Elementos de Programação

Projecto de Computação Evolutiva

Departamento de Matemática, IST

LMAC, MEBiom
Novembro de 2014

Satisfazibilidade

É sabido que toda a fórmula da lógica clássica pode ser escrita em *forma normal conjuntiva*, isto é, como uma conjunção (AND) de *cláusulas*, em que cada cláusula é uma disjunção (OR) de *literais* (*positivo* x_i, ou *negativo* $\overline{x_i}$). No exemplo abaixo temos uma forma normal conjuntiva em 3 variáveis proposicionais, com 4 cláusulas.

$$(x_1 \text{ OR } x_2 \text{ OR } \overline{x_3}) \text{ AND } (\overline{x_1} \text{ OR } \overline{x_2}) \text{ AND } (\overline{x_2} \text{ OR } \overline{x_3}) \text{ AND } (\overline{x_1} \text{ OR } x_2 \text{ OR } x_3)$$

Uma fórmula é *satisfazível* se existe uma atribuição de valores lógicos às variáveis (*valoração*) que a torna verdadeira. É o caso da fórmula dada acima, atribuindo o valor verdadeiro a x_1, x_3 e o valor falso a x_2, como é fácil de inferir do esquema abaixo.

$$\begin{array}{lllllll}
(& \textcircled{x_1} & \text{OR} & x_2 & \text{OR} & \overline{x_3} &) \text{ AND} \\
(& \overline{x_1} & \text{OR} & \textcircled{\overline{x_2}} & & &) \text{ AND} \\
(& & & \textcircled{\overline{x_2}} & \text{OR} & \overline{x_3} &) \text{ AND} \\
(& \overline{x_1} & \text{OR} & x_2 & \text{OR} & \textcircled{x_3} &)
\end{array}$$

Em geral, o problema de determinar se uma forma normal conjuntiva dada é ou não satisfazível (vulgo **SAT**) é extremamente difícil, não sendo conhecido nenhum algoritmo eficiente para o resolver. O desempenho dos melhores algoritmos conhecidos é essencialmente equivalente ao da tarefa de verificar, uma a uma, todas as possíveis valorações às variáveis. Por exemplo, para uma fórmula com 100 variáveis, testar todas as 2^{100} possíveis valorações num computador com 10GHz (que executa cerca de 10^{10} operações por segundo) requer um tempo na ordem de $2^{100}/10^{10}$ segundos, cerca de 300 vezes a idade estimada do universo.

Há muitos exemplos de problemas de largo espectro para os quais não é conhecida uma solução geral eficiente. O problema **SAT** é talvez o mais famoso desses problemas, pois foi o primeiro problema *NP-difícil* estudado por Stephen Cook em 1971, o fundador da moderna teoria da complexidade. Para além do seu interesse específico, uma solução eficiente para **SAT** teria um impacto espantoso em inúmeras áreas, nomeadamente em problemas de planeamento e optimização, em matemática, economia, engenharia, inteligência artificial, biociências (vide o excelente livro *The Golden Ticket* de Lance Fortnow, Princeton University Press, 2013). Infelizmente (ou não), conjectura-se que não existe nenhum algoritmo eficiente para **SAT**, e portanto, apesar de toda a investigação aplicada à construção de **SAT**-*solvers*, é usual na prática recorrer a soluções aproximadas do problema.

Computação Evolutiva

O objectivo deste projecto é desenvolver em *Mathematica* um programa que calcule uma solução (exacta ou aproximada) para o problema da satisfazibilidade, utilizando o paradigma de computação evolutiva. A ideia é implementar uma variante do algoritmo adaptativo **ASAP** proposto por Claudio Rossi, Elena Marchiori e Joost Kok no ano 2000.

Dada uma fórmula *Form* na forma normal conjuntiva, pretende-se simular durante um período de tempo dado *TFim* a evolução de uma população formada por um número dado *In* de indivíduos (que no caso devemos entender como valorações às variáveis de *Form*) de acordo com a sua adaptação ao problema (que deve ser máxima para uma valoração que satisfaça *Form*). O programa termina exibindo uma solução exacta, caso seja encontrada nalgum momento, ou exibindo a melhor solução aproximada presente na população no final do período de simulação.

Ao longo da simulação, cada indivíduo evolui por processos de *mutação*, *melhoramento* e *regeneração*, de acordo com leis aleatórias que dependem de parâmetros dados *TMut*, *TMelh*, *TReg*. Estes mecanismos de evolução produzem alterações nos indivíduos e valorações associadas. Obviamente, cada valoração V atribui um valor lógico verdadeiro/falso a cada uma das variáveis $x_1, \ldots, x_k$ de *Form*, assumindo que tem k variáveis. Denominaremos $V(x_1), \ldots, V(x_k)$ por *bits* de V. Cada indivíduo da população deve ter um identificador único, podendo coexistir vários indivíduos com a mesma valoração associada, para além de atributos *Mem* - que lista as últimas valorações da evolução do indivíduo, *Actv* - que lista os bits activos da valoração do indivíduo, e *PrMut* - a probabilidade de mutação dos bits da valoração do indivíduo. Se *Form* é uma conjunção de n cláusulas, C_1 AND ... AND C_n, então o *coeficiente de adaptação* de um indivíduo com valoração V num certo instante é dado por

$$\frac{\#\{1 \leq i \leq n : V \text{ satisfaz } C_i\}}{n}.$$

Cada indivíduo da população inicial tem associada uma valoração obtida pelo processo de melhoramento (descrito abaixo) a partir de uma das 2^k valorações possíveis, obtida de forma aleatória uniforme. No início, *Mem* está vazia, todos os bits estão activos, e *PrMut*$= 0.5$.

O simulador deve ser construído de acordo com a técnica de simulação digital estocástica por sequenciamento de eventos pendentes, havendo que considerar eventos de 3 tipos:

- *mutação*, de certo indivíduo, cuja cadência é uma variável aleatória exponencial de valor médio *TMut*, e cuja ocorrência resulta na avaliação de uma valoração obtida trocando, com probabilidade *PrMut**, cada um dos bits activos da valoração associada ao indivíduo nesse momento; a nova valoração substitui a anterior desde que não tenha pior coeficiente de adaptação, sendo descartada e ficando o indivíduo inalterado em caso contrário;

- *melhoramento*, de certo indivíduo, cuja cadência é uma variável aleatória exponencial de valor médio *TMelh*, e cuja ocorrência resulta na substituição da valoração associada ao indivíduo por uma valoração obtida a partir dela do seguinte modo - cada um dos bits activos da valoração é trocado, sequencialmente por ordem aleatória uniforme†, desde que isso não traga prejuízo ao seu coeficiente de adaptação;

*Note que se `X` tem um valor no intervalo $[0, 1]$ a probabilidade de a expressão `Random[] < X` ter valor `True`, em *Mathematica*, é precisamente `X`.

†Note que em *Mathematica* a expressão `RandomInteger[{1,N}]` toma uniformemente cada um dos valores de `1` a `N`, isto é, cada valor tem probabilidade $1/N$. Da mesma forma, a expressão `RandomSelect[w]` escolhe uniformemente um dos elementos da lista `w`.

- *regeneração*, cuja cadência é uma variável aleatória exponencial de valor médio *TReg*, e cuja ocorrência resulta na depuração simultânea de todos os indivíduos da população; o que acontece a cada indivíduo depende da lista *Mem*:
  - se *Mem* ainda não tem 10 valorações, então o indivíduo permanece inalterado;
  - se *Mem* tem 10 valorações, mas não há 3 valorações distintas na lista, então o indivíduo é *colonizado*, sendo reinicializado com uma valoração obtida pelo processo de melhoramento a partir de uma das valorações memorizadas pelos restantes indivíduos da população, escolhida aleatoriamente;
  - se *Mem* tem 10 valorações e há pelo menos 3 valorações distintas na lista, então passam a estar inactivos os bits cujo valor coincide em todas as valorações memorizadas, *Mem* é reinicializada, e *PrMut* é actualizado para o número de bits ~~inactivos~~ activos a dividir por $2k$.

Objectivos

O projecto deve ser desenvolvido de acordo com o método de programação modular, por camadas, centrado nos dados.

1. Comece por identificar os tipos de dados relevantes, nomeadamente *fórmula* (fornecido com o enunciado), *valoração*, *indivíduo*, *população*, *evento* e *cadeia de acontecimentos pendentes*, e respectivas operações.
2. Desenvolva de seguida o programa abstracto pretendido sobre a camada que disponibiliza estes objectos.
3. Implemente estas camadas sobre a camada básica do *Mathematica*.
4. Integre o programa obtido em 2 com os pacotes desenvolvidos em 3.
5. Experimente o programa desenvolvido com dados à sua escolha, nomeadamente usando as operações sobre fórmulas disponibilizadas no pacote `formula.m`, e discuta os resultados obtidos.

Entrega e Avaliação

O projecto, a realizar em grupos de 3 alunos, será entregue através do sistema Fénix, após a inscrição do respectivo grupo, até ao final do dia 13 de Dezembro de 2014, **impreterivelmente**. A entrega do trabalho deve consistir de um único arquivo (zip ou rar) contendo o simulador, os pacotes desenvolvidos, e um pequeno relatório que descreva sucintamente a solução obtida, os tipos de dados utilizados e respectivas opções de implementação, e uma discussão dos resultados obtidos.

O trabalho vale 10 valores da nota final da disciplina, que se distribuem da seguinte forma: descrição dos tipos de dados relevantes e suas operações (2.5 valores), implementação eficiente dos tipos de dados e sua apresentação sob a forma de pacotes (3 valores), simulador (3 valores), experimentação (1.5 valores).