Solutions for Chapter 1
Introduction

1.1

a. Dictionary definitions of **intelligence** talk about "the capacity to acquire and apply knowledge" or "the faculty of thought and reason" or "the ability to comprehend and profit from experience." These are all reasonable answers, but if we want something quantifiable we would use something like "the ability to apply knowledge in order to perform better in an environment."

b. We define **artificial intelligence** as the study and construction of agent programs that perform well in a given environment, for a given agent architecture.

c. We define an **agent** as an entity that takes action in response to percepts from an environment.

d. We define **rationality** as the property of a system which does the "right thing" given what it knows. See Section 2.2 for a more complete discussion. Both describe perfect rationality, however; see Section 27.3.

e. We define **logical reasoning** as the a process of deriving new sentences from old, such that the new sentences are necessarily true if the old ones are true. (Notice that does not refer to any specific syntax oor formal language, but it does require a well-defined notion of truth.)

1.2 See the solution for exercise 26.1 for some discussion of potential objections.

The probability of fooling an interrogator depends on just how unskilled the interrogator is. One entrant in the 2002 Loebner prize competition (which is not quite a real Turing Test) did fool one judge, although if you look at the transcript, it is hard to imagine what that judge was thinking. There certainly have been examples of a chatbot or other online agent fooling humans. For example, see See Lenny Foner's account of the Julia chatbot at foner.www.media.mit.edu/people/foner/Julia/. We'd say the chance today is something like 10%, with the variation depending more on the skill of the interrogator rather than the program. In 50 years, we expect that the entertainment industry (movies, video games, commercials) will have made sufficient investments in artificial actors to create very credible impersonators.

1.3 Yes, they are rational, because slower, deliberative actions would tend to result in more damage to the hand. If "intelligent" means "applying knowledge" or "using thought and reasoning" then it does not require intelligence to make a reflex action.

第1章的解决方案
Introduction

1.1

a. 智力的字典定义谈论"获取和应用知识的能力"或"思想和理性的能力"或"理解和从经验中获利的能力。"这些都是合理的答案，但如果我们想要一些可量化的东西，我们会使用类似"应用知识的能力，以便在环境中表现得更好。"b． 我们将人工智能定义为针对给定代理体系结构，在给定环境中表现良好的代理程序的研究和构建。 c. 我们将代理定义为响应环境感知而采取行动的实体。 d. 我们将理性定义为一个系统的属性，该系统根据它所知道的做"正确的事情"。 有关更完整的讨论，请参阅第2.2节。 然而，两者都描述了完美的理性;请参阅第27.3节。 e. 我们将逻辑推理定义为从旧句子中推导新句子的过程，如果旧句子是真的，新句子就一定是真的。 （请注意，它没有引用任何特定的语法或形式语言，但它确实需要一个定义良好的真理概念。）

1.2 有关可能的反对意见的一些讨论，请参阅练习26.1的解决方案。

欺骗询问者的可能性取决于interroga tor的不熟练程度。 2002年Loebner prize竞赛（这不是一个真正的图灵测试）的一位参赛者确实愚弄了一位法官，尽管如果你看看成绩单，很难想象那个法官在想什么。 当然也有聊天机器人或其他在线代理欺骗人类的例子。 例如，请参阅Lenny Foner对Julia chatbot的描述，网址为foner.www.media.mit.edu/people/foner/Julia我们认为今天的机会大约是10%，变化更多地取决于询问者的技能而不是程序。 50年后，我们预计娱乐业（电影，视频游戏，com mercials）将对人造演员进行足够的投资，以创造非常可信的模仿者。

1.3是的，他们是理性的，因为较慢的，深思熟虑的行动往往会导致更多的手的伤害。 如果"智能"意味着"应用知识"或"使用思想和推理"，那么它不需要智力来做出反射动作。

1.4 No. IQ test scores correlate well with certain other measures, such as success in college, ability to make good decisions in complex, real-world situations, ability to learn new skills and subjects quickly, and so on, but *only* if they're measuring fairly normal humans. The IQ test doesn't measure everything. A program that is specialized only for IQ tests (and specialized further only for the analogy part) would very likely perform poorly on other measures of intelligence. Consider the following analogy: if a human runs the 100m in 10 seconds, we might describe him or her as *very athletic* and expect competent performance in other areas such as walking, jumping, hurdling, and perhaps throwing balls; but we would not desscribe a Boeing 747 as *very athletic* because it can cover 100m in 0.4 seconds, nor would we expect it to be good at hurdling and throwing balls.

Even for humans, IQ tests are controversial because of their theoretical presuppositions about innate ability (distinct from training effects) adn the generalizability of results. See *The Mismeasure of Man* by Stephen Jay Gould, Norton, 1981 or *Multiple intelligences: the theory in practice* by Howard Gardner, Basic Books, 1993 for more on IQ tests, what they measure, and what other aspects there are to "intelligence."

1.5 In order of magnitude figures, the computational power of the computer is 100 times larger.

1.6 Just as you are unaware of all the steps that go into making your heart beat, you are also unaware of most of what happens in your thoughts. You do have a conscious awareness of some of your thought processes, but the majority remains opaque to your consciousness. The field of psychoanalysis is based on the idea that one needs trained professional help to analyze one's own thoughts.

1.7

- Although bar code scanning is in a sense computer vision, these are not AI systems. The problem of reading a bar code is an extremely limited and artificial form of visual interpretation, and it has been carefully designed to be as simple as possible, given the hardware.
- In many respects. The problem of determining the relevance of a web page to a query is a problem in natural language understanding, and the techniques are related to those we will discuss in Chapters 22 and 23. Search engines like Ask.com, which group the retrieved pages into categories, use clustering techniques analogous to those we discuss in Chapter 20. Likewise, other functionalities provided by a search engines use intelligent techniques; for instance, the spelling corrector uses a form of data mining based on observing users' corrections of their own spelling errors. On the other hand, the problem of indexing billions of web pages in a way that allows retrieval in seconds is a problem in database design, not in artificial intelligence.
- To a limited extent. Such menus tends to use vocabularies which are very limited – e.g. the digits, "Yes", and "No" — and within the designers' control, which greatly simplifies the problem. On the other hand, the programs must deal with an uncontrolled space of all kinds of voices and accents.

1.4不。 智商测试分数与某些其他衡量标准很好地相关，例如大学成功，在复杂的现实环境中做出正确决策的能力，快速学习新技能和科目的能力等等，但前提是 智商测试并不能衡量一切。 一个专门用于智商测试的程序（并且专门用于类比部分）很可能在其他智力测量方面表现不佳。 考虑下面的类比：如果一个人在10秒内跑完100米，我们可能会把他或她描述为非常运动，并期望在其他领域有足够的表现，如步行、跳跃、跨栏，也许还有扔球；但我们不会把波音747描述为非常运动，因为它可以在0.4秒内跑完100米，我们也不会期望它擅长跨栏和扔球。

即使对于人类来说，智商测试也是有争议的，因为它们关于先天能力（与训练效果不同）和结果的可概括性的理论前提。 参见斯蒂芬*杰伊*古尔德（Stephen Jay Gould），诺顿（Norton），1981年或多重智能：实践中的理论，霍华德*加德纳（Howard Gardner），基本书籍，1993年，有关智商测试，他们测量的内容以及"智力"的其他方″

1.5按数量级计算，计算机的计算能力要大100倍。

1.6正如你不知道所有的步骤，进入使你的心脏跳动，你也不知道大部分发生在你的想法。 你确实有意识地意识到你的一些思维过程，但大多数人对你的意识仍然是不透明的。 精神分析的领域是基于这样一个想法，即一个人需要训练有素的专业帮助来分析自己的想法。

1.7

*虽然条形码扫描在某种意义上是计算机视觉，但这些都不是AI系统。 阅读条形码的问题是一种极其有限和人为的视觉解释形式，考虑到硬件，它被精心设计为尽可能简单。

•在许多方面。 确定网页与查询的相关性的问题是自然语言理解中的一个问题，这些技术与我们将在第22章和第23章中讨论的技术有关。 搜索引擎喜欢Ask.com，将检索到的页面分组到类别中，使用类似于我们在第20章中讨论的聚类技术。同样，搜索引擎提供的其他功能使用智能技术;例如，拼写校正器使用基于观察用户对自己拼写错误的更正的数据挖掘形式。 另一方面，以允许在几秒钟内检索的方式索引数十亿网页的问题是数据库设计中的问题，而不是人工智能中的问题。

*在有限的范围内。 这些菜单倾向于使用非常有限的词汇–例如数字，"是"和"否"—并且在设计师的控制范围内，这大大简化了问题。 另一方面，程序必须处理各种声音和口音的不受控制的空间。

The voice activated directory assistance programs used by telephone companies, which must deal with a large and changing vocabulary are certainly AI programs.

- This is borderline. There is something to be said for viewing these as intelligent agents working in cyberspace. The task is sophisticated, the information available is partial, the techniques are heuristic (not guaranteed optimal), and the state of the world is dynamic. All of these are characteristic of intelligent activities. On the other hand, the task is very far from those normally carried out in human cognition.

1.8 Presumably the brain has evolved so as to carry out this operations on visual images, but the mechanism is only accessible for one particular purpose in this particular cognitive task of image processing. Until about two centuries ago there was no advantage in people (or animals) being able to compute the convolution of a Gaussian for any other purpose.

The really interesting question here is what we mean by saying that the "actual person" can do something. The person can see, but he cannot compute the convolution of a Gaussian; but computing that convolution is *part* of seeing. This is beyond the scope of this solution manual.

1.9 Evolution tends to perpetuate organisms (and combinations and mutations of organisms) that are successful enough to reproduce. That is, evolution favors organisms that can optimize their performance measure to at least survive to the age of sexual maturity, and then be able to win a mate. Rationality just means optimizing performance measure, so this is in line with evolution.

1.10 This question is intended to be about the essential nature of the AI problem and what is required to solve it, but could also be interpreted as a sociological question about the current practice of AI research.

A *science* is a field of study that leads to the acquisition of empirical knowledge by the scientific method, which involves falsifiable hypotheses about what is. A pure *engineering* field can be thought of as taking a fixed base of empirical knowledge and using it to solve problems of interest to society. Of course, engineers do bits of science—e.g., they measure the properties of building materials—and scientists do bits of engineering to create new devices and so on.

As described in Section 1.1, the "human" side of AI is clearly an empirical science—called cognitive science these days—because it involves psychological experiments designed out to find out how human cognition actually works. What about the the "rational" side? If we view it as studying the abstract relationship among an arbitrary task environment, a computing device, and the program for that computing device that yields the best performance in the task environment, then the rational side of AI is really mathematics and engineering; it does not require any empirical knowledge about the *actual* world—and the *actual* task environment—that we inhabit; that a given program will do well in a given environment is a *theorem*. (The same is true of pure decision theory.) In practice, however, we are interested in task environments that do approximate the actual world, so even the rational side of AI involves finding out what the actual world is like. For example, in studying rational agents that communicate, we are interested in task environments that contain humans, so we have

电话公司使用的语音激活目录援助程序，必须处理大量和不断变化的词汇当然是人工智能程序。 •这是边界。 对于将这些视为在网络空间中工作的智能代理，有话要说。 任务是复杂的，可用的信息是部分的，技术是启发式的（不能保证是最佳的），世界的状态是动态的。 所有这些都是智能活动的特征。 另一方面，任务与人类认知中通常进行的任务相差甚远。

1.8据推测，大脑已经进化，以便在视觉图像上进行这种操作，但在图像处理的这个特定认知任务中，该机制只能用于一个特定目的。 直到大约两个世纪前，人们（或动物）能够为任何其他目的计算高斯的卷积是没有优势的。

这里真正有趣的问题是我们说"实际的人"可以做一些事情的意思。 这个人可以看到，但是他不能计算高斯的卷积；但是计算卷积是看到的一部分。 这超出了本解决方案手册的范围。

1.9进化倾向于使足以成功繁殖的生物体（以及器官isms的组合和突变）永久化。 也就是说，进化有利于能够优化其性能度量的生物，以至少存活到性成熟的年龄，然后能够赢得配偶。 理性只是意味着优化性能度量，所以这是符合进化的。

1.10这个问题的目的是关于人工智能问题的本质和解决这个问题所需要的东西,但也可以解释为一个关于当前人工智能研究实践的社会学问题。

科学是一个研究领域，通过科学方法获得经验知识，其中涉及关于什么是可证伪的假设。 一个纯粹的工程领域可以被认为是一个固定的经验知识基础，并用它来解决社会感兴趣的问题。 当然，工程师做一些科学-例如，他们测量建筑材料的属性-科学家做一些工程来创造新的设备等等。

如第1.1节所述，AI的"人"方面显然是一种经验科学-这些天被称为认知科学-因为它涉及旨在找出人类认知如何实际工作的心理实验。 "理性"的一面呢？ 如果我们把它看作是研究任意任务环境、计算设备和在任务环境中产生最佳性能的计算设备的程序之间的抽象关系，那么人工智能的理性方面实际上是数学和工程；它不需要任何关于我们所居住的实际世界和实际任务环境的经验知识；给定的程序在给定的环境中会做得很好是一个定理。 （纯决策理论也是如此。）然而，在实践中，我们对确实近似实际世界的任务环境感兴趣，所以即使是人工智能的理性方面也涉及找出实际世界是什么样子。 例如，在研究沟通的理性代理时，我们对包含人类的任务环境感兴趣，因此我们有

to find out what human language is like. In studying perception, we tend to focus on sensors such as cameras that extract useful information from the actual world. (In a world without light, cameras wouldn't be much use.) Moreover, to design vision algorithms that are good at extracting information from camera images, we need to understand the actual world that generates those images. Obtaining the required understanding of scene characteristics, object types, surface markings, and so on is a quite different kind of science from ordinary physics, chemistry, biology, and so on, but it is still science.

In summary, AI is definitely engineering but it would not be especially useful to us if it were not also an empirical science concerned with those aspects of the real world that affect the design of intelligent systems for that world.

1.11 This depends on your definition of "intelligent" and "tell." In one sense computers only do what the programmers command them to do, but in another sense what the programmers consciously tells the computer to do often has very little to do with what the computer actually does. Anyone who has written a program with an ornery bug knows this, as does anyone who has written a successful machine learning program. So in one sense Samuel "told" the computer "learn to play checkers better than I do, and then play that way," but in another sense he told the computer "follow this learning algorithm" and it learned to play. So we're left in the situation where you may or may not consider learning to play checkers to be s sign of intelligence (or you may think that learning to play in the right way requires intelligence, but not in this way), and you may think the intelligence resides in the programmer or in the computer.

1.12 The point of this exercise is to notice the parallel with the previous one. Whatever you decided about whether computers could be intelligent in 1.11, you are committed to making the same conclusion about animals (including humans), *unless* your reasons for deciding whether something is intelligent take into account the mechanism (programming via genes versus programming via a human programmer). Note that Searle makes this appeal to mechanism in his Chinese Room argument (see Chapter 26).

1.13 Again, the choice you make in 1.11 drives your answer to this question.

1.14

a. (ping-pong) A reasonable level of proficiency was achieved by Andersson's robot (Andersson, 1988).

b. (driving in Cairo) No. Although there has been a lot of progress in automated driving, all such systems currently rely on certain relatively constant clues: that the road has shoulders and a center line, that the car ahead will travel a predictable course, that cars will keep to their side of the road, and so on. Some lane changes and turns can be made on clearly marked roads in light to moderate traffic. Driving in downtown Cairo is too unpredictable for any of these to work.

c. (driving in Victorville, California) Yes, to some extent, as demonstrated in DARPA's Urban Challenge. Some of the vehicles managed to negotiate streets, intersections, well-behaved traffic, and well-behaved pedestrians in good visual conditions.

4 Chapter 1. Introduction

找出人类的语言是什么样的。 在研究感知时，我们倾向于关注从实际世界中提取有用信息的传感器，例如相机。 （在一个没有光线的世界里，相机没有多大用处。）此外，要设计擅长从相机图像中提取信息的视觉算法，我们需要了解生成这些图像的实际世界。 获得对场景特征，物体类型，表面标记等的所需理解是一种与普通物理，化学，生物学等完全不同的科学，但它仍然是科学。

总之，人工智能肯定是工程，但如果它不是一门与现实世界中影响该世界智能系统设计的方面有关的经验科学，它对我们就不会特别有用。

1.11这取决于你对"智能"和"告诉"的定义。"从某种意义上说，计算机只做程序员命令他们做的事情，但在另一个意义上，程序员有意识地告诉计算机做的事情往往与计算机实际做的事情无关。 任何编写过带有ornery bug的程序的人都知道这一点，任何编写过成功机器学习程序的人也是如此。 所以在某种意义上，塞缪尔"告诉"计算机"学会比我更好地玩跳棋，然后这样玩"，但在另一种意义上，他告诉计算机"遵循这种学习算法"，它学会了玩。 所以我们留下的情况是，你可能会或可能不会考虑学习玩跳棋是智力的标志（或者你可能认为学习以正确的方式玩需要智力，但不是这样），你可能认为智

1.12本练习的重点是注意与前一练习的平行性。 无论你在1.11中决定计算机是否智能化，你都致力于对动物（包括人类）做出同样的结论，除非你判断某些东西是否智能的理由考虑到机制（通过基因编程而不是通过人类程序员编程）。 请注意，塞尔在他的中国房间论证中对机制提出了这种呼吁（见第26章）。

1.13 同样，你在1.11中做出的选择推动了你对这个问题的回答。

1.14

a.（乒乓）通过Andersson的机器人达到了合理的熟练程度（Andersson，1988）。 b.（在开罗开车）没有。 虽然在自动驾驶方面已经取得了很大的进展，但所有这些系统目前都依赖于某些相对恒定的线索：道路有肩膀和中心线，前方的汽车将以可预测的路线行驶，汽车将保持在路边，等等。 在轻度至中度交通情况下，可在清晰标示的道路上更改及转弯。 在开罗市中心开车太不可预测了，任何这些都无法工作。 c.（在加利福尼亚州维克多维尔开车）是的，在某种程度上，正如DARPA的城市挑战所证明的那样。 一些车辆设法在良好的视觉条件下谈判街道，十字路口，行为良好的交通和行为良好的行人。

d. (shopping at the market) No. No robot can currently put together the tasks of moving in a crowded environment, using vision to identify a wide variety of objects, and grasping the objects (including squishable vegetables) without damaging them. The component pieces are nearly able to handle the individual tasks, but it would take a major integration effort to put it all together.

e. (shopping on the web) Yes. Software robots are capable of handling such tasks, particularly if the design of the web grocery shopping site does not change radically over time.

f. (bridge) Yes. Programs such as GIB now play at a solid level.

g. (theorem proving) Yes. For example, the proof of Robbins algebra described on page 360.

h. (funny story) No. While some computer-generated prose and poetry is hysterically funny, this is invariably unintentional, except in the case of programs that echo back prose that they have memorized.

i. (legal advice) Yes, in some cases. AI has a long history of research into applications of automated legal reasoning. Two outstanding examples are the Prolog-based expert systems used in the UK to guide members of the public in dealing with the intricacies of the social security and nationality laws. The social security system is said to have saved the UK government approximately $150 million in its first year of operation. However, extension into more complex areas such as contract law awaits a satisfactory encoding of the vast web of common-sense knowledge pertaining to commercial transactions and agreement and business practices.

j. (translation) Yes. In a limited way, this is already being done. See Kay, Gawron and Norvig (1994) and Wahlster (2000) for an overview of the field of speech translation, and some limitations on the current state of the art.

k. (surgery) Yes. Robots are increasingly being used for surgery, although always under the command of a doctor. Robotic skills demonstrated at superhuman levels include drilling holes in bone to insert artificial joints, suturing, and knot-tying. They are not yet capable of planning and carrying out a complex operation autonomously from start to finish.

1.15

The progress made in this contests is a matter of fact, but the impact of that progress is a matter of opinion.

- **DARPA Grand Challenge for Robotic Cars** In 2004 the Grand Challenge was a 240 km race through the Mojave Desert. It clearly stressed the state of the art of autonomous driving, and in fact no competitor finished the race. The best team, CMU, completed only 12 of the 240 km. In 2005 the race featured a 212km course with fewer curves and wider roads than the 2004 race. Five teams finished, with Stanford finishing first, edging out two CMU entries. This was hailed as a great achievement for robotics and for the Challenge format. In 2007 the Urban Challenge put cars in a city setting, where they had to obey traffic laws and avoid other cars. This time CMU edged out Stanford.

d.（在市场购物）没有。 目前没有机器人能够将在拥挤的环境中移动的任务放在一起，使用视觉识别各种各样的物体，并抓住物体（包括可压扁的蔬菜）而不会损坏它们。 组件部分几乎能够处理单个任务，但需要进行重大的集成工作才能将它们放在一起。 e.（在网上购物）是的。 软件机器人能够处理这些任务，特别是如果网络购物网站的设计不会随着时间的推移而发生根本变化。 f.（桥牌）是的。 像GIB这样的程序现在发挥在一个坚实的水平. g.（定理证明）是的。 例如，第360页描述的罗宾斯代数的证明。 h.（有趣的故事）不。 虽然一些计算机生成的散文和诗歌是歇斯底里的滑稽，但这总是无意的，除了那些回响散文的程序，他们已经记住了。 i.（法律咨询）是的，在某些情况下。 人工智能在研究自动法律推理的应用方面有着悠久的历史。 两个突出的例子是英国使用的基于Prolog的专家系统，用于指导公众处理社会保障和国籍法的复杂性。 据说，社会保障制度在运作的第一年为英国政府节省了大约1.5亿美元。 然而，扩展到更复杂的领域，如合同法，需要对与商业交易、协议和商业惯例有关的大量常识知识进行令人满意的编码。 j.（翻译）是的。 以有限的方式，这已经在做。 参见Kay，Gawron和Norvig（1994）和Wahlster（2000），了解语音翻译领域的概述，以及对当前艺术状态的一些限制。 k.（手术）是的。 机器人越来越多地被用于手术，尽管总是在医生的指挥下。 在超人水平上展示的机器人技能包括在骨骼上钻孔以插入人工关节，缝合和打结。 他们还没有能力自始至终自主规划和执行复杂的行动。

1.15

在这一竞赛中取得的进展是一个事实，但这一进展的影响是一个意见问题。

*DARPA机器人汽车大挑战2004年的大挑战是穿越莫哈韦沙漠的240公里比赛。 它明确强调了自动驾驶的艺术状态，事实上没有竞争对手完成比赛。 最好的团队CMU只完成了240公里中的12公里。 在2005年的比赛中，比2004年的比赛有一个212公里的赛道，弯道更少，道路更宽。 五支球队完成了比赛，斯坦福获得了第一名，淘汰了两个CMU参赛作品。 这被誉为机器人技术和挑战形式的伟大成就。 2007年，城市挑战将汽车置于城市环境中，他们必须遵守交通法规并避开其他汽车。 这次CMU淘汰了斯坦福。

The competition appears to have been a good testing ground to put theory into practice, something that the failures of 2004 showed was needed. But it is important that the competition was done at just the right time, when there was theoretical work to consolidate, as demonstrated by the earlier work by Dickmanns (whose VaMP car drove autonomously for 158km in 1995) and by Pomerleau (whose Navlab car drove 5000km across the USA, also in 1995, with the steering controlled autonomously for 98% of the trip, although the brakes and accelerator were controlled by a human driver).

- **International Planning Competition** In 1998, five planners competed: Blackbox, HSP, IPP, SGP, and STAN. The result page (`ftp://ftp.cs.yale.edu/pub/mcdermott/aipscomp-results.html`) stated "all of these planners performed very well, compared to the state of the art a few years ago." Most plans found were 30 or 40 steps, with some over 100 steps. In 2008, the competition had expanded quite a bit: there were more tracks (satisficing vs. optimizing; sequential vs. temporal; static vs. learning). There were about 25 planners, including submissions from the 1998 groups (or their descendants) and new groups. Solutions found were much longer than in 1998. In sum, the field has progressed quite a bit in participation, in breadth, and in power of the planners. In the 1990s it was possible to publish a Planning paper that discussed only a theoretical approach; now it is necessary to show quantitative evidence of the efficacy of an approach. The field is stronger and more mature now, and it seems that the planning competition deserves some of the credit. However, some researchers feel that too much emphasis is placed on the particular classes of problems that appear in the competitions, and not enough on real-world applications.
- **Robocup Robotics Soccer** This competition has proved extremely popular, attracting 407 teams from 43 countries in 2009 (up from 38 teams from 11 countries in 1997). The robotic platform has advanced to a more capable humanoid form, and the strategy and tactics have advanced as well. Although the competition has spurred innovations in distributed control, the winning teams in recent years have relied more on individual ball-handling skills than on advanced teamwork. The competition has served to increase interest and participation in robotics, although it is not clear how well they are advancing towards the goal of defeating a human team by 2050.
- **TREC Information Retrieval Conference** This is one of the oldest competitions, started in 1992. The competitions have served to bring together a community of researchers, have led to a large literature of publications, and have seen progress in participation and in quality of results over the years. In the early years, TREC served its purpose as a place to do evaluations of retrieval algorithms on text collections that were large for the time. However, starting around 2000 TREC became less relevant as the advent of the World Wide Web created a corpus that was available to anyone and was much larger than anything TREC had created, and the development of commercial search engines surpassed academic research.
- **NIST Open Machine Translation Evaluation** This series of evaluations (explicitly not labelled a "competition") has existed since 2001. Since then we have seen great advances in Machine Translation quality as well as in the number of languages covered.

竞争似乎是将理论付诸实践的良好试验场，2004年的失败表明这是必要的。 但重要的是，比赛是在正确的时间进行的，当时有理论上的工作需要巩固，正如Dickmanns（他的鞋面车在1995年自动驾驶158公里）和Pomerleau（他的Navlab车在美国行驶5000公里，同样在1995年，98%的行程中自动控制转向，尽管刹车和加速器由人类驾驶员控制）的早期工作所证明的那样。

•国际规划比赛1998年，五位规划师参加了比赛：Blackbox，HSP，IPP，SGP和STAN。 结果页面（ftp://ftp.cs.yale.edu/pub/麦克德莫特/aipscomp-结果.html）表示"与几年前的艺术状态相比，所有这些规划师都表现得非常好。"发现的大多数计划是30或40步，其中一些超过100步。 在2008年，竞争已经扩大了很多：有更多的轨道（满足与优化；顺序与时间；静态与学习）。 约有25名策划者，包括1998年团体（或其后代）和新团体的意见书。 找到的解决方案比1998年要长得多。 总之，该领域在参与，广度和规划者的权力方面取得了相当大的进展。 在20世纪90年代，有可能发表一份只讨论理论方法的规划文件;现在有必要展示一种方法有效性的定量证据。 场现在更强更成熟，看来策划大赛值得一些功劳。 然而，一些研究人员认为，过于强调比赛中出现的特定问题类别，而在现实世界的应用中还不够。

*Robocup Robotics Soccer这项比赛非常受欢迎，2009年吸引了来自43个国家的407支球队（1997年来自11个国家的38支球队）。 机器人平台已经发展到一个更有能力的人形形式，战略和战术也进步了。 尽管比赛刺激了分布式控制的创新，但近年来获胜的球队更多地依赖于个人的控球技巧，而不是先进的团队合作。 这项比赛有助于增加对机器人技术的兴趣和参与，尽管目前尚不清楚他们在实现到2050年击败人类团队的目标方面进展得有多好。

*TREC信息检索会议这是最古老的比赛之一，始于1992年。 这些比赛使一群研究人员聚集在一起,产生了大量的出版物,多年来在参与和结果质量方面取得了进展。 在早期，TREC的目的是作为一个地方，对当时很大的文本集合进行检索算法的评估。 然而，从2000年左右开始，随着万维网的出现，trec创建了一个任何人都可以使用的语料库，并且比TREC创建的任何语料库都要大得多，商业搜索引擎的发展超过了学术研究。

•Nist开放式机器翻译评估这一系列评估（明确未标记为"竞争"）自2001年以来一直存在。 从那时起，我们看到了机器翻译质量以及所涵盖语言数量的巨大进步。

The dominant approach has switched from one based on grammatical rules to one that relies primarily on statistics. The NIST evaluations seem to track these changes well, but don't appear to be driving the changes.

Overall, we see that whatever you measure is bound to increase over time. For most of these competitions, the measurement was a useful one, and the state of the art has progressed. In the case of ICAPS, some planning researchers worry that too much attention has been lavished on the competition itself. In some cases, progress has left the competition behind, as in TREC, where the resources available to commercial search engines outpaced those available to academic researchers. In this case the TREC competition was useful—it helped train many of the people who ended up in commercial search engines—and in no way drew energy away from new ideas.

占主导地位的方法已经从基于语法规则的方法转变为主要依赖统计的方法。 NIST评估似乎很好地跟踪这些变化，但似乎没有推动这些变化。

总的来说，我们看到，无论你衡量什么，都会随着时间的推移而增加。 对于大多数这些比赛来说，测量是一个有用的，并且艺术的状态已经进步。 在ICAPS的情况下，一些规划研究人员担心，太多的注意力已经被浪费在竞争本身。 在某些情况下，进展已经将竞争抛在了脑后，就像在TREC中一样，商业搜索引擎可用的资源超过了学术研究人员可用的资源。 在这种情况下，TREC竞争是有用的—它帮助培训了许多最终进入商业搜索引擎的人–并且绝不会从新思想中汲取能量。

Solutions for Chapter 2
Intelligent Agents

2.1 This question tests the student's understanding of environments, rational actions, and performance measures. Any sequential environment in which rewards may take time to arrive will work, because then we can arrange for the reward to be "over the horizon." Suppose that in any state there are two action choices, a and b, and consider two cases: the agent is in state s at time T or at time $T-1$. In state s, action a reaches state s' with reward 0, while action b reaches state s again with reward 1; in s' either action gains reward 10. At time $T-1$, it's rational to do a in s, with expected total reward 10 before time is up; but at time T, it's rational to do b with total expected reward 1 because the reward of 10 cannot be obtained before time is up.

Students may also provide common-sense examples from real life: investments whose payoff occurs after the end of life, exams where it doesn't make sense to start the high-value question with too little time left to get the answer, and so on.

The environment state can include a clock, of course; this doesn't change the gist of the answer—now the action will depend on the clock as well as on the non-clock part of the state—but it does mean that the agent can never be in the same state twice.

2.2 Notice that for our simple environmental assumptions we need not worry about quantitative uncertainty.

a. It suffices to show that for all possible actual environments (i.e., all dirt distributions and initial locations), this agent cleans the squares at least as fast as any other agent. This is trivially true when there is no dirt. When there is dirt in the initial location and none in the other location, the world is clean after one step; no agent can do better. When there is no dirt in the initial location but dirt in the other, the world is clean after two steps; no agent can do better. When there is dirt in both locations, the world is clean after three steps; no agent can do better. (Note: in general, the condition stated in the first sentence of this answer is much stricter than necessary for an agent to be rational.)

b. The agent in (a) keeps moving backwards and forwards even after the world is clean. It is better to do *NoOp* once the world is clean (the chapter says this). Now, since the agent's percept doesn't say whether the other square is clean, it would seem that the agent must have some memory to say whether the other square has already been cleaned. To make this argument rigorous is more difficult—for example, could the agent arrange things so that it would only be in a clean left square when the right square

第2章的解决方案
智能代理

2.1本题考查学生对环境、理性行动、绩效衡量的理解。 任何奖励可能需要时间才能到达的顺序环境都会起作用，因为这样我们就可以安排奖励"超过地平线"。"假设在任何状态下都有两个动作选择，a和b，并考虑两种情况：代理在时间T或在时间T-1处于状态s。 在状态s中，动作a以奖励0到达状态s'，而动作b以奖励1再次到达状态s;在s'中，任何一个动作都获得奖励10。 在时间T-1时，在时间到之前用预期总奖励10做a是合理的；但在时间T时，用预期总奖励1做b是合理的，因为在时间到之前无法获得10的奖励。

学生还可以提供现实生活中的常识性例子：在生命结束后产生回报的投资，在没有时间得到答案的情况下开始高价值问题没有意义的考试，等等。

当然，环境状态可以包括时钟;这不会改变答案的要点-现在动作将取决于时钟以及状态的非时钟部分-但它确实意味着代理永远不会两次处于相同的状

2.2请注意，对于我们简单的环境假设，我们不必担心量化的不确定性。

a. 足以表明，对于所有可能的实际环境（即，所有污垢分布和初始位置），该代理至少与任何其他代理一样快地清洁方块。 当没有污垢时，这是微不足道的。 当初始位置有污垢而另一个位置没有污垢时，一步之后世界是干净的;没有任何代理人可以做得更好。 当初始位置没有污垢，而另一个位置没有污垢时，两步之后世界是干净的;没有任何代理可以做得更好。 当两个位置都有污垢时，三步之后世界就干净了;没有任何代理可以做得更好。 （注意：一般来说，这个答案的第一句话中陈述的条件比代理人理性所必需的要严格得多。）

b. （A）中的代理即使在世界清洁之后也会继续前后移动。 这是更好地做NoOp一旦世界是干净的（章节说这）。 现在，由于代理的percept没有说明另一个方块是否干净，因此代理似乎必须有一些记忆来说明另一个方块是否已经被清理。 要使这个论点严谨是比较困难的—例如，代理可以安排的东西，以便它只会在一个干净的左广场时，右广场

EXTERNAL MEMORY

was already clean? As a general strategy, an agent *can* use the environment itself as a form of **external memory**—a common technique for humans who use things like appointment calendars and knots in handkerchiefs. In this particular case, however, that is not possible. Consider the reflex actions for $[A, Clean]$ and $[B, Clean]$. If either of these is $NoOp$, then the agent will fail in the case where that is the initial percept but the other square is dirty; hence, neither can be $NoOp$ and therefore the simple reflex agent is doomed to keep moving. In general, the problem with reflex agents is that they have to do the same thing in situations that look the same, even when the situations are actually quite different. In the vacuum world this is a big liability, because every interior square (except home) looks either like a square with dirt or a square without dirt.

c. If we consider asymptotically long lifetimes, then it is clear that learning a map (in some form) confers an advantage because it means that the agent can avoid bumping into walls. It can also learn where dirt is most likely to accumulate and can devise an optimal inspection strategy. The precise details of the exploration method needed to construct a complete map appear in Chapter 4; methods for deriving an optimal inspection/cleanup strategy are in Chapter 21.

2.3

a. *An agent that senses only partial information about the state cannot be perfectly rational.*
False. Perfect rationality refers to the ability to make good decisions given the sensor information received.

b. *There exist task environments in which no pure reflex agent can behave rationally.*
True. A pure reflex agent ignores previous percepts, so cannot obtain an optimal state estimate in a partially observable environment. For example, correspondence chess is played by sending moves; if the other player's move is the current percept, a reflex agent could not keep track of the board state and would have to respond to, say, "a4" in the same way regardless of the position in which it was played.

c. *There exists a task environment in which every agent is rational.*
True. For example, in an environment with a single state, such that all actions have the same reward, it doesn't matter which action is taken. More generally, any environment that is reward-invariant under permutation of the actions will satisfy this property.

d. *The input to an agent program is the same as the input to the agent function.*
False. The agent function, notionally speaking, takes as input the entire percept sequence up to that point, whereas the agent program takes the current percept only.

e. *Every agent function is implementable by some program/machine combination.*
False. For example, the environment may contain Turing machines and input tapes and the agent's job is to solve the halting problem; there is an agent *function* that specifies the right answers, but no agent program can implement it. Another example would be an agent function that requires solving intractable problem instances of arbitrary size in constant time.

外部存储器

已经干净了吗？ 作为一般策略，代理可以将环境本身用作外部记忆的一种形式—这是人类使用以下内容的常用技术
预约日历和手帕结。 然而，在这种特殊情况下，这是不可能的。 考虑[A，清洁]和[B，清洁]的反射动作。 如果其中任何一个是NoOp，那么代理将在初始感知但另一个方块是脏的情况下失败;因此，两者都不能是NoOp，因此简单的反射代理注定要继续移动。 一般来说，反射代理的问题是，他们必须在看起来相同的情况下做同样的事情，即使情况实际上是完全不同的。 在真空世界中，这是一个很大的负担，因为每个内部广场（除了家庭）看起来要么像一个有污垢的广场，要么像一个没有污垢的广场。 c. 如果我们考虑渐近的长寿命，那么很明显，学习地图（以某种形式）会带来优势，因为这意味着代理可以避免撞到墙壁。 它还可以了解污垢最可能积聚的位置，并可以制定最佳检测策略。 构建完整地图所需的勘探方法的精确细节见第4章；推导最佳检查/清理策略的方法见第21章。

2.3

a. 一个只感知有关国家的部分信息的代理不可能是完美的。 错误。 完美理性是指在接收到的传感器信息的情况下做出良好决策的能力。 b. 存在没有纯反射剂可以合理行为的任务环境。

没错。 纯反射代理忽略先前的感知，因此无法在部分可观察的环境中获得最佳状态估计。 例如，对应棋是通过发送动作来进行的；如果另一个棋手的动作是当前的感知器，反射代理就不能跟踪棋盘的状态，并且必须以同样的方式对"a4"做出反应，而不管它的位置如何。 c. 存在一个任务环境，其中每个代理都是理性的。

没错。 例如，在具有单一状态的环境中，使得所有操作都具有相同的奖励，则采取哪种操作并不重要。 更一般地说，任何在操作排列下是奖励不变的环境都将满足此属性。 d. 代理程序的输入与代理函数的输入相同。

错误。 从概念上讲，代理函数将整个percept se quence作为输入，而代理程序只接受当前的percept。 e. 每个代理功能都可以通过一些程序/机器组合来实现。

错误。 例如，环境可能包含图灵机和输入磁带，代理的工作是解决停止问题;有一个代理函数指定正确的答案，但没有代理程序可以实现它。 另一个例子是需要在恒定时间内求解任意大小的棘手问题实例的代理函数。

f. *Suppose an agent selects its action uniformly at random from the set of possible actions. There exists a deterministic task environment in which this agent is rational.*
True. This is a special case of (c); if it doesn't matter which action you take, selecting randomly is rational.

g. *It is possible for a given agent to be perfectly rational in two distinct task environments.*
True. For example, we can arbitrarily modify the parts of the environment that are unreachable by any optimal policy as long as they stay unreachable.

h. *Every agent is rational in an unobservable environment.*
False. Some actions are stupid—and the agent may know this if it has a model of the environment—even if one cannot perceive the environment state.

i. *A perfectly rational poker-playing agent never loses.*
False. Unless it draws the perfect hand, the agent can always lose if an opponent has better cards. This can happen for game after game. The correct statement is that the agent's expected winnings are nonnegative.

2.4 Many of these can actually be argued either way, depending on the level of detail and abstraction.

A. Partially observable, stochastic, sequential, dynamic, continuous, multi-agent.
B. Partially observable, stochastic, sequential, dynamic, continuous, single agent (unless there are alien life forms that are usefully modeled as agents).
C. Partially observable, deterministic, sequential, static, discrete, single agent. This can be multi-agent and dynamic if we buy books via auction, or dynamic if we purchase on a long enough scale that book offers change.
D. Fully observable, stochastic, episodic (every point is separate), dynamic, continuous, multi-agent.
E. Fully observable, stochastic, episodic, dynamic, continuous, single agent.
F. Fully observable, stochastic, sequential, static, continuous, single agent.
G. Fully observable, deterministic, sequential, static, continuous, single agent.
H. Fully observable, strategic, sequential, static, discrete, multi-agent.

2.5 The following are just some of the many possible definitions that can be written:

- *Agent*: an entity that perceives and acts; or, one that *can be viewed* as perceiving and acting. Essentially any object qualifies; the key point is the way the object implements an agent function. (Note: some authors restrict the term to *programs* that operate *on behalf of* a human, or to programs that can cause some or all of their code to run on other machines on a network, as in **mobile agents**.)
- *Agent function*: a function that specifies the agent's action in response to every possible percept sequence.
- *Agent program*: that program which, combined with a machine architecture, implements an agent function. In our simple designs, the program takes a new percept on each invocation and returns an action.

MOBILE AGENT

f. 假设一个代理从一组可能的动作中随机地统一选择其动作。 存在一个确定性任务环境，其中该代理是理性的。

没错。 这是（c）的一个特例;如果你采取哪种行动无关紧要，随机选择是合理的。 g. 一个给定的代理在两个不同的任务环境中是完全合理的是可能的。 没错。例如，我们可以任意修改任何最优策略无法访问的环境部分，只要它们保持不可访问状态。 h. 在不可观察的环境中，每个代理都是理性的。

错误。 有些操作是愚蠢的--如果代理有一个环境模型，它可能会知道这一点--即使你不能感知环境状态。 i. 一个完全理性的扑克代理永远不会输。

错误。 除非它画出完美的手，否则如果对手有更好的牌，代理总是会输。 这可能发生在一场又一场的比赛中. 正确的说法是代理人的预期奖金是非负性的。

2.4其中许多实际上可以根据细节和抽象的程度来论证。

A.部分可观察，随机，顺序，动态，连续，多代理。
B.部分可观察的，随机的，顺序的，动态的，连续的，单一的代理（除非有有用地建模为代理的外星生命形式）。 C.部分可观察，确定性，顺序，静态，离散，单一代理。 这可以是多代理和动态的，如果我们通过拍卖购买书籍，或者动态的，如果我们购买足够长的规模，书籍提供变化。 D.完全可观察的，随机的，偶发的（每个点都是分开的），动态的，连续的，多代理的。 E.完全可观察的，随机的，偶发的，动态的，连续的，单一的代理。

F.完全可观察，随机，顺序，静态，连续，单一代理。
G.完全可观察，确定性，顺序，静态，连续，单一代理。
H.完全可观察，战略，顺序，静态，离散，多代理。

2.5 以下只是可以编写的许多可能定义中的一些:

*代理：感知和行动的实体；或者，可以被视为感知和行动的实体。 基本上任何对象都符合条件;关键点是对象实现代理功能的方式。 （注意：一些作者将该术语限制为代表人操作的程序，或者限制为可能导致其部分或全部代码在网络上的其他计算机上运行的程序，例如在移动代理中。）

流动代理

*代理函数：指定代理响应每个可能的percept序列的动作的函数。 *代理程序：与机器体系结构相结合，具有代理功能的程序。 在我们的简单设计中，程序对每个调用进行新的感知并返回一个操作。

- *Rationality*: a property of agents that choose actions that maximize their expected utility, given the percepts to date.
- *Autonomy*: a property of agents whose behavior is determined by their own experience rather than solely by their initial programming.
- *Reflex agent*: an agent whose action depends only on the current percept.
- *Model-based agent*: an agent whose action is derived directly from an internal model of the current world state that is updated over time.
- *Goal-based agent*: an agent that selects actions that it believes will achieve explicitly represented goals.
- *Utility-based agent*: an agent that selects actions that it believes will maximize the expected utility of the outcome state.
- *Learning agent*: an agent whose behavior improves over time based on its experience.

2.6 Although these questions are very simple, they hint at some very fundamental issues. Our answers are for the simple agent designs for *static* environments where nothing happens while the agent is deliberating; the issues get even more interesting for dynamic environments.

a. Yes; take any agent program and insert null statements that do not affect the output.

b. Yes; the agent function might specify that the agent print $true$ when the percept is a Turing machine program that halts, and $false$ otherwise. (Note: in dynamic environments, for machines of less than infinite speed, the rational agent function may not be implementable; e.g., the agent function that always plays a winning move, if any, in a game of chess.)

c. Yes; the agent's behavior is fixed by the architecture and program.

d. There are 2^n agent programs, although many of these will not run at all. (Note: Any given program can devote at most n bits to storage, so its internal state can distinguish among only 2^n past histories. Because the agent function specifies actions based on percept histories, there will be many agent functions that cannot be implemented because of lack of memory in the machine.)

e. It depends on the program and the environment. If the environment is dynamic, speeding up the machine may mean choosing different (perhaps better) actions and/or acting sooner. If the environment is static and the program pays no attention to the passage of elapsed time, the agent function is unchanged.

2.7

The design of goal- and utility-based agents depends on the structure of the task environment. The simplest such agents, for example those in chapters 3 and 10, compute the agent's entire future sequence of actions in advance before acting at all. This strategy works for static and deterministic environments which are either fully-known or unobservable

For fully-observable and fully-known static environments a policy can be computed in advance which gives the action to by taken in any given state.

*合理性：考虑到迄今为止的感知，选择最大化其预期效用的行为的代理人的属性。 *自治：代理的一种属性，其行为取决于他们自己的经验，而不仅仅是他们最初的编程。 *反射代理：其作用仅取决于当前感知的代理。 *基于模型的代理：一个代理，其操作直接源自随时间更新的当前世界状态的内部模型。 •基于目标的代理：选择它认为将实现明确表示的目标的操作的代理。 *基于效用的代理：选择它认为将最大化结果状态的预期效用的操作的代理。 *学习代理：根据其经验，其行为随着时间的推移而改善的代理。

2.6 这些问题虽然很简单，但却暗示了一些非常根本的问题。
我们的答案是针对静态环境的简单代理设计，在静态环境中，代理在考虑时什么也不发生；对于动态环境来说，这些问题变得更加有趣。

a. 是;采取任何代理程序并插入不影响输出的null语句。 b. 是；代理函数可能指定代理在percept是停止的图灵机程序时打印true，否则打印false。 （注意：在动态环境中，对于速度小于无限的机器，rational agent函数可能无法实现;例如，在国际象棋游戏中始终播放获胜动作的agent函数（如果有的话）。)c． 是；代理的行为由体系结构和程序固定。 d. 有2n个代理程序，尽管其中许多程序根本不会运行。 （注意：任何给定的程序最多可以投入n位存储，因此其内部状态只能区分2n个过去的历史。 由于代理函数根据percept历史记录指定操作，因此会有许多代理函数由于机器内存不足而无法实现。)e． 这取决于程序和环境。 如果环境是动态的，加速机器可能意味着选择不同的（也许更好的）动作和/或更快地采取行动。 如果环境是静态的，并且程序不关注经过的时间的流逝，则代理函数不变。

2.7

基于目标和实用程序的代理的设计取决于任务环境的结构。 最简单的代理，例如第3章和第10章中的代理，在采取行动之前预先计算代理的整个未来行动序列。 此策略适用于完全已知或不可观察的静态和确定性环境

对于完全可观察和完全已知的静态环境，可以预先计算策略，该策略可以在任何给定状态下执行操作。

```
function GOAL-BASED-AGENT(percept) returns an action
  persistent: state, the agent's current conception of the world state
              model, a description of how the next state depends on current state and action
              goal, a description of the desired goal state
              plan, a sequence of actions to take, initially empty
              action, the most recent action, initially none

  state ← UPDATE-STATE(state, action, percept, model)
  if GOAL-ACHIEVED(state,goal) then return a null action
  if plan is empty then
plan ← PLAN(state,goal,model)
  action ← FIRST(plan)
  plan ← REST(plan)
  return action
```

Figure S2.1 A goal-based agent.

For partially-observable environments the agent can compute a conditional plan, which specifies the sequence of actions to take as a function of the agent's perception. In the extreme, a conditional plan gives the agent's response to every contingency, and so it is a representation of the entire agent function.

In all cases it may be either intractable or too expensive to compute everything out in advance. Instead of a conditional plan, it may be better to compute a single sequence of actions which is likely to reach the goal, then monitor the environment to check whether the plan is succeeding, repairing or replanning if it is not. It may be even better to compute only the start of this plan before taking the first action, continuing to plan at later time steps.

Pseudocode for simple goal-based agent is given in Figure S2.1. GOAL-ACHIEVED tests to see whether the current state satisfies the goal or not, doing nothing if it does. PLAN computes a sequence of actions to take to achieve the goal. This might return only a prefix of the full plan, the rest will be computed after the prefix is executed. This agent will act to maintain the goal: if at any point the goal is not satisfied it will (eventually) replan to achieve the goal again.

At this level of abstraction the utility-based agent is not much different than the goal-based agent, except that action may be continuously required (there is not necessarily a point where the utility function is "satisfied"). Pseudocode is given in Figure S2.2.

2.8 The file `"agents/environments/vacuum.lisp"` in the code repository implements the vacuum-cleaner environment. Students can easily extend it to generate different shaped rooms, obstacles, and so on.

2.9 A reflex agent program implementing the rational agent function described in the chapter is as follows:

```
(defun reflex-rational-vacuum-agent (percept)
  (destructuring-bind (location status) percept
```



```
函数G OAL-B ASED-A GENT(percept)返回一个动作
  persistent：状态，代理人对世界状态的当前概念
              模型，下一个状态如何取决于当前状态和行动目标的描述，期望目标
              状态计划的描述，要采取的行动序列，最初为空的行动，最近的行动，
              最初为无


 state←U PDATE-S TATE(state,action,percept,model)if
G OAL-A CHIEVED(state,goal)then return a null action i
f plan is empty then plan←P LAN(state,goal,model)acti
on←F IRST(plan)plan←R EST(plan)return action
```

Figure S2.1 A goal-based agent.

对于部分可观察的环境，代理可以计算条件计划，该计划指定作为代理感知函数要采取的操作序列。 在ex treme中，条件计划给出了代理对每个意外事件的响应，因此它是对整个代理功能的重新陈述。

在所有情况下，提前计算所有内容可能是棘手的或太昂贵的。 而不是一个有条件的计划，它可能是更好地计算一个单一的行动序列，这是可能达到的目标，然后监视环境，以检查计划是否成功，修复或重新规划，如果它 在采取第一个行动之前，只计算这个计划的开始，然后在以后的时间步骤继续计划可能会更好。

简单的基于目标的代理的伪代码在图S2.1中给出。 G OAL-A CHIEVED测试，看看当前状态是否满足目标与否，如果它做什么都不做。 P LAN计算为实现目标而要采取的一系列操作。 这可能只返回完整计划的前缀，其余的将在前缀执行后计算。 这个代理将采取行动来维持目标：如果在任何时候目标不满意，它将（最终）重新计划以再次实现目标。

在这个抽象级别上，基于效用的代理与基于目标的代理没有太大的不同，除了可能持续需要行动（不一定存在效用函数"满足"的点）。 伪代码在图S2.2中给出。

2.8文件"代理/环境/真空。lisp"在代码存储库imple ments真空吸尘器环境。 学生可以很容易地扩展它以生成不同形状的房间，障碍物等等。

2.9实现第三章中描述的理性代理功能的反射代理程序如下:

```
(defun reflex-rational-vacuum-agent (percept)
  (destructuring-bind (location status) percept
```

function UTILITY-BASED-AGENT(*percept*) **returns** an action
  persistent: *state*, the agent's current conception of the world state
    model, a description of how the next state depends on current state and action
    $utility - function$, a description of the agent's utility function
    plan, a sequence of actions to take, initially empty
    action, the most recent action, initially none

state ← UPDATE-STATE(*state*, *action*, *percept*, *model*)
if *plan* is empty **then**
plan ← PLAN(*state*,$utility - function$,*model*)
action ← FIRST(*plan*)
plan ← REST(*plan*)
return *action*

Figure S2.2 A utility-based agent.

```
(cond ((eq status 'Dirty) 'Suck)
      ((eq location 'A) 'Right)
      (t 'Left))))
```

For states 1, 3, 5, 7 in Figure 4.9, the performance measures are 1996, 1999, 1998, 2000 respectively.

2.10

a. No; see answer to 2.4(b).

b. See answer to 2.4(b).

c. In this case, a simple reflex agent can be perfectly rational. The agent can consist of a table with eight entries, indexed by percept, that specifies an action to take for each possible state. After the agent acts, the world is updated and the next percept will tell the agent what to do next. For larger environments, constructing a table is infeasible. Instead, the agent could run one of the optimal search algorithms in Chapters 3 and 4 and execute the first step of the solution sequence. Again, no internal state is *required*, but it would help to be able to store the solution sequence instead of recomputing it for each new percept.

2.11

a. Because the agent does not know the geography and perceives only location and local dirt, and cannot remember what just happened, it will get stuck forever against a wall when it tries to move in a direction that is blocked—that is, unless it randomizes.

b. One possible design cleans up dirt and otherwise moves randomly:

```
(defun randomized-reflex-vacuum-agent (percept)
  (destructuring-bind (location status) percept
    (cond ((eq status 'Dirty) 'Suck)
          (t (random-element '(Left Right Up Down))))))
```


函数U TILITY-B ASED-A GENT(percept)返回一个动作
  persistent：状态，代理人对世界状态的当前概念
    模型，下一个状态如何依赖于当前状态和动作效用函数的描述，代理的效用函数计划的描述，要采取的动作序列，最初为空的动作，最近的动作，最初为无

state←U PDATE-S TATE(state,action,percept,model)如果plan为空则plan←P LAN(state,utility-function,model)action←F IRST(plan)plan←R EST(plan)return action

Figure S2.2 A utility-based agent.

（cond（（eq状态'脏）'吸）（（eq位置'A'）'右）（t'左））））

对于图4.9中的国家1、3、5、7，业绩计量分别为1996年、1999年、1998年和2000年。

2.10

a. 否；见对2.4(b)的答复。 b. 见对2.4(b)的答复。 c. 在这种情况下，一个简单的反射剂可以是完全合理的。 代理可以由一个表组成，其中包含八个条目，由percept索引，该表指定要为每个可能的状态采取的操作。 代理行为后，世界被更新，下一个percept会告诉代理下一步该做什么。 对于较大的环境，构建表是不可行的。 相反，代理可以运行第3章和第4章中的最佳搜索算法之一，并执行解决方案序列的第一步。 同样，不需要内部状态，但能够存储解决方案序列而不是为每个新percept重新计算它会有所帮助。

2.11

a. 因为代理不知道地理，只感知位置和局部污垢，并且不记得刚刚发生的事情，所以当它试图向被阻挡的方向移动时，它会永远卡在墙上—也就是说，除非它随 b. 一个可能的设计清理污垢，否则随机移动:

（defun randomized-reflex-vacuum-agent（percept）（destructuring-bind（location status）percept（cond（（eq status'Dirty）'Suck）
(t(random-element'(Left Right Up Down))))))

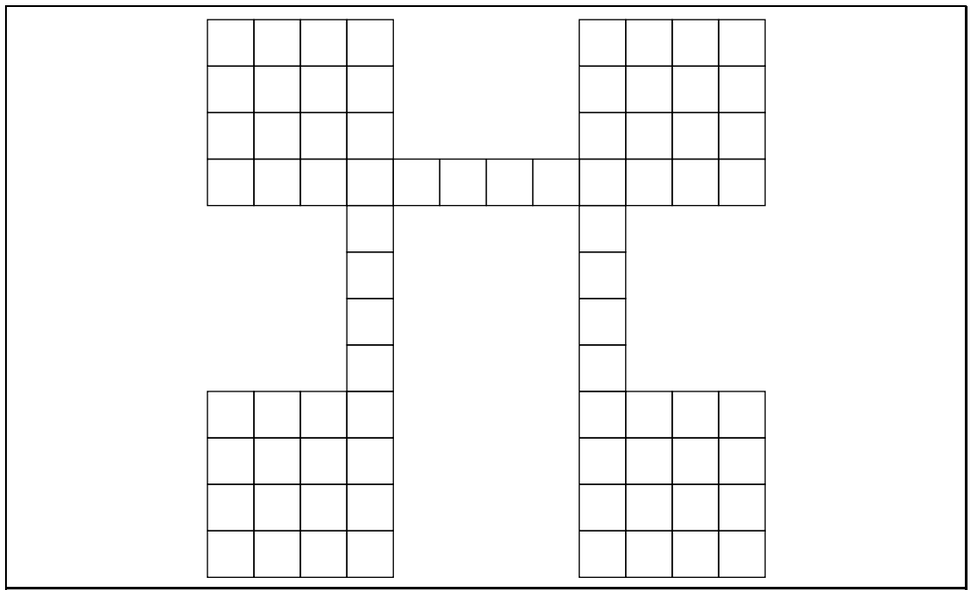


Figure S2.3 An environment in which random motion will take a long time to cover all the squares.

This is fairly close to what the Roomba^{TM} vacuum cleaner does (although the Roomba has a bump sensor and randomizes only when it hits an obstacle). It works reasonably well in nice, compact environments. In maze-like environments or environments with small connecting passages, it can take a very long time to cover all the squares.

c. An example is shown in Figure S2.3. Students may also wish to measure clean-up time for linear or square environments of different sizes, and compare those to the efficient online search algorithms described in Chapter 4.

d. A reflex agent with state can build a map (see Chapter 4 for details). An online depth-first exploration will reach every state in time linear in the size of the environment; therefore, the agent can do much better than the simple reflex agent.

The question of rational behavior in unknown environments is a complex one but it is worth encouraging students to think about it. We need to have some notion of the prior probability distribution over the class of environments; call this the initial **belief state**. Any action yields a new percept that can be used to update this distribution, moving the agent to a new belief state. Once the environment is completely explored, the belief state collapses to a single possible environment. Therefore, the problem of optimal exploration can be viewed as a search for an optimal strategy in the space of possible belief states. This is a well-defined, if horrendously intractable, problem. Chapter 21 discusses some cases where optimal exploration is possible. Another concrete example of exploration is the Minesweeper computer game (see Exercise 7.22). For very small Minesweeper environments, optimal exploration is feasible although the belief state

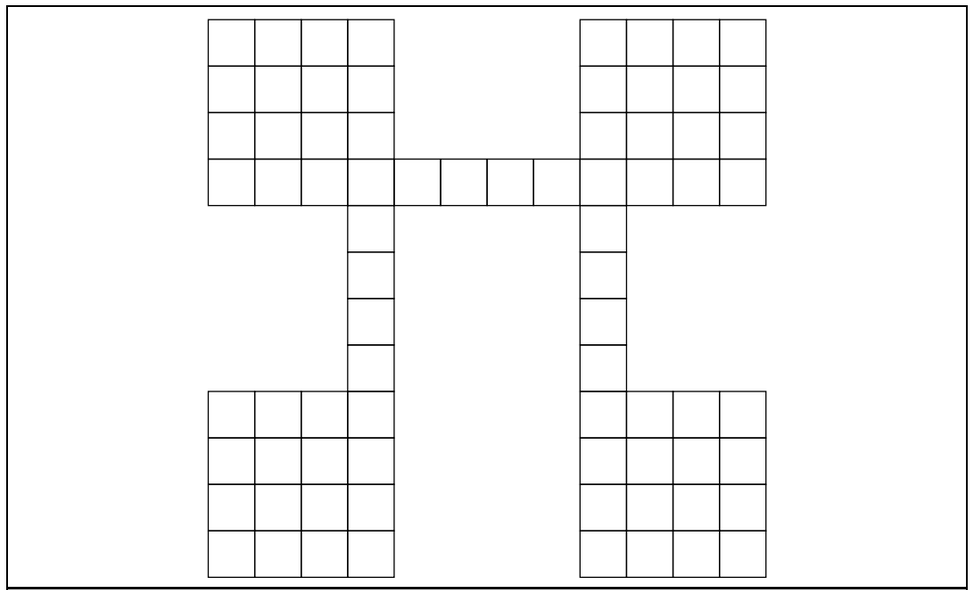


Figure S2.3 随机运动需要很长时间才能复盖所有的环境
广场。

这与Roomba TM真空吸尘器的功能相当接近（尽管Roomba有一个凹凸传感器，并且只有在碰到障碍物时才会随机化）。 它在漂亮，紧凑的环境中工作得相当好。 在迷宫般的环境或具有小的连接通道的环境中，可能需要非常长的时间来复盖所有的方块。 c. 示例如图S2.3所示。 学生可能还希望测量不同大小的线性或方形环境的清理时间，并将其与第4章中描述的高效在线搜索算法进行比较。 d. 具有状态的反射代理可以构建一个地图（详见第4章）。 在线深度优先探索将在时间上达到环境大小的每个状态;因此，代理可以比简单的反射代理做得更好。

在未知环境下的理性行为问题是一个复杂的问题，但值得鼓励学生思考。 我们需要有一些关于环境类上的先验概率分布的概念;称之为初始信念状态。 任何操作都会产生一个新的percept，可用于更新此分布，将代理移动到新的信念状态。 一旦环境被完全探索，信仰状态就会崩溃到一个可能的环境中。 因此，最优探索问题可以被看作是在可能的信念状态空间中寻找最优策略。 这是一个明确的，如果可怕的棘手的问题。 第21章讨论了可能进行最优探索的一些情况。 探索的另一个具体例子是扫雷电脑游戏（见练习7.22）。 对于非常小的扫雷环境，最佳探索是可行的，尽管信念状态

update is nontrivial to explain.

2.12 The problem appears at first to be very similar; the main difference is that instead of using the location percept to build the map, the agent has to "invent" its own locations (which, after all, are just nodes in a data structure representing the state space graph). When a bump is detected, the agent assumes it remains in the same location and can add a wall to its map. For grid environments, the agent can keep track of its (x, y) location and so can tell when it has returned to an old state. In the general case, however, there is no simple way to tell if a state is new or old.

2.13

a. For a reflex agent, this presents no *additional* challenge, because the agent will continue to $Suck$ as long as the current location remains dirty. For an agent that constructs a sequential plan, every $Suck$ action would need to be replaced by "$Suck$ until clean." If the dirt sensor can be wrong on each step, then the agent might want to wait for a few steps to get a more reliable measurement before deciding whether to $Suck$ or move on to a new square. Obviously, there is a trade-off because waiting too long means that dirt remains on the floor (incurring a penalty), but acting immediately risks either dirtying a clean square or ignoring a dirty square (if the sensor is wrong). A rational agent must also continue touring and checking the squares in case it missed one on a previous tour (because of bad sensor readings). it is not immediately obvious how the waiting time at each square should change with each new tour. These issues can be clarified by experimentation, which may suggest a general trend that can be verified mathematically. This problem is a partially observable Markov decision process—see Chapter 17. Such problems are hard in general, but some special cases may yield to careful analysis.

b. In this case, the agent must keep touring the squares indefinitely. The probability that a square is dirty increases monotonically with the time since it was last cleaned, so the rational strategy is, roughly speaking, to repeatedly execute the shortest possible tour of all squares. (We say "roughly speaking" because there are complications caused by the fact that the shortest tour may visit some squares twice, depending on the geography.) This problem is also a partially observable Markov decision process.

更新是无关紧要的解释。

2.12问题一开始看起来非常相似；主要区别在于，代理必须"发明"自己的位置（毕竟，这些位置只是表示状态空间图的数据结构中的节点），而不是使用位置感知器来构建地图。当检测到凹凸时，代理假定它保持在同一位置，并可以向其地图添加墙。对于网格环境，代理可以跟踪其（x，y）位置，因此可以告诉它何时返回到旧状态。然而，在一般情况下，没有简单的方法来判断一个国家是新的还是旧的。

2.13

a. 对于反射代理来说，这没有额外的挑战，因为只要当前位置保持脏，代理就会继续吸。对于构建顺序计划的代理，每个吸操作都需要用"吸到干净"来代替。"如果污垢传感器在每个步骤上都可能出错，那么代理商可能需要等待几步才能获得更可靠的测量，然后再决定是否吸到或移动到新的正方形。显然，有一个权衡，因为等待太长时间意味着污垢留在地板上（招致惩罚），但立即采取行动的风险要么弄脏一个干净的正方形，要么忽略一个肮脏的正方形（一个理性的代理还必须继续巡回演出并检查方块，以防它在前一次巡回演出中错过了一个（因为传感器读数不好）。每个广场的等待时间应该如何随着每个新的旅游而变化并不是很明显。这些问题可以通过实验来澄清，这可能表明可以通过数学验证的一般趋势。这个问题是一个部分可观察到的马尔可夫决策过程-见第17章。这些问题一般都很难，但有些特殊情况可能需要仔细分析。 b. 在这种情况下，代理必须无限期地巡回演出广场。一个正方形是脏的概率随着自上次清理以来的时间单调增加，因此合理的策略是，粗略地说，重复执行所有正方形的最短可能游。（我们说"粗略地说"，因为有一个事实，即最短的游览可能会访问一些广场两次，这取决于地理造成的并发症。）这个问题也是一个部分可观察的马尔可夫决策过程。

Solutions for Chapter 3
Solving Problems by Searching

3.1 In goal formulation, we decide which aspects of the world we are interested in, and which can be ignored or abstracted away. Then in problem formulation we decide how to manipulate the important aspects (and ignore the others). If we did problem formulation first we would not know what to include and what to leave out. That said, it can happen that there is a cycle of iterations between goal formulation, problem formulation, and problem solving until one arrives at a sufficiently useful and efficient solution.

3.2

a. We'll define the coordinate system so that the center of the maze is at $(0, 0)$, and the maze itself is a square from $(-1, -1)$ to $(1, 1)$.
Initial state: robot at coordinate $(0, 0)$, facing North.
Goal test: either $|x| > 1$ or $|y| > 1$ where (x, y) is the current location.
Successor function: move forwards any distance d; change direction robot it facing.
Cost function: total distance moved.

The state space is infinitely large, since the robot's position is continuous.

b. The state will record the intersection the robot is currently at, along with the direction it's facing. At the end of each corridor leaving the maze we will have an exit node. We'll assume some node corresponds to the center of the maze.
Initial state: at the center of the maze facing North.
Goal test: at an exit node.
Successor function: move to the next intersection in front of us, if there is one; turn to face a new direction.
Cost function: total distance moved.

There are $4n$ states, where n is the number of intersections.

c. Initial state: at the center of the maze.
Goal test: at an exit node.
Successor function: move to next intersection to the North, South, East, or West.
Cost function: total distance moved.

We no longer need to keep track of the robot's orientation since it is irrelevant to

第3章的解决方案
通过搜索解决问题

3.1在目标制定中，我们决定我们对世界的哪些方面感兴趣，哪些方面可以忽略或抽象。 然后在问题制定中，我们决定如何操纵重要方面（而忽略其他方面）。 如果我们首先提出问题，我们就不知道应该包括什么，应该遗漏什么。 也就是说，可能会发生目标制定，问题制定和解决问题之间的迭代循环，直到一个人到达一个足够有用和有效的解决方案。

3.2

我们将定义坐标系，使迷宫的中心在（0，0），迷宫本身是一个从（-1，-1）到（1，1）的正方形。
初始状态:机器人在坐标(0,0),朝北.
目标测试：/x/>1或/y/>1其中（x，y）是当前位置。
后继功能：向前移动任何距离d;改变朝向它的方向。
成本函数：移动的总距离。

状态空间无限大，因为机器人的位置是连续的。 状态将记录机器人当前所在的十字路口，以及它所面对的方向。 在每个离开迷宫的走廊的尽头，我们将有一个出口节点。 我们假设某个节点对应于迷宫的中心。

初始状态：在迷宫中心朝北。
目标测试：在出口节点。
后继功能：移动到我们面前的下一个十字路口，如果有一个;转向面对新的方向。 成本函数：移动的总距离。

有4个n状态，其中n是交叉点的数量。 C.初始状态：在迷宫的中心。 目标测试：在出口节点。

后继功能：移至北、南、东或西的下一个路口。
成本函数：移动的总距离。

我们不再需要跟踪机器人的方向，因为它与

predicting the outcome of our actions, and not part of the goal test. The motor system that executes this plan will need to keep track of the robot's current orientation, to know when to rotate the robot.

d. State abstractions:

(i) Ignoring the height of the robot off the ground, whether it is tilted off the vertical.
(ii) The robot can face in only four directions.
(iii) Other parts of the world ignored: possibility of other robots in the maze, the weather in the Caribbean.

Action abstractions:

(i) We assumed all positions we safely accessible: the robot couldn't get stuck or damaged.
(ii) The robot can move as far as it wants, without having to recharge its batteries.
(iii) Simplified movement system: moving forwards a certain distance, rather than controlled each individual motor and watching the sensors to detect collisions.

3.3

a. State space: States are all possible city pairs (i, j). The map is *not* the state space.

Successor function: The successors of (i, j) are all pairs (x, y) such that $Adjacent(x, i)$ and $Adjacent(y, j)$.

Goal: Be at (i, i) for some i.

Step cost function: The cost to go from (i, j) to (x, y) is $max(d(i, x), d(j, y))$.

b. In the best case, the friends head straight for each other in steps of equal size, reducing their separation by twice the time cost on each step. Hence (iii) is admissible.

c. Yes: e.g., a map with two nodes connected by one link. The two friends will swap places forever. The same will happen on any chain if they start an odd number of steps apart. (One can see this best on the graph that represents the state space, which has two disjoint sets of nodes.) The same even holds for a grid of any size or shape, because every move changes the Manhattan distance between the two friends by 0 or 2.

d. Yes: take any of the unsolvable maps from part (c) and add a self-loop to any one of the nodes. If the friends start an odd number of steps apart, a move in which one of the friends takes the self-loop changes the distance by 1, rendering the problem solvable. If the self-loop is not taken, the argument from (c) applies and no solution is possible.

3.4 From http://www.cut-the-knot.com/pythagoras/fifteen.shtml, this proof applies to the fifteen puzzle, but the same argument works for the eight puzzle:

Definition: The goal state has the numbers in a certain order, which we will measure as starting at the upper left corner, then proceeding left to right, and when we reach the end of a row, going down to the leftmost square in the row below. For any other configuration besides the goal, whenever a tile with a greater number on it precedes a tile with a smaller number, the two tiles are said to be **inverted**.

Proposition: For a given puzzle configuration, let N denote the sum of the total number of inversions and the row number of the empty square. Then $(Nmod2)$ is invariant under any

预测我们行动的结果，而不是目标测试的一部分。 执行此计划的电机系统需要跟踪机器人的当前方向，以了解何时旋转机器人。 d.状态抽象:

(i)忽略机器人离地面的高度，是否垂直倾斜。(ii)机器人只能面向四个方向。 （三）世界其他地区被忽视：迷宫中其他机器人的可能性，加勒比海的天气。

Action abstractions:

（i）我们假设我们可以安全到达的所有位置：机器人不会卡住或损坏。(ii)机器人可以随心所欲地移动，而不必给电池充电。 （iii）简化的运动系统：向前移动一定距离，而不是控制每个单独的电机并观察传感器以检测碰撞。

3.3

a. 状态空间：状态都是可能的城市对（i，j）。 地图不是状态空间。
后继函数：（i，j）的后继者都是对（x，y），使得相邻（x，i）和相邻（y，j）。
目标：在（我，我）为一些我。
步进成本函数：从（i，j）到（x，y）的成本是max（d（i，x），d（j，y））。
b. 在最好的情况下，朋友们以相同大小的步骤直奔对方，将他们的分离减少两倍的时间成本在每个步骤。 因此(iii)是可以接受的。 c. 是的：例如，一个由一个链接连接的两个节点的地图。 这两个朋友将永远交换位置。 如果它们分开开始奇数步，则在任何链上都会发生同样的情况。 （人们可以在表示状态空间的图形上看到这一点，该图形具有两个不相交的节点集。）对于任何大小或形状的网格也是如此，因为一举一动都会使两个朋友之间的曼哈顿距离改变0或2。 d. 是：从部分（c）中获取任何不可解的映射，并将自循环添加到节点中的任何一个。 如果朋友开始相隔奇数步，则其中一个朋友采取自循环的移动会将距离更改1，从而使问题可解。 如果不采取自循环，则来自（c）的参数适用，并且没有解决方案是可能的。

3.4从http://www.cut-the-knot.com/pythagoras/fifteen.shtml，这个证明适用于十五个难题，但同样的论点适用于八个难题:

定义：目标状态有一定顺序的数字，我们将测量为从左上角开始，然后从左到右，当我们到达一行的末尾时，向下到下面行中最左边的正方形。 对于除目标之外的任何其他配置，每当其上具有更大数量的瓦片在具有更小数量的瓦片之前时，两个瓦片被说成是倒置的。

命题：对于给定的拼图配置，让N表示反转的总数和空正方形的行号之和。 然后（N mod2）是不变的，在任何

legal move. In other words, after a legal move an odd N remains odd whereas an even N remains even. Therefore the goal state in Figure 3.4, with no inversions and empty square in the first row, has $N = 1$, and can only be reached from starting states with odd N, not from starting states with even N.

Proof: First of all, sliding a tile horizontally changes neither the total number of inversions nor the row number of the empty square. Therefore let us consider sliding a tile vertically.

Let's assume, for example, that the tile A is located directly over the empty square. Sliding it down changes the parity of the row number of the empty square. Now consider the total number of inversions. The move only affects relative positions of tiles A, B, C, and D. If none of the B, C, D caused an inversion relative to A (i.e., all three are larger than A) then after sliding one gets three (an odd number) of additional inversions. If one of the three is smaller than A, then before the move B, C, and D contributed a single inversion (relative to A) whereas after the move they'll be contributing two inversions - a change of 1, also an odd number. Two additional cases obviously lead to the same result. Thus the change in the sum N is always even. This is precisely what we have set out to show.

So before we solve a puzzle, we should compute the N value of the start and goal state and make sure they have the same parity, otherwise no solution is possible.

3.5 The formulation puts one queen per column, with a new queen placed only in a square that is not attacked by any other queen. To simplify matters, we'll first consider the n–rooks problem. The first rook can be placed in any square in column 1 (n choices), the second in any square in column 2 except the same row that as the rook in column 1 ($n-1$ choices), and so on. This gives $n!$ elements of the search space.

For n queens, notice that a queen attacks at most three squares in any given column, so in column 2 there are at least $(n-3)$ choices, in column at least $(n-6)$ choices, and so on. Thus the state space size $S \geq n \cdot (n-3) \cdot (n-6) \cdots$. Hence we have

$$\begin{aligned} S^3 &\geq n \cdot n \cdot n \cdot (n-3) \cdot (n-3) \cdot (n-3) \cdot (n-6) \cdot (n-6) \cdot (n-6) \cdot \cdots \\ &\geq n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdot (n-4) \cdot (n-5) \cdot (n-6) \cdot (n-7) \cdot (n-8) \cdot \cdots \\ &= n! \end{aligned}$$

or $S \geq \sqrt[3]{n!}$.

3.6

a. Initial state: No regions colored.
Goal test: All regions colored, and no two adjacent regions have the same color.
Successor function: Assign a color to a region.
Cost function: Number of assignments.

b. Initial state: As described in the text.
Goal test: Monkey has bananas.
Successor function: Hop on crate; Hop off crate; Push crate from one spot to another; Walk from one spot to another; grab bananas (if standing on crate).
Cost function: Number of actions.

合法移动。 换句话说，在合法移动后，奇数N仍然是奇数，而偶数N仍然是偶数。因此，图3.4中的目标状态，在第一行中没有反转和空平方，具有N=1，并且只能从具有奇数N的起始状态到达，而不能从具有偶数N的起始状态到达。

证明：首先，水平滑动瓷砖既不会改变in版本的总数，也不会改变空方块的行号。因此，让我们考虑垂直滑动瓷砖。

例如，让我们假设瓷砖A直接位于空方块上方。
向下滑动它会改变空方块的行号的奇偶校验。 现在考虑反转的总数。 此移动仅影响图块a、b、C和D的相对位置。 如果B，C，D都没有引起相对于A的反转（即所有三个都大于A），那么在滑动之后，一个得到三个（奇数）额外的反转。 如果三者中的一个小于A，那么在移动之前B，C和D贡献了一个反转（相对于A），而在移动之后，它们将贡献两个反转1的变化，也是一个奇数。 另外两种情况显然导致相同的结果。 因此，总和N的变化总是偶数。 这正是我们要展示的。

所以在我们解决一个难题之前，我们应该计算开始和目标状态的N值，并确保它们具有相同的奇偶校验，否则没有解决方案是可能的。

3.5该提法每列放置一个女王，一个新的女王只放置在一个不被任何其他女王攻击的广场上。 为了简化问题，我们首先考虑n-rooks问题。 第一个rook可以放置在第1列（n个选择）中的任何正方形中，第二个在第2列的任何正方形中，除了与第1列（n-1个选择）中的rook相同的行，依此类推。 这给出了n！ 搜索空间的元素。

对于n个皇后，请注意，一个皇后在任何给定的列中最多攻击三个正方形，因此在第2列中至少有（n-3）个选择，在第2列中至少有（n-6）个选择，依此类推。 因而状态空间大小S≥n·(n-3)·(n-6)…。 因此，我们有

$$\begin{aligned} &\geq \quad \cdot \quad \cdot \quad \cdot \quad - \quad \cdot \quad - \quad \cdot \quad - \quad \cdot \quad - \quad \cdot \quad - \quad \cdot \quad - \quad \cdots \\ &\geq n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdot (n-4) \cdot (n-5) \cdot (n-6) \cdot (n-7) \cdot (n-8) \cdot \cdots \\ &= n! \end{aligned}$$

or $S \geq \sqrt[3]{n!}$.

3.6

a. 初始状态:没有区域着色.
目标测试：所有区域着色，并且没有两个相邻区域具有相同的颜色。
后继功能：为区域分配颜色。
成本函数：分配数量。

b. 初始状态：如文中所述。 目标测试：猴子有香蕉。
后继功能：跳上板条箱；跳下板条箱；将板条箱从一个地方推到另一个地方;从一个地方走到另一个地方;抓住香蕉（如果站在板条箱上）。
成本函数：操作数。

c. Initial state: considering all input records.
Goal test: considering a single record, and it gives "illegal input" message.
Successor function: run again on the first half of the records; run again on the second half of the records.
Cost function: Number of runs.
Note: This is a **contingency problem**; you need to see whether a run gives an error message or not to decide what to do next.

d. Initial state: jugs have values $[0, 0, 0]$.
Successor function: given values $[x, y, z]$, generate $[12, y, z]$, $[x, 8, z]$, $[x, y, 3]$ (by filling); $[0, y, z]$, $[x, 0, z]$, $[x, y, 0]$ (by emptying); or for any two jugs with current values x and y, pour y into x; this changes the jug with x to the minimum of $x + y$ and the capacity of the jug, and decrements the jug with y by by the amount gained by the first jug.
Cost function: Number of actions.

3.7

a. If we consider all (x, y) points, then there are an infinite number of states, and of paths.

b. (For this problem, we consider the start and goal points to be vertices.) The shortest distance between two points is a straight line, and if it is not possible to travel in a straight line because some obstacle is in the way, then the next shortest distance is a sequence of line segments, end-to-end, that deviate from the straight line by as little as possible. So the first segment of this sequence must go from the start point to a tangent point on an obstacle – any path that gave the obstacle a wider girth would be longer. Because the obstacles are polygonal, the tangent points must be at vertices of the obstacles, and hence the entire path must go from vertex to vertex. So now the state space is the set of vertices, of which there are 35 in Figure 3.31.

c. Code not shown.

d. Implementations and analysis not shown.

3.8

a. Any path, no matter how bad it appears, might lead to an arbitrarily large reward (negative cost). Therefore, one would need to exhaust all possible paths to be sure of finding the best one.

b. Suppose the greatest possible reward is c. Then if we also know the maximum depth of the state space (e.g. when the state space is a tree), then any path with d levels remaining can be improved by at most cd, so any paths worse than cd less than the best path can be pruned. For state spaces with loops, this guarantee doesn't help, because it is possible to go around a loop any number of times, picking up c reward each time.

c. The agent should plan to go around this loop forever (unless it can find another loop with even better reward).

d. The value of a scenic loop is lessened each time one revisits it; a novel scenic sight is a great reward, but seeing the same one for the tenth time in an hour is tedious, not

c. 初始状态：考虑所有输入记录。
目标测试：考虑单个记录，并且它给出"非法输入"消息。
后继函数：在前半段记录上再次运行；在后半段记录上再次运行。 成本函数：运行次数。

注意：这是一个偶然性问题；您需要查看运行是否给出错误消息，以决定下一步该做什么。 d. 初始状态：壶具有值[0,0,0]。

后继函数：给定值[x，y，z]，生成[12，y，z]，[x，8，z]，[x，y，3]（通过填充ing）;[0，y，z]，[x，0，z]，[x，y，0]（通过清空）;或者对于任何两个具有当前值x和y的水壶，将y倒入x;这将第一个水壶。 成本函数：操作数。

3.7

a. 如果我们考虑所有（x，y）点，那么有无限数量的状态和路径。 b. （对于这个问题，我们认为起点和目标点是顶点。)两点之间的最短距离是一条直线，如果因为有障碍物妨碍了它，所以不能直线行驶，那么下一个最短距离就是一串端到端的线段，它们尽可能少地偏离直线。 因此，这个序列的第一段必须从起点到障碍物上的切点-任何给障碍物一个更宽周长的路径都会更长。 由于障碍物是多边形的，切点必须位于障碍物的顶点，因此整个路径必须从顶点到顶点。 所以现在状态空间是顶点的集合，其中在图3.31中有35个。 c. 未示出的代码。 d. 实现和分析未示出。

3.8

a. 任何路径，无论它看起来多么糟糕，都可能导致任意大的奖励（负成本）。 因此，人们需要用尽所有可能的路径，以确保找到最好的路径。 b. 假设最大可能的奖励是c。 然后，如果我们也知道状态空间的最大深度（例如，当状态空间是树时），那么任何剩余d级的路径最多可以通过cd来改进，因此任何比cd差的路径小于最佳 对于具有循环的状态空间，这种保证没有帮助，因为可以绕循环任意次数，每次都拾取c奖励。 c. 代理应该计划永远围绕这个循环（除非它可以找到另一个具有更好奖励的循环）。 d. 每次重新审视一个风景圈的价值就会减少；一个新颖的风景是一个很大的回报，但是在一个小时内第十次看到同一个风景是很乏味的，而不是

rewarding. To accommodate this, we would have to expand the state space to include a memory—a state is now represented not just by the current location, but by a current location and a bag of already-visited locations. The reward for visiting a new location is now a (diminishing) function of the number of times it has been seen before.

e. Real domains with looping behavior include eating junk food and going to class.

3.9

a. Here is one possible representation: A state is a six-tuple of integers listing the number of missionaries, cannibals, and boats on the first side, and then the second side of the river. The goal is a state with 3 missionaries and 3 cannibals on the second side. The cost function is one per action, and the successors of a state are all the states that move 1 or 2 people and 1 boat from one side to another.

b. The search space is small, so any optimal algorithm works. For an example, see the file `"search/domains/cannibals.lisp"`. It suffices to eliminate moves that circle back to the state just visited. From all but the first and last states, there is only one other choice.

c. It is not obvious that almost all moves are either illegal or revert to the previous state. There is a feeling of a large branching factor, and no clear way to proceed.

3.10 A **state** is a situation that an agent can find itself in. We distinguish two types of states: world states (the actual concrete situations in the real world) and representational states (the abstract descriptions of the real world that are used by the agent in deliberating about what to do).

A **state space** is a graph whose nodes are the set of all states, and whose links are actions that transform one state into another.

A **search tree** is a tree (a graph with no undirected loops) in which the root node is the start state and the set of children for each node consists of the states reachable by taking any action.

A **search node** is a node in the search tree.

A **goal** is a state that the agent is trying to reach.

An **action** is something that the agent can choose to do.

A **successor function** described the agent's options: given a state, it returns a set of (action, state) pairs, where each state is the state reachable by taking the action.

The **branching factor** in a search tree is the number of actions available to the agent.

3.11 A world state is how reality is or could be. In one world state we're in Arad, in another we're in Bucharest. The world state also includes which street we're on, what's currently on the radio, and the price of tea in China. A state description is an agent's internal description of a world state. Examples are $In(Arad)$ and $In(Bucharest)$. These descriptions are necessarily approximate, recording only some aspect of the state.

We need to distinguish between world states and state descriptions because state description are lossy abstractions of the world state, because the agent could be mistaken about

奖励。 为了适应这一点，我们必须扩展状态空间以包含内存—状态现在不仅由当前位置表示，还由当前位置和一袋已经访问过的位置表示。 访问新位置的奖励现在是以前看到的次数的（递减）函数。 e. 具有循环行为的真实域名包括吃垃圾食品和去上课。

3.9

a. 这是一个可能的表示：一个状态是一个六元组的整数，列出传教士，食人族和船只的数量在第一面，然后是河流的第二面。 目标是在第二侧有3个传教士和3个食人族的状态。 成本函数是每个动作一个，一个州的继承者是所有将1或2人和1艘船从一侧移动到另一侧的州。 b. 搜索空间很小，因此任何最优算法都有效。 有关示例，请参阅文件"search/domains/cannibals.lisp"。 消除返回到刚刚访问的状态的移动就足够了。 除了第一个和最后一个州之外，只有另一个选择。 c. 并不明显，几乎所有的动作要么是非法的，要么恢复到以前的状态。 有一个大的分支因素的感觉，并没有明确的方式进行。

3.10状态是代理人可以发现自己的情况。 我们区分两种类型的国家：世界国家（现实世界中的实际具体情况）和代表性国家（代理人在考虑做什么时使用的现实世界的抽象描述）。

状态空间是一个图，其节点是所有状态的集合，其链接是将一个状态转换为另一个状态的操作。

搜索树是一棵树（没有无向循环的图），其中根节点是开始状态，每个节点的子节点集由通过执行任何操作可到达的状态组成。

搜索节点是搜索树中的节点。

目标是代理试图达到的状态。

操作是代理可以选择执行的操作。

一个后继函数描述了代理的选项：给定一个状态，它返回一组（动作，状态）对，其中每个状态是通过采取动作可到达的状态。

搜索树中的分支因子是代理可用的操作数。

3.11世界状态是现实是或可能是怎样的。 在一个世界国家，我们在阿拉德，在另一个国家，我们在布加勒斯特。 世界国家还包括我们在哪条街上，目前在电台上的节目，以及中国茶叶的价格。 状态描述是代理对世界状态的内部描述。 例子是在（阿拉德）和在（布加勒斯特）。 这些描述必然是近似的，仅记录状态的某些方面。

我们需要区分世界状态和状态描述，因为状态描述是世界状态的有损抽象，因为代理可能会被误解

how the world is, because the agent might want to imagine things that aren't true but it could make true, and because the agent cares about the world not its internal representation of it.

Search nodes are generated during search, representing a state the search process knows how to reach. They contain additional information aside from the state description, such as the sequence of actions used to reach this state. This distinction is useful because we may generate different search nodes which have the same state, and because search nodes contain more information than a state representation.

3.12 The state space is a tree of depth one, with all states successors of the initial state. There is no distinction between depth-first search and breadth-first search on such a tree. If the sequence length is unbounded the root node will have infinitely many successors, so only algorithms which test for goal nodes as we generate successors can work.

What happens next depends on how the composite actions are sorted. If there is no particular ordering, then a random but systematic search of potential solutions occurs. If they are sorted by dictionary order, then this implements depth-first search. If they are sorted by length first, then dictionary ordering, this implements breadth-first search.

A significant disadvantage of collapsing the search space like this is if we discover that a plan starting with the action "unplug your battery" can't be a solution, there is no easy way to ignore all other composite actions that start with this action. This is a problem in particular for informed search algorithms.

Discarding sequence structure is not a particularly practical approach to search.

3.13

The graph separation property states that "every path from the initial state to an unexplored state has to pass through a state in the frontier."

At the start of the search, the frontier holds the initial state; hence, trivially, every path from the initial state to an unexplored state includes a node in the frontier (the initial state itself).

Now, we assume that the property holds at the beginning of an arbitrary iteration of the GRAPH-SEARCH algorithm in Figure 3.7. We assume that the iteration completes, i.e., the frontier is not empty and the selected leaf node n is not a goal state. At the end of the iteration, n has been removed from the frontier and its successors (if not already explored or in the frontier) placed in the frontier. Consider any path from the initial state to an unexplored state; by the induction hypothesis such a path (at the beginning of the iteration) includes at least one frontier node; except when n is the only such node, the separation property automatically holds. Hence, we focus on paths passing through n (and no other frontier node). By definition, the next node n' along the path from n must be a successor of n that (by the preceding sentence) is already not in the frontier. Furthermore, n' cannot be in the explored set, since by assumption there is a path from n' to an unexplored node not passing through the frontier, which would violate the separation property as every explored node is connected to the initial state by explored nodes (see lemma below for proof this is always possible). Hence, n' is not in the explored set, hence it will be added to the frontier; then the path will include a frontier node and the separation property is restored.

The property is violated by algorithms that move nodes from the frontier into the ex-

世界是怎样的，因为代理人可能想想象那些不是真实的东西，但它可以使真实的东西，并且因为代理人关心世界而不是它的内部表示。

搜索节点在搜索过程中生成，表示搜索过程知道如何到达的状态。 除了状态描述之外，它们还包含其他信息，例如用于达到此状态的操作顺序。 这种区分很有用，因为我们可能会生成具有相同状态的不同搜索节点，并且因为搜索节点包含的信息比状态表示更多。

3.12 状态空间是一个深度树，具有初始状态的所有状态继承者。

在这样的树上没有深度优先搜索和广度优先搜索的区别。 如果序列长度是无界的，则根节点将具有无限多的继承者，因此只有在我们生成继承者时测试目标节点的算法才能工作。

接下来会发生什么取决于复合操作的排序方式。 如果没有特定的排序，则会发生对潜在解决方案的随机但系统的搜索。 如果它们按字典顺序排序，那么这实现了深度优先搜索。 如果它们先按长度排序，然后按字典排序，则实现广度优先搜索。

像这样折叠搜索空间的一个显着缺点是，如果我们发现以"拔掉电池"动作开始的计划不能成为解决方案，那么没有简单的方法可以忽略以此动作开始的所 这是一个问题，特别是对知情的搜索算法。

丢弃序列结构并不是一种特别实用的搜索方法。

3.13

图分离属性指出"从初始状态到未绘制状态的每条路径都必须经过边界中的一个状态。"

在搜索开始时，边界保存初始状态;因此，平凡地，从初始状态到未探索状态的每条路径都包括边界中的一个节点（初始状态本身）。

现在，我们假设该属性在图3.7中的GZ-S EARCH算法的任意迭代开始时成立。 我们假设迭代完成，即边界不为空，选择的叶子节点n不是目标状态。 在迭代结束时，n已从边界中移除，其后继者（如果尚未探索或在边界中）放置在边界中。 考虑从初始状态到未探索状态的任何路径；通过归纳假设这样的路径（在迭代开始时）包括至少一个边界节点；除了当n是唯一这样的节点时，分离属性自 因此，我们专注于通过n（而不是其他前沿节点）的路径。 根据定义，沿着从n开始的路径的下一个节点n'必须是n的后继节点，该后继节点（通过前一句）已经不在边界中。 此外，n'不能在探索集合中，因为假设有一条从n'到未探索节点的路径没有通过边界，这将违反分离属性，因为每个探索节点都通过探索节点连接到初始状态（请参阅下面的引理以证明这始终是可能的）。 因此，n'不在探索集中，因此它将被添加到边界;然后路径将包括一个边界节点，并恢复分离属性。

将节点从边界移动到ex的算法违反了该属性

plored set before all of their successors have been generated, as well as by those that fail to add some of the successors to the frontier. Note that it is not necessary to generate *all* successors of a node at once before expanding another node, as long as partially expanded nodes remain in the frontier.

Lemma: Every explored node is connected to the initial state by a path of explored nodes.

Proof: This is true initially, since the initial state is connected to itself. Since we never remove nodes from the explored region, we only need to check new nodes we add to the explored list on an expansion. Let n be such a new explored node. This is previously on the frontier, so it is a neighbor of a node n' previously explored (i.e., its parent). n' is, by hypothesis is connected to the initial state by a path of explored nodes. This path with n appended is a path of explored nodes connecting n' to the initial state.

3.14

a. *False*: a lucky DFS might expand exactly d nodes to reach the goal. A* largely dominates any graph-search algorithm that is *guaranteed to find optimal solutions*.

b. *True*: $h(n) = 0$ is always an admissible heuristic, since costs are nonnegative.

c. *True*: A* search is often used in robotics; the space can be discretized or skeletonized.

d. *True*: depth of the solution matters for breadth-first search, not cost.

e. *False*: a rook can move across the board in move one, although the Manhattan distance from start to finish is 8.

3.15

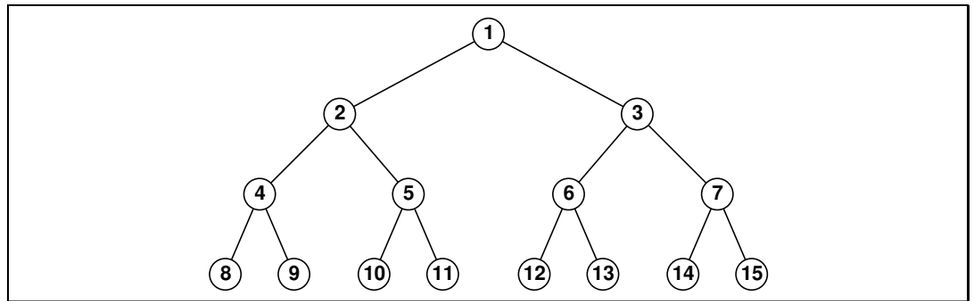


Figure S3.1 The state space for the problem defined in Ex. 3.15.

a. See Figure S3.1.

b. Breadth-first: 1 2 3 4 5 6 7 8 9 10 11
Depth-limited: 1 2 4 8 9 5 10 11
Iterative deepening: 1; 1 2 3; 1 2 4 5 3 6 7; 1 2 4 8 9 5 10 11

c. Bidirectional search is very useful, because the only successor of n in the reverse direction is $\lfloor (n/2) \rfloor$. This helps focus the search. The branching factor is 2 in the forward direction; 1 in the reverse direction.

在他们所有的继任者都产生之前，以及那些没有将一些继任者添加到边疆的人。 请注意，在扩展另一个节点之前，不必一次生成一个节点的所有successors，只要部分扩展的节点保留在边界中。

引理：每个探索的节点都通过探索节点的路径连接到初始状态。

证明：最初是如此，因为初始状态与自身相连。 由于我们从不从探索区域中删除节点，我们只需要检查我们在扩展中添加到探索列表中的新节点。 设n是这样一个新的探索节点。 这是以前在边界上，因此它是先前探索的节点n'的邻居（即其父节点）。 n'是，通过假设被探索节点的路径连接到初始状态。 这个附加了n的路径是连接n'到初始状态的探索节点的路径。

3.14

a. 错误：幸运的DFS可能会完全扩展d节点以达到目标。 A 在很大程度上主导了任何保证找到最优解的图搜索算法。 b. True：h（n）=0总是一个可接受的启发式，因为成本是非负的。 c. True：a*搜索通常用于机器人;空间可以离散化或骨架化。 d. True：解决方案的深度对于广度优先搜索至关重要，而不是成本。 e. 错误：一个车可以在一个移动板上移动，尽管曼哈顿从开始到结束的距离是8。

3.15

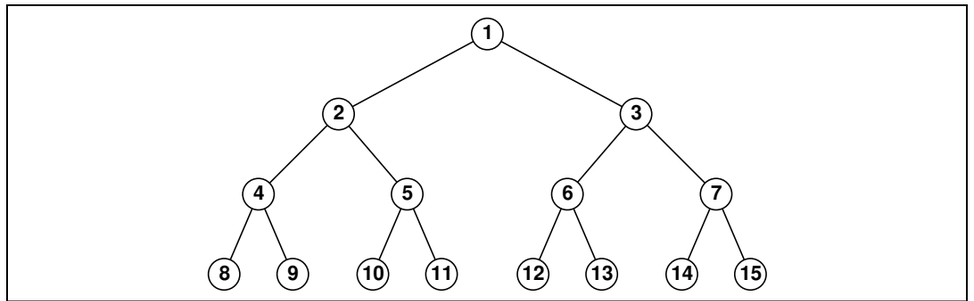


Figure S3.1 Ex中定义的问题的状态空间。 3.15.

a. 见图S3.1。 b. 广度优先: 1 2 3 4 5
6 7 8 9 10 11 深度限制：1248951
011

迭代深化: 1; 1 2 3; 1 2 4 5 3 6 7; 1 2 4 8 9 5 10 11 c． 双向搜索非常有用，因为n在反向方向上的唯一后继者是 （n/2） 。 这有助于集中搜索。 分支因子在正向方向上为2;在反向方向上为1。

d. Yes; start at the goal, and apply the single reverse successor action until you reach 1.

e. The solution can be read off the binary numeral for the goal number. Write the goal number in binary. Since we can only reach positive integers, this binary expansion beings with a 1. From most- to least- significant bit, skipping the initial 1, go Left to the node $2n$ if this bit is 0 and go Right to node $2n + 1$ if it is 1. For example, suppose the goal is 11, which is 1011 in binary. The solution is therefore Left, Right, Right.

3.16

a. **Initial state**: one arbitrarily selected piece (say a straight piece).
Successor function: for any open peg, add any piece type from remaining types. (You can add to open holes as well, but that isn't necessary as all complete tracks can be made by adding to pegs.) For a curved piece, add *in either orientation*; for a fork, add *in either orientation* and (if there are two holes) connecting *at either hole*. It's a good idea to disallow any overlapping configuration, as this terminates hopeless configurations early. (Note: there is no need to consider open holes, because in any solution these will be filled by pieces added to open pegs.)
Goal test: all pieces used in a single connected track, no open pegs or holes, no overlapping tracks.
Step cost: one per piece (actually, doesn't really matter).

b. All solutions are at the same depth, so depth-first search would be appropriate. (One could also use depth-limited search with limit $n - 1$, but strictly speaking it's not necessary to do the work of checking the limit because states at depth $n - 1$ have no successors.) The space is very large, so uniform-cost and breadth-first would fail, and iterative deepening simply does unnecessary extra work. There are many repeated states, so it might be good to use a closed list.

c. A solution has no open pegs or holes, so every peg is in a hole, so there must be equal numbers of pegs and holes. Removing a fork violates this property. There are two other "proofs" that are acceptable: 1) a similar argument to the effect that there must be an even number of "ends"; 2) each fork creates two tracks, and only a fork can rejoin those tracks into one, so if a fork is missing it won't work. The argument using pegs and holes is actually more general, because it also applies to the case of a three-way fork that has one hole and three pegs or one peg and three holes. The "ends" argument fails here, as does the fork/rejoin argument (which is a bit handwavy anyway).

d. The maximum possible number of open pegs is 3 (starts at 1, adding a two-peg fork increases it by one). Pretending each piece is unique, any piece can be added to a peg, giving at most $12 + (2 \cdot 16) + (2 \cdot 2) + (2 \cdot 2 \cdot 2) = 56$ choices per peg. The total depth is 32 (there are 32 pieces), so an upper bound is $168^{32}/(12! \cdot 16! \cdot 2! \cdot 2!)$ where the factorials deal with permutations of identical pieces. One could do a more refined analysis to handle the fact that the branching factor shrinks as we go down the tree, but it is not pretty.

3.17 **a.** The algorithm expands nodes in order of increasing path cost; therefore the first goal it encounters will be the goal with the cheapest cost.

d. 是;从目标开始，并应用单个反向后继动作，直到达到1。 e. 解可以读出目标数的二进制数字. 用二进制写入目标数。 由于我们只能达到正整数，因此这种二进制扩展具有1。 从mostto leastsignificant位开始，跳过初始1，如果该位为0，则向左转到节点2n，如果为1，则向右转到节点2n+1。 例如，假设目标是11，即二进制中的1011。 因此，解决方案是左，右，右。

3.16

a. 初始状态:一个任意选择的片断(说一个直的片断).
后继功能：对于任何打开的挂钩，从剩余类型中添加任何棋子类型。 （你也可以添加到开孔，但这是没有必要的，因为所有完整的轨道可以通过添加到钉子。）对于弯曲的部件，在任一方向添加;对于叉子，在任一方向添加并且（如果有两个孔）在任一孔处连接。 不允许任何重叠配置是一个好主意，因为这会提前终止无望的配置。 （注意：没有必要考虑开放孔，因为在任何解决方案中，这些将被添加到开放钉中的碎片填充。）目标测试：在一个单一的连接轨道中使用的所有部件，没有开放的钉子或孔，没有过度研磨轨道。 步骤成本：每件一件（实际上，并不重要）。 b. 所有解决方案都在相同的深度，因此深度优先搜索将是合适的。（人们也可以使用极限n-1的深度限制搜索，但严格来说，检查极限的工作并不是必要的，因为深度n-1的状态没有成功。）空间非常大，因此uniform-cost和广度优先会失败，迭代深化只是做了不必要的额外工作。 有许多重复的状态，所以使用封闭列表可能会很好。 c. 一个解决方案没有开放的钉子或孔，所以每个钉子都在一个孔中，所以必须有相等数量的钉子和孔。 删除分叉会违反此属性。 还有另外两个"证明"是可以接受的：1）类似的论点，大意是必须有偶数个"结束";2）每个叉子创建两个轨道，只有叉子可以将这些轨道重新合并为一个，所以如果 使用钉和孔的论点实际上更一般，因为它也适用于具有一个孔和三个钉或一个钉和三个孔的三向叉的情况。 "Ends"参数在这里失败，fork/rejoin参数也是如此（无论如何这有点handwavy）。 d. 最大可能的开放钉数是3（从1开始，添加一个双钉叉增加一个）。 假装每件作品都是独一无二的，任何一件作品都可以添加到一个钉子上，最多可以给出12 + (2 · 16) + (2 · 2) + (2 · 2 · 2) = 56 每个peg的选择。 总深度是32（有32件），所以上界是168 32/（12！ · 16! · 2! · 2! 因式分解处理相同部分的排列。 我们可以做一个更精细的分析来处理分支因子在我们下树时缩小的事实，但它并不漂亮。

3.17a.该算法以增加路径成本的顺序扩展节点;因此它遇到的第一个目标将是成本最低的目标。

b. It will be the same as iterative deepening, d iterations, in which $O(b^d)$ nodes are generated.

c. d/ϵ

d. Implementation not shown.

3.18 Consider a domain in which every state has a single successor, and there is a single goal at depth n. Then depth-first search will find the goal in n steps, whereas iterative deepening search will take $1 + 2 + 3 + \cdots + n = O(n^2)$ steps.

3.19 As an ordinary person (or agent) browsing the web, we can only generate the successors of a page by visiting it. We can then do breadth-first search, or perhaps best-search search where the heuristic is some function of the number of words in common between the start and goal pages; this may help keep the links on target. Search engines keep the complete graph of the web, and may provide the user access to all (or at least some) of the pages that link to a page; this would allow us to do bidirectional search.

3.20 Code not shown, but a good start is in the code repository. Clearly, graph search must be used—this is a classic grid world with many alternate paths to each state. Students will quickly find that computing the optimal solution sequence is prohibitively expensive for moderately large worlds, because the state space for an $n \times n$ world has $n^2 \cdot 2^n$ states. The completion time of the random agent grows less than exponentially in n, so for any reasonable exchange rate between search cost ad path cost the random agent will eventually win.

3.21

a. When all step costs are equal, $g(n) \propto depth(n)$, so uniform-cost search reproduces breadth-first search.

b. Breadth-first search is best-first search with $f(n) = depth(n)$; depth-first search is best-first search with $f(n) = -depth(n)$; uniform-cost search is best-first search with $f(n) = g(n)$.

c. Uniform-cost search is A* search with $h(n) = 0$.

3.22 The student should find that on the 8-puzzle, RBFS expands more nodes (because it does not detect repeated states) but has lower cost per node because it does not need to maintain a queue. The number of RBFS node re-expansions is not too high because the presence of many tied values means that the best path changes seldom. When the heuristic is slightly perturbed, this advantage disappears and RBFS's performance is much worse.

For TSP, the state space is a tree, so repeated states are not an issue. On the other hand, the heuristic is real-valued and there are essentially no tied values, so RBFS incurs a heavy penalty for frequent re-expansions.

3.23 The sequence of queues is as follows:

L[0+244=244]

M[70+241=311], T[111+329=440]

L[140+244=384], D[145+242=387], T[111+329=440]

D[145+242=387], T[111+329=440], M[210+241=451], T[251+329=580]

B.它将与迭代深化，d迭代相同，其中生成O（b d）节点。 C.d/△d.未示出的实现。

3.18考虑一个域，其中每个状态都有一个单一的后继者，并且在深度n处有一个单一的目标。 然后深度优先搜索将在n个步骤中找到目标，而迭代深化搜索将采取1 + 2 + 3 + · · · + n=O(n2)个步骤。

3.19作为浏览网页的普通人（或代理），我们只能通过访问它来生成一个页面的成功。 然后，我们可以进行广度优先搜索，或者可能是最佳搜索搜索，其中启发式是开始和目标页面之间共有单词数量的一些函数;这可能有助于保持链接在目 搜索引擎保留完整的网络图表，并可能为用户提供访问链接到页面的所有（或至少部分）页面的权限;这将允许我们进行双向搜索。

3.20代码未显示，但一个良好的开端是在代码存储库中。 显然，必须使用图形搜索—这是一个经典的网格世界，每个州都有许多备用路径。 学生很快就会发现，计算最优解序列对于中等大的世界来说是昂贵的，因为n×n世界的状态空间有n个2·2n状态。 随机代理的完成时间以n为单位小于指数增长，因此对于搜索成本和路径成本之间的任何合理汇率，随机代理最终都会获胜。

3.21

a. 当所有步长成本相等时，g(n)∈depth(n)，因此uniform-cost搜索再现广度优先搜索。 b. 广度优先搜索是f(n)=深度(n)的最佳优先搜索；深度优先搜索是f(n)=−深度(n)的最佳优先搜索；均匀成本搜索是f(n)=g(n)的最佳优先搜索。 c. Uniform-cost搜索是h(n)=0的 搜索。

3.22学生应该发现，在8-puzzle上，RBFS扩展了更多的节点（因为它没有检测到重复的状态），但每个节点的成本更低，因为它不需要维护队列。 RBFS节点重新扩展的数量不会太高，因为存在许多绑定值意味着最佳路径很少发生变化。 当启发式稍微扰动时，这种优势消失，RBFS的性能要差得多。

对于TSP，状态空间是一棵树，因此重复的状态不是问题。 另一方面，启发式是实值的，基本上没有绑定的值，因此RBFS会对频繁的重新扩展造成严重的惩罚。

3.23 队列的顺序如下:

L[0+244=244]

M[70+241=311], T[111+329=440]

L[140+244=384], D[145+242=387], T[111+329=440]

D[145+242=387], T[111+329=440], M[210+241=451], T[251+329=580]

C[265+160=425], T[111+329=440], M[210+241=451], M[220+241=461], T[251+329=580]
T[111+329=440], M[210+241=451], M[220+241=461], P[403+100=503], T[251+329=580], R[411+193=604], D[385+242=627]
M[210+241=451], M[220+241=461], L[222+244=466], P[403+100=503], T[251+329=580], A[229+366=595], R[411+193=604], D[385+242=627]
M[220+241=461], L[222+244=466], P[403+100=503], L[280+244=524], D[285+242=527], T[251+329=580], A[229+366=595], R[411+193=604], D[385+242=627]
L[222+244=466], P[403+100=503], L[280+244=524], D[285+242=527], L[290+244=534], D[295+242=537], T[251+329=580], A[229+366=595], R[411+193=604], D[385+242=627]
P[403+100=503], L[280+244=524], D[285+242=527], M[292+241=533], L[290+244=534], D[295+242=537], T[251+329=580], A[229+366=595], R[411+193=604], D[385+242=627], T[333+329=662]
B[504+0=504], L[280+244=524], D[285+242=527], M[292+241=533], L[290+244=534], D[295+242=537], T[251+329=580
A[229+366=595], R[411+193=604], D[385+242=627], T[333+329=662], R[500+193=693], C[541+160=701]

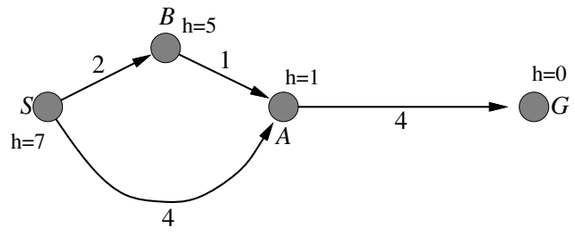


Figure S3.2 A graph with an inconsistent heuristic on which GRAPH-SEARCH fails to return the optimal solution. The successors of S are A with $f=5$ and B with $f=7$. A is expanded first, so the path via B will be discarded because A will already be in the closed list.

3.24 See Figure S3.2.

3.25 It is complete whenever $0 \leq w < 2$. $w = 0$ gives $f(n) = 2g(n)$. This behaves exactly like uniform-cost search—the factor of two makes no difference in the *ordering* of the nodes. $w = 1$ gives A* search. $w = 2$ gives $f(n) = 2h(n)$, i.e., greedy best-first search. We also have

$$f(n) = (2-w)[g(n) + \frac{w}{2-w}h(n)]$$

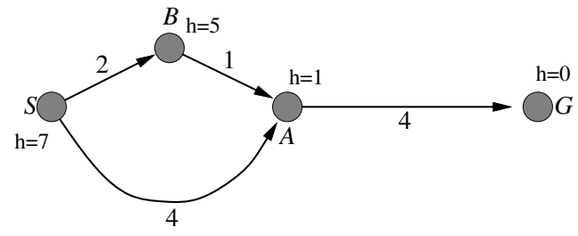
which behaves exactly like A* search with a heuristic $\frac{w}{2-w}h(n)$. For $w \leq 1$, this is always less than $h(n)$ and hence admissible, provided $h(n)$ is itself admissible.

3.26

a. The branching factor is 4 (number of neighbors of each location).

b. The states at depth k form a square rotated at 45 degrees to the grid. Obviously there are a linear number of states along the boundary of the square, so the answer is $4k$.

C[265+160=425], T[111+329=440], M[210+241=451], M[220+241=461], T[251+329=580]
T[111+329=440], M[210+241=451], M[220+241=461], P[403+100=503], T[251+329=580], R[411+193=604], D[385+242=627]
M[210+241=451], M[220+241=461], L[222+244=466], P[403+100=503], T[251+329=580], A[229+366=595], R[411+193=604], D[385+242=627]
M[220+241=461], L[222+244=466], P[403+100=503], L[280+244=524], D[285+242=527], T[251+329=580], A[229+366=595], R[411+193=604], D[385+242=627]
L[222+244=466], P[403+100=503], L[280+244=524], D[285+242=527], L[290+244=534], D[295+242=537], T[251+329=580], A[229+366=595], R[411+193=604], D[385+242=627]
P[403+100=503], L[280+244=524], D[285+242=527], M[292+241=533], L[290+244=534], D[295+242=537], T[251+329=580], A[229+366=595], R[411+193=604], D[385+242=627], T[333+329=662]
B[504+0=504], L[280+244=524], D[285+242=527], M[292+241=533], L[290+244=534], D[295+242=537], T[251+329=580
A[229+366=595], R[411+193=604], D[385+242=627], T[333+329=662], R[500+193=693], C[541+160=701]



图S3.2一个具有不一致启发式的图，在该图上G R-S EARCH未能返回最优解。 S的继任者是f=5的A和f=7的B。 A首先被展开，因此通过B的路径将被丢弃，因为A将已经在封闭列表中。

3.24 见图S3.2。

3.25每当0≤w<2时即完成。 w=0给出f（n）=2g（n）。 这与均匀成本搜索的行为完全相同-二的因子在节点的顺序上没有区别。 w=1给出了一个 搜索。 w=2给出f(n)=2h(n)，即贪婪的最佳优先搜索。 我们也有

$$f(n) = (2-w)[g(n) + \frac{w}{2\quad w}h(n)]$$

w2-w h(n)。 对于w≤1,这总是小于h(n),因此可以接受,只要h(n)本身是可以接受的.

3.26

a. 分支因子为4（每个位置的邻居数）。 b. 深度k处的状态形成以45度旋转到网格的正方形。 显然沿着正方形的边界存在线性数量的状态，所以答案是4k。

c. Without repeated state checking, BFS expends exponentially many nodes: counting precisely, we get $((4^{x+y+1}-1)/3)-1$.

d. There are quadratically many states within the square for depth $x+y$, so the answer is $2(x+y)(x+y+1)-1$.

e. True; this is the Manhattan distance metric.

f. False; all nodes in the rectangle defined by $(0,0)$ and (x,y) are candidates for the optimal path, and there are quadratically many of them, all of which may be expended in the worst case.

g. True; removing links may induce detours, which require more steps, so h is an underestimate.

h. False; nonlocal links can reduce the actual path length below the Manhattan distance.

3.27

a. n^{2n}. There are n vehicles in n^2 locations, so roughly (ignoring the one-per-square constraint) $(n^2)^n = n^{2n}$ states.

b. 5^n.

c. Manhattan distance, i.e., $|(n-i+1)-x_i|+|n-y_i|$. This is exact for a lone vehicle.

d. Only (iii) $\min\{h_1,\ldots,h_n\}$. The explanation is nontrivial as it requires two observations. First, let the *work* W in a given solution be the total *distance* moved by all vehicles over their joint trajectories; that is, for each vehicle, add the lengths of all the steps taken. We have $W \geq \sum_i h_i \geq\geq n \cdot min\{h_1,...,h_n\}$. Second, the total work we can get done per step is $\leq n$. (Note that for every car that jumps 2, another car has to stay put (move 0), so the total work per step is bounded by n.) Hence, completing all the work requires at least $n \cdot min\{h_1,...,h_n\}/n = min\{h_1,...,h_n\}$ steps.

3.28 The heuristic $h = h_1 + h_2$ (adding misplaced tiles and Manhattan distance) sometimes overestimates. Now, suppose $h(n) \leq h^*(n)+c$ (as given) and let G_2 be a goal that is suboptimal by more than c, i.e., $g(G_2) > C^* + c$. Now consider any node n on a path to an optimal goal. We have

$$\begin{aligned} f(n) &= g(n)+h(n) \\ &\leq g(n)+h^*(n)+c \\ &\leq C^*+c \\ &\leq g(G_2) \end{aligned}$$

so G_2 will never be expanded before an optimal goal is expanded.

3.29 A heuristic is consistent iff, for every node n and every successor n' of n generated by any action a,

$$h(n) \leq c(n,a,n') + h(n')$$

One simple proof is by induction on the number k of nodes on the shortest path to any goal from n. For $k=1$, let n' be the goal node; then $h(n) \leq c(n,a,n')$. For the inductive

c. 在没有重复状态检查的情况下，BFS以指数方式扩展了许多节点：精确计数，我们得到（（4x+y+1 −1）/ 3）−1 . d. 对于深度x+y，正方形内有四边形许多状态，所以答案是2（x+y）（x+y+1）−1。 e. True;这是曼哈顿距离度量。 f. False;由（0，0）和（x，y）定义的矩形中的所有节点都是最优路径的候选者，并且其中有四元组，所有这些节点都可能在最坏的情况下被消耗。 g. 诚然;删除链接可能会导致弯路，这需要更多的步骤，所以h是低估的。 h. 假；非局部链路可以将实际路径长度减小到曼哈顿距离以下。

3.27

a. n2n． 在n2个位置有n辆车，所以粗略地（忽略每平方一个约束）（n2）n=n2 n状态。 b. 5n． c. 曼哈顿距离，即|(n−i+1)−x i|+|n−y i|。 这是一个单独的车辆的确切。 d. 仅(iii)min{h1,。 ..，h n}。 这个解释并不重要，因为它需要两个观察。首先，设给定解中的功W是所有车辆在其关节轨迹上移动的总距离;也就是说，对于每个车辆，添加所采取的所有步骤的长度。 我们有w≥

i h i≥≥n*min{h1,。..，h n}。 其次，我们每步所能完成的总工作量≤n。 （请注意，对于每辆跳过2的汽车，另一辆汽车必须保持原地（移动0），因此每步的总工作量以n为界。）因此，完成所有工作至少需要n*min{h1，。..,h n}/n=min{h1,。..，h n}的步骤。

3.28启发式h=h1+h2（添加错位的瓷砖和曼哈顿距离）有时会高估。 现在，假设h（n）≤h∈（n）+c（如给定的），并且让G2是超过c的次优目标，即g（G2）>C∈+c。 现在考虑一条通往最优目标的路径上的任何节点n。 我们有

$$\begin{aligned} f(n) &= g(n)+h(n) \\ &\leq g(n)+h^*(n)+c \\ &\leq C^*+c \\ &\leq g(G_2) \end{aligned}$$

所以在一个最优目标展开之前，G2永远不会展开。

3.29启发式是一致的iff，对于任何动作a产生的n的每个节点n和每个后继n'，

$$h(n) \leq c(n,a,n') + h(n')$$

一个简单的证明是通过对从n到任何目标的最短路径上的节点数k进行归纳。 对于k=1，设n'为目标节点；则h(n)≤c(n,a,n')。 对于感应

case, assume n' is on the shortest path k steps from the goal and that $h(n')$ is admissible by hypothesis; then

$$h(n) \leq c(n, a, n') + h(n') \leq c(n, a, n') + h^*(n') = h^*(n)$$

so $h(n)$ at $k+1$ steps from the goal is also admissible.

3.30 This exercise reiterates a small portion of the classic work of Held and Karp (1970).

a. The TSP problem is to find a minimal (total length) path through the cities that forms a closed loop. MST is a relaxed version of that because it asks for a minimal (total length) graph that need not be a closed loop—it can be any fully-connected graph. As a heuristic, MST is admissible—it is always shorter than or equal to a closed loop.

b. The straight-line distance back to the start city is a rather weak heuristic—it vastly underestimates when there are many cities. In the later stage of a search when there are only a few cities left it is not so bad. To say that MST dominates straight-line distance is to say that MST always gives a higher value. This is obviously true because a MST that includes the goal node and the current node must either be the straight line between them, or it must include two or more lines that add up to more. (This all assumes the triangle inequality.)

c. See `"search/domains/tsp.lisp"` for a start at this. The file includes a heuristic based on connecting each unvisited city to its nearest neighbor, a close relative to the MST approach.

d. See (Cormen *et al.*, 1990, p.505) for an algorithm that runs in $O(E \log E)$ time, where E is the number of edges. The code repository currently contains a somewhat less efficient algorithm.

3.31 The misplaced-tiles heuristic is exact for the problem where a tile can move from square A to square B. As this is a relaxation of the condition that a tile can move from square A to square B if B is blank, Gaschnig's heuristic cannot be less than the misplaced-tiles heuristic. As it is also admissible (being exact for a relaxation of the original problem), Gaschnig's heuristic is therefore more accurate.

If we permute two adjacent tiles in the goal state, we have a state where misplaced-tiles and Manhattan both return 2, but Gaschnig's heuristic returns 3.

To compute Gaschnig's heuristic, repeat the following until the goal state is reached: let B be the current location of the blank; if B is occupied by tile X (not the blank) in the goal state, move X to B; otherwise, move any misplaced tile to B. Students could be asked to prove that this is the optimal solution to the relaxed problem.

3.32 Students should provide results in the form of graphs and/or tables showing both run-time and number of nodes generated. (Different heuristics have different computation costs.) Runtimes may be very small for 8-puzzles, so you may want to assign the 15-puzzle or 24-puzzle instead. The use of pattern databases is also worth exploring experimentally.

情况下，假设n'在距离目标k步的最短路径上，并且h（n'）通过假设可以接受;然后

$$h(n) \leq c(n, a, n') + h(n') \leq c(n, a, n') + h^*(n') = h^*(n)$$

所以距离目标k+1步的h(n)也是可以接受的.

3.30 这个练习重申了Hold和Karp（1970）经典作品的一小部分。

a. TSP问题是找到通过形成闭环的城市的最小（总长度）路径。 MST是一个宽松的版本，因为它要求一个不需要是闭环的最小（总长度）图—它可以是任何完全连接的图。 作为启发式，MST是可接受的-它总是短于或等于闭环。 b. 回到起始城市的直线距离是一个相当弱的启发式-当有许多城市时，它大大低估了。 在搜索的后期，当只剩下几个城市时，它并不是那么糟糕。 要说MST主导直线距离，就是说MST总是给出更高的值。 这显然是正确的，因为包括目标节点和当前节点的MST必须是它们之间的直线，或者它必须包括两条或两条以上加起来更多的线。 （这都假设三角形不等式。）c． 请参阅"搜索/域/tsp。lisp"从这开始。 该文件包括一个启发式，该启发式基于将每个未被访问的城市与其最近的邻居连接起来，这是一个与MST方法的近亲。 d. 见(Cormen et al.，1990，第505页）用于在O（e log E）时间内运行的算法，其中E是边的数量。 代码存储库目前包含一个效率较低的算法。

3.31错位瓷砖启发式是准确的问题，其中一个瓷砖可以从正方形a移动到正方形B.因为这是一个条件的松弛，瓷砖可以从正方形a移动到正方形b，如果B是空白，Gaschnig的启发式不能小于错位瓷砖启发式。 由于它也是可以接受的（对于原始问题的放松是精确的），因此Gaschnig的启发式更准确。

如果我们在目标状态中排列两个相邻的瓦片，我们有一个状态，其中错位-瓦片和曼哈顿都返回2，但Gaschnig的启发式返回3。

要计算Gaschnig的启发式，重复以下操作，直到达到目标状态：让B是空白的当前位置;如果b在目标状态下被tile X（不是空白）占据，则将X移动到B;否则，将任何错位的tile移动到B。

3.32学生应以图表和/或表格的形式提供结果，显示运行时间和生成的节点数。 （不同的启发式有不同的计算成本。）对于8个谜题来说，运行时可能非常小，所以你可能想分配15个谜题或24个谜题。 模式数据库的使用也值得实验探索。