SATISFAÇÃO E OPTIMIZAÇÃO BOOLEANAS: JOGOS, PUZZLES E GENÉTICA

RUBEN MARTINS E ANA GRAÇA

Resumo. Como escolher alguns amigos para jantar? Qual a solução daqueles problemas de Sudoku que nos quebram a cabeça? Conseguiremos resolver o complicado *puzzle* das Torres de Hanói? Como separar a informação genética que herdamos dos nossos pais? Este artigo explica como satisfação (sat) e optimização booleanas podem ser usadas para modelar diversos problemas "difíceis" reais; descreve ainda a importância de sat para a teoria da complexidade. Será o leitor capaz de resolver um dos Problemas do Milénio?

1. Introdução

Desejo convidar três amigos para o meu jantar de aniversário [6]. Posso escolher três pessoas de um grupo de cinco amigos. No entanto, de forma a ter um jantar tranquilo, terei de ter em conta os relacionamentos pessoais entre eles. O António não fala com a Beatriz, sua antiga namorada. Esta e o Carlos são inseparáveis. O Carlos está zangado com o Daniel, e este, por sua vez, é inseparável da Eduarda. A Eduarda não tolera o António. Que amigos devo escolher para o jantar sem criar conflitos? Este quebra-cabeças lógico é um exemplo simples de um problema de satisfação (sat). Consideremos os problemas sat como problemas de decisão (problemas de resposta sim/não) e procura (em caso de resposta afirmativa, é necessário encontrar uma solução). No exemplo do jantar de anos, queremos decidir se é possível ou não escolher três amigos, sem colocar nenhum deles numa situação desconfortável e fornecer uma resposta (que amigos convidar?) no caso de o problema poder ser satisfeito.

Outro conhecido quebra-cabeças é o Problema de Einstein [15] que, ao que parece, foi formulado no fim do século xix. Uma das suas variantes pode ser enunciada da seguinte forma: "Existem cinco casas de diferentes cores. Em cada casa, mora uma pessoa de uma diferente nacionalidade. Os cinco proprietários bebem diferentes bebidas, praticam diferentes desportos e têm diferentes animais de estimação. O inglês vive na casa vermelha. O sueco tem um cão como animal de estimação. O dinamarquês bebe chá. A casa verde fica à esquerda da casa branca. O dono da casa verde bebe café. O homem que joga voleibol cria pássaros. O dono da casa amarela pratica natação. O dono da casa do centro bebe leite. O norueguês vive na primeira casa. O homem que pratica futebol vive ao lado do que tem gatos. O homem que cria cavalos vive ao lado do que pratica natação. O homem que joga basquetebol bebe cerveja. O alemão pratica ténis de mesa. O norueguês vive ao lado da casa azul. O homem que joga futebol é vizinho do que bebe água. A questão é: quem tem o peixe?" Este problema também pode ser formulado através de uma codificação em sat. Será possível distribuir as cinco pessoas/cores/bebidas/desportos/animais pelas cinco casas, sem desrespeitar nenhuma restrição? Se sim, qual é essa distribuição?

Publicado em *Números, cirurgias e nós de gravata: 10 anos de Seminário Diagonal no IST*, J. P. Boavida, R. P. Carpentier, L. Cruz-Filipe, P. S. Gonçalves, E. Grifo, D. Henriques, A. R. Pires (editores), IST Press, 2012.

Deixamos a resolução destes dois problemas para divertimento do leitor interessado.

Mas, afinal, o que é SAT? O problema da satisfação booleana (SAT) pode ser caracterizado por lógica proposicional e complexidade computacional. SAT é um problema que, aparentando ser fácil, é difícil. É fácil na sua formulação: os problemas SAT são codificados através de lógica proposicional, uma lógica com uma expressividade reduzida. É difícil na sua resolução: conjectura-se que encontrar uma solução ou provar que não existe solução tem uma complexidade exponencial[1] na dimensão do problema. Provar a veracidade ou falsidade desta conjectura implica resolver o problema P vs. NP, um dos Problemas do Milénio.

E qual a importância de SAT? O problema SAT é aplicado em inúmeras situações como, por exemplo, a produção de horários, problemas de planeamento, automação de desenho de circuitos electrónicos, verificação de modelos, bioinformática, modelação e verificação de *software*. Consequentemente, um grande esforço tem sido realizado para desenvolver teoria sobre o problema SAT e algoritmos eficientes capazes de o resolver.

2. Satisfação e Optimização Booleanas

Em *lógica booleana* (ou *lógica proposicional*), cada variável proposicional[2] pode tomar um de dois valores: 1 (verdadeiro) ou 0 (falso). Seja $X = \{x_i : i \in \mathbb{N}\}$ o conjunto das variáveis proposicionais e dados os símbolos $\neg$ (negação), $\wedge$ (e), $\vee$ (ou), $\Rightarrow$ (implicação), $\Leftrightarrow$ (equivalência) podemos definir o conjunto das fórmulas proposicionais.

O conjunto Φ das *fórmulas proposicionais* pode ser definido indutivamente[3] da seguinte forma:

- $x_i \in \Phi$, para cada $i \in \mathbb{N}$,
- $(\neg\varphi) \in \Phi$, desde que $\varphi \in \Phi$,
- $(\varphi_1 \vee \varphi_2) \in \Phi$, desde que $\varphi_1, \varphi_2 \in \Phi$,
- $(\varphi_1 \wedge \varphi_2) \in \Phi$, desde que $\varphi_1, \varphi_2 \in \Phi$.

define-se $(\bigvee_{i=1}^{n} \varphi_i)$ como $(\varphi_1 \vee \varphi_2 \vee \ldots \vee \varphi_n)$, $n \in \mathbb{N}$. De forma semelhante, define-se $(\bigwedge_{i=1}^{n} \varphi_i)$ como $(\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n)$, $n \in \mathbb{N}$.

A implicação, $(\varphi_1 \Rightarrow \varphi_2)$, apesar de não ter sido definida, pode ser vista como uma abreviatura de $(\neg\varphi_1 \vee \varphi_2)$. A equivalência, $(\varphi_1 \Leftrightarrow \varphi_2)$, é uma abreviatura de $((\varphi_1 \Rightarrow \varphi_2) \wedge (\varphi_2 \Rightarrow \varphi_1))$.

De notar que o *ou* $(\vee)$ não é exclusivo, isto é, para a frase "vou jogar futebol ou vou ver um filme" ser verdadeira, temos três possibilidades: "jogar futebol", "ver filme", ou ambas as orações serem verdadeiras "jogar futebol e ver filme". Também é de notar que a *implicação* $(\Rightarrow)$ só pode ser lida num sentido. Por exemplo, a partir da frase "se chover então fico em casa" nada pode ser inferido no caso de não chover, isto é, se não chover poderei ficar em casa ou não.

Uma *atribuição* ρ de valores às variáveis proposicionais é uma função que a cada variável associa o valor 1 (verdadeiro) ou 0 (falso), isto é, $\rho : X \to \{0, 1\}$.

---

[1] Diz-se que um problema tem complexidade exponencial se o tempo de computação do algoritmo mais eficiente que o resolva for limitado superior e inferiormente por funções exponenciais no tamanho da instância do problema.

[2] Uma *variável* é um símbolo usado para representar um elemento não especificado de um determinado conjunto. Uma *variável proposicional* ou *booleana* pode apenas tomar valores no conjunto $\{0, 1\}$.

[3] Informalmente, por *definição indutiva* de um conjunto entende-se que apenas lhe pertencem os elementos cuja existência é forçada pelas condições listadas.

TABELA 1: Tabela de verdade para a fórmula $\varphi = (\omega_1 \wedge \omega_2 \wedge \omega_3)$, onde $\omega_1 = (x_1 \vee x_2 \vee \neg x_3)$, $\omega_2 = (x_2 \vee x_3)$ e $\omega_3 = (\neg x_1 \vee \neg x_2 \vee \neg x_3)$.

x_1	x_2	x_3	ω_1	ω_2	ω_3	$(\omega_1 \wedge \omega_2 \wedge \omega_3)$
1	1	1	1	1	0	0
1	1	0	1	1	1	1
1	0	1	1	1	1	1
0	1	1	1	1	1	1
1	0	0	1	0	1	0
0	1	0	1	1	1	1
0	0	1	0	1	1	0
0	0	0	1	0	1	0

Satisfazer uma fórmula proposicional significa encontrar uma atribuição de valores (verdadeiro ou falso) às variáveis de forma a tornar a fórmula verdadeira. Para isso, é necessário conhecer a definição de satisfação de uma fórmula proposicional.

A *satisfação de uma fórmula proposicional* φ através de uma atribuição ρ (escrito $\rho \models \varphi$) é definida indutivamente da seguinte forma:

- $\rho \models x_i$ se e só se $\rho(x_i) = 1$,
- $\rho \models (\neg\varphi)$ se e só se $\rho \not\models \varphi$,
- $\rho \models (\varphi_1 \vee \varphi_2)$ se e só se $\rho \models \varphi_1$ ou $\rho \models \varphi_2$,
- $\rho \models (\varphi_1 \wedge \varphi_2)$ se e só se $\rho \models \varphi_1$ e $\rho \models \varphi_2$.

Por exemplo, considere o problema do jantar de anos, definido na introdução desta secção. Seja $x_i = 1$ se o amigo i for convidado para o jantar, com $i \in \{a, b, c, d, e\}$, onde a, b, c, d e e são usados para abreviar, respectivamente, António, Beatriz, Carlos, Daniel e Eduarda. Desta forma, $(x_a \Rightarrow \neg x_b)$ é equivalente a dizer que, caso o António seja convidado para o jantar, então a Beatriz não é. Por outro lado, para a fórmula $(x_b \Leftrightarrow x_c)$ ser satisfeita é necessário que tanto a Beatriz como o Carlos sejam convidados ou que ambos não sejam convidados.

Um *problema de satisfação booleana* (SAT) consiste em, dada uma fórmula φ, decidir se existe uma atribuição que satisfaça φ e, em caso afirmativo, devolver essa atribuição.

Definiremos ainda alguns conceitos. Um *literal* ℓ_i é definido como uma variável x_i (literal positivo) ou o seu complementar $\neg x_i$ (literal negativo). Uma *cláusula* é a disjunção de literais, $(\bigvee_i \ell_i)$. Uma fórmula proposicional φ é CNF (forma normal conjuntiva, do inglês *conjunctive normal form*) se for a conjunção ($\wedge$) de disjunções de literais, i.e., $\bigwedge_i(\bigvee_j \ell_{ij})$.

Dada uma valoração, uma fórmula CNF é satisfeita se *todas* as suas cláusulas são satisfeitas. Uma cláusula é satisfeita se pelo menos um dos seus literais é satisfeito. Um literal positivo x_i é satisfeito se o valor da variável x_i for 1, enquanto um literal negativo $\neg x_i$ é satisfeito se o valor da variável x_i for 0. Uma fórmula proposicional φ diz-se *satisfazível* se existe uma atribuição de valores às variáveis que satisfaça φ.

Prova-se que, para todas as fórmulas proposicionais, existe uma fórmula CNF equivalente. Por este facto, e de forma a usar um formato padrão que facilite as inferências, em geral em SAT trabalha-se com fórmulas CNF.

A título de exemplo, consideremos a seguinte fórmula CNF: $\varphi = (\omega_1 \wedge \omega_2 \wedge \omega_3)$ onde $\omega_1 = (x_1 \vee x_2 \vee \neg x_3)$, $\omega_2 = (x_2 \vee x_3)$ e $\omega_3 = (\neg x_1 \vee \neg x_2 \vee \neg x_3)$. Esta fórmula tem três variáveis e três cláusulas. A cláusula ω_1 tem dois literais positivos e um literal negativo. A cláusula ω_2 apenas tem literais positivos, enquanto a cláusula ω_3 apenas tem literais negativos. A atribuição $\{x_1 = 1, x_2 = 1, x_3 = 0\}$ satisfaz a fórmula φ, enquanto a atribuição $\{x_1 = 1, x_2 = 1, x_3 = 1\}$ não satisfaz a fórmula

×	1	8				7		
×	×	×	③			2		
×	7	(a)						
				7	1			
6							4	
③								
4			5					3
	2			8				
							6	

	1	8				⑦		
×	×	×	3	×	(b)	2	×	×
	⑦	3						
				⑦	1			
6							4	
3								
4			5					3
	2			8				
							6	

FIGURA 1: Resolução de um quebra-cabeças de Sudoku.

φ. Uma forma possível (para este pequeno exemplo) de verificar que atribuições satisfazem a fórmula φ é através da tabela de verdade—Tabela 1. É de notar, no entanto, que verificar todas as possíveis atribuições só é praticável para fórmulas com número reduzido de variáveis. A Tabela 1 tem $2^3 = 8$ linhas, pois a fórmula φ tem três variáveis. Uma tabela de verdade para uma fórmula com quatro variáveis teria $2^4 = 16$ linhas. Uma fórmula com dez variáveis originaria uma tabela de verdade com $2^{10} = 1024$ linhas.

Os problemas SAT podem ser estendidos para problemas de optimização combinatória. Um *problema de optimização booleana* consiste em encontrar uma atribuição para um conjunto de variáveis booleanas de forma a satisfazer um conjunto de restrições e optimizar uma dada função objectivo. No nosso exemplo inicial do jantar de aniversário, um problema de optimização seria: qual o número máximo de amigos que posso convidar para o jantar, satisfazendo todas as restrições?

Problemas de optimização booleana têm sido estudados desde 1968 [12], em diferentes contextos, tais como investigação operacional, inteligência artificial e automação de desenho de circuitos.

3. APLICAÇÃO 1: SUDOKU

O Sudoku é um quebra-cabeças que se tornou popular no Japão em 1986 e que recentemente ganhou popularidade a nível mundial. Este quebra-cabeças é cativante, visto que as suas regras são simples de perceber, mas no entanto o raciocínio necessário para o resolver é desafiante. O quebra-cabeças do Sudoku é representado por uma matriz 9×9, com nove submatrizes 3×3. Algumas das entradas da matriz estão pré-preenchidas com números de 1 a 9 e as restantes entradas estão vazias. O Sudoku está resolvido quando a matriz estiver preenchida de acordo com as seguintes regras: cada coluna da matriz contém os números de 1 a 9; cada linha da matriz contém os números de 1 a 9; cada submatriz contém os números de 1 a 9. As regras que definem o Sudoku explicam o seu nome em japonês: "número único".

A Figura 1 ilustra como se resolve um quebra-cabeças de Sudoku. As casas pré-preenchidas estão assinaladas com a cor cinzenta, enquanto as entradas vazias estão assinaladas com a cor branca. Consideremos a entrada (a) na matriz do lado esquerdo da figura. Na submatriz 3×3 onde a entrada (a) está incluída, o número 3 tem de ocorrer necessariamente na posição (a): o número 3 tem de ocorrer nesta submatriz; o número 3 não pode ocorrer na segunda linha pois já está presente nessa linha; e o número 3 não pode ocorrer na primeira coluna pois já se encontra presente nessa coluna. De forma similar, se considerarmos a entrada (b) na matriz

1					7		9	
	3			2				8
		9	6			5		
		5	3			9		
	1			8				2
6					4			
3							1	
	4							7
		7				3		

FIGURA 2: "AI Escargot": um Sudoku de difícil resolução.

do lado direito da mesma figura, podemos concluir que esta posição é o único local onde o número 7 pode ser colocado na segunda linha. Se aplicarmos sucessivamente regras de inferência conseguimos preencher toda a matriz e resolver desta forma o Sudoku.

Dois Sudokus diferem entre si no número de casas pré-preenchidas e na forma como estas são preenchidas. Um Sudoku é denominado *mínimo* se tiver o menor número de casas pré-preenchidas de forma a que tenha apenas uma solução. Estudos recentes [28] mostraram que não existem Sudokus mínimos com 16 casas pré-preenchidas e que os Sudokus são mínimos se tiverem 17 casas pré-preenchidas. Gordon Royle tem a maior colecção de Sudokus com 17 casas pré-preenchidas. Em [30] encontra-se disponível esta colecção, que contém de momento 49 151 Sudokus.

A dificuldade de cada quebra-cabeças de Sudoku é muitas vezes associada com o número de casas pré-preenchidas. No entanto, Inkala argumenta que o número de casas pré-preenchidas não é um bom indicador da dificuldade do problema [14]. Inkala apresentou um Sudoku denominado "AI Escargot" que contém 23 casas pré-preenchidas e foi considerado em Novembro de 2006 como o Sudoku mais difícil de resolver. Na Figura 2 apresentamos este Sudoku e desafiamos o leitor a tentar resolver este quebra-cabeças.

A dificuldade de um Sudoku é normalmente medida através das técnicas necessárias para o resolver. No entanto, SAT também pode ser usado para avaliar a dificuldade de um Sudoku através das diferentes técnicas de inferência necessárias para o resolver [13]. Entre essas técnicas de inferência destacamos a *propagação unitária* e a *regra do literal falhado*. Dada uma cláusula unitária[4] $\omega = (x_i)$, a propagação unitária consiste em simplificar todas as cláusulas que contenham o literal x_i através da propagação do seu valor. Para ω ser satisfeita, é necessário que x_i tome o valor 1. Logo, todas as cláusulas que contenham x_i serão satisfeitas. Por outro lado, as cláusulas onde o literal $\neg x_i$ ocorre podem ser simplificadas. Uma vez que $\neg x_i$ tomará sempre o valor 0, podemos remover este literal de todas as cláusulas onde ocorre. A propagação unitária é aplicada iterativamente até não existirem mais cláusulas unitárias. Na Figura 1 ilustramos como resolver um Sudoku. A técnica de resolução mostrada pode ser vista como a aplicação da propagação unitária sobre as casas pré-preenchidas do Sudoku. Outra regra de inferência usada para resolver o Sudoku é a *regra do literal falhado*. Esta regra consiste em atribuir um valor (0 ou 1) a um literal x_i e verificar através de propagação unitária se chegamos

[4]Uma *cláusula unitária* é uma cláusula que apenas contém um literal.

a um *conflito*. Encontramos um conflito se o valor que atribuímos a x_i não satisfizer a fórmula. Neste caso, podemos concluir que x_i terá de ter o valor oposto ao que foi inicialmente atribuído. No caso do Sudoku, esta técnica pode ser vista se, ao atribuirmos um valor a uma casa e aplicarmos propagação unitária, concluirmos que iremos chegar a um estado inválido do Sudoku. Neste caso, podemos inferir que esse valor não pode ser atribuído a essa casa. Usando apenas propagação unitária é possível resolver cerca de metade dos Sudokus disponíveis na colecção de Gordon Royle [22]. Por outro lado, se usarmos propagação unitária conjuntamente com a regra do literal falhado é possível resolver todos os Sudokus presentes nesta colecção [22]. No entanto, há Sudokus que não conseguem ser resolvidos apenas com o uso destas técnicas. Por exemplo, para resolver o "AI Escargot" é necessário o uso de outras técnicas de inferência [13].

Para modelar um problema SAT começamos por atribuir um significado a cada variável booleana. Se usássemos variáveis com domínio $\{1, \ldots, 9\}$ poderíamos codificar o problema usando 9×9 variáveis, isto é, para cada entrada usaríamos uma variável que denotaria o número a ser atribuído a essa entrada. No entanto, visto que temos variáveis booleanas iremos usar nove variáveis booleanas para cada entrada, cada uma destas variáveis irá representar se um determinado número ocorre ou não nesta posição. Isto é, no total usaremos $9 \times 9 \times 9$ variáveis booleanas.

Consideremos que $\mathcal{S}$ é a matriz do Sudoku. Sejam s_{xyz} as variáveis do problema, a variável s_{xyz} tem o valor 1 (verdadeiro) se e só se o número z ocorrer na entrada da linha x e coluna y. Logo, $s_{483} = 1$ significa que $\mathcal{S}[4, 8] = 3$. Analogamente, $s_{483} = 0$ significa que $\mathcal{S}[4, 8] \neq 3$. Os números pré-preenchidos da matriz do Sudoku podem ser simplesmente representados por cláusulas unitárias.

Após definirmos as variáveis do problema é necessário codificar as restrições que modelam o problema. Um quebra-cabeças de Sudoku pode ser modelado em SAT usando uma codificação *mínima* ou *estendida* [22]. A codificação *mínima* é suficiente para modelar o problema, enquanto a codificação *estendida* codifica restrições adicionais para facilitar a resolução do problema.

A codificação mínima garante que: existe pelo menos um número em cada entrada; cada número aparece no máximo uma vez em cada linha; cada número aparece no máximo uma vez em cada coluna; e cada número aparece no máximo uma vez em cada submatriz 3×3.

Por exemplo, a seguinte cláusula garante que existe pelo menos um número na entrada $\mathcal{S}[1, 1]$:

$$(s_{111} \vee s_{112} \vee s_{113} \vee s_{113} \vee s_{114} \vee s_{115} \vee s_{116} \vee s_{117} \vee s_{118} \vee s_{119}).$$

Cláusulas semelhantes podem ser definidas para as outras entradas da matriz $\mathcal{S}$.

Consideremos também a primeira linha da matriz $\mathcal{S}$. As seguintes cláusulas garantem que o número 1 ocorre no máximo uma vez na linha 1:

$$\bigwedge_{y=1}^{8} \bigwedge_{i=y+1}^{9} (\neg s_{1y1} \vee \neg s_{1i1}).$$

Ou seja, para cada duas posições distintas na linha 1, temos de garantir que, se o número 1 ocorre numa posição, não irá ocorrer na outra posição. Se considerarmos as posições $\mathcal{S}[1, 1]$ e $\mathcal{S}[1, 7]$, a seguinte cláusula, que é gerada pela fórmula acima indicada, garante que o número 1 pode ocorrer no máximo uma vez numa destas posições:

$$(\neg s_{111} \vee \neg s_{171}).$$

Consideremos agora a segunda coluna da matriz $\mathcal{S}$. As seguintes cláusulas garantem que o número 3 ocorre, no máximo, uma vez na coluna 2:

$$\bigwedge_{x=1}^{8} \bigwedge_{i=x+1}^{9} (\neg s_{x23} \vee \neg s_{i23}).$$

Para cada duas posições distintas na coluna 2, temos de garantir que, se o número 3 ocorre numa posição, não irá ocorrer na outra posição.

É também necessário garantir que cada número aparece no máximo uma vez em cada submatriz 3×3. A título de exemplo, consideremos a submatriz no canto superior esquerdo da matriz $\mathcal{S}$. As seguintes cláusulas garantem que o número 5 ocorre, no máximo, uma vez nessa submatriz:

$$\bigwedge_{x=1}^{3} \bigwedge_{y=1}^{2} \bigwedge_{k=y+1}^{3} (\neg s_{xy5} \vee \neg s_{xk5}), \qquad \bigwedge_{x=1}^{2} \bigwedge_{y=1}^{3} \bigwedge_{k=x+1}^{3} \bigwedge_{l=1}^{3} (\neg s_{xy5} \vee \neg s_{kl5}).$$

Analogamente aos exemplos anteriores, para duas posições distintas na submatriz temos de garantir que, se o número 5 ocorre numa posição, não irá ocorrer na outra posição.

Esta codificação pode ser estendida através da introdução de restrições adicionais que garantem que *exactamente* um número ocorre em cada entrada, linha, coluna e submatriz. Além das restrições já apresentadas, esta codificação inclui também as seguintes restrições adicionais: existe no máximo um número em cada entrada; cada número aparece pelo menos uma vez em cada linha; cada número aparece pelo menos uma vez cada coluna; e cada número aparece pelo menos uma vez em cada submatriz. Uma descrição mais detalhada da codificação estendida encontra-se em [22]. A codificação do problema pode ser melhorada. Em vez de usarmos cláusulas unárias para definir as entradas pré-preenchidas, podemos usar o conhecimento sobre as entradas pré-preenchidas para remover cláusulas redundantes durante a fase de codificação. Isto permite uma codificação mais compacta do problema [19].

Para codificar o problema do Sudoku usámos $9 \times 9 \times 9 = 729$ variáveis. Para construir a respectiva tabela de verdade seriam necessárias 2^{729} linhas, o que não é praticável para resolver este problema. Na realidade, os algoritmos de SAT são bastante mais complexos e eficientes do que o uso de tabelas de verdade. De facto, actualmente os algoritmos de SAT são capazes de resolver problemas com milhões de variáveis e cláusulas. Uma descrição detalhada das várias técnicas utilizadas nos algoritmos de SAT pode ser vista em [25]. Na resolução do problema do Sudoku, os algoritmos de SAT são bastante eficientes e podem resolver qualquer Sudoku em apenas alguns segundos.

4. Aplicação 2: Torres de Hanói

As Torres de Hanói são um *puzzle* matemático formalizado pelo matemático francês Édouard Lucas em 1883 [23]. Cada problema consiste em três torres e n discos de diferentes tamanhos que podem ser colocados em qualquer torre. Na Figura 3 é apresentado o estado inicial do problema com três discos. Todos os discos são empilhados por ordem decrescente de tamanho na torre 1. O objectivo do problema é mover todos os discos para a torre 3, obedecendo às seguintes regras: apenas um disco pode ser movido de cada vez; nenhum disco pode ser colocado em cima de um disco mais pequeno; e cada movimento consiste em tirar o disco que se encontra no topo de uma das torres e colocá-lo noutra torre, no topo de outros discos que possam já estar presentes nessa torre ou directamente na torre se esta estiver vazia.

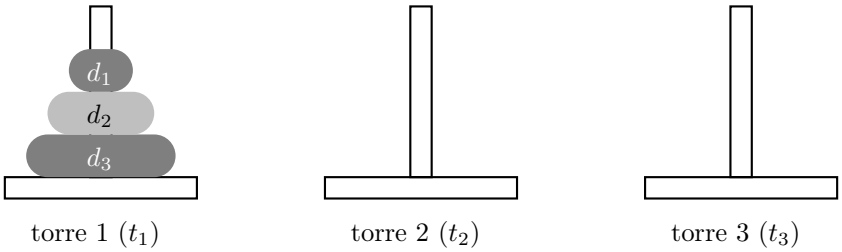

FIGURA 3: Torres de Hanói com três discos.

As Torres de Hanói têm diversas propriedades [31] que podem ser usadas na sua modelação. Por exemplo, dadas três torres e n discos, o número de movimentos necessários ($M(n)$) para resolver as Torres de Hanói é dado pela seguinte relação:

$$M(1) = 1, \qquad M(n) = 2 \times M(n-1) + 1 = 2^n - 1.$$

Logo, podemos concluir que um problema com n discos é resolvido em $2^n - 1$ passos. Apesar do número exponencial de passos, o problema das Torres de Hanói pode ser resolvido através de um simples procedimento. Se o número de discos no problema das Torres de Hanói for *par* então a solução é dada pelo seguinte método: fazer o movimento válido[5] entre os discos das torres t_1 e t_2; fazer o movimento válido entre os discos das torres t_1 e t_3; fazer o movimento válido entre os discos das torres t_2 e t_3; e repetir os passos anteriores até todos os discos estarem na torre t_3. No caso de o número de discos no problema das Torres de Hanói ser ímpar, a solução é semelhante à apresentada, mas com movimentos entre torres diferentes a cada passo. Desafiamos o leitor a descobrir quais as torres que devem ser consideradas quando o número de discos é ímpar. De forma a ajudar o leitor, damos em seguida a solução para o problema das Torres de Hanói com 3 discos: (1) mover o disco d_1 da torre t_1 para a torre t_3 ($d_1 : t_1 \to t_3$), (2) $d_2 : t_1 \to t_2$, (3) $d_1 : t_3 \to t_2$, (4) $d_3 : t_1 \to t_3$, (5) $d_1 : t_2 \to t_1$, (6) $d_2 : t_2 \to t_3$ e, por fim, (7) $d_1 : t_1 \to t_3$.

Apesar de SAT não ser o método mais eficiente para resolver o problema das Torres de Hanói, devido à sua simplicidade iremos usar este exemplo para descrever como podemos modelar problemas de planeamento usando SAT. Existem várias codificações do problema das Torres de Hanói em SAT [17, 16, 29, 26, 27]. Todas estas codificações modelam o problema tendo como base no modelo STRIPS [8] para problemas de planeamento. O modelo STRIPS é composto por: conjunto de variáveis, conjunto de acções, estado inicial e estado final. Para cada acção é necessário considerar as *pré-condições* sobre o estado, que são necessárias para que a acção possa ser realizada e as *implicações* que serão as mudanças no estado causadas pela acção. Em seguida, apresentamos uma das modelações mais simples para o problema das Torres de Hanói [17, 16].

O conjunto das variáveis do problema das Torres de Hanói será $\{\text{sobre}(d, dt, i), \text{livre}(dt, i), \text{move}(d, dt, dt', i)\}$, onde d denota um disco, dt um disco ou uma torre e i o instante de tempo a que se refere a variável. A variável $\text{sobre}(d, dt, i)$ indica que um determinado disco se encontra directamente por cima de outro disco ou torre no instante i, enquanto a variável $\text{livre}(dt, i)$ indica se um determinado disco ou torre tem algum disco por cima de si no instante i. Por outro lado, a variável $\text{move}(d, dt, dt', i)$ indica que um determinado disco se move *de cima* de um disco ou torre *para cima* de outro disco ou torre, num dado instante i. Uma vez que apenas

[5]O *movimento válido* entre discos de duas torres corresponde a mover o disco mais pequeno para cima do disco maior. Se apenas uma das torres contiver discos, o disco do topo é movido directamente para a outra torre.

temos variáveis booleanas, para codificar as variáveis do tipo move precisaríamos de um número elevado de variáveis. De modo a resolver este problema podemos definir o operador move da seguinte forma [17]:

$$\text{move}(d, dt, dt', i) = \text{obj}(d, i) \wedge \text{origem}(dt, i) \wedge \text{dest}(dt', i).$$

A variável $\text{move}(d, dt, dt', i)$ corresponde a mover um disco d no instante i ($\text{obj}(d, i)$) de uma origem dt ($\text{origem}(dt, i)$) para um destino dt' ($\text{dest}(dt', i)$). A origem corresponde ao disco ou torre que se encontra imediatamente abaixo do disco a ser movido. Analogamente, o destino corresponde ao disco ou torre no qual o disco a ser movido será colocado.

Por exemplo, $\text{sobre}(d_1, d_2, 3)$ denota que o disco d_1 se encontra por cima do disco d_2 no instante 3. Se esta variável tomar o valor 1 é porque esta situação é verdadeira, caso contrário, é porque é falsa. $\text{livre}(t_2, 4)$ denota que não existe nenhum disco por cima da torre t_2 no instante 4. $\text{move}(d_1, d_2, t_3, 1)$ denota que o disco d_1 se encontra por cima do disco d_2 e será movido para cima da torre t_3 no instante 1.

Analisando as variáveis deste problema podemos verificar que, para codificar problemas de planeamento, é necessária a noção de tempo, ou seja, é necessário associar o valor de cada variável a cada instante de tempo i. Esta é a maior diferença entre a codificação do problema das Torres de Hanói e do Sudoku.

As acções são caracterizadas pelas variáveis $\text{move}(d, dt, dt', i)$. Para que uma acção ocorra é necessário que se verifiquem as pré-condições definidas através do conjunto

$$\{\text{livre}(d, i), \text{livre}(dt', i), \text{sobre}(d, dt, i)\}.$$

Adicionalmente, após ocorrer uma acção, esta terá as implicações definidas através do conjunto

$$\{\text{sobre}(d, dt', i+1), \text{livre}(dt, i+1), \neg\,\text{sobre}(d, dt, i+1), \neg\,\text{livre}(dt', i+1)\}.$$

Sempre que uma acção ocorre é necessário garantir que não existe nenhum disco em cima do disco, que está a ser movido ($\text{livre}(d, i)$) e que não existe nenhum disco em cima do disco/torre destino ($\text{livre}(dt', i)$). Adicionalmente, é necessário garantir que o disco que está a ser movido está em cima da origem ($\text{sobre}(d, dt, i)$). Se a acção ocorrer, então é necessário garantir que: o disco que foi movido estará em cima do destino no instante $i + 1$ ($\text{sobre}(d, dt', i + 1)$); a origem do movimento não tem nenhum disco por cima dela no instante $i+1$ ($\text{livre}(dt, i+1)$); o disco movido já não se encontra na origem do movimento no instante $i + 1$ ($\neg\,\text{sobre}(d, dt, i + 1)$); e o destino do movimento já não se encontra livre no instante $i + 1$ ($\neg\,\text{livre}(dt', i + 1)$).

Por exemplo, $\text{move}(d_1, d_2, t_3, 1)$ denota que o disco d_1 será movido de cima do disco d_2 para cima da torre t_3, no instante 1. Para que esta acção seja realizada é necessário que se verifiquem as seguintes pré-condições: $\text{livre}(d_1, 1)$, $\text{livre}(t_3, 1)$ e $\text{sobre}(d_1, d_2, 1)$. Se a acção for realizada, então isso terá as seguintes implicações: $\text{sobre}(d_1, t_3, 2)$, $\text{livre}(d_2, 2)$, $\neg\,\text{sobre}(d_1, d_2, 2)$ e $\neg\,\text{livre}(t_3, 2)$.

O *estado inicial* das Torres de Hanói é definido através do conjunto

$$\{\,\text{sobre}(d_1, d_2, 0), \ldots, \text{sobre}(d_{n-1}, d_n, 0), \text{sobre}(d_n, t_1, 0),$$
$$\text{livre}(d_1, 0), \text{livre}(t_2, 0), \text{livre}(t_3, 0)\}.$$

Todos os elementos que não estão neste conjunto são inicializados com o valor falso. Esta inicialização garante que todos os discos estão na torre 1 no instante 0, empilhados por ordem crescente de tamanho. Além disso, garante que não existe nenhum disco em cima de d_1, t_2 e t_3.

O *estado final* é definido através do conjunto

$$\{\text{sobre}(d_1, d_2, 2^n - 1), \ldots, \text{sobre}(d_{n-1}, d_n, 2^n - 1), \text{sobre}(d_n, t_3, 2^n - 1)\}.$$

Todos os restantes elementos que não estão neste conjunto não têm uma atribuição definida. O estado final garante que todos os discos estão na torre 3 no instante $2^n - 1$, empilhados por ordem decrescente de tamanho.

Poderemos fazer a passagem da linguagem STRIPS para SAT. Em SAT temos o conjunto de variáveis booleanas e um conjunto de cláusulas. Com base no modelo de STRIPS descrito anteriormente, definimos as nossas variáveis como

$$\{\text{sobre}(d, dt, i), \text{livre}(dt, i),\ obj(d, i), \text{origem}(dt, i), \text{dest}(dt', i)\}.$$

As cláusulas codificam as restrições seguintes, dadas pelo modelo STRIPS: exactamente um disco pode ser movido em cada passo; em cada passo existe exactamente um movimento (o que implica que exista exactamente um par *origem/dest*); não existem movimentos que movam um disco para a posição onde se encontrava no instante anterior (isto implicaria que nenhum movimento seria feito na prática); para que um movimento seja realizado é necessário que as pré-condições sejam satisfeitas; após realizar um movimento as suas implicações serão forçadas; nenhum disco pode ser movido para cima de discos mais pequenos. Adicionalmente, o estado inicial é válido no instante 0 e o estado final é válido no instante $2^n - 1$. Finalmente, cláusulas adicionais são necessárias para preservar o que não é afectado pelos movimentos.

Por limitações de espaço, não apresentamos a representação em cláusulas de todas as restrições acima referidas. No entanto, todos os pontos podem ser facilmente representados em cláusulas. Por exemplo, as restrições que codificam que *exactamente um* disco pode ser movido em cada instante são dadas pelas cláusulas que codificam que: *pelo menos um* disco é movido em cada instante; e *no máximo um* disco é movido em cada instante.

Por exemplo, consideremos o problema das Torres de Hanói com três discos e o instante de tempo 5. A seguinte cláusula garante que pelo menos um disco é movido no instante 5:

$$(\text{obj}(d_1, 5) \vee \text{obj}(d_2, 5) \vee \text{obj}(d_3, 5)).$$

As seguintes cláusulas garantem que no máximo um disco é movido no instante 5:

$$(\neg\,\text{obj}(d_1, 5) \vee \neg\,\text{obj}(d_2, 5)), (\neg\,\text{obj}(d_1, 5) \vee \neg\,\text{obj}(d_3, 5)), (\neg\,\text{obj}(d_2, 5) \vee \neg\,\text{obj}(d_3, 5)).$$

A codificação do problema pode ser melhorada de diversas maneiras. Por exemplo, a acção $\text{move}(d, dt, dt', i)$ pode ser reformulada para apenas considerar movimentos de discos entre torres [29]. Adicionalmente, outras propriedades do problema podem ser consideradas para tornar a modelação mais compacta [27]. Uma comparação entre as diferentes codificações é apresentada em [26].

Para modelar o problema das Torres de Hanói foi necessária a noção de tempo. Para um problema com n discos, são necessários $2^n - 1$ passos para transportar todos os discos da torre 1 para a torre 3. No entanto, se não soubéssemos o número de passos necessários estaríamos perante um problema de optimização, onde o objectivo seria transportar todos os discos da torre 1 para a torre 3 no menor número de passos possível. De facto, é possível estender os algoritmos de satisfação booleana para resolver problemas de optimização booleana. Na próxima secção, iremos mostrar como codificar e resolver problemas de optimização booleana.

5. Aplicação 3: Inferência de Haplótipos

O problema da inferência de haplótipos surge no âmbito da biologia computacional. É um assunto de elevada importância, pois tem implicações na área da medicina, podendo certos haplótipos estar relacionados com doenças ou com a possível resposta a medicamentos.

	genótipo g	T/T	A/G	C/C	C/T	G/G	A/A	C/T
Solução A	haplótipo h^a	T	A	C	C	G	A	C
	haplótipo h^b	T	G	C	T	G	A	T
Solução B	haplótipo h^a	T	A	C	C	G	A	T
	haplótipo h^b	T	G	C	T	G	A	C
Solução C	haplótipo h^a	T	A	C	T	G	A	C
	haplótipo h^b	T	G	C	C	G	A	T
Solução D	haplótipo h^a	T	A	C	T	G	A	T
	haplótipo h^b	T	G	C	C	G	A	C

FIGURA 4: Exemplo do problema da inferência de haplótipos.

Sucintamente, o ADN (ácido desoxirribonucleico) é uma molécula constituída por nucleótidos—adenina (A), citosina (C), guanina (G) e timina (T)—e constitui a base genética dos organismos vivos. O ADN é organizado em cromossomas. Em organismos diplóides, como é o caso dos seres humanos, os cromossomas estão organizados em pares, sendo um cromossoma do par herdado da mãe, e o outro cromossoma herdado do pai.

A informação contida no ADN é muito semelhante para todos os seres humanos. No entanto, existem algumas excepções, derivadas, por exemplo, de mutações em algumas posições específicas do ADN. Estas posições são chamadas de SNPs (*single nucleotide polymorphism*, *polimorfismos de um único nucleótido*) se o nucleótido menos comum ocorrer em pelo menos 1% da população. Por exemplo, a sequência de ADN CCTAAG pode sofrer uma mutação na quarta posição transformando-se em CCTGAG; a posição que mutou de A para G será chamada de SNP quando 1% da população tiver o nucleótido G. A larga maioria dos SNPs tem apenas dois valores possíveis, chamados *alelos*.

Um *haplótipo* corresponde a uma sequência de SNPs próximos num único cromossoma. Determinar experimentalmente os haplótipos é um processo demorado e dispendioso e, por isso, normalmente não são determinados. Ao invés, são obtidos os *genótipos*, que correspondem à informação conjunta do cromossoma herdado da mãe e do cromossoma herdado do pai. Dado o genótipo de um indivíduo, o problema da inferência de haplótipos consiste em determinar exactamente que informação foi herdada de cada progenitor, isto é, o par de haplótipos que deu origem a um dado genótipo.

A Figura 4 apresenta um exemplo do problema da inferência de haplótipos. Dado o genótipo g = [T/T A/G C/C C/T G/G A/A C/T], o objectivo do problema é escolher o par de haplótipos, de entre quatro soluções possíveis A–D, que originou o genótipo g. Uma posição heterozigótica de um genótipo é uma posição para a qual o nucleótido herdado da mãe é diferente do nucleótido herdado do pai. O genótipo g tem três posições heterozigóticas. Em geral, se o genótipo tem n posições heterozigóticas, o número de possíveis soluções é 2^{n-1}, pois o par da solução é não ordenado, isto é, consideramos $(h^a, h^b) = (h^b, h^a)$.

Uma das abordagens para este problema é a inferência de haplótipos por parcimónia pura [11, 9]: dado um conjunto de genótipos, encontrar um conjunto de haplótipos que *explica* o conjunto de genótipos e usa o *menor número* possível de haplótipos distintos.

No que se segue, n representa o número de genótipos, m representa o número de posições (SNPs), e os índices i e j são tais que $1 \leq i \leq n$ e $1 \leq j \leq m$.

Representaremos cada haplótipo h como um vector binário em que 0 representa o alelo mais frequente e 1 representa o alelo menos comum. Cada genótipo g_i será representado como um vector de elementos $g_{ij} \in \{0, 1, 2\}$. Se $g_{ij} = 0$, então o indivíduo herdou um alelo 0 de cada um dos pais; se $g_{ij} = 1$, o indivíduo herdou o alelo 1 de cada um dos pais; e se $g_{ij} = 2$, então o indivíduo herdou o 0 de um dos progenitores e 1 do outro progenitor.

Diz-se que um genótipo g_i é *explicado* por um par de haplótipos h_{2i-1} e h_{2i} se:

$$g_{ij} = \begin{cases} 0, & \text{se } h_{2i-1,\,j} = h_{2i,\,j} = 0, \\ 1, & \text{se } h_{2i-1,\,j} = h_{2i,\,j} = 1, \\ 2, & \text{se } h_{2i-1,\,j} \neq h_{2i,\,j}. \end{cases}$$

Por exemplo, o genótipo [2 0 0 2 2 1] é explicado pelos haplótipos [0 0 0 1 0 1] e [1 0 0 0 1 1].

O *problema da inferência da haplótipos por parcimónia pura* (HIPP, *haplotype inference by pure parsimony*) é definido da forma que se segue. Considere-se uma matriz $G_{n\times m} = [g_{ij}]_{i,j=1}^{n,m}$, tal que cada $g_{ij} \in \{0, 1, 2\}$. O problema HIPP consiste em encontrar uma matriz binária (i.e., cada entrada é 0 ou 1) $H_{2n\times m} = [h_{ij}]_{i,j=1}^{2n,m}$, tal que g_i seja explicado por h_{2i-1} e h_{2i}, e seja minimizado o número de linhas distintas da matriz H. Cada linha da matriz G representa um genótipo e cada linha da matriz H representa um haplótipo.

Por exemplo, considere-se a matriz de genótipos G:

$$G = \begin{bmatrix} 0 & 2 & 2 \\ 2 & 2 & 1 \\ 2 & 2 & 2 \end{bmatrix}.$$

Existem soluções usando seis haplótipos distintos, como por exemplo a apresentada na matriz H_1 (H_1 tem todas as linhas distintas). No entanto, a solução HIPP requer apenas quatro haplótipos distintos e pode ser vista na matriz H_2 (H_2 tem apenas quatro tipos de linhas diferentes):

$$H_1 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}, \qquad H_2 = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix}.$$

Uma possível formulação [2] para este problema é apresentada na Tabela 2 e descrita como se segue.

O conjunto das variáveis é dado por $\{t_{ij}, x_{ik}, u_i : \text{com } 1 \leq i, k \leq 2n \text{ e } 1 \leq j \leq m\}$. Associa-se uma variável t_{ij} com cada entrada da matriz h_{ij}, de modo que $t_{ij} = h_{ij}$ (note-se que a matriz H é binária). Para além disso, por cada duas linhas da matriz, h_i e h_k, associa-se uma variável x_{ik} que tomará valor 1 quando as linhas h_i e h_k forem diferentes e tomará valor 0 caso contrário, isto é,

$$x_{ik} = \begin{cases} 1, & \text{se } h_i \neq h_k, \\ 0, & \text{se } h_i = h_k. \end{cases}$$

Terceiro, considerem-se as variáveis u_i para cada linha h_i da matriz H, de forma a determinar se essa linha é diferente de todas as anteriores, isto é,

$$u_i = \begin{cases} 1, & \text{se para todo o } k < i,\ h_k \neq h_i, \\ 0, & \text{se existe } k < i \text{ tal que } h_k = h_i. \end{cases}$$

TABELA 2: Formulação do problema da inferência de haplótipos.

minimizar $\sum_{i=1}^{2n} u_i$		(1)
sujeito a:		
$(\neg t_{2i-1,j} \wedge \neg t_{2i,j})$, se $g_{ij} = 0$	$\forall_{i:1\leq i\leq n}$, $\forall_{j:1\leq j\leq m}$	(2)
$(t_{2i-1,j} \wedge t_{2i,j})$, se $g_{ij} = 1$	$\forall_{i:1\leq i\leq n}$, $\forall_{j:1\leq j\leq m}$	(3)
$((t_{2i-1,j} \vee t_{2i,j}) \wedge (\neg t_{2i-1,j} \vee \neg t_{2i,j}))$, se $g_{ij} = 2$	$\forall_{i:1\leq i\leq n}, \forall_{j:1\leq j\leq m}$	(4)
$((t_{kj} \wedge \neg t_{ij}) \vee (\neg t_{kj} \wedge t_{ij})) \Rightarrow x_{ik}$	$\forall_{i,k:1\leq k<i\leq 2n}$, $\forall_{j:1\leq j\leq m}$	(5)
$(\bigwedge_{k=1}^{i-1} x_{ik}) \Rightarrow u_i$	$\forall_{i:1\leq i\leq 2n}$	(6)

A *função de optimização* deve minimizar o número de linhas distintas da matriz H, ou seja, minimizar a soma das variáveis u_i,

$$\min \sum_{i=0}^{2n} u_i. \tag{1}$$

As *restrições* são definidas como se segue. De forma a garantir que cada genótipo é explicado por um par de haplótipos é necessário que:

$$(\neg t_{2i-1,j} \wedge \neg t_{2i,j}), \text{ se } g_{ij} = 0, \tag{2}$$

$$(t_{2i-1,j} \wedge t_{2i,j}), \text{ se } g_{ij} = 1, \tag{3}$$

$$((t_{2i-1,j} \vee t_{2i,j}) \wedge (\neg t_{2i-1,j} \vee \neg t_{2i,j})), \text{ se } g_{ij} = 2. \tag{4}$$

Para garantir que dois haplótipos, h_i e h_k, são iguais, é necessário que, para todas as posições j, sejam iguais $h_{ij} = h_{kj}$. A restrição que garante este facto é dada pela fórmula

$$((t_{kj} \wedge \neg t_{ij}) \vee (\neg t_{kj} \wedge t_{ij})) \Rightarrow x_{ik}. \tag{5}$$

Finalmente, é necessário garantir que, se h_i é diferente de todas as linhas anteriores, então $u_i = 1$. Este facto pode ser descrito pela equação:

$$\Big(\bigwedge_{k=1}^{i-1} x_{ik}\Big) \Rightarrow u_i. \tag{6}$$

Este modelo pode ser melhorado usando diversas técnicas, entre as quais, cálculo de limites inferiores/superiores no número óptimo de haplótipos, quebra de simetrias, restrições de cardinalidade [10].

6. MAS QUÃO DIFÍCIL É RESOLVER SAT?

Informalmente, a teoria da complexidade é responsável por classificar os problemas quanto à sua dificuldade. Os problemas ditos "fáceis", encontram-se na classe P. Por contraponto, os problemas "difíceis" são não–P. Para os problemas da classe de complexidade P, existem algoritmos que os resolvem em tempo polinomial (daí classe P): quer isto dizer que, o tempo de computação é limitado superiormente por alguma função polinomial no tamanho da instância do problema. Para um problema da classe não–P, não existe qualquer algoritmo que o resolva e cujo tempo de computação seja limitado superiormente por um polinómio.

Por outro lado, existe uma classe de problemas, a classe NP (polinomial não-determinístico) definida como a classe dos problemas que, embora não sejam necessariamente resolúveis em tempo polinomial, a verificação de uma solução pode ser feita em tempo polinomial. Isto é, resolver o problema pode ser "difícil" mas verificar a validade de uma solução é "fácil".

Apesar da importância que a temática tem e do esforço que matemáticos e cientistas da computação têm vindo a desenvolver, ainda não está provado que P seja diferente de NP, embora na comunidade científica essa seja a conjectura dominante. Na verdade, a validade de igualdade $P = NP$ teria sérias consequências práticas. Por exemplo, a criptografia usada em várias operações, como em transacções comerciais e financeiras através da Internet, baseia-se no facto de certos problemas serem "difíceis" e tornar-se-ia insegura.

O problema é tão interessante que existe um prémio monetário a ser atribuído a quem prove que P difere de NP (ou o contrário). No ano 2000, em Paris, foram anunciados os Problemas do Milénio, uma lista de sete problemas em aberto premiados, mediante resolução, com um milhão de dólares. Entre os sete, encontra-se o problema P vs. NP. De notar, por curiosidade, que a Conjectura de Poincaré, outro problema da lista, foi, entretanto, resolvido, por Grigoriy Perelman em 2002–2003. Estas e outras informações podem ser lidas em [4].

Voltando a SAT, apesar da sua formulação aparentemente simples, SAT é um problema "difícil". SAT é um problema NP e, para além disso, SAT foi o primeiro problema a ser provado NP–completo. Um problema NP–completo tem a particularidade de que todos os problemas NP se podem reduzir (polinomialmente)[6] a ele, ou, de outra forma: se conseguíssemos provar que SAT está em P, então todos os problemas NP estariam em P (isto é, provar-se-ia $P = NP$).

Existe um diversificado conjunto de problemas de decisão e optimização combinatória, em variadas áreas, que podem ser modelados em satisfação booleana. De facto, todos os problemas de decisão em domínios finitos podem ser modelados em SAT. Em particular, para os amantes de futebol, poderá ser interessante a leitura do capítulo de divulgação sobre a complexidade de algoritmos relativos ao tema [3] ou o artigo científico [18].

Por todo o foco que o problema SAT assume e o auxílio que presta a outras áreas, o desenvolvimento de algoritmos eficientes para resolução de SAT e suas extensões é muito relevante e, por isso, regularmente são feitas competições de algoritmos de satisfação e optimização booleanas (mais informação pode ser lida em [20] e [24]).

Bibliografia

Para aprofundar ou esclarecer conteúdos sobre o problema SAT referimos o livro *Handbook of Satisfiability* [1]. O sítio `http://www.satlive.org` [21] mantém informação actualizada sobre o trabalho da comunidade de SAT. Para o leitor mais interessado em usar um SAT *solver* para resolver um modelo, sugerimos o programa MiniSat [7].

Mais detalhes sobre o problema P vs. NP podem ser encontrados no artigo $P =^{?} NP$ desta mesma colectânea [5].

Para o leitor interessado em aprofundar a aplicação de SAT ao Sudoku, sugerimos a leitura do artigo [22]. A modelação do problema das Torres de Hanói em SAT é descrita em [26]. Para estudar o problema da inferência de haplótipos por parcimónia pura sugere-se o artigo [9].

Referências

[1] A. Biere, M. Heule, H. van Maaren, e T. Walsh (editores), *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications, vol. 185, IOS Press, 2009.

[6]Uma *redução* é uma transformação de um problema em outro. Um problema Z é *redutível polinomialmente* a um problema X (o que se representa por $Z \leq_p X$) se existir uma função, $f : Z \to X$, calculável em tempo polinomial tal que, para qualquer instância z do problema Z ($z \in Z$), se verifique $Z(z) = X(f(z))$. Desta forma, resolver o problema Z não pode ser mais difícil do que resolver o problema X.

[2] D. Brown e I. Harrower, *Integer Programming Approaches to Haplotype Inference by Pure Parsimony*, ieee/acm Transactions on Computational Biology and Bioinformatics, vol. 3, no. 2, 2006, pp. 141–154.
[3] J. Buescu, *Da Falsificação de Euros aos Pequenos Mundos*, Capítulo: Teorema: o Futebol é Complexo, Gradiva, 2003.
[4] Clay Mathematics Institute, *The Millennium Prize Problems*, consultado em Agosto de 2012. `http://www.claymath.org/millennium`
[5] Bruno Conchinha, $P =^{?} NP$,, neste volume.
[6] N. Crato, *A Matemática das Coisas*, Capítulo: O Algoritmo do Jantar de Anos, Gradiva, 2008.
[7] N. Eén e N. Sörensson, *MiniSAT* sat *Solver*. `http://www.minisat.se` [consultado em Ago. 2012]
[8] R. Fikes e N. J. Nilsson, strips*: A New Approach to the Application of Theorem Proving to Problem Solving*, Artificial Intelligence, vol. 2, no. 3/4, 1971, pp. 189–208.
[9] A. Graça, I. Lynce, J. Marques-Silva, e A. Oliveira, *Haplotype Inference by Pure Parsimony: a Survey*, Journal of Computational Biology, vol. 17, no. 8, 2010, pp. 969–992.
[10] A. Graça, J. Marques-Silva, I. Lynce, e A. Oliveira, *Haplotype Inference with Pseudo-Boolean Optimization*, Annals of Operations Research, vol. 184, no. 11, 2011, pp. 137–162.
[11] D. Gusfield, *Haplotype Inference by Pure Parsimony*, Symposium on Combinatorial Pattern Matching (Morelia, Michocán, Mexico), Lecture Notes in Computer Science, vol. 2676, Springer, 2003, pp. 144–155.
[12] P. L. Hammer e S. Rudeanu, *Boolean Methods in Operations Research and Related Areas*, Springer Verlag, 1968.
[13] M. Henz e H. Truong *SudokuSat – A Tool for Analyzing Difficult Sudoku Puzzles*, Tools and Applications with Artificial Intelligence (eds. C. Koutsojannis e S. Sirmakessis), Studies in Computational Intelligence, vol. 166, Springer, 2009, pp. 25–35.
[14] A. Inkala, *AI Escargot – The Most Difficult Sudoku Puzzle*, Lulu, 2007.
[15] Life International. *Who Wants the Zebra?*, Life International Magazine, vol. 17, no. 95, 1962.
[16] H. Kautz, D. McAllester, e B. Selman, *Encoding Plans in Propositional Logic*, International Conference on the Principle of Knowledge Representation and Reasoning (Cambridge, Massachusetts, USA), Morgan Kaufmann, 1996, pp. 374–384.
[17] H. Kautz e B. Selman, *Planning as Satisfiability*, European Conference on Artificial Intelligence (Vienna, Austria), John Wiley & Sons, 1992, pp. 359–363.
[18] W. Kern e D. Paulusma, *The New FIFA Rules are Hard: Complexity Aspects of Sports Competitions*, Discrete Applied Mathematics, vol. 108, 2001, pp. 317–323.
[19] G. Kwon e H. Jain, *Optimized* cnf *Encoding for Sudoku Puzzles*, International Conference on Logic for Programming Artificial Intelligence and Reasoning (Phnom Penh, Cambodia), 2006.
[20] D. Le Berre, *SAT Competitions*. `http://www.satcompetition.org` [consultado em Ago. 2012]
[21] D. Le Berre, *Up-to-date links for the Satisfiability Problem*. `http://www.satlive.org` [consultado em Ago. 2012]
[22] I. Lynce e J. Ouaknine, *Sudoku as a* sat *Problem*, International Symposium on Artificial Intelligence and Mathematics (Fort Lauderdale, Florida, USA), 2006.
[23] E. Lucas, *Récréations Mathématiques*, Capítulo: La Tour d'Hanoï, Gauthier-Villars, Paris, 1883.
[24] V. Manquinho e O. Roussel, *Pseudo-Boolean Competition 2011*. `http://www.cril.univ-artois.fr/PB11` [consultado em Ago. 2012]
[25] J. Marques-Silva, I. Lynce, e S. Malik, *Conflict-Driven Clause Learning* sat *Solvers*, Handbook of Satisfiability, ios Press, 2009.
[26] R. Martins, *O Impacto da Modelação na Resolução de Problemas de Satisfação Proposicional*, tese de mestrado, Universidade Técnica de Lisboa, 2007.
[27] R. Martins e I. Lynce, *Effective* cnf *Encodings of the Towers of Hanoi*, International Conference on Logic for Programming Artificial Intelligence and Reasoning (Doha, Qatar), 2008.
[28] G. McGuire, B. Tugemann e G. Civario, *There is no 16-Clue Sudoku: Solving the Sudoku Minimum Number of Clues Problem*, 2012. `arXiv:1201.0749[cs.DS]`
[29] S. Prestwich, *Variable Dependency in Local Search: Prevention Is Better Than Cure*, International Conference on Theory and Applications of Satisfiability Testing (Lisbon, Portugal), Lecture Notes in Computer Science, vol. 4501, Springer, 2007, pp. 107–120.
[30] G. Royle, *Minimum Sudoku*. `http://mapleta.maths.uwa.edu.au/~gordon/sudokumin.php` [consultado em Ago. 2012]

[31] D. Wood, *The Towers of Brahma and Hanoi Revisited*, Journal of Recreational Mathematics, vol. 14, no. 1, 1981–1982, pp. 17–24.