GEOVANE APARECIDO RIBEIRO

Orientador: Marco Antonio Moreira de Carvalho

UM MÉTODO EVOLUTIVO APLICADO AO EQUILÍBRIO DO FLUXO DE LINHAS DE PRODUÇÃO AUTOMOTIVAS

Ouro Preto

Agosto de 2017

Universidade Federal de Ouro Preto
Instituto de Ciências Exatas
Bacharelado em Ciência da Computação

UM MÉTODO EVOLUTIVO APLICADO AO EQUILÍBRIO DO FLUXO DE LINHAS DE PRODUÇÃO AUTOMOTIVAS

Monografia apresentada ao Curso de Bacharelado em Ciência da Computação da Universidade Federal de Ouro Preto como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

GEOVANE APARECIDO RIBEIRO

Ouro Preto
Agosto de 2017

UNIVERSIDADE FEDERAL DE OURO PRETO

FOLHA DE APROVAÇÃO

Um Método Evolutivo Aplicado ao Equilíbrio do Fluxo de Linhas de Produção Automotivas

GEOVANE APARECIDO RIBEIRO

Monografia defendida e aprovada pela banca examinadora constituída por:

Dr. Marco Antonio Moreira de Carvalho – Orientador
Universidade Federal de Ouro Preto

Dr. Dayanne Gouveia Coelho
Universidade Federal de Ouro Preto

Dr. Reinaldo Silva Fortes
Universidade Federal de Ouro Preto

Ouro Preto, Agosto de 2017

Resumo

Este trabalho tem como objetivo tratar o Problema de Sequenciamento de Carros – *Car Sequencing Problem* (CSP), caracterizado como um problema NP-Difícil. O CSP surge no contexto da produção de carros em uma indústria automobilística. Os carros a serem produzidos são sequenciados em uma linha de produção que está em movimento e possui uma ou mais estações de trabalho capazes de instalar um único item opcional em cada carro por vez.

Ao terminar uma operação, esta estação deve se deslocar na linha de produção a fim de começar a instalação de um item opcional em um outro carro posterior na linha de produção. Portanto, deve existir um intervalo entre os carros que exigem o mesmo item opcional para que as estações de trabalho tenham a capacidade de atendê-los, dando origem às restrições de capacidade que devem ser respeitadas.

Estas restrições indicam que em uma determinada sequência, não deve haver mais carros requerendo um mesmo item opcional tal que não seja possível atendê-los com as estações de trabalho disponíveis. Portanto, os carros devem ser organizados de forma que nenhuma restrição de capacidade seja violada, ou seja, de maneira que seja possível atender todas as exigências por itens opcionais sem haver sobrecarga das estações de trabalho.

É apresentada neste trabalho a aplicação de uma metaheurística evolutiva, o Algoritmo Genético de Chaves Aleatórias Viciadas para a solução do CSP, e também métodos de busca local e operadores adaptativos como meios de refinamento para a metaheurística proposta. A motivação para aplicação desta metaheurística se dá pelo fato de nunca ter sido aplicada para a solução do problema. Os experimentos computacionais consideraram 109 instâncias disponíveis na literatura. O tempo de execução foi aceitável para todas as instâncias e o método proposto foi capaz de gerar soluções ótimas em 55 instâncias.

Abstract

This work addresses the Car Sequencing Problem (CSP), a combinatorial optimization problem characterized as NP-Hard. The CSP arises in the context of cars production in a auto industry. The cars to be produced are sequenced on a moving production line that has one or more workstations able to install a single optional item in one car at a time.

After an operation is finished, each workstation must move in the production line in order to start a new installation of an optional item in another car in the production line. Therefore, there should be a slack between the cars that require the same option so that workstations have capacity to serve it, avoiding an overload. During the car sequencing on a production line, capacity constraints arise and must be respected.

These constraints indicate that in a given sequence, there should be no more cars requiring the same option than the capacity of the available workstations. Therefore, this ration is planned in a way that is possible to meet all optional requirements without overloading the workstations and making the production line infeasible.

This work employs an evolutionary metaheuristic, the Biased Random Key Genetic Algorithm (BRKGA) as well as local search methods and adaptive operators as means of refinement for the proposed metaheuristic. The motivation arises from the fact that it has never been applied to the problem. The computational experiments considered 109 benchmark instances available in the literature. The running time was acceptable for all instances and the proposed method was able to generate optimal solutions for 55 instances. conforma a Figura **??**

Dedico este trabalho aos meus pais Geraldo e Aparecida pela educação de berço, aos meus irmãos e familiares que me influenciaram direta e indiretamente para que eu chegasse até aqui.

Agradecimentos

Agradeço primeiramente a Deus pelo dom da vida;

À Universidade Federal de Ouro Preto (UFOP) que me recebeu como "Mãe";

Ao meu orientador Marco Antonio Moreira de Carvalho;

Agradeço a Ouro Preto pelos quatro anos de amadurecimento;

Aos meus amigos de perto: Carolina, Estêvão, Gustavo, Júlia, João Pedro, dentre outros;

Aos amigos de longe: Arlete, Vera, Inês, Dulcineia, Helena, às Rosângelas, dentre outros;

Agradeço pelo acolhimento da minha família adotiva: minhas irmãs Zilda, Margarete, Elisete, Dora e minha "mãe" Marina;

E a todos que oraram e acreditaram em mim.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

As demandas por bens de consumo tendem a aumentar ao longo do tempo, ocasionando uma ampla concorrência entre as indústrias responsáveis por atender tais demandas. Como exemplo, pode-se citar as indústrias automotivas, responsáveis pelo desenvolvimento e fabricação de veículos automotores. De acordo com o Anuário da Indústria Automobilística Brasileira (2016), no início do século XX, algumas empresas iniciaram em São Paulo a montagem de carros, estimulando o crescimento do mercado através da exposição de novos modelos.

O processo de importação que havia começado no fim do século XX foi interrompido devido à Segunda Guerra Mundial, surgindo então, as primeiras fábricas de peças com a finalidade de suprir as demandas da frota existente. No início da década de 50, Getúlio Vargas proibiu a importação de carros, originando as primeiras fábricas da *Volkswagen* e *Mercedes-Benz* em São Paulo.

Em 1956, fundou-se a Associação Nacional dos Fabricantes de Veículos Automotores (ANFAVEA), uma entidade que passou a representar os fabricantes de carros junto à sociedade. A indústria automotiva passou a ser vista como símbolo de progresso na economia brasileira. Segundo as informações mais recentes contidas no Anuário da Indústria Automobilística Brasileira (2016), existem atualmente 31 fabricantes de automóveis no Brasil, representando um total de 61 plantas industriais em 11 estados espalhadas em mais de 50 municípios.

Em 2014, o faturamento foi em torno de 95 bilhões de dólares e sua participação no PIB foi de 20,4% no processo industrial. A média de empregos gerados por essas empresas é aproximadamente 1,5 milhão. Encontram-se instalados no Brasil, os maiores fabricantes mundiais, como *Toyota*, *Ford*, *Chevrolet*, *Volkswagen*, *Fiat*, *Mitsubishi*, *Peugeot*, *Citröen*, *Mercedes-Benz* e *Renault*, dentre outros.

Uma indústria automotiva é composta por uma ou mais linhas de montagem nas quais vários carros passam por estações de trabalho móveis ao decorrer do processo de produção. Cada estação de trabalho possui uma função específica. Estes carros não necessariamente são iguais, embora possam ser do mesmo modelo, cada carro pertence a uma classe que requer um conjunto de itens opcionais distintos.

Dessa forma, um carro pode demandar teto solar enquanto outro requer som de fábrica, um terceiro ar condicionado e freios ABS e um quarto pode exigir bancos de couro, alarme, *airbag* e assim por diante. O planejamento de produção de uma indústria automotiva é explicado da seguinte forma: os carros a serem produzidos são sequenciados em uma linha de produção que está em movimento, cada estação de trabalho é responsável por instalar um item opcional em cada carro por vez.

Ao terminar a instalação de um item opcional, esta estação deve se deslocar na linha de produção a fim de começar a instalação daquele item opcional em um outro carro posterior na linha de produção. Portanto, deve existir um intervalo entre os carros que exigem um mesmo opcional. Cada item opcional é instalado por estações de trabalho diferentes, capazes de atender apenas uma fração de carros ao longo do processo. Durante o sequenciamento dos carros na linha de produção surgem restrições de capacidade que devem ser respeitadas.

Estas restrições podem ser representadas pela razão p_j/q_j, indicando que não mais que p_j carros que exigem um mesmo item opcional j devem estar numa subsequência de tamanho q_j. Essa fração é planejada de forma que nenhuma restrição de capacidade seja violada, assim, os carros devem ser organizados de forma que seja possível atender todas as exigências por itens opcionais sem haver sobrecarga das estações de trabalho. Surge então, o Problema de Sequenciamento de Carros – *Car Sequencing Problem* (CSP) proposto por Parrello et al. (1986) e posteriormente abordado pelas comunidades de Pesquisa Operacional.

O CSP é abordado tanto como um problema de decisão como de otimização. De acordo com Kis (2004), o CSP é caracterizado como um problema $\mathcal{NP}$-difícil, implicando que não se conhece algoritmo que gere soluções ótimas em tempo determinístico polinomial. Entretanto, existe um campo de algoritmos que têm como objetivo estudar métodos de soluções aproximadas para tais problemas. Dentre estes métodos de solução encontram-se as metaheurísticas, métodos de busca por soluções ótimas globais e estratégias para fugir de ótimos locais, encontrando a solução ótima ou se aproximando dela sem a necessidade de explorar todo o espaço de busca do problema.

Uma categoria de metaheurísticas são os algoritmos evolutivos, baseados no conceito da Teoria da Seleção Natural proposta por Darwin (1859). Uma população é constituída por vários indivíduos e devido a fatores externos do ambiente, os mais fortes terão maior probabilidade de fazer parte das próximas gerações, resultando numa população cada vez mais propícia a se adaptar às condições de sobrevivência no ambiente. Uma classe de algoritmos evolutivos muito utilizados são os algoritmos genéticos baseados na proposta original de Holland (1975).

Este trabalho propõe minimizar o número de violações de restrições de capacidade através do Algoritmo Genético de Chaves Aleatórias Viciadas – *Biased Random-Key Genetic Algorithm* (BRKGA). O método proposto tem se mostrado robusto em diversas aplicações e nunca foi aplicado ao CSP. Após a implementação do método, serão efetuados experimentos a fim de comparar os resultados com os melhores métodos da literatura. Os experimentos serão base-

ados em instâncias disponíveis em uma biblioteca de problemas de teste para solucionadores de restrições denominada CSPlib, criada por CSPLib (1999).

A biblioteca contém uma lista de linguagens de restrições e ferramentas que podem ser usadas para resolver problemas através destas linguagens. A principal motivação da CSPlib é o foco em pesquisa em restrições de problemas puramente aleatórios ou mais estruturados. Existem atualmente 81 problemas pertencentes a 12 categorias propostos por 12 autores. Coincidentemente, o primeiro problema a fazer parte da CSPlib foi o CSP, proposto por Barbara Smith (1999). Várias instâncias do CSP encontram-se disponíveis nesta biblioteca.

1.1 Motivação

Empresas de desenvolvimento e fabricação de veículos buscam formas de produção sem sobrecarga na produtividade e com o menor número de violações possível. O Brasil é um dos maiores fabricantes de carros, portanto, há uma necessidade de solução através de métodos que busquem resolver o problema de sequenciamento de carros. De acordo com Kis (2004), o CSP é um problema $\mathcal{NP}$-difícil, e portanto, não existe algoritmo que gere soluções ótimas em tempo determinístico polinomial, dessa forma, o CSP é de grande interesse das comunidades de Pesquisa Operacional.

1.2 Objetivos

São objetivos gerais deste trabalho, propor a implementação de um método para determinação de uma sequência de veículos que atenda toda a demanda por opcionais respeitando todas as restrições de capacidade. O que inclui a utilização de heurísticas e metaheurísticas, avaliando-as com base em dados reais e com problemas teste publicamente disponíveis. Além dos objetivos principais, outros produtos deste projeto de pesquisa serão trabalhos publicados em periódicos e eventos nacionais e internacionais, os quais contribuem para a promoção dos centros de pesquisas nacionais e também da tecnologia. São objetivos específicos:

- Propor a implementação da metaheurística BRKGA para a solução do problema tratado com foco na versão de otimização;
- Modificar a API do BRKGA para manipular a forma de evolução, utilizando estratégias disponíveis na literatura;
- Propor a implementação de operadores adaptativos no algoritmo genético;
- Pesquisar métodos de busca local a serem adaptados ao método proposto para obtenção de melhores resultados;

- Comparar o desempenho do método proposto frente ao estado da arte relacionado ao CSP.

1.3 Organização do Trabalho

O restante do trabalho está organizado da seguinte forma: no Capítulo 2, encontra-se a revisão bibliográfica, que descreve os trabalhos mais relevantes presentes na literatura. O Capítulo 3 introduz a fundamentação teórica necessária para o entendimento do CSP. No Capítulo 4 é descrito o algoritmo genético de chaves aleatórias viciadas. No Capítulo 5, há o detalhamento do BRKGA aplicado ao CSP. No Capítulo 6 são relatados os experimentos computacionais e, por fim, as conclusões acerca deste trabalho são apresentadas no Capítulo 7.

Capítulo 2

Revisão da Literatura

O CSP é comumente abordado em duas versões: a de decisão e a de otimização. A versão de decisão, tem por objetivo analisar o problema a fim de obter como retorno uma resposta na forma sim ou não, indicando que o problema tem solução sem violar restrições ou não. Na versão de otimização, o problema é analisado de forma que a melhor solução é apresentada dentre um grupo de soluções viáveis de acordo com alguma função objetivo, como por exemplo, minimizar o número de violações de restrições.

Vários autores propuseram métodos heurísticos e mataheurísticos para obterem soluções ótimas, baseando seus resultados através de um conjunto de instâncias disponíveis a este fim. O objetivo deste trabalho é tratar a versão de otimização e os trabalhos mais relevantes são descritos a seguir.

O primeiro artigo que descreveu o CSP foi Parrello et al. (1986), que propuseram e avaliaram abordagens com a finalidade de obter boas sequências na linha de montagem baseadas em um programa de raciocínio automático. O principal objetivo é selecionar o carro que possui menos restrições e inserí-lo na sequência ao longo da linha de montagem. O raciocínio automático é usado então, para desenvolver métodos alternativos para resolver o problema. Outro objetivo é descobrir heurísticas úteis para a solução do problema.

Foram propostas por Drexl e Kimms (2001) técnicas para geração de colunas, um método formulado como um modelo de programação inteira que permite flexibilizar os limites inferiores através de programação linear com folgas. Este método é capaz de gerar bons limites inferiores, trazendo bons resultados para o problema de sequenciamento de carros.

Uma busca local baseada em permutações com heurísticas para decidir qual a melhor forma de se mover no espaço de busca, a fim de encontrar uma solução ótima para o problema, foi utilizada por Puchta e Gottlieb (2002). Este método se mostra eficiente e robusto em relação a abordagens de busca gulosa, resolvendo 6 instâncias geradas aleatoriamente com 1000 carros.

A metaheurística *Large Neighborhood Search* (LNS) foi utilizada por Perron e Shaw (2004). Nesta abordagem, foram propostos dois tipos de busca local: método de trocas de blocos, uma adaptação do LNS para torná-lo mais eficiente e métodos de deslocamento, que desloca uma

quantidade de carros para o início da sequência a fim de reorganizá-los. Este método mostrou ser muito eficiente obtendo ótimos resultados de acordo com instâncias disponíveis na CSPlib.

O CSP foi considerado no ROADEF'2005 *Challenge* Nguyen (2005). O problema era semelhante, porém, levava em consideração as cores dos carros. Um conjunto de carros não necessariamente da mesma cor eram sequenciados numa linha de produção e ao final, todos deveriam ser produzidos cada qual com seus opcionais e suas cores. Quando dois carros consecutivos eram de cores distintas, havia a necessidade de limpar as pistolas de pulverização com solvente. Dessa forma, o foco era minimizar o consumo de solvente, ou seja, organizar os carros na sequência de forma que os da mesma cor sejam colocados em ordem consecutiva.

De acordo com Solnon et al. (2008), o ROADEF'2005 *Challenge* é organizado a cada dois anos pela Sociedade Francesa de Pesquisa Operacional e Apoio à Tomada de Decisão. O objetivo é permitir que indústrias possam acompanhar a evolução no domínio da pesquisa operacional e análise de decisões. O tema do ROADEF ocorrido em 2005, foi o CSP, proposto pelo fabricante de automóveis *Renault*.

O problema consiste em uma sequência de carros numa linha de produção com limitações de dosagem de tinta e minimização do consumo de solventes usados para a limpeza das pistolas de pulverização. É apresentada no trabalho uma vasta revisão de métodos exatos e métodos heurísticos existentes na literatura com o objetivo de resolver o CSP. A equipe vencedora do desafio Estellon et al. (2008), inicialmente propôs um método de busca em vizinhança em larga escala, obtendo resultados limitados em um tempo de execução 10 minutos.

Na fase final, a equipe mudou sua abordagem para um rápido método de busca local. As instâncias foram todas fornecidas pela fabricante *Renault* e divididas em três conjuntos, o primeiro com 16 instâncias, o segundo com 45 e o terceiro com 19 instâncias.

Foram revisados e comparados por Gravel et al. (2005) três métodos para resolver o problema de sequenciamento de carros. O primeiro foi um modelo de programação inteira. O segundo, uma abordagem por satisfação de restrições e o terceiro método foi uma adaptação da Otimização de Colônia de Formigas. Os métodos propostos são eficazes e foi possível resolver diversas instâncias com grande facilidade. Um novo conjunto de problemas foi gerado e solucionado através dos métodos propostos.

Uma ideia de poda de busca em árvore baseada em regras de prioridade foi trazida por Drexl et al. (2006). Nesta abordagem, comparava-se o tempo de execução gasto para encontrar uma solução ótima, levando em consideração os passos de falhas (isto é, o número de ramificações induzidas por um espaço vazio na solução de um subproblema), até que uma solução seja encontrada. Usando regras de prioridade, nota-se que uma solução ótima é encontrada rapidamente mesmo para problemas mais difíceis. Comparando a quantidade de passos de falhas com o grande número de sequências, concluiu-se que o procedimento é poderoso, diminuindo consideravelmente o espaço de busca.

O método *Very Large-Scale Neighborhood* (VLNS) – Busca em Vizinhança em Larga Es-

cala foi proposto por Estellon et al. (2006). Esta abordagem baseia-se em um método simples de descida onde cada iteração consiste em resolver exatamente um subproblema por programação linear inteira. O método foi comparado com o *Very Fast Local Search* – Busca Local Extremamente Rápida, considerado o estado da arte daquele momento que foi usado no ROADEF'2005 *Challenge* (Nguyen, 2005) e o VLNS se mostrou eficiente em diversos casos. Nesta abordagem também houve uma formulação de Programação Linear Inteira para produzir o número de variáveis de carros que pertencem a uma mesma classe.

O algoritmo *Ant Colony Optimization* (ACO) foi descrito por Solnon (2008) a fim de resolver o CSP. Foram apresentadas duas estruturas de feromônio para este algoritmo: a primeira estrutura de feromônio visa aprendizagem de boas sequências de carros enquanto a segunda visa aprendizagem de carros críticos. As duas estruturas foram comparadas experimentalmente, tendo performances complementares. Ambas estruturas foram comparadas com *Very Fast Local Search* (VFLS) e com uma metaheurística baseada em busca local. Através da combinação das duas estruturas de feromônio, foi possível resolver 82 instâncias geradas por Perron e Shaw (2004).

Uma extensão do Problema de Sequenciamento de Carros foi desenvolvida por Bautista et al. (2008a). Esta extensão, consiste em uma conveniente adição de restrições para um número mínimo de operações em algum intervalo de tempo de determinado comprimento s. Surge então, um grupo de restrições suaves representadas da forma r_j/s_j, em que para cada sub-sequência de s_j carros, o opcional j tem que estar presente pelo menos r_j vezes. Para resolver as instâncias desta nova extensão, foi proposta uma metaheurística *Greedy Randomized Adaptive Search Procedure* (GRASP) – Procedimento de Busca Gulosa Aleatória. Esta metaheurística foi capaz de obter uma solução ótima em 74 instâncias da CSPlib.

Uma abordagem *Branch & Bound* foi proposta por Fliedner e Boysen (2008). O espaço de solução é representado por uma árvore de disjunção, que contém todas as soluções viáveis. Há uma desvantagem quando a busca é efetuada em profundidade, caso não sejam feitas boas escolhas. Sendo assim, surge uma extensão do algoritmo denominada *Scattered Branch & Bound* (SBB). A ideia desse algoritmo é subdividir a árvore em várias regiões, "soluções de corte". O algoritmo resolve todos os problemas de otimização em tempo de execução extremamente curto, sendo que a primeira solução é sempre ótima, abortando o processo de busca quando há violações.

O método *Beam Search* foi aplicado ao CSP por Bautista et al. (2008b), um algoritmo de busca em grafos capaz de resolver muitas instâncias conhecidas como viáveis. Este método é capaz de resolver problemas com maior número de carros com maior tempo de execução. O algoritmo se mostra eficaz na busca por soluções que respeitam as restrições de capacidade.

Propostas de algoritmos de busca foram trazidas por Boysen et al. (2011), tais como *Breadth-First Search* (BFS), *Beam Search*, *Iterative Beam Search* (IBS) e Busca A*. Comparando os algoritmos citados anteriomente, concluiu-se que BFS, A* e IBS retornam soluções

ótimas e não excedem o tempo limite. Para um pequeno conjunto de instâncias, o método A* obteve soluções ótimas, já para problemas com instâncias maiores, o método IBS se mostra melhor em termos de desempenho comparado com A* e BFS.

Novamente, uma implementação do método *Iterated Beam Search* foi proposta por Yavuz (2013). O algoritmo foi executado com 54 instâncias se mostrando muito eficiente, sendo que para 35 dos 36 casos existentes na literatura, o algoritmo encontrou solução ótima em menos de uma hora. Para as instâncias já resolvidas anteriormente por outros métodos, o algoritmo encontrou solução ótima em menos de um minuto. Para 10 de 18 instâncias recém-criadas, o algoritmo encontrou com sucesso solução ótima em uma hora, sendo considerado de alto desempenho tenho resolvido 45 problemas de um conjunto de 54 instâncias. Dessas instâncias, 36 foram desenvolvidas por Drexl et al. (2006) e outras 18 foram obtidas através do gerador de Drexl.

Três tipos de codificações da Forma Normal Conjuntiva (FNC) foram estudadas por Mayer-Eichberger (2013), que envolvem tanto a versão de decisão quanto a de otimização. Na versão de otimização, concluiu-se que os limites inferiores e superiores na literatura possuem diversas definições e não podem ser diretamente comparados. Esta abordagem mostra que em muitos casos, as distintas definições forçam os mesmos limites tanto inferior quanto superior. Juntando os dois limites, foi possível resolver 21 de 30 novas instâncias propostas por Gravel et al. (2005).

Novamente, foi implementada uma IBS por Golle et al. (2014). Tal heurística foi aplicada de duas formas: IBS1, que utiliza limites inferiores, e IBS2, que utiliza a ideia baseada em um novo limite inferior proposto no trabalho. Desta forma, o IBS2 é capaz de reduzir a quantidade de nós analisados quando comparado com o IBS1. O IBS se mostrou superior à solução exata mais conhecida, uma implementação do método *Scattered Branch and Bound* (SBB) proposta por Fliedner e Boysen (2008), resolvendo 15 de 18 instâncias disponíveis na CSPlib.

Por fim, Gunay e Kula (2017) apresentaram uma abordagem um pouco distinta. Os carros a serem produzidos devem passar por uma oficina de pintura e portanto, devem ser organizados para formarem blocos com o maior número de carros com a mesma cor. Porém, devido a defeitos de pintura, alguns carros devem ser retirados da linha de produção e armazenados em um *buffer* para serem reinseridos novamente na sequência original em posições mais adequadas.

Foi proposto um algoritmo de aproximação da média, *Sample Average Approximation* (SAA). Foram gerados aleatoriamente 4 conjuntos contendo 90, 180, 360 e 450 carros e 3 tamanhos iniciais para formar o *buffer*: 3, 6 e 12, sendo estes tamanhos duplicados a cada iteração subsequente. Concluiu-se que o tempo de execução aumenta à medida que o número de carros cresce. Para a sequência com 450 carros, o algoritmo foi capaz de determinar o tamanho do *buffer* com uma capacidade de 60 veículos em aproximadamente 3,5 horas. Este período de tempo é suficiente para executar o algoritmo e determinar a sequência de produção, visto que o plano de produção é comunicado com um ou mais dias de antecedência.

Dentre os métodos exatos e metaheurísticos, alguns se destacaram obtendo bons resultados em diversas instâncias, como VFLS proposto por Estellon et al. (2008), VLNS descrito por Estellon et al. (2006), ACO apresentado por Solnon (2008), o *Branch & Bound* proposto por Fliedner e Boysen (2008), a metaheurística GRASP de Bautista et al. (2008a), FNC apresentadas por Mayer-Eichberger (2013), a IBS por Golle et al. (2014), dentre outros. Tais algoritmos podem ser caracterizados como o estado da arte, mas não houve dominância de nenhum método. O BRKGA difere-se destes métodos pelo fato de se tratar de uma metaheurística evolutiva, sendo de grande interesse saber o seu comportamento diante do problema tratado, visto que não foi aplicado ao problema anteriormente.

Capítulo 3

O Problema de Sequenciamento de Carros

O *Car Sequencing Problem* (CSP) surge da necessidade de gerar o sequenciamento da produção de carros em uma linha de montagem de uma indústria automotiva. O objetivo é obter uma sequência viável que respeite todas as restrições de capacidade, evitando sobrecarga das estações de trabalho. Uma instância do CSP com n carros, m itens opcionais e k classes pode ser definida como uma tupla (C, O, p, q, r) em que:

- $C = \{c_1, \cdots, c_k\}$ indica as classes de carros a serem produzidos;
- $O = \{o_1, \cdots, o_m\}$ corresponde ao conjunto de itens opcionais disponíveis;
- $p : O \to \mathbb{N}$ e $q : O \to \mathbb{N}$, definem as restrições de capacidade associadas à cada item opcional $o_i \in O$, ou seja, em uma subsequência de tamanho $q(o_i)$ não deve haver mais que $p(o_i)$ carros requerendo o item opcional o_i;
- $r : C \times O \to \{0, 1\}$ determina se há demanda por um item opcional ou não, retornando 1 se o item opcional se aplicar à uma determinada classe e 0, caso contrário.

A partir destes dados é possível obter algumas definições, como o que determina uma solução, o tamanho de uma sequência, conjunto de itens opcionais pertencentes a cada classe e as subsequências que fazem parte de uma sequência. Tais definições estão descritas a seguir:

- $S = [1, \cdots, n]$, representa uma sequência de carros, ou seja, uma solução;
- $|S|$ indica o tamanho de uma sequência de carros, ou seja, quantos carros ela possui;
- $O_c \subseteq \{1, \cdots, k\}, \forall c \in C$, indica o conjunto de itens opcionais de cada classe;
- Uma sequência S_f é dada em função de outra sequência S, de forma que $S_f \subseteq S$, se existirem duas possíveis sequências S_1 e S_2 de forma que $S = S_1 \ . \ S_f \ . \ S_2$.

A violação de uma restrição de capacidade é dada pela Equação 3.1. Sempre que houver mais que $p(o_i)$ carros requerendo o item opcional o_i numa determinada subsequência $q(o_i)$, ocorrerá uma violação e será retornado o valor 1, caso contrário, não haverá violação e consequentemente, o valor retornado será 0.

$$Violação(S_k, o_i) = \begin{cases} 0 & \text{se} \quad r(S_k, o_i) \leq p(o_i) \\ 1 & \text{caso contrário} \end{cases} \tag{3.1}$$

A função objetivo pode ser calculada através do número de violações de uma sequência S, dado pela Equação 5.1, obtido através do somatório de todas as violações ocorridas. A solução é considerada ótima se violar o mínimo de restrições possível.

$$\min Custo(S) = \sum_{o_i \in O} \sum_{\substack{S_k \subseteq S| \\ |S_k| = q(o_i)}} Violação(S_k, o_i) \tag{3.2}$$

Define-se para cada item opcional o_j, o conjunto de classes de carros que requerem este opcional, $C_{o_j} = \{c | o_j \in O_c\}$ e a demanda por itens opcionais, $D_{o_j} = \sum_{c \in C_{o_j}} D_c$ que representa o número de ocorrências de determinada classe na sequência. Como exemplo, considere um processo de produção de 8 carros, portanto, $n = 8$. Considere também 4 classes $\{c_{o_1}, c_{o_2}, c_{o_3}, c_{o_4}\}$ e 5 itens opcionais de forma que:

- $O_{c_1} = \{1, 3\}, \quad O_{c_2} = \{2, 3\}, \quad O_{c_3} = \{4, 5\}, \quad O_{c_4} = \{1, 4\};$
- $D_{c_1} = 2, \quad D_{c_2} = 2, \quad D_{c_3} = 2, \quad D_{c_4} = 2;$
- Restrições de capacidade na razão $3/5,\ 1/3,\ 2/5,\ 2/3$ e $1/4$, respectivamente.

De acordo com os dados, temos que os itens opcionais 1 e 3 são requeridos pelos carros da classe 1, os opcionais 2 e 3 pelos carros da classe 2, os carros da classe 3 requerem os opcionais 4 e 5 e os da classe 4, requerem os opcionais 1 e 4. Pode-se observar também a partir dos dados que a demanda por itens opcionais de cada classe é igual a 2.

Consequentemente, temos: $C_{o_1} = \{1, 4\}$, $C_{o_2} = \{2\}$, $C_{o_3} = \{1, 2\}$, $C_{o_4} = \{3, 4\}$, ou seja, as classes de carro 1 e 4 requerem o opcional 1, a classe 2 requer o opcional 2, as classes 1 e 2 requerem o opcional 3 e o opcional 4 é requerido pelas classes de carro 3 e 4. A demanda por itens opcionais de cada classe é $D_{o_1} = 4$, $D_{o_2} = 2$, $D_{o_3} = 4$, $D_{o_4} = 4$.

Podemos obter três soluções viáveis que respeitam todas as restrições: $[c_2; c_3; c_1; c_4; c_4; c_2; c_3; c_1]$, $[c_1; c_3; c_2; c_4; c_4; c_1; c_3; c_2]$ e $[c_1; c_3; c_2; c_4; c_4; c_2; c_3; c_1]$. Uma instância do problema proposta por Kis (2004) sintetizada na Tabela 3.1 é composta por estes dados e está organizada da seguinte maneira: na primeira coluna, existem 5 itens opcionais a serem instaladas nos carros.

Nas quatro colunas do meio estão as classes de carros: cada classe requer um conjunto distinto de itens opcionais, como por exemplo, as classes 3 e 4 requerem o opcional 4, mas a classe

3 requer também o opcional 5 e a classe 4, o opcional 1. Na coluna mais à direita, encontram-se as restrições de capacidade, analisando a primeira restrição, na razão 3:5, conclui-se que em uma subsequência de tamanho 5, não deve haver mais que 3 carros exigindo o opcional 1. O número de requisições por opcionais de cada classe é mostrado na última linha da tabela.

Tabela 3.1: Instância do problema com 4 classes de carros e 5 itens opcionais.

Itens Opcionais	Classes de Carros				Restrições
	1	2	3	4	$p:q$
1	√			√	3:5
2		√			1:3
3	√	√			2:5
4			√	√	2:3
5			√		1:4
Req.	2	2	2	2	

Como exemplo, a Tabela 3.2 ilustra a sequência $[c_2; c_3; c_1; c_4; c_4; c_2; c_3; c_1]$ citada anteriormente. É possível analisar a produção de oito carros, sendo dois de cada tipo. Os carros foram organizados de forma a não violar nenhuma restrição de capacidade, portanto, podemos usar o conceito de janela deslizante para verificar tal afirmação.

Pode-se perceber que o item opcional 1 é requerido pelos carros das classes 1 e 4 e a razão de capacidade é 3:5 indicando que não mais que 3 carros que requerem o opcional 1 podem estar numa subsequência de tamanho 5. Considerando os carros que requerem o item opcional 1 e uma janela deslizante começando dos 5 primeiros carros, pode-se perceber que ao deslizarmos a janela por todos os oito carros na Tabela 3.2, nenhuma violação ocorre.

Tabela 3.2: Solução da sequência $[c_2; c_3; c_1; c_4; c_4; c_2; c_3; c_1]$.

Itens Opcionais	Classes de Carros								Restrições
	2	3	1	4	4	2	3	1	$p:q$
1			√	√	√			√	3:5
2	√					√			1:3
3	√		√			√		√	2:5
4		√		√	√		√		2:3
5		√					√		1:4
Req.	2	2	2	2	2	2	2	2	

É preciso avaliar o valor da solução de acordo com as violações das restrições. Ao analisar a Tabela 3.2, percebe-se que nenhuma restrição foi violada. Como a função objetivo foi formulada para minimizar o número de violações de uma sequência, verifica-se que esta sequência possui valor igual a zero, portanto, a solução é ótima. Podem existir mais de uma solução, mas nenhuma é melhor que a outra.

Capítulo 4

Algoritmo Genético de Chaves Aleatórias Viciadas

Neste capítulo, são descritos em detalhes todos os passos do Algoritmo Genético de Chaves Aleatórias Viciadas (BRKGA, do inglês *Biased Random-Key Genetic Algorithm*). Trata-se de uma metaheurística evolutiva para problemas de otimização discreta e global. O algoritmo é baseado na teoria da evolução do fisiologista e naturalista Darwin (1859), em que os indivíduos mais fortes têm maior probabilidade de encontrar um parceiro e perpetuar seu material genético nas próximas gerações.

O BRKGA, por meio de uma analogia, evolui uma população de soluções até satisfazer um critério de parada, como por exemplo, o número de populações evoluídas, tempo de execução ou qualidade da melhor solução encontrada. As etapas do método se dividem em codificação, decodificação, população inicial, elitismo, mutação e *crossover* e são explicadas em detalhes nas seções a seguir.

4.1 Codificação

Cada solução é representada por um vetor de n chaves aleatórias. Cada um dos genes pertencentes ao cromossomo é uma chave que pode assumir valores num intervalo fechado $[0,1] \in \mathbb{R}$. As chaves aleatórias permitem variedade de indivíduos, tornando o processo de evolução da população mais dinâmico.

A codificação dos vetores de chaves aleatórias, possibilita o desacoplamento do BRKGA em relação ao problema. Desta forma, o algoritmo pode acessar diretamente as chaves aleatórias, sendo necessário implementar somente a parte de decodificação da solução, facilitando o reuso do código. Um exemplo de cromossomo com oito chaves aleatórias é mostrado na Figura 4.1 em que cada posição contém uma chave aleatória correspondente às suas características.

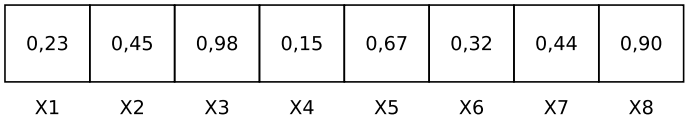

Figura 4.1: Cromossomo com 8 chaves aleatórias. Adaptado de Gonçalves e Resende (2011).

4.2 População Inicial

O algoritmo começa com uma população inicial que será evoluída ao longo das gerações. Por se tratar de um algoritmo genético, cada solução é representada por um indivíduo (ou cromossomo) e cada gene representa uma parte da solução para um problema. Inicialmente, a população é formada por p vetores compostos por n chaves geradas de forma aleatória obtidas através do gerador de números aleatórios desenvolvido por Matsumoto e Nishimura (1998), em que cada vetor representa um indivíduo.

O tamanho recomendado para uma população é $p = an$, em que a corresponde a um valor constante maior ou igual a 1 e n corresponde ao tamanho de um cromossomo, medido pelo número de genes. O processo de evolução da população corresponde à busca por uma solução ótima local ou global em que são realizados procedimentos de busca local e perturbação.

4.3 Decodificação

Um decodificador é um algoritmo determinístico que toma como entrada um vetor com n chaves aleatórias e produz como saída uma solução com o menor número de violações. Após a inicialização da população, a decodificação é realizada. O decodificador calcula o valor originalmente ordenando o vetor de chaves aleatórias, gerando uma permutação que corresponde aos índices dos elementos ordenados. A Figura 4.2 ilustra como é feito o mapeamento pelo decodificador em que os vetores de chaves aleatórias são mapeados no espaço de solução do problema.

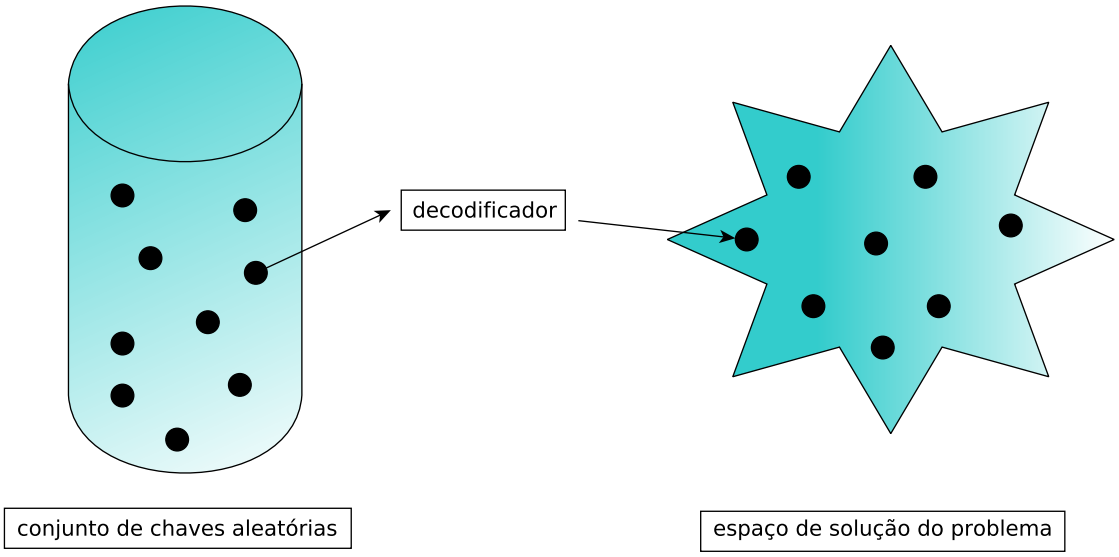

Figura 4.2: Mapeamento feito pelo decodificador, em que uma solução do espaço de chaves aleatórias é mapeada no espaço de solução do problema. Adaptado de Gonçalves e Resende (2011).

4.4 Elitismo

De acordo com a teoria evolucionista de Darwin (1859), os indivíduos mais fortes têm maior probabilidade de formarem as próximas gerações. No BRKGA, o grupo composto pelos elementos mais fortes é denominado conjunto elite e os outros indivíduos compõem o conjunto não-elite. Em termos de otimização, os indivíduos mais fortes são aqueles que mais se aproximam da solução ótima. Também funciona como mecanismo de memória de longo prazo.

O valor recomendado para formar o conjunto elite está entre 10% e 25% da população. O objetivo do elitismo é garantir que os melhores indivíduos façam parte das próximas gerações, fazendo com que a solução seja direcionada para um ótimo local ou global. Todos os elementos do conjunto elite são então adicionados sem alteração na população da próxima geração.

4.5 Mutação

A população de uma nova geração é composta por três conjuntos de indivíduos, o conjunto elite, os mutantes e o restante é obtido a partir do cruzamento entre os elementos dos conjuntos elite e não-elite. Após adicionar a população elite na próxima geração, é necessário criar indivíduos a fim de compor o segundo conjunto. Desta forma, é gerado aleatoriamente um conjunto de cromossomos denominados mutantes, com a finalidade de obter uma diversidade de indivíduos, possibilitando a evolução da mesma, evitando que a solução fique aprisionada

em ótimos locais que não sejam globais. O recomendado é que a nova população seja composta por um percentual entre 10% e 30% de mutantes.

4.6 *Crossover*

O terceiro grupo da população da próxima geração é obtido pela combinação de pares de soluções, em que um pai pertence ao conjunto elite e o outro pertence ao conjunto não-elite. O *crossover* é realizado através do *Parameterized Uniform Crossover* de Spears e De Jong (1995). O algoritmo seleciona os pais aleatoriamente e com reposição, portanto, um pai pode ter mais de um filho na mesma geração.

Para garantir que a nova população contenha bons indivíduos, existe uma moeda viciada com probabilidade entre 50% e 80% de um filho herdar as características do pai elite. Para cada gene do filho, é gerado um valor aleatório que determina se o filho herdará o gene do pai elite ou do pai não-elite. O processo de *crossover* é mostrado na Figura 4.3.

Cromossomo A	0,23	0,56	0,98	0,15	0,67	0,51	0,44	0,77
Cromossomo B	0,60	0,45	0,68	0,36	0,10	0,32	0,60	0,90
Número Aleatório	0,55	0,90	0,65	0,26	0,55	0,90	0,65	0,26
70% probabilidade	<	>	<	<	<	>	<	>
Cromossomo C	0,23	0,45	0,98	0,15	0,67	0,32	0,44	0,90

Figura 4.3: *Crossover*. Adaptado de Gonçalves e Resende (2011).

Nota-se pela Figura 4.3 a geração de um filho C a partir dos pais A, do conjunto elite e o B, do conjunto não-elite. Para cada gene, um valor aleatório é gerado com 70% de probabilidade do filho C herdar as características do pai elite. Considerando os dois primeiros genes, os valores gerados foram 0,55 e 0,90. Como o primeiro valor foi menor que a probabilidade, então o filho herdou o gene 0,45 do pai A, já no segundo gene, o valor aleatório foi maior que 70%, portanto, o gene escolhido foi o do pai não-elite. O mesmo processo se repete para os demais genes.

4.7 Visão Geral

Na Figura 4.4, é apresentado um fluxograma que ilustra o funcionamento do BRKGA de forma geral. Basicamente, o algoritmo inicia-se com uma população gerada aleatoriamente, na sequência, cada solução é decodificada, o critério de parada é verificado, a solução é ordenada e classificada em elite e não-elite, e por fim, uma nova população é gerada a partir de três

conjuntos, o elite, os mutantes e o *crossover*. O processo se repete até que um critério de parada seja satisfeito.

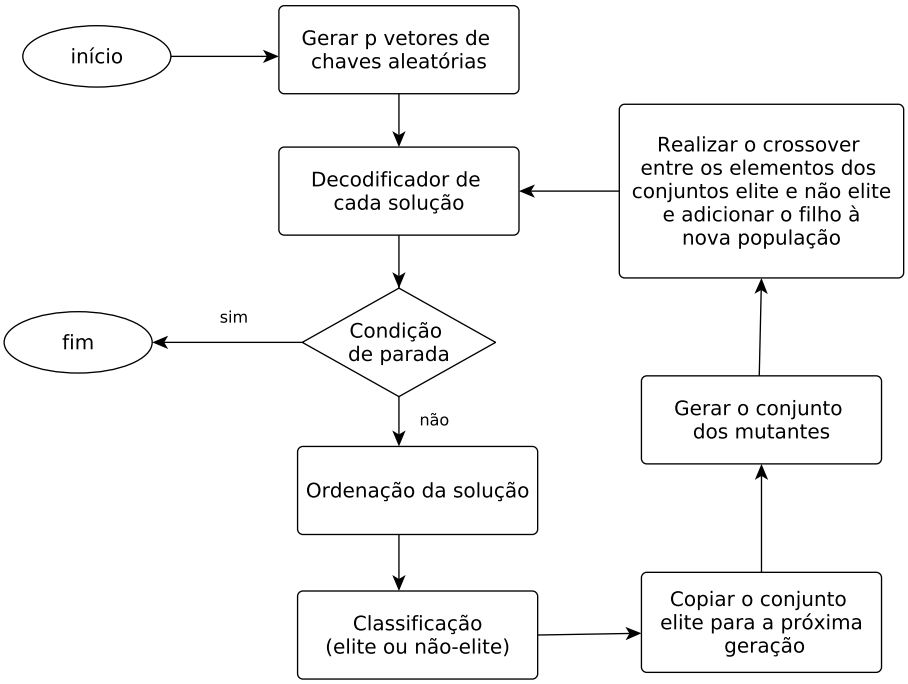

Figura 4.4: Fluxograma do algoritmo BRKGA. Adaptado de Gonçalves e Resende (2011).

A partir da Figura 4.5, pode-se ver a formação de uma nova população formada por três conjuntos. O primeiro conjunto a compor a nova população, é o conjunto elite, correspondente aos melhores indivíduos da geração atual. O segundo é o conjunto dos mutantes, formado por cromossomos aleatórios com o objetivo de diversificar a população. O terceiro conjunto é gerado a partir das combinações entre os elementos dos conjuntos elite e não-elite.

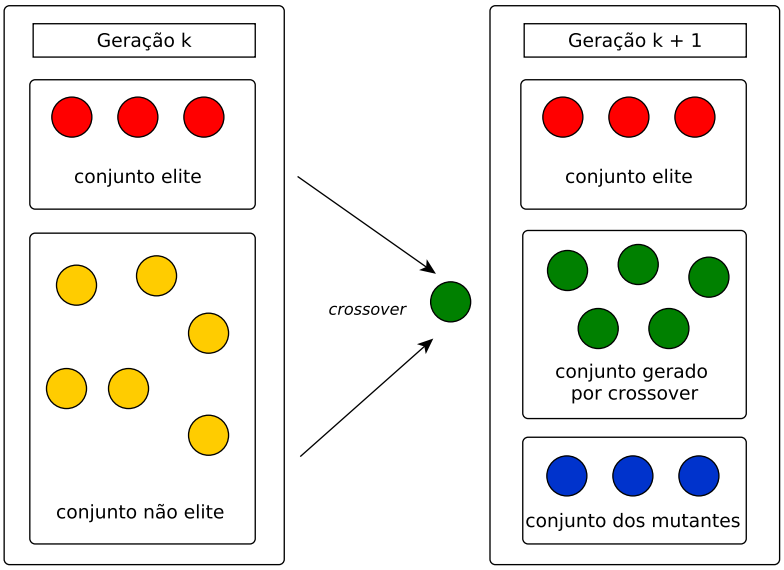

Figura 4.5: Geração da população da próxima iteração. Adaptado de Gonçalves e Resende (2011).

4.8 Pseudocódigo

A execução do algoritmo pode ser explicada através do Algoritmo 1. Na linha 1, o valor da melhor solução encontrada começa com um valor infinitamente grande, caso o problema seja de minimização, pois não existe ainda uma solução encontrada. Nas linhas 2 até 30, ocorrem as evoluções das populações e o algoritmo termina quando o critério de parada for satisfeito. Esse critério pode ser por exemplo, o número de populações evoluídas, tempo de execução ou qualidade da melhor solução encontrada.

O critério de parada do BRKGA aplicado ao CSP foi o número de populações evoluídas. Na linha 3, a população a ser evoluída é inicializada com p vetores compostos por n chaves aleatórias. A evolução de uma população específica ocorre nas linhas 4 a 29 e termina quando um critério de reinicialização for satisfeito, que pode ser um número máximo de gerações sem que a melhor solução tenha sido alterada. A cada iteração, as seguintes operações são realizadas: na linha 5, todas as soluções são decodificadas e seus valores avaliados.

Na linha 6, a população é dividida em dois conjuntos, conjunto elite e o conjunto não-elite. A população da próxima geração é inicializada na linha 7 com o conjunto elite da geração atual sem haver alterações neste conjunto. Na linha 8, o conjunto mutante é gerado. Esse conjunto também é adicionado à população da próxima iteração na linha 9. Por enquanto, a nova população é formada pelo conjunto elite e pelo conjunto dos mutantes, porém, a população ainda está incompleta, pois seu tamanho é inferior ao da geração atual.

Dessa forma, é necessário a geração de filhos para completar os vetores da população da nova geração. Esse procedimento ocorre nas linhas 10 a 22. Nas linhas 11 e 12, os pais a

e b são escolhidos, respectivamente, de forma aleatória das populações dos conjuntos elite e não-elite. A geração do filho c a partir dos pais a e b ocorre entre as linhas 13 a 20. Uma "moeda viciada", com probabilidade maior que 50% de dar cara é lançada n vezes. Caso o i-ésimo lance der cara, então o filho herda a i-ésima chave do pai a, caso contrário, ele herda a chave do pai b.

O filho c é então adicionado à nova população na linha 20. A geração da nova população termina quando seu tamanho for igual ao da população anterior. Na linha 23, a população é atualizada, dessa forma, a nova população passa a ser a população gerada descrita anteriormente. A melhor geração da atual população é computada na linha 24 e se o valor for melhor que as soluções encontradas até o momento, ela e seu valor são gravados nas linhas 26 e 27. O valor da melhor solução encontrada é retornado na linha 31.

Algoritmo 1: BRKGA $(|P|, |P_{\bar{e}}|, |P_m|, n, p_a)$

```
1  Inicialize o valor da melhor solução encontrada: f* ← ∞;
2  enquanto critério de parada não for satisfeito faça
3  |  Gere a população P com vetores de n chaves aleatórias;
4  |  enquanto critério de reinicialização não for satisfeito faça
5  |  |  Avalie o custo de cada solução nova em P;
6  |  |  Particione P em dois conjuntos: P_e e P_ē;
7  |  |  Inicialize a população da próxima geração: P+ ← P_e ;
8  |  |  Gere o conjunto de mutantes P_m, cada um com n chaves aleatórias ;
9  |  |  Adicione P_m à população da próxima geração: P+ ← P+ ∪ P_m;
10 |  |  para todo i ← 1 até |P| − |P_e| − |P_m| faça
11 |  |  |  Escolha o pai a aleatoriamente do conjunto P_e;
12 |  |  |  Escolha o pai b aleatoriamente do conjunto P_ē;
13 |  |  |  para todo j ← 1 até n faça
14 |  |  |  |  Jogue uma moeda viciada com probabilidade p_a > 1/2 de dar cara;
15 |  |  |  |  se jogada der cara então
16 |  |  |  |  |  c[j] ← a[j];
17 |  |  |  |  senão
18 |  |  |  |  |  c[j] ← b[j];
19 |  |  |  |  fim
20 |  |  |  fim
21 |  |  |  Adicione o filho à população da próxima geração: P+ ← P+ ∪ {c};
22 |  |  fim
23 |  |  Atualize a população: P ← P+;
24 |  |  Ache a melhor solução X+ em P: X+ ← argmin{f(x)|x ∈ P};
25 |  |  se f(X+) < f* então
26 |  |  |  X* ← x+;
27 |  |  |  f* ← f(X+);
28 |  |  fim
29 |  fim
30 fim
31 retorna X*
```

4.9 Interface de Programação de Aplicações

Foi desenvolvida por Toso e Resende (2015) uma *Application Programming Interface* (API), implementada em C++, orientada a objetos e baseada em algoritmos genéticos de chaves aleatórias viciadas. Como a API é implementada em C++, o uso de paralelismo através de memória compartilhada é facilitado, sendo necessário somente implementar o decodificador e especificar os critérios de parada, ou outras operações necessárias.

Capítulo 5

Algoritmo Genético de Chaves Aleatórias Viciadas Aplicado ao Problema de Sequenciamento de Carros

Este capítulo tem por objetivo detalhar todas as etapas do BRKGA aplicado ao CSP. A primeira seção mostra detalhadamente como é formada uma instância do CSP. As demais seções descrevem o processo de decodificação, a variedade do conjunto elite, a utilização do *OpenMP*, os métodos de busca local implementados e a etapa de correção do cromossomo.

5.1 Instância do Problema de Sequenciamento de Carros

Cada instância é composta pelo número de carros a serem produzidos, pela quantidade de itens opcionais e classes, pela quantidade de carros a serem produzidos em cada classe e por uma matriz binária em que cada linha representa um carro e cada coluna representa um item opcional. O valor 1 indica que o carro requer o item opcional correspondente àquela coluna juntamente com as restrições de capacidade referente àquele item opcional.

No caso do CSP, o número de chaves aleatórias corresponde ao número de carros a serem produzidos e as características correspondem aos itens opcionais que o carro requer. Simbolicamente, cada chave aleatória representa um carro. O tamanho do cromossomo é dado pelo número de carros a serem produzidos.

É mostrada na Figura 5.1 uma instância do CSP composta pela produção de 8 carros, 5 itens opcionais, 4 classes, 5 restrições de capacidade e 2 modelos a serem produzidos de cada classe. As classes de carros 0 e 3 requerem o item opcional 1 e a restrição de capacidade correspondente é 3:5, o item opcional 2 é requerido apenas pela classe 1 com restrição de

capacidade 1:3. Já as classes 0 e 1 requerem o item opcional 3 com restrição 3:5. O item opcional 4 é requerido pelos carros das classes 2 e 3 com restrição 2:3 e por fim, a classe 2 requer o item opcional 5 com restrição de capacidade 2:4.

```
8 5 4
3 1 3 2 2
5 3 5 3 4
0 2 1 0 1 0 0
1 2 0 1 1 0 0
2 2 0 0 0 1 1
3 2 1 0 0 1 0
```

Figura 5.1: Exemplo de uma instância.

5.2 Decodificação

A decodificação é responsável por apresentar uma sequência de produção dos carros e seu valor é calculado pelo número de violações ocorridas. O processo é ilustrado pela Figura 5.2. No processo de decodificação, é criado um gabarito (a) que corresponde a uma sequência de carros que será permutada usando os índices dos vetores de chaves aleatórias após a ordenação dos mesmos. O decodificador recebe um cromossomo (b), em seguida este cromossomo é ordenado pelas chaves aleatórias (c) e por fim, é feita a permutação do gabarito com base nos índices do vetor, resultando em uma solução (d).

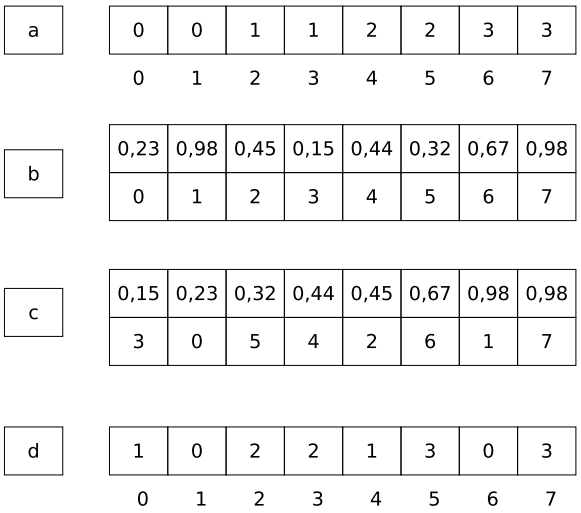

Figura 5.2: Exemplo de uma solução a partir do gabarito.

O próximo passo, é avaliar o valor da solução de acordo com o número de violações ocorridas. O processo de decodificação se repete para todos os cromossomos que compõem a população ao longo das gerações, até atingir o número máximo de gerações ou outro critério de parada. Em seguida, o melhor cromossomo é retornado juntamente com a sequência determinada através dele e é considerado o melhor valor para a função objetivo, pois contém a sequência com o menor número de violações.

5.3 Visão Geral

A Figura 5.3 apresenta a forma de execução de uma instância do CSP usando o BRKGA. Inicialmente, os dados da instância são lidos e armazenados em estruturas apropriadas e posteriormente são acessados pelo BRKGA em busca da solução ótima para o problema. A decodificação é feita para cada indivíduo e está descrita na Seção 5.2. Ao final de todas as gerações, é retornada a melhor solução encontrada assim como seu valor.

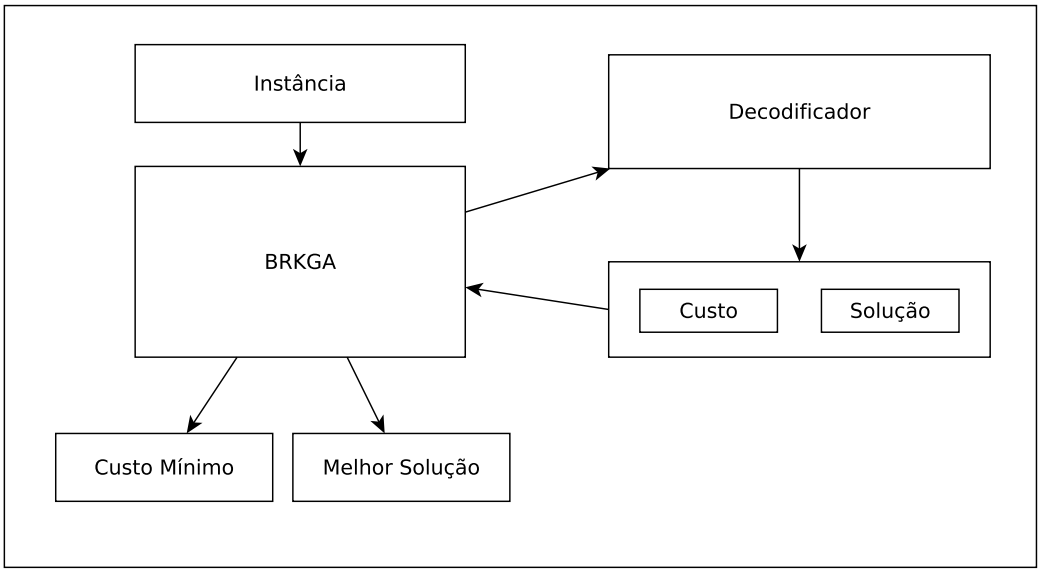

Figura 5.3: Exemplo da execução do BRKGA em uma instância do CSP.

5.4 Diversidade do Conjunto Elite

Como mencionado anteriormente, todos os elementos do conjunto elite são adicionados na população da próxima geração, porém, deve haver uma diversificação entre eles para evitar a convergência precoce da solução. Dessa maneira, são removidos os indivíduos que se assemelham entre si de acordo com uma determinada porcentagem. É usada uma memória auxiliar

para evitar a perda da melhor solução encontrada. Neste trabalho, foi considerada o valor da taxa de variação entre 10% e 60%.

5.5 Utilização do *OpenMP*

OpenMP trata-se de uma API portátil baseada no modelo de programação paralela de memória compartilhada para arquitetura de múltiplos processadores que surgiu no final de 1998 e foi disponibilizada para compiladores das linguagens C e C++. O uso do *openMP* permite a execução do programa com múltiplas *threads* e pode ser controlado pelo programador que deverá definir as partes a serem executadas em paralelo.

Neste trabalho, foi utilizada a API do BRKGA com o objetivo de resolver o CSP e o uso do *openMP* ocorreu sempre no processo de decodificação por demandar maior quantidade de dados para processamento. A API permite a manipulação das operações mais comuns relativas ao BRKGA, como o gerenciamento de populações e toda a dinâmica evolutiva. Em sistemas que utilizam OpenMP (2013), a API é capaz de codificar chaves em paralelo.

5.6 Operadores e Parâmetros Adaptativos

A convergência prematura é um problema que afeta a eficiência de algoritmos genéticos em geral. Isto ocorre principalmente pela falta de diversidade de indivíduos que compõem a população, fazendo com que a solução fique presa dentro de uma pequena região do espaço de busca, ou seja, um ótimo local. Algumas estratégias podem ser adotadas para contornar esta situação, tais como a aplicação de parâmetros adaptativos em tempo de execução. Os principais operadores adaptativos de um algoritmo genético são *crossover* e mutação, como citado por Laoufi et al. (2006).

Foram implementados no BRKGA parâmetros adaptativos para *crossover* e mutação. Uma vez que torna-se necessário a diversidade de elementos do conjunto elite, este conjunto pode reduzir sua porcentagem, havendo a necessidade de aumentar a porcentagem da taxa do operador de mutação para manter a variedade de indivíduos. Portanto, a porcentagem de indivíduos removidos do conjunto elite é adicionada à porcentagem do conjunto mutante.

Há a necessidade de alterar também a taxa de aplicação do operador de *crossover*, pois quando o conjunto elite for menor, a probabilidade do filho herdar suas características deve ser menor para evitar a replicação de indivíduos semelhantes. De forma análoga, quando houver muita variedade de indivíduos do conjunto elite, a probabilidade do filho herdar suas características deve ser maior. A fórmula usada para calcular a taxa do operador de *crossover*, de acordo com Laoufi et al. (2006) é apresentada pela Equação 5.1.

$$P_c = \begin{cases} K(f_{min} - f')/(\overline{f} - f_{min}) & f' \leq f_{min} \\ K & f' > f_{min} \end{cases} \tag{5.1}$$

Na referida Equação, K é o valor pré-definido da probabilidade de herança dos genes do pai elite no *crossover*, f_{min} é o valor mínimo de função objetivo da população corrente, f' é o melhor valor encontrado até o momento e $\overline{f}$ é o valor médio da população atual. Caso o valor de P_c fique maior que 1.0, então P_c passa a assumir o valor de K.

Outra estratégia comum para evitar convergência precoce em algoritmos genéticos é alterar o tamanho da população dinamicamente. Este tamanho deve diminuir de forma que a população elite não se torne muito pequena, pois ela é responsável por propagar parte dos indivíduos para a próxima geração. Quando os indivíduos apresentam características distintas, o tamanho da população deve se estender, possibilitando a inserção de novos indivíduos que possam gerar melhores resultados.

5.7 Métodos de Busca Local

De acordo com a dificuldade de cada problema, o BRKGA pode enfrentar problemas para convergir uma solução para um ótimo global. Dessa forma, modificações podem ser feitas para auxiliar no processo de evolução. Tais modificações envolvem o uso de buscas locais durante a fase de geração dos cromossomos e de evolução da população.

Após a aplicação dos métodos de busca local, é possível fazer uma correção no cromossomo recebido pelo decodificador para que ele corresponda à solução retornada pela busca local. São descritos nesta seção, os métodos de busca local utilizados para auxiliar o BRKGA no processo de geração e evolução de uma população. Durante a geração da população inicial, a busca local auxilia para que bons indivíduos sejam gerados ao invés de indivíduos aleatórios.

Na etapa de decodificação, a busca local é importante para uma convergência gradativa dos indivíduos, a taxa de aplicação de busca local pode variar de 30% a 60%. Tais métodos incluem *2-swap* e *best insertion* com duas versões. Todos os métodos são chamados durante o processo de decodificação, em que cada um recebe uma sequência de carros a serem produzidos e retornam uma nova configuração para esta sequência que possua um valor menor quando submetida à função de avaliação.

5.7.1 *2-Swap*

No método de busca local *2-Swap*, existe a troca entre dois elementos de um sequência, com o objetivo de obter um resultado melhor. Primeiramente, são gerados todos os possíveis pares de índices e armazenados em uma estrutura. Em seguida, há uma permutação nesta estrutura com o objetivo de causar uma perturbação na solução e por fim, os carros contidos em cada par de posições são trocados e o valor da solução é avaliado. Caso a solução tenha resultado melhor que a solução atual, esta passa a ser considerada então, a nova solução corrente e o processo de troca dos pares é reiniciado, permutando novamente o vetor de índices. O processo se repete até que nenhuma melhoria ocorra na função objetivo.

A Figura 5.4 ilustra uma aplicação do método *2-Swap* aplicado a uma sequência de carros, inicialmente formada pela sequência de carros $S = [0, 3, 1, 0, 2, 1, 3, 2]$. Suponhamos que a primeira e a quinta posições foram selecionadas para a troca. Após aplicação do método, o carro 2 passa a ocupar a posição do carro 0 e vice versa, gerando a nova sequência $S = [2, 3, 1, 0, 0, 1, 3, 2]$

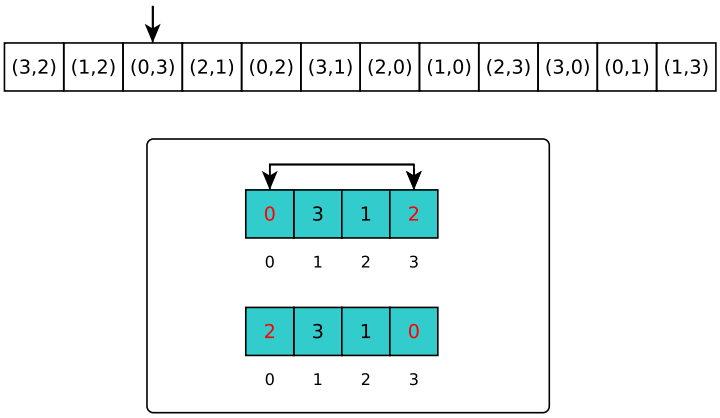

Figura 5.4: Exemplo de uma aplicação do método *2Swap*.

5.7.2 *Best Insertion*

De maneira semelhante ao método anterior, este método de busca local insere um mesmo elemento em cada posição de uma sequência para obter uma solução de melhor valor. Entretanto, todas as possibilidades de inserção são consideradas. Este método também pode ser aplicado tanto a uma solução completa quanto a uma solução parcial, de maneira construtiva.

As Figuras 5.5 a 5.10, ilustram a aplicação deste método, dividida em 6 passos. Considerando uma solução inicial $S = [0, 1, 3, 1, 2]$, na Figura 5.5 há a inserção do elemento 0, havendo apenas uma possibilidade de inserção. Na Figura 5.6, o carro 1 é inserido antes e depois do carro 0, mantendo-se apenas a inserção de menor valor.

Na Figura 5.7, o carro 3 é inserido em todas as posições, permanecendo na última posição. Outro carro da classe 1 é inserido na segunda posição como pode-se observar na Figura 5.8, dado o melhor valor obtido. O último carro a ser inserido é o carro 2 na segunda posição conforme a Figura 5.9, e por fim, na Figura 5.10 a solução $S = [1, 2, 1, 0, 3]$ é completa.

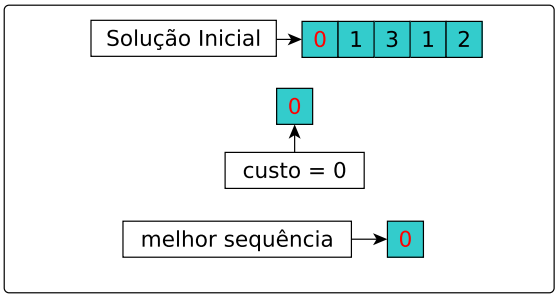

Figura 5.5: *Best Insertion*: Passo 1.

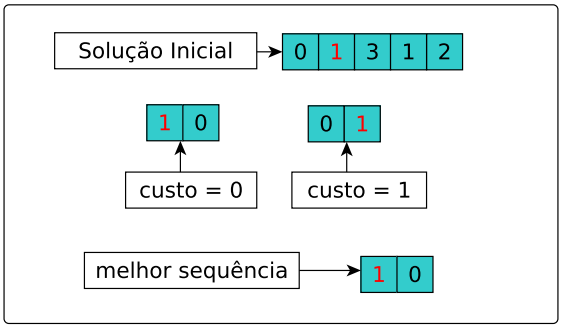

Figura 5.6: *Best Insertion*: Passo 2.

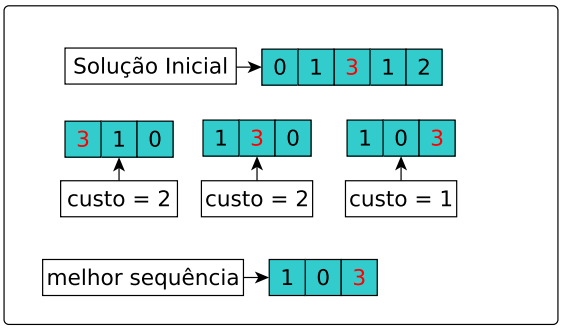

Figura 5.7: *Best Insertion*: Passo 3.

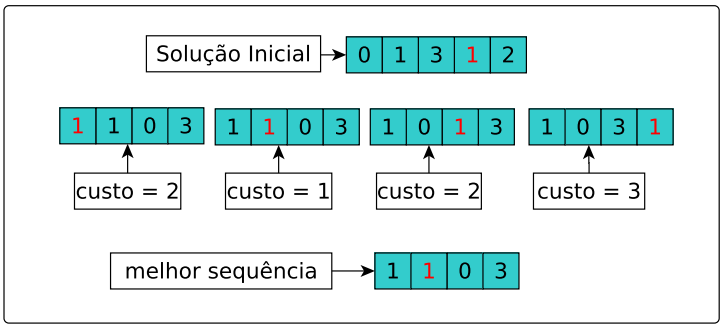

Figura 5.8: *Best Insertion*: Passo 4.

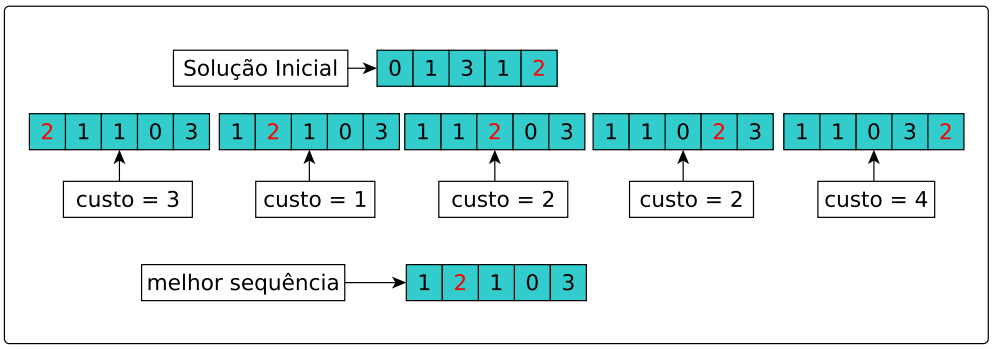

Figura 5.9: *Best Insertion*: Passo 5.

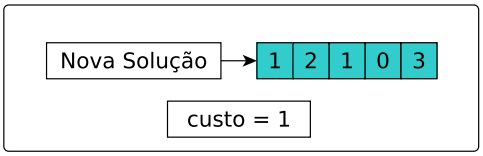

Figura 5.10: *Best Insertion*: Passo 6.

5.7.3 *Best Insertion First-Improvement*

Este método de busca local tem como objetivo inserir um mesmo elemento em cada posição de uma sequência se necessário, a fim de determinar a primeira posição que gera uma solução melhor do que a original. Este método pode ser aplicado tanto a uma solução completa quanto a uma solução parcial, de maneira construtiva. As Figuras 5.11 a 5.16 apresentam um exemplo de aplicação deste método.

No exemplo, uma solução é criada iterativamente pela inserção de elementos pelo método *Best Insertion First-Improvement* até que se obtenha uma solução completa. Na Figura 5.11, ocorre a inserção do primeiro carro, não havendo mais que uma possibilidade de inserção.

O processo de inserção do segundo carro é ilustrado pela Figura 5.12, em que o carro 1 foi inserido antes e após o carro 0, mantendo-se a segunda configuração.

A inserção do carro 3 ocorre na Figura 5.13. Nota-se que a inserção na última posição não foi analisada, dado que a segunda posição obteve um valor menor que inserido na primeira posição, portanto, o método termina para este elemento. Para o próximo carro Figura 5.14, ocorre a mesma situação, sendo a inserção realizada na terceira posição.

Por fim, o último carro é inserido na segunda posição Figura 5.15, uma vez que o valor da solução é menor do que o valor de inserção na posição anterior. Após a inserção de todos os carros Figura 5.16, a nova solução é retornada $S = [0, 2, 3, 1, 1]$ e obteve valor igual a 2, sendo este o número mínimo de violações das restrições de capacidade.

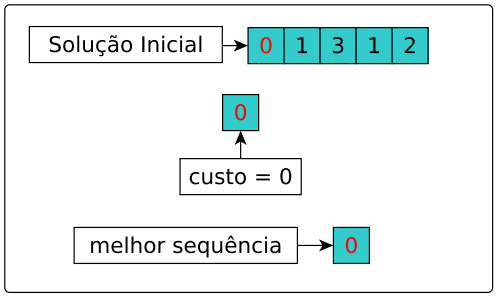

Figura 5.11: *Best Insertion First-Improvement*: Passo 1.

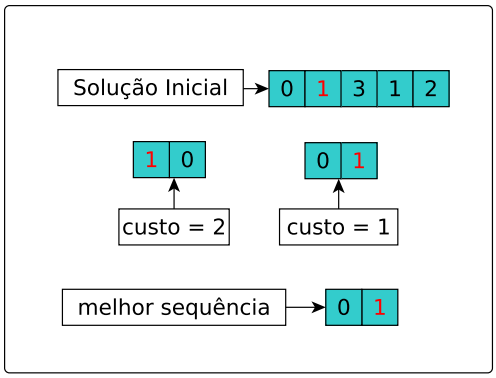

Figura 5.12: *Best Insertion First-Improvement*: Passo 2.

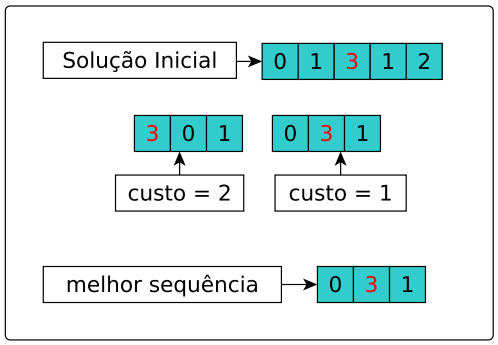

Figura 5.13: *Best Insertion First-Improvement*: Passo 3.

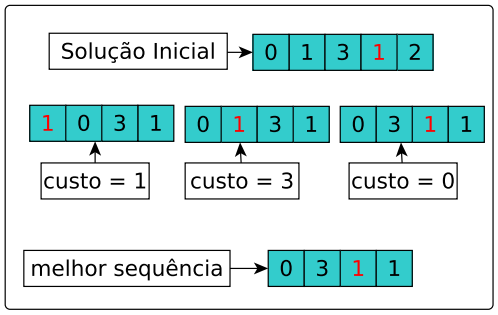

Figura 5.14: *Best Insertion First-Improvement*: Passo 4.

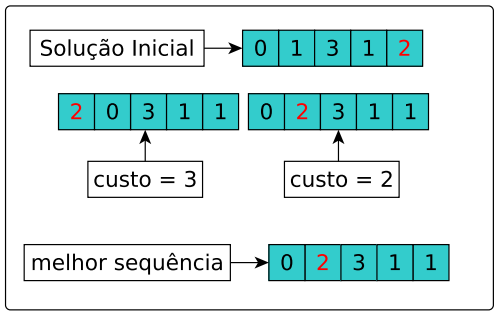

Figura 5.15: *Best Insertion First-Improvement*: Passo 5.

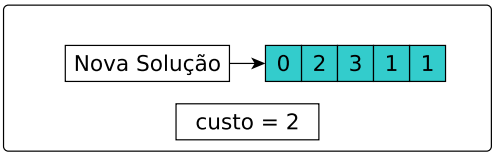

Figura 5.16: *Best Insertion First-Improvement*: Passo 6.

5.8 Correção do Cromossomo

A solução inicial utilizada pelas buscas locais é gerada a partir de um gabarito que associa índices a cada carro a ser produzido e um cromossomo composto por chaves aleatórias, do qual se obtém a sequência de índices de carros para produção. A Figura 5.17 apresenta o processo de obtenção de uma solução a partir de um cromossomo e posterior aplicação das buscas locais.

Inicialmente, um exemplo de gabarito e cromossomo podem ser observados respectivamente na Figura 5.17 em (a) e (b). Durante a decodificação do BRKGA, cada cromossomo possui suas chaves ordenadas (c), dando origem a uma permutação dos índices. Esta permutação de índices é verificada junto ao gabarito para geração de uma permutação dos carros a serem produzidos (d).

Por exemplo, observa-se em (c) que o primeiro índice do cromossomo ordenado é o 2, portanto, o carro correspondente no gabarito será inserido na primeira posição da solução inicial, neste caso, o carro 1. O próximo carro a ser inserido será o carro 0 na posição 0 do gabarito, o terceiro carro é o 2, pois ocupa a posição 5 e assim sucessivamente até que se obtenha uma solução completa todos os carros na solução.

Após a obtenção de uma solução completa, esta é submetida aos métodos de busca local, que alteram a permutação dos índices dos carros a serem produzidos (e). Entretanto, após alterada, a nova permutação não corresponde mais ao cromossomo gerado pelo BRKGA, sendo necessária uma etapa de correção (f) e (g), de maneira que o cromossomo passe a refletir novamente a solução gerada.

Para cada carro pertencente à solução obtida após a busca local, é necessário verificar qual é o índice correspondente no gabarito, como por exemplo, o carro 0 da solução obtida encontra-se na posição de índice 0. O segundo carro da solução é o carro de número 2 que ocupa a posição de índice 4, o terceiro carro 1 ocupa a posição do índice 2, já o outro carro 0 está na posição de índice 1 e assim, até completar todos os carros da solução retornada.

Por fim, cada índice em (f) é associado com os índices do cromossomo original (g), e consequentemente, as chaves passam então a ocupar aquela posição em ordem crescente. Como o primeiro índice de (f) é 0, então a chave de menor valor 0,13 é inserida na posição 0 do cromossomo. O segundo índice é o 4, portanto, a segunda chave a ser inserida é 0,22 na posição 4 do cromossomo original até obter o cromossomo corrigido, correspondendo dessa forma à

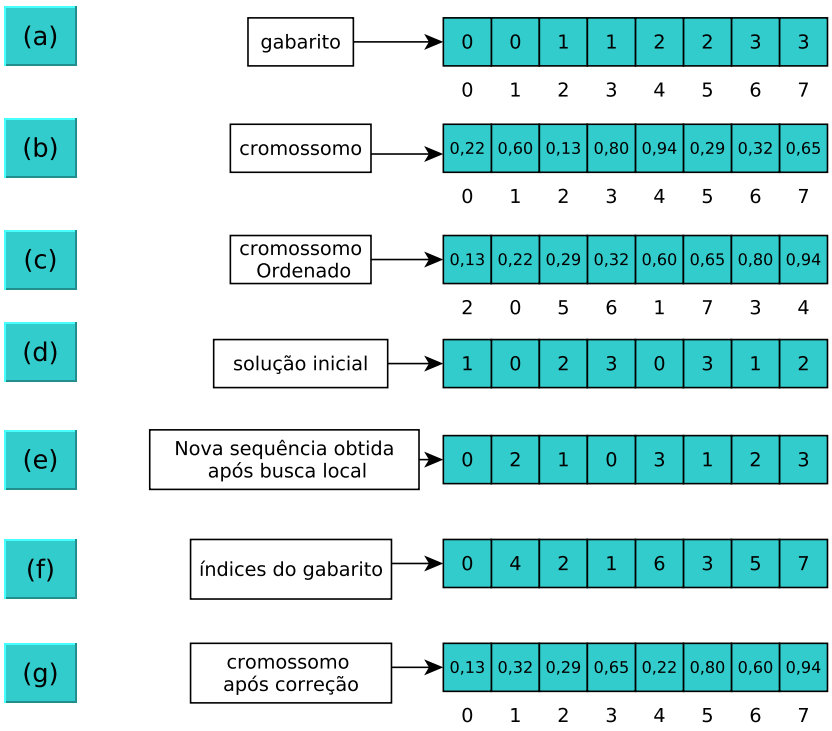

Figura 5.17: Correção de um cromossomo após uma busca local.

solução retornada pela busca local.

Capítulo 6

Experimentos Computacionais

Este capítulo tem como objetivo descrever todos os experimentos realizados, assim como o ambiente computacional utilizado, o ajuste dos parâmetros utilizados pelo BRKGA e os conjuntos de instâncias utilizadas como referência para fins comparativos.

6.1 Ambiente Computacional

Todos os testes foram executados em um computador com 16 GB de memória RAM, processador Intel Core™ i7-4790 CPU @ 3.60GHZ × 8 e o sistema operacional Ubuntu 16.04 LTS 64-bit. Os códigos foram implementados na linguagem C++ e compilados com a opção de otimização -O3.

6.2 Ajuste de Parâmetros

Como o BRKGA é um algoritmo evolutivo, é necessário estabelecer alguns parâmetros para executá-lo. Tais parâmetros envolvem o tamanho da população, quantidade de elementos que formam a população elite, a quantidade de mutantes, a probabilidade de um cromossomo filho herdar as características do cromossomo pai elite, o número de gerações, o tamanho da população elite, entre outros. Não é possível determinar a *priori* quais valores devem ser atribuídos a tais parâmetros, para isso, foi utilizada a ferramenta *irace* proposta por López-Ibáñez et al. (2016) versão 2.1.1662.

O *irace* recebe os parâmetros e o intervalo em que as variáveis deverão ser calibradas e efetua sucessivas execuções com base nas instâncias do CSP, alternando os valores dentro do intervalo de cada parâmetro. Ao fim da execução, os melhores valores a serem atribuídos a cada parâmetro são retornados. Na Tabela 6.1, verifica-se a descrição das variáveis juntamente com intervalo de valores considerados neste experimento. Nota-se que o tamanho da população total é dado por $pop = pop \times$ número de genes;

Tabela 6.1: Parâmetros para ajuste na ferramenta irace.

Descrição	Intervalo	Resultado
Tamanho total da População (*pop*)	[1..4]	3
Tamanho da população elite (*pe*)	[0,10..0,25]	0,11
Tamanho da população mutante (*pm*)	[0,10..0,30]	0,20
Número máximo de gerações (*MaxGens*)	[100..300]	184
Probabilidade do filho herdar o gene do pai elite (*rhoe*)	[0,50..0,80]	0,80
Taxa de diversificação da população elite (*Div*)	[0,10..0,60]	0,10
Taxa de aplicação de busca local na inicialização (*BL*)	[0,30..0,60]	0,50

6.3 Comparação de Resultados

Todas as instâncias utilizadas nos experimentos encontram-se disponíveis na biblioteca CSPLib (1999) e são divididas em 3 conjuntos: o primeiro é formado por 70 instâncias propostas por Lee et al. (1998), as denominadas *easy*, conhecidas pela facilidade em se obter a solução ótima. O segundo conjunto é composto por 30 instâncias propostas por Gravel et al. (2005) conhecidas como instâncias *hard*, e o último contém 9 tradicionais instâncias denominadas *harder* propostas por Gent e Walsh (1999).

As tabelas de comparação contêm dados de 10 execuções do BRKGA para cada instância e possuem um padrão para todos os conjuntos de instâncias: a primeira coluna indica qual instância está sendo analisada, a segunda contém o resultado obtido pelo melhor método da literatura, a terceira contém o valor médio obtido pelo método BRKGA (S) após as 10 execuções e a quarta contém o melhor resultado obtido pelo método (S^*). A quinta coluna contém o tempo de execução médio (T) em segundos e a última coluna contém a convergência média para cada instância, ou seja, em qual geração a melhor solução foi obtida. Nas seções a seguir, são apresentados os três conjuntos de instâncias: instâncias *easy*, instâncias *hard* e as instâncias *harder*.

6.3.1 Instâncias *easy*

O primeiro conjunto é formado por 70 instâncias com 200 carros em cada, 5 itens opcionais e de 17 a 30 classes de carros, todas satisfatíveis, ou seja, é possível obter solução ótima em todas elas. Este conjunto é dividido em 7 subconjuntos com 10 instâncias, dispostos de acordo com sua taxa de utilização na ordem 60%, 65%, 70%, 75%, 80%, 85% e 90%. De acordo com Gottlieb et al. (2003), a dificuldade de resolução de uma instância é dada pelo número de carros a serem produzidos, pelo número das configurações dos diferentes itens opcionais e também sobre a taxa de utilização.

A taxa de utilização de um item opcional o_i dada pela Equação 6.1 corresponde ao número de carros que exigem o opcional o_i em uma subsequência de tamanho $q(o_i)$ em relação ao

número máximo de carros que poderiam requerer o item opcional o_i nesta subsequência. Uma taxa de utilização superior a 1 indica que a demanda é maior que a capacidade, portanto, não poderá ser atendida pela estação de trabalho disponível. Já uma taxa de utilização próxima de 0, indica que a procura é muito baixa em relação à capacidade da estação de trabalho.

$$Taxa\ de\ Utilização(O_i) = \frac{r(C, o_i).q(o_i)}{|C|.p(o_i)} \tag{6.1}$$

A Tabela 6.2 apresenta as instâncias com taxas de utilização de 60% e 65%. Os melhores resultados para estas instâncias foram obtidos pela metaheurística *Ant Colony Optimization* (ACO) proposta por Gottlieb et al. (2003) e *Very Fast Local Search* (VFLS) de Estellon et al. (2006). O valor médio referente ao número de violações e desvio padrão das instâncias com taxas de utilização de 60% foram iguais a 0, o tempo de execução médio e desvio padrão foram de 19,7 segundos e 0,2 respectivamente.

Já o valor médio e desvio padrão para as instâncias com taxa de utilização de 65%, foram de 0,03 violações e 0,09, respectivamente, enquanto o tempo de execução médio foi de 23,29 segundos com desvio padrão de 7,71. Conclui-se que a convergência média ocorre com maior frequência na primeira geração e a solução apresenta valor igual a 0, porque a taxa de utilização é baixa.

Tabela 6.2: Instâncias propostas por Lee et al. (1998) com taxas de utilização de 60% e 65%.

Instância	Literatura	S	S^*	T	Convergência
60-01	0	0,0	0	19,8	1,0
60-02	0	0,0	0	19,9	1,0
60-03	0	0,0	0	19,9	1,0
60-04	0	0,0	0	19,6	1,0
60-05	0	0,0	0	19,7	1,0
60-06	0	0,0	0	19,8	1,0
60-07	0	0,0	0	19,2	1,0
60-08	0	0,0	0	19,7	1,0
60-09	0	0,0	0	19,7	1,0
60-10	0	0,0	0	19,7	1,0
65-01	0	0,0	0	19,72	1,0
65-02	0	0,0	0	27,30	1,0
65-03	0	0,0	0	20,03	1,0
65-04	0	0,3	0	45,49	5,6
65-05	0	0,0	0	20,17	1,0
65-06	0	0,0	0	20,25	1,0
65-07	0	0,0	0	19,72	1,0
65-08	0	0,0	0	20,12	1,0
65-09	0	0,0	0	20,20	1,0
65-10	0	0,0	0	19,92	1,0

A Tabela 6.3 apresenta as instâncias com taxas de utilização de 70% e 75%. É possível

perceber que há uma leve oscilação no custo médio das soluções, pois a taxa de utilização é maior para este subconjunto. O valor médio e desvio padrão para as instâncias com 70% de taxa de utilização foram de 0,07 violações e 0,18, respectivamente e o tempo de execução médio e desvio padrão foram de 26,67 segundos e 13,57. O valor médio igual a 0,14 violações e desvio padrão de 0,31 com tempo de execução médio de 34,32 segundos e desvio padrão 26,08 foram obtidos das instâncias com 75% de taxa de utilização.

Tabela 6.3: Instâncias propostas por Lee et al. (1998) com taxas de utilização de 70% e 75%.

Instância	Literatura	S	S^*	T	Convergência
70-01	0	0,0	0	20,2	1,0
70-02	0	0,1	0	40,4	1,0
70-03	0	0,0	0	20,2	1,0
70-04	0	0,6	0	63,2	7,0
70-05	0	0,0	0	20,3	1,0
70-06	0	0,0	0	20,3	1,0
70-07	0	0,0	0	20,1	1,0
70-08	0	0,0	0	20,5	1,0
70-09	0	0,0	0	20,7	1,0
70-10	0	0,0	0	20,4	1,0
75-01	0	0,0	0	20,5	1,0
75-02	0	0,4	0	88,9	32,0
75-03	0	0,0	0	20,4	1,0
75-04	0	1,0	0	83,7	25,7
75-05	0	0,0	0	20,5	1,0
75-06	0	0,0	0	22,5	1,0
75-07	0	0,0	0	20,4	1,0
75-08	0	0,0	0	24,7	1,0
75-09	0	0,0	0	20,6	1,0
75-10	0	0,0	0	20,5	1,0

Na Tabela 6.4 encontram-se os resultados para as instâncias com taxas de utilização de 80% e 85%. Percebe-se que à medida que a taxa de utilização aumenta, a dificuldade de obtenção da solução ótima também aumenta. A média de convergência é maior comparada às instâncias com taxa de utilização menor.

Em relação ao valor médio da soluções das instâncias com 80% de taxa de utilização, o resultado foi de 0,98 violações com 1,19 de desvio padrão. Já o tempo de execução médio foi igual a 63,54 segundos e desvio padrão igual a 21,91. Para o conjunto com 85% de taxa, a solução média e o desvio padrão foram de 1,62 violações e 1,66, respectivamente. O tempo de execução médio é 73,47 segundos com desvio padrão igual a 34,41. O BRKGA conseguiu resultado igual ao melhor método da literatura em 12 de 20 instâncias, perdendo desempenho em 8 das 20 instâncias.

Tabela 6.4: Instâncias propostas por Lee et al. (1998) com taxas de utilização de 80% e 85%.

Instância	Literatura	S	S^*	T	Convergência
80-01	0	0,0	0	26,1	1,0
80-02	0	0,0	0	59,0	14,8
80-03	0	0,0	0	20,7	1,0
80-04	0	3,4	2	80,6	2,9
80-05	0	0,0	0	56,3	2,5
80-06	0	2,9	2	81,4	2,1
80-07	0	0,3	0	83,7	39,8
80-08	0	1,1	1	79,7	3,0
80-09	0	1,4	1	76,0	12,5
80-10	0	0,7	0	71,4	1,7
85-01	0	0,5	0	64,1	2,0
85-02	0	1,2	0	73,3	18,6
85-03	0	0,0	0	20,9	1,0
85-04	0	2,9	2	82,6	20,4
85-05	0	2,4	1	107,0	17,1
85-06	0	0,6	0	69,1	3,2
85-07	0	0,6	0	69,0	7,7
85-08	0	2,4	1	84,0	3,9
85-09	0	5,6	4	143,2	30,7
85-10	0	0,0	0	20,9	1,0

A Tabela 6.5 apresenta os resultados para cada instância com taxa de utilização de 90%. Este conjunto é o que mais teve dificuldade em busca da solução. Percebe-se um alto valor na média das convergências e números de violações elevados. O valor médio para este conjunto foi de 5,4 violações e desvio padrão igual a 5,3. O tempo de execução médio foi de 156 segundos e desvio padrão igual a 79,93.

Tabela 6.5: Instâncias propostas por Lee et al. (1998) com taxas de utilização de 90%.

Instância	Literatura	S	S^*	T	Convergência
90-01	0	7,8	6	235,7	62,5
90-02	0	1,9	1	122,1	20,3
90-03	0	6,1	3	177,7	57,7
90-04	0	0,3	0	60,5	13,8
90-05	0	18,2	14	244,9	77,0
90-06	0	0,0	0	20,8	1,0
90-07	0	9,8	8	239,7	64,0
90-08	0	6,3	3	236,8	86,1
90-09	0	0,3	0	72,0	2,1
90-10	0	3,3	2	149,6	45,6

Apesar deste conjunto possuir instâncias de fácil solução, diferentemente dos métodos ACO e VFLS, o método proposto obteve solução ótima em apenas 3 de 10 instâncias. Percebe-se

então, que as instâncias com menor taxa de utilização por itens opcionais foram mais fáceis de se obter solução ótima, mas à medida em que a taxa de utilização cresce, o número de violações também cresce, como observado nas instâncias com taxas de utilização superior a 80%.

6.3.2 Instâncias *hard*

Este conjunto é composto por 30 instâncias conhecidas como *hard*, e é dividido em três subconjuntos com 10 instâncias cada. O primeiro subconjunto é formado por 200 carros, o segundo por 300 e o último por 400 carros. Cada instância tem 5 itens opcionais e possui de 19 a 26 classes de carros.

Os métodos *Very Fast Local Search* (VFLS) de Estellon et al. (2006), as codificações da Forma Normal Conjuntiva (CNF) propostas por Mayer-Eichberger (2013), a *Iterated Beam Search* (IBS) proposta por Golle et al. (2014) e a metaheurística *Ant Colony Optimization* (ACO) de Gottlieb et al. (2003), obtiveram os melhores resultados entre todos os métodos presentes na literatura que consideraram estas instâncias. A Tabela 6.6 apresenta as instâncias com 200, 300 e 400 carros.

Tabela 6.6: Instâncias *hard* propostas por Gravel et al. (2005).

Instância	Literatura	S	S*	T	Convergência
Pb-200-01	0	40,9	40	245,2	68,0
Pb-200-02	2	28,7	27	239,4	83,6
Pb-200-03	3	43,6	41	245,1	61,5
Pb-200-04	7	45,2	42	242,3	45,2
Pb-200-05	3	26,3	24	236,6	52,1
Pb-200-06	6	31,9	31	243,0	76,8
Pb-200-07	0	28,9	27	243,7	80,7
Pb-200-08	8	23,4	22	239,8	80,1
Pb-200-09	10	33,8	31	243,9	84,4
Pb-200-10	17	31,4	30	233,3	63,8
Pb-300-01	0	49,5	47	1.159,7	101,2
Pb-300-02	6	50,2	45	1.166,1	74,6
Pb-300-03	13	57,6	51	1.176,1	84,9
Pb-300-04	7	53,2	48	1.156,7	76,9
Pb-300-05	16	84,9	58	1.192,2	102,3
Pb-300-06	2	57,9	56	1.178,9	96,8
Pb-300-07	0	54,9	48	1.176,3	73,7
Pb-300-08	8	48,13	46	1.165,5	52,5
Pb-300-09	7	52,92	50	1.177,0	74,0
Pb-300-10	13	58,97	54	1.180,2	102,0
Pb-400-01	1	65,43	62	3.547,9	47,2
Pb-400-02	15	83,13	59	3.566,5	69,6
Pb-400-03	9	60,47	56	3.479,2	77,5
Pb-400-04	19	67,33	54	3.538,6	91,1
Pb-400-05	0	55,15	51	3.514,3	68,5
Pb-400-06	0	56,87	54	3.540,9	77,8
Pb-400-07	4	63,45	62	3.534,1	79,5
Pb-400-08	4	68,45	66	3.592,6	93,3
Pb-400-09	5	74,35	72	3.613,2	84,4
Pb-400-10	0	51,70	48	3.572,2	60,0

Neste conjunto, o BRKGA apresentou maior dificuldade em busca pela solução ótima, não alcançando resultado igual ao melhor método da literatura em nenhuma instância. Tal conjunto demanda maior tempo de execução, atinge maior número para convergência e no entanto, os valores obtidos quando comparados ao melhor método são altos, não obtendo solução ótima em nenhuma instância. O valor médio obtido pelas instâncias com 200 carros foi de 33,4 violações e desvio padrão de 7,0. O tempo de execução médio foi de aproximadamente 241,2 segundos (4 min) e desvio padrão de 3,7.

Para o conjunto com 300 carros, o valor médio ficou em torno de 56,8 violações com desvio padrão de 10,0 e tempo de execução médio de 1.172,9 segundos (19 minutos e meio) com desvio padrão de 10,2. Para o conjunto com 400 carros, o valor médio obtido foi igual a 64,6 e

desvio padrão de 8,9 e tempo de execução médio de 3.550 (59 minutos) segundos com desvio padrão igual a 36,5.

6.3.3 Instâncias *harder*

O último conjunto apresentado na Tabela 6.7, é composto por 9 tradicionais instâncias propostas por Gent e Walsh (1999), denominadas *harder* pela dificuldade em se obter as soluções ótimas. Cada instância é formada por 100 carros, 5 itens opcionais e de 19 a 26 classes de carros.

A Tabela 6.7 apresenta os resultados para este conjunto de instâncias. Os métodos *Very Fast Local Search* (VFLS) proposto por Estellon et al. (2006), as codificações da Forma Normal Conjuntiva (CNF) de Mayer-Eichberger (2013) e a metaheurística *Ant Colony Optimization* (ACO) de Gottlieb et al. (2003) obtiveram os melhores resultados para este conjunto de instâncias e são apresentados na referida tabela.

Tabela 6.7: Instâncias propostas por Gent e Walsh (1999).

Instância	Literatura	S	S^*	T	Convergência
4/72	0	14,5	13	19,8	64,7
6/76	6	8,8	7	20,1	50,6
10/93	3	25,2	23	23,3	65,0
16/81	0	19,6	19	21,0	76,6
19/71	2	18,0	17	21,6	87,1
21/90	2	12,4	12	22,7	57,3
26/82	0	12,4	10	19,2	48,7
36/92	1	15,4	14	26,0	101,8
41/66	0	5,7	5	20,0	46,1

O valor médio obtido neste conjunto foi de 14,6 violações com desvio padrão de 5,5. O conjunto obteve um tempo de execução médio de 21,5 segundos com desvio padrão igual a 2,04. Apesar do baixo tempo de execução, o método proposto não encontrou solução ótima em nenhuma solução devido à alta taxa de utilização por itens opcionais presentes nas instâncias.

6.4 Análise Geral

Após a realização dos experimentos, realiza-se a análise dos dados de acordo com critérios de desempenho. O sumário dos resultados encontra-se na Tabela 6.8. Na primeira coluna, há a descrição de cada critério considerado e na segunda coluna, o valor correspondente.

Tabela 6.8: Sumário dos resultados.

Tempo de execução máximo	60 min
Tempo de execução médio	8,2 min
Número médio de violações	16,1
Número de soluções sem violações	55
Número de resultados iguais aos da literatura	55

Os resultados obtidos não são satisfatórios. Para as instâncias com taxa de utilização menor, o método proposto consegue encontrar solução ótima em aproximadamente 20 segundos e à medida que a dificuldade aumenta, o tempo de execução e número de violações também aumentam. A solução ótima foi encontrada em 55 das 109 instâncias consideradas nos experimentos disponíveis na CSPLib (1999). Portanto, o método não é robusto para solução do CSP, pois não consegue obter soluções ótimas para as instâncias com maior dificuldade de solução.

Capítulo 7

Conclusões

Este trabalho considera parte do processo de produção de carros em uma indústria, denominada Problema de Sequenciamento de Carros – *Car Sequencing Problem* (CSP). Este problema foi caracterizado como $\mathcal{NP}$-Difícil e é abordado pelas comunidades de Pesquisa Operacional. Neste trabalho, o CSP foi tratado por meio de um método evolutivo denominado Algoritmo Genético de Chaves Aleatórias Viciadas. Este método foi descrito em detalhes, adaptado para utilização no problema tratado e teve seus parâmetros ajustados em experimentos preliminares.

Foram implementados os métodos de busca local *2-swap* e *best insertion*, sendo o último implementado em duas versões e operadores adaptativos para refinamento da metaheurística proposta. Os experimentos computacionais foram realizados considerando três conjuntos de instâncias disponíveis na literatura, utilizados amplamente. Estes conjuntos diferem entre si pelas dimensões e também pela dificuldade em solucioná-los. Após a realização dos experimentos, concluiu-se que o método proposto obteve resultados satisfatórios em 55 das 109 instâncias consideradas.

Porém, nas instâncias que exigem maior esforço para encontrar a solução ótima, em que a taxa de utilização de opcionais é maior, o método não obteve bom desempenho comparado aos resultados obtidos por outros métodos da literatura. O tempo de execução para cada instância varia conforme a quantidade de carros e o nível de dificuldade, sendo que a mais demorada gastou em média uma hora pra ser solucionada. Apesar de parecer alto, este tempo de execução é aceitável, visto que as indústrias de produção de carros organizam o plano de produção com pelo menos um dia de antecedência.

Como trabalhos futuros, a proposta é fazer um estudo para entender o motivo pelo qual o método não obteve melhores resultados, pesquisando e propondo a implementação de novos métodos de refinamento para o BRKGA. Fazer uma análise de desempenho após a implementação de cada melhoria e manter a que obtiver melhor resultado. Utilizar outras estratégias, como modificar a forma de calibragem dos parâmetros e aumentar o número de gerações para execução do algoritmo e ao fim, testar o método com as melhorias propostas, de acordo com

o conjunto de instâncias disponíveis na literatura, fazendo a comparação com os melhores métodos da literatura.

Referências Bibliográficas

Anuário da Indústria Automobilística Brasileira, A. . (2016). Anuário da indústria automobilística brasileira | brazilian automotive industry yearbook - 2016. http://www.virapagina.com.br/anfavea2016/. Acessado: 10/12/2016.

Barbara Smith, M. (1999). CSPLib problem 001: Car sequencing. http://www.csplib.org/Problems/prob001. Acessado: 15/12/2016.

Bautista, J.; Pereira, J. e Adenso-Diaz, B. (2008a). A GRASP approach for the extended car sequencing problem. *Journal of Scheduling*.

Bautista, J.; Pereira, J. e Adenso-Díaz, B. (2008b). A beam search approach for the optimization version of the car sequencing problem. *Annals of Operations Research*, 159(1):233–244.

Boysen, N.; Golle, U. e Rothlauf, F. (2011). The car resequencing problem with pull-off tables. *BuR-Business Research*, 4(2):276–292.

CSPLib (1999). http://www.csplib.org/Problems/prob001/. Acessado: 22/12/2016.

Darwin, R. (1859). Charles, on the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life.

Drexl, A. e Kimms, A. (2001). Sequencing jit mixed-model assembly lines under station-load and part-usage constraints. *Management Science*, 47(3):480–491.

Drexl, A.; Kimms, A. e Matthießen, L. (2006). Algorithms for the car sequencing and the level scheduling problem. *Journal of Scheduling*.

Estellon, B.; Gardi, F. e Nouioua, K. (2006). Large neighborhood improvements for solving car sequencing problems. *RAIRO - Operations Research*.

Estellon, B.; Gardi, F. e Nouioua, K. (2008). Two local search approaches for solving real-life car sequencing problems. *European Journal of Operational Research*, 191(3):928–944.

Fliedner, M. e Boysen, N. (2008). Solving the car sequencing problem via Branch & Bound. *European Journal of Operational Research*.

Gent, I. P. e Walsh, T. (1999). Csplib: a benchmark library for constraints. In *International Conference on Principles and Practice of Constraint Programming*, pp. 480–481. Springer.

Golle, U.; Rothlauf, F. e Boysen, N. (2014). Iterative beam search for car sequencing. *Annals of Operations Research*.

Gonçalves, J. F. e Resende, M. G. (2011). Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, 17(5):487–525.

Gottlieb, J.; Puchta, M. e Solnon, C. (2003). A study of greedy, local search, and ant colony optimization approaches for car sequencing problems. In *Workshops on Applications of Evolutionary Computation*, pp. 246–257. Springer.

Gravel, M.; Gagne, C. e Price, W. (2005). Review and comparison of three methods for the solution of the car sequencing problem. *Journal of the Operational Research Society*, 56:1287–1295.

Gunay, E. E. e Kula, U. (2017). A stochastic programming model for resequencing buffer content optimisation in mixed-model assembly lines. *International Journal of Production Research*, 55(10):2897–2912.

Holland, J. H. (1975). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press.

Kis, T. (2004). On the complexity of the car sequencing problem. *Operations Research Letters*, 32(4):331–335.

Laoufi, A.; Hadjeri, S. e Hazzab, A. (2006). Adaptive probabilities of crossover and mutation in genetic algorithms for power economic dispatch. *International Journal of Applied Engineering Research*, 1(3):393–408.

Lee, J. H.-M.; Leung, H.-F. e Won, H.-W. (1998). Performance of a comprehensive and efficient constraint library based on local search. In *Australian Joint Conference on Artificial Intelligence*, pp. 191–202. Springer.

López-Ibáñez, M.; Dubois-Lacoste, J.; Cáceres, L. P.; Birattari, M. e Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

Matsumoto, M. e Nishimura, T. (1998). Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30.

Mayer-Eichberger, Valentin e Walsh, T. (2013). Sat encodings for the car sequencing problem. In *POS@ SAT*, pp. 15–27.

Nguyen, A. (2005). Challenge roadef 2005: Car sequencing problem. *Online reference at http://www. prism. uvsq. fr/˜ vdc/ROADEF/CHALLENGES/2005/, last visited on March*, 23.

OpenMP (2013). http://www.openmp.org/. Acessado: 08/01/2017.

Parrello, B. D.; Kabat, W. C. e Wos, L. (1986). Job-Shop Scheduling Using Automated Reasoning: A Case Study of the Car-Sequencing Problem. *Journal of Automated Reasoning*, 2:1–42.

Perron, L. e Shaw, P. (2004). Combining forces to solve the car sequencing problem. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pp. 225–239. Springer.

Puchta, M. e Gottlieb, J. (2002). Solving car sequencing problems by local optimization. In *Workshops on Applications of Evolutionary Computation*, pp. 132–142. Springer.

Solnon, C. (2008). Combining two pheromone structures for solving the car sequencing problem with ant colony optimization. *European Journal of Operational Research*, 191(3):1043–1055.

Solnon, C.; Cung, V. D.; Nguyen, A. e Artigues, C. (2008). The car sequencing problem: Overview of state-of-the-art methods and industrial case-study of the roadef'2005 challenge problem. *European Journal of Operational Research*, 191(3):912–927.

Spears, W. M. e De Jong, K. D. (1995). On the virtues of parameterized uniform crossover. Technical report, DTIC Document.

Toso, R. F. e Resende, M. G. (2015). A c++ application programming interface for biased random-key genetic algorithms. *Optimization Methods and Software*, 30(1):81–93.

Yavuz, M. (2013). Iterated beam search for the combined car sequencing and level scheduling problem. *International Journal of Production Research*, 51(12):3698–3718.