIES (AND (TURBINE tur1 s2 s3) ;; Applicability condition for this
              (COOLER clr1 s3 s4)) ;; suggestion
  (APPLICABLE (:SKOLEM:dsn-tr)))
(TRANSFORMATION-OF (:SKOLEM:dsn-tr)
  (STEPS (DISCONNECT (OUT tur1) (IN clr1) s3) ;; Removes stuff joining them
    (INSERT-DEVICE (:SKOLEM heater) ;; Skolems for device types replaced
                   (:SKOLEM htr2)) ;; with same type, since these are
    (CONNECT (OUT tur1) (IN (:SKOLEM htr2)) ;; within-domain analogies.
             (:SKOLEM s5))
    (INSERT-DEVICE (:SKOLEM turbine) ;; Skolems for device names lead to
                   (:SKOLEM tur2)) ;; introduction of unique new device.
    (CONNECT (OUT (:SKOLEM htr2))
             (IN (:SKOLEM tur2))
             (:SKOLEM s6))
    (CONNECT (OUT (:SKOLEM tur2)) (IN clr1)
             (:SKOLEM s7))
    (INVOKE-ASN (SATURATED (:SKOLEM s5))) ;; Non-structural assumptions
    (ASSIGN (DRYNESS (:SKOLEM s5)) ;; required to complete analysis
            (:SKOLEM 1.0)) ;; are also included as a source
    (INVOKE-ASN (REHEATER (:SKOLEM htr2))) ;; of issues that the student
    (INVOKE-ASN (ISOBARIC (:SKOLEM htr2))) ;; needs to consider.
    (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))))
```

Fig. 29. Candidate inferences for the retrieved cases.

initial MAC/FAC results lead to fewer than two suggestions, because the case turned out to not be applicable to the student's design. In this case MAC/FAC is used iteratively, with the previously retrieved cases temporarily removed from the case library. To avoid exhaustive search, we place a tight bound on the number of iterations permitted, usually two. This works well because MAC/FAC does a reasonable job at picking out relevant cases.

Fig. 30 shows the advice generated from the candidate inferences in Fig. 29. 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, Fig. 30 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.

A case compiler automatically generates new cases in the Guru's design library. 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.

Our case library currently consists of 13 cases, averaging 47 expressions involving 22 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 iteration. Second, SME uses a polynomial-time greedy algorithm in its merge step, making its overall complexity quadratic in the size of descriptions compared [20].

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

```
From: robota@godzilla.cs.nwu.edu
Date: Mon, 20 Sep 99 00:49:47 -0600
To: forbus@nwu.edu
Subject: The CyclePad Guru’s response to your message: Need
help improving my Rankine cycle

You asked for help with your design.
I have 2 suggestions.
=============================================================
Suggestion #1
Your problem reminds me of a method: increasing boiler tem-
perature in a Rankine cycle. Increasing the boiler temperature
increases the efficiency of the cycle.
You can find out more about this at <URL>.
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 cy-
cle. 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 <URL>.
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]
=============================================================
```

Fig. 30. Design advice from the CyclePad Guru.

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 tips [8].

8. Discussion

CyclePad has moved out of the laboratory and into routine use in classrooms scattered all over the world. Starting in 1995, collaborators at Northwestern University and the US Naval Academy started using it with their students on an experimental basis. As it became stable, we started distributing it via the Web (September 1997). At this writing (September, 1999) we get on the average four downloads per day. Some of whom no doubt never use the software or try it a few times and find it not to their liking, but some of whom have turned out to adopt and adapt it to their teaching and learning. As mentioned in Section 7.4, we have had over 2593 distinct downloads from 63 countries from our web site alone. The foundation for distributed case-based coaching, in the form of RoboTA and the CyclePad Guru, was in development and testing for eighteen months, and has been available to students and instructors continuously since the end of 1998. CyclePad will continue to be available as a set of Windows binaries, and the source will be made publicly available through an open-source license in 2000, so that its evolution can be taken over by the thermodynamics community. While the project is not yet over (see below), we have already learned enough to draw some conclusions.

8.1. The value of articulate software for education

Our hypothesis was that articulate software—software that uses AI techniques to understand the domain being learned, so it can provide better scaffolding to learners—could be built. We think that CyclePad provides strong evidence for this hypothesis. Moreover, there is mounting evidence (cf. [4–6]) that CyclePad can be valuable in education, including:

- CyclePad has been adopted by instructors in a variety of educational institutions for both introductory and advanced courses. Some institutions use it with traditional textbooks, while others are developing new curricula around it.
- Advanced thermodynamics students at the US Naval Academy were able to tackle more complex term projects than they were able to previously, resulting in some cases in publishable technical papers (cf. [45,46]).

The design approach fits quite naturally into advanced thermodynamics courses. Although in the US there is currently strong interest in using design tasks throughout the engineering curriculum, this has not been the practice in introductory courses. Traditional thermodynamics curricula are often analysis-centered, lavishing classroom time on mathematical derivations of thermodynamic principles at the expense of helping students understand the principles themselves and their implications. Many courses still focus on teaching students how to do complex analyses, including table interpolations, with just a simple calculator, even though practicing engineers tend to have more sophisticated computer supports these days. The work that students typically do in these classes often results in learning “formula picking” strategies rather than developing an understanding of

(a) how thermodynamic principles relate to both the parts and the whole of a system, and
(b) learning real-world design skills such as optimizing designs or working within environmental constraints.

Introducing CyclePad into such courses can lead to a drop in student performance, since students are being tested on mechanical calculation skills that in practice are automated. One of the major problems we have found for instructors using CyclePad is incorporating a design orientation into introductory courses [7].

This problem is analogous to the introduction of calculators in mathematics education, where "drill and practice" on tasks that could be done automatically gave way to problems that put these calculations to work in broader contexts. To gain the full potential benefits of CyclePad will require rethinking what is taught in engineering thermodynamics. We are currently carrying out such experiments. For example, we piloted an intervention at Northwestern in the spring of 1999. The goal of the intervention was to integrate CyclePad with the introductory course material so that students had a seamless experience of both learning thermodynamics and the CyclePad software for the first time. In doing so, we supported novices in learning basic thermodynamics concepts, vocabulary, relationships and laws through doing CyclePad-based problems alongside the traditional textbook problems. In using CyclePad, students were able to go beyond the textbook-type problem and ask pertinent "why" and "how" questions about systems. For example, in one CyclePad problem, students were asked to find the temperature of a specific state point. They were next asked to explain why it was the same as the initial starting temperature. To do this, students entered CyclePad's explanation system to learn what formulas and assumptions were involved in the calculation of the two temperatures. A second experiment is being carried out in the fall of 1999, and should lead to some guidelines for curriculum designers who want to incorporate articulate software.

One community for whom CyclePad is proving to be especially valuable is what in the US is called *engineering technology* education, i.e., curricula aimed at producing technicians rather than engineers. An analysis-heavy approach is not feasible with such students, who often do not learn calculus until late in their education (if at all) and who find complex algebraic manipulations to be difficult. CyclePad provides a "simulated hands-on" experience for such students, helping them build solid, accurate intuitions about thermodynamics. For example, at University of Arkansas, Little Rock, students use CyclePad in laboratory exercises to experiment with systems that would be too expensive or dangerous to physically build.

The articulate virtual laboratory architecture CyclePad embodies can, we believe, be fruitfully applied to many other engineering domains. The nature of the analysis tools will vary from domain to domain. AVLs for electronics or chemical engineering might end up looking very much like CyclePad, whereas AVLs for mechanism design or computer programming might be able to utilize similar structured explanation systems and distributed coaching, but with very different analysis and design methods. With appropriately simplified domains, AVLs could also be used in science teaching; learning how animals work by designing them could be a highly motivating task, but existing software relies on simple case-based reasoning [11], and often provides minimal explanations and limited generativity.

8.2. Technical innovations and surprises

CyclePad relies on a synthesis of existing AI techniques. However, it does incorporate several extensions to the state of the art:

- The idea of structured explanation systems started with self-explanatory simulators [19], which constructed the entire explanation system and its contents at simulator compilation time. As Section 6.2 discussed, CyclePad demonstrates that structured explanation systems can also be productively used with dynamically generated inferences. The closest precursors are presentation-based interfaces [10] and automatic hypertext generators [37].
- Everett's teleological reasoning theory [14] was developed to support coaching in CyclePad, and as described in Section 7.4, provides the basis for generating advice about parameter values and in helping the design coach come up with relevant cases.
- CyclePad provides a robust demonstration that the ideas of compositional modeling [17,18] can be used in analyzing complex systems, including large quantitative knowledge bases (Section 4).
- Techniques for fact-level garbage collection in truth-maintenance systems [16] were invented to support CyclePad (Section 3).

Some of our expectations about what technologies would be appropriate, or how they would be used, turned out to be inaccurate. Some surprises were:

- Automatic model formulation, the primary focus of most research on compositional modeling, is irrelevant for this task. What is important about compositional modeling for this task is the ability to explicitly state and reason about modeling assumptions—something we want to educate the user in doing, since understanding their appropriateness requires the deep understanding of thermodynamics that we want them to have.
- Qualitative simulation was irrelevant for this task. Qualitative reasoning about constraints imposed by physical processes, on the other hand, proved to be a crucial source of knowledge for detecting physically impossible choices for parameters.
- Teleological reasoning proved to be essential, since it provides the context for many kinds of advice.
- As suggested by [39], sophisticated natural language generation techniques were inappropriate for CyclePad. The ability to automatically generate hypertext in response to a user's questions obviates the need for discourse planning, and the fixed nature of the task means that issues such as selecting the appropriate level of detail in an explanation are less pressing. Hypertext allows users to select how much they want to know about a topic, and since the hypertext is only generated on demand, many navigation problems common in fixed hypertexts are ameliorated.
- Some domain-specific extensions (i.e., table interpolation and automatic construction of global equations) to simple constraint propagation provided sufficient algebraic and numerical capabilities. The commercial world has developed many powerful symbolic algebra packages, such as Mathematica, Maple, and Macsyma, and it has been argued that AI software should rely on commercial software for such services. However, relying on such commercial products would have increased the size and complexity

of CyclePad, and made it impossible to distribute it via the web, which has been crucial for our experiments.

We suspect that some version of these surprises will hold for many future articulate educational software systems. For example, we suspect that reasoning about models will be as important as automatic model formulation, if not more so. Qualitative simulation, as traditionally practiced, will only rarely be useful, although other forms of qualitative reasoning will continue to prove essential. Teleology will be increasingly important for articulate software that places the phenomena under study into broader contexts, because function provides the link between phenomena and intentions. Structured explanation systems, dynamic hypertexts, and constraint propagation will be a “sweet spot” for articulate educational software.

8.3. Learner support systems

We are also working on an additional incentive for instructors to send us data: helping them with grading. We have deployed a grading support system to the CyclePad/RoboTA system, designed with extensive input from our instructor-collaborators. In this system, 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

The basic research on qualitative physics and compositional modeling is supported by the Artificial Intelligence Program of the Office of Naval Research. The basic research on analogical matching and retrieval is supported by the Cognitive Science Division of the Office of Naval Research. The initial development of CyclePad was supported in part by a grant from the Science and Engineering Research Council in the UK. Research on articulate virtual laboratories is supported by the Advanced Applications of Technology program of the EHR directorate of the National Science Foundation. The collaboration with the US Naval Academy was made possible by support from the Cognitive Science Division of the Office of Naval Research. Other subject-matter collaborators and instructors providing formative feedback include Dr. Siavash Sohrab and Dr. Art Kovitz (Northwestern), and Dr. C. Wu, and Dr. Sheila Palmer (USNA).

References

[1] Accreditation Board for Engineering and Technology, 1997. Engineering criteria 2000.http://www.abet.org/EAC/eac2000.html.

[2] S. Addanki, R. Cremonini, J.S. Penberthy, Reasoning about assumptions in graphs of models, in: Proc. IJCAI-89, Detroit, MI, 1989.

[3] J.R. Anderson, A.T. Corbett, K.R. Koedinger, R. Pelletier, Cognitive tutors: Lessons learned, J. Learning Sci. 4 (2) (1995) 167–207.

[4] J. Baher, How articulate virtual labs can help in thermodynamics education: A multiple case study, in: Proc. Frontiers in Education 1998 Conference, Tempe, AZ, 1998.

[5] J. Baher, Integrating “Articulate educational software” into engineering courses: A formative evaluation, in: Proc. Annual Meeting of the American Educational Research Association, San Diego, CA, 1998.

[6] J. Baher, Re-constructing engineering education through the use of new technologies, in: Proc. American Association for Higher Education (AAHE) National Conference on Higher Education, Washington, DC, 1999.

[7] J. Baher, Faculty attitudes towards teaching, technology and thermodynamics, in: Proc. Frontiers in Education Conference, San Juan, Puerto Rico, 1999.

[8] D.G. Bell, D.G. Bobrow, O. Raiman, M.H. Shirley, Dynamic documents and situated processes: Building on local knowledge in field service, in: Proc. International Working Conference on Integration of Enterprise Information and Processes, IPIC’96 Rethinking Documents, Cambridge, MA, 1996.

[9] J.S. Brown, R. Burton, J. de Kleer, Pedagogical, natural language, and knowledge engineering techniques in SOPHIE I, II, and III, in: D. Sleeman, J.S. Brown (Eds.), Intelligent Tutoring Systems, Academic Press, New York, 1982.

[10] E.C. Ciccarelli, Presentation based user interfaces, AI Technical Report 794, Massachusetts Institute of Technology, Cambridge, MA, 1984.

[11] D.C. Edelson, Learning from stories: Indexing and reminding in a Socratic case-based teaching system for elementary school biology, Ph.D. Thesis, Northwestern University, Evanston, IL, 1994.

[12] M.M. El-Wakil, Powerplant Technology, McGraw-Hill, New York, 1984.

[13] Electronics Workbench Inc., 908 Niagara Falls Boulevard, Suite #068 North Tonawanda, NY 14120–2060, USA.

[14] J.O. Everett, Topological inference of teleology: Deriving function from structure via evidential reasoning, Artificial Intelligence 113 (1999) 149–202.

[15] J.O. Everett, A theory of mapping from structure to function applied to engineering domains, in: Proc. IJCAI-95, Quebec, Montreal, 1995.

[16] J.O. Everett, K. Forbus, Scaling up logic-based truth maintenance systems via fact garbage collection, in: Proc. AAAI-96, Portland, OR, 1996.

[17] B. Falkenhainer, K. Forbus, Compositional modeling: Finding the right model for the job, Artificial Intelligence 51 (1–3) (1991) 95–143.

[18] B. Falkenhainer, K. Forbus, Setting up large-scale qualitative models, in: Proc. AAAI-88, St. Paul, MN, 1988.

[19] K.D. Forbus, B. Falkenhainer, Self-explanatory simulations: An integration of qualitative and quantitative knowledge, in: Proc. AAAI-90, Boston, MA, 1990, pp. 380–387.

[20] K.D. Forbus, R.W. Ferguson, D. Gentner, Incremental structure mapping, in: Proc. 16th Annual Conference of the Cognitive Science Society, Erlbaum, Hillsdale, NJ, 1994.

[21] K.D. Forbus, D. Gentner, Law. MAC/FAC: A model of similarity-based retrieval, Cognitive Sci. 19 (2) (1995) 141–205.

[22] K.D. Forbus, S.E. Kuehne, RoboTA: An agent colony architecture for supporting education, in: Proc. 2nd International Conference on Autonomous Agents, Minneapolis/St. Paul, MN, 1998.

[23] K. Forbus, J. de Kleer, Building Problem Solvers, MIT Press, Cambridge, MA, 1993.

[24] K. Forbus, J. Everett, D. Gentner, M. Wu, Towards a computational model of evaluating and using analogical inference, in: Proc. CogSci-97, Erlbaum, Hillsdale, NJ, 1997.

[25] K. Forbus, A. Stevens, Using qualitative simulation to generate explanations, in: Proc. Third Annual Conference of the Cognitive Science Society, Berkeley, CA, 1981.

[26] D. Gentner, Structure-mapping: A theoretical framework for analogy, Cognitive Sci. 23 (1983) 155–170.

[27] D. Gentner, B. Falkenhainer, J. Skorstad, Metaphor: The good, the bad and the ugly, in: Proc. 3rd Conference on Theoretical Issues in Natural Language Processing, Las Cruces, New Mexico, 1987.

[28] D. Gentner, M.J. Rattermann, A.B. Markman, L. Kotovsky, Two forces in the development of relational structure, in: T. Simon, G. Halford (Eds.), Developing Cognitive Competence: New Approaches to Process Modeling, Erlbaum, Hillsdale, NJ, 1995.

[29] T. Govindaraj, Qualitative approximation methodology for modeling and simulation of large dynamic systems: Applications to a marine steam power plant, IEEE Trans. Systems Man Cybernet. SMC-17 (6) (1987).

[30] T. Gruber, P. Gautier, Machine-generated explanations of engineering models: A compositional modeling approach, in: Proc. IJCAI-93, Chambery, France, 1993.

[31] P. Hayes, Naive physics 1: Ontology for liquids, in: J. Hobbs, R. Moore (Eds.), Formal Theories of the Commonsense World, Ablex, Norwood, NJ, 1985.

[32] R.W. Haywood, Analysis of Engineering Cycles: Power, Refrigerating and Gas Liquefaction Plant, Pergamon Press, 1985.

[33] S. Lajoie, S. Derry (Eds.), Computers as Cognitive Tools, Erlbaum, Hillsdale, NJ.

[34] D. Leake (Ed.), Case-Based Reasoning: Experiences, Lessons, and Future Directions, MIT Press, Cambridge, MA, 1996.

[35] L. Massey, J. de Bruin, B. Roberts, A training system for system maintenance, in: J. Psotka, L. Massey, S. Mutter (Eds.), Intelligent Tutoring Systems: Lessons Learned, Erlbaum, Hillsdale, NJ, 1988.

[36] D. McAllester, An outlook on truth maintenance, MIT AI Lab memo AIM-551, Cambridge, MA, 1980.

[37] J.D. Moore, W.R. Swartout, Pointing: A way toward explanation dialogue, in: Proc. AAAI-90, Boston, MA, 1990.

[38] P. Nayak, Automated modeling of physical systems, Ph.D. Thesis, Computer Science Department, Stanford University, 1992.

[39] E. Reiter, C. Mellish, Optimizing the costs and benefits of natural language generation, in: Proc. IJCAI-93, Chambery, France, 1993.

[40] J. Roschelle, Collaborative conceptual change: Jointly acting social and cognitive processes, in: Proc. CogSci-93.

[41] R. Schank, C. Cleary, Engines for Education, Erlbaum, Hillsdale, NJ, 1994.

[42] R.M. Stallman, G.J. Sussman, Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis, Artificial Intelligence 9 (1977) 135–196.

[43] MSC.Software, Inc. 66 Bovet Road, Suite 200, San Mateo, CA 94402, USA.

[44] B. White, J. Frederiksen, Causal model progressions as a foundation for intelligent learning environments, Artificial Intelligence 42 (1990) 99–157.

[45] C. Wu, T.J. Burke, Intelligent computer aided optimization on specific power of an OTEC Rankine power plant, Applied Thermal Engineering 18 (5) (1998) 295–300.

[46] C. Wu, M. Dieguez, Intelligent computer aided design on optimization of specific power of finite-time Rankine cycle using CyclePad, J. Computer Application in Engineering Education 16 (1) (1998) 9–13.