ALGORITMO HIBRIDO DE SELEÇÃO CLONAL BASEADO EM REGRA DE PRIORIDADE PARA O PROBLEMA DE SEQUENCIAMENTO EM PROJETOS COM RESTRIÇÃO EM RECURSOS E MÚLTIPLOS MODOS DE EXECUÇÃO

Gustavo Alves Fernandes

Sérgio Ricardo de Souza

Moacir Felizardo de França Filho

gustavo.alfer@gmail.com

sergio@dppg.cefetmg.br

franca@des.cefetmg.br

Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG)

Av. Amazonas, 7675 - Nova Gameleira- CEP 30510-000 - Belo Horizonte, MG, Brasil

Resumo. *Esse artigo considera o problema não preemptivo de sequenciamento em projetos com restrição em recursos e múltiplos modos de execução para a minimização do makespan. Para solucionar esse problema, é apresentada a metaheurística populacional de Seleção Clonal (CLONALG). Para a construção de cada solução são definidos os modos de execução de cada tarefa. Após fixados os modos, um método baseado em regra de prioridade é executado para criar a lista de atividades e após, sequenciá-las. Finalmente, uma heurítica de busca local é realizada sobre as soluções viáveis da população corrente. Os resultado computacionais mostraram que o algoritmo proposto obteve boa performance encontrado soluções de boa qualidade em tempo relevante para um grupo de instâncias entre classes de problemas da biblioteca pública PSPLIB.*

Keywords: *Sequenciamento em Projetos, Algotimo de Selecão Clonal, Meta-Heurística*

1 INTRODUÇÃO

O problema clássico de Sequenciamento em Projetos com Restrição em Recursos *(Resource Constrained Project Scheduling Problem – RCPSP)* consiste em sequenciar todas as atividades de um projeto, respeitando a ordem de precedência dessas atividades, que, uma vez iniciadas, não podem ser interrompidas. As atividades possuem tempo de processamento e consumo de recursos, que podem ser renováveis ou não renováveis. O objetivo é minimizar o *makespan* (ou tempo de duração do projeto), encontrando tempos de início para cada atividade do projeto. A versão generalizada do RCPSP é o Problema de Sequenciamento em Projetos com Restrição em Recursos e Múltiplos Modos de Execução *(Multi-mode Resource Constrained Project Scheduling Problem – MRCPSP)*. Neste último, o objetivo permanece a minimização do *makespan*, porém, as atividades podem ser processadas com várias combinações de modos de execução, o que consiste em diferentes combinações de tempo de processamento e consumo de recursos.

De acordo com Blazewicz (1986), encontrar um sequenciamento para o RCPSP é um problema NP-Difícil. Por ser uma generalização do RCPSP, o MRCPSP também é um problema NP-Difícil. Já Drexl and Gruenewald (1993) apontam que obter soluções viáveis para o MRCPSP é uma tarefa complicante. Foi demonstrado por Kolisch and Drexl (1997) que obter uma solução viável para o MRCPSP, com pelo menos 2 tipos de recursos não renováveis, é um problema NP-Completo. Ou seja, não é possível desenvolver um procedimento determinístico que construa e garanta um sequenciamento viável, com relação aos recursos não renováveis, em tempo polinomial. Conforme apontam Sprecher and Drexl (1998), projetos com mais de 20 atividades com no mínimo 3 modos de execução não podem ser solucionados na otimalidade por métodos exatos. Assim, várias abordagens baseadas em heurísticas vêem sendo propostas para solucionar o MRCPSP.

Em Jozefowska et al. (2001), os autores propuseram uma implementação da metaheurística *Simulated Annealing* (SA) fazendo a busca alternadamente em vizinhanças baseadas em atividades e modos de execução. Já em Bouleimen and Lecocq (2003), os autores também apresentaram um SA, tendo a busca local em dois estágios. No primeiro estágio, a vizinhança é baseada apenas em modos de execução e, para casos em que é possível melhorar o *makespan* encontrado até então, o segundo estágio é executado com vizinhanças baseadas em atividades, solucionando-se o subproblema RCPSP. Os autores em Zhang et al. (2006) propuseram a estratégia evolucionária *Particle Swarm Optimization* (PSO), com a adição de um procedimento para reparar soluções inviáveis. Em Jarboui et al. (2008) também foi utilizada a metaheurística PSO para atribuir modos de execução às atividades, combinando com uma busca local, que determina o sequenciamento das atividades durante a evolução do PSO. A mesma abordagem via PSO foi utilizada por Truong and Kachitvichyanukul (2007), porém, os autores hibridizaram o PSO com um Algoritmo Genético (AG), sendo que o PSO gera as prioridades das atividades, enquanto o AG define modos de execução para as atividades. Os autores em Cui and Yu (2014) utilizaram uma abordagem puramente via PSO. Em Damak et al. (2009) é apresentada uma abordagem via *Differential Evolution* (DE), em que os autores restringem o tempo de busca pelo DE, comparando-a com resultados obtidos por Jarboui et al. (2008) e Bouleimen and Lecocq (2003). Já os autores Alcaraz et al. (2003) desenvolveram um AG, com modificações no cruzamento e mutação postas a partir de observações dos mesmos em seus trabalhos com o RCPSP, assim como modificações observadas na função objetivo do AG proposto por Hartmann (2001). Em Van Peteghem and Vanhoucke (2010) também é utilizado

um AG, combinado com a heurística *Forward/Backward Improvement* (FBI). Em Lova et al. (2009) também é apresentado um AG, mas é aplicada uma variação da FBI somente em indivíduos viáveis. Em Bilolikar et al. (2012) é sugerida uma hibridização do AG com uma SA. Uma abordagem via Sistema Imunológico Artificial (AIS) foi utilizada por Van Peteghem and Vanhoucke (2009), hibridizando-a com um procedimento para controlar as soluções inviáveis na população. Várias outras metaheurísticas vêem sendo propostas para solucionar o MRCPSP. Para uma revisão mais detalhada, o leitor é referenciado para Kolisch and Hartmann (1999), Kolisch and Hartmann (2006), Brucker et al. (1999), Hartmann and Kolisch (2000), Van Peteghem and Vanhoucke (2014) e Mika et al. (2015).

O presente artigo apresenta uma adaptação da metaheurística Seleção Clonal (CLONALG) para solucionar o MRCPSP. A aplicação de um algoritmo baseado em um AIS, de acordo com a literatura, mostra-se válida para a resolução de problemas de sequenciamento, tais como *job-shop* em Coello et al. (2003) e Hart et al. (1998), *flow-shop* em Engin and Döyen (2004) e Sun and Yang (2008), para o RCPSP em Agarwal et al. (2007), MRCPSP em Van Peteghem and Vanhoucke (2009) e MRCPSP com múltiplos projetos em Ju and Chen (2012).

O artigo está organizado da seguinte forma. O problema objeto de interesse é descrito na Seção 2. A Seção 3 apresenta a Teoria de Seleção Clonal. Na Seção 4 é discutida as heurísticas utilizadas no presente trabalho assim como a adaptação da metaheurística CLONALG para o MRCPSP. A Seção 5 mostra os resultados dos experimentos computacionais realizados. A Seção 6apresenta as conclusões derivadas dos desenvolvimentos e experimentos realizados.

2 DESCRIÇÃO DO PROBLEMA

Considere um projeto com n atividades. Cada atividade $j, j = 1, ..., n$ deve ser processada sem interrupção, respeitando-se a relação de precedência entre as mesmas, i.e. cada atividade j pode ser iniciada somente após todas as suas predecessoras $p \in Pj$ estiverem finalizadas, sendo Pj o conjunto de atividades que precedem a atividade j. Convencionalmente, as atividades j_1 e j_n são fictícias, de modo que j_1 deve ser a primeira atividade a ser iniciada sem algum predecessor e j_n deve ser a última com nenhum sucessor. Cada atividade do projeto deve ser executada em um modo de execução que estabelece o tempo de processamento e demanda de recursos para essa atividade. As atividades fictícias possuem um único modo de execução, associado com zero tempo de duração e consumo de recursos. O número de modos em que uma atividade j pode ser executada é dado por $|M_j|$ e o tempo de processamento é denotado por d_{jm}. A disponibilidade de recursos renováveis e não renováveis é dada, respectivamente, por R_k^ρ e R_k^υ sendo $k = 1, ..., K$ o tipo de recurso utilizado. As quantidades r_{jmk}^ρ e r_{jmk}^υ definem, respectivamente, o consumo de recursos renováveis e não renováveis do tipo k no modo m pela atividade j. Considera-se que todos os parâmetros são valores inteiros não negativos. Então, o objetivo do MRCPSP é encontrar uma combinação dos modos de execução para todas as atividades, resultando em um sequenciamento, com tempo de início e consumo de recursos, tal que o *makespan* (duração total do projeto) seja minimizado.

Para representar um projeto, é comum na literatura o uso da estrutura em rede *Activity on Node* (AoN) (cf. Demeulemeester (2002)). Nessa estrutura, as atividades são os nós e os arcos denotam as precedências entre atividades. A Fig. 1 apresenta um exemplo nessa estrutura em rede para um projeto com 8 tarefas, sendo 2 fictícias. A Tabela 1 apresenta os modos de execução das atividades, com seus respectivos tempos de processamento e consumo de recursos.

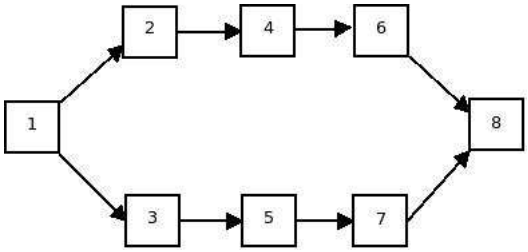

Figura 1: Exemplo de projeto em estrutura de rede

Tabela 1: Modos de execução para as tarefas no exemplo de projeto apresentado pela Fig. 1. Adaptado de Hartmann (2001)

j	m	d_{jm}	r^{ρ}_{jmk}	r^{ν}_{jmk}	R^{ρ}_{k}	R^{ν}_{k}
1	1	0	0	0	4	15
2	1	3	2	5		
	2	4	1	1		
3	1	2	3	6		
	2	4	3	2		
4	1	2	4	2		
	2	3	2	2		
5	1	2	3	6		
	2	2	4	4		
6	1	3	3	1		
	2	3	1	7		
7	1	4	2	1		
	2	6	1	1		
8	1	0	0	0		

3 TEORIA DE SELEÇÃO CLONAL

O Algoritmo de Seleção Clonal, proposto por De Castro and Von Zuben (2002), foi inicialmente apresentado para reconhecimento de padrões e, logo após, adaptado para resolução de funções multimodais, assim como problemas de otimização. O princípio da seleção clonal é usado para descrever as características de respostas em um sistema imunológico para antígenos. Esse sistema estabelece que somente células que respondem ou reconhecem os antígenos, são selecionadas para proliferação, sendo que estas células fornecem as melhores respostas aos antígenos apresentados. A resposta que as células fornecem são conhecidas como anticorpos. Então, as células selecionadas passam por um processo de maturação, que aprimora sua afinidade em relação aos antígenos apresentados.

Para a aplicação de um algoritmo baseado em Seleção Clonal, segundo De Castro and Von Zuben (2002), alguns passos devem ser realizados:

Passo 1 - Inicialização: Criação de uma população de anticorpos $AB(t)$;

Passo 2 - Apresentação do antígeno: Avaliação da afinidade de cada anticorpo;

Passo 3 - Seleção Clonal: Criação de uma população de clones $C(t)$ dos melhores anticorpos por meio da afinidade;

Passo 4 - Maturação: Hipermutação dos clones em $C(t)$ dada pela afinidade de cada clone;

Passo 5 - Re-seleção: Os melhores anticorpos em $C(t)$ e $AB(t)$ são selecionados para uma nova população $AB(t+1)$;

Passo 6 - Diversidade: Introdução de d novos anticorpos em $AB(t+1)$ substituindo anticorpos com menor afinidade.

De acordo com De Castro and Von Zuben (2002), algumas observações são necessárias sobre o processo de seleção clonal, quando aplicadas a um problema de otimização.

(i) Para o passo 1, por simplificação, somente os anticorpos são apresentados para avaliação, sendo que cada anticorpo representa uma solução ou sequenciamento. No passo 2 não há antígeno para ser reconhecido, mas uma função objetivo a ser otimizada. Então, a afinidade do anticorpo corresponde à sua avaliação *(fitness)* associada à função objetivo.

(ii) Se o problema consiste em localizar múltiplos ótimos locais, então é preferível que toda população seja selecionada para clonagem com o mesmo número de clones cada anticorpo. A ideia é que cada anticorpo poderá visto localmente, ou seja, se toda a população, ou um subconjunto dela, for selecionada para clonagem, cada anticorpo poderá representar uma solução em uma região do espaço de busca.

(iii) A afinidade será utilizada para determinar a taxa de mutação de cada anticorpo, que é inversamente proporcional a sua afinidade, ou seja, quanto melhor o anticorpo responde à função, menor é a probabilidade de hipermutação desse anticorpo.

4 CLONALG APLICADO AO MRCPSP

Esta seção apresenta o detalhamento da adaptação do CLONALG para solucionar o Problema de Sequenciamento em Projetos com Restrição em Recursos e Múltiplos Modos de Execução (MRCPSP).

4.1 Representação das soluções

Várias estruturas de representação de soluções para o MRCPSP podem ser encontradas na literatura, como as apresentadas em Demeulemeester (2002) e em Kolisch and Hartmann (1999). No presente artigo, uma solução s é representada por um par de vetores $s = (J, M)$, em que J é uma permutação viável de todas as atividades e M uma permutação válida de modos de execução. Assim, os elementos $j_i \in J$ e $m_i \in M$ informam que, na posição i, a atividade j é executada pelo modo m. Essa representação é conhecida pela literatura como Lista de Prioridade, devido à ordem das atividades ser definida por uma Regra de Prioridade. A Figura 2 mostra esse esquema de representação. Já a Figura 3 apresenta um exemplo de solução para o problema de projeto apresentado pela Fig. 1.

Atividade	j_1	j_2	-	-	-	-	-	j_n
Modo	m_1	m_2	-	-	-	-	-	m_n

Figura 2: Esquema de representação de solução.

Atividade	1	3	5	2	7	4	6	8
Modo	1	2	2	1	1	1	1	1

Figura 3: Exemplo de solução para o problema de projeto apresentado pela Fig. 1.

4.2 Regras de prioridade

Regras de prioridade definem valores de prioridade $\upsilon(j)$ para uma atividade $j \in D_i$, permitindo a seleção da melhor atividade j^*, baseando-se no seu valor de prioridade em conjunto com um objetivo O, e, assim, construir um sequenciamento. Logo, de acordo com Kolisch (1996b), regras de prioridade são heurísticas que geram vários sequenciamentos. O conjunto D_i, conhecido como conjunto de decisão, é formado por atividades, no estágio i, que possuem todos os seus predecessores já sequenciados.

Nesse trabalho, a regra de prioridade será utilizada para definir a ordem em que as atividades serão selecionadas para receberem atribuição de tempos de início, através da ordenação dada pela representação da solução. Através da Figura 3, foi apresentado um exemplo de lista de prioridade, sendo a ordem dada como $< 1; 3; 5; 2; 7; 4; 6; 8 >$. Essa lista foi construída da seguinte maneira: no estágio $i = 1$, $D_1 = \{1\}$, e então a atividade 1 é retirada de D_1, sendo a primeira a entrar na lista. Após, quando $i = 2$, $D_2 = \{2, 3\}$. Através de um valor $\upsilon(j) : j \in D_2$, a atividade 3 foi a próxima a ser selecionada para a lista de prioridade, implicando, para $i = 3$, que $D_3 = \{2, 5\}$. A lista de prioridade será finalizada, quando a atividade 8 entrar para a lista. É importante esclarecer que a primeira e última atividades do projeto devem ser, respectivamente, a primeira e última atividades da lista de prioridade.

Métodos que utilizam uma regra de prioridade de forma determinística são conhecidos como abordagem *Single-Pass* e geram apenas um sequenciamento. Por outro lado, procedimentos que visam gerar mais de um sequenciamento devem realizá-los por alteração na regra de prioridade. Estes procedimentos são conhecidos como abordagem *Multi-Pass*. Os dois tipos de procedimentos *Multi-Pass*, apresentados por Kolisch (1996b), são caracterizados como *Multi-Priority Rule*, que utiliza de um Esquema de Geração de Sequenciamento (SGS) com várias regras de prioridade, e *Sampling*, que utiliza um SGS e uma regra de prioridade, porém, os diferentes sequenciamentos são obtidos por um dispositivo aleatório aplicado sobre a regra de prioridade.

Conforme exposto em Kolisch (1996b), a utilização de um dispositivo aleatório pode ser interpretado como um mapeamento $\psi : j \in D_i \rightarrow [0, 1]$, dado que, a cada estágio i, é atribuída a cada atividade j, $j \in D_i$, uma probabilidade $\psi(j)$ a ser selecionada, com $\sum_{j \in D_i} \psi(j) = 1$. Três métodos diferentes para *Multi-Pass Sampling* são apresentados em Kolisch (1996b). Neste trabalho, será utilizado o método *Regret Based Biased Random Sampling*.

Considere uma regra de prioridade a ser definida por $\upsilon : j \in D_i \rightarrow \Re_{\geq 0}$, que atribui a cada atividade $j \in D_i$ um valor de prioridade $\upsilon(j)$, e um objetivo O, indicando se a atividade do conjunto de decisão deve ser selecionada pelo mínimo $O = \min$ ou pelo máximo $O = \max$ valor de prioridade. Então, ϱ_j, conhecido como *Regret*, compara o valor de prioridade da atividade j de acordo com:

$$\varrho_j \leftarrow \begin{cases} \max\limits_{\bar{j}\in D_i} v(\bar{j}) - v(j), & \text{se } O = \min \\ v(j) - \min\limits_{\bar{j}\in D_i} v(\bar{j}), & \text{se } O = \max \end{cases} \tag{1}$$

Então, a probabilidade para escolher uma atividade em D_i é definida por:

$$\psi(j) \leftarrow \frac{(\varrho_j + 1)^\alpha}{\sum_{\bar{j}\in D_i}(\varrho_{\bar{j}} + 1)^\alpha} \tag{2}$$

De acordo com Kolisch (1996b), adicionando 1 a ϱ_j, é assegurado um valor diferente de zero para cada atividade e, então, qualquer sequenciamento pode ser gerado pelo SGS. Conforme discutido em Kolisch (1996b), o parâmetro α controla a probabilidade de escolha de uma atividade. Para altos valores de α, a Eq. 2 vai se tornando determinística, enquanto para o valor de α igual a zero, será dada a característica de probabilidade uniforme para todas as atividades. No presente trabalho, utiliza-se $\alpha = 1$.

A regra de prioridade utilizada neste trabalho é conhecida como *Latest Finish Time - LFT*, cujo valor é obtido pelo Método do Caminho Crítico (*Critical Path Method - CPM*), por meio de um procedimento recursivo *Forward-Backward*, sobre a duração das atividades, após a atribuição de modos de execução. Em Demeulemeester (2002) detalha-se o CPM. A literatura sugere a regra de prioridade LFT como aquela que apresenta melhores soluções para o SGS Serial. Alguns destes estudos podem ser encontrados em Kolisch (1996b), Kolisch (1996a), Lova et al. (2006) e Browning and Yassine (2010).

A utilização de uma regra de prioridade aliada a uma meta-heurística é justificado pelas discussões em Hartmann and Kolisch (2000), dado que os autores avaliaram várias meta-heurísticas e concluíram que regras de prioridade de fato auxiliam tais meta-heurísticas para resolução do RCPSP e derivados.

Por fim, uma vez criada uma lista de prioridade, o SGS deverá construir uma solução, ou seja, construir um sequenciamento.

4.3 Heurísticas Construtivas

De acordo com Kolisch (1996b), uma heurística para sequenciamento baseada em regras de prioridade é construída por dois componentes, um Esquema de Geração de Sequenciamento (*Scheduling Generation Scheme* - SGS) e uma Regra de Prioridade.

Um SGS atribui tempos de início para cada atividade do projeto sempre respeitando as restrições de precedência e consumo de recursos renováveis. Quando todas as atividades receberem tempos de início, um sequenciamento é construído. A literatura distingue dois SGSs, Serial e Paralelo. Em Kolisch (1996b) são discutidos os dois SGSs em detalhe. Nesse trabalho, o SGS em Serial será utilizado como heurística construtiva.

Seja πR_{kt} o restante disponível do recurso renovável k por período t, definido como $\pi R_{kt} \leftarrow R^\rho_{kt} - \sum_{j\in A_t} r^\rho_{jk}$, sendo A_t o conjunto das atividades já sequenciadas, consumindo recursos no período t. Então, o SGS em Serial é apresentado na forma do Algoritmo 1.

Algorithm 1: Esquema de Geração de Sequenciamento Serial - (SSS)

Data: Lista de Prioridade: $s = (J, M)$

1 Inicializar $A = \emptyset, i \leftarrow 1$;
2 **while** $|A| \leq |J|$ **do**
3 Calcular $\pi R_{kt}, t = 1, ..., T, \forall k \in K$;
4 $j^* \leftarrow J(i)$;
5 $EFT_{j^*} \leftarrow max\{FT_p : p \in P_{j^*}\} + d_{j^*}$;
6 $FT_{j^*} \leftarrow min\{t : EFT_{j^*} \leq t \leq LFT_{j^*}, r^{\rho}_{j^*k} \leq \pi R_{k\tau}, \tau = t - d_{j^*} + 1, ..., t, \forall k \in K\}$;
7 $A \leftarrow A \cup \{j^*\}$;
8 $i \leftarrow i + 1$;
9 **end**

Pelo Algoritmo 1, FT_j representa o tempo de término atribuído à atividade j, EFT_j e LFT_j representam, respectivamente, os períodos em que uma atividade j pode ser finalizada o mais cedo possível e o mais tarde possível e T representa o horizonte do projeto. Então, sendo conhecida a lista de prioridade $s = (J, M)$, a cada estágio i, uma atividade j^* na posição i da lista de atividades J, recebe o menor tempo de término viável, FT_{j^*}, respeitando o consumo de recursos durante todo o período em que j^* é executada. Após o final da repetição dada pelas linhas 2-8, todas as atividades receberam um tempo de término, ou seja, todas as atividades foram sequenciadas. A Figura 4 mostra o sequenciamento obtido pelo SSS a partir da lista de prioridade apresentada pela Fig. 3.

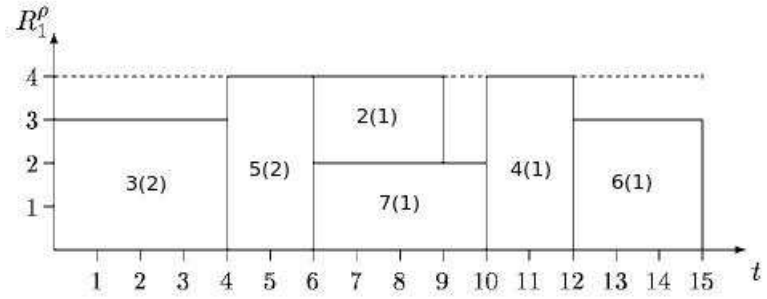

Figura 4: Exemplo de sequenciamento obtido pelo método SSS para a solução apresentada pela Fig. 3. Em Hartmann (2001).

Observa-se, pela Figura 4, que o sequenciamento é viável com relação à disponibilidade de recurso não renovável R_k^v, pois a quantidade requerida pelas atividade obteve soma de 15 unidades. O *makespan* é de 15 unidades de tempo e a disponibilidade de recurso renovável R_k^{ρ} é respeitada ao longo de todo o horizonte de realização do projeto.

4.4 Heurística de Refinamento

O procedimento heurístico utilizado neste trabalho como busca local foi descrito inicialmente como *Bouding Rule* no algoritmo exato em Sprecher et al. (1997) e também utilizado em Hartmann (2001). Essa heurística é baseada no conceito de *Multi-Mode Left Shift - MMLF*. Um MMLF em uma atividade j é uma operação em um sequenciamento que reduz o tempo de término de j sem alterar os modos ou tempos de término das outras atividades no sequenciamento. Conforme Hartmann (2001), duas características tornam esse movimento promissor:

(i) MMLF considera tempos de início das atividades e modos simultaneamente;

(ii) O MMLF não deteriora a qualidade de uma solução, ou seja, a solução permanece viável e o *makespan* da solução não aumenta.

Há duas maneiras de construir uma heurística baseada no movimento MMLF, utilizando uma abordagem *Single-Pass* ou *Multi-Pass*. Ambas as abordagens são descritas em Hartmann (2001). Neste artigo será utilizada a abordagem *Single-Pass*. Seja j uma atividade do projeto já sequenciada e m o modo atual dessa atividade. A heurística de busca local é apresentada pelo Algoritmo 2.

Algorithm 2: Busca Local MMLF

for $i \leftarrow 2$ **to** $|J| - 1$ **do**
  $\pi R_{kt} \leftarrow \pi R_{kt} + r^{\rho}_{j_i mk}, t = FT_{j_i m} - d_{j_i m}, ..., FT_{j_i m}, \forall k \in K$;
  for $\bar{m} \leftarrow 1$ **to** $|M_{j_i}|$ **do**
    if $\bar{m} \neq m$ *&& modo viável w.r.t. recursos não renováveis* **then**
      $FT^{'}_{j_i \bar{m}} \leftarrow min\{t : EFT_{j_i \bar{m}} \leq t \leq LFT_{j^* \bar{m}}, r^{\rho}_{j^* \bar{m} k} \leq \pi R_{k\tau}, \tau = t - d_{j^* \bar{m}} + 1, ..., t, \forall k \in K\}$;
      if $FT^{'}_{j_i \bar{m}} \leq FT_{j_i m}$ **then**
        $FT_{j_i m} \leftarrow FT^{'}_{j_i \bar{m}}$;
        $m \leftarrow \bar{m}$;
        break ;
      end
    end
  end
  $\pi R_{kt} \leftarrow \pi R_{kt} - r^{\rho}_{j_i mk}, t = FT_{j_n m} - d_{j_i m}, ..., FT_{j_i m}, \forall k \in K$;
end

O Algoritmo 2 busca melhorar o *makespan* de uma solução da seguinte maneira. Para cada atividade $j_i, i \in \{2, ..., |J| - 1\}$ do sequenciamento, libera-se o consumo de recursos de j_i pelo modo m corrente, e verifica-se se um MMLF pode ser realizado. Assim, os modos M_{j_i} são testados. Para cada atividade, o primeiro modo viável $\bar{m}$ encontrado, com relação ao recursos não renováveis, é atribuído à atividade e ao sequenciamento. O resultado é um sequenciamento viável com *makespan* menor ou igual ao *makespan* anterior. De acordo com a literatura, os modos de execução de todas as atividades devem estar ordenados de forma não decrescente, com relação ao tempo de duração, para que o modo com menor duração seja verificado primeiramente. Claramente, o método é denominado *Single-Pass*, pois cada atividade é considerada apenas uma vez. Experimentos preliminares realizados por Hartmann (2001) mostraram que o ganho na qualidade das soluções obtidas pelo método *Multi-Pass* sobre o método *Single-Pass* não vale o esforço computacional do primeiro. Um exemplo da execução da heurística *Single-Pass* apresentada pelo Algoritmo 2 é dado pela Fig. 5.

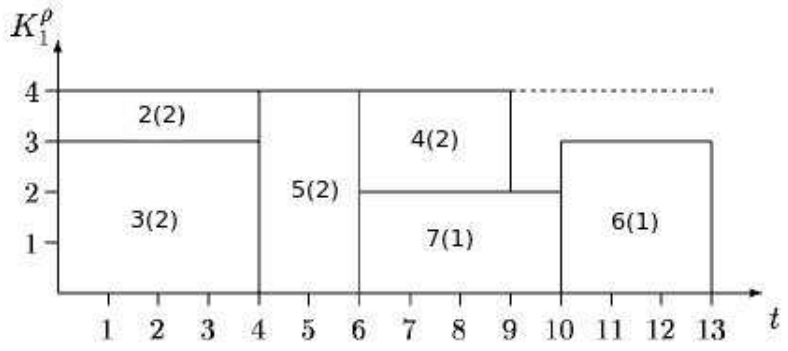

Figura 5: Exemplo de aplicação da busca local *single-pass* MMLF no sequenciamento apresentada pela Fig. 4. Em Hartmann (2001).

Utilizando o sequenciamento apresentado pela Fig. 4 como solução corrente, ao iniciar a heurística *Single-Pass*, as atividades 3 e 5 não realizam o movimento MMLF, mas um MMLF pode ser realizado na atividade 2. A atividade 7 permanece com seu modo corrente e, então, é possível aplicar o MMLF na atividade 4. Por último, a atividade 6 tem seu tempo de término reduzido devido a redução nas atividades anteriores. Ao fim da heurística *single-pass*, o *makespan* anterior foi reduzido para 13 unidades de tempo.

4.5 Função de avaliação

A função objetivo é frequentemente utilizada como função de avaliação, pois provê uma medida de qualidade para uma solução. Quando o espaço de soluções inviáveis é considerado, a função de avaliação utiliza a função objetivo com o acréscimo de uma penalidade. Assim sendo, a função de avaliação foi discutida na utilização de heurísticas aplicadas ao MRCPSP com minimização do *makespan*.

Em Hartmann (2001), definiu-se uma função de avaliação considerando o *makespan* para soluções viáveis. Para as inviáveis, foi utilizado um *upper bound*, conhecido como horizonte do projeto, com adição da quantidade de recursos não renováveis consumida em excesso. Essa mesma função de avaliação foi usada em Jozefowska et al. (2001). Entretanto, os autores Alcaraz et al. (2003) apontaram que, usando essa avaliação, soluções inviáveis com *makespan* distintos e consumindo a mesma quantia em excesso de recursos não renováveis irão receber a mesma penalização. Em consequência, propuseram uma nova função de avaliação para soluções inviáveis, considerando o excesso de consumo de recursos não renováveis, um *lower bound*, conhecido como caminho crítico, o *makespan* da solução avaliada e o máximo *makespan* das soluções na população. Os autores em Van Peteghem and Vanhoucke (2010) utilizaram essas duas funções de avaliação. Contudo, Lova et al. (2009) observaram que essa última avaliação adiciona unidade de tempo do *makespan* a unidades de recursos a partir do excesso consumido de consumo de recursos não renováveis. Claramente, a magnitude de ambos aspectos podem distorcer o significado da solução. Assim, Lova et al. (2009) apresentaram uma função de avaliação, que mantém a mesma magnitude entre as parcelas envolvidas na função. A mesma função de avaliação é adotada em Bilolikar et al. (2012).

Portanto, no presente artigo, essa última função de avaliação é utilizada, no entanto, com a troca do máximo *makespan* pelo horizonte do projeto. Assim sendo, o excesso consumido de

recursos não renováveis por uma solução s, dado por $RC(s)$, e a função de avaliação adotada, dada por $f(s)$, são definidos como:

$$RC(s) \leftarrow \sum_{\forall k \in R_k^v} \max \left\{ 0, \sum_{j=1}^{n} r_{jmk} - R_k^v \right\} \tag{3}$$

$$f(s) \leftarrow \begin{cases} 1 - \dfrac{UB - mak(s)}{UB}, & \text{se } RC(s) = 0 \\ 1 + \dfrac{mak(s) - LB}{mak(s)} + \displaystyle\sum_{\forall k \in R_k^v} \max \left\{ 0, \dfrac{\sum_{j=1}^{n} r_{jmk} - R_k^v}{R_k^v} \right\}, & \text{caso contrário} \end{cases} \tag{4}$$

Na Equação 4, UB é o horizonte do projeto e LB é o caminho crítico minimal obtido em decorrência da atribuição de modos com menor tempo de duração para cada atividade.

4.6 Clonagem

Após ordenar as soluções em ordem não decrescente pelo seu valor de *fitness*, as N melhores soluções são escolhidas para proliferação ou clonagem.

Como observado por De Castro and Von Zuben (2002), para problemas de otimização cujo objetivo é explorar ótimos locais, é preferível manter o mesmo número de clones para toda solução selecionada para clonagem. Nesse trabalho, toda solução $s \in AB$, selecionada para clonagem, possui c clones, tal que $c = \lfloor(\beta \times |AB|)\rfloor$, sendo $|AB|$ o número de soluções na geração e β um fator de multiplicação. Assim, para toda população de soluções $AB(t)$ na geração t, existe uma população associada de clones $C(t)$ cuja cardinalidade é:

$$|C(t)| \leftarrow \sum_{s=1}^{N} \lfloor \beta \times |AB| \rfloor \tag{5}$$

sendo N o número de soluções selecionadas para clonagem.

4.7 Hipermutação

A fase de hipermutação tem como objetivo realizar alterações aleatórias em genes com diferentes graus de modificações. Como exposto em De Castro and Von Zuben (2002), células com menor afinidade receberão maiores modificações, e, como uma regra, essas células desaparecerão se não aprimorarem sua afinidade e quantidade de clones. Para o problema MRCPSP, o grau de modificações em uma solução é dado pelo *makespan*, ou seja, quanto menor o *makespan*, menor será a modificação. Com esta regra, as melhores soluções proporcionarão uma busca em uma vizinhança pequena, ao contrário das últimas soluções, cuja busca definida pela quantidade de modificações, explorará uma vizinhança de cardinalidade mais elevada. Os autores em Van Peteghem and Vanhoucke (2009) implementaram uma hipermutação baseada na

exploração de vizinhanças de cardinalidade distintas de acordo com a taxa de hipermutação de uma solução. Essa mesma hipermutação será utilizada nesse trabalho e é definida como:

$$\eta(s) \leftarrow 100 \times \exp^{-0.05 \times afinidade(s)} \tag{6}$$

sendo $\eta(s)$ a taxa de hipermutação de uma solução s e $afinidade(s)$ é definida pelo valor através de:

$$afinidade(s) \leftarrow makespan(s) - best\_makespan \tag{7}$$

tal que $best\_makespan$ é o melhor ***makespan*** presente na população. Logo, o grau de mutação $mut\_degree(s)$ em uma solução s é identificado como:

$$mut\_degree(s) \leftarrow 1 + \frac{(100 - \eta(s)) \times |J|}{100} \tag{8}$$

sendo $|J|$ a quantidade de atividades no projeto. É importante esclarecer que $|J|$ não contempla as atividades fictícias j_1 e j_n.

De acordo com Eq. 8, quanto menor (melhor) o *makespan* de uma solução, menor será o grau de modificações.

Dado que uma solução contém uma lista de atividades e uma lista de modos, o grau de modificações é aplicado sobre ambas listas na população de clones C, obtendo clones mutados C^*. Entretanto, o processo de maturação é diferente para cada uma das listas.

Uma mutação sobre a lista de atividades é dada pela troca na posição de uma atividade escolhida aleatoriamente, contudo sempre respeitando a restrição de precedência. Para uma atividade j, o predecessor e sucessor mais próximos, $\bar{j_1} \in P_j$ e $\bar{j_2} \in S_j$, são encontrados determinando-se o quão longe j pode mover-se para frente ou para trás. A Figura 6 apresenta um exemplo dessa mutação em uma atividade.

s

Atividade	1	3	5	2	7	4	6	8
Modo	1	2	2	1	1	1	1	1

s'

Atividade	1	2	3	5	7	4	6	8
Modo	1	1	2	2	1	1	1	1

Figura 6: Exemplo de troca na posição de uma atividade sobre uma solução s levando a outra solução s', utilizando como exemplo a solução apresentada pela Fig. 3.

O processo de troca na posição de uma atividade não será aplicado em soluções inviáveis. Em seguida, a mutação acontece na lista de modos. Uma atividade j é aleatoriamente selecionada e, então, da mesma forma, seu modo é trocado. A Figura 7 apresenta um exemplo da mutação de modo em uma atividade.

s

Atividade	1	2	3	5	7	4	6	8
Modo	1	1	2	2	1	1	1	1

s'

Atividade	1	2	3	5	7	4	6	8
Modo	1	1	2	2	1	2	1	1

Figura 7: Exemplo de troca de um modo em uma atividade s levando a outra solução s', utilizando como exemplo a solução apresentada pela Fig. 3.

O processo de hipermutação tende a inserir sequências distintas e modos distintos que não foram inseridos na população corrente como exploração da vizinhança de uma solução.

4.8 Seleção

Após o processo de hipermutação, uma nova parcela de soluções será introduzida na população. Para isso, os clones C^* são ordenados em valores não decrescentes de *fitness* e os C_{best} clones serão mantidos para a próxima geração, sendo o restante eliminados e substituídos por novas soluções AB_d, com o objetivo de completar a nova população $AB(t+1)$ e manter a característica de diversificação para exploração do espaço de busca.

4.9 Algoritmo CLONALG-MRCPSP

O algoritmo CLONALG-MRCPSP utilizado no presente artigo está mostrado no Algoritmo 3.

Algorithm 3: algoritmo CLONALG-MRCPSP

Data: $|AB|, N, \beta, d, C_{best}, GER$

Result: Vetores: AB

1 $AB(t=0) = \{s^1, ..., s^{|AB|}\} \leftarrow gerar\_anticorpos(|AB|)$ // de acordo com Alg. 1 ;

2 $f_{AB}(t=0) = \{f(s^1), ..., f(s^{AB})\} \leftarrow avaliar\_anticorpos(AB(t))$ // de acordo com Eq. 4 ;

3 **while** *iteração atual* $t \leq GER$ **do**

4 $AB(t) = \{s^1, ..., s^{|AB|}\} \leftarrow busca\_local(AB(t))$ // de acordo com Alg. 2 ;

5 $f_{AB}(t) = \{f(s^1), ..., f(s^{AB})\} \leftarrow avaliar\_anticorpos(AB(t))$;

6 $C(t), f_C(t) \leftarrow clonar\_anticorpos(AB(t), N, \beta, f_{AB}(t))$ // de acordo com Eq. 5 ;

7 $C^*(t), f_{C^*}(t) \leftarrow mutar\_anticorpos(C(t), f_C(t))$ // de acordo com Eq. 8 ;

8 $AB(t+1) \leftarrow selecionar\_clones(C^*(t), f_{C^*}(t), C_{best})$;

9 $AB(t+1) \leftarrow gerar\_anticorpos(|AB_d|)$;

10 **end**

No Algoritmo 3, $f(.)$ representa a avaliação de uma solução, de acordo com a Eq. 4. Na linha 1, uma permutação aleatória de modos é atribuída para cada atividade. Se a atribuição de modos for inviável com relação aos recursos não renováveis, uma heurística simples buscará viabilizar a atribuição de modo da seguinte maneira. Aleatoriamente, será selecionada uma atividade j com $|M_j| > 1$ e um modo $\bar{m_j}$ diferente do modo corrente, levando a uma nova atribuição de modo. Se essa nova atribuição de modo for tão boa quanto a anterior, então $\bar{m_j}$ será aceito como novo modo para j. Repete-se esse processo até que J atividades consecutivas

tenham sido exploradas sem sucesso ou até que a atribuição se torne viável com relação aos recursos não renováveis.

Os testes preliminares, após aplicar essa heurística, mostraram que aproximadamente 98% das soluções obtidas são viáveis, corroborando os resultados alcançados pelo mesmo procedimento em Hartmann (2001). Em seguida, a regra de prioridade LFT é utilizada para gerar uma lista de prioridade que será sequenciada pelo SGS Serial. Esse mesmo procedimento de geração de uma solução é utilizado na linha 9.

5 EXPERIMENTOS COMPUTACIONAIS

O CLONALG-MRCPSP foi implementado em linguagem C, usando compilador GNU GCC 4.8.2. Os testes foram realizados em um computador com processador Intel Core i3-3217U; 1.80GHz; 4GB RAM; e sistema operacional Ubuntu Linux. Foram utilizadas três classes de instâncias da biblioteca pública *Project Scheduling Problem Library* (PSPLIB) (cf. Kolisch and Sprecher (1997)). As características dessas classes são apresentadas na Tabela 2.

Tabela 2: Classes de instâncias PSPLIB

Classe	**# instâncias viáveis**	**# atividades**	**# modos de execução**	**# máxima de sucessores por atividade**	**# recursos renováveis**	**# recursos não renováveis**
J10	536	10	3	3	2	2
J20	554	20	3	3	2	2
J30	552	30	3	3	2	2

O algoritmo foi testado sobre os conjuntos J10, J20 e J30. Para as classes J10-J20, valores ótimos são conhecidos na literatura. Para o conjunto J30, soluções para as instâncias foram encontradas apenas pela utilização de métodos heurísticos. O percentual do desvio médio em relação às melhores soluções conhecidas, ou seja, soluções alcançadas por métodos exatos ou técnicas heurísticas, é calculado como:

$$Desvio_{medio} \leftarrow \frac{\bar{f} - f^*}{f^*} \times 100 \tag{9}$$

Para as classes J10 e J20, $Desvio_{medio}$ indica o percentual do desvio da média das soluções $\bar{f}$, para as 30 execuções, em relação ao valor ótimo f^* conhecido na literatura. Já para a classe J30, $Desvio_{medio}$ indica o percentual do desvio da média das soluções $\bar{f}$, para as 30 execuções, em relação às melhores soluções f^* conhecidas.

A literatura apresenta duas formas de critério de parada: tempo de processamento e número de soluções geradas. Os autores em Hartmann and Kolisch (2000) sugerem que, mesmo em diferentes arquiteturas, o número de soluções geradas é a maneira adequada de adotar como critério de parada. O motivo dado pelos autores é que o esforço de gerar um sequenciamento é o mesmo, independentemente do computador ou até mesmo das heurísticas utilizadas. Sendo assim, o critério de parada adotado nesse trabalho foi o número de soluções geradas, fixadas em 5000. De acordo com experimentos preliminares e observações extraídas da literatura,

os parâmetros foram configurados como: a população é dada por $|AB| = 200$ anticorpos; o número de anticorpos selecionados para clonagem é dado por $N = 0.5 \times |AB|$; $\beta = 0.1$; e a quantidade de inserção de novos anticorpos será dada por $d = 0.2 \times |AB|$. Após essas configurações, foram realizados os experimentos e os valores obtidos foram comparados com trabalhos da literatura conhecida pelos autores desse trabalho, que possuem o mesmo critério de parada e apresentam abordagens para resolução do MRCPSP. Os resultados comparativos são apresentados pelas Tabelas 3, 4 e 5.

Tabela 3: Resultados obtidos pelo algoritmo CLONALG-MRCPSP sobre os conjuntos J10 da PSPLIB para 5000 soluções.

Autores	**Soluções ótimas encontradas (%)**	$Desvio_{medio}$ **(%)**
Jozefowska et al. (2001)	85.6	1.16
Alcaraz et al. (2003)	-	0.24
Lova et al. (2009)	98.51	0.06
Van Peteghem and Vanhoucke (2009)	99.44	0.02
Tseng and Chen (2009)	95.16	0.33
Van Peteghem and Vanhoucke (2010)	99.63	0.01
Elloumi and Fortemps (2010)	97.31	0.14
Bilolikar et al. (2012)	97.39	0.16
Okada et al. (2014)	99.43	-
CLONALG-MRCPSP (este trabalho)	**99.68**	**0.01**

Tabela 4: Resultados obtidos pelo algoritmo CLONALG-MRCPSP sobre os conjuntos J20 da PSPLIB para 5000 soluções.

Autores	**Soluções ótimas encontradas (%)**	$Desvio_{medio}$ **(%)**
Jozefowska et al. (2001)	35.7	6.74
Alcaraz et al. (2003)	-	1.91
Lova et al. (2009)	-	0.87
Van Peteghem and Vanhoucke (2009)	81.59	0.7
Tseng and Chen (2009)	66.655	1.713
Van Peteghem and Vanhoucke (2010)	**85.74**	**0.57**
Elloumi and Fortemps (2010)	67.43	1.62
Bilolikar et al. (2012)	68.53	1.04
Okada et al. (2014)	**85.99**	-
CLONALG-MRCPSP (este trabalho)	75.55	1.09

Tabela 5: Resultados obtidos pelo algoritmo CLONALG-MRCPSP sobre os conjuntos J30 da PSPLIB para 5000 soluções.

Autores	**Melhores soluções encontradas (%)**	$Desvio_{medio}$ **(%)**
Jozefowska et al. (2001)	25.6	11.76
Van Peteghem and Vanhoucke (2009)	65.22	1.55
Tseng and Chen (2009)	-	18.332
Van Peteghem and Vanhoucke (2010)	**71**	**1.08**
Elloumi and Fortemps (2010)	-	-
Bilolikar et al. (2012)	48.91	11.86
Okada et al. (2014)	69.82	-
CLONALG-MRCPSP (este trabalho)	55.90	2.51

Os trabalhos citados nas Tabelas 3, 4 e 5 são encontrados em publicações destacadas na literatura para congressos ou periódicos internacionais. A comparação mais recente entre as abordagens citadas, conhecida pelos autores desse artigo, pode ser verificada em Van Peteghem and Vanhoucke (2014).

Como pode ser observado nas Tabelas 3, 4 e 5, o algoritmo CLONALG-MRCPSP oferece boa capacidade na resolução dos problemas testados, superando várias outras meta-heurísticas destacadas pela literatura. Pela Tabela 3, o algoritmo CLONALG-MRCPSP obteve soluções, com desvio médio para o ótimo, levemente superior ao algoritmo de Van Peteghem and Vanhoucke (2010). Claramente, a abordagem de Van Peteghem and Vanhoucke (2010) se mantém como a melhor meta-heurística para resolução do MRCPSP, utilizando como critério de parada 5000 sequenciamentos, como pode ser observado pelas Tabelas 4 e 5. Entretanto, a abordagem utilizada nesse trabalho ficou aproximadamente 15% abaixo do trabalho de Van Peteghem and Vanhoucke (2010) na determinação das melhores soluções e 1.43% abaixo em relação ao desvio médio das soluções, sendo esse resultado considerado relevante para os autores desse trabalho, na resolução do MRCPSP. O algoritmo CLONALG-MRCPSP apresentou baixo desvio médio para as classes testadas, evidenciando a capacidade de encontrar boas soluções quando a melhor solução conhecida não foi determinada. Os autores desse trabalho acrescentam que, para todas as três classes, a meta-heurística CLONALG-MRCPSP finalizou sua execução com 100% de soluções viáveis, o que torna relevante o resultado devido à dificuldade de encontrar soluções viáveis para o problema, conforme Drexl and Gruenewald (1993), Kolisch and Drexl (1997) e Kolisch and Drexl (1997).

Outra característica do algoritmo CLONALG-MRCPSP é apresentada pela Tabela 6. Essa tabela apresenta o tempo médio, para as 30 execuções, utilizado pela meta-heurística proposta para resolver as 3 classes apresentadas.

Tabela 6: Média no tempo de processamento, em segundos, para resolução dos problemas das classes J10, J20 e J30

Tempo de processamento (seg.)		
J10	**J20**	**J30**
0.5602	1.0888	1.9553

Pela Tabela 6, nota-se que o algoritmo CLONALG-MRCPSP demanda, em média, baixo tempo computacional para encontrar boas soluções em relação às classes de problemas testados. Como a classe J30 é aquela que oferece as maiores instâncias pela PSPLIB para o MRCPSP com minimização do *makespan* e fazendo uma relação entre as Tabelas 5 e 6, o algoritmo CLONALG-MRCPSP terminou sua execução em tempo médio de 1.95s determinando 55.9% das melhores soluções conhecidas. Quando tal solução não era encontrada, uma solução aproximada era determinada, sendo tal característica evidenciada pelo baixo desvio médio de 2.5%. Os autores do trabalho em discussão deixam claro que não são apresentadas comparações entre o tempo de busca, pois há diferença significativa entre a configuração dos computadores utilizados para teste em relação aos trabalhos apresentados nas Tabelas 3, 4 e 5. Além do mais, alguns desses autores, não apresentam informações sobre o tempo de busca de suas propostas.

De acordo com as discussões em Lova et al. (2006), Van Peteghem and Vanhoucke (2014) e Mika et al. (2015), os autores desse trabalho acrescentam que o algoritmo CLONALG-MRCPSP foi criado através de técnicas conhecidos no estado-da-arte.

6 CONCLUSÕES

Problemas de sequenciamento de projetos acontecem em grandes áreas econômicas, como pesquisa e desenvolvimento de projetos, desenvolvimento de softwares, Engenharia Civil, etc. Algoritmos que oferecem boas respostas a projetos que apresentam diversas formas de executar uma tarefa ganham destaque como boas ferramentas de apoio a decisão. Este trabalho tratou o problema de sequenciamento de projeto com restrição de recursos e múltiplos modos de execução, com o objetivo de minimizar o *makespan*.

Dada a complexidade do problema, foi proposta a meta-heurística Seleção Clonal (CLONALG), baseada em Sistemas Imunológicos Artificiais, para solucioná-lo. Na abordagem proposta neste artigo, uma solução inicial é gerada por meio de uma regra de prioridade e após, aprimorada por uma busca local. Então, itera-se no processo de evolução das soluções, utilizando o paradigma de um Sistema Imunológico. Nesse sistema, uma solução ou anticorpo, caso possua boa resposta para uma função de afinidade, é selecionada e clonada para maturação. Pequenas trocas acontecem sobre os clones para exploração da vizinhança da solução clonada. As soluções com melhores afinidades tendem a se proliferar e ótimos locais tendem a ser encontrados no espaço de busca.

Três classes de instâncias da PSPLIB foram submetidas a testes. Os resultados mostram que o algoritmo proposto foi capaz de determinar boas soluções para um grupo de instâncias testadas, em curto tempo de processamento. Quando a melhor solução conhecida pela literatura não foi determinada, o algoritmo proposto foi capaz de determinar soluções de boa qualidade, fato evidenciado pelo desvio médio para o melhor valor conhecido. Como propostas futuras, sugere-se agregar ao método características inerentes às instâncias apresentadas, pois algumas delas são mais complicadas em sua resolução, assim como explorar outras características específicas do problema proposto.

AGRADECIMENTOS

Os autores agradecem às agências CAPES, FAPEMIG e CNPq, assim como ao CEFET-MG, pelo apoio ao desenvolvimento do trabalho.

REFERÊNCIAS

Agarwal, R., Tiwari, M. and Mukherjee, S. (2007). Artificial immune system based approach for solving resource constraint project scheduling problem, *The International Journal of Advanced Manufacturing Technology* **34**(5-6): 584–593.

Alcaraz, J., Maroto, C. and Ruiz, R. (2003). Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms, *Journal of the Operational Research Society* **54**(6): 614–626.

Bilolikar, V., Jain, K. and Sharma, M. (2012). An annealed genetic algorithm for multi mode resource constrained project scheduling problem, *International Journal of Computers and Applications* **60**(1): 36–42.

Blazewicz, J. (1986). *Scheduling under resource constraints: Deterministic models*, Vol. 7, JC Baltzer.

Bouleimen, K. and Lecocq, H. (2003). A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version, *European Journal of Operational Research* **149**(2): 268–281.

Browning, T. R. and Yassine, A. A. (2010). Resource-constrained multi-project scheduling: Priority rule performance revisited, *International Journal of Production Economics* **126**(2): 212–228.

Brucker, P., Drexl, A., Mohring, R., Neumann, K. and Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models, and methods, *European journal of operational research* **112**(1): 3–41.

Coello, C. A. C., Rivera, D. C. and Cortes, N. C. (2003). Use of an artificial immune system for job shop scheduling, *Artificial Immune Systems*, Springer, pp. 1–10.

Cui, J. and Yu, L. (2014). An efficient discrete particle swarm optimization for solving multi-mode resource-constrained project scheduling problem, *Industrial Engineering and Engineering Management (IEEM), 2014 IEEE International Conference on*, IEEE, pp. 858–862.

Damak, N., Jarboui, B., Siarry, P. and Loukil, T. (2009). Differential evolution for solving multi-mode resource-constrained project scheduling problems, *Computers & Operations Research* **36**(9): 2653–2659.

De Castro, L. N. and Von Zuben, F. J. (2002). Learning and optimization using the clonal selection principle, *Evolutionary Computation, IEEE Transactions on* **6**(3): 239–251.

Demeulemeester, E. L. (2002). *Project scheduling: a research handbook*, Vol. 102, Springer.

Drexl, A. and Gruenewald, J. (1993). Nonpreemptive multi-mode resource-constrained project scheduling, *IIE transactions* **25**(5): 74–81.

Elloumi, S. and Fortemps, P. (2010). A hybrid rank-based evolutionary algorithm applied to multi-mode resource-constrained project scheduling problem, *European Journal of Operational Research* **205**(1): 31–41.

Engin, O. and Döyen, A. (2004). A new approach to solve hybrid flow shop scheduling problems by artificial immune system, *Future generation computer systems* **20**(6): 1083–1095.

Hart, E., Ross, P. and Nelson, J. (1998). Producing robust schedules via an artificial immune system, *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, IEEE, pp. 464–469.

Hartmann, S. (2001). Project scheduling with multiple modes: a genetic algorithm, *Annals of Operations Research* **102**(1-4): 111–135.

Hartmann, S. and Kolisch, R. (2000). Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem, *European Journal of Operational Research* **127**(2): 394–407.

Jarboui, B., Damak, N., Siarry, P. and Rebai, A. (2008). A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems, *Applied Mathematics and Computation* **195**(1): 299–308.

Jozefowska, J., Mika, M., Rozycki, R., Waligora, G. and Weglarz, J. (2001). Simulated annealing for multi-mode resource-constrained project scheduling, *Annals of Operations Research* **102**(1-4): 137–155.

Ju, C. and Chen, T. (2012). Simplifying multiproject scheduling problem based on design structure matrix and its solution by an improved ainet algorithm, *Discrete Dynamics in Nature and Society* **2012**.

Kolisch, R. (1996a). Efficient priority rules for the resource-constrained project scheduling problem, *Journal of Operations Management* **14**(3): 179–192.

Kolisch, R. (1996b). Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation, *European Journal of Operational Research* **90**(2): 320–333.

Kolisch, R. and Drexl, A. (1997). Local search for nonpreemptive multi-mode resource-constrained project scheduling, *IIE transactions* **29**(11): 987–999.

Kolisch, R. and Hartmann, S. (1999). *Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis*, Springer.

Kolisch, R. and Hartmann, S. (2006). Experimental investigation of heuristics for resource-constrained project scheduling: An update, *European Journal of Operational Research* **174**(1): 23–37.

Kolisch, R. and Sprecher, A. (1997). Psplib - a project scheduling problem library, *European Journal of Operational Research* **96**(1): 205–216.

Lova, A., Tormos, P. and Barber, F. (2006). Multi-mode resource constrained project scheduling: Scheduling schemes, priority rules and mode selection rules, *Inteligencia artificial* **30**(10): 69–86.

Lova, A., Tormos, P., Cervantes, M. and Barber, F. (2009). An efficient hybrid genetic algorithm for scheduling projects with resource constraints and multiple execution modes, *International Journal of Production Economics* **117**(2): 302–316.

Mika, M., Waligóra, G. and Weglarz, J. (2015). Overview and state of the art, *Handbook on Project Management and Scheduling Vol. 1*, Springer, pp. 445–490.

Okada, I., Takahashi, K., Zhang, W., Zhang, X., Yang, H. and Fujimura, S. (2014). A genetic algorithm with local search using activity list characteristics for solving resource-constrained project scheduling problem with multiple modes, *IEEJ Transactions on Electrical and Electronic Engineering* **9**(2): 190–199.

Sprecher, A. and Drexl, A. (1998). Multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm, *European Journal of Operational Research* **107**(2): 431–450.

Sprecher, A., Hartmann, S. and Drexl, A. (1997). An exact algorithm for project scheduling with multiple modes, *Operations-Research-Spektrum* **19**(3): 195–203.

Sun, K. and Yang, G.-k. (2008). Hybrid artificial immune system and extremal optimization algorithm for permutation flowshop scheduling problem, *Journal of Shanghai University (English Edition)* **12**: 352–357.

Truong, V. X. and Kachitvichyanukul, V. (2007). A hybrid pso algorithm for multi-mode resource-constrained project scheduling problems, *Proceedings of the 8th Asia Pacific Industrial Engineering and Management System Conference (APIEMS 2007)*, p. 1.

Tseng, L.-Y. and Chen, S.-C. (2009). Two-phase genetic local search algorithm for the multimode resource-constrained project scheduling problem, *Evolutionary Computation, IEEE Transactions on* **13**(4): 848–857.

Van Peteghem, V. and Vanhoucke, M. (2009). An artificial immune system for the multi-mode resource-constrained project scheduling problem, *Evolutionary computation in combinatorial optimization*, Springer, pp. 85–96.

Van Peteghem, V. and Vanhoucke, M. (2010). A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem, *European Journal of Operational Research* **201**(2): 409–418.

Van Peteghem, V. and Vanhoucke, M. (2014). An experimental investigation of metaheuristics for the multi-mode resource-constrained project scheduling problem on new dataset instances, *European Journal of Operational Research* **235**(1): 62–72.

Zhang, H., Tam, C. and Li, H. (2006). Multimode project scheduling based on particle swarm optimization, *Computer-Aided Civil and Infrastructure Engineering* **21**(2): 93–103.