2º Projecto de Inteligência Artificial

2016/2017

1º Semestre

O projecto está dividido em 7 ficheiros, mas apenas 5 fundamentais. Um deles chama-se **satplan**, e é o responsável por correr todos os outros. O **Encoder** é o ficheiro que tem as principais funções do codificador e descodificador das frases do problema, e o **EncoderUtils** tem as funções úteis a este, por motivos de maior organização. O ficheiro **satsolver** é o que poteriormente vai ser chamado para a resolução do problema SAT, que tanto pode ser resolvido com o **DPLL** (o que criou melhores resultados), o **DPLL2,** e o **WalkSAT.**

O primeiro passo do algoritmo é chamar a função initPlan, do Encoder, que para além de ler do ficheiro guarda imediatamente a linha correspondente ao initial e goal state, e um vector com as linhas das acções. Analisando todas estas, cria-se um vector constants, que terá a lista de todas as constantes do problema, por exemplo : constants = [‘A’,’B’,’Table’]. Para além disso, cria-se ainda uma lista de listas chamada actions, com 3 posições em cada campo: o nome de uma acção, as suas precondições, e os seus efeitos, bem como uma lista com todos os nomes dos atomos existentes, ex: ['on(b,f)', 'clear(b)']

A função organizeVariables é também chamada e retorna o initialState e o atomListComb. O initialState tem as proposições iniciais indicadas no ficheiro e todas as outras possível negadas, já com o grounding feito. O grounding, tanto para os atomos como para as acções, é feito através da ferramenta combinations do itertools para cada número pretendido de constantes, e substituindo as respectivas variáveis usando regex. O atomListComb é a lista das combinações de todos os atomos, com o gorunding feito.

Esta função inicial retorna no fim as listas: actName, initialState, goalState, constants, actions, atoms e atomListComb, que nunca serão alteradas durante todo o algoritmo.

(a) Describe the SAT encoding used, if different than the one described here.

Depois disto feito, o satplan coloca o horizonte a 1, e começa por chamar o Encoder. O objectivo é criar uma frase constituida por números, cada um correspondente a uma variável, que introduzida no solver pode ou não ser satisfeita. Caso seja, o problema pode ser resolvido aplicando apenas uma acção, de tempo 0 para 1, chamando-se de imediato o decoder para a descodificar. Caso contrário, o solver retorna ‘UNSAT’, incrementa-se o horizonte, e chama-se de novo o encoder. Desta segunda vez que é chamado, a função encode do encoder segue exactamente os mesmos passos, apenas com tempos entre 1 e 2 em vez de 0 e 1 como da primeira vez. É importante notar que o que acontece agora é que no fim dos 5 passos do linear encoder, a sentence final será a soma de todas as sentences dos tempos anteriores, pelo que o que é repetido é a codificação da sentence completa para o formato de SAT desejado (com um número representando cada variável).

O algoritmo criado no encoder foi exactamente o descrito no enunciado. A sentence final será uma conjunção de cada um dos 5 processos: o estado inicial (com os predicados dados no ficheiro e os negados que não se verificam em t=0), o estado final (atualizado para cada h), as actions em CNF, os frame axioms, e as at least e at most one clauses.

Tendo todas as expressões pretendidas para certo t, é a função convertToSAT que descobre todas as variáveis diferentes e converte para números.

A principal melhoria implementada foi a eliminação de constantes repetidas emc ada conjunto de atomos e acções. Ou seja, em vez de ver o produto de todas as constantes, ex: move(A,A,A), com a utilização de itertools, passa a ser calculada apenas a lista de permutações das mesmas,

ex: move(A,A,B). Isto provou-se bastante útil na redução do tempo de cálculo da solução, tendo permitido a resolução do ficheiro blocks3.dat em tempo útil.

(b) Describe all improvements done (if any) over the basic DPLL algorithm.

Foram implementadas duas versões melhoradas de DPLL e uma implementação WalkSAT/GSAT. Inicialmente foi implementado DPLL recursivo em que apenas o modelo (dicionário dVar) era duplicado para cada ramo de tentativa (guardando as cláusulas em variável global), juntamente com uma propagação de cláusulas unitárias e literais puros melhoradas por propagar até não haver mais. A escolha do símbolo a atribuir é escolhido linearmente com o número. Esta foi a implementação que mostrou melhores resultados.

Na segunda implementação é duplicado o modelo e a sentence, em que a sentence decresce de tamanho à medida que símbolos são atribuídos. Em adição às melhorias de propagação de unidade e literais puros da implementação anterior, foi implementada a escolha do símbolo a atribuir para o branching por DLIS (Dynamic Largest Individual Sum), escolhendo a variável que aparece com mais frequência em cláusulas insatisfeitas. Também está presente, em comentário a escolha linear, como também aleatória. Esta é a implementação com o nome DPLL2.

Foi também implementada WalkSAT. Apesar de não provar a insatisfazibilidade, pode ser utilizado de forma útil, escolhendo parâmetros MaxFlips e MaxRestarts adequados, pois é um algoritmo muito rápido. Este foi implementado com as features de escolha entre atribuição aleatória e atribuição greedy por probabilidade ajustável.

(c) Evaluate quantitatively the benefits of the improvements introduced to the SAT solver (if any).

Todas as melhorias implementadas se mostraram muito importantes e fundamentais para a duração do algoritmo.

Alguns valores estão representados na tablela seguinte:

Nome do problema	DPLL	DPLL2		
	Tempo	Tempo	#Variáveis	#Clauses
Trivial1.dat	0.0s	0.0s	5	8
Trivial3.dat	0.05s	0.17s	39	258
Blocks2.dat	0.07s	0.29s	53	556
Blocks3.dat	2.42s	21.60s	175	5261