Algoritmos Evolutivos Híbridos Aplicados no Sequenciamento de Produção em uma Indústria de Alimentos

Geraldo Pereira
Universidade Tecnológica Federal do Paraná
Av. Alberto Carazzai, 1640
geraldopereirajr@hotmail.com

Ozeas Quevedo de Carvalho
Universidade Tecnológica Federal do Paraná
Av. Alberto Carazzai, 1640
ozeasx@gmail.com

Orides Morandin Jr.
Universidade Federal de São Carlos
Rod. Washington Luís, Km 235
orides@ufscar.br

Josimar da Silva Rocha
Universidade Tecnológica Federal do Paraná
Av. Alberto Carazzai, 1640
jrocha@utfpr.edu.br

Danilo Sipoli Sanches
Universidade Tecnológica Federal do Paraná
Av. Alberto Carazzai, 1640
danilosanches@utfpr.edu.br

RESUMO

Este trabalho descreve um estudo comparativo entre seis abordagens heurísticas aplicadas no estudo de caso real na indústria de alimentos Dori, localizada na cidade de Marília, SP. Este problema é caracterizado como *Job Shop Scheduling* (JSS) e foram utilizados dois tipos de algoritmos evolutivos: Algoritmos Genéticos e Evolução Diferencial. Além disso, propõe-se o uso de técnicas de busca local e uma heurística de balanceamento de carga a fim de alcançar regiões promissoras no espaço de busca. Para a validação das abordagens propostas, foi utilizado um cenário real disponibilizado pela Dori alimentos.

PALAVRAS CHAVE. Algoritmos Evolutivos Híbridos, Evolução Diferencial, Sequenciamento de Produção

PO na Indústria, Metaheuristicas, Otimização Combinatória.

ABSTRACT

This paper describes a comparative study of six approaches for a real case at Dori company in Marília, SP. For this problem, characterized as Job-Shop Scheduling (JSS), two types of evolutionary algorithms were used: Genetic Algorithms and Differential Evolution. In addition, it is proposed the inclusion of local search techniques and load balancing heuristic in order to reach promising regions of the search space. All experiments were applied using a real scenario from Dori company.

KEYWORDS. Hybrid Evolutionary Algorithms, Differential Evolution, Job Shop Scheduling

Operational Research in Industry, Metaheuristics, Combinatorial Optimization.

1. Introdução

Nas industrias, a atividade de programação é uma das mais complexas tarefas no gerenciamento de produção. Primeiro, os programadores precisam lidar com diversos tipos diferentes de recursos simultaneamente. As máquinas terão diferentes capacidades e o pessoal, diferentes habilidades. De maneira mais importante, o número de programações possíveis cresce rapidamente à medida que o número de atividades e processos aumenta. Ou seja, para n tarefas há $n!$ (n fatorial) maneiras diferentes de programação dos trabalhos em um processo simples. Considerando mais de uma máquina ($M > 1$), o número de programações possíveis passa para $n!^M$ [Pezzella et al., 2008].

Embora pesquisadores voltados à Pesquisa Operacional e da comunidade de Inteligência Artificial estarem investigando o sequenciamento de produção por décadas, ainda há uma lacuna entre os problemas reais e aqueles de caráter acadêmico [Fayad e Petrovic, 2005].

Apesar da Evolução Diferencial e os Algoritmos Genéticos serem considerados boas meta-heurísticas para resolver o *Job Shop Scheduling* (JSS), torna-se relevante a utilização dos algoritmos híbridos, pois alguns estudos denotam a grande possibilidade de melhorar os resultados obtidos por estas meta-heurísticas através do uso de algumas técnicas de busca local [Čičková e Števo, 2010], [Yuan e Xu, 2013].

O JSS não é só mais um problema da classe NP-*hard*, mas é um dos mais difíceis dessa classificação. Uma indicação disso é dada pelo fato de que um problema $10x10$ formulado por Muth e Thompson permaneceu por mais de 40 anos sem solução [Muth e Thompson, 1963].

Em função da complexidade e do elevado número de soluções possíveis para o problema de sequenciamento (*scheduling*), é praticamente impossível modelar todas as possibilidades envolvidas no processo fazendo uso de algoritmos exatos. Nesses algoritmos, o tempo de resposta cresce de forma considerável, com grande dificuldade de obtenção de uma solução ótima em tempo satisfatório. Algoritmos exatos para otimização são computacionalmente viáveis quando tratamos problemas simples, com baixa ou média complexidade [Helsgaun, 2009].

Para problemas da classe NP-*hard*, é comum abrir mão da busca de uma solução ótima, em função dos custos de processamento e tempo, por uma solução "quase" ótima, ou boa solução, com um tempo de processamento aceitável [Morton e Pentico, 1993].

Diante da complexidade dos ambientes reais e consequente dificuldade na obtenção de ótimos resultados na aplicação de meta-heurísticas, este trabalho propõe a investigação de meta-heurísticas modificadas, substituindo o processo de mutação aleatório por algoritmos de busca local, explorando o espaço de busca sem a perda de diversidade, possibilitando uma melhor avaliação no direcionamento para regiões mais promissoras. Utilizando como *dataset* uma matriz com os tempos de produção para a relação "máquina *versus* ordem de produção" de uma indústria de alimentos da cidade de Marília, SP, aplicamos o método de pesquisa baseado em algoritmos híbridos, combinando Algortimo Genético (AG), ou *Genetic Algorithm* (GA), e Evolução Diferencial (ED), ou *Differential Evolution* (DE). Estas meta-heurísticas foram modificadas durante o processo de mutação utilizando a heurística Recozimento Simulado (RS), ou *Simulated Annealing* (SA), e as heurísticas de balanceamento de carga para busca local e inversão entre dois Genes, conhecida como 2-opt.

No capítulo 2 deste artigo, descrevemos os algoritmos utilizados no trabalho e suas respectivas adaptações. As abordagens propostas são apresentadas no capítulo 3, bem como os fluxogramas do funcionamento de cada algoritmo. No capítulo 4, descrevemos a importância e relevância do problema abordado, *Job Shop Scheduling* de um caso real, composto por um espaço de busca de $188!^{19}$ possibilidades. Detalhamos os resultados obtidos com as meta-heurísticas e suas variações no capítulo 5. Finalmente, algumas conclusões são apresentadas no capítulo 6.

2. Algoritmos Evolutivos

Neste capítulo, descreveremos as principais características dos algoritmos evolutivos utilizados neste trabalho: Algoritmos Genéticos e Evolução Diferencial. Também introduziremos as

respectivas heurísticas de busca local: RS, Inversão entre dois Genes (2-opt) e balanceamento de carga.

Introduzidos por John Holland, na década de 70 e popularizados por David Goldberg no final da década de 80, Algoritmos Genéticos são técnicas de busca e otimização inspiradas na Teoria da Evolução de Charles Darwin e fazem parte de uma área de pesquisa denominada Computação Evolucionária [De Jong, 2006].

Métodos genéticos são técnicas que otimizam soluções. Esses métodos são baseados na Evolução Natural, onde os indivíduos que conseguirem melhor adaptação ao ambiente irão sobrevier e gerar descendentes, e os demais irão desaparecer. Estes métodos utilizam uma estratégia de criação e testes de gerações, permitindo uma pesquisa paralela no espaço de soluções possíveis. Tais métodos são denominados estocásticos e não dão a garantia de solução ótima, porém na prática, eles são amplamente utilizados, gerando bons resultados em diversas aplicações [Mitchell, 1997].

De acordo com Lawrynowicz [Ławrynowicz, 2011], o Algoritmo Genético deve considerar seis etapas, sendo: 1) definição da representação do problema; 2) mecanismo de criação da população inicial, comumente obtida através de um método probabilístico, mas também podem ser utilizados métodos determinísticos; 3) definição e aplicação da função de avaliação do *fitness*, ou seja, determinar o valor de cada indivíduo; 4) processo de seleção dos pais que irão gerar novos indivíduos para as populações seguintes; 5) heurísticas de cruzamento e 6) mutação para pesquisar melhores indivíduos. Também é necessária a adequada calibração dos parâmetros, tais como taxas de mutação, número de interações, tamanho da população etc.

Assim como os Algoritmos Genéticos, a Evolução Diferencial pertence à classe dos Algoritmos Evolucionários. Todavia, existem diferenças fundamentais entre os Algoritmos Genéticos e a Evolução Diferencial. Na Evolução Diferencial um indivíduo é criado a partir de quatro pais e sofre dupla mutação, enquanto no algoritmo genético tradicional um indivíduo é criado a partir de dois pais e sofre mutação simples [Čičková e Števo, 2010].

A Evolução Diferencial, *Differential Evolution* (DE), é uma heurística para minimização de funções não-lineares e não-diferenciáveis no espaço contínuo. Os testes com a Evolução Diferencial em diversas instâncias comprovaram que esta heurística obteve resultados superiores comparados à outras heurísticas [Storn e Price, 1997].

Segundo Storn e Price [Storn e Price, 1997], o funcionamento do DE também exige a geração de uma população inicial de forma aleatória, composta de N_p indivíduos, denominados vetores, abrangendo todo o espaço de busca. Em geral, esta população é gerada por uma distribuição de probabilidade uniforme, caso não haja nenhum conhecimento sobre o problema. A população seguirá a evolução natural, mantendo o mesmo número de indivíduos para todas as gerações.

Após a geração da população inicial, são necessários mais dois parâmetros iniciais, sendo: o fator escalar F da equação de mutação diferencial, número real positivo pertencente ao intervalo $[0, 2]$, que irá controlar a amplitude da diferença vetorial. De forma indireta, essa amplitude irá controlar a taxa em que a população corrente evolui e a sua distribuição no espaço de busca; o segundo parâmetro é o escalar C_r, taxa de cruzamento de um indivíduo da população durante o *crossover*.

A operação de mutação para o DE acontece através da geração de novos indivíduos, denominados vetores modificados, que são obtidos através de um processo envolvendo três outros indivíduos selecionados aleatoriamente, conforme a Equação 1.

$$v_i^{G+1} = X_a^G + F(X_\beta^G - X_\gamma^G) \tag{1}$$

Para obter o vetor modicado v_i^{G+1}, devemos considerar os vetores X_a^G, X_β^G e X_γ^G, distintos entre si, selecionados em uma população com N_p indivíduos de forma aleatória. Os índices aleatórios α, β e $\gamma \in \{1, ..., N_p\}$ e são inteiros e distintos entre si. Para o par de vetores X_β^G e X_γ^G da G-ésima geração, obtemos o vetor diferença $X_\beta^G - X_\gamma^G$. Após isso, multiplicamos esta

diferença pelo escalar F, sendo este resultado a diferença ponderada, que por sua vez será usada para perturbar o vetor X_a^G. Na Figura 1, é possível observar um exemplo bidimensional composto pelos diferentes vetores que geram o vetor modificado v_i^{G+1}.

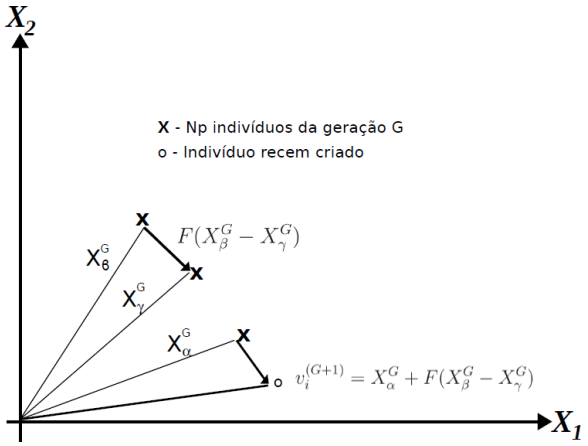

Figura 1: Processo de geração do vetor modificado da função bidimensional

Após o processo de mutação, é executado o cruzamento, onde um indivíduo é escolhido aleatoriamente para que os seus componentes sejam misturados com os componentes do vetor modificado, gerando assim o vetor experimental U^{G+1}. Portanto, utilizando o vetor alvo e o vetor modificado, os componentes do vetor experimental U^{G+1} são selecionados de acordo com a Equação 2.

$$U^{G+1} = \begin{cases} v^{G+1}(i) & \text{se } rand_i \leq C_r \\ X_a^G(i) & \text{se } rand_i \geq C_r, i = 1, ..., n \end{cases} \quad (2)$$

Na Equação 2, $v^{G+1}(i)$ é o i-ésimo componente do vetor modificado v_i^{G+1}; $X_a^G(i)$ é o i-ésimo componente do vetor alvo X_a^G; $rand_i$ é um número gerado aleatoriamente no intervalo $[0, 1]$; e C_r representa a probabilidade do vetor experimental herdar os valores das variáveis do vetor modificado.

No processo de seleção, caso o custo do vetor experimental for menor que o custo do vetor alvo, o vetor experimental passa a ser o vetor alvo, para ser utilizado nas próximas gerações. O procedimento poderá ser encerrado através da convergência, ou mesmo por um número máximo de avaliações.

O DE, comparado com outros algoritmos evolucionários, tem obtido melhores resultados, apesar da sua simplicidade e facilidade de implementação. Este bom desempenho está relacionado com a sua habilidade na diversificação dos indivíduos e na execução de buscas locais com qualidade. Em função disto, o DE tem despertado na comunidade acadêmica e industrial um grande interesse nas últimas décadas.

2.1. Algoritmos de Busca Local

Neste capítulo, descreveremos os algoritmos de busca local utilizados neste trabalho para substituir o processo de mutação aleatória, originalmente utilizado pelas meta-heurísticas.

O primeiro algoritmo é o *Simulated Annealing* (SA), que é um método inspirado no processo de tratamento de metais na indústria metalúrgica, no qual um sólido é aquecido lentamente e esfriado sob uma condição natural. Também é uma técnica probabilística que tem sido aplicada em grandes áreas, como Engenharia de Controle, Aprendizado de Máquina e Processamento de Imagens. Este tipo de abordagem é reconhecida como uma abordagem prática para a solução de muitos problemas complexos, incluindo casos reais de diferentes naturezas. O SA também fornece

um conjunto de soluções que se aproximam do ótimo, ao invés de uma solução ótima que pode ser alcançada pelos métodos clássicos de otimização determinística existentes, o que para casos mais complexos, pode se tornar impeditivo pelo seu custo de processamento [TAVAKOLI et al., 2016].

A probabilidade de aceitação de uma solução é dada pela Equação 3.

$$P(S_t \rightarrow S_{t+1}) = \begin{cases} e^{\left(-\frac{\Delta E(S_t)}{T(t)}\right)} & \Delta E(S_t) \geq 0 \\ 1 & \Delta E(S_t) \leq 0 \end{cases} \quad (3)$$

O segundo algoritmo é o k-opt, também considerado um algoritmo de busca local. Seu processo inicia uma pesquisa no espaço amostral e faz movimentos de um determinado local do indivíduo para locais em sua vizinhança. Seu mecanismo é baseado em movimentos ou trocas entre genes que poderão converter uma solução candidata em outra. Diante de um deslocamento viável, o algoritmo executa repetidamente intercâmbios visando melhorar o *fitness* do indivíduo a cada movimento. Estes movimentos podem percorrer todo o indivíduo, buscando a evolução, ou executar movimentos pré-determinados. Todavia, a solução candidata somente é substituída caso o *fitness* seja melhorado. O k representa a quantidade de movimentos que podem ser feitos de uma única vez, como por exemplo, 2-opt, 3-opt etc. [Helsgaun, 2009]

Finalmente, o algoritmo de balanceamento de carga, ou *Load Balance* (LB), é um algoritmo utilizado para melhorar o sequenciamento de produção, que consiste na adequada distribuição de trabalhos para cada máquina, visando cumprir os objetivos de produção. Em função das mais diversas aplicações no mundo real, os pesquisadores têm estudado o problema do sequenciamento de produção paralelo. Encontrar o equilíbrio na distribuição dos trabalhos é fundamental para maximizar o uso dos recursos. Diante deste cenário, o balanceamento de carga vem sendo utilizado com sucesso nas aplicações para reduzir o tempo total de trabalho. A máquina com o tempo máximo de trabalho é denominada gargalo, que deverá ser tratada, ou seja, redistribuída, para evitar o estrangulamento do sistema. Assim, o balanceamento de carga é utilizado para remover o gargalo existente, buscando atingir o desempenho global ótimo [Rajakumar et al., 2007].

Na Figura 2, podemos observar a redução do *makespan* de 103 horas para 81 horas, movimentando o *job* 100 para o cenário 2 (quadro à direita).

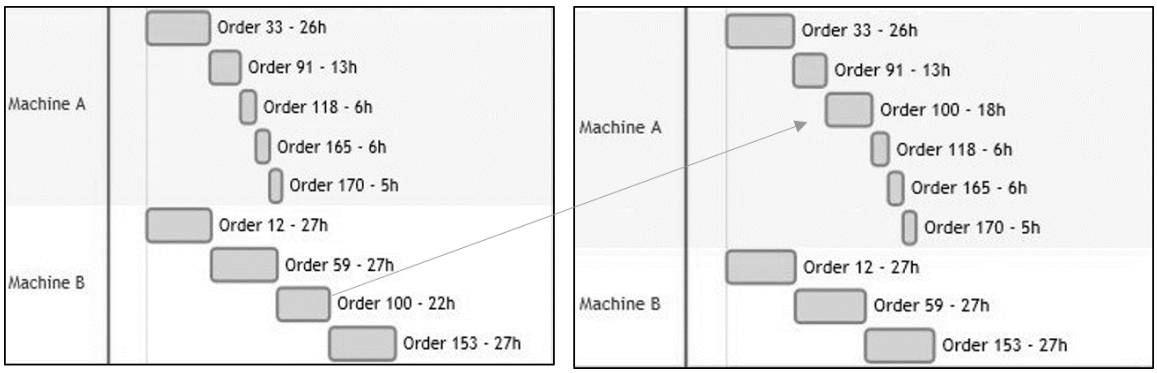

Figura 2: Execução do balanceamento de carga entre máquinas

3. Abordagens Propostas

Em função da dificuldade de obtenção de resultados para problemas de um ambiente real complexo, esta abordagem tem como objetivo a minimização do *makespan* do sequenciamento de produção. Foram desenvolvidos dois algoritmos utilizando a linguagem de programação R e o ambiente de programação RStudio. O primeiro algoritmo foi escrito para executar as etapas do Algoritmo Genético tradicional e o segundo algoritmo representa o funcionamento da Evolução Diferencial, ambos descritos no capítulo 2. Para os dois algoritmos, foram criadas duas derivações:

Modificação na mutação, utilizando a heurística 2-opt como busca local; e modificação na mutação, utilizando a heurística *Simulated Annealing* conjuntamente com o balanceamento de carga, também descritas no capítulo 2 deste trabalho.

Os fluxogramas dessas abordagens estão ilustrados nas figuras 3, 4, 5 e 6.

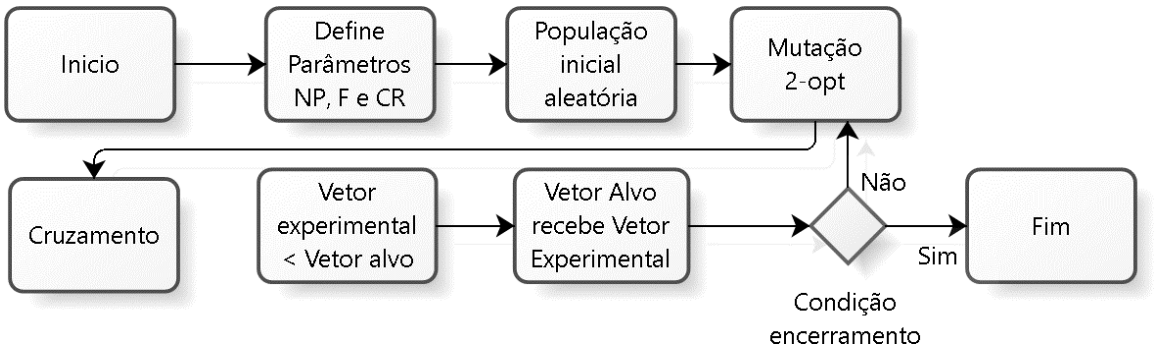

Figura 3: Evolução Diferencial com Mutação 2-opt

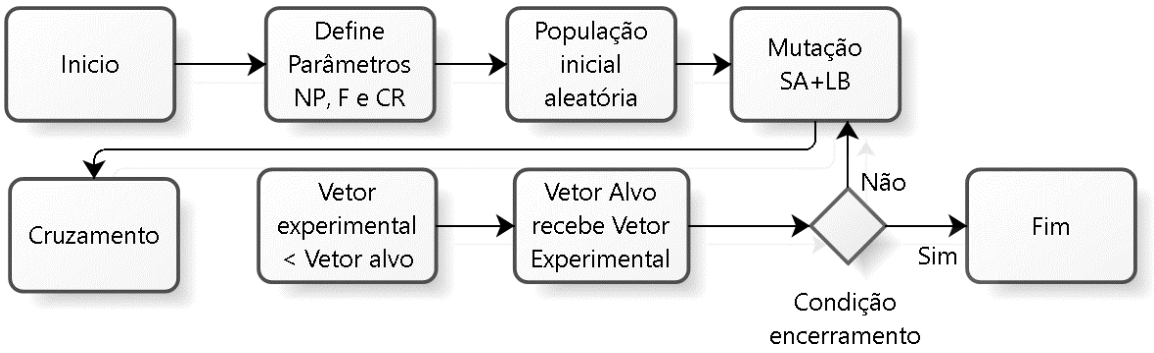

Figura 4: Evolução Diferencial com Mutação SA + LB

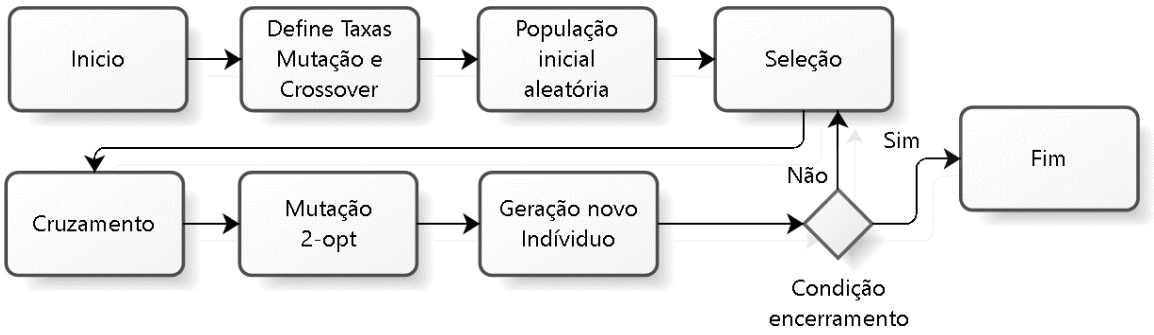

Figura 5: Algoritmo Genético com Mutação 2-opt

4. Definição do Problema

Neste capítulo, contextualizaremos as características do ambiente onde as técnicas foram aplicadas, no caso, uma indústria de alimentos produtora de guloseimas. Também serão apresentadas as variáveis consideradas para a construção do *dataset* utilizado, a configuração do cromossomo e seus respectivos genes, bem como a função de avaliação *fitness*.

4.1. Ambiente

Para o estudo e aplicação das meta-heurísticas e heurísticas de busca local abordadas neste artigo, utilizamos como estudo de caso uma indústria do ramo de alimentos, mais especificamente,

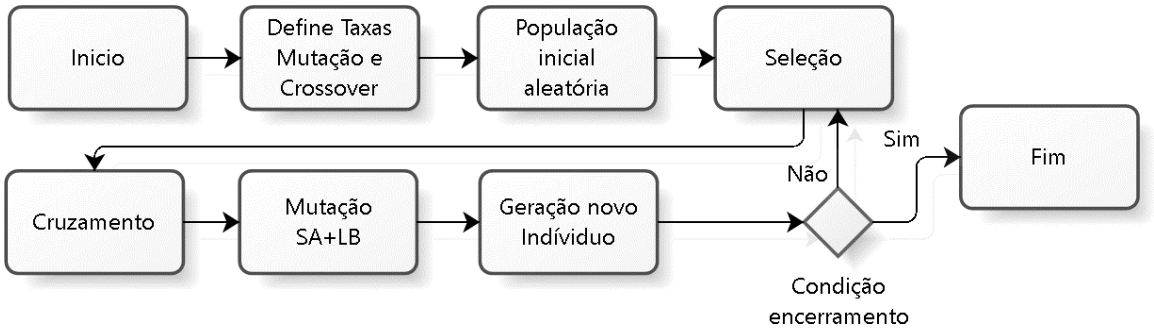

Figura 6: Algoritmo Genético com Mutação SA + LB

Candies & Snacks, sendo a maior fabricante brasileira de balas e confeitos, contando com mais de 2.000 funcionários, 3 unidades fabris e 4 centros de distribuição, exportando para mais de 60 países.

O escopo do trabalho limita-se a um dos setores de empacotamento de uma das três unidades fabris, porém com aplicabilidade para qualquer outro setor da mesma unidade ou das demais unidades. Nesta unidade, são produzidos amendoins de diferentes formatos e sabores, entre doces e salgados, compondo aproximadamente 50 *Stock Key Units* (SKUs). Nas linhas de amendoins salgados, além do amendoim japonês tradicional, também são produzidos amendoins nos sabores pimenta e "cebola & salsa". Já nas linhas de amendoins doces, são produzidos os amendoins sabor chocolate, chocolate branco e amendoim confeitado colorido. Para estes sabores, existem pacotes para os segmentos de atacado e varejo. Para o segmento de atacado, são produzidos os pacotes de 500g, 700g, 1.010g e 5kg. Já para o segmento do varejo, são consideradas as cartelas de 70g, pacotes de 90g, 100g, 150g e 200g.

Este setor está configurado para trabalhar com até 19 máquinas de empacotamento, e também é responsável pela execução de aproximadamente 200 ordens de produção por mês. A operação atua com dois turnos para a maioria das máquinas, mas havendo gargalos, o terceiro turno é ativado para cumprir o planejamento de produção. São consideradas 14,5 horas úteis para dois turnos e 21 horas no caso de 3 turnos. Recentemente, foi realizada a cronoanálise em todas as máquinas visando melhorar a acuracidade do planejamento e a consequente entrega de produtos. Do número total obtido na cronoanálise, a organização considera apenas 80% desta capacidade, pois os 20% restantes são considerados para limpeza, manutenção, troca de produtos e sabores, de acordo com estudos internos realizados pela empresa. Diante deste cenário, considerando um mês de produção a partir do volume proposto pela área de planejamento da empresa, este volume foi dividido igualmente durante as quatro semanas do mês. Em função do processo de atendimento ao cliente ser pelo método *make-to-stock*, não houve a necessidade de impor restrições para o sequenciamento das ordens durante o mês. Considerando este cenário, foi criado um *dataset* de 188 linhas (*jobs*) e 19 colunas (máquinas) considerando as combinações de cada *job* para cada máquina. Como alguns produtos não podem ser empacotados por determinadas maquinas (Tabela 1), no *dataset* foi utilizada a técnica de penalização, atribuindo para esta combinação um custo elevado, para que os algoritmos possam identificá-los e evitá-los durante o processo de geração dos indivíduos, seja na geração da população inicial como nas etapas de cruzamento e mutação.

4.2. Representação Cromossomial

Tanto para o algoritmo genético, quanto para a evolução diferencial, a representação cromossomial é fundamental para o bom funcionamento dos algoritmos [Abdelmaguid, 2010]. Neste artigo, por se tratar de um problema combinatório, utilizamos a representação por números inteiros. Para todos os experimentos, nossa população, definida de maneira empírica, é composta por 100 indivíduos. Cada indivíduo, ou cromossomo, é composto por 188 genes que representam as 188 ordens de produção, ou *jobs* do mês. Cada gene poderá conter um número inteiro entre 1 e 19, que representa uma das 19 máquinas possíveis que poderão receber uma das 188 ordens de

Tabela 1: Possibilidades de produção Máquina *versus* Produto

	Máquina																		
Produto	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
70g		☺	☺	☺							☺	☺		☺		☺	☺	☺	
90g							☺	☺								☺			
100g				☺					☺	☺			☺		☺	☺	☺		
150g	☺						☺	☺					☺	☺			☺		
200g				☺			☺	☺						☺		☺			
500g				☺	☺	☺	☺	☺						☺		☺			
700g					☺	☺										☺			
1.010g						☺	☺												
5kg																			☺

produção. Existem diversas máquinas que não poderão ser atribuídas a determinados *jobs* pela limitação técnica daquela máquina em produzir determinado produto (Tabela 1). Para isso, utilizamos a penalização, atribuindo um custo elevado para estas situações, para que os algoritmos saiam rapidamente deste local do espaço amostral. Portanto, durante os processos de geração da população inicial e demais populações, bem como durante o processo de cruzamento e mutação, tomou-se o cuidado de não efetuar a atribuição de uma máquina a um *job* com elevado custo. A Figura 7 representa um exemplo aleatório de uma possível solução de cromossomo contendo 188 genes que representam o problema estudado neste artigo.

6	6	4	4	16	11	5	5	16	4	14	2	5	14	19	4	17	8
4	3	18	8	7	7	7	19	9	8	7	13	7	7	1	8	18	7
7	8	19	7	19	8	8	7	19	19	8	6	6	16	16	4	12	16
5	16	4	14	2	5	5	19	4	17	14	14	12	13	8	16	7	8
19	9	7	8	19	7	7	7	8	13	7	8	7	19	7	13	7	7
1	19	13	8	5	6	16	4	14	2	16	6	16	16	16	3	16	6
19	17	17	8	4	12	18	8	7	1	8	19	9	8	7	19	8	7
8	16	18	8	8	8	19	7	18	8	8	7	19	19	7	5	16	16
14	4	3	16	5	4	14	10	2	6	14	19	15	17	14	9	12	13
8	8	1	8	19	9	7	1	19	7	8	8	16	19	8	7	8	19
8	18	7	8	8	19	13	7										

Figura 7: Possível Solução (Indivíduo ou Cromossomo)

4.3. Função Fitness

Em geral, a aptidão do indivíduo é calculada através da função objetivo, que está diretamente relacionada a cada problema. Neste artigo, cada indivíduo, ou cromossomo com os seus 188 genes, é a entrada para uma função de análise de desempenho que irá calcular o *fitness* de cada indivíduo, possibilitando assim a comparação da evolução dos diferentes indivíduos para as diferentes gerações.

O *makespan* é calculado a partir da Equação 4, onde primeiramente criamos um vetor contendo as máquinas e suas respectivas alocações.

$$Cost_m = \sum_{i=1}^{j} C(j, m) \tag{4}$$

Na Equação 4, m representa o número da máquina, que para o nosso *dataset* terá um intervalo entre 1 e 19, ou seja, 19 máquinas possíveis para cada *job*. O j representa a quantidade de

jobs contidos em cada cromossomo, neste caso, 188. C é a matriz que possui o custo em horas da execução de um determinado *job* (j) em uma determinada máquina (m).

Após calcular o custo para cada máquina, o *fitness* é obtido a partir do caminho critico (max), ou seja, a máquina que contiver o maior número de horas alocadas será o caminho critico e, consequentemente, representará o *makespan*, conforme a Equação 5.

$$f(x) = max(Cost_m) \tag{5}$$

Na Equação 5, x é o cromossomo a ser calculado e $Cost_m$ é o vetor que contem a duração de tempo para execução de um conjunto de *jobs* em cada máquina.

5. Resultados obtidos

Neste capitulo, apresentaremos os resultados obtidos a partir da abordagem proposta e seus respectivos algoritmos aplicados ao *dataset* obtido a partir do ambiente real de produção. A Tabela 2 contém os parâmetros utilizados para a aplicação executar 188 *jobs* (ordens de produção) para as 19 máquinas disponíveis, considerando as respectivas restrições. Após executar cada um dos 6 cenários por 30 vezes, obtivemos os resultados apresentados na Tabela 3.

Tabela 2: Parâmetros de Execução

Parâmetro	**Valor**
Tamanho da população	100 Indivíduos
Taxa *Crossover* / Mutação	80% / 3%
Fator de Ponderação Diferencial	0,8
Critério de Parada	Estagnação em 100 operações
Seleção	Torneio
Jobs/Máquinas	188/19
Espaço de busca	$188!^{19}$
Execuções	30
Taxa SA / 2-opt	20%

Tabela 3: Resumo dos resultados obtidos

Meta-heurística	Busca Local	Média	Desvio Padrão	Tempo Médio (Minutos)
GA	N/A	303,27	9,76	10
DE	N/A	320,95	8,83	5
GA	2-opt	298,85	9,4	12
DE	2-opt	318,88	7,48	60
GA	SA+LB	283,27	4,55	28
DE	SA+LB	278,49	1,24	128

A Tabela 3 reúne as configurações e resultados obtidos para as duas meta-heurísticas e suas respectivas derivações. Na coluna "Meta-heurística", temos GA e DE. Na coluna "Busca Local", identificamos a heurística utilizada para a busca local, podendo ser: "N/A", no caso em que foi utilizado o algoritmo tradicional sem modificação; 2-opt, que faz a troca entre dois genes do cromossomo até que se obtenha otimização do *fitness*, no entanto, se não há esta otimização ao final do cromossomo, sua configuração original é mantida; SA é aplicado durante o processo de mutação juntamente com o balanceamento de carga, também como heurística de busca local. As colunas "Média" e "Desvio Padrão" são calculadas a partir dos resultados dos melhores *fitness* identificados durante as 30 execuções de cada versão dos algoritmos. Por fim, a coluna "Tempo Médio" contém

Tabela 4: Comparação entre as abordagens utilizando o test-t para múltiplas comparações.

#	DE	DE-2opt	GA	GA-2opt	GA-SA+LB	DE-SA+LB
DE-SA+LB	+	+	+	+	+	#
GA-SA+LB	+	+	+	+	#	#
GA-2opt	+	+	+	#	#	#
GA	+	+	#	#	#	#
DE-2opt	=	#	#	#	#	#
DE	#	#	#	#	#	#

o custo computacional em minutos para um computador com 4GB de memória RAM e processador Intel Core i5 de 2,20GHz.

Além disso, a Figura ilustra em forma de gráficos de caixa os resultados obtidos para cada abordagem avaliada. Os resultados apresentados foram baseados nos valores obtidos a partir da execução de 30 experimentos.

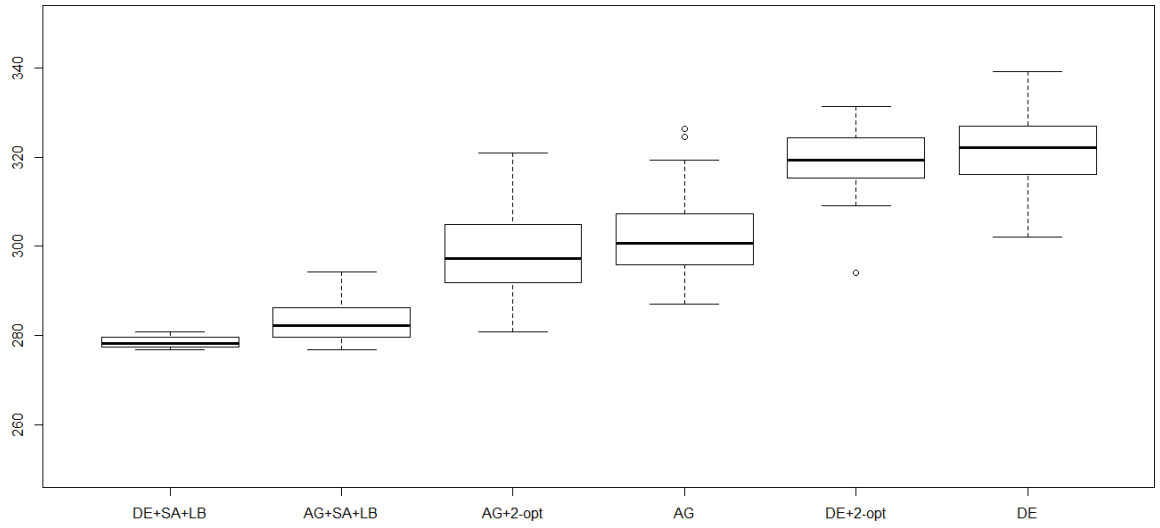

Figura 8: Resultados obtidos a partir de 30 experimentos.

Aplicou-se, inicialmente, o teste de normalidade, indicando que os resultados pertencem a uma distribuição normal. Diante disto, foi realizado um teste t para múltiplas comparações, considerando um valor p de 0,005. As diferenças encontradas entre as abordagens são sintetizadas na Tabela 4, de acordo com os resultados ilustrados no gráfico da Figura 8.

Ainda na Tabela 4, o símbolo $+$ indica que há diferença estatística entre os grupos enquanto que o símbolo $=$ indica que não há diferenças entre os grupos comparados. Além disso, podemos observar que a abordagem evolução diferencial com as técnicas *simulated annealing* e *load balance* apresentou um desempenho superior em relação as demais abordagens em relação ao valor do *makespan* obtido, uma vez que este método possue diferença estatíticas entre todas as outras abordagens comparadas.

6. Conclusão

Neste trabalho, algoritmos evolutivos híbridos foram propostos para um JSS real da indústria de alimentos. Diante disto, foram propostas a utilização de dois algoritmos híbridos: Algoritmos Genéticos e Evolução Diferencial com algoritmos de busca local. Neste sentido, foram utilizados os algoritmos 2-opt, *Simulated Annealing* e *Load Balance*. Para validação dos experimentos, foram utilizados a programação da produção de um específico mês diponível em planilhas eletrônicas da

Dori Alimentos. A partir destes dados foi elaborado o *dataset* utilizado nos experientos. Os programadores de produção conseguiram executar a programação normal deste dataset (sem o uso das abordagens propostas) em 423 horas as ordens de produção nas respectivas máquinas, já considerando as restrições de máquinas *versus* produtos. Com base nos resultados obtidos, a abordagem Evolução Diferencial com *Simulated Annealing* e *Load Balance* apresentou um melhor desempenho em relação as demais abordagens, atingindo um *makespan* médio de 278 horas, reduzindo em 34% o tempo total de 423 horas calculado incialmente pelos programadores da produção sem o uso de técnicas baseadas em computação evolutiva. Além disso, para o sequenciamento de produção através da abordagem proposta, houve uma redução significativa neste tempo, gerando ganhos de produtividade e otimização de mão de obra. Adicionalmente, devemos considerar que, ao evitar planilhas eletrônicas, diminuímos o risco de erro, dado o volume de dados manuseado, bem como o tempo despendido, principalmente em comparação ao custo computacional dos algoritmos. Por fim, a configuração sugerida pelo algoritmo foi validada pela equipe de planejamento e se mostrou factível.

7. Agradecimentos

Os autores gostariam de agradecer à Universidade Tecnológical Federal do Paraná (UTFPR) e CNPq (processo 458598/2014-3) pelo suporte financeiro concedido para esta pesquisa. Os autores gostariam de agradecer também à Dori Alimentos pela disponibilização dos dados utilizados nete trabalho.

Referências

Abdelmaguid, T. F. (2010). Representations in Genetic Algorithm for the Job Shop Scheduling Problem: A Computational Study. *J. Softw. Eng. Appl.*, 03(12):1155–1162. ISSN 1945-3116. URL http://www.scirp.org/journal/PaperDownload.aspx?DOI=10.4236/jsea.2010.312135.

Čičková, Z. e Števo, S. (2010). Flow Shop Scheduling using Differential Evolution. *Manag. Inf. Syst.*, 5(2):8–13. URL http://www.ef.uns.ac.rs/mis/archive-pdf/2010-No2/2010{_}2{_}2.pdf.

De Jong, K. (2006). *Evolutionary Computation: A Unified Approach*. MIT Press. ISBN 9780262041942. URL https://books.google.com.br/books?id=OIRQAAAAMAAJ.

Fayad, C. e Petrovic, S. (2005). A Fuzzy Genetic Algorithm for Real-World Job Shop Scheduling. p. 524–533. URL http://link.springer.com/10.1007/11504894{_}71.

Helsgaun, K. (2009). General k-opt submoves for the Lin-Kernighan TSP heuristic. *Math. Program. Comput.*, 1(2-3):119–163. ISSN 1867-2949. URL http://link.springer.com/10.1007/s12532-009-0004-6.

Ławrynowicz, A. (2011). Genetic Algorithms for Solving Scheduling Problems in Manufacturing Systems. *Found. Manag.*, 3(2). ISSN 2080-7279. URL http://www.degruyter.com/view/j/fman.2011.3.issue-2/v10238-012-0039-2/v10238-012-0039-2.xml.

Mitchell, T. M. (1997). Machine Learning. p. 1–414. URL http://www.cs.cmu.edu/{~}tom/mlbook.html.

Morton, T. E. e Pentico, D. W. (1993). *Heuristic Scheduling Systems - With applications to production systems and project management*. Wiley. ISBN 0471578193. URL http://library.wur.nl/WebQuery/clc/356698.

Muth, J. F. e Thompson, G. L. (1963). *Industrial scheduling,*. Prentice-Hall, Englewood Cliffs N.J. URL http://www.worldcat.org/title/industrial-scheduling/oclc/230145.

Pezzella, F., Morganti, G., e Ciaschetti, G. (2008). A genetic algorithm for the Flexible Job-shop Scheduling Problem. *Comput. Oper. Res.*, 35(10):3202–3212. ISSN 03050548. URL http://linkinghub.elsevier.com/retrieve/pii/S0305054807000524.

Rajakumar, S., Arunachalam, V. P., e Selladurai, V. (2007). Workflow balancing in parallel machines through genetic algorithm. *Int. J. Adv. Manuf. Technol.*, 33(11-12):1212–1221. ISSN 0268-3768. URL http://www.emeraldinsight.com/doi/10.1108/17410380610642296http://link.springer.com/10.1007/s00170-006-0553-z.

Storn, R. e Price, K. (1997). No Title. *J. Glob. Optim.*, 11(4):341–359. ISSN 09255001. URL http://link.springer.com/10.1023/A:1008202821328.

TAVAKOLI, R., SADJEDI, H., e POURMIR FIROOZABADI, S. M. (2016). An application of simulated annealing to optimal transcranial direct current stimulation of the human brain. *TURKISH J. Electr. Eng. Comput. Sci.*, 24:1135–1149. ISSN 13000632. URL http://online.journals.tubitak.gov.tr/openDoiPdf.htm?mKodu=elk-1305-134.

Yuan, Y. e Xu, H. (2013). Flexible job shop scheduling using hybrid differential evolution algorithms. *Comput. Ind. Eng.*, 65(2):246–260. ISSN 03608352. URL http://linkinghub.elsevier.com/retrieve/pii/S0360835213000739.