专家系统 知识库 知识获取 评测 (12)

计算机科学 1998 Vol. 25№. 2

46-48

专家系统的评测方法

Criteria for Expert System Design

肖 斐 黄海晖 廖光裕

(复旦大学计算机系 上海 200433)

TP18

摘 要 Expert critics have been built to critique human performance in various areas such as engineering design, decision making, etc. We suggest that critics can also be useful in the building and use of knowledge-based design systems (KBDSs). Knowledge engineers elicit knowledge from domain experts and build a knowledge-based design system. The system generates designs. The amount of knowledge the system possesses and the way it applies the knowledge directly influence the performance of its designs. Therefore, critics are proposed to assist 1) acquiring sufficient knowledge for constructing a desirable system, and 2) applying proper knowledge to generating designs.

关键词 Expert system, Knowledge-based system, Knowledge acquisition, Domain facts

一、引言

专家在进行工程设计时通常要遵循一般的规律，应用专业知识，联系相关线索，凭借丰富的经验。人们用基于知识的系统——专家系统来完成工程设计时，期望它具备专家的行为。因此建立专家系统，首先要获取领域专家知识，并把知识编码成计算机内部表示。在理想的情况下，系统能针对实际问题选用适当的知识完成任务。专家系统的性能取决于专家如何选择、概括知识，知识库设计者如何获取、表示、应用知识。

一般地，专家系统建立在假设的基础上，不是唯一的，并且易于出错。片面的、不相关的、不完整的知识，错误的假设很容易引进到系统中。必须采取有效的方法防止它们进入系统。

评测系统包括了专家系统。它输入问题的描述，输出问题的解答与对专家系统的评测，如错误是什么，错误发生在哪里。它给用户提供纠正系统错误，提高系统性能的指导。本文将探讨一系列评测的方法。

二、专家系统的评测

在这一部分，首先通过专家系统设计中四个问题的讨论来说明评测方法的必要性。然后讨论评测方法并给出实现的算法。

1. 评测的必要性

评测的必要性体现在专家系统设计时要考虑的四个问题中。前二个问题是关于获取与应用知识。后二个问题是关于系统的设计。

1)获取足够的知识。知识获取是一个主观性的过程。知识库设计者和领域专家的观点与经验都会有倾向性，从而导致有用的知识被忽略——完备性问题。关键问题是如何才能知道获取的知识是否足够。获取一个领域全部的知识是不可能的。但是足够的知识就能使专家系统应用成功。在一个中等规模的领域中，通过少数测试实例很难发现缺少的知识。增加测试实例数会导致大量时间的消耗而不可行。因此，需要一种不增加测试实例数的方法解决完备性问题。它能提供获取更多知识和重组知识库的线索与指导。

2)选取适当的知识。当专家系统中的知识增加时，我们必然会碰到如何为一个具体的事例选择适当的知识——知识应用问题。由于不可能预先知道所有的事例，这一问题十分棘手。专家系统采用启发式规则减少或限制在大问题空间的搜索。精心设计的启发式规则可能会得到所需的结果，也可能产生不可预料的结果。所以为保证专家系统正确的应用，有时必须采用其他知识或者采用别的启发式规则。

3)设计的正确性。正确性是专家系统设计的最低标准，是指问题的解答必须满足问题描述所明确

的要求。任何不满意的结果表明系统结构上的缺陷或知识库中的概念错误。

4)设计的一致性。问题描述具体说明某一问题的条件,可以用于检测系统的正确性;领域事实包括该领域的一般性知识,检测系统的一致性。一致性包括两个要点:a)系统各组成部分之间的一致性;b)系统各组成部分与领域事实之间的一致性。缺少一致性表明系统结构存在逻辑错误,知识库中包含错误的概念或假设。

2. 评测系统

上述四方面的问题是建立一个专家系统的主要障碍,评测方法能够系统地加以解决。一个评测系统包括以下几点:

1)评测系统必须要考虑上述四方面的问题。

2)问题描述、领域事实、专家系统产生的设计均是评测系统可以利用的信息。

3)评测系统既可以是独立的(依据整个知识库),也可以是演绎的(依据知识库中的部分知识)。

4)专家系统是评测系统的组成部分。

2.1 知识完备性的评测。一般说来,我们试图集中所有的专家,组合他们各自的专业知识,建立一个强大的知识库。然而当领域知识过于复杂时,这种方法并不可行。即使建立了一个强大的知识库,完备性问题仍然得不到解决。当系统产生一个正确的解答,我们无法知道是否还有更佳的答案。系统本身不能判断自身是否完备。评测系统则能使强大的知识库更加强大。它通过使用相互独立的两个系统,保证最终产生的结果至少是每个系统单独运行所得结果中最优者。我们说两个系统相互独立是指它们在实现机制、搜索算法、知识使用这几方面是互不相同的。评测系统比较它们产生的设计方案,研究不同之处,追究原因,最后确定缺少的知识。当然,获得两个相互独立的系统是比较困难的,它们必须满足:①采用不同的机制和不同的搜索算法;②应用不同的知识;③输入与输出采用相同的表示形式;④可以实现与运行。

算法 1 实现知识完备性评测,用于系统的创建阶段。假设我们要建立一个专家系统 KB;DS、IS 是另外两个相互独立的系统,不采用 KB 中的知识。d_d、d_i 分别是 DS、IS 产生的设计方案;diff 存储 d_d 与 d_i 的差别。对已知事例集合 P 中的每个事例进行检测直到所有事例被测试过且 diff 为空;或者当测试达到一定的满意程度,由专家中断。算法 1 如下:

```
for(each p∈P)do
begin
  di=Apply(IS,p);
  Knowledge Acquisition (p,KB,di);
end
KnowledgeAcquisition(p,KB,di)
begin
  dd=Apply(DS,p);
  diff=Compare(dd,di);
  if(diff<>NULL)
  begin
    ModifyKB(KB,diff);
    KnowledgeAcquisition(p,KB,di)
  end
end
```

上面,Compare(d_d,d_i)给出 d_d 与 d_i 之间的差别;在 ModifyKB(KB,diff)中,专家系统设计者首先分析 diff,探讨是否能通过重组知识库结构,增加新的知识来消除 diff。

2.2 知识选取的评测。寻求一个特定设计问题的最优解,常常难以确定该应用哪些知识。按照代价函数或专家判定,评测系统确定最恰当的知识。检测的过程一直进行到找不到更好的方案或达到某种专家认定的满意度为止。以下是两个实现知识选取的评测算法。

算法 2 假设 T 是最优设计方案数量的阈值;c_1 是设计方案 d_1 的代价。C 是各种代价的集合。d_t 是在时间 t 完成的设计方案。p,P 同上定义。每一个设计方案有一个 WorstParts 栈,最初这个栈是空的。DS 产生一个具有费用 c_t 的设计方案,并且由应用程序决定 WorstParts 中的内容。算法 2 如下:

```
for(each p∈P)do
begin
  numOfDesign=0;
  WorstParts=Null;
  AutoReDesign(p,numOfDesign,WorstParts);
end

AutoReDesign(p,numOfDesign,WorstParts)
begin
  SetInitial(WorstParts);
  dt=Apply(DS,p);
  num OfDesign++;
  ct=Evaluate(dt);
  WorstParts=Report(DS,dt);
  If(numOfDesign>T OR,ct∈C)
    Stop;
  else
  begin
    C=Append(ct,C);
    AutoReDesign(p,numOfDesign,WorstParts);
  end
end
```

此处,SetInitial(WorstParts)找出 d_t 中最关键的决策点,并重置启发性法则。每一个决策点都有一些相应的启发性法则,它们被轮流使用。Evaluate(d_t)根据代价函数计算 d_t 的代价。Report(DS,d_t)给出 d_t 中最差部分的列表。Append(c_t,C)把 c_t 加入到集合 C 中。除非 numOfDesign 达到阈值 T,或者 DS 不能生成相同代价的不同设计方案,该算法一直试

图根据 WorstParts 的内容在关键决策点上选用不同的变化。

显然，评测方法的性能依靠代价函数的实现。代价函数是衡量方案质量的尺度，如果寻找一个准确的尺度十分困难，就会影响系统性能的提高。另外一种方法是算法 3，让专家决定设计方案的哪一部分应该改善。假设 D 是设计方案的集合，suggest 是专家在检查过以图形方式显示的设计方案后输入的建议。SetInitial(Suggest)在指定的决策点采用新的启发性法则。ExpertExamine(d_t)以图形方式显示 d_t，然后返回专家可能提出的改善建议。算法 3 如下：

```
for(each p∈P)do
begin
  suggest=NULL;
  D=NULL;
  InteractiveReDesign(suggest,D);
end

InteractiveReDesign(suggest,D)
begin
  SetInitial(suggest);
  dt=Apply(DS,p);
  D=Append(dt,D);
  suggest=ExpertExamine(dt,D);
  If(suggest<>NULL)
    Interactive ReDesign(suggest,D);
end
```

算法 2 与算法 3 实现了两种可以互相替代的评测方法，一种是自动方法，一种是手工方法。使用者可以任选其一。

2.3 准确性与一致性的评测。这两种评测方法比较一致，因为它们都依赖于上下文相关的信息，如域事实、问题描述。准确性指设计方案满足问题描述提出的要求；一致性指设计方案与域事实、设计方案的各个组成部分之间没有冲突。

假设我们要构造一个系统 DS。P 是要解决的问题的集合。T 是域事实，d 是一个设计方案。Sys 是 S 的一个版本。$S_{incorrect}$，$S_{inconsistent}$分别是被报告错误的集合，它们为空时算法结束。应用域事实与输入事实，从非空的 $S_{incorrect}$或 $S_{inconsistent}$开始提纯 DS。Create(T)应用所有领域专家提供的和说明书中的知识，按照产生式规则以过程的形式生成 DS 的初始版本。Apply(Sys,p)中，Sys 用于构造一个针对 p 的设计方案。Refine(Sys,d,T,$S_{incorrect}$,$S_{inconsistent}$)根据这些变量来修正 Sys 并产生 DS 的新版本。CorrectnessCheck(d,p)针对 p 检查 d，把发现的错误记录在 $S_{incorrect}$中。ConsistencyCheck(d,T)针对 p 检查 T，把发现的错误记录在 $S_{inconsistent}$中。SystemDesign(Sys,p)是一个递归定义的过程，用于提纯 DS 直到 DS 的某一个版本为问题 p 生成的设计方案中没有任何不正确或不一致的地方。其算法 4 如下：

```
Sys=Create(T);
for(each p∈P)do
  SystemDesign(Sys,p);
SystemDesign(Sys,p)
begin
  d=Apply(DS,p);
  Sincorrect=CorrectnessCheck(d,p);
  Sinconsistent=Consistencycheck(d,T);
  If(Sincorrect<>NULL OR Sinconsistent<>NULL)
  begin
    Sys=Refine(Sys,d,T,Sincorrect,Sinconsistent);
    SystemDesign(Sys,p);
  end
end
```

三、结论与进一步的研究

我们致力于寻找一种方法，能指导专家系统设计者客观与全面地完成设计任务，并能够消除知识获取这一瓶颈问题，以及设计时的主观倾向性。我们已经看到评测方法所起的作用。然而还需要把它应用到更多的领域，扩充更多的评测方法。我们将进一步研究评测方法在知识库建模，专家系统验证、优化，机器学习等方面的应用。

参考文献

[1] Weiss S. M. et al. , EXPERT: A System of Developing Consultation Nodes, Artificial Intelligence, 1979, 11

[2] Wong A. K. C. et al. , A Gray-Level Threshold Selection Method Based on Maximum Entropy Principle, IEEE Trans. on Systems, Man and Cybernetics, 1989

（上接第 28 页）

参考文献

[1] Y. Shoham, Agent-oriented programming, Artificial Intelligence, 60, 1993

[2] R. Goodwin, Formalizing properties of agents, TR CMU CS-93-159, Carnegie Mellon University, Stanford, 1992

[3] Zeng Guangzhou, Sun Boqi, Requirements engineering method based on system simulation, In: Proc. of the Changsha International CASE symposium (CICS'95), Changsha, China 1995

[4] R. Davis, R. Smith, Negotiation as a metaphor for distributed problem solving, Artificial Intelligence, 20(1), 1983