An analysis of reward shaping for reinforcement learning in a multi-agent framework for combinatorial optimization.

Jardell Fillipe da Silva[1,2], **Maria Amélia Lopes Silva**[1,2], **Sérgio Ricardo de Souza**[1]

[1]Programa de Pós-Graduação em Modelagem Matemática e Computacional
Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG)
Av. Amazonas 7675, CEP 30.510-000, Belo Horizonte (MG), Brasil

[2]Instituto de Ciências Exatas e Tecnológicas - Campus Florestal
Universidade Federal de Viçosa (UFV) - Florestal, MG - Brasil

{jardell.silva,mamelia}@ufv.br, sergio@cefetmg.br

Abstract. *This article evaluates reward shaping for the reinforcement learning algorithm used by a multi-agent framework for combinatorial optimization. This evaluation consists of six different reward shaping scenarios, applied to a set of identical agents built on the framework, all implementing the Iterated Local Search (ILS) metaheuristic, to solve the Vehicle Routing Problem with Time Windows (VRPTW). Computational tests, applied to VRPTW instances from the literature, showed that the results obtained for the various shapes of reward are comparable in terms of the quality of the objective function values achieved, the execution time, and the learning speed to results already existing in the literature. The conclusions show that it is possible to define a reward shaping with greater autonomy concerning the knowledge degree of the addressed problem and, at the same time, efficient in terms of runtime and learning speed.*

Resumo. *Este trabalho avalia modelagens de recompensas para o algoritmo de aprendizado usado por um framework multiagente para otimização combinatória. Esta avaliação consiste em seis cenários diferentes de modelagem de recompensas, aplicados a um conjunto de agentes idênticos construídos no framework, que implementam a metaheurística Iterated Local Search (ILS) para a solução do Problema de Roteamento de Veículos com Janelas de Tempo. Testes computacionais, aplicados a instâncias da literatura, mostram que os resultados obtidos para as diversas formas de recompensa são comparáveis quanto a qualidade dos valores de função objetivo alcançados, ao tempo de execução, e à velocidade de aprendizado frente a resultados já existentes na literatura. As conclusões mostram que é possível definir uma forma de aprendizado que seja autônoma quanto ao conhecimento do problema objeto de interesse e eficiente no que diz respeito a tempo computacional e velocidade de aprendizado.*

1. Introdução

Maximizar a recompensa de um agente, atendendo às diversas finalidades para as quais sua inteligência é direcionada, é suficiente para expressar uma ampla variedade de objetivos contidos neste agente. [Silver et al. 2021] discutem se as distintas formas de inteligência podem estar sujeitas à maximização de recompensas e se as várias habilidades,

associadas às formas de inteligência do agente, podem surgir subjacente da busca de recompensas, através da atuação do agente em seu ambiente. [Silver et al. 2021], além disso, discutem a importante questão de como construir um agente que maximize a recompensa e indica que a resposta pode ser alcançada através da habilidade do agente em aprender a maximizar sua recompensa, mediante sua experiência contínua de interação com o ambiente. O aprendizado do agente através de suas experiências com o ambiente é a forma mais natural de maximizar suas recompensas.

Em relação à discussão sobre aprendizado por reforço e modelagem de recompensas, [Agogino and Tumer 2008] discutem sobre a importância de analisar a eficácia das estruturas de recompensas do agente, além de destacarem que o sucesso da política de um bom comportamento é fortemente dependente do domínio. Avaliam também que a modelagem potencial é fortemente ligada ao domínio do problema e que a modelagem de diferença, não necessariamente, requer um forte domínio do problema. Em [Devlin et al. 2014], são discutidas e implementadas modelagens de recompensas baseadas em diferenças, em potencial e uma hibridização destas duas. [Grzes 2017] discutem técnicas de aprendizado por reforço que podem resolver problemas de grande escala, levando a uma tomada de decisão autônoma e de alta qualidade. Porém, algumas destas técnicas de aprendizagem por reforço podem ser demoradas, devido à complexidade em recompensar os agentes de forma eficiente.

Assim, o principal objetivo deste artigo é avaliar o comportamento de um *framework* multiagente para solução de problemas de otimização combinatória quanto à modelagem de recompensas do seu algoritmo de aprendizagem por reforço. Para isso, este trabalho utiliza-se de uma aplicação proposta para a solução do Problema de Roteamento de Veículos com Janelas de Tempo (PRVJT). As características específicas do problema foram inseridas no ambiente do sistema multiagente, fazendo uso das facilidades apresentadas pelo *framework*. O propósito desta experimentação é demonstrar que a modelagem de recompensas pode facilitar o desenvolvimento destas aplicações, pois definir recompensas reais e eficientes pode requerer um conhecimento mais profundo do problema. A partir daí, testes computacionais e estatísticos foram realizados para avaliar estes modelos. Para isso, busca-se (i) analisar os dados individual e unificadamente, com o intuito de avaliar tendências no comportamento dos dados em relação ao valor da função objetivo e ao tempo computacional; (ii) realizar testes com intuito de verificar a existência de diferenças significativas entre as médias dos modelos criados; (iii) comparar o tempo de aprendizado dos diferentes modelos de recompensas; por fim, (iv) validar as modelagens de recompensas como formas mais autônomas de aprendizagem dos agentes.

O restante deste artigo está estruturado da seguinte forma: a Seção 2 apresenta uma revisão conceitual de aprendizado por reforço e de modelagem de recompensas. A Seção 3 descreve os experimentos computacionais. Na Seção 4 são apresentados e discutidos os resultados computacionais. A Seção 5 conclui o artigo.

2. Caracterização do Problema

O problema de aprendizagem por reforço [Sutton and Barto 2018] representa alcançar metas através de recompensas cumulativas. Uma recompensa é uma observação escalar especial R_t, emitida a cada passo de tempo t por um sinal de recompensa no ambiente, que fornece uma medida instantânea do progresso em direção a um objetivo. Uma instância

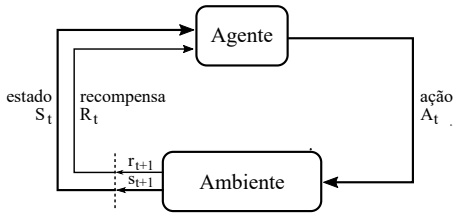


Figura 1. Modelo de Aprendizado por Reforço. [Sutton and Barto 2018].

do problema de aprendizado por reforço é definida por um ambiente, um agente, um estado S_t, um sinal de recompensa R_t, uma ação A_t e por um objetivo cumulativo a se maximizar, conforme apresentado na Figura 1.

Na Figura 1 observa-se que a ação A_t altera o estado do ambiente e o valor dessa transição de estado é comunicado ao agente através de um sinal de reforço R_{t+1}. A próxima ação do agente é dada através do estado S_{t+1}. O comportamento do agente consiste em escolher ações que tendem a aumentar a soma dos valores de reforço a longo prazo. Um algoritmo de aprendizado por reforço bem difundido e de importante relevância na literatura é o *Q-learning* [Watkins and Dayan 1992]. Este algoritmo consiste em encontrar uma política ótima de tomada de decisão, não requer um modelo do ambiente e pode tratar problemas transitórios e estocásticos sem modificações. A equação de atualização do valor estado/ação da tabela Q do *Q-learning* em um determinado tempo t é dada por:

$$Q(s,a) = Q(s,a) + \alpha \left[r + \gamma \max_{\alpha'} Q(s',a') - Q(s,a) \right] \quad (1)$$

em que a é a ação tomada e r corresponde ao valor de recompensa de mudar do estado atual s para o estado resultante s'. O parâmetro α representa a taxa de aprendizagem e γ é o fator de desconto. Estes dois parâmetros dependem diretamente do problema tratado. A função $Q(\cdot,\cdot)$ estima a utilidade esperada da tomada de uma ação em um determinado estado s. Um episódio é definido aqui como uma sequência de estados, que vão desde um estado inicial até um estado final. A partir destes valores, um mapeamento de política de estado/ação é gerado. Para equilibrar os requisitos de exploração e intensificação, um método de seleção de ação é determinado por $\epsilon - greedy$. Este método escolhe a ação de maior valor para o estado atual com probabilidade $1-\epsilon$ e escolhe uma ação aleatória com a probabilidade restante [Sutton and Barto 2018].

O aprendizado por reforço pode consumir tempo computacional, pois algoritmos de aprendizagem precisam determinar as consequências de suas ações a longo prazo utilizando *feedback* ou recompensas atrasadas. A modelagem de recompensas é um método de incorporar o conhecimento do domínio no aprendizado por reforço para que os algoritmos sejam guiados mais rapidamente para soluções mais promissoras [Grzes 2017]. No entanto, modelar estas recompensas pode requerer conhecimento prévio do domínio ou uma definição de técnicas de modelagem. [Devlin et al. 2014] mostram que modelagens de recompensas por diferenças e modelagens de recompensas baseadas em potencial podem melhorar significativamente a política conjunta aprendida por múltiplos agentes de aprendizado por reforço atuando simultaneamente no mesmo ambiente. No aprendizado por reforço de agente único, o agente tem como objetivo maximizar apenas sua recompensa. Porém, se estes agentes estão inseridos em um ambiente multiagente cooperativo, este aprendizado deixa de ser apenas direcionado ao individual e passa a ser avaliado a

partir de uma função global. As duas principais formas de se recompensar um agente é através de recompensas locais, representativas do estado do agente, ou recompensas globais, representativas do desempenho do sistema [Agogino and Tumer 2008]. Os modelos de aprendizado por reforço aqui avaliados são:

(i) **Recompensa Local**: uma recompensa local (L) é uma recompensa baseada na parte do sistema que representa o valor alcançado pelo agente de forma direta. O uso desse sinal de recompensa geralmente incentiva um comportamento "egoísta", no qual o agente pode agir apenas direcionando suas ações para sua observação direta. Este modelo de recompensa pode levar a uma tomada de decisão que pode ir contra o objetivo do sistema, pois, enquanto tenta cegamente aumentar sua própria recompensa, um desempenho indesejado do sistema pode ocorrer.

(ii) **Recompensa Global**: uma recompensa global (G) é dada através do valor de objetivo alcançado pelo sistema como forma de aprendizado. Isso encoraja o agente a agir no interesse do sistema, mas inclui uma quantidade substancial de ruído de outros agentes agindo simultaneamente. A própria contribuição de um agente para a recompensa global pode ser ofuscada pela contribuição de centenas de outros agentes, resultando em uma baixa relação sinal-ruído [Agogino and Tumer 2008].

(iii) **Recompensa por Diferenças**: uma forma de avaliar quanto o agente está contribuindo para um sistema como um todo é através de modelos de recompensas por diferenças. Os modelos de recompensas por diferenças capturam a contribuição de um agente para o desempenho do sistema. Isso pode se tornar difícil devido ao fato dos demais agentes estarem atuando no ambiente, obscurecendo a contribuição individual do agente para o objetivo do sistema [Devlin et al. 2014]. A recompensa por diferença (D) é um sinal de recompensa moldado que ajuda um agente a aprender as consequências de suas ações no objetivo do sistema, removendo grande parte do ruído criado pelas ações de outros agentes ativos no sistema [Agogino and Tumer 2008]. A modelagem de recompensa por diferenças é descrita em [Devlin et al. 2014] como $D(z) = G(z) - G(z - i)$, em que z é um termo geral representativo de estados ou pares estado-ação dependendo da aplicação, $G(z)$ é o termo global de desempenho do sistema, e $G(z - i)$ é $G(z)$ para um sistema teórico sem a contribuição do agente i. Qualquer ação tomada para aumentar D consequentemente aumenta G, enquanto o impacto do agente i em sua própria recompensa é muito maior que seu impacto relativo em G [Devlin et al. 2014]. Estas propriedades permitem que o modelo de recompensa por diferença aumente o desempenho do aprendizado em um sistema multiagente em um grau significativo. A recompensa por diferenças geralmente pode ser estimada, mesmo em domínios extremamente complexos. No entanto, no sistema em análise, D é diretamente calculável.

3. Descrição dos Experimentos

Esta seção descreve o experimento realizado para avaliação dos modelos de aprendizagem por reforço. A Subseção 3.1 descreve o *Framework* AMAM, adotado para a realização do experimento. A Subseção 3.2 apresenta uma descrição do problema instanciado usado para os experimentos, qual seja, o Problema de Roteamento de Veículos com Janela de Tempo (PRVJT). A Subseção 3.3 descreve as modelagens de recompensa em relação ao agente do *Framework* AMAM.

3.1. *Framework* AMAM

O *Framework* AMAM[1] [Silva et al. 2019] é uma arquitetura multiagente para metaheurísticas, na qual cada agente implementa uma metaheurística e age de forma a buscar uma solução para um problema. O espaço de busca do problema corresponde ao ambiente do sistema multiagente. Assim, com uma simples troca do ambiente, altera-se com facilidade o problema a ser tratado. A capacidade de movimentação de cada agente pelo ambiente se dá pelas formas de manipulação da solução que cada problema disponibiliza. A facilidade de inclusão de novos agentes na estrutura multiagente do *framework* garante a sua escalabilidade, e a adição desses agentes afeta minimamente a estrutura. Estes agentes possuem a capacidade de interagir e perceber o ambiente de forma cooperativa, além de possuírem capacidades adaptativas. Seus principais elementos são:

(i) `Ambiente`: o ambiente do *Framework* AMAM é o próprio espaço de busca do problema;

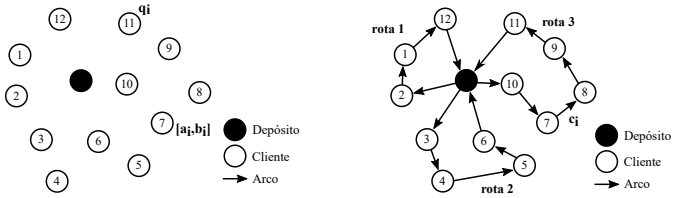
(ii) `Agente`: o agente no *Framework* AMAM incorpora uma metaheurística/metaheurística híbrida e tem a função de buscar a solução para um dado problema. O agente possui as seguintes habilidades de interação com o ambiente: (i) Percepção do Ambiente: característica que torna o agente capaz de enxergar parcialmente o ambiente ao qual faz parte; (ii) Posicionamento: capacidade de se posicionar no espaço de busca, seja através de uma nova solução ou pela escolha de uma solução existente; (iii) Movimento: condição de se movimentar pelo espaço de busca, de uma solução a outra, usando, por exemplo, estruturas de vizinhanças ou operadores genéticos (Algoritmos Genéticos). O agente apresenta também habilidades autoadaptativas [Silva et al. 2019]. As habilidades adaptativas permitem ao agente se adaptar às características específicas do problema de otimização, utilizando conceitos de aprendizado de máquina;

(iii) `Cooperação`: o *Framework* AMAM possui estruturas que permitem a interação e cooperação entre os agentes. A principal estrutura responsável pela troca de informações é denominada *Pool* de Soluções, no qual são compartilhadas soluções geradas pelos agentes no decorrer de sua busca. A estrutura de cooperação da AMAM tem, como principal objetivo, guiar os agentes por regiões mais promissoras no espaço de busca.

3.2. Problema de Roteamento de Veículos com Janela de Tempo (PRVJT)

No PRVJT [Toth and Vigo 2014], um conjunto $K = \{k : k = 1, 2, \ldots, |K|\}$ de veículos está localizado em um único depósito e deve servir a um conjunto $C = \{i : i = 1, 2, \ldots, N\}$ de clientes espalhados geograficamente. No caso aqui considerado, a frota de veículos é homogênea, ou seja, todos os veículos são iguais e possuem a mesma capacidade Q. Cada cliente i tem uma dada demanda q_i e deve ser atendido dentro de uma janela de tempo especificada $[a_i, b_i]$ (veja a Figura 2a). Uma solução para o PRVJT é um conjunto de rotas (veja a Figura 2b), em que cada rota é representada por uma lista ordenada de clientes, que determina a sequência na qual eles devem ser atendidos por um veículo. Os arcos mostram a conexão entre os clientes e têm um valor associado c_{ij}, que representa o custo de viagem entre o cliente

[1]A versão mais recente, lançada em outubro de 2020, está disponível em `https://github.com/mamelials/AMAM-Multiagente-Architecture-for-Metaheuristics.git`, sob a licença GNU LGPLv3.



(a) Elementos do PRVJT. **(b)** Uma solução do PRVJT.

Figura 2. Uma solução para o PRVJT [Silva et al. 2019].

i e o cliente j, que pode ser relacionado à distância entre os clientes. Na solução $x\,[0, 2, 1, 12, 0, 3, 4, 5, 6, 0, 10, 7, 8, 9, 11, 0]$ mostrada na Figura (2b), o índice 0 indica o depósito e $\mathit{rota}_1 = [0, 2, 1, 12, 0]$, $\mathit{rota}_2 = [0, 3, 4, 5, 6, 0]$ e $\mathit{rota}_3 = [0, 10, 7, 8, 9, 11, 0]$ são as três rotas desta solução. Assim, a solução x também pode ser descrita como $x = [\mathit{rota}_1, \mathit{rota}_2, \mathit{rota}_3]$.

O objetivo do PRVJT é determinar um conjunto de rotas para minimizar o custo total envolvido com essa operação. Cada rota está associada a um único veículo. As rotas devem começar e terminar no depósito. O custo de uma solução x é calculado como:

$$f(x) = \omega K(x) + \sum_{(i,j)\in E} c_{ij} \tag{2}$$

em que E é o conjunto de arcos pertencentes à solução x; $K(x)$ é o número de veículos na solução x; e ω é um fator de penalidade de alto valor, não negativo e arbitrário. Nesta função, a prioridade é minimizar o número de veículos (ou rotas, consequentemente). Em caso de empate no número de veículos, a distância total percorrida deve ser minimizada.

As estruturas de vizinhanças são responsáveis pela movimentação do agente no ambiente do *framework*. Para explorar o espaço de soluções, oito diferentes funções de vizinhança são usadas na instanciação do *Framework* AMAM para resolver o PRVJT:

(i) *Intra-Route Swap*: função de vizinhança que realiza o movimento de troca de um cliente com outro cliente da mesma rota;
(ii) *Inter-Route Swap*: função de vizinhança que realiza o movimento de troca de um cliente de uma rota com um cliente de outra rota;
(iii) *Intra-Route Shift*: função de vizinhança que realiza o movimento de realocação de um cliente para outra posição na mesma rota;
(iv) *Inter-Route Shift*: função de vizinhança que realiza a realocação de um cliente de uma rota para outra;
(v) *Two Intra-Route Swap*: função de vizinhança que consiste na troca de clientes na mesma rota, assim como na função de vizinhança *Intra-Route Swap*. No entanto, na função *Two Intra-Route Swap*, dois clientes consecutivos são trocados por dois outros clientes consecutivos da mesma rota;
(vi) *Two Intra-Route Shift*: função de vizinhança que consiste na realocação de clientes na mesma rota, assim como na função de vizinhança *Intra-Route Shift*. No entanto, na função *Two Intra-Route Shift*, dois clientes consecutivos são removidos de suas posições e reinseridos em outra posição da mesma rota;

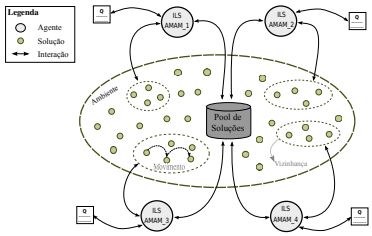


Figura 3. Instanciação do *Framework* `AMAM` para o PRVJT com 4 agentes ILS.

(vii) *Eliminates Smaller Route*: função de vizinhança que procura eliminar a menor rota da solução. A menor rota é definida como a rota que possui o menor número de clientes. Para isso, os clientes da menor rota da solução são removidos e reinseridos em outras rotas da solução;

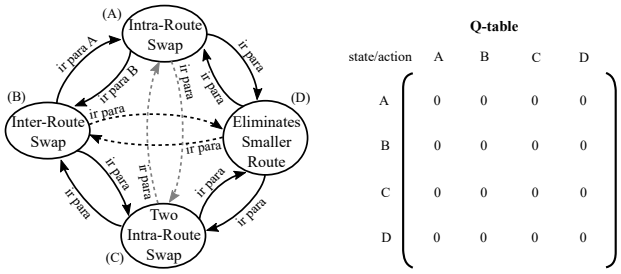
(viii) *Eliminates Random Route*: função de vizinhança *Eliminates Random Route* opera de forma semelhante à função *Eliminates Smaller Route*, mas a rota a ser excluída é escolhida aleatoriamente.

3.2.1. Instanciação do PRVJT no *Framework* `AMAM`

Para avaliar o comportamento da modelagem de recompensas no aprendizado de agentes no *Framework* `AMAM`, utiliza-se uma instanciação deste *framework*, mostrada em [Silva et al. 2019], na qual os agentes implementam, para a solução do PRVJT, uma metaheurística híbrida baseada na metaheurística *Iterated Local Search* (ILS) [Lourenço et al. 2003], com uma solução inicial baseada na fase de construção da metaheurística GRASP [Feo and Resende 1995] e busca local baseada no método VND [Mladenović and Hansen 1997]. Esses agentes são denominados `ILS_PRVJT`. A instanciação do *Framework* `AMAM` para o problema consiste em replicar agentes n idênticos `ILS_PRVJT`, conforme mostrado na Figura 3 para o caso de quatro agentes. O *Framework* `AMAM` possibilita uma estrutura de cooperação entre agentes, descrita em [Silva et al. 2019], de forma que os elementos do `Ambiente` (espaço de busca das soluções) sejam compartilhados entre todos os agentes. O `Ambiente` é passado por referência aos agentes para garantir esse compartilhamento, o que significa que todos os agentes têm acesso ao mesmo local na memória e que a modificação realizada por um agente fica imediatamente disponível para os demais. Pode-se observar que os agentes agem simultaneamente no `Ambiente` através da movimentação por meio das estruturas de vizinhanças. Observa-se também que os agentes têm acesso ao *Pool* de soluções de forma simultânea e ordenada. É importante ressaltar que a tabela Q é parte integrante do agente, e não está externa ao seu funcionamento.

3.3. Modelagem de recompensas no aprendizado do agente do *Framework* `AMAM`

A estrutura do *framework* possibilita atribuir aos agentes habilidades baseadas em conceitos de aprendizado de máquina. Para isso, o agente construído implementou um algoritmo *Q-Learning* de aprendizado por reforço. Este algoritmo, nomeado ALS-*QLearning*, é introduzido em [Silva et al. 2019] e consiste em aprender qual ordem de vizinhança o algoritmo `ILS_PRVJT` irá utilizar na busca local VND. Os resultados apresentados em [Silva et al. 2019] comprovam que a utilização do algoritmo de aprendizagem por reforço



(a) Gráfico estado/ação. **(b)** Tabela Q inicial.

Figura 4. Relação entre estados (funções de vizinhança) e ações possíveis [Silva et al. 2019].

alcançou melhores resultados que a estrutura quando as vizinhanças foram determinadas via complexidade.

O objetivo do aprendizado é avaliar o ganho obtido com a aplicação de uma sequência de duas vizinhanças e, a partir daí, premiar as melhores sequências e maximizar a recompensa acumulada. Cada vizinhança a ser utilizada pelo método de busca é considerada como um estado de aprendizagem. Desta forma, o conjunto de estados S é formado pelas funções de vizinhanças disponíveis, para que o problema seja tratado pelo *framework*. Para o PRVJT, o conjunto de estados é formado pelas funções de vizinhança S_{prvjt} = {*Intra-Route Swap*, *Inter-Route Swap*, *Intra-Route Shift*, *Inter-Route Shift*, *Intra-Route Swap*, *Two Intra-Route Shift*, *Eliminates Smaller Route*, *Eliminates Random Route*}. No conjunto de ações $A(s)$, uma ação é definida como a mudança de um estado para outro. Dessa forma, o conjunto de ações pode ser representado por um grafo completo, no qual cada ação é representada por um arco conectando dois estados (nós do grafo), como apresentado na Figura 4. Um exemplo de gráfico representando a relação entre estados (funções de vizinhança) e possíveis ações é mostrado na Figura 4a. Neste exemplo, apenas quatro funções de vizinhança do PRVJT são usadas para facilitar a visualização. A tabela Q, referente a este exemplo, é mostrada na Figura 4b. A tabela Q tem dimensões dadas por $M \times M$, em que M é o número de estados, ou seja, o número de funções de vizinhança do problema tratado.

Na Figura 4a são consideradas apenas quatro vizinhanças para o problema PRVJT, sendo o par estado/ação determinado. Já na Figura 4b, a tabela Q define uma tabela inicial. Essa tabela é responsável por armazenar os valores dos pares estado-ação no decorrer do processo de aprendizagem. Nesta proposta, cada elemento da tabela Q é inicializado com valor zero, de modo que o processo de aprendizagem seja realizado de forma não tendenciosa. Uma vez que as funções de vizinhança são parâmetros específicos do problema, o objetivo do uso do aprendizado por reforço é permitir que o *Framework* se adapte melhor às características próprias do problema.

Em [Silva et al. 2019], a recompensa r é determinada de forma paramétrica através de conhecimento especifico do problema. Diferentemente, o presente artigo propõe modelar a recompensa r do algoritmo ALS-QLearning. Para isso, seis cenários de recompensa foram criados e divididos em três categorias de modelagem de recompensas: locais (2 cenários), globais (1 cenário) e diferenciais (3 cenários). Os cenários

de recompensas locais são: (i) r_linha: recompensa que consiste em atribuir o valor da função objetivo corrente do método VND, como forma de recompensa para o agente; e (ii) r_local: recompensa que consiste em atribuir o valor da melhor função objetivo da busca local do agente, como forma de recompensa para o agente. O cenário de recompensa global é (i) r_global: recompensa que consiste em atribuir o valor da melhor função objetivo encontrada pelo sistema, como forma de recompensa para o agente. Por fim, os cenários baseado em recompensas por diferenças são: (i) rd_lilo: recompensa que consiste em atribuir o valor da diferença entre r_linha por r_local, como forma de recompensa para o agente; (ii) rd_ligl: recompensa que consiste em atribuir o valor da diferença entre r_linha por r_global, como forma de recompensa para o agente; (iii) rd_logl: recompensa que consiste em atribuir o valor da diferença entre r_local por r_global, como forma de recompensa para o agente. As modelagens de recompensas por diferença expressam a distância entre a solução corrente do agente e a melhor solução objetivo encontrada pelo *Framework* AMAM em determinado instante de tempo.

4. Experimentos Computacionais

Para investigar os resultados obtidos com a aplicação das modelagens de recompensas do algoritmo *QLearning* do agente no *Framework* AMAM, foi realizado um experimento no qual o algoritmo completo (ou seja, equipado com todas as seis modelagens de recompensas) foi comparado com os resultados obtidos utilizando a instanciação do AMAM apresentada em [Silva et al. 2019], na qual a recompensa r do agente é definida de forma paramétrica, a partir de conhecimento prévio do problema.

O *Framework* AMAM foi implementado em Java, com o `JDK 8`, utilizando a `IDE Eclipse`. Os resultados foram obtidos utilizando o *cluster* da Universidade Federal de Viçosa (UFV). Para isto utilizou-se de um computador (nó de cálculo) com processador AMD Opteron(tm) Processor 6376 (16M Cache, 2.3 GHz, 32 cores), totalizando 64 núcleos de execução, com o sistema operacional Linux CentOS. Cada agente do *Framework* foi executado em sua própria *thread* Java e em sua própria *thread* física (processador/núcleo). A composição do sistema multiagente usada nos experimentos envolveu agentes ILS idênticos. Como a máquina na qual foram executados os experimentos possui 64 núcleos, a cada execução, 60 núcleos foram utilizados pelos agentes e os outros 4 núcleos foram deixados à disposição da JVM do Java e do sistema operacional. Desta forma, os testes utilizam 60 *threads* em um cenário em que cada iteração implementa 4 execuções por vez dos cenários com um, dois, quatro e oito agentes. Todos os testes foram executados por 30 vezes. Finalmente, para este experimento foram utilizadas 6 instâncias do PRVJT, cada uma com 100 clientes [Solomon 1987], representando diferentes formas de distribuição geográfica dos clientes e largura de janela de tempo.

A Tabela 1 identifica as instâncias utilizadas e apresenta o gap($\%$) médio das 30 execuções para as instâncias testadas nos 6 modelos de recompensas implementados, dado por $\frac{res_{mod} - res_{amam}}{res_{amam}} \times 100$, em que res_{mod} corresponde ao valor obtido pela modelagem de recompensa mod e res_{amam} corresponde ao resultado obtido utilizando o algoritmo de [Silva et al. 2019]. A partir desta tabela, conclui-se que a modelagem de recompensa rd_lilo apresentou os menores valores médios de gap($\%$), enquanto a modelagem de recompensa r_gl apresentou os maiores valores de gap($\%$). De forma coerente, as modelagens que usam recompensa global apresentaram os piores resultados quanto ao gap($\%$). Por outro lado, os melhores resultados foram alcançados com a recompensa r_linha.

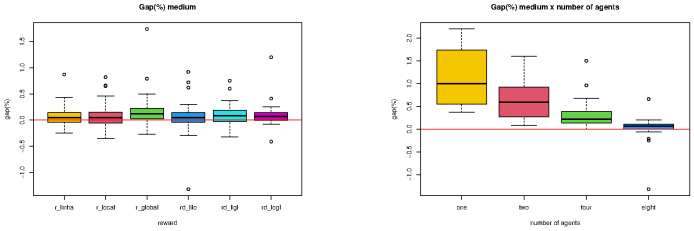
Tabela 1. Gap(%) médio para as instâncias testadas.

		Numero de agentes				
reward	Instância	one	two	four	eight	gap(%)
	C109	0,31	0,21	0,11	0,09	0,18
	C208	0,03	-0,13	-0,04	0,00	-0,03
	R112	0,11	0,34	0,11	0,04	0,15
r_li	R211	-0,12	-0,11	-0,10	0,18	-0,04
	RC108	-0,04	0,03	0,87	-0,25	0,15
	RC208	0,05	0,43	0,09	-0,05	0,13
	gap(%)	0,06	0,13	0,17	0,00	0,09
	C109	0,46	0,13	0,16	0,10	0,21
	C208	0,04	0,05	-0,06	0,00	0,01
	R112	0,14	0,00	0,05	0,06	0,06
r_lo	R211	0,11	0,01	-0,10	0,16	0,04
	RC108	-0,35	0,82	0,65	0,66	0,45
	RC208	-0,23	-0,08	-0,07	-0,06	-0,11
	gap(%)	0,03	0,16	0,10	0,15	0,11
	C109	0,50	0,24	0,12	0,16	0,25
	C208	0,06	-0,12	0,00	0,00	-0,02
	R112	0,16	0,00	0,06	0,08	0,08
r_gl	R211	0,23	0,05	-0,07	0,15	0,09
	RC108	-0,27	0,79	1,74	0,07	0,58
	RC208	0,48	0,12	0,22	0,13	0,23
	gap(%)	0,19	0,18	0,34	0,10	0,20

		Numero de agentes				
reward	Instância	one	two	four	eight	gap(%)
	C109	0,30	0,14	0,15	0,11	0,18
	C208	0,05	-0,10	-0,06	0,00	-0,03
	R112	0,72	0,04	0,07	0,09	0,23
rd_lilo	R211	0,00	0,11	-0,03	0,20	0,07
	RC108	-0,05	0,62	0,92	-1,32	0,04
	RC208	-0,01	-0,10	-0,29	-0,01	-0,10
	gap(%)	0,17	0,12	0,13	-0,16	0,06
	C109	0,31	0,15	0,11	0,06	0,15
	C208	-0,32	-0,03	-0,06	0,00	-0,10
	R112	0,75	-0,05	0,03	0,08	0,20
rd_ligl	R211	0,09	0,22	-0,01	0,14	0,11
	RC108	-0,08	0,60	0,37	-0,21	0,17
	RC208	0,16	0,13	0,23	-0,03	0,12
	gap(%)	0,15	0,17	0,11	0,01	0,11
	C109	0,41	0,07	0,07	0,07	0,16
	C208	-0,41	0,16	-0,08	0,00	-0,08
	R112	0,12	-0,04	0,07	0,03	0,04
rd_logl	R211	0,08	-0,01	-0,04	0,09	0,03
	RC108	-0,02	0,25	1,20	0,07	0,37
	RC208	0,18	0,16	0,01	0,02	0,09
	gap(%)	0,06	0,10	0,21	0,05	0,10

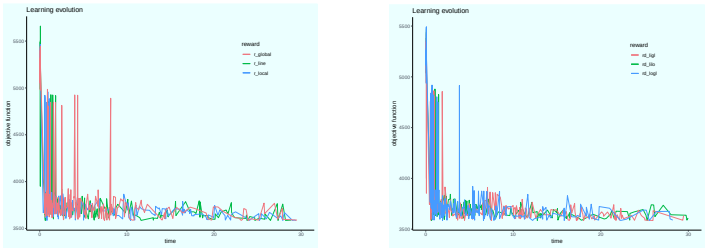
Além disso, as execuções com oito agentes levaram aos menores valores de gap($\%$).

A Figura 5a apresenta um *boxplo*t com os valores de gap($\%$) obtidos com as modelagens de recompensas implementadas. Observa-se que a tendência é que não exista diferenças significativas entre as modelagens. Já a Figura 5b mostra os valores de gap($\%$) encontrados ao se alterar o número de agentes utilizados pelo AMAM. Neste caso, claramente identifica-se a superioridade das soluções geradas com oito agentes frente aos demais cenários. Além disso, este é o único cenário em que as soluções obtidas foram melhores que as postas em [Silva et al. 2019]. Para verificar a existência ou não de diferenças significativas entre os modelos de recompensas implementados, foi realizado um teste estatístico não-paramétrico. O teste utilizado foi o teste de Kruskal-Wallis. Este teste foi realizado para verificar a qualidade dos valores da função objetivo nas diferentes modelagens de recompensa. O teste de Kruskall-Wallis apresentou $p - value > 0,05$. Desta forma, constata-se que não há diferenças significativas entre as modelagens para um nível de significância de 5%. Assim, não rejeita-se a hipótese nula de igualdade das médias. Desta forma, comprova-se que a utilização de modelagens de recompensas é eficiente e pode substituir modelagens fortemente ligadas ao problema e, portanto, a modelagem de recompensas torna a estrutura mais autônoma. Uma comparação do tempo computacional obtido com cada modelagem de recompensas em relação ao tempo computacional obtido com a estrutura de [Silva et al. 2019] mostra que (i) a estrutura que obteve maior tempo computacional médio foi a estrutura com quatro agentes; e (ii) o tempo computacional foi, em média, $16,33\%$ maior com as modelagens de recompensas. O aumento do tempo computacional não é significativo, pois, em média, os $16,33\%$ correspondem a 25 segundos. A Figura 6 apresenta os testes obtidos com a execução da instância C208 com um agente. A Figura 6a mostra a evolução do valor da função objetivo em relação ao tempo computacional, para as modelagens r_linha, r_local e r_global. Pode-se observar que a modelagem de recompensa r_global é menos estável e sua evolução é levemente menor. Observa-se também que a evolução da modelagem r_linha é levemente mais rápida. Na Figura 6b é apresentada a evolução das modelagens de recompensas rd_lilo, rd_ligl e rd_logl. Observar-se que a modelagem rd_logl se mostra menos estável e a modelagem



(a) Recompensa. **(b)** Numero de agentes.

Figura 5. Gap(%) médio por recompensa e por numero de agentes.



(a) Evolução das recompensas locais. **(b)** Evolução das recompensas diferenciais

Figura 6. Evolução das modelagens de recompensas - instância C208.

de recompensa rd_lilo se mostra mais estável, porém, a evolução da modelagem rd_ligl aparentemente se mostra mais rápida.

Portanto, conclui-se, a partir desta análise, que (i) o melhor cenário, em relação ao numero de agentes, é a utilização da estrutura com oito agentes, pois ela se mostra mais estável, obtém os menores gap(%) e é o único cenário que melhora as soluções de [Silva et al. 2019], quando comparado às melhores soluções obtidas, independente do numero de agentes utilizados; e (ii) em relação às modelagens de recompensas, indica-se a utilização das modelagens de diferenças, pois elas se mostram mais eficientes e encontraram os menores gap(%). Ainda em relação às modelagens de recompensas, indica-se a utilização da modelagem rd_lilo, pois a mesma se mostrou mais estável e obteve os menores gap(%).

5. Considerações Finais

Testes computacionais foram realizados e observa-se que as modelagens de recompensas não apresentaram diferenças significativas entre a utilização das modelagens de recompensas e a modelagem de [Silva et al. 2019], em que a recompensa é fortemente ligada ao problema. Além disso, com os experimentos, conclui-se que as modelagens de recompensas não apresentaram diferenças significativas entre si, e indica-se a utilização de modelagens de recompensas baseadas em diferenças, uma vez que obtiveram os menores gap(%) e mostraram-se mais estáveis, em especial a modelagem rd_lilo. Adicionalmente, indica-se a utilização da estrutura com oito agentes, visto que obteve os menores gap(%) e foi a única estrutura a melhorar as soluções das modelagens de recompensas em

relação as melhores soluções obtidas a partir da modelagem de recompensas apresentada em [Silva et al. 2019]. Para a continuidade destes estudos, propõe-se: (i) desenvolver técnicas de recompensas de modelagens baseadas em potencial e diferenças ao mesmo tempo; (ii) realizar testes computacionais com um numero maior de agentes.

Agradecimentos

Os autores agradecem ao Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG), à Universidade Federal de Viçosa - Campus Florestal (UFV) e à Divisão de Suporte ao Desenvolvimento Científico e Tecnológico da Universidade Federal de Viçosa (`https://dct.ufv.br`) pelo apoio ao desenvolvimento do presente estudo.

Referências

Agogino, A. K. and Tumer, K. (2008). Analyzing and visualizing multiagent rewards in dynamic and stochastic domains. Autonomous Agents and Multi-Agent Systems, 17(2):320–338.

Devlin, S., Yliniemi, L., Kudenko, D., and Tumer, K. (2014). Potential-based difference rewards for multiagent reinforcement learning. In Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems, pages 165–172.

Feo, T. A. and Resende, M. G. C. (1995). Greedy randomized adaptive search procedures. Journal of Global Optimization, 6:109–133.

Grzes, M. (2017). Reward shaping in episodic reinforcement learning. Autonomous Agents and Multi-Agent Systems.

Lourenço, H. R., Martin, O. C., and Stützle, T. (2003). Iterated local search. In Glover, F. and Kochenberger, G. A., editors, Handbook of Metaheuristics, volume 57 of International Series in Operations Research & Management Science, chapter 11, pages 321–353. Kluwer Academic Publishers.

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. Computers & Operations Research, 24(11):1097–1100.

Silva, M. A. L., de Souza, S. R., Souza, M. J. F., and Bazzan, A. L. C. (2019). A reinforcement learning-based multi-agent framework applied for solving routing and scheduling problems. Expert Systems with Applications, 131:148 – 171.

Silver, D., Singh, S., Precup, D., and Sutton, R. S. (2021). Reward is enough. Artificial Intelligence, 299:103535.

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Research, 2(35):254–264.

Sutton, R. S. and Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

Toth, P. and Vigo, D. (2014). Vehicle routing: problems, methods, and applications. SIAM.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. Machine Learning, 8(3):279–292.