**Capítulo 1: Problemas a serem resolvidos**

Neste capítulo, discutimos problemas a serem resolvidos, como encontrados frequentemente por engenheiros, cientistas da computação, etc. Argumentamos que os problemas e solucionadores de problemas podem, e devem, ser distinguidos, e observamos que o campo da computação evolutiva está principalmente preocupado com os solucionadores de problemas . No entanto, para caracterizar qualquer solucionador de problemas, é útil identificar o tipo de problemas aos quais ele pode ser aplicado. Portanto, começamos este livro discutindo várias classes de problemas e, na verdade, até mesmo maneiras diferentes de classificá-los.

Na discussão informal a seguir, apresentamos os conceitos e a terminologia necessários para nossos propósitos por meio de exemplos, apenas usando um tratamento formal quando for necessário para uma boa compreensão dos detalhes. **Para evitar polêmica, não nos preocupamos com problemas sociais ou políticos. Os problemas que temos em mente são os típicos aos quais a inteligência artificial está associada:** mais parecidos com quebra-cabeças (por exemplo, o famoso quebra-cabeça da zebra), problemas numéricos (por exemplo, qual é o caminho mais curto de uma cidade do norte para uma cidade do sul), ou descoberta de padrões (por exemplo, o que um novo cliente comprará em nossa livraria online, dado seu sexo, idade, endereço, etc.).

**1.1 Problemas de otimização, modelagem e simulação**

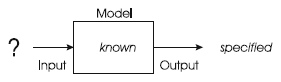
**A classificação de problemas usada nesta seção é baseada em um modelo de caixa preta de sistemas de computador.** Informalmente, podemos pensar em qualquer sistema baseado em computador da seguinte maneira. O sistema inicialmente fica parado, aguardando alguma entrada de uma pessoa, um sensor ou outro computador. Quando a entrada é fornecida, o sistema processa essa entrada por meio de algum modelo computacional, cujos detalhes não são especificados em geral (daí o nome caixa preta). O objetivo deste modelo é representar alguns aspectos do mundo relevantes para o aplicativo específico. Por exemplo, o modelo pode ser uma fórmula que calcula o comprimento total do trajeto a partir de uma lista de locais consecutivos, uma ferramenta estatística que estima a probabilidade de chuva com base em alguns dados meteorológicos, um mapeamento de dados em tempo real sobre a velocidade de um carro até o nível de aceleração necessário aproximar-se de alguma velocidade alvo pré-especificada ou de uma série complexa de regras que transformam uma série de pressionamentos de tecla em uma versão na tela da página que você está lendo agora. Depois de processar a entrada, o sistema fornece algumas saídas - que podem ser mensagens na tela, valores gravados em um arquivo ou comandos enviados a um atuador, como um motor. Dependendo do aplicativo, pode haver uma ou mais entradas de diferentes tipos e o modelo computacional pode ser simples ou muito complexo. É importante ressaltar que conhecer o modelo significa que podemos calcular a saída para qualquer entrada. Para fornecer alguns exemplos concretos:

* Ao projetar asas de aeronaves, as entradas podem representar uma descrição de um formato de asa proposto. O modelo pode conter equações de dinâmica de fluidos complexa para estimar os coeficientes de arrasto e sustentação de qualquer formato de asa. Essas estimativas constituem a saída do sistema.
* Um sistema de controle de voz para casas inteligentes toma como entrada o sinal elétrico produzido quando um usuário fala em um microfone. As saídas adequadas podem ser comandos a serem enviados ao sistema de aquecimento, ao aparelho de TV ou às luzes. Portanto, neste caso, o modelo consiste em um mapeamento de certos padrões em formas de onda elétricas provenientes de uma entrada de áudio para as saídas que normalmente seriam criadas pressionando-se as teclas de um teclado.
* Para um reprodutor de música portátil, as entradas podem ser uma série de gestos e botões pressionados - talvez escolhendo uma lista de reprodução que o usuário criou. Aqui, a resposta do modelo pode envolver selecionar uma série de arquivos mp3 de um banco de dados e processá-los de alguma forma para fornecer a saída desejada para essa sequência de gestos. Nesse caso, a saída seria um sinal elétrico flutuante alimentado por um par de fones de ouvido que, por sua vez, produziam o som das músicas escolhidas.

**Em essência, a visualização da caixa preta dos sistemas distingue três componentes: a entrada, o modelo e a saída.** A seguir, descreveremos três tipos de problemas, dependendo de qual deles é desconhecido.

**1.1.1 Otimização**

**Em um problema de otimização, o modelo é conhecido, junto com a saída desejada (ou uma descrição da saída desejada), e a tarefa é encontrar a (s) entrada (s) que levam a essa saída (Fig. 1.1).**



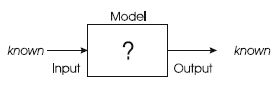
Por exemplo, consideremos o **problema do caixeiro viajante**. Esse problema aparentemente bastante abstrato é popular na ciência da computação, pois há muitas aplicações práticas que podem ser reduzidas a isso, como organização de rotas de entrega, layout de fábrica, programações de produção e cronograma. Na versão abstrata, recebemos um conjunto de cidades e temos que encontrar o passeio mais curto que visita cada cidade exatamente uma vez. Para uma determinada instância desse problema, temos uma fórmula (o modelo) que, para cada sequência de cidades (as entradas), calculará a duração do passeio (a saída). O problema é encontrar uma entrada com uma saída desejada, ou seja, uma sequência de cidades com comprimento ótimo (mínimo). Observe que, neste exemplo, a saída desejada é definida implicitamente.

Ou seja, em vez de especificar a duração exata, é necessário que o passeio seja mais curto do que todos os outros, e estamos procurando entradas para perceber isso.

Outro exemplo é o do **problema das oito rainhas**. Aqui, recebemos um tabuleiro de xadrez e oito rainhas que precisam ser colocados no tabuleiro de forma que duas rainhas não possam se confrontar, ou seja, elas não devem compartilhar a mesma linha, coluna ou diagonal. Este problema pode ser capturado por um sistema computacional onde uma entrada é uma certa configuração de todas as oito rainhas, o modelo calcula se as rainhas em uma determinada configuração se verificam ou não, e a saída é o número de rainhas que não estão sendo verificadas. Ao contrário do problema do caixeiro viajante, aqui a saída desejada é especificada explicitamente: o número de rainhas que não estão sendo verificadas deve ser oito. Um sistema alternativo capturando este problema poderia ter o mesmo conjunto de entradas, o mesmo modelo, mas a saída pode ser um valor binário simples, representando “OK” ou “não OK”, referindo-se à configuração como um todo. Neste caso, estamos procurando uma entrada que gere “OK” como saída. Intuitivamente, esse problema pode não parecer uma otimização real, porque não existe uma medida graduada de bondade. Na seção 1.3 discutiremos este assunto com mais detalhes.

**1.1.2 Modelagem**

**Em um problema de modelagem ou identificação de sistema, conjuntos correspondentes de entradas e saídas são conhecidos e um modelo do sistema é procurado que forneça a saída correta para cada entrada conhecida** (Fig. 1.2). Em termos de aprendizagem humana, isso corresponde a encontrar um modelo de mundo que corresponda à nossa experiência anterior e que possa generalizar para exemplos ainda não vistos.



Tomemos a **bolsa de valores** como exemplo, onde alguns índices econômicos e sociais (por exemplo, a taxa de desemprego, o preço do ouro, a taxa de câmbio euro-dólar, etc.) formam a entrada e o índice Dow Jones é visto como saída. A tarefa agora é encontrar uma fórmula que ligue as entradas conhecidas às saídas conhecidas, representando assim um modelo deste sistema econômico. Se for possível encontrar um modelo correto para os dados conhecidos (do passado), e se tivermos boas razões para acreditar que as relações capturadas neste modelo permanecem verdadeiras, então temos uma ferramenta de previsão para o valor do índice Dow Jones dado novos dados.

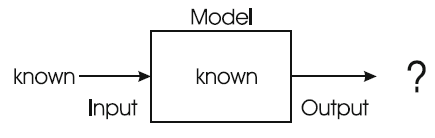
Como outro exemplo, tomemos a tarefa de **identificar sinais de trânsito em imagens** - talvez de feeds de vídeo em um carro inteligente. Nesse caso, o sistema é composto por dois elementos. Em um estágio de pré-processamento, as rotinas de processamento de imagens pegam os sinais elétricos produzidos pela câmera, dividem-nos em regiões de interesse que podem ser sinais de trânsito e para cada uma delas produzem um conjunto de descritores numéricos de tamanho, forma, brilho, contraste, etc. Estes valores representam a imagem em formato digital e consideramos o componente de pré-processamento a ser fornecido por agora. Então, no sistema principal, cada entrada é um vetor de números que descreve um possível sinal, e a saída correspondente é um rótulo de um conjunto predefinido, por exemplo, "pare", "dê passagem", "50", etc. (o tráfego placa). O modelo é então um algoritmo que obtém imagens como entrada e produz rótulos de sinais de trânsito como saída. A tarefa aqui é produzir um modelo que responda com as etiquetas de sinais de trânsito apropriadas em cada situação. Na prática, o conjunto de todas as situações possíveis seria representado por uma grande coleção de imagens, todas rotuladas de forma adequada. Então, o problema de modelagem se reduz a encontrar um modelo que forneça uma saída correta para cada imagem na coleção.

Além disso, o **sistema de controle de voz para residências inteligentes** descrito no início desta seção inclui um problema de modelagem. O conjunto de todas as frases pronunciadas pelo usuário (entradas) deve ser mapeado corretamente no conjunto de todos os comandos de controle no repertório da casa inteligente.

**É importante observar que problemas de modelagem podem ser transformados em problemas de otimização.** O truque geral é designar a taxa de erro de um modelo como a quantidade a ser minimizada ou sua taxa de acerto a ser maximizada. Como exemplo, vejamos o **problema da identificação dos sinais de trânsito**. Isso pode ser formulado como um problema de modelagem: encontrar o modelo correto m que mapeia cada uma de uma coleção de imagens na (s) etiqueta (s) apropriada (s) que identificam os sinais de trânsito nessa imagem. O modelo m que resolve o problema é desconhecido de antemão, daí o ponto de interrogação na Figura 1.2. Para encontrar uma solução, precisamos começar escolhendo uma tecnologia. Por exemplo, podemos desejar tê-lo como uma árvore de decisão, uma rede neural artificial, um pedaço de código Java ou uma expressão MATLAB. Essa escolha nos permite especificar a forma ou sintaxe necessária de m. Feito isso, podemos definir o conjunto de todas as soluções possíveis M para a nossa tecnologia escolhida, sendo todas as expressões corretas na sintaxe dada, por exemplo, todas as árvores de decisão com as variáveis ​​apropriadas ou todas as redes neurais artificiais possíveis com uma determinada topologia. Agora podemos definir um problema de otimização relacionado. O conjunto de entradas é M e a saída para um dado m ∈ M é um número inteiro que indica quantas imagens foram rotuladas corretamente por m. É claro que uma solução deste problema de otimização com o número máximo de imagens corretamente rotuladas é uma solução para o problema de modelagem original.

**1.1.3 Simulação**

**Em um problema de simulação, conhecemos o modelo do sistema e algumas entradas, e precisamos calcular as saídas correspondentes a essas entradas** (Fig. 1.3). Como exemplo, pense em um **circuito eletrônico**, digamos, um filtro cortando as baixas frequências em um sinal. Nosso modelo é um sistema complexo de fórmulas (equações e desigualdades) que descrevem o funcionamento do circuito. Para qualquer sinal de entrada, este modelo pode calcular o sinal de saída. Usar este modelo (por exemplo, para comparar dois projetos de circuito) é muito mais barato do que construir o circuito e medir suas propriedades no mundo físico. Outro exemplo é o de um **sistema de previsão do tempo**. Neste caso, as entradas são os dados meteorológicos relativos à temperatura, vento, umidade, precipitação, etc., e as saídas são na verdade as mesmas: temperatura, vento, umidade, precipitação, etc., mas em um horário diferente. O modelo aqui é temporal para prever dados meteorológicos.



Problemas de simulação ocorrem em muitos contextos e o uso de simuladores oferece várias vantagens em diferentes aplicações. Por exemplo, a simulação pode ser mais econômica do que estudar os efeitos do mundo real, por exemplo, para os **projetistas de circuitos eletrônicos**. A alternativa do mundo real pode não ser viável, por exemplo, realizar análises hipotéticas de vários sistemas fiscais in vivo é praticamente impossível. E a simulação pode ser a ferramenta que nos permite ***olhar para o futuro***, como nos sistemas de previsão do tempo.

**1.2 Problemas de pesquisa**

**Uma suposição profundamente enraizada por trás da visão da caixa preta dos sistemas é que um modelo computacional é direcional: ele calcula a partir das entradas para as saídas e não pode ser simplesmente invertido. Isso implica que resolver um problema de simulação é diferente de resolver um problema de otimização ou modelagem. Para resolver um problema de simulação, precisamos apenas aplicar o modelo a algumas entradas e simplesmente aguardar o resultado.** No entanto, resolver uma otimização ou um problema de modelagem requer a identificação de um objeto particular em um espaço de possibilidades. Este espaço pode ser, e geralmente é, enorme. Isso nos leva à noção de que o processo de resolução de problemas pode ser visto como uma busca por meio de um conjunto potencialmente enorme de possibilidades para encontrar a solução desejada. Conseqüentemente, os problemas que devem ser resolvidos dessa forma podem ser vistos como problemas de busca. Em termos da classificação dos problemas discutidos na Seção 1.1, **os problemas de otimização e modelagem podem ser naturalmente percebidos como problemas de pesquisa, enquanto isso não acontece com os problemas de simulação.**

**Essa visão leva naturalmente ao conceito de um espaço de busca, sendo a coleção de todos os objetos de interesse, incluindo a solução que buscamos**. Dependendo da tarefa em mãos, **o espaço de busca consiste em todas as entradas possíveis para um modelo (problemas de otimização), ou todos os modelos computacionais possíveis que descrevem o fenômeno que estudamos (problemas de modelagem)**. Esses espaços de busca podem ser muito grandes; por exemplo, o número de passeios diferentes por n cidades é (n − 1)! e o número de árvores de decisão com parâmetros de valor real é infinito. **A especificação do espaço de busca é a primeira etapa na definição de um problema de busca. A segunda etapa é a definição de uma solução**. Para problemas de otimização, tal definição pode ser explícita, por exemplo, uma configuração de tabuleiro onde o número de rainhas verificadas é zero, ou implícita, por exemplo, um passeio que é o mais curto de todos os passeios. **Para problemas de modelagem, uma solução é definida pela propriedade de produzir a saída correta para cada entrada**. Na prática, entretanto, isso é freqüentemente relaxado, exigindo apenas que o número de entradas para as quais a saída está correta seja máximo. Observe que essa abordagem transforma o problema de modelagem em um de otimização, conforme ilustrado na Seção 1.1.2.

Essa noção de resolução de problemas como busca nos dá um benefício imediato: **podemos fazer uma distinção entre problemas (de busca) - que definem espaços de busca - e solucionadores de problemas - que são métodos que nos dizem como nos mover através dos espaços de busca.**

**1.3 Otimização versus Satisfação de Restrições**

O esquema de classificação discutido nesta seção é baseado na distinção entre funções objetivo a serem otimizadas e restrições a serem satisfeitas. Em geral, podemos considerar uma função objetivo como uma forma de atribuir um valor a uma solução possível que reflita sua qualidade em uma escala, enquanto uma restrição representa uma avaliação binária que nos informa se um determinado requisito é válido ou não. Nas seções anteriores, ***várias funções objetivo foram mencionadas***, incluindo:

(1) o número de rainhas não verificadas em um tabuleiro de xadrez (a ser maximizado);

(2) a duração de um passeio visitando cada cidade em um determinado conjunto exatamente uma vez (a ser **minimizado**);

(3) o número de imagens em uma coleção que são rotuladas corretamente por um determinado modelo m (a ser **maximizado**).

Esses exemplos ilustram que as soluções para um problema podem ser identificadas em termos de otimização em relação a alguma função objetivo. Além disso, as soluções podem estar sujeitas a **restrições** formuladas como critérios que devem ser satisfeitos.

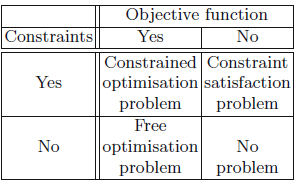
**Restrição: diminui os espaços de busca e elimine soluções inviáveis**

Por exemplo:

(4) Encontre uma configuração de oito rainhas em um tabuleiro de xadrez de forma que não haja duas rainhas se checando.

(5) Encontre um passeio com duração mínima para um caixeiro viajante, de forma que a cidade X seja visitada após a cidade Y.

Há uma série de observações a serem feitas sobre esses exemplos. O exemplo 2 se refere a um problema cuja solução é definida puramente em termos de otimização. Por outro lado, o exemplo 4 ilustra o caso em que uma solução é definida apenas em termos de uma restrição: uma determinada configuração é boa ou não. Observe que essa restrição geral em relação a uma configuração inteira é, na verdade, composta de restrições mais elementares relativas a pares de rainhas. Uma configuração completa está OK se todos os pares de rainhas estiverem OK. O Exemplo 5 é uma mistura desses dois tipos básicos, já que tem uma função objetivo (duração do passeio) e uma restrição (visite X após Y). Com base nessas observações, podemos configurar outro sistema de classificação de problemas, dependendo da presença ou ausência de uma função objetivo e das restrições na definição do problema. As quatro categorias resultantes são mostradas na Tabela 1.1.



Nestes termos, o problema do caixeiro viajante (item 2 acima) é um **problema de otimização livre (FOP)**, o problema das oito rainhas (item 4 acima) é um **problema de satisfação de restrição (CSP)** e o problema mostrado no item 5 é um **problema de otimização restrita (COP)**. Comparando os itens 1 e 4, podemos ver que os problemas de satisfação das restrições podem ser transformados em problemas de otimização. O truque básico é o mesmo que transformar problemas de modelagem em problemas de otimização: em vez de exigir perfeição, apenas contamos o número de restrições satisfeitas (por exemplo, pares de rainhas sem verificação) e apresentamos isso como uma função objetivo a ser maximizada. Obviamente, um objeto (por exemplo, uma configuração de placa) é uma solução para o problema de satisfação da restrição original se e somente se for uma solução deste problema de otimização associado.

Para sustentar outras percepções interessantes sobre os problemas, vamos dar uma olhada mais de perto no problema das oito rainhas. Sua formulação original está em linguagem natural:

**Coloque oito rainhas em um tabuleiro de xadrez de forma que não haja duas rainhas entre si.**

Esta definição de problema é informal no sentido de que não possui qualquer referência aos construtos formais que apresentamos aqui, como entradas / saídas, um espaço de busca, etc. Para desenvolver um algoritmo para este problema, ele precisa ser formalizado. Por acaso, pode ser formalizado de diferentes maneiras, e isso leva a diferentes tipos de problemas formais ao descrevê-lo. A maneira mais fácil de ilustrar uma série de opções é adotar a perspectiva da pesquisa.

**FOP** Se definirmos o espaço de busca S como o conjunto de todas as configurações de tabuleiro com oito rainhas, podemos capturar o problema original como um problema de otimização livre com uma função objetivo f que relata o número de rainhas livres para uma determinada configuração e definir um solução como uma configuração s ∈ S com f (s) = 8.

**CSP** Alternativamente, podemos formalizá-lo como um problema de satisfação de restrição com o mesmo espaço de busca S e definir uma restrição φ tal que φ (s) = verdadeiro se e somente se duas rainhas não verificam uma a outra quanto à configuração s.

**COP** Ainda outra formalização é obtida se tomarmos um espaço de busca diferente. Isso pode ser motivado pela observação de que em qualquer solução do problema das oito rainhas, o número de rainhas em cada coluna deve ser exatamente um. Obviamente, o mesmo vale para as linhas. Portanto, poderíamos distinguir restrições verticais (para colunas), restrições horizontais (para linhas) e restrições diagonais e decidir nos restringir às configurações de placa que já satisfaçam as restrições verticais e horizontais. Esta é uma abordagem viável, visto que é bastante fácil encontrar configurações com uma rainha em cada coluna e em cada linha. Essas configurações são um subconjunto do espaço de busca original - vamos chamá-lo de S. Formalmente, podemos então definir um problema de otimização restrito sobre S com uma restrição modificada ψ tal que ψ (s) = verdadeiro se e somente se todas as restrições verticais e horizontais forem satisfeitas em s (ou seja, φ (s) = verdadeiro se e apenas se s estiver em S) e uma nova função g que relata o número de pares de rainhas em s que violam as restrições diagonais. É fácil ver que uma configuração de tabuleiro é uma solução para o problema das oito rainhas se e somente se ela for uma solução desse problema de otimização restrita com g (s) = 0 e φ (s) = verdadeiro.

**Esses exemplos ilustram que a natureza de um problema é menos óbvia do que pode parecer. Na verdade, tudo depende de como escolhemos formalizá-lo.** Qual a formalização a ser preferida é um assunto para discussão. Pode-se argumentar que algumas formalizações são mais naturais ou se adaptam melhor ao problema do que outras. Por exemplo, pode-se preferir ver o problema das oito rainhas como um problema de satisfação de restrição por natureza e considerar todas as outras formalizações como transformações secundárias. Da mesma forma, pode-se considerar o problema de reconhecimento de sinais de trânsito como um problema de modelagem em primeiro lugar e transformá-lo em um problema de otimização para fins práticos. Considerações algorítmicas também podem ser uma grande influência aqui. Se alguém tem um algoritmo que pode resolver bem os problemas de otimização livre, mas não consegue lidar com as restrições, então é muito sensato formalizar os problemas como otimização livre.

**1.4 Os famosos problemas NP**

Até este ponto, discutimos várias maneiras diferentes de categorizar problemas e deliberadamente evitamos as discussões sobre os solucionadores de problemas. Conseqüentemente, é possível classificar um problema de acordo com um desses esquemas apenas olhando para o problema. Nesta seção, discutimos um esquema de classificação em que isso não é possível porque as categorias de problemas são definidas por meio das propriedades dos algoritmos de solução de problemas. A motivação por trás dessa abordagem é a intenção de falar sobre os problemas em termos de sua dificuldade, por exemplo, serem difíceis ou fáceis de resolver. Em termos gerais, a ideia básica é chamar um problema de fácil se houver um solucionador rápido para ele e difícil de outra forma. **Essa noção de dureza do problema leva ao estudo da complexidade computacional.**

Antes de prosseguir, precisamos fazer uma distinção adicional entre os problemas de otimização, dependendo do tipo de objetos no espaço de busca correspondente. Se o espaço de busca S é definido por variáveis ​​contínuas (ou seja, números reais), então temos um problema de otimização numérica. Se S é definido por variáveis ​​discretas (por exemplo, booleanos ou inteiros), então temos um problema de otimização combinatória. As várias noções de dureza do problema discutidas mais adiante são definidas para problemas de otimização combinatória. Observe que os espaços de busca discretos são sempre finitos ou, no pior caso, contavelmente infinitos.

Não tentamos fornecer uma visão geral completa da complexidade computacional, pois isso é bem abordado em muitos livros, como [180, 330, 331, 318]. Em vez disso, fornecemos um breve esboço de alguns conceitos importantes, suas implicações para a solução de problemas e também de alguns equívocos muito comuns. Além disso, não tratamos o assunto com rigor matemático, pois não seria apropriado para este livro. Assim, não damos definições precisas de conceitos essenciais, como algoritmo, tamanho do problema ou tempo de execução, mas usamos tais termos de maneira intuitiva, explicando seu significado por exemplos, se necessário.

**A primeira noção-chave na complexidade computacional é a do tamanho do problema, que se baseia na dimensionalidade do problema em questão (ou seja, o número de variáveis) e o número de valores diferentes para as variáveis ​​do problema**. Para os exemplos discutidos antes, o número de cidades a visitar ou o número de rainhas a serem colocadas no tabuleiro podem ser medidas sensatas para indicar o tamanho do problema. **A segunda noção diz respeito a algoritmos, ao invés de problemas**. O tempo de execução de um algoritmo é o número de etapas ou operações elementares que leva para terminar. A intuição geral, embora nem sempre correta, por trás da complexidade computacional é que problemas maiores precisam de mais tempo para serem resolvidos. As definições mais conhecidas de dureza do problema relacionam o tamanho de um problema ao (pior caso) tempo de execução de um algoritmo para resolvê-lo. Essa relação é expressa por uma fórmula que especifica um limite superior para o pior caso de tempo de execução como uma função do tamanho do problema. Simplificando, esta fórmula pode ser polinomial (considerada para indicar tempos de execução relativamente curtos) ou superpolinomial, por exemplo, exponencial (indicando tempos de execução longos). A noção final é a de redução do problema, que é a ideia de que podemos transformar um problema em outro por meio de um mapeamento adequado. Observe que a transformação pode não ser reversível. Embora essa ideia de transformar ou reduzir problemas seja ligeiramente complexa, não é totalmente estranha, pois vimos na seção anterior que um determinado problema no mundo real pode muitas vezes ser formalizado de maneiras diferentes, mas equivalentes. As noções freqüentemente usadas com relação à dureza do problema podem agora ser formuladas da seguinte forma.

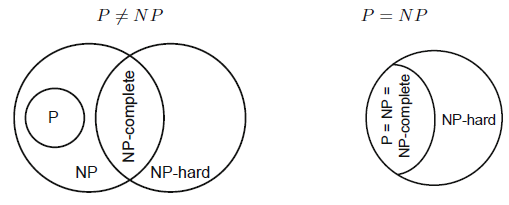
Diz-se que um problema pertence à classe P se existe um algoritmo que o pode resolver em tempo polinomial. Ou seja, se existe um algoritmo para ele cujo tempo de execução de pior caso para o tamanho do problema n é menor que F (n) para alguma fórmula polinomial F. **Em linguagem comum, o conjunto P contém os problemas que podem ser facilmente resolvidos, por exemplo, o problema de Minimum Spanning Tree.**

Diz-se que um problema pertence à **classe NP** se ele pode ser resolvido por algum algoritmo (sem alegações sobre seu tempo de execução) e qualquer solução pode ser verificada dentro do tempo polinomial por algum outro algoritmo. Observe que se segue que P é um subconjunto de NP, uma vez que um solver polinomial também pode ser usado para verificar soluções em tempo polinomial. **Um exemplo de um problema NP é o problema da soma do subconjunto:** dado um conjunto de inteiros, existe algum conjunto de um ou mais elementos desse conjunto cuja soma é zero? Claramente, dar uma resposta negativa a esse problema para um determinado conjunto de números exigiria o exame de todos os subconjuntos possíveis. Infelizmente, o número de subconjuntos possíveis é mais do que polinomial no tamanho do conjunto. No entanto, verificar se uma solução é válida envolve apenas somar o conteúdo do subconjunto descoberto.

Diz-se que um problema pertence à classe NP-completo se pertence à classe NP e qualquer outro problema em NP pode ser reduzido a este problema por um algoritmo que roda em tempo polinomial. Na prática, isso representa problemas difíceis que surgem o tempo todo. Várias listas grandes de exemplos bem conhecidos de problemas NP-completos podem ser facilmente encontradas na Internet - não tentaremos resumir, a não ser dizer que a vasta maioria dos problemas interessantes em ciência da computação acabam sendo NP-completos.

Finalmente, diz-se que um problema pertence à classe NP-difícil se for pelo menos tão difícil quanto qualquer problema em NP-completo (então todos os problemas em NP-completo podem ser reduzidos a um em NP-difícil), mas onde as soluções não pode necessariamente ser verificado dentro do tempo polinomial. Um exemplo é o problema da parada.

A existência de problemas onde uma solução não pode ser verificada em tempo polinomial prova que a classe P não é igual à classe NP-difícil. O que não se sabe é se as duas classes P e NP são de fato iguais. Se fosse esse o caso, as implicações seriam enormes para a ciência da computação e a matemática, pois seria sabido que devem existir algoritmos rápidos para problemas que antes se pensava serem difíceis. Portanto, se P = NP é um dos grandes desafios da teoria da complexidade, e há uma recompensa de um milhão de dólares oferecida por qualquer prova de que P = NP ou P = NP. Observe que, embora o último seja o assunto de matemática muito complexa, o primeiro poderia simplesmente ser provado pela criação de um algoritmo rápido para qualquer um dos problemas NP-completos, por exemplo, um algoritmo para o problema do caixeiro-viajante cujo pior caso tempo de execução escalado polinomialmente com o número de cidades. A Figura 1.4 mostra as classes de dureza do problema dependendo da igualdade de P e NP. Se P = NP então os conjuntos P = NP = NP-completo, mas eles ainda são um subconjunto de NP-difícil.



Embora pareça um tanto teórico, tem algumas implicações muito importantes para a solução de problemas. Se um problema for NP-completo, embora possamos ser capazes de resolver instâncias particulares dentro do tempo polinomial, não podemos dizer que seremos capazes de fazer isso para todas as instâncias possíveis. Assim, se quisermos aplicar métodos de resolução de problemas a esses problemas, devemos atualmente aceitar que provavelmente só podemos resolver instâncias muito pequenas (ou de outra forma fáceis), ou desistir da ideia de fornecer soluções exatas e confiar em aproximação ou metaheurísticas para criar soluções boas o suficiente. Isso está em contraste com problemas que são conhecidos em P. Embora o número de soluções possíveis para esses problemas possa escalar exponencialmente, existem algoritmos que encontram soluções e cujos tempos de execução escalam polinomialmente com o tamanho da instância.

Para resumir esta seção, há um grande número de problemas práticos que, ao serem examinados, acabam sendo uma variante de um problema abstrato que se sabe estar na classe NP-completo. Embora algumas instâncias de tal problema possam ser fáceis, a maioria dos cientistas da computação acredita que nenhum algoritmo de tempo polinomial existe para tais problemas, e certamente um ainda não foi descoberto. Portanto, se quisermos ser capazes de criar soluções aceitáveis ​​para qualquer instância de tal problema, devemos nos voltar para o uso de aproximação e metaheurísticas e abandonar a ideia de encontrar definitivamente uma solução que seja provavelmente a melhor para a instância.