Problemas de Satisfação de Restrição Em Jogos Determinísticos

Victor Ribeiro Prata* Ajalmar Rêgo da Rocha Neto Raissa Leandro de Sousa
Fagner José de Matos Macêdo

Instituto Federal de Educação, Ciência e Tecnologia do Ceará, Departamento de Computação, Brasil

RESUMO

O uso de jogos eletrônicos como ambiente de desenvolvimento de agentes inteligentes tem se mostrado uma ótima alternativa para realizar experimentos com novos algoritmos. Metodologias sofisticadas tem sido utilizadas para o desenvolvimento de agentes inteligentes em jogos eletrônicos, porém técnicas tradicionais de inteligência artificial também podem ser adequadamente utilizadas com o mesmo propósito conseguindo resultados equivalentes ou possivelmente superiores. Com o objetivo de apresentar uma alternativa simples e bastante funcional ao desenvolvimento de agente inteligentes em jogos, neste trabalho foi investigado e desenvolvido um agente inteligente baseado em Problemas de Satisfação de Restrições utilizando algoritmos de resolução de problemas por meio de busca capaz de aprender e passar das fases iniciais do clássico jogo de Nintendo Super Mario World.

Palavras-chave: Problemas de Satisfação de Restrições, Inteligência Artificial, Jogos Eletrônicos, Agentes Inteligentes.

1 INTRODUÇÃO

Um dos maiores desafios da inteligência artificial é desenvolver agentes inteligentes capazes de aprender sobre o ambiente em que se encontram e como realizar tarefas nesses ambientes com pouco conhecimento inicial [5]. O campo dos jogos eletrônicos tem-se mostrado uma ótima opção para investigar e desenvolver técnicas de inteligência artificial (IA) por suprir a dificuldade que a área de desenvolvimento de agentes inteligentes tem enfrentado devido a falta de ambientes de teste para algoritmos inteligentes [1].

Desenvolver agentes inteligentes capazes de aprender sobre o ambiente e com pouco conhecimento inicial não é uma tarefa fácil. Diversas metodologias podem ser encontradas na literatura, mas em geral, o desempenho dos algoritmos é inferior ao de um ser humano. Sendo assim, o estudo de novas metodologias que supram as deficiências ainda existentes nessa área ou que complementam as técnicas já existentes é de grande importância para esse ramo da inteligência artificial.

Metodologias que utilizam algoritmos neuroevolutivos [5, 1] e algoritmos genéticos [13] tem sido bastante utilizadas, visto que, tais técnicas são baseadas no aprendizado por experiência. Assim, são gerados diversos agentes inteligentes, cada um experimentando uma abordagem própria para o problema e sendo avaliado segundo o seu desempenho no jogo, baseado no aprendizado adquirido, assim, o agente com o melhor desempenho é escolhido como a solução.

Agentes baseados no aprendizado por experiência conseguem obter resultados satisfatórios para jogos estocásticos, visto que o aprendizado por experiência permite o agente encontrar uma solução para um problema nunca visto antes, caso seja semelhante a algo que ele aprendeu. Entretanto, para jogos determinísticos, como os jogos de consoles clássicos, algoritmos mais tradicionais conseguem solucionar os desafios apresentados em tais jogos de maneira mais satisfatória e eficiente, caso a solução seja tratada como uma sequência de comandos que levam o jogador de um estado inicial a um estado final.

Por se tratar de uma área de pesquisa relativamente nova, há várias estratégias clássicas de inteligência artificial ainda não estudadas e que podem ser adaptadas para solucionar problemas em jogos eletrônicos. Com esse propósito, foi escolhido um clássico da Nintendo para o desenvolvimento de um agente, no qual seria capaz aprender e passar das fases do jogo utilizando uma técnica clássica da IA, o Problema de Satisfação de Restrição (PSR).

2 TRABALHOS RELACIONADOS

Diversos trabalhos que estudam o uso de técnicas de inteligência artificial aplicadas a jogos eletrônicos podem ser encontradas na literatura. Dentre esses jogos eletrônicos, os de vídeo games clássicos, como Super Mario World e Super Mario Bros, possuem um certo atrativo para os pesquisadores de agentes inteligentes por terem uma interface de ação simples o suficiente para o aprendizado dos agentes, mas lógicas suficientemente complexas para torna-los desafiadores para os agentes [5].

Em seus experimentos, [10] utilizou e comparou seis métodos diferentes para simular o comportamento humano em uma adaptação do jogo da Nintendo Super Mario Bros desenvolvido para o *Mario AI Championship*[1], o *Mario AI Benchmark*[2]. Três dos métodos utilizados foram implementados especificamente para o estudo, sendo eles baseados em *backpropagation* (*Supervised learning*), neuroevolução (*Neuroevolution*) e *scripting* dinâmico (*Dynamic scripting*). Nos experimentos, parte dos agentes foram treinados em dados de jogabilidade humana e o restante foram tentativas simples ou sofisticadas de desenvolver agentes ótimos. Entretanto, os autores concluem que nenhum dos agentes foi julgado ser tão humano quanto um humano.

Em [6] é proposto uma máquina de estados finitos apropriada para o jogo *Infinite Mario Bross*[3] (ligeiramente diferente do jogo de plataforma clássico da Nintendo, o Super Mario Bros). O Algoritmo Genético (AG) é usado juntamente com a máquina de estados finitos proposta para desenvolver um agente que é capaz de passar por alguns níveis do jogo. Os resultados dos experimentos mostraram que a máquina de estados finitos evoluiu com o AG, sendo capaz de passar por pelo menos 3 níveis diferentes do jogo.

O uso de algoritmos clássicos de inteligência artificial também tem se mostrado uma ótima alternativa no desenvolvimento de agentes inteligentes para solucionar jogos em geral.

Em [8], os autores utilizaram algoritmos de PSR associados à abordagem de têmpora simulada na proposição de três heurísticas capazes de solucionar, de formar eficiente, o quebra-cabeça Sudoku. Segundo os autores, como o espaço de busca das soluções é enorme, as heurísticas propostas tentam reduzir o espaço de busca das soluções através dos algoritmos de satisfação de restrição, aumentando a eficiência da heurística e permitindo assim resolver

*e-mail: victorrprata@gmail.com

[1] http://www.marioai.org/

[2] Disponível para download em: http://www.marioai.org/gameplay-track/marioai-benchmark

[3] Disponível para download em: https://infinitemariobros.codeplex.com/

um numero maior de instâncias em menos tempo. Os resultados mostraram que o uso de PSR acelerou o tempo necessário para as heurísticas solucionar o problema do quebra-cabeça, expondo assim a capacidade dos algoritmos de PSR em otimizar algoritmos de buscas.

Além dos trabalhos científicos publicados, é possível encontrar em diversos sites ou blogs[4] [5], experimentos que utilizam inteligência artificial para criar agentes inteligentes que consigam aprender a jogar jogos eletrônicos, nos quais é disponibilizado o código fonte para o público interessado em usar ou avaliar.

[12] utilizou o algoritmo neuroevolucional NEAT no desenvolvimento de um agente inteligente capaz de aprender a chegar ao final de uma fase no jogo Super Mario World, sem conhecimento específico da fase. Ao longo de diversas gerações o agente é evoluído, e a partir da 34º geração todos os indivíduos conseguem aprender o caminho correto para conseguir chegar ao final da fase. O autor disponibilizou um vídeo[6] demonstrativo e o código fonte[7] do experimento.

3 Ambiente do Super Mario World

O Super Mario World foi um jogo desenvolvido e publicado pela Nintendo em 1990 para o Super Nintendo Entertaiment System. Super Mario World é um jogo 2D de plataforma, em que o objetivo geral consiste em chegar ao final de cada fase do jogo. Os comandos possíveis do *joystick* do console são: (A) pula e gira , (B) pula e (X)/(Y) corre, interage e habilidades especiais.

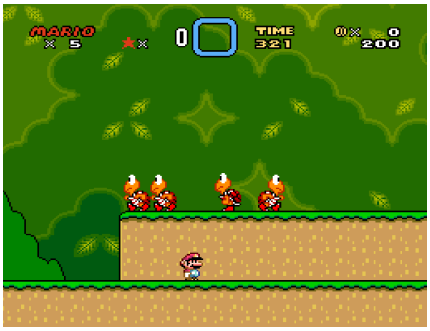

Figura 1: Super Mario World

Para alcançar o objetivo do jogo, o jogador deve realizar uma sequência de movimentos que leva o personagem do jogo até o final da fase. Tendo em vista que cada movimento é gerado por um comando vindo do *joystick* do console, e levando em consideração que o ambiente do jogo é determinístico, pode-se dizer que cada desafio do jogo pode ser completado através de uma sequência de comandos que levará o personagem de um estado inicial à um estado final. Diversas sequências de comandos podem levar o jogador a atingir o objetivo final, e diversas outras podem ser realizadas, mesmo que não levem o jogador a atingir o objetivo. Assim, uma fase do jogo pode ser representada como um estado local disposto de um conjunto de variáveis (ações) que podem assumir qualquer valor (comandos), e que torna-se um estado de aceitação quando os valores atribuídos às variáveis satisfazem todas as restrições sobre essa variável. Tal representação pode ser tratada como um Problema de Satisfação de Restrição (PSR), de modo que podemos utilizar algoritmos de busca PSR aproveitando a estrutura de estados e utilizando heurísticas de propósito geral para permitir solucionar os problemas de cada fase do jogo.

[4]https://www.gamedev.net/forums/forum/6-artificial-intelligence/
[5]http://aigamedev.com/
[6]Disponível em: www.youtube.com/watch?v=qv6UVOQ0F44
[7]Disponível em: https://pastebin.com/ZZmSNaHX

4 Problemas de Satisfação de Restrições

Segundo [9], muitos algoritmos foram desenvolvidos para resolver problemas de satisfação de restrição (PSR). Podemos ver isto tanto em problemas de construção de horários de escolas [2] como de uma universidade [15], problemas de alocação de recursos de rede virtual [4], customização de pacotes de serviços de telefonia móvel para clientes [14], e problema de operação de ilha em linha no caso de um apagão em uma rede de distribuição de energia com Geradores Distribuídos [7].

Um problema de satisfação de restrição é definido por três conjuntos, X, D e C, onde

- $X = \{X_1, ..., X_n\}$ é um conjunto de variáveis,
- $D = \{D_1, ..., D_n\}$ é um conjunto de domínios, um para cada variável,
- $C = \{C_1, ..., C_m\}$ é um conjunto de restrições que define possíveis combinações de valores.

Cada domínio D_i constitui-se de um conjunto de valores possíveis, $\{v_1, ..., v_k\}$ para a variável X_i.

Cada restrição $C_j \in C$ é um par <escopo, rel>, onde escopo é uma tupla de variáveis que participam da restrição e rel é uma relação que define os valores que cada variável pode assumir [11].

A questão é se existe uma solução para <X,D,C> que é um mapeamento que atribui um valor de D_i para toda variável x_i, de modo que para cada restrição C_j , a imagem do escopo da restrição seja um membro dessa relação da restrição [16].

4.1 Backtracking

Backtracking é uma busca em profundidade que atribui valores para uma variável de cada vez e que efetua um retrocesso quando uma variável não tem valores válidos restantes a serem atribuídos de acordo com as restrições da variável [11]. Para cada variável não atribuída, são experimentados todos os valores no domínio da variável, buscando encontrar uma solução. Entretanto, se for encontrada alguma inconsistência, o processo retorna para a chamada anterior, que tentará outro valor.

O backtracking tem três fases: a fase de ir à frente, onde a próxima variável na ordem é selecionada e colocada como variável atual; A fase em que a solução parcial atual é estendida atribuindo um valor consistente à variável atual, se existir; E a fase de retrocesso, que ocorre quando o valor não é consistente e a variável atual retorna à variável anterior [3].

4.2 Inferência em PSR

Os algoritmos de busca em PSR podem realizar a busca de duas maneiras: a maneira comum, escolhendo uma nova atribuição para a variável entre várias possibilidades ou através da inferências de propagação de restrições, utilizando-se das restrições para reduzir o número de valores válidos para uma variável [11], o que torna a busca por uma solução mais eficiente.

Para tanto, cada variável é tratada como um nó em um grafo e cada restrição binária (restrição que relaciona duas variáveis) é tratada como um arco. Além disso uma variável X_i pode ser considerada arco-consistente com relação à outra variável X_j se para cada valor no domínio de X_i existir algum valor no domínio de X_j que satisfaça a restrição binária sobre o arco (X_i, X_j).

5 PSR aplicado ao ambiente do Super Mario

Conforme visto na seção 4, um problema de satisfação de restrição é composto de três componentes: um conjunto de variáveis, um conjunto de domínios e um conjunto de restrições que especificam combinações de valores possíveis; assim, a representação do ambiente do Super Mario World como um problema de satisfação de restrição é dado da seguinte maneira: Variáveis (cada ação a ser

realizada pelo personagem); Domínio (comandos disponíveis que movimentam o personagem) e Restrições (restrição a determinados movimentos em certas partes do jogo).

O conjunto de variáveis é a sequência de ações realizadas durante a execução de uma determinada fase do jogo, enquanto que o domínio são os comandos que resultam nos movimentos disponíveis do personagem, assim, cada movimento é atribuído a uma ação, de modo que o conjunto de ações ao final da fase representem todos os movimentos realizados pelo personagem. Entretanto, para o personagem atingir o objetivo de chegar ao final da fase, os movimentos atribuídos a cada ação não podem ter restrições para aquela ação.

Os comandos selecionados para representar o domínio do PSR foram divididos em duas categorias: os comandos de deslocamentos, responsáveis por movimentar o personagem para direita (*RIGHT*); e os comandos de efeitos, responsáveis por gerar movimentos como pular (*JUMP*), que movimenta o personagem para cima por um curto período de tempo e pular para frente(*JUMP+RIGHT*), que além de movimentar o personagem para cima, move-o para frente. Os demais comandos não foram utilizados, devido ao fato de que, em geral, andar para frente e pular é tudo que o personagem precisa para chegar ao fim das fases iniciais do jogo.

De modo geral, as restrições do Super Mario World se resumem em basicamente duas restrições: um comando não pode levar o personagem a morte e quando o personagem encontrar alguma barreira à sua frente, o movimento de ir para frente ou apenas pular não deve ser escolhido, visto que esses movimentos não resultam em uma ação concreta. Assim, as restrições de cada ação dependem do ambiente em volta do personagem.

As restrições são adicionadas durante a busca em profundidade através da inferência de verificação à frente. Assim, sempre que uma ação *X* é atribuída ao personagem, o processo de verificação à frente estabelece a consistência de arco para ela: para cada ação não atribuída *Y* que está conectada por uma restrição a *X*, exclui do domínio de *Y* qualquer comando que leve a uma ação que esteja inconsistente com a ação escolhida para *X*. Desta maneira, quando houver uma barreira a frente do personagem, a inferência de verificação a frente perceberá o obstáculo e colocará as restrições ao comando *RIGHT* e *JUMP* para a próxima ação, eliminando assim esses comandos do domínio da próxima ação.

Em alguns casos não é possível olhar a frente e verificar se há obstáculos ou algo que impedirá o personagem de seguir o seu caminho. Tais casos ocorrem quando o personagem cai em uma fenda ou atinge algum inimigo, o que leva a morte do personagem. Nesses casos, a restrição ao comando é adicionada após a execução do mesmo. O algoritmo retorna uma ação a cada chamada, de modo que apenas um comando é executado por vez (com exceção do comando *JUMP+RIGHT*, que é a junção de dois comandos).

Com o ambiente do jogo representado como um PSR, foi escolhido o algoritmo de backtracking modelado sobre a busca recursiva em profundidade para o PSR, pois, como visto na seção 4.1, caso algum valor atribuído para a variável leve a um ponto sem saída (morte do personagem), o algoritmo desfaz as últimas atribuições das ações uma a uma até encontrar uma alternativa que leve o personagem adiante.

6 EXPERIMENTOS

Para realização dos experimentos, utilizou-se o simulador de jogos Bizhawk[8] para emular o jogo Super Mario World. Os algoritmos desenvolvidos foram implementado na linguagem de programação Lua, tendo em vista que é uma linguagem aceita por diversas ferramentas de simulação de jogos, inclusive o Bizhawk.

O algoritmo desenvolvido tem como objetivo final encontrar uma sequência de comandos, a partir da busca feita pelo PSR, que levem o personagem ao fim da fase, assim, ao chegar no final de uma fase, a sequência de comandos executados que levou ao objetivo é salvo em um arquivo para poder ser reproduzida novamente e o algoritmo é encerrado.

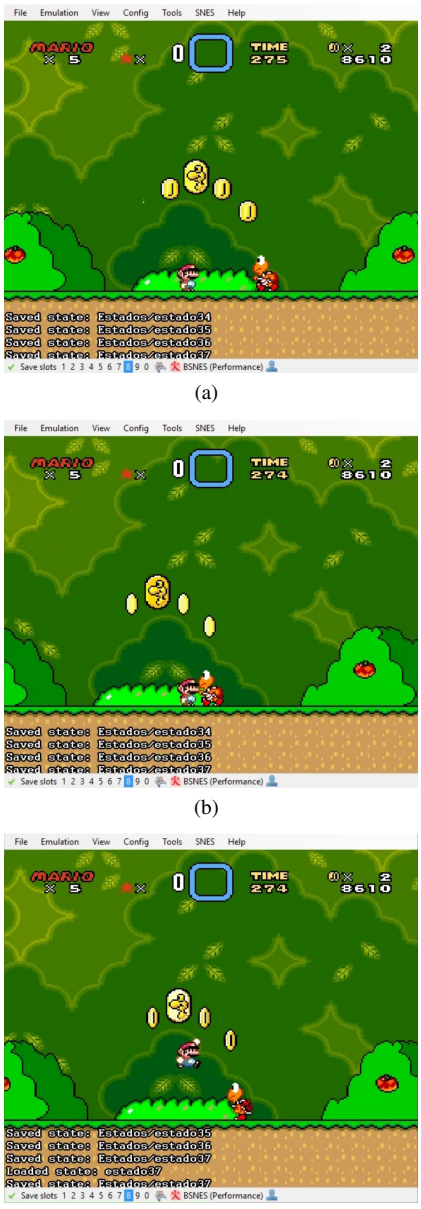

Figura 2: (a) Antes do comando *RIGHT* ser escolhido. (b) Comando *RIGHT* fez o personagem morrer. (c) Retrocedeu ao estado anterior ao comando *RIGHT* e escolheu *JUMP+RIGHT*, que fez o personagem pular o inimigo

Antes de cada movimento ser executado, o estado do jogo é salvo, permitindo ao algoritmo realizar o retrocesso e voltar para o estado anterior a um movimento quando necessário. A Figura 2 demonstra o processo de retrocesso, onde o comando *RIGHT* levou o personagem a morrer por tocar o inimigo e, como citado na seção 5, quando o personagem morrer, o processo de backtracking volta ao estado anterior aquele comando e escolhe o próximo comando, que nesse caso foi o *JUMP+RIGHT*, livrando o personagem do inimigo. A duração de cada movimento é equivalente a aproximada-

[8]Disponível para download em: http://tasvideos.org/BizHawk.html

mente o tempo que o personagem leva para realizar completamente o movimento de pular.

Para os experimentos, o algoritmo foi executado 50 vezes na primeira fase do Super Mario World. Em nenhum dos experimento o algoritmo falhou, ou seja, sempre conseguiu chegar ao fim da fase antes do tempo da fase acabar. Os resultados do desempenho do agente durante o processo de aprendizagem e o desempenho das soluções encontradas podem ser analisados no gráfico da Figura 6. Em vermelho está representado o tempo necessário para o agente encontrar uma solução válida em cada execução do algoritmo. Em azul está representado o tempo que o personagem levou para executar a solução encontrada em cada execução do algoritmo. A Tabela 1 exibe o desempenho máximo, médio, mínimo e o desvio padrão que o algoritmo levou para obter uma solução e para executa-la após encontrada.

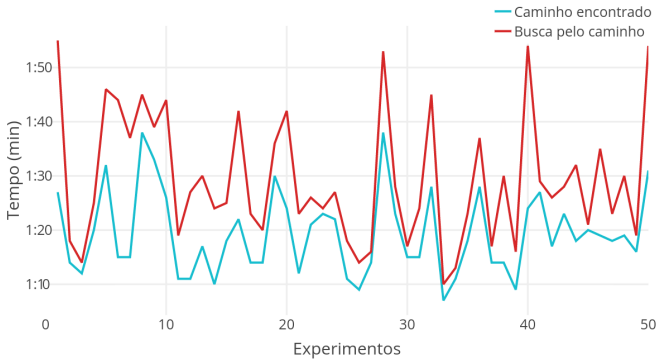

Figura 3: Desempenho do agente nos experimentos

Tabela 1: Estatísticas dos experimentos

	Tempo (min)
Pior desempenho na busca	01:55
Média desempenho na busca	01:29
Melhor desempenho na busca	01:10
Desvio padrão	00:11
Pior caminho encontrado	01:38
Média dos caminhos encontrados	01:19
Melhor caminho encontrado	01:07
Desvio padrão	00:07

Em comparação ao trabalho realizado por [12], que utiliza o algoritmo NEAT, este trabalho mostrou-se melhor no tempo que o agente leva para aprender a solucionar a fase, tendo em vista que para que o agente implementado com NEAT consiga gerar uma geração no qual todos os indivíduos consigam aprender a sequência correta de comandos para chegar ao fim da fase, são necessárias 34 gerações, o que demanda um tempo altíssimo. Entretanto, em relação ao tempo que o melhor agente utilizado em [12] consegue finalizar a fase é mais rápido, por utilizar todos os comandos disponíveis para o jogo, enquanto este trabalho utilizou apenas três comandos.

7 Conclusão

Os experimentos realizados mostraram que os resultados são satisfatórios, tendo em vista que o personagem conseguiu concluir a fase inicial do Super Mario em um tempo pouco acima de um ser humano com pouca experiência, e cada iteração realizada para se obter a melhor sequência de comandos levou em torno de 1 minuto e meio. Assim, este trabalho mostrou que técnicas clássicas são uma boa alternativa a jogos com ambientes determinísticos, já que exigem tempo e custo computacional menor que técnicas mais sofisticadas, como redes neurais e algoritmos genéticos.

Referências

[1] K. S. M. Araujo and F. O. França. Um ambiente de jogo eletrônico para avaliar algoritmos coevolutivos. In *Congresso Brasileiro de Inteligência Computacional*, pages 1–6, Curitiba, 2015.

[2] J. J. Blanco and L. Khatib. Course scheduling as a constraint satisfaction problem. In *In Proc. PACT98*, 1998.

[3] R. Dechter and D. Frost. Backtracking algorithms for constraint satisfaction problems - a tutorial survey. Technical report, 1998.

[4] F. Guenane, P. Y. Dumas, M. Nogueira, and G. Pujolle. Solving virtual network resource allocation problems using a constraint satisfaction problem model. In *2013 Fourth International Conference on the Network of the Future (NoF)*, pages 1–5, Oct 2013.

[5] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone. A neuroevolution approach to general atari game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(4):355–366, Dec 2014.

[6] N. C. Hou, N. S. Hong, C. K. On, and J. Teo. Infinite mario bross ai using genetic algorithm. In *2011 IEEE Conference on Sustainable Utilization and Development in Engineering and Technology (STUDENT)*, pages 85–89, Oct 2011.

[7] X. Li, Y. Xu, L. Zhang, and S. Gao. On-line islanding operation based on csp. In *2010 Asia-Pacific Power and Energy Engineering Conference*, pages 1–3, March 2010.

[8] M. C. Machado and L. Chaimowicz. Combining metaheuristics and csp algorithms to solve sudoku. In *2011 Brazilian Symposium on Games and Digital Entertainment*, pages 124–131, Nov 2011.

[9] B. A. Nadel. Constraint satisfaction algorithms1. *Computational Intelligence*, 5(3):188–224, 1989.

[10] J. Ortega, N. Shaker, J. Togelius, and G. N. Yannakakis. Imitating human playing styles in super mario bros. *Entertainment Computing*, 4(2):93 – 104, 2013.

[11] S. Russel and P. Norving. *Inteligência artificial: tradução da terceira edição*, chapter Problemas de satisfação de restrições, pages 249–279. Elsevier Editora Ltda, Rio de Janeiro, third edition, 2013.

[12] SethBling. Mari/o - machine learning for video games. https://www.youtube.com/watch?v=qv6UVOQ0F44. Acessado: 02/04/2017.

[13] M. Silva and R. Martins. Comparação entre multilayer perceptron e redes neurais lógicas no problema presa-predador usando algoritmos genéticos. In *Seminário de Formação Científica e Tecnológica*, pages 13–14, Ijuí, 2015.

[14] D. Yang. A csp approach for customizing mobile service bundles. In *2010 2nd International Conference on Information Engineering and Computer Science*, pages 1–3, Dec 2010.

[15] L. Zhang and S. Lau. Constructing university timetable using constraint satisfaction programming approach. In *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, volume 2, pages 55–60, Nov 2005.

[16] D. Zhuk. An algorithm for constraint satisfaction problem. In *2017 IEEE 47th International Symposium on Multiple-Valued Logic (ISMVL)*, pages 1–6, May 2017.