Modelos matemáticos e problemas de optimização combinatória*

Iouliia Skliarova, António B. Ferrari

Resumo – **Este artigo apresenta alguns resultados da análise de modelos matemáticos, tais como conjuntos, grafos, matrizes discretas e funções booleanas, utilizados para a especificação e a resolução de problemas de optimização combinatória. É demonstrado que estes modelos são mutuamente convertíveis uns nos outros. São apresentados também exemplos de problemas combinatórios típicos que surgem na área de projecto de dispositivos digitais. A maioria destes problemas podem ser resolvidos com a ajuda dos modelos referidos, através da aplicação de métodos combinatórios. Por fim, analisam-se diferentes possibilidades da implementação de algoritmos combinatórios.**

Abstract – **The paper presents the results of the analysis of different models that are used in problems of combinatorial optimization, such as graphs, sets, discrete matrices, and Boolean functions. It is shown that these models can be mutually converted one into another. Many examples of typical combinatorial tasks, which appear at different steps of the design of digital devices, are considered. The majority of these tasks can be solved with the aid of the presented models and the respective combinatorial methods. Finally, different ways of implementation of combinatorial algorithms are analyzed.**

I. INTRODUCÃO

Os problemas de optimização combinatória surgem em várias áreas práticas tais como desenvolvimento de circuitos digitais, diagnóstico técnico, cartografia, reconhecimento de padrões, planeamento de circulação de transporte, etc. Uma das características distintivas destes problemas é que a sua resolução está ligada com o exame de elementos de um conjunto finito especificado pelos dados iniciais. Os problemas de optimização possuem um conjunto de soluções $S = \{s_1, ..., s_n\}$ no qual é definida uma função de custo $c(s_i)$. Na resolução dum problema de optimização é preciso encontrar uma solução com a função de custo mínima. Os algoritmos desenvolvidos e utilizados para a resolução destes problemas são avaliados com os 2 critérios seguintes: a qualidade da solução e a sua complexidade. De acordo com o critério da complexidade os algoritmos são divididos normalmente em duas classes: polinomiais e exponenciais. Os algoritmos exactos, que garantem a obtenção da solução exacta do problema, têm frequentemente complexidade muito elevada, o que impede a sua aplicação na prática. Por outro lado os algoritmos aproximados nem sempre levam à solução exacta mas asseguram uma aproximação bastante boa a esta.

A resolução de problemas combinatórios é bastante díficil especialmente quando é necessário encontrar uma solução óptima. Contudo a complexidade dos dados iniciais em muitos problemas práticos é limitada, o que poderia levar a concluir que não é preciso desenvolver algoritmos eficientes teoricamente, i.e. basta que os algoritmos sejam eficientes na prática. Na resolução de muitos problemas combinatórios os requisitos de qualidade da solução podem ser diminuidos, i.e. frequentemente a obtenção da solução óptima não é necessária, portanto os métodos exactos podem ser substituidos pelos aproximados que são normalmente bastante mais rápidos. Em muitas situações práticas é possível aplicar métodos de redução que, diminuindo a dimensão do problema em questão, reduzem essencialmente o número de variantes a rever [1]. O desenvolvimento de métodos deste tipo com base na investigação de características de problemas concretos é a área essencial da aplicação da análise combinatória.

Entre vários modelos matemáticos utilizados no desenvolvimento de algoritmos combinatórios destacamos conjuntos, grafos, matrizes e funções lógicas. Notamos que um modelo pode ser transformado noutro.

II. EXEMPLOS DE PROBLEMAS COMBINATÓRIOS

Analisemos alguns exemplos de problemas combinatórios típicos que surgem frequentemente em projecto de dispositivos digitais. Classificamos os problemas de acordo com os modelos matemáticos usados normalmente para a sua especificação. Vários algoritmos de resolução destes problemas podem ser encontrados nas referências indicadas.

* Trabalho financiado com a bolsa da FCT-PRAXIS XXI/BD/21353/99

A. Conjuntos

A.1. Obtenção de partição mínima

O problema é formulado de maneira seguinte. Num conjunto U é necessário encontrar uma partição π que é composta por um número mínimo de blocos. Cada bloco U^i desta partição deve corresponder a determinados requisitos e

$$\pi = \{U^1, ..., U^n\}, \quad U^i \subseteq U, \quad U^i \neq 0, \quad U^i \cap U^j = 0 \quad \forall i, j, \; i \neq j, \quad \bigcup_{i=1}^{n} U^i = U$$

B. Grafos

B.1. Coloração de grafos

Este problema é um dos mais conhecidos problemas combinatórios. Normalmente exige-se colorir um grafo usando um número mínimo de cores atribuindo a todos os vértices adjacentes cores diferentes [2, 3]. A fig. 1 ilustra a coloração mínima de um grafo com 3 cores.

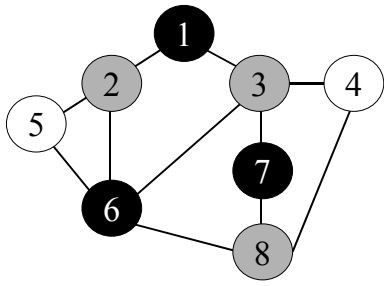


Fig. 1 – Coloração mínima do grafo

B.2. Encontro de caminhos

Problema de encontro de caminhos num grafo também é muito conhecido [3, 4]. Este é formulado de modo seguinte. Há um grafo orientado e interpretado G(E,V,W), um vertice inicial v_b e um vertice final v_e. È preciso encontrar o caminho mais curto ou mais comprido (em termos de pesos dos arcos) do vertice v_b para o v_e. Na fig. 2 é apresentado um grafo interpretado, se $v_b = v_2$ e $v_e = v_5$ então o caminho mais curto é o v_2-v_3-v_5, e o mais comprido é o v_2-v_5.

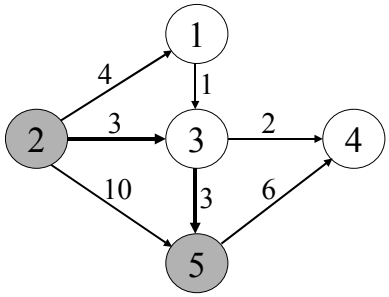


Fig. 2 – O caminho mais curto de v_2 a v_5 é v_2-v_3-v_5

B.3. Partição em cliques

Uma clique dum grafo é um subgrafo completo. Um grafo diz-se decomposto em cliques quando o conjunto dos seus vertices fica dividido em subconjuntos que não se intersectam, cada um dos quais representa uma clique [5]. O problema mais frequente é a procura da partição mínima. O grafo G(V, E) na fig. 3 pode ser dividido em duas cliques $\{v_1, v_2, v_3\}$ $\{v_4\}$.

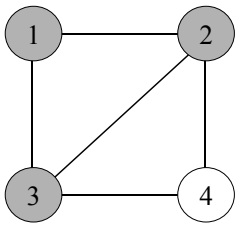


Fig. 3 – Partição em cliques

B.4. Corte

Este problema consiste em cortar um grafo em vários subgrafos de tal maneira que o número de vértices em cada um dos subgrafos não exceda *k* e o peso total de arestas eliminadas fique minimizado [6]. Na fig. 4 é apresentado um exemplo de corte de um grafo com $k = 2$.

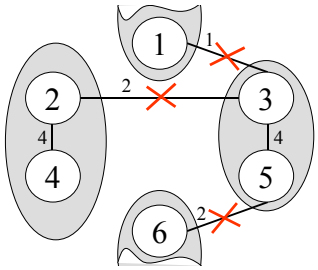


Fig. 4 – Corte do grafo com $k = 2$

C. Matrizes

Aqui consideramos apenas matrizes booleanas e ternárias, utilizadas para a especificação de relações binárias e ternárias arbitrárias.

C.1. Encontro de menores

Muitos problemas estão relacionados com o encontro de um menor (subconjunto de linhas ou colunas) com determinadas características numa matriz. Um dos mais conhecidos é o problema da obtenção da cobertura mínima, onde se procura descobrir um menor mínimo composto pelas colunas (linhas) que contenham em cada linha (coluna) pelo menos um elemento igual a 1 [7]. O problema de determinar o teste diagnóstico mínimo é um outro representante desta classe. Aqui pretende-se descobrir um menor mínimo, cujas linhas sejam todas diferentes (supõe-se que na matriz inicial estas também

são diferentes) [8]. Consideremos a matriz booleana **B**. Neste caso o vector booleano [01100] representa a cobertura mínima (composta pela segunda e terceira colunas da matriz **B**) e o vector [01011] representa o teste mínimo.

$$\mathbf{B} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

C.2. Relação de ortogonalidade

Um dos exemplos de problemas deste tipo é o seguinte: é necessário encontrar um vector que seja ortogonal a todas as linhas de uma dada matriz ou concluir que este não existe [7]. Relembramos que dois vectores estão em relação de ortogonalidade se pelo menos numa das suas componentes um vector tem o valor 0 e outro o valor 1.

C.3. Construção de uma nova matriz

Muitos problemas exigem a construção de uma nova matriz **C** que deve estar numa determinada relação com uma dada matriz **B**. Um exemplo desta classe de problemas é o da determinação da base disjuntiva mínima, onde é preciso construir a matriz **C** com o número mínimo de linhas de tal maneira que cada linha da matriz **B** seja igual à disjunção de várias linhas de **C** [1, 7]. Ilustramos este problema com as seguintes duas matrizes **B** e **C**.

$$\mathbf{B} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

C.4. Minimização

Quando uma matriz representa a DNF (*Disjunctive Normal Form*) de uma função (ver secção III) surgem frequentemente problemas de minimização da mesma. Por exemplo, para uma dada matriz ternária é preciso encontrar uma outra matriz equivalente com o número de linhas minimizado. Assim, a seguinte matriz **T** representa uma DNF inicial e a matriz **U** representa a DNF mais curta equivalente à inicial [7].

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & - \\ - & 0 & 1 & 1 \\ - & 0 & 1 & 0 \end{bmatrix} \qquad \mathbf{U} = \begin{bmatrix} 1 & 0 & - & - \\ 0 & 0 & 1 & - \end{bmatrix}$$

D. Funções booleanas

D.1. Minimização

Relembramos que qualquer função booleana pode ser representada em DNF, i.e. em forma de disjunção de um conjunto finito de conjunções elementares diferentes. O problema de obtenção de DNFs mais simples para uma dada função booleana é bastante complicado. Normalmente por critério de simplicidade toma-se o número total de argumentos (quanto à procura da DNF mínima) ou o número de conjunções elementares (quanto à procura da DNF mais curta) [7, 9]. Por exemplo, para a função $f = x_1 x_2 \vee \bar{x}_1 x_2 \vee \bar{x}_1 \bar{x}_2$ obtemos a seguinte DNF mais curta $f = \bar{x}_1 \vee x_2$. Note-se que o problema de encontro da DNF mais curta coincide com o da determinação da matriz ternária mínima (com o número de linhas mínimo) equivalente à dada.

D.2. Decomposição

Este problema consiste na representação de uma função booleana em forma de composição de várias funções booleanas cada uma com um menor número de argumentos [7].

D.3. Satisfabilidade

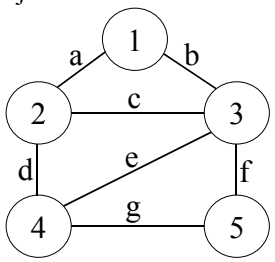
Este problema consiste na verificação da existência de valores de argumentos tais que tornem a função igual a 1 [9]. Tautologia é um problema complementar que verifica se a função é equivalente a 1, i.e. adquire o valor 1 independemente dos valores dos seus argumentos. Notemos que este problema coincide com o da determinação de um vector que é ortogonal a cada linha da matriz que representa a DNF da função (caso seja impossível encontrá-lo, pode-se concluir que a função é equivalente a 1).

III. TRANSFORMAÇÕES MÚTUAS DE MODELOS

Com a ajuda de matrizes lógicas pode-se formular muitos problemas combinatórios e construir algoritmos para a sua resolução. As matrizes possuem uma estrutura que as torna bastante convenientes para o armazenamento e processamento em computador. Portanto mostramos como especificar grafos e funções booleanas através das matrizes respectivas.

Um grafo G(V, E) pode ser representado em forma de uma matriz de adjacência ou de incidência. Uma matriz de adjacência **A** é uma matriz quadrada de dimensão |V|*|V|, onde cada elemento a_i^j é igual a 1 caso o vértice *i* seja adjacente ao vértice *j*, e é igual a 0 caso contrário. Uma matriz de incidência **I** é uma matriz com as dimensões |V|*|E|. Para um grafo não orientado cada elemento (i, j)

da matriz é igual a 1 caso a aresta *j* seja incidente ao vértice *i*, e é igual a 0 caso contrário. Para um grafo orientado um elemento em posição (i, j) é igual a 1 se o vértice *i* é a origem do arco *j*, é igual a -1 se o vértice *i* é o destino do arco *j*, e é igual a 0 caso contrário. Na fig. 5 é representado um grafo não orientado e as suas matrizes de incidência e de adjacência.



$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}\begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & g \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \end{array} \qquad \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}\begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

Fig. 5 – Grafo e as matrizes de incidência e de adjacência respectivas

As funções booleanas também podem ser representadas em forma de matrizes booleanas e ternárias. Quando uma função booleana está em DNF então as colunas da matriz correspondem aos argumentos da função e as linhas determinam as conjunções elementares. Por exemplo, para a função seguinte é bastante fácil construir as respectivas matrizes booleana e ternária:

$$x_1x_2x_3 \vee \bar{x}_1x_2x_3 \vee x_1\bar{x}_2x_3 \vee x_1\bar{x}_2\bar{x}_3 \vee x_1x_2\bar{x}_3 = x_1 \vee x_2x_3$$

$$\mathbf{B} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \qquad \mathbf{T} = \begin{bmatrix} 1 & - & - \\ - & 1 & 1 \end{bmatrix}$$

Notemos que a matriz booleana **B** pode ser obtida a partir da matriz ternária **T** através da substituição das componentes que possuem valor "-" com todas as combinações possíveis de 0 e 1, e da eliminação posterior de linhas iguais.

Um sistema de funções booleanas **y=f(x)** pode ser especificado com a ajuda de um par de matrizes **X** e **Y**. Se as funções booleanas estão em DNF então as colunas da matriz **X** determinam um sistema de intervalos juntamente com as conjunções elementares respectivas, e as linhas da matriz **Y** correspondem às DNFs, i.e. cada elemento y_i^j é igual a 1 quando a conjunção elementar *j* entra na DNF *i*.

O seguinte exemplo ilustra a representação de um sistema de DNFs com as matrizes **X** e **Y**:

$$y_1 = \bar{x}_1 \vee x_2x_3$$
$$y_2 = x_1\bar{x}_2\bar{x}_3 \vee x_2x_4$$
$$y_3 = \bar{x}_1 \vee \bar{x}_1\bar{x}_4 \vee x_2x_3$$

$$\mathbf{X} = \begin{bmatrix} 0 & - & 1 & - & 0 \\ - & 1 & 0 & 1 & - \\ - & 1 & 0 & - & - \\ - & - & - & 1 & 0 \end{bmatrix} \qquad \mathbf{Y} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

IV. APLICAÇÕES PRÁTICAS DE PROBLEMAS COMBINATÓRIOS

Muitos problemas práticos reduzem-se à resolução de vários problemas combinatórios típicos. Consideremos alguns exemplos destes problemas práticos destacando os que surgem na área do desenvolvimento de circuitos digitais.

É conhecido que uma das fases da síntese e optimização arquitectural consiste na distribuição de operações no tempo, i.e. na determinação do início da execução de cada operação com base na análise da sequência e interacção de todas as operações [9]. A possibilidade da resolução deste problema com dadas restrições temporais pode ser verificada aplicando o método da determinação do caminho mais comprido num grafo. Consideremos um grafo orientado e interpretado, cujos vértices representam as operações e cujos arcos correspondem às dependências entre as operações. O peso de cada arco é igual ao atraso da operação colocada na sua origem. Os arcos complementares especificam as restrições temporais e as restrições máximas $U_{ij}^{\max}$ definem o intervalo máximo de tempo que pode decorrer entre duas operações *i* e *j*. Sendo assim, para que o problema tenha solução, o caminho mais comprido (o caminho com o peso maior) neste grafo entre os vértices v_i e v_j deve ser $\leq U_{ij}^{\max}$.

Na fase seguinte da síntese arquitectural efectua-se a atribuição das operações aos recursos existentes. Um dos objectivos a atingir nesta fase consiste na diminuição da área do circuito permitindo que várias operações distribuidas no tempo utilizem o mesmo hardware [9]. Neste caso duas operações chamam-se compatíveis caso não executem simultâneamente e possam ser implementadas em recursos do mesmo tipo. Constroi-se o grafo de compatibilidade, cujos vértices correspondem às operações e cujas arestas ligam os pares de operações compatíveis. Sendo assim o problema da distribuição óptima de recursos reduz-se à partição do grafo num número mínimo de cliques.

Para a minimização do número de condições lógicas numa FSM (Máquina de Estados Finitos) usa-se a informação complementar sobre as suas alterações possíveis no processo de execução. Designemos por $A(x_i)$ o conjunto de estados cujas transições dependem da condição lógica x_i, i=1,..,L. Colocamos em correspondência a cada conjunto $A(x_i)$ um vector $\mathbf{g}_i = (g_{i1},...g_{iL})$, e representamos todos os vectores por uma matriz **G**, onde cada elemento g_{ij} é igual a 1 se em todos os estados de $A(x_i)$ $x_j = 1$, g_{ij} é igual a 0 se em todos os estados de $A(x_i)$ $x_j = 0$, e g_{ij} é igual a "-" se nos estados de $A(x_i)$ a condição x_j pode tomar tanto o valor 1 como o valor 0. Com base na matriz **G** construimos um grafo com os vértices correspondentes às condições lógicas. Os dois vértices x_i e x_j são ligados por uma aresta caso $g_{ij} \neq g_{ji}$. Depois de colorir o grafo com o número mínimo de cores podemos unir (de um modo especial [10]) aquelas condições lógicas que correspondem aos vértices da mesma cor.

Na fase da síntese lógica de circuitos efectua-se a minimização de funções booleanas. O objectivo é construir um sistema de funções que sendo equivalente ao inicial corresponde aos requisitos especificados (por exemplo, a complexidade mínima ou o atraso de propagação mínimo, etc.).

O método da determinação das partições mínimas de um conjunto aplica-se na fase de mapeamento de elementos lógicos em células de uma dada biblioteca. Por exemplo, temos um sistema de funções booleanas em DNF com L argumentos, N funções e B conjunções diferentes. Designamos com L^i, i=1, ..., B, o número de argumentos na conjunção *i*, então $L_{max} = \max_i L_i$. É preciso implementar este sistema de funções booleanas em PLAs(s,t,q) (PLA com s entradas, t saídas e q linhas intermédias) e L > s, N > t, B > q, L^{max} <= s. Consideremos o conjunto de todas as conjunções elementares diferentes $E=\{e_1,...,e_B\}$. Então o problema reduz-se ao encontro de partição mínima no conjunto E que para cada bloco E^u faz cumprir as seguintes restrições: o número de argumentos em E^u < s, o número de funções que dependem de E^u <= t e $|E^u| <$ q [11].

Frequentemente surge o problema de decomposição de uma FSM em várias sub-FSMs mais simples. Um dos objectivos que se coloca consiste em minimizar a quantidade de ligações entre sub-FSMs, o que é equivalente à minimização do número de transições entre os estados pertencentes a sub-FSMs diferentes [6, 12]. Para resolver este problema é preciso cortar o grafo da FSM inicial em subgrafos minimizando a quantidade de sub-FSMs e o número de arestas eliminadas.

Suponhamos que temos um conjunto de vectores booleanos e precisamos de construir um array AND (o mais simples possível) que implemente estes vectores. Pede-se também arranjar os vectores de entrada respectivos que serão transformados em dados vectores de saída. Notamos que a estrutura de um array AND pode ser especificada com a ajuda de uma matriz **B**, e a sua funcionalidade descreve-se com a equação $\mathbf{Y} = \overline{\mathbf{B} \times \mathbf{X}}$ [7]. Sendo assim, temos de encontrar as matrizes booleanas **B** (com o número mínimo de colunas *k*) e **X** (com o mesmo número de linhas *k*) que satisfazem a equação $\mathbf{Y} = \overline{\mathbf{B} \times \mathbf{X}}$. Substituimos a matriz **Y** com a sua negação **Y'** e analisamos a equação $\mathbf{Y'} = \mathbf{B} \times \mathbf{X}$. Neste caso cada coluna da matriz **Y'** é igual à disjunção de várias colunas da matriz **B** (a matriz **X** indica quais). Portanto o problema reduz-se à determinação da base disjuntiva mínima.

Às vezes surge o problema de determinação de causas a partir dos efeitos que estas provocam (por exemplo, em diagnóstico). Neste caso as causas abrangem os fenómenos cuja observação é difícil ou mesmo impossível. Por outro lado, os efeitos são directamente detectáveis. Contudo a detecção dos efeitos tem um custo associado portanto o seu número deve ser restringido. Sendo assim, surge o problema de determinar o subconjunto mínimo de efeitos cuja observação permite descobrir a respectiva causa. A relação entre os conjuntos de causas e efeitos pode ser representada em forma de uma matriz booleana **X** onde cada elemento x_i^j é igual a 1 se a causa *j* provoca o efeito *i*, e é igual a 0 caso contrário. Então o problema reduz-se a determinar o teste diagnóstico mínimo para a matriz **X** [7].

V. MÉTODOS DE RESOLUÇÃO DE PROBLEMAS COMBINATÓRIOS

Na sua forma mais simples a resolução de um problema combinatório consiste no exame de todas as variantes possíveis. Cada variante é examinada uma por uma e logo que se verifique que uma destas é a solução, o processo de pesquisa termina. Em algumas situações este método simples é o único que pode ser aplicado. Por exemplo, é preciso encontrar o número mínimo num conjunto de *n* números. É evidente a inevitabilidade de examinar os *n* elementos.

O conjunto entre cujos elementos se procura a solução é finito, portanto ao examinar exaustivamente os seus elementos podemos sempre encontrar a solução ou concluir que esta não existe. Portanto qualquer problema lógico pode ser resolvido em princípio, mas isto não significa que possa ser resolvido em tempo aceitável. Por exemplo, sabemos que a solução de um problema pode ser representada por uma função booleana desconhecida de 10 variáveis. Existem 2^{2^n} de funções booleanas diferentes de *n* argumentos, logo o método de revisão completa vai exigir o exame de $2^{2^{10}} = 2^{1024} > 10^{256}$ variantes!

Sendo assim o método de revisão completa só pode ser aplicado em situações muito simples. No caso geral devemos reduzir de alguma maneira a quantidade de

variantes a considerar utilizando informação complementar sobre o problema em causa.

Uma das técnicas gerais da resolução de problemas combinatórios consiste na construção de uma árvore de pesquisa. A raíz da árvore corresponde à situação inicial e os restantes vértices representam todas as situações que podem ser atingidas no processo de pesquisa. Os arcos correspondem aos passos do processo decisório. Na maioria dos casos a árvore é desconhecida inicialmente, e só se constroi durante o processo de pesquisa da solução. Ao atingir uma dada situação devemos determinar todas as variantes possíveis da continuação da pesquisa, i.e. todos os arcos de saída. É evidente que é necessário reduzir de algum modo o seu número para que a solução seja encontrada o mais depressa possível. Normalmente esta redução é baseada na eliminação dos passos que não vêm a alterar a situação actual, na rejeição das variantes já analisadas anteriormente e em parar de procurar aquelas soluções cujo custo no ponto actual já se tornou maior do que o custo de alguma das soluções já encontradas.

Este método simples sendo universal é pouco eficaz, i.e. só pode ser aplicado na prática caso os dados iniciais do problema sejam bastante simples. Uma das técnicas que aumentam significativamente a eficâcia deste método é a redução. Nesta técnica uma situação actual substitui-se por uma outra situação mais simples, o que permite reduzir a quantidade de cálculos necessários para analisar o conjunto de variantes resultantes desta situação. As maneiras concretas de redução são extremamente diversas. Por exemplo, é preciso encontrar a cobertura mínima de uma matriz booleana. Neste caso a matriz inicial pode ser diminuida através da eliminação de linhas dominantes a colunas dominadas (se procuramos a cobertura composta por colunas) [7], i.e. a simplificação da situação consiste na diminuição da matriz. Um exemplo clássico desta técnica é o da determinação das raízes de uma equação pelo método da bisecção do intervalo, em que a simplificação da situação consiste na diminuição do intervalo no qual se procura a raíz da equação.

Contudo nem sempre se pode recorrer à técnica de redução, sendo necessário considerar também o processo da decomposição de situações. Neste caso uma das situações torna-se crítica e por esta razão deve ser substituida por um conjunto de situações mais simples, cada uma das quais deve ser examinada. É de notar que convém decompor no menor número possível de situações mais simples. O processo de decomposição é representado por uma árvore de pesquisa de cujos vértices saem vários arcos. Por exemplo, temos uma função booleana $f(x_1,...,x_n)$ e pede-se para encontrar um vector booleano **x** que faz com que a função fique igual a 1. De acordo com o teorema de Shannon podemos decompor uma situação corrente em duas, escolhendo uma variável x_i e supondo que numa situação esta variável fica igual a 1 e noutra fica igual a 0:

$$f(x_1, \ldots, x_n) = x_i f(x_1, \ldots, 1, \ldots, x_n) \vee \overline{x}_i f(x_1, \ldots, 0, \ldots, x_n)$$

A seguir analisamos a primeira situação. Se esta nos levar à solução, o processo da pesquisa para; caso contrário teremos de examinar a segunda situação.

As vezes consegue-se ordenar as situações para as quais se decompõe a situação corrente, i.e. pode-se analisar algumas características do problema e com base nesta informação escolher uma situação preferível que tem maiores probabilidades de nos levar à solução. Isto melhora significativamente a eficiência dos algoritmos. Por exemplo, quando procuramos um vector ortogonal a cada linha de uma dada matriz e não podemos reduzir uma situação actual, a única saída consiste no exame sequencial dos valores possíveis de todas as componentes desconhecidas deste vector. Contudo, para encontrar a solução o mais rápido possível, recomenda-se escolher primeiro a componente que corresponde à coluna mais determinada da matriz, i.e. à coluna que tem o número mínimo de valores "-"[7].

Em muitos problemas práticos nem sempre é possível encontrar a solução óptima, pelo que os métodos aproximados têm ampla difusão. Nos métodos aproximados a busca que se efectua é bastante reduzida devido à eliminação das variantes menos promissoras. Contudo entre as variantes eliminadas pode estar a óptima. A eficiência dos métodos aproximados é avaliada segundo a qualidade da solução obtida e o tempo gasto para a obter.

VI. IMPLEMENTAÇÃO DE ALGORITMOS COMBINATÓRIOS

Os algoritmos de resolução de problemas combinatórios podem ser implementados em software, em hardware ou de uma maneira mista. Devido à necessidade do exame de muitas variantes no processo de solução, a implementação em software requer consumos significativos de tempo. Os modelos considerados são de bastante difícil representação na memória dos computadores, portanto devemos aplicar métodos especiais para a organização, armazenamento e processamento de dados específicos, tais como grafos, etc.

Pode-se implementar os algoritmos combinatórios em hardware mas isto também revela algumas dificuldades. Por um lado cada problema é único em termos do conjunto de operações que requer. O tipo de operações é normalmente bastante reduzido e estas são aplicadas às estruturas uniformes de dados (por exemplo, linhas e colunas de matriz). Por outro lado, é muito difícil propor algum dispositivo universal que possa ser usado para a resolução de qualquer problema arbitrário. Isto inviabiliza a construção de hardware dedicado porque este seria muito caro e utilizado de uma maneira ineficiente.

Contudo o uso de dispositivos reconfiguráveis, tais como FPGAs (*Field Programmable Gate Arrays*), seria justificado e eficaz. É evidente que a implementação em hardware reduz significativamente o tempo necessário para a resolução de problemas respectivos. Neste caso podemos usar as FPGAs de duas maneiras diferentes:

1. Como um co-processador reconfigurável que pode ser reprogramado para a resolução de problemas diferentes. Sendo assim, a reconfiguração consiste em implementar um conjunto de operações reduzido. A construção de um co-processador deste tipo não é facil, mas é possível [13, 14].
2. Como um dispositivo especial que mapeia directamente uma determinada estrutura de dados. Por exemplo, um grafo pode ser modelado por um sistema de flip-flops interligados que correspondem aos vértices do grafo. Uma ligação entre dois flip-flops representa um arco entre os vértices respectivos. Tais estruturas permitem resolver muitos problemas combinatórios, por exemplo os de encontro de caminhos num grafo. É possível especificar as estruturas de maneira a poderem ser guardas numa biblioteca expansível.

Os algoritmos combinatórios podem também ser realizados de uma maneira mista, sendo umas partes implementadas em software e outras em hardware, i.e. num dispositivo baseado em FPGA e ligado ao barramento do computador. Uma estrutura deste tipo pode aproveitar das vantagens de implementações diferentes.

VII. CONCLUSÕES

Neste artigo apresentamos alguns exemplos de problemas combinatórios típicos e consideramos vários modelos matemáticos usados normalmente para a sua especificação e resolução. Foi demonstrado que estes modelos são mutuamente convertíveis. Analisámos alguns exemplos práticos que surgem na área de projecto de dispositivos digitais e requerem a resolução de problemas combinatórios. Consideraram-se também os principais métodos utilizados na fase de desenvolvimento de algoritmos de resolução de problemas combinatórios. Estes algoritmos podem ser realizados quer em software, quer em hardware. A implementação em software requer normalmente muitos recursos do computador e consome bastante tempo. Existem muitos problemas que nem sequer podem ser resolvidos devido à sua complexidade elevada (a maioria dos problemas combinatórios não são resolúveis em tempo polinomial). A implementação em hardware permite reduzir o tempo necessário para a execução, mas neste caso cada aplicação exige um circuito dedicado. Para resolver este problema pode-se usar hardware reconfigurável, tal como FPGAs. Concluímos que o melhor resultado será obtido caso combinemos de uma maneira razoável a implementação em software, com a utilização de hardware reconfigurável baseado em FPGAs.

REFERÊNCIAS

[1] A.D.Zakrevski, “Combinatorial Theory of Logical Design”, Avtomatica i Vychislitelnaya Tekhnica, Nº2, 1990, pp. 68-79 (em russo).

[2] A.D.Zakrevski, “Graph Coloring and Decomposition of Boolean Functions “, Logical Design, Nº. 5, 2000 (em russo).

[3] R.Diestel, “Graph Theory”, 2nd ed., Springer, 2000, p.310.

[4] A.Aho, J.Hopcroft, J.Ullman, "Data Structures and Algorithms", Addison-Wesley, 1983.

[5] M.Golumbic, "Algorithmic Graph Theory and Perfect Graphs", Academic Press, 1980.

[6] V.A.Sklyarov, “Synthesis of Finite State Machines Based on Matrix LSI”, Minsk: Science and Techniques, 1984 (em russo).

[7] A.D.Zakrevski, “Logical Synthesis of Cascade Networks”, Moscow: Nauka, 1981 (em russo).

[8] A.D.Zakrevski, “Combinatorial Problems over Logical Matrices in Logic Design and Artificial Intelligence”, Electrónica e Telecomunicações, vol. 2, Nº 2, 1998, pp. 261-268.

[9] Giovanni De Micheli, Synthesis and Optimization of Digital Circuits, McGraw-Hill, Inc., 1994.

[10] S.I.Baranov, L.N.Zhuravina, V.A.Skliarov, “Computer Aided Design”, Minsk, High School, 1981 (em russo).

[11] S.I.Baranov, V.A.Sklyarov, “Digital Devices Based on Programmable Matrix LSI”, Moscow: Radio and Communications, 1986 (em russo).

[12] G.D.Hachtel, F.Somenzi, “Logic Synthesis and Verification Algorithms”, Kluwer Academic Publishers, 1996.

[13] I.Skliarova, A.B.Ferrari, “Exploiting FPGA-based Architectures and Design Tools for Problems of Reconfigurable Computations”, Proceedings of the SBCCI 2000 XIII Symposium on Integrated Circuits and System Design, Manaus, Brazil, September 2000, pp. 347-352.

[14] I. Skliarova, A.B.Ferrari, "Development tools for problems of combinatorial optimization", Proceedings of the 4th Portuguese Conference on Automatic Control (CONTROLO'2000), Guimarães, Portugal, October 2000, pp. 552-557.