Uma visão geral sobre General Game Playing

Ulysses Bonfim

Universidade Federal do Paraná

Resumo

Os tradicionais jogadores, como Deep Blue, apesar de terem grande desempenho contra os seres humanos, são desenvolvidos para apenas um jogo específico. General Game Playing é uma nova classe de jogadores construídos para conseguirem jogar qualquer jogo, dada apenas a sua descrição. O presente artigo mostra como é a estrutura básica de um jogador e os já existentes submetidos à competição anual de General Game Playing.

Keywords:: General Game Playing, GGP Competition

Author's Contact:

ubds06@c3sl.ufpr.br

1 Introdução

A ideia de construir um programa de computador capaz de derrotar um ser humano em um jogo complexo, xadrez por exemplo, foi alcançada a tempos, porém este resultado não mostra que a máquina tenha mais inteligência que o jogador. O DeepBlue, computador responsável pela vitória sobre o mestre de xadrez Kasparov, foi projetado especificamente para jogar um único jogo, se o confronto fosse um jogo da velha, ou mesmo alguma variante do xadrez, a máquina mal saberia responder uma jogada válida.

Em contrapartida aos robôs desenvolvidos para jogos específicos, existe um ramo da Inteligência Aritificial que aplica conhecimentos de lógica, busca heurística, representação do conhecimento, entre outros, para a criações de jogadores que possam jogar qualquer tipo de jogo. O desafio dos Jogos Generalistas (*General Game Playing*, **GGP**) é contruir um programa que possa interagir e, principalmente, ganhar, sem qualquer conhecimento prévio da mecânica do jogo. O programa deve raciocinar apenas sobre a descrição do jogo que lhe é dada. Para promover o avanço nesta área, a Universidade de Stanford organiza, desde 2005, uma competição anual entre jogadores generalistas.

Este artigo apresentará como pode ser feita a construção de um jogador, a linguagem usada para descrever os jogos e quais são as características dos jogadores submetidos à competição.

2 General Game Playing

Jogadores tradicionais como DeepBlue[Campbell and Hsu 2002], Chinok[Schaeffer and Szafron 1992] e TD-Gammon[Tesauro 1994] usam árvores como estrutura computacional para representar e analisar as jogadas (ou nós). Isto só é possível porque na construção do jogador, o programador tem o conhecimento prévio das regras, logo, sabe quais jogadas podem ser feitas, independente se são boas ou ruins. No entanto, na maioria dos jogos, o espaço de estados para busca é muito grande, tornando inviável percorrer todos os nós até achar um caminho que leve a vitória (busca exaustiva).

A solução é usar uma função que avalie quais nós são mais promissores, permitindo que os piores não sejam expandidos. Esta função, chamada de heurística, é gerada à partir da análise sobre as regras e de experiência adquirida durante os jogos. No jogo de damas, por exemplo, uma heurística plausível seria a diferença do número de peças em relação ao adversário. Utilizando esta heurística, pode-se comparar dois nós e decidir qual caminho tomar na árvore.

O cenário proposto pelo GGP inviabiliza as técnicas clássicas na construção de jogadores porque elas são específicas para cada jogo. Nos jogos generalistas, não sabemos de antemão qual jogo o programa irá enfrentar, logo a ávore de busca só será criada depois que receber quais são as regras e delas tirar os movimentos válidos. Também não podemos contar com experiência no jogo, para derivar uma heuristica. Esta é construída com técnicas que diferem de jogador para jogador.

A descrição dos jogos na competição é feita através de uma linguagem derivada da lógica de primeira ordem, *Game Description Language* (**GDL**). As decisões do jogador serão baseadas nas informações que podemos retirar e inferir desta descrição.

3 Game Description Language

Em GDL os jogos são tratados como máquinas de estado. A descrição de um jogo consiste em um conjunto de fatos verdadeiros, que descrevem o estado atual, as relações que modificam o estado atual, as regras do jogo, e qual o estado deve ser atingido para o que o jogo acabe.

Apesar do objetivo do GGP englobar todos os tipos de jogos, a competição trata apenas jogos *determinísticos* e *perfeitamente informados*. Se em um jogo conseguirmos determinar seu próximo estado, dado o atual e as ações tomadas por todos os jogadores, então ele é *determinístico*. Go e Otelo são exemplos de jogos desta classe, já Gamão fica fora porque o próximo estado depende de um fator não determinístico, os dados. Nos jogos *perfeitamente informados* o estado do jogo é conhecido por todos os participantes. São considerados perfeitamente informados xadrez e jogo da velha, pois o estado atual está no campo de visão dos jogadores. Entretanto, truco e batalha naval, onde os jogadores escondem para si parte do estado atual, não entram nesta categoria. Jogos podem ter um ou mais jogadores, baseados em rodadas ou simultâneos.

Um pequeno conjunto de palavras chaves é usado em GDL para criar a descrição dos jogos: *role, init, next, true, does, terminal, goal* e *distinct*. Como exemplo de construção na linguagem, usaremos o jogo da velha, onde os estados correspondem a uma matriz 3x3 em que cada célula pode estar vazia, preenchida com *x* ou *o*.

A palavra, ou *relação*, *role* é usada para descrever quais serão os jogadores na partida. No jogo da velha temos a seguinte declaração:

```
(role xplayer)
(role oplayer)
```

indicando dois jogadores, *x* e *o*.

A relação *init* representa os fatos que são verdades no início do jogo. No estado inicial do jogo da velha todos as células são vazias e quem começa é o jogador *x*.

```
(init (cell 1 1 b))
(init (cell 1 2 b))
(init (cell 1 3 b))
(init (cell 2 1 b))
(init (cell 2 2 b))
(init (cell 2 3 b))
(init (cell 3 1 b))
(init (cell 3 2 b))
(init (cell 3 3 b))
(init (control xplayer))
```

Relações adicionais são usadas em conjunto com as palavras chaves para que o jogo possa ser descrito. Para as células temos *cell*, que recebe três tokens como argumento, a posição vertical, horizontal e o conteúdo guardado. Entretanto, a declaração acima poderia ser feita da seguinte maneira:

```
(init (booo foo foo beh))
(init (booo foo bar beh))
(init (booo foo xyz beh))
```

⋮

```
(init (trew xplayer))
```

sem perder a semântica da estrutura. Essa falta de comprometimento com a sintaxe tem um papel importante na formulação da heurística, como veremos adiante.

As relações *true* e *init* tomam uma relação ou um token como parâmetro, que são verdadeiros no estado atual do jogo. A diferença entre eles é que *init* é usada para criar o estado inicial do jogo e não é mais usada depois, enquanto *true* fará as atualizações dos fatos nos demais estados. A declaração:

```
(true (cell 2 2 b))
(true (control xplayer))
```

indica que no estado atual a célula do centro da matriz (posição 2,2) contém um branco e o jogador *x* deverá jogar.

Análoga à *true*, a relação *next* refere aos fatos que serão verdades no próximo estado. Em nosso exemplo, o controle das rodadas é feito da seguinte maneira:

```
(<= (next (control xplayer))
    (true (control oplayer)))

(<= (next (control oplayer))
    (true (control xplayer)))
```

O símbolo <= indica uma implicação reversa.

No jogo da velha um jogador poderá marcar uma das células se ela estiver vazia e se for a sua vez de jogar. A declaração de quais jogadas podem ser feitas em um determinado estado do jogo é descrita usando a palavra chave *legal*. Essa relação toma como parâmetro um jogador e uma ação (ou movimento).

```
(<= (legal ?player (mark ?x ?y))
    (true (cell ?x ?y b))
    (true (control ?player)))

(<= (legal xplayer noop)
    (true (control oplayer)))

(<= (legal oplayer noop)
    (true (control xplayer)))
```

O token *noop* é usado no controle de rodadas quando não é a vez do jogador, sua única ação é não fazer nada. Tokens iniciados com "?" são variáveis.

Caso um jogador possa marcar uma célula, verificando através do *legal* se é possível, no próximo estado do jogo é esperado que aquela célula contenha a marca deixada. O resultado das ações legais tomadas é descrita usando a relação *does*:

```
(<= (next (cell ?m ?n x))
    (does xplayer (mark ?m ?n))
```

No próximo estado a célula *?m ?n* conterá um *x* se a jogada puder ser efetuada pelo jogador *x* no atual estado.

Para o problema do quadro (*frame problem*) temos uma adição dos axiomas:

```
(<= (next (cell ?x ?y b))
    (does ?player (mark ?m ?n))
    (true (cell ?x ?y b))
    (distinctCell ?x ?y ?m ?n))

(<= (distinctCell ?x ?y ?m ?n)
    (distinct ?x ?m))

(<= (distinctCell ?x ?y ?m ?n)
    (distinct ?y ?n))
```

indicando que se uma célula contém um branco no estado atual e o jogador não a marca, no próximo estado ela continuará contendo branco. A palavra chave *distinct* é usada para comparar dois axiomas.

O fim de jogo é alcançado quando um dos jogadores consegue marcar uma sequência de três células, em uma coluna, linha ou na diagonal, ou quando não há mais espaços em branco.

```
(<= terminal
    (line x))

(<= terminal
    (line o))

(<= terminal
    (not open))
```

A pontuação que cada jogador consegue ao atingir o final do jogo é encontrada nas regras *goal*.

```
(<= (goal xplayer 100)
    (line x))

(<= (goal xplayer 50)
    (not (line x))
    (not (line o))
    (not open))

(<= (goal xplayer 0)
    (line o))
```

O jogador *x* conseguirá 100 pontos se tiver uma sequência, 50 se der "velha" e nenhum se o adversário fizer a sequência, quando um dos estados terminais for atingido. A descrição completa do jogo da velha pode ser encontrada nos anexos.

Para simplificar a descrição, usamos uma relação adicional *line*, que é verdadeira se o jogador conseguiu fazer uma sequência. No domínio do jogo da velha não se mostra necessário, porém outras regras podem existir para simular relações numéricas.

```
(succ 1 2) (succ 2 3) (succ 3 4)
(nextcol a b) (nextcol b c) (nextcol c d)
```

A relação *succ* define como é incrementado algum contador, o número da rodada em algum jogo que possui um limite de rodadas para acabar, por exemplo. No xadrez, a localização das colunas seria feita usando a relação *nextcol*. Encontrar este tipo de relação numérica na descrição tem um grande valor, pois servem de ponte entre representação lógica e numérica.

Para que os jogos sejam *perfeitamente informados* as regras da competição especificam que, antes de cada rodada, os jogadores recebem todas as jogadas feitas no último turno.

```
(does x (mark 2 2))
(does o noop)
```

Desta maneira é possível calcular em qual estado do jogo estamos.

3.1 Prova de teoremas

Para que seja possível derivar os movimentos legais o jogador deve utilizar um provador de teoremas. Com a declaração do estado inicial, e as atualizações dos movimentos, podemos inferir quais jogadas são legais no estado atual. Para ilustrar, tomemos como exemplo o seguinte estado parcial:

1. `cell(1 1 b)`
2. `control(xplayer)`

Os movimentos legais e seus efeitos na base de conhecimento podem ser traduzidos como implicações.

3. `(cell(?x ?y b)` $\wedge$ `control(?player))` $\Rightarrow$ `legal(?player (mark ?x ?y))`
4. `legal(xplayer (mark ?x ?y)` $\Rightarrow$ `next(cell(?x ?y x))`

Usando o provador de teoremas, podemos inferir

5. `legal(xplayer (mark 1 1))`, por *1, 2* e *3*
6. `next(cell(1 1 x))`, por *4* e *5*

Assim, conseguimos descobrir que é possivel no estado atual marcar a primeira célula (posição 1,1) do tabuleiro. De maneira análoga, outras jogadas são inferidas. No entanto, a prova de teoremas é custosa computacionalmente, o que agrava o problema da construção da árvore de buscas, aumentando ainda mais o valor de uma boa heurística.

4 Heurística

A menos que o espaço de estado seja pequeno o bastante para ser gerado integralmente, considerando também o peso da prova de teoremas, o jogador deve usar uma função heurística para avaliar os novos estados encontrados e, assim, limitar a busca. Tal função de avaliação é específica de jogo para jogo. Contar o número de peças a nosso favor é bom para um jogo como Damas, mas produz resultados catastróficos em Resta Um. Grande parte do esforço despendido na criação de um jogador tradicional, é feita por seres humanos tentando melhorar a função de avaliação.

Em GGP, a construção da heurística deve ser baseada apenas nas informações contidas na descrição. O fator determinante para o sucesso do jogador está em como utilizar as características (*features*) para compor a função de avaliação. Cada jogador submetido à competição analisa a descrição e constroi a função heurística de maneira ímpar. Veremos a seguir uma técnica simples para o reconhecimento das *features* descrita por [Kuhlmann and Dresner 2006] e como ele gera a função heurística.

4.1 Identificando estruturas sintáticas

Na descrição podemos reconhecer cinco estruturas básica: relações de sucessão, contadores, tabuleiros, marcadores, peças e quantidades. A identificação destas é feita comparando as relações, contidas na descrição, com modelos das estruturas. Apesar dos modelos serem específicos para GDL, eles podem ser conceitualmente adaptados para qualquer outra linguagem lógica.

Como mencionado anteriormente, não podemos depender dos tokens para a construção do modelo. Na competição, a descrição dos jogos tem o nome dos tokens e relações embaralhados para que o jogador não obtenha dicas da estrutura do jogo apenas lendo relações como *cell*. A relação sucessor, por exemplo, poderia ser reescrita como:

```
(foo on off) (foo off one) (foo one blue)
(jap ichi ni) (jap ni san) (jap san shi)
```

Ainda sim podemos identificar esta como sendo uma relação de sucessão, pois a sucessão é uma propriedade estrutural e não léxica.

```
<succ> <atom 1> <atom 2>
<succ> <atom 2> <atom 3>

        ⋮

<succ> <atom n-1> <atom n>
```

Sem a identificação desta relação, não conseguiríamos saber que a coluna *a* esta próxima da coluna *b*, em um jogo como Xadrez ou Damas.

Um jogo pode conter várias realções de sucessão, que são usadas para identificar *contadores*. Um contador é um termo funcional incrementado em cada estado do jogo e podem ser identificados usando o seguinte modelo:

```
(<= (next (<counter> ?<var1>))
    (true (<counter> ?<var2>))
    (<successor> ?<var2> ?<var1>))
```

onde <counter> é a função identificada e <sucessor> uma relação de sucessão.

Outra estrutura identificavel é o *tabuleiro*. Um tabuleiro é uma matriz de duas dimenções de células que mudam de valor. [Kuhlmann and Dresner 2006] assume que toda relação ternária de um estado é um tabuleiro. No entanto, um dos argumentos do tabuleiro precisa corresponder a um estado da célula, que nunca pode ter dois valores simultâneos. No jogo da velha temos um tabuleiro, a relação *cell*. Se os argumentos de um tabuleiro são ordenados por alguma relação de sucessão, como as colunas e linhas no Xadrez, o tabuleiro é considerado *ordenado*.

Após reconhecido o tabuleiro, podemos identificar *marcadores*, que são objetos que ocupam as células do tabuleiro. Se um marcador esta em apenas uma célula de cada vez, então ele recebe o nome de *peça*. Em nosso exemplo, os marcadores *x* e *o* não são considerados peças porque podem estar em mais de uma célula do tabuleiro ao mesmo tempo.

4.2 Construção da Heurística

A estruturas identificadas sugerem *features*, valores numéricos extraídos das estruturas. Se um tabuleiro é considerado ordenado, então podemos atribuir um valor numérico para as cordenadas *x* e *y* para cada peça. Utilizando as relações *foo* e *jap*, declaradas anteriormente, o exemplo:

```
(init (cell off shi p))
```

denota que a peça, ou marcador, *p* esta na coordenada (1,2). Destas coordenadas o agente pode calcular a distância Manhattan entre as peças. Se o tabuleiro não está ordenado, ainda podemos contar o número de ocorrências dos marcadores.

A função heurística é derivada das *features* descobertas. Para aumentar as chances de escolher uma boa heurística, [Kuhlmann and Dresner 2006] cria um conjunto de heurísticas candidatas, cada uma sendo a máximização e a minimização das *features* descobertas. Deste modo, em um jogo como Resta Um, a minimização da *feature* número de peças é uma boa heurística candidata.

A função de avaliação terá um valor que varia de $R^- + 1$ e $R^+ - 1$, onde R^- e R^+ são o mínimo e máximo que um jogador pode atingir na relação *goal*. Os valores de maximização e minimização da função heurística são calculados respectivamente da seguinte forma:

$$H(s) = 1 + R^- + (R^+ - R^- - 2) * V(s)$$
$$H(s) = 1 + R^- + (R^+ - R^- - 2) * [1 - V(s)]$$

onde $H(s)$ é o valor da heurística no estado s e $V(s)$ o valor da *feature* escalado entre 0 e 1. O conjunto de heurísticas é testado em um torneio interno, para que se possa haviliar qual é a melhor.

4.3 Fluxplayer

O método demostrado acima é apenas uma das possíveis abordagens para a construção da Heurística. O Fluxplayer ([Schiffel and Thielscher 2006]), ganhador da competição realizada em 2006, combina as *features* extraídas da descrição com uma heurística formulada usando lógica *fuzzy*. A ideia central da função de avaliação é ver quão próximo o estado atual está de atingir as condições impostas pelas regras *goal* e *terminal*. Os valores para essas regras são atribuidos de forma que os estados terminais sejam evitados enquanto os objetivos (*goal*) não sejam preenchidos.

Seja um mundo de blocos onde temos três blocos, *a*, *b* e *c* e um objetivo *g = on(a,b) ∧ on(b,c) ∧ ontable(c)*. A heurística avalia quais condições do objetivo são satisfeitas no estado atual, resultando em um valor 0 ou 1. Entretanto, enquanto os pré-requisitos não forem satisfeitos, a função retornará sempre o valor 0. A solução para esse problema é descrita no artigo original.

5 Conclusão

O novo ramo da Inteligência Artificial, denominado General Game Playing, apresenta problemas que circundam várias áreas de pesquisa. Diferente dos jogadores clássicos, os jogadores generalistas são construídos sem qualquer conhecimento prévio do jogo. Este cenário gera desafios para os pesquisadores, como deinferir as regras do jogo e heurísticas plausíveis.

A metodologia para retirar características da descrição do jogo e construír a heurística, é o fator determinante para o sucesso de um jogador. Outras áreas podem contribuir para esta função, como o uso de lógica *fuzzy* pelo Fluxplayer.

Referências

CAMPBELL, M.; JR., A. J. H., AND HSU, F. H. 2002. Deep blue. *Artificial Intelligence 134*, 1–2, 57–83.

KUHLMANN, G.; STONE, P., AND DRESNER, K. 2006. Automatic heuristic construction in a complete general game player. *Proceedings of the Twenty-First National Conference on Artificial Intelligence*.

SCHAEFFER, J.; CULBERSON, J., AND SZAFRON, D. 1992. A world championship caliber checkers program. *Artificial Intelligence 53*, 2–3, 273–289.

SCHIFFEL, S., AND THIELSCHER, M. 2006. Fluxplayer: A successful general game player. *Artificial Intelligence*.

TESAURO, G. 1994. Td-gammon, a self-teaching backgammon program, achieves masterlevel play. *Neural Computation 6*, 215–219.