LIII Simpósio Brasileiro de Pesquisa Operacional
João Pessoa - PB, 3 a 5 de novembro de 2021

Algoritmos eficientes para o Problema de Escalonamento de Projetos com Múltiplas Habilidades

André Renato Villela da Silva
Universidade Federal Fluminense - Instituto de Ciência e Tecnologia
R. Recife s/n, Jardim Bela Vista, Rio das Ostras - RJ
arvsilva@id.uff.br

RESUMO

O Problema de Escalonamento de Projetos com Múltiplas Habilidades (MS-RCPSP) é uma variante do clássico Problema de Escalonamento de Projetos com Restrição de Recursos. Nesta variante, tanto as tarefas quantos os recursos possuem níveis pré-determinados de habilidades. Desta forma, um recurso pode estar apto a ser alocado a uma tarefa específica, mas não a todas. O problema pode ser aplicado a projetos que envolvam a utilização de recursos humanos, por exemplo. O artigo apresenta um Algoritmo Evolutivo e modelagens matemáticas de Programação por Restrições e Programação Inteira Mista para tratar do problema. Os resultados obtidos mostram o bom desempenho dos métodos propostos, inclusive com a descoberta de novas soluções ótimas.

PALAVRAS CHAVE. Multi-Skill RCPSP, Programação por Restrições, Algoritmos Evolutivos, Métodos híbridos.

Tópicos: Otimização Combinatória; Programação Matemática

ABSTRACT

The Multiple Skills Project Scheduling Problem (MS-RCPSP) is a variant of the classical Resource Constrained Project Scheduling Problem. In this variant, both tasks and resources have predetermined skill levels. In this way, a resource may be able to be allocated to a specific task, but not all. The problem can be applied to projects that involve the use of human resources, for example. The article presents an Evolutionary Algorithm and mathematical modeling of Constraint Programming and Mixed Integer Programming to address the problem. The results obtained show the good performance of the proposed methods, including the discovery of new optimal solutions.

KEYWORDS. Multi-Skills RCPSP. Constraint Programming. Evolutionary Algorithms, Hybrid Methods.

Paper topics: Combinatorial Optimization; Mathematical Programming

1. Introdução

Os Problemas de Escalonamento de Projetos constituem uma grande classe de problemas de Otimização Combinatória e são muito estudados por permitirem a modelagem de diversas situações reais. Os projetos são conjuntos de pequenas atividades individuais, chamadas de tarefas, que precisam ser executadas para que o projeto seja considerado concluído.

A execução das tarefas é feita mediante a utilização de recursos específicos, que precisam estar disponíveis em uma determinada unidade de tempo para que a tarefa seja iniciada. Cada tarefa possui uma duração (medida em unidades de tempo), normalmente impossível de ser interrompida (não-preemptiva). Após a execução das tarefas, os recursos empregados podem voltar a ficar disponíveis (renováveis) ou não (não-renováveis). A execução das tarefas deve ocorrer dentro de um intervalo contíguo de unidades de tempo, chamado de horizonte de planejamento.

Algumas tarefas também precisam ser executadas antes de outras. Essa relação de precedência normalmente é do tipo *finish-to-start*, ou seja, uma tarefa predecessora precisa ser totalmente concluída antes da tarefa sucessora começar. Esse tipo de problema é chamado de Problema de Escalonamento de Projeto com Restrição de Recursos (RCPSP, na sigla em inglês). O objetivo do problema usualmente é executar todas as tarefas no menor tempo possível, chamado de *makespan* do projeto. O RCPSP encontra aplicações nas mais diversas áreas: processos de manufatura, acompanhamento de obras da construção civil, escalas de tripulação, desenvolvimento de *software*, dentre outros. Por ser um modelo muito básico e conciso, o RCPSP apresenta diversas variantes que agregam características e/ou restrições às tarefas, aos recursos ou à função-objetivo que se deseja otimizar.

Este artigo trata do Problema de Escalonamento de Projetos com Múltiplas Habilidades (MS-RCPSP). Nesta variante, as tarefas possuem requisitos específicos chamados de habilidades, que fazem com que ela só possa ser executada se associada a recursos específicos que atendam a esses requisitos. Por exemplo, podemos imaginar que o desenvolvimento de um *software* de grande porte é feito por meio da implementação de pequenos módulos, que exigem conhecimentos de áreas como: banco de dados, redes, interfaces, processamento paralelo, criptografia entre outros. Alguns dos profissionais contratados apresentam mais familiaridade em algumas dessas áreas do que em outras, o que os tornam aptos a desenvolverem apenas alguns módulos, mas não todos. O objetivo, portanto, é alocar os profissionais aos módulos adequados, de forma a minimizar o tempo de desenvolvimento do projeto.

O restante deste artigo é dado da seguinte forma. Na Seção 2, o MS-RCPSP é formalmente descrito. Também é explicado como o problema será tratado neste artigo. A Seção 3 destaca a literatura sobre o problema. As Seções 4 e 5 apresentam os métodos propostos e os resultados computacionais, respectivamente. A Seção 6 traz as conclusões deste trabalho.

2. Definição formal e tratamento do problema

Uma descrição formal do MS-RCPSP é dada a seguir. Seja r um dos R recursos renováveis à disposição tal que $r \in 1...|R|$ e K um conjunto de habilidades. Cada recurso r possui um valor não-negativo para cada habilidade $k \in K$, dado por r_k, que representa o nível de familiaridade do recurso r com a habilidade k.

Um projeto P é composto por tarefas $y_i (i = 1...n)$. P pode ser representado pelo grafo $G = V, E$, onde V é o conjunto de tarefas e E é o conjunto de relações de precedência entre as tarefas de V. Cada tarefa, para ser executada, requer a utilização de um recurso r tal que $r_k \geq x_i^k$, $\forall k \in K$, onde x_i^k representa o nível mínimo de habilidade k necessário para a execução da tarefa i. Uma vez alocado o recurso k à tarefa y_i, esse recurso só voltará a ficar disponível após o encerramento da tarefa. Só existe uma unidade disponível de cada recurso.

Cada tarefa y_i tem duração fixa de d_i unidades de tempo, independente do recurso que ela utilize, e sua execução é não-preemptiva. Também podem existir relações de precedência entre duas tarefas y_i e y_j de forma que a tarefa y_j só possa começar a ser executada após o encerramento da tarefa y_i. Enquanto um recurso r está sendo utilizado, é necessário pagar um salário r_s (valor positivo) por cada unidade tempo. A soma de todos os salários pagos, chama-se custo total do projeto, denotada por C_P.

O objetivo do problema é reduzir o tempo de encerramento do projeto, chamado *makespan*, e o custo total, sem violar as restrições de precedência e de disponibilidade de recursos. Normalmente, é utilizada uma função mono-objetivo com pesos para a parcela do *makespan* e do custo total.

A Tabela 1 mostra um exemplo com 4 tarefas e 3 recursos. Nesta tabela, os pares (X, Y) indicam uma habilidade X e um nível de familiaridade Y. A Tarefa1 requer a habilidade 1 com nível de familiaridade 1 e a habilidade 2 com nível 1 (valores entre parênteses). Somente o recurso R1 pode atender esses requerimentos. A mesma situação ocorre para a Tarefa2. A Tarefa3 pode ser atendida tanto pelo recurso R2 quanto pelo recurso R3.

Tabela 1: Exemplo de possibilidade de alocação tarefas, mediante recursos e habilidades necessárias

Recurso	Tarefa1 (1,1) (2,1)	Tarefa2 (2,1)	Tarefa3 (3,1)	Tarefa4 (1,3)
R1 (1,2) (2,2)	possível	possível	impossível	impossível
R2 (3,2)	impossível	impossível	possível	impossível
R3 (1,3) (3,2)	impossível	impossível	possível	possível

Como proposto em [Myszkowski et al., 2015a], um único recurso deve ser alocado a cada tarefa. Uma vez que os requisitos das habilidades e dos níveis de familiaridade são dados de entrada do problema, é possível em tempo computacional $\mathcal{O}(nRK)$ processar esses dados e saber quais recursos podem ser utilizados pelas tarefas. Como normalmente os valores de R e K são pequenos comparados ao valor de n, esse processamento não é custoso computacionalmente.

Como será explicado na Seção 3, os artigos mais recentes sobre o MS-RCPSP abordam apenas a minimização do custo total ou a minimização do *makespan*, não tendo sido utilizadas combinações desses fatores. Este artigo trata apenas da minimização do *makespan*, uma vez que a minimização apenas do custo total pode ser obtida de forma simples, fazendo, para cada tarefa, a alocação do recurso que tiver o menor salário relacionado.

3. Revisão da literatura

Os Problemas de Escalonamento de Projetos com Restrição de Recursos são estudados há muito tempo e possuem uma vasta literatura científica. Existem diversos *surveys* contribuindo com definições de conceitos e classificação de problemas e variantes como em [Brucker et al., 1999; Hartmann e Briskorn, 2010]. Desta forma, praticamente todas as abordagens existentes já foram empregadas, desde métodos exatos [Demeulemeester e Herroelen, 1992], heurísticas simples [Boctor, 1990; Kreter et al., 2018] e até abordagens híbridas [Debels et al., 2006; Valls et al., 2008; Myszkowski et al., 2015a].

Em [Myszkowski et al., 2015b], foi proposta uma primeira heurística gulosa, chamada Greedy, onde as tarefas são alocadas de forma sequencial, primeiro as tarefas que têm mais sucessoras até as tarefas com nenhuma sucessora. O início da execução da cada tarefa é igual ao mínimo entre o tempo de término da última predecessora e o tempo de disponibilidade de um recurso previamente alocado. O objetivo do problema, nesse artigo, incluía uma combinação entre a minimização do *makespan* e do custo total do projeto. No entanto, a métrica utilizada para determinar a duração do *makespan* envolvia dias úteis (excluindo-se finais de semana e feriados locais) o que dificultava

a comparação com outras técnicas, onde o resultado era constituído apenas de horas de execução das tarefas.

Algumas meta-heurísticas foram propostas, tomando como base esse algoritmo guloso, como um algoritmo de Colônia de Formigas, chamado HAntCO [Myszkowski et al., 2015b], e um algoritmo GRASP [Myszkowski e Siemiénski, 2016]. Em Myszkowski et al. [2018], foi proposto um algoritmo de Evolução Diferencial, chamado DEGR. A solução é codificada como um vetor de números reais que passava por um processo de transformação, correção e avaliação, dentro do mecanismo de Evolução Diferencial. Os autores também propuseram operadores de mutação e cruzamento entre as soluções como forma de melhorar os resultados do algoritmo.

Por fim, em [Dai et al., 2018], foi proposto um algoritmo de Busca Tabu Iterativa (ITS). Neste artigo, o problema também é tratado possuindo um passo de deteriorização. Isso modela a fadiga do uso constante de recursos, que faz com que a duração das tarefas aumente, caso o tempo de início dela seja maior que um determinado limite.

4. Métodos Propostos

Nesta seção, serão apresentados e detalhados os métodos estudados para resolução do MS-RCPSP. Por se tratar de um problema NP-difícil, não são conhecidos algoritmos eficientes para encontrar soluções ótimas em tempo computacional razoável. Assim, o uso de métodos heurísticos são justificáveis principalmente para instâncias de grande porte. No entanto, este artigo também apresenta modelos matemáticos para balizar os resultados encontrados pelas abordagens estocásticas.

4.1. Algoritmo Evolutivo

Os Algoritmos Evolutivos formam uma classe de heurísticas populacionais, que tratam simultaneamente um conjunto de soluções, chamadas de indivíduos. Os indivíduos são modificados e recombinados entre si ao longo de várias etapas, chamadas de gerações. O Algoritmo Evolutivo, no decorrer das gerações, procura melhorar a qualidade dos indivíduos, tentando extrair e aprimorar as melhores características deles. O Algoritmo Evolutivo proposto neste artigo é chamado de EA e será descrito a seguir.

4.1.1. Codificação do Indivíduo

Na representação proposta, cada indivíduo é representado como um vetor *n*-dimensional de número reais, onde o valor da posição *i* representa a prioridade de se escolher a tarefa *i*. O algoritmo de escalonamento vai, repetidamente, selecionar a tarefa com maior prioridade que ainda não esteja escalonada e verificar a disponibilidade de cada recurso que possa ser utilizado para executar a tarefa. O recurso escolhido será aquele que permitir a tarefa ser encerrada o mais cedo possível (caso haja mais de um recurso nessa situação, um deles é escolhido aleatoriamente). O recurso então é associado à tarefa no tempo correspondente e a tarefa é considerada escalonada. Desta forma, a decodificação ocorre de maneira heurística e determinística, de acordo com o algoritmo explicado.

4.1.2. Geração da população Inicial

Como os Algoritmos Evolutivos trabalham com a interação de um conjunto de indivíduos, é necessário estabelecer uma primeira população para que os operadores evolutivos possam ser executados. A população inicial deve ser criada de tal forma que os indivíduos apresentem um bom nível de aptidão, mas também um nível satisfatório de diversificação, já que serão combinados entre si nas gerações futuras. Foram propostos 3 critérios para a geração desta população inicial, explicados a seguir. Em todos os casos, a prioridade p_i de uma tarefa *i* é definida em função de uma variável aleatória *u* com distribuição uniforme no intervalo [0,100].

Critério 0: a prioridade de cada tarefa é dada apenas pelo valor da variável aleatória correspondente ($p_i = u$). **Critério 1**: a prioridade leva em consideração a quantidade de modos de

execução de uma tarefa m_i, de forma a favorecer tarefas que tenham poucos modos de execução ($p_i = (|Q| - m_i + 1) \times u$), onde Q é a quantidade total de recursos da instância. **Critério 2**: a prioridade leva em consideração a quantidade de tarefas sucessoras diretas (s_i) de uma tarefa, de forma a favorecer tarefas que tenham mais sucessoras ($p_i = (s_i + 1) \times u$).

A população inicial (e as demais gerações) do algoritmo EA apresentam uma quantidade fixa de indivíduos, dada pelo parâmetro "totalIndiv". Uma vez que todos os critérios propostos apresentam significativo impacto causado pela variável aleatória u - o que pode fazer com que a aptidão desses indivíduos não seja tão boa), serão gerados $10 \times totalIndiv$ indivíduos dos quais apenas os "totalIndiv" melhores efetivamente farão parte da população inicial. Os demais indivíduos serão descartados imediatamente.

4.1.3. Intensificação

No algoritmo EA, a cada *k* gerações, os melhores indivíduos da população atual passam por uma etapa de intensificação. Para esta etapa, foi proposto que as prioridades das tarefas desses indivíduos sejam trocadas de duas em duas e o algoritmo de escalonamento seja reexecutado. Caso a solução apresente uma aptidão melhor do que a solução original, a alteração das prioridades é mantida; caso contrário, é desfeita. Este operador visa provocar buscas locais na vizinhança de soluções de boa qualidade, verificando se vetores com prioridades semelhantes poderiam apresentar resultado final melhor.

4.1.4. Seleção dos Pais e Cruzamento

O Algoritmo Evolutivo EA utilizou uma estratificação da população em três classes: a classe A é composta pelos 20% melhores indivíduos; a classe C é composta pelos 20% piores indivíduos; e a classe B, pelos indivíduos restantes. Sempre são escolhidos um pai da classe A e outro da classe B.

O operador de cruzamentos do algoritmo EA calcula a prioridade dos filhos como sendo a média das prioridades dos pais, para cada tarefa. A esta média, é somada ou subtraída um pequena porcentagem, baseada em um parâmetro λ, que com o passar das gerações aumenta de módulo. O objetivo principal desta alteração é permitir que a diversificação dos indivíduos aumente com o passar do tempo.

4.1.5. Seleção Natural e Critério de Parada

A seleção natural escolhida utiliza o modelo elitista, onde os melhores indivíduos, dentre pais e filhos, sobrevive para geração seguinte. O critério de parada adotado é um limite de tempo fornecido como parâmetro para o algoritmo.

O Algoritmo 1 mostra um pseudo-código do algoritmo EA. A população inicial é gerada na linha 1, informando-se a quantidade de indivíduos desejados e o critério a ser utilizado para gerá-los. A recombinação (escolha dos pais e cruzamento) é realizada na linha 5 e a seleção natural, na linha 6. De tempos em tempos (linhas 9-11), o valor do parâmetro λ aumenta, para permitir que indivíduos filhos mais diversificados em relação aos pais possam ser gerados. Por fim, em alguns momentos (linhas 2-14) alguns dos melhores indivíduos passam pelo mecanismo de intensificação descrito. Os parâmetros utilizados serão informados na Seção 5.

4.2. Comparação com o Algoritmo BRKGA

O algoritmo BRKGA é bastante conhecido e utilizado na área de Otimização Combinatória [Gonçalves et al., 2009]. Embora compartilhe muitos aspectos semelhantes com o algoritmo EA, existem algumas diferenças que merecem destaque.

A codificação dos indivíduos é feita por meio das chaves aleatórias (*random-keys*), que são valores reais restritos ao intervalo [0,1). Isso pode ser facilmente obtido a partir do algoritmo

Algorithm 1 Pseudo-código do Algoritmo EA

```
pop ← PopulacaoInicial(totalIndiv,critério)
geracao ← 1
melhorSol ← MelhorIndivIduo(pop)
while tempo total não excedido do
    filhos ← Recombinacao(pop)
    pop ← SelecaoNatural(pop+filhos)
    melhorSol ← MelhorIndividuo(pop)
    geracao ← geracao + 1
    if geracao mod mesmoLambda = 0 then
        lambda = lambda * incLambda
    end if
    if geracao mod intGer = 0 then
        Intensificacao(pop)
    end if
end while
return melhorSol
```

EA, dividindo-se todos os valores de prioridade por uma constante M, grande o suficiente para que o resultado fique sempre dentro do intervalo [0,1). No BRKGA, não existe intensificação dos indivíduos nem um operador de mutação específico.

A estratificação da população também é feita em 3 classes e a escolha dos pais também seleciona um indivíduo da classe A e um indivíduo da classe B. No entanto, todos os indivíduos da classe A (elite) são repassados à geração seguinte. Todos os indivíduos da classe C são descartados e substituídos por novos indivíduos. A geração seguinte é então completada com filhos gerados a partir do cruzamento entre os pais selecionados.

O cruzamento também é diferente, uma vez que utiliza um parâmetro $p_e \in [0, 1)$ para definir o valor da chave aleatória de cada tarefa do indivíduo filho. Caso uma variável aleatória tenha valor menor do que p_e, a chave aleatória do pai proveniente da classe A é repassada ao filho. Caso contrário, a chave aleatória do pai da classe B é repassada.

4.3. Modelagem por Programação por Restrições

Devido aos recentes avanços na eficiência dos algoritmos de resolução, a modelagem baseada em Programação por Restrições vem ganhando bastante destaque, principalmente na área de Problemas de Escalonamento de Projetos [Gérault et al., 2018; Ochoa et al., 2019; Kreter et al., 2018; Hojabri et al., 2018].

Como não está limitada pelo uso de expressões lineares, a Programação por Restrições pode utilizar diversos tipos de funções, inclusive algumas próprias para tratamento de intervalos de valores, que são normalmente utilizados para representar tarefas. Em Programação por Restrições, os intervalos correspondem a pares *(s,f)* de valores, onde s corresponde ao início de um intervalo e f ao seu fim. Desta forma, utilizando intervalos para representar tarefas, é possível estabelecer relações de precedência entre duas tarefas i e j, por exemplo, com a restrição $s_j \geq f_i$.

Tabela 2: Variáveis e constantes da modelagem CP

Notação	Tipo	Descrição
y_i	Var. Intervalo	Representa a tarefa i
x_i^r	Var. Intervalo	Representa a tarefa i, executada com o recurso r
d_i	Constante	Representa a duração da tarefa i
R_i	Constante	Representa o conjunto de recursos (modos) disponíveis para a tarefa i

A Tabela 2 apresenta as variáveis utilizadas na modelagem e uma breve descrição para as mesmas. A utilização de um recurso r corresponde a um modo de execução de uma tarefa, caso o problema seja visto sob perspectiva do MMRCPSP.]

$$minimize \quad max(endOf(x)) \tag{1}$$
$$s.t. \quad lenght(y_i) = d_i \quad \forall_{i=1,..,n} \tag{2}$$
$$presenceOf(y_i) = true \quad \forall_{i=1,..,n} \tag{3}$$
$$alternative(y_i, x_i^r) \quad \forall_{i=1,..,n} \quad \forall_{r \in R_i} \tag{4}$$
$$endBeforeStart(y_i, y_j) \quad \forall_{i=1,..,n} \quad \forall_{(i,j) \in E} \tag{5}$$
$$cumulative(presenceOf(x_i^r), 1) \quad \forall_{i=1,..,n} \quad \forall_{r \in R_i} \tag{6}$$

A função-objetivo (Eq. 1) busca minimizar maior o tempo de encerramento das tarefas do projeto. A duração e a presença das tarefas são modeladas pelas Eqs. (2) e Eqs. (3), respectivamente. As Eqs. (4) modelam que cada tarefa y_i deve ser executada utilizando um e somente um recurso r disponível para ela. As Eqs. (5) indicam que as relações de precedência entre as tarefas devem ser respeitadas. De acordo com as Eqs. (6), durante a execução de uma tarefa, o recurso r escolhido não poderá ser utilizado mais de uma vez, ou seja, o recurso não poderá estar alocado a mais de uma tarefa em qualquer unidade de tempo.

4.4. Modelagem por Programação Inteira Mista

A Programação Inteira Mista, por somente poder empregar expressões lineares, normalmente precisa contar com o uso de alguns artifícios para modelar corretamente alguns tipos de problema. Na classe de Problemas de Escalonamento de Projetos, é comum discretizar o horizonte de planejamento em T unidades de tempo para poder manipulá-las individualmente. Também é usual criar uma tarefa artificial y_{n+1}, com duração $d_{n+1} = 0$, sucessora de todas as demais tarefas, para marcar o encerramento do projeto.

Na modelagem proposta, as variáveis binárias $x_{i,t}^r$ possuem valor igual a 1, se a tarefa i inicia no tempo t utilizando o recurso r; elas valem 0, caso contrário. As variáveis inteiras s_i marcam o tempo de início das tarefas.

$$minimize \quad s_{n+1} \tag{7}$$
$$s.t. \quad \sum_{r \in M_i} \sum_{t=0}^{T} x_{i,t}^r = 1 \quad \forall_{i=1,..,n+1} \tag{8}$$
$$s_i = \sum_{r \in M_i} \sum_{t=0}^{T} t \cdot x_{i,t}^r \quad \forall_{i=0,..,n+1} \tag{9}$$
$$s_j \geq s_i + D_i \quad \forall_{(i,j) \in E} \tag{10}$$
$$\sum_{i=1}^{n+1} \sum_{t'=t-D_i+1}^{t} x_{i,t'}^r \leq 1 \quad \forall_{t=0,..,T} \quad \forall_{r \in R} \tag{11}$$

A Função-Objetivo (Eq. 7) busca minimizar o *makespan*, dado pelo início da execução da tarefa artificial $n + 1$. As Eqs. (8) garantem que cada tarefa será executada uma única vez. As Eqs. (9) associam as variáveis $x_{i,t}^r$ com as variáveis s_i. As relações de precedência são preservadas pelas Eqs. (10). Por fim, pelas Eqs. (11), a cada instante de tempo t, só pode haver no máximo uma tarefa que esteja usando o recurso r e ainda não tenha terminado no tempo t.

5. Resultados Computacionais

Foi utilizada a versão 12.9 tanto do CPLEX (Programação Inteira Mista) como do CP Optimizer (Programação por Restrições) e foi estabelecido um limite máximo de 1800 segundos para cada execução de cada método. Os demais parâmetros dos dois otimizadores foram mantidos com os valores padrão. O sistema computacional utilizado foi um notebook Intel I7-3630QM, com 8GB de memória RAM e sistema operacional Windows 8.0.

Para os experimentos computacionais, foram utilizadas as mesmas instâncias de [Myszkowski et al., 2018], que fazem parte do Projeto iMOPSE (intelligent Multi Objective Project Scheduling Environment) e podem ser acessadas a partir do endereço eletrônico `http://imopse.ii.pwr.wroc.pl/`. As instâncias possuem até 200 tarefas por projeto, até 40 recursos diferentes e conjuntos de 9 ou 15 habilidades distribuídas pelas tarefas e pelos recursos.

A Tabela 3 apresenta os principais resultados de métodos heurísticos da literatura: HAntCO [Myszkowski et al., 2015a], GRASP [Myszkowski e Siemiénski, 2016], DEGR [Myszkowski et al., 2018], Greedy [Myszkowski et al., 2015b] e ITS [Dai et al., 2018]. Destaca-se o algoritmo DEGR, que obteve o melhor resultado em quase todas as instâncias utilizadas. Infelizmente, poucas informações foram fornecidas a respeito das execuções desses algoritmos. O tempo computacional e cada execução ou o ambiente computacional utilizado foram omitidos nos respectivos artigos.

A Tabela 4 apresenta os resultados dos dois modelos matemáticos propostos. A coluna CP refere-se ao modelo de Programação por Restrições executado pelo CP Optimizer. A coluna ”CPLEX+CP”representa a modelagem de Programação Inteira executada pelo otimizador CPLEX, tendo o horizonte de planejamento H igual ao valor da melhor solução obtida pela Programação por Restrições. A coluna $CPLEX + CP_{Sol}$ indica que além do horizonte de planejamento, a modelagem de Programação por Restrições também forneceu a solução obtida como solução inicial para o otimizador CPLEX. As células vazias das colunas ”Tempo(s)”indicam que o tempo limite de 1800 segundos de execução foi atingido sem conclusão da otimização. As células vazias da coluna ”Sol.”do método $CPLEX + CP$ indicam que não foi encontrada solução factível.

Pela Tabela 5 nota-se uma clara distinção entre as abordagens. Enquanto o CP Optimizer encontrou soluções factíveis para todas as instâncias, o CPLEX teve enorme dificuldade de encontrar uma solução factível dentro do limite de tempo (apenas 7 em 36 instâncias testadas). Pode-se imaginar que parte dessa dificuldade se deve a trabalhar com um parâmetro (o horizonte de planejamento) fornecido por outro método. No Programa Inteiro Misto proposto neste artigo, grande quantidade de variáveis binárias (y) são criadas em função das unidades de tempo do horizonte.

Dessa forma, um valor muito elevado faria a Programação por Restrições ter que operar uma quantidade significativa dessas variáveis. Por outro lado, se o parâmetro fornecido fosse realmente ruim, provavelmente o CPLEX conseguiria melhorar o *makespan*, ainda mais quando uma solução factível também fosse fornecida. No entanto, isso não ocorreu como se pode ver na coluna $CPLEX + CP_{Sol}$, comparada à coluna CP. Além disso, o método $CPLEX + CP_{Sol}$ conseguiu provar em 4 instâncias que a solução passada pelo CP era de fato a ótima.

O último experimento envolveu o Algoritmo Evolutivo proposto nesse artigo, juntamente com o algoritmo BRKGA. Dentre os critérios propostos para geração dos indivíduos da população inicial, o critério 2 (favorecimento às tarefas com maior número de sucessoras diretas) foi o que obteve o melhor resultado, bem à frente dos demais. O Algoritmo Evolutivo com esse critério, foi denominado EA2 na Tabela 5. Os parâmetros utilizados foram: ”totalIndiv”=50; ”lambda inicial”=0.1; ”incLambda”=0.1; ”mesmoLambda”= 100; ”intGer”= 10000.

O operador de cruzamento do EA2 gera os filhos calculando as médias das prioridades dos pais para cada tarefa e aplicando, em seguida, um fator λ que pode aumentar ou reduzir ligei-

Tabela 3: Comparação das melhores soluções da literatura. O método "Greedy"é determinístico. Em negrito estão os melhores resultados.

	HAntCO		GRASP		DEGR		Greedy	ITS
Instância	Melhor	Média	Melhor	Média	Melhor	Média		Melhor
100_5_20_9_D3	437	442	401	402	**392**	393	625	
100_5_22_15	504	505	503	504	**484**	485	630	485
100_5_46_15	604	604	552	556	**529**	529	693	544
100_5_48_9	521	523	510	510	490	490	779	**471**
100_5_64_9	507	515	494	494	**474**	475	597	476
100_5_64_15	516	523	501	502	**481**	483	640	486
100_10_26_15	266	270	251	258	**234**	235	370	239
100_10_27_9_D2	294	306	221	223	**215**	220	646	
100_10_47_9	297	302	263	264	**255**	256	549	259
100_10_48_15	278	286	256	257	**244**	245	344	251
100_10_64_9	287	296	255	257	**243**	246	533	251
100_10_65_15	281	287	256	260	**244**	245	426	258
100_20_22_15	161	170	134	137	**130**	131	353	133
100_20_23_9_D1	219	223	**172**	172	**172**	172	617	
100_20_46_15	194	199	170	174	**164**	164	394	176
100_20_47_9	180	187	133	140	**125**	128	390	139
100_20_65_9	180	185	135	135	**126**	129	408	143
100_20_65_15	218	220	213	213	240	240	310	**205**
200_10_50_9	546	556	506	508	**485**	488	817	490
200_10_50_15	529	543	524	528	**487**	488	763	519
200_10_84_9	571	581	526	527	**507**	510	999	508
200_10_85_15	526	538	496	498	**475**	478	706	482
200_10_128_15	522	528	491	496	**461**	463	780	507
200_10_135_9_D6	1115	1133	**584**	617	608	693	1426	
200_20_54_15	336	349	304	308	**258**	261	488	292
200_20_55_9	313	317	257	258	**249**	258	999	273
200_20_97_9	326	332	253	256	**241**	248	816	271
200_20_97_15	356	368	347	347	**336**	336	680	345
200_20_145_15	309	314	262	271	**237**	239	480	291
200_20_150_9_D5	1177	1234	**900**	913	**900**	907	1930	
200_40_45_9	209	215	**144**	147	153	183	821	158
200_40_45_15	206	212	**164**	164	**164**	164	616	167
200_40_90_9	211	214	**148**	153	**148**	181	963	173
200_40_91_15	207	212	153	159	**136**	145	519	149
200_40_130_9_D4	642	675	**513**	513	**513**	513	1710	
200_40_133_15	214	221	163	170	**141**	151	512	182

ramente essa média. O cruzamento clássico do BRKGA utiliza um limiar e uma variável aleatória fazendo com que a chave-aleatória do filho tenha mais chance de ser igual à do pai com maior aptidão. Esses dois operadores são muito importantes para tentar fazer com que a população evolua adequadamente, aprimorando a qualidade de seus melhores indivíduos, mas também tentando impedir que o nível de diversificação deles diminua muito.

Foi criada então uma estratégia híbrida, chamada de Cruzamento Misto (cm), onde nas populações pares, o cruzamento é feito conforme o algoritmo EA2 e nas populações ímpares, conforme o BRKGA. Esse cruzamento foi aplicado tanto a um quanto ao outro. Os resultados podem ser vistos nas colunas $EA2cm$ e $BRKGAcm$ da Tabela 5.

Pelos resultados da Tabela 5, pode-se concluir que o algoritmo EA2 com Cruzamento Misto teve o melhor resultado dentre os métodos testados, aprimorando os resultados da sua versão original em diversas instâncias. O mesmo não ocorreu com o BRKGA - na verdade, a utilização do cruzamento misto piorou os resultados da versão original.

O algoritmo BRKGA só conseguiu superar os Algoritmos Evolutivos $EA2$ e $EA2cm$ em apenas 3 instâncias, o que mostra a robustez e o bom desempenho dos algoritmos evolutivos

Tabela 4: Resultados das execuções das modelagens matemáticas. Em negrito, a melhor solução obtida para cada instância. Com sublinhado, as soluções ótimas provadas dentro do tempo limite.

	CP			$CPLEX + CP$			$CPLEX + CP_{Sol}$		
Instância	Tempo(s)	Sol.	L. I.	Tempo(s)	Sol.	L. I.	Tempo(s)	Sol.	L. I.
100_5_20_9_D3		**387**	158			218,0	1616,8	**387**	387,0
100_5_22_15		**484**	393			261,0		**484**	261,0
100_5_46_15	1168,6	**528**	528			292,5		**528**	292,0
100_5_48_9		**490**	396			271,0		**490**	271,0
100_5_64_9		**474**	265			268,0		**474**	268,0
100_5_64_15		**481**	365			269,0		**481**	269,0
100_10_26_15		**233**	190			137,2		**233**	137,0
100_10_27_9_D2		**206**	166			166,0	1160,3	**206**	206,0
100_10_47_9		**252**	106			153,4		**252**	154,0
100_10_48_15		**242**	150			148,0	1656,6	**242**	242,0
100_10_64_9		**241**	178			153,0		**241**	153,0
100_10_65_15		**242**	207			152,0	1424,4	**242**	242,0
100_20_22_15		**124**	102		**124**	102,0		**124**	102,0
100_20_23_9_D1	0,7	**172**	172	55,4	**172**	172,0	17,7	**172**	172,0
100_20_46_15	2,6	**161**	161	545,0	**161**	161,0	329,6	**161**	161,0
100_20_47_9		**121**	113			113,0		**121**	113,0
100_20_65_9		**121**	108		**121**	108,0		**121**	108,0
100_20_65_15	1,1	**205**	205	427,6	**205**	205,0	160,3	**205**	205,0
200_10_50_9		**484**	298			96,0		**484**	261,7
200_10_50_15		**484**	378			261,6		**484**	261,6
200_10_84_9		**505**	120			278,4		**505**	278,4
200_10_85_15		**472**	222			259,9		**472**	259,9
200_10_128_15		**459**	391			121,0		**459**	121,0
200_10_135_9_D6	1,6	**534**	534	577,9	**534**	534,0	46,1	**534**	534,0
200_20_54_15		**256**	192			147,5		**256**	147,5
200_20_55_9		**245**	80			142,3		**245**	142,3
200_20_97_9		**239**	162			144,6		**239**	144,6
200_20_97_15	1,2	**336**	336			204,1	807,8	**336**	336,0
200_20_145_15		**234**	165			145,2		**234**	145,2
200_20_150_9_D5	0,8	**900**	900			900,0	229,6	**900**	900,0
200_40_45_9		**133**	97			97,0		**133**	97,0
200_40_45_15	3,7	**159**	159			101,0	1766,5	**159**	159,0
200_40_90_9		**130**	115			115,0		**130**	115,0
200_40_91_15	6,9	**127**	127			121,5	351,0	**127**	127,0
200_40_130_9_D4	0,4	**513**	513	667,8	**513**	513,0	582,6	**513**	513,0
200_40_133_15		**130**	127			127,0		**130**	127,0
Ótimos provados	10			5			13		

propostos. As duas últimas colunas da Tabela 5 mostram os melhores resultados da literatura e o resultado encontrado pelo método exato CP. Pode-se observar que vários resultados da literatura foram superados (sublinhados), o que indica que os métodos (exatos e heurísticos) propostos neste artigo possuem bom desempenho e potencial para encontrar soluções ótimas ou quase ótimas. A ressalva que deve ser feita para uma comparação direta é que os tempos computacionais dos métodos exatos podem ser bastante longos em instâncias de maior porte. Infelizmente não foi possível fazer uma comparação direta dos tempos de execução entre os métodos heurísticos propostos neste artigo com os da literatura, pois não há dados disponíveis para estes últimos.

6. Conclusões e Trabalhos Futuros

Este trabalho abordou o Problema de Escalonamento de Projetos com Múltiplas Habilidades (MS-RCPSP), que é uma variante do Problema de Escalonamento de Projetos com Restrição de Recursos. Nesta variante, cada tarefa demanda níveis mínimos de habilidades para serem executadas. Os recursos disponíveis também possuem níveis de familiaridade com cada habilidade e só podem ser alocados se esses valores foram suficientes para a tarefa em questão.

Tabela 5: Resultados das meta-heurísticas testadas. As médias foram feitas a partir de 20 execuções, com tempo limite de 15 segundos cada. Em negrito, os melhores resultados de cada instância.

	EA2		EA2 cm		BRKGA		BRKGA cm		Lit.	Exato
Instância	best	media	best	media	best	media	best	media		
100_5_20_9_D3	388	388,5	**387**	387,9	**387**	387,4	388	388,8	392	387
100_5_22_15	**484**	485,0	**484**	485,0	**484**	484,8	**484**	485,2	484	484
100_5_46_15	**528**	529,0	**528**	528,6	**528**	529,5	**528**	528,9	529	528
100_5_48_9	491	491,6	**490**	491,4	**490**	491,1	491	491,7	471	490
100_5_64_9	**475**	476,5	**475**	476,5	**475**	475,9	476	476,7	474	474
100_5_64_15	482	483,6	**481**	482,9	482	482,8	482	483,4	481	481
100_10_26_15	234	235,8	234	235,2	**233**	235,7	234	236,3	234	233
100_10_27_9_D2	210	212,0	210	211,3	**209**	210,7	212	213,1	215	206
100_10_47_9	254	255,4	**253**	254,6	**253**	254,7	254	255,6	255	252
100_10_48_15	245	246,5	**244**	246,1	**244**	245,9	246	247,1	244	242
100_10_64_9	245	246,6	**243**	245,5	244	245,2	245	246,6	243	241
100_10_65_15	246	247,3	**245**	246,9	246	247,6	246	247,7	244	242
100_20_22_15	129	130,1	**127**	129,4	128	129,4	129	131,1	130	124
100_20_23_9_D1	**172**	172,0	**172**	172,0	**172**	172,0	**172**	172,0	172	172
100_20_46_15	**161**	161,0	**161**	161,0	**161**	162,7	**161**	161,0	164	161
100_20_47_9	127	128,7	**126**	127,5	127	128,1	127	128,5	125	121
100_20_65_9	126	128,2	**125**	128,0	126	129,7	127	128,7	126	121
100_20_65_15	**205**	205,0	**205**	205,0	**205**	205,0	**205**	205,0	205	205
200_10_50_9	486	487,7	**485**	486,4	486	487,2	487	487,8	485	484
200_10_50_15	485	487,2	486	486,6	**484**	487,4	486	487,9	487	484
200_10_84_9	**506**	508,7	507	508,1	507	508,9	508	509,5	507	505
200_10_85_15	**475**	476,9	476	476,6	476	478,3	476	477,8	475	472
200_10_128_15	**464**	466,7	465	466,3	466	468,0	465	467,1	461	459
200_10_135_9_D6	**534**	534,0	**534**	534,0	**534**	534,0	**534**	534,0	584	534
200_20_54_15	261	262,2	**260**	261,3	**260**	262,0	261	263,0	258	256
200_20_55_9	248	249,5	**247**	248,6	248	249,3	249	251,2	249	245
200_20_97_9	243	244,8	**241**	243,7	245	246,1	244	245,7	241	239
200_20_97_15	**336**	336,0	**336**	336,0	**336**	336,0	**336**	336,0	336	336
200_20_145_15	240	241,6	**238**	240,8	**238**	243,7	**238**	241,3	237	234
200_20_150_9_D5	**900**	900,0	**900**	900,0	**900**	900,0	**900**	900,0	900	900
200_40_45_9	138	139,4	**136**	138,6	140	141,1	140	142,4	144	133
200_40_45_15	**159**	160,0	**159**	159,0	**159**	167,5	**159**	167,2	164	159
200_40_90_9	135	136,9	**134**	136,2	138	141,2	138	140,0	148	130
200_40_91_15	132	133,9	**131**	134,3	133	139,9	134	139,0	136	127
200_40_130_9_D4	**513**	513,0	**513**	513,0	**513**	513,0	**513**	513,0	513	513
200_40_133_15	**136**	138,5	**136**	138,2	140	142,2	138	139,3	141	130

Foram propostos algoritmos evolutivos e modelagens matemáticas para resolver esse problema. As modelagens incluem Programação por Restrições, Programação Inteira Mista e um método híbrido envolvendo as duas. As heurísticas incluem um Algoritmo Evolutivo (com diversos critérios para geração da população inicial) e a implementação de um BRKGA.

Os resultados mostraram que a modelagem por Programação por Restrições consegue encontrar excelentes limites primais, inclusive algumas soluções ótimas, para as instâncias testadas. O Algoritmo Evolutivo $EA2$, usando uma estratégia chamada de Cruzamento Misto, obteve os melhores resultados dentre as heurísticas testadas, superando o BRKGA. Diversas soluções da literatura foram melhoradas pelos métodos apresentados nesse artigo.

Referências

Boctor, F. (1990). Some efficient multi-heuristic procedures for resource constrained project scheduling. *E. J. of Oper. Res.*, 49:3–13.

Brucker, P., Drexl, A., Möhring, R., Neumann, K., e Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models, and methods. *E. J. of Oper. Res.*, 112:3–41.

Dai, H., Cheng, W., e Guo, P. (2018). An improved tabu search for multi-skill resource-constrained project scheduling problems under step-deterioration. *Arabian Journal for Science and Engineering*, 43:3279–3290. 10.1007/s13369-017-3047-4.

Debels, D., De Reyck, B., Leus, R., e Vanhoucke, M. (2006). A hybrid scatter searchelectromagnetism metaheuristic for project scheduling. *E. J. of Oper. Res.*, 169:638–653.

Demeulemeester, E. e Herroelen, W. (1992). A branch-and-bound procedure for multiple resource-constrained project scheduling problem. *Manag. Science*, 38:1803–1818.

Gonçalves, J. F., Mendes, J. J. M., e Resende, M. G. C. (2009). A random key based genetic algorithm for the resource constrained project scheduling problems. *Inter. J. of Prod. Res.*, 36: 92–109.

Gérault, D., Lafourcade, P., Minier, M., e Solnon, C. (2018). Revisiting aes related-key differential attacks with constraint programming. *Information Processing Letters*, 139:24–29. DOI: 10.1016/j.ipl.2018.07.001.

Hartmann, S. e Briskorn, D. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem. *E. J. of Oper. Res.*, 207(1):1–14.

Hojabri, H., Gendreau, M., Potvin, J.-Y., e Rousseau, L.-M. (2018). Large neighborhood search with constraint programming for a vehicle routing problem with synchronization constraints. *Computers & Operations Research*, 92:87–97. DOI: 10.1016/j.cor.2017.11.011.

Kreter, S., Schutt, A., Stuckey, P., e Zimmermann, J. (2018). Mixed-integer linear programming and constraint programming formulations for solving resource availability cost problems. *European Journal of Operational Research*, 266(2):472–486. DOI: 10.1016/j.ejor.2017.10.014.

Myszkowski, P. B., Skowroński, M. E., Olech, Ł. P., e Oślizło, K. (2015a). Hybrid ant colony optimization in solving multi-skill resource-constrained project scheduling problem. *Soft Computing*, 19(12):3599–3619.

Myszkowski, P. B., Łukasz P. Olech, Laszczyk, M., e Skowrónski, M. E. (2018). Hybrid differential evolution and greedy algorithm (degr) for solving multi-skill resource-constrained project scheduling problem. *Applied Soft Computing*, 62:1 – 14. ISSN 1568-4946.

Myszkowski, P. e Siemiénski, J. (2016). Grasp applied to multi-skill resource-constrained project scheduling problem. *Computational Collective Intelligence. Lecture Notes in Computer Science*, 9875:402–411.

Myszkowski, P., Skowrónski, M., e Sikora, K. (2015b). A new benchmark dataset formulti-skill resource-constrained project scheduling problem. In *Proc. of the 2015 Federated Conference onComputer Science and Information Systems, ACSIS 5*, p. 129–138.

Ochoa, L., González-Rojas, O., Cardozo, N., González, A., Chavarriaga, J., Casallas, R., e Díaz, J. (2019). Constraint programming heuristics for configuring optimal products in multi product lines. *Information Sciences*, 474:33–47. DOI: 10.1016/j.ins.2018.09.042.

Valls, V., Ballestin, F., e Quintanilla, M. (2008). A hybrid genetic algorithm for the resource constrained project scheduling problem. *European Journal of Operational Research*, 185(2):495–508.