GEOVANE APARECIDO RIBEIRO

Orientador: Marco Antonio Moreira de Carvalho

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

Ouro Preto

Março 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 AUTOMOTIVA

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
Março de 2017

UNIVERSIDADE FEDERAL DE OURO PRETO

FOLHA DE APROVAÇÃO

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

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, Março 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 a mesma opção para que as estações de trabalho tenham a capacidade de atendê-los, dando origem a restrições de capacidade que devem ser respeitadas. Estas restrições indicam que em uma determinada sequência, não deve haver mais carros requerendo uma mesma opção 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 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. Os experimentos computacionais consideraram 109 instâncias disponíveis na literatura e são reportados os resultados obtidos através de experimentos iniciais, além de uma comparação com os melhores resultados disponíveis na literatura.

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)to solve the CSP. Computational experiments considered 109 benchmark instances available in literature and the results obtained are reported.

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 e Margarete 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 carros automóveis. De acordo com o **?**, 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 por causa da Segunda Guerra Mundial. Surgiram 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, iniciou-se então a construção das 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 **?**, 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 de trabalho específica. Estes carros não necessariamente são iguais, embora possam ser do mesmo modelo, portanto, cada carro pertence a uma classe que requer um conjunto de opções distintas. Por exemplo, um carro pode demandar teto solar enquanto outro requer som de fábrica, um outro 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, 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 a mesma opção. Cada opção é instalada 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 podem ser representadas pela razão p_j/q_j, indicando que não mais que p_j carros que exigem a opção 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 opcionais sem haver sobrecarga das estações de trabalho. Surge então, o Problema de Sequenciamento de Carros – *Car Sequencing Problem* (CSP) proposto por **?** e posteriormente abordado pelas comunidades de Pesquisa Operacional. O CSP é abordado tanto como um problema de decisão como de otimização, sendo que o foco deste trabalho é a versão de otimização.

De acordo com **?**, o CSP é caracterizado como um problema $\mathcal{NP}$-difícil, implicando que não se conhece algoritmo eficiente para sua solução, ou seja, algoritmos que gerem 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 **?**, em que 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 adaptadar às condições de sobrevivência no ambiente. Uma classe de algoritmos evolutivos muito utilizados atualmente são os algoritmos genéticos propostos por **?**. Este trabalho propõe resolver o CSP através de um recente tipo de algoritmo genético denominado Algoritmo Genético de Chaves Aleatórias Viciadas – *Biased Random-Key Genetic Algorithm* (BRKGA).

1.1 Motivação

Empresas de desenvolvimento e fabrição de veículos buscam formas de produção sem sobrecarga na produtividade e com o menor custo 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 **?**, 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 métodos heurísticos e meta-heurísticos, avaliando-os 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:

- Elaborar heurísticas consistentes e robustas que possam ser utilizadas no contexto do problema do sequenciamento de carros que permitam a obtenção rápida de soluções próximas da solução ótima sem que se perca a vantagem da busca sistemática inicialmente considerando problemas específicos, mas com uma possibilidade de generalização;
- Buscar a aplicação prática dos métodos desenvolvidos em contextos reais, a fim de que também seja constituído um avanço para as indústrias nacionais.

1.3 Organização do Trabalho

O restante do trabalho está organizado da seguinte forma: no Capítulo 2, encontra-se a revisão da literatura, que descreve os trabalhos mais relevantes em duas partes. Uma delas é a versão de decisão, em que o objetivo é avaliar um problema e informar se o mesmo possui solução ou não. A outra, é a versão de otimização, em que vários autores propuseram métodos para obterem soluções ótimas, baseando seus resultados através de um conjunto de instâncias disponíveis a este fim. O Capítulo 3 mostra a fundamentação teórica necessária para o entendimento do CSP. No Capítulo 4 há o desenvolvimento, em que é descrito o algoritmo genético de chaves aleatórias viciadas aplicado ao CSP, no Capítulo 5 são relatados os experimentos computacionais. No capítulo 6 há o planejamento para as atividades restantes e por fim, a conclusão é feita no Capítulo 7.

Capítulo 2

Revisão da Literatura

Neste capítulo, serão apresentados os trabalhos mais relevantes presentes na literatura sobre o CSP até o momento, alguns focados na versão de decisão enquanto outros na versão de otimização do problema. 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 dito anteriormente, o principal foco desta abordagem é a versão de otimização, que visa otimizar o tempo total de processamento ou o número de violações de restrições de capacidade, minimizando os custos das sequências.

Uma biblioteca de problemas de teste para solucionadores de restrições denominada CSPlib foi criada por **?**. 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 **?**. Várias instâncias do CSP encontram-se disponíveis nesta biblioteca.

2.1 Versão de Otimização

O primeiro artigo que descreveu o CSP foi **?**, 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 **?**, 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 **?**. 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 **?**. 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 se mostrou muito eficiente obtendo ótimos resultados de acordo com instâncias disponíveis na CSPlib.

O CSP foi considerado no ROADEF'2005 *Challenge* **?**. 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. Entretanto, 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 **?**, 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 **?**, inicialmente propôs um método de busca em vizinhança em larga escala, obtendo resultados limitados em um tempo de 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 **?** 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 **?**.

Nesta abordagem, comparava-se o tempo 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 Escala foi proposto por **?**. 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* (**?**) 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 **?** 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 **?**.

Uma extensão do Problema de Sequenciamento de Carros foi desenvolvida por **?**. 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, a opção 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* – 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 **?**. 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 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 **?**, 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 **?**, 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 **?**. 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 **?** e outras 18 foram obtidas através do gerador de Drexl.

Três tipos de codificações da Forma Normal Conjuntiva (FNC) foram estudadas por **?**, 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 **?**.

Novamente, foi implementada uma IBS por **?**. 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 **?**, resolvendo 15 de 18 instâncias disponíveis na CSPlib. Dos métodos exatos, este algoritmo é considerado o estado da arte da versão de otimização.

2.2 Versão de Decisão

Foi formulado por **?** um algoritmo de filtragem baseado em *Global Sequencing Constraint* - Restrição de Sequenciamento Global em cinco instâncias fornecidas por um grande fabricante de automóveis. O algoritmo é eficiente sendo capaz de resolver tais instâncias com poucos *backtracks*.

De acordo com **?**, o CSP pertence à classe $\mathcal{NP}$-Completo, através da redução ao problema

de ciclo hamiltoniano. É apontado neste trabalho que o algoritmo de filtragem proposto por **?**, não gera soluções ótimas.

Uma abordagem por *Constraint Handling Prolog* foi apresentada por **?**. A ideia é combinar programação lógica com técnicas de programação por restrições. O problema pode ser dividido em dois subproblemas. No primeiro, todas as restrições são computadas com um padrão *Prolog*. No segundo, há uma busca por uma solução viável. Através dessa abordagem, é possível efetuar podas eficientes no espaço de busca, portanto, a abordagem se mostrou eficiente, exigindo um baixo tempo de execução para instâncias geradas aleatoriemente. Numa sequência com menos de 50 carros, o tempo foi insignificante, para 50 carros, o tempo foi de 15 segundos, para 100 carros, 1 minuto e para 200 carros, o tempo foi de aproximadamente 5 minutos.

Abordagens baseadas em métodos de busca local e colônia de formigas foram utilizadas por **?**. A solução é modificada pela busca local de forma aleatória por diferentes movimentos. A solução atual é comparada à solução candidata atual, que é aceita se não piorar a solução. São seis os tipos de movimentos:

- *Inserção*, em que um carro é movido de uma posição para outra numa sequência;
- *Swap*, troca de dois carros dentro de uma subsequência;
- *SwapS*, efetua a troca entre dois carros que são semelhantes mas não necessitam das mesmas opções;
- *SwapT*, caso especial em que há a troca entre dois carros vizinhos;
- *Lin2Opt*, responsável por inverter a subsequência de carros definida em um intervalo;
- *Random*, que embaralha aleatoriamente uma subsequência.

? trouxe duas contribuições relevantes. Em primeiro lugar, uma prova que mostra facilmente que o CSP é $\mathcal{NP}$-difícil e em segundo lugar, um algoritmo pseudo-polinomial baseado em programação dinâmica. O algoritmo foi testado usando instâncias descritas em **?** e a conclusão é que somente instâncias com até 20 carros podem ser resolvidas em tempo razoável. Com instâncias de 10 carros, o resultado é obtido em aproximadamente 1 segundo, mas para 20 carros, o tempo de execução exigido é de alguns minutos. Quanto maior o número de carros na fase de processamento, maior o uso de memória virtual e consequentemente, maior o tempo de execução.

Um algoritmo genético foi implementado por **?**. Tal algoritmo possui três operadores, *Interest Based Crossover*, que efetua cortes aleatórios a fim de rearranjar a sequência de forma viável, tentando satisfazer o número máximo de restrições. O *Uniform Interest Crossover*, cria uma máscara de bits com valor nulo indicando quais as classes de carros que serão deslocadas

para outra posição. Por fim, *Non Conflict Position Crossover*, que tenta usar posições sem conflitos entre pais a fim de gerar bons filhos.

Três tipos de codificações na forma normal conjuntiva foram apresentadas por **?**, como descrito anteriormente na versão de otimização. A primeira foi codificação de contador sequencial, que descreve como codificar uma restrição, a segunda é a restrição de capacidade, e a terceira é a conexão entre os carros com as classes e opções, um processo baseado em cláusulas. Os resultados para a versão de decisão são baseados em limites inferiores e superiores, e assim, mostrou-se que codificações FNC possuem bom desempenho nas instâncias da biblioteca de testes do CSP denotada CSPLib.

Recentemente, foi criado por **?** um algoritmo de filtragem em tempo linear a fim de atingir um estado válido sobre a restrição de capacidade. O resultado é satisfatório quando usado com instâncias da CSPLib. Foram propostos por **?** modelos CP/SAT, programação por restrições e satisfatibilidade. O modelo foi comparado com codificações SAT. Desenvolvido um mecanismo em tempo linear para explicar o fracasso e a restrição de poda. O modelo foi capaz de obter bons resultados em 13 de 23 instâncias disponíveis na CSPLib.

Um algoritmo de filtragem baseado em folgas foi criado por **?**. Apesar de simples e fácil de implementar, este algoritmo se mostrou bastante eficiente. Criado a partir de heurísticas baseadas em quatro critérios: variáveis de ramificação, direções de exploração, parâmetros para seleção de variáveis de ramificação e funções de agregação para estes critérios. Dessa forma, o algoritmo de filtragem se mostrou muito eficiente em relação a outros propagadores considerados estados da arte, como: *Sum*, *Gsc*, *AtMostSeqCard*, *Gsc* $\oplus$ *AtMostSeqCard*. Este algoritmo de filtragem é considerado, portanto, o estado da arte da versão de decisão.

Uma abordagem proposta por **?** com o objetivo de resolver o CSP é apresentada. O problema é dividido em dois subproblemas: o primeiro gera sequências de carros viáveis que satisfaçam as restrições de capacidade para todos os modelos de automóveis no plano de produção e o segundo determina uma sequência admissível com o menor *makespan*. Na segunda parte, **?** descrevem os detalhes das técnicas utilizadas. Uma ferramenta *Constraint Logic Programming* foi utilizada para resolver o CSP. Um predicado que assegura que um número de carros será escolhido para compor a sequência e que controla as limitações de capacidade das estações de trabalho foi também utilizado. Como resultado, foi possível obter 239 sequências viáveis num espaço de busca equivalente a $4,23 \times 10^{19}$.

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 opções 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 opções disponíveis;
- $p : O \to \mathbb{N}$ e $q : O \to \mathbb{N}$, definem as restrições de capacidade associadas à cada opção $o_i \in O$, ou seja, em uma subsequência de tamanho $q(o_i)$ não deve haver mais que $p(o_i)$ carros requerendo a opção o_i;
- $r : C \times O \to \{0, 1\}$ determina se há demanda por uma opção ou não, retornando 1 se a opção 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 opções pertencentes a cada classe e as subsequências que fazem parte de uma sequência. Tais definições estão descritas a sequir:

- $S = [1, \cdots, n]$, 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 opções 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 opção 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 custo de uma sequência S, dado pela Equação 3.2, obtido através do somatório de todas as violações ocorridas. Se a solução não violar nenhuma restrição, a solução é ótima.

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

Define-se para cada opção o_j, o conjunto de classes de carros que requerem esta opção, $C_{o_j} = \{c | o_j \in O_c\}$ e a demanda por opções, $D_{o_j} = \sum_{c \in C_{o_j}} D_c$ 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 opções de forma que:

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

De acordo com os dados, temos: $C_{o_1} = \{1, 4\}$, $C_{o_2} = \{2\}$, $C_{o_3} = \{1, 2\}$, $C_{o_4} = \{3, 4\}$ e $D_{o_1} = 3$, $D_{o_2} = 3$, $D_{o_3} = 5$, $D_{o_4} = 3$. Portanto, 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 **?** sintetizada na Tabela 3.1 é composta por estes dados e está organizada da seguinte maneira: na primeira coluna, existem 5 opções a serem instaladas nos carros. Nas quatro colunas do meio estão as classes de carros: cada classe requer um conjunto distinto de opções, como por exemplo, as classes 3 e 4 requerem a opção 4, mas a classe 3 requer também a opção 5 e a classe 4, a opção 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 a opção 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 tipos de opções.

Opções	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 a opção 1 é requerida pelos carros das classes 1 e 4 e a razão de capacidade é 3:5 indicando que não mais que 3 carros que requerem a opção 1 podem estar numa subsequência de tamanho 5. Considerando os carros que requerem a opção 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 $[c_2; c_3; c_1; c_4; c_4; c_2; c_3; c_1]$.

Opções	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 custo da solução de acordo com as violações das restrições. Ao analisar a Tabela 3.2, percebe-se que nenhuma restriçao foi violada. Como a função objetivo foi formulada para minimizar o custo de uma sequência em função da quantidade de violações, verifica-se que o custo desta sequência possui valor zero, portanto, a solução é ótima.

Capítulo 4

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

Neste capítulo, é descrita em detalhes a forma de resolução do CSP através do método Algoritmo Genétido 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 **?**, em que os indivíduos mais fortes têm maior probabilidade de encontrar um parceiro e perpetuar seu material genético nas próximas gerações.

Para o CSP, os indivíduos mais fortes correspondem às sequências de produção de carros que contêm o menor número de violações de restrições de capacidade. 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 ou qualidade da melhor solução encontrada. De uma forma geral, o algoritmo é executado em alguns passos. O BRKGA manipula os dados de cada instância do CSP a fim de obter uma sequência viável para a produção dos carros.

Cada instância é composta pelo número de carros a serem produzidos, pela quantidade de opções 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 uma opção. O valor 1 indica que o carro requer a opção correspondente àquela coluna juntamente com as restrições de capacidade referente àquela opção. Nas seções a seguir, cada um dos passos são apresentados.

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 é mostrado na Figura 4.1.

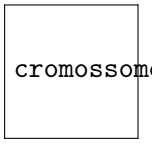

Figura 4.1: Cromossomo com 8 chaves aleatórias. Adaptado de **?**.

O cromossomo da Figura 4.1 é representado por um vetor de oito posições, em que cada posição contém uma chave aleatória que corresponde às suas características. No caso do CSP, o número de chaves aleatórias corresponde ao número de carros a serem produzidos e as características correspondem às opções que o carro requer. Simbolicamente, cada chave aleatória representa um carro.

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 **?**, 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. Novamente, no CSP, o tamanho do cromossomo é dado pelo número de carros a serem produzidos. 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 e o custo da mesma. Após a inicialização da população, a decodificação é realizada. O decodificador calcula o custo 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.

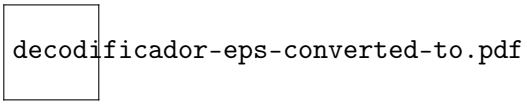

Figura 4.2: Mapeamento feito pelo decodificador, em que uma solução do espaço de chaves aleatórias é mapeada no espaço de solução do problema. Adaptado de **?**.

A decodificação é responsável por apresentar uma sequência de produção dos carros e seu custo é calculado pelo número de violações ocorridas. É mostrada na Figura 4.3 uma instância do CSP composta pela produção de 8 carros, 5 opções, 4 classes, 5 restrições de capacidade e 2 modelos a serem produzidos de cada classe. As classes de carros 0 e 3 requerem a opção 1 e a restrição de capacidade correspondente é 3:5, a opção 2 é requerida apenas pela classe 1 com restrição de capacidade 1:3. Já as classes 0 e 1 requerem a opção 3 com restrição 3:5. A opção 4 é requerida pelos carros das classes 2 e 3 com restrição 2:3 e por fim, a classe 2 requer a opção 5 com restrição de capacidade 2:4.

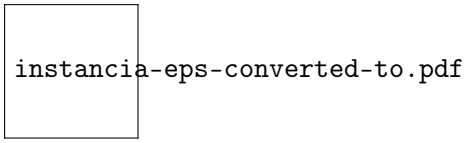

Figura 4.3: Exemplo de uma instância.

No processo de decodificação, é criado um gabarito 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, em seguida seus genes são ordenados e por fim, é feita a permutação do gabarito com base nos índices do vetor, resultando em uma solução. O processo descrito é apresentado pela Figura 4.4.

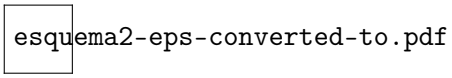

Figura 4.4: Exemplo de uma solução a partir do gabarito.

O próximo passo, é avaliar o custo da solução de acordo com o número de violações ocorridas. No caso desta sequência, o custo é igual a 1, pois viola a restrição de capacidade

2:4. Ao aplicar a javela deslizante no intervalo de carros [2, 2, 1, 3] da penúltima linha da Tabela 4.2, observa-se que a opção 5 é requerida por três dos quatro carros presentes no intervalo, sendo que deveria ter no máximo 2. Esta é a única violação ocorrida.

Tabela 4.1: Solução da sequência $[1, 0, 2, 2, 1, 3, 0, 3]$.

Opções	Classes de Carros								Restrições
	1	0	2	2	1	3	0	3	$p : q$
1		√				√	√	√	3:5
2	√				√				1:3
3	√	√			√		√		3:5
4			√	√				√	2:3
5			√	√		√			2:4
Req.	2	2	2	2	2	2	2	2	

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.

4.4 Elitismo

De acordo com a teoria evolucionista de **?**, 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. Todos os elementos do conjunto elite são então adicionados sem alterações na população da próxima geração. 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.

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 **?**. 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, ou seja, para cada gene do filho, é gerado um valor aleatório que decide se o filho herdará o gene do pai elite ou do pai não-elite. O processo de *crossover* é mostrado na Figura 4.5.

crossover-eps-converted-to.pdf

Figura 4.5: *Crossover*. Adaptado de **?**.

Nota-se pela Figura 4.5 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 Troca de Cromossomos Entre Populações

Durante o processo de evolução, é possível utilizar estratégias para convergir a solução para um ótimo global. Assim, é possível executar o BRKGA de forma que uma ou mais populações independentes são evoluídas simultaneamente ao longo das gerações possibilitando a troca dos melhores cromossomos de cada população após um determinado número de gerações preestabelecido. Porém, como relatado por **?**, um alto número de trocas leva à ruptura do processo evolutivo. Dessa maneira, após um determinado número de gerações, um número fixo de melhores cromossomos de cada população são escolhidos e transferidos para as populações em que eles não estão presentes e em compensação, são excluídos o mesmo número fixo de piores cromossomos, com o objetivo de manter a população com o mesmo número de indivíduos.

4.8 Visão Geral

A Figura 4.6 mostra um fluxograma ilustrando 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.

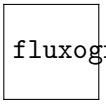

Figura 4.6: Fluxograma do algoritmo BRKGA. Adaptado de **?**.

Quando o BRKGA é utilizado para a resolução do CSP, apenas o processo de decodificação é alterado. A Figura 4.7 apresenta a 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 dentro do BRKGA e está descrita na Seção 4.3. Para a instância apresentada no exemplo, a solução é ótima, pois possui valor da função objetivo igual a 0 e a sequência apresentada é $S = [1, 0, 2, 3, 0, 2, 1, 3]$.

Figura 4.7: Exemplo da execução do BRKGA em uma instância do CSP.

Através da Tabela 4.2 é possível analisar com detalhes a solução da instância. Pode-se observar que todas as exigências são respeitadas: são produzidos 2 modelos de cada classe, totalizando 8 carros e ao aplicar a janela deslizante na solução, nenhuma violação por restrição de capacidade ocorre, confirmando o fato de que a solução é ótima.

Tabela 4.2: Solução da sequência $[1, 0, 2, 3, 0, 2, 1, 3]$.

Opções	Classes de Carros								Restrições
	1	0	2	3	0	2	1	3	$p : q$
1		√		√	√			√	3:5
2	√						√		1:3
3	√	√			√		√		3:5
4			√	√		√		√	2:3
5			√			√			2:4
Req.	2	2	2	2	2	2	2	2	

A partir da Figura 4.8, 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 4.8: Geração da população da próxima iteração. Adaptado de **?**.

4.9 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 quando o critério de parada for satisfeito. Esse critério pode ser por exemplo, o número de populações evoluídas, tempo ou qualidade da melhor solução encontrada. 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 custos 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 custo 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 stisfeito 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*
```

4.10 Interface de Programação de Aplicações

Foi desenvolvida por **?** 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. 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 **?**, a API é capaz de codificar chaves em paralelo, sendo necessário somente implementar o decodificador e especificar os critérios de parada, ou outras operações necessárias. Neste trabalho, será utilizada a API do BRKGA com o objetivo de resolver o CSP.

Capítulo 5

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.

5.1 Ambiente Computacional

Todos os testes foram executados em um computador com 4GB de memória RAM, processador Intel Core™ i3-3217U CPU @ 1.80GHZ × 4 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.

5.2 Ajuste de Parâmetros

Como o BRKGA é um algoritmo evolutivo, é necessário estabelecer alguns parâmetros para executá-lo, como 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 **?** versão 2.1.1662. Na Tabela 5.1, verifica-se a descrição das variáveis juntamente com intervalo de valores considerados neste experimento.

Tabela 5.1: Parâmetros para ajuste na ferramenta *irace*.

Descrição	Intervalo
Tamanho total da População (*pop*)	[1..8]
Tamanho da população elite (*pe*)	[0,10..0,25]
Tamanho da população mutante (*pm*)	[0,10..0,30]
Número máximo de gerações (*MaxGens*)	[500..1500]
Número de indivíduos a serem trocados (*XNumber*)	[2..4]
Probabilidade do filho herdar o gene do pai elite (*rhoe*)	[0,50..0,80]
Intervalo para a troca de elementos (*XIntvl*)	[50..200]

Abaixo são listados os valores recomendados pela ferramenta *irace* após realizados os experimentos de ajustes de parâmetros:

- *pop*: 7. O tamanho da população total é dado por $pop = pop \times$ número de genes;
- *pe*: 0,15;
- *pm*: 0,22;
- *MaxGens*: 1185;
- *XNumber*: 2;
- *rhoe*: 0,72;
- *xIntvl*: 86.

5.3 Comparação de Resultados

Todas as instâncias utilizadas nos experimentos encontram-se disponíveis na biblioteca **?** e são divididas em 3 conjuntos: o primeiro é formado por 70 instâncias propostas por **?**, as denominadas *easy*, conhecidas pela facilidade em se obter a solução ótima. O segundo conjunto é composto por 30 instâncias propostas por **?** conhecidas como instâncias *hard*, e o último contém 9 tradicionais instâncias denominadas *harder* propostas por **?**.

As tabelas de comparação possuem um padrão para todos os conjuntos de instâncias. A primeira coluna indica qual instância está sendo analisada, a segunda contém o número de restrições violadas na solução obtida pelo método BRKGA e a terceira coluna contém o tempo de execução deste método, em segundos. A quarta coluna apresenta o melhor resultado presente na literatura para cada instância e a quinta coluna apresenta o tempo necessário para obter tal resultado. Quando aplicável, há uma sexta coluna indicando qual método da literatura é responsável pelo melhor resultado.

5.3.1 Instâncias *easy*

O primeiro conjunto é formado por 70 instâncias com 200 carros, 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 carros cada, dispostos de acordo com sua taxa de utilização na ordem 60%, 65%, 70%, 75%, 80%, 85% e 90%. De acordo com **?**, a dificuldade de resolução de uma instância é dada pelo número de carros a serem produzidos, pelo número das configurações das diferentes opções e também sobre a taxa de utilização. A taxa de utilização de uma opção o_i dada pela Equação 5.1 corresponde ao número de carros que exigem a opção o_i em uma subsequência de tamanho $q(o_i)$ em relação ao número máximo de carros que poderiam requerer a opção 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.

$$\textit{Taxa de Utilização}(O_i) = \frac{r(C, o_i).q(o_i)}{|C|.p(o_i)} \tag{5.1}$$

A Tabela 5.2 apresenta as instâncias com taxa de utilização de 60% e 65%, a Tabela 5.3 com 70% e 75%, a Tabela 5.4 as taxas 80% e 85% e a Tabela 5.5 a taxa de 90%. Os melhores resultados para este conjunto de instâncias foram obtidos pela metaheurística *Ant Colony Optimization* (ACO) proposta por **?**. Os resultados foram baseados em 100 execuções e o tempo foi dado através do desvio padrão, o método obteve solução ótima e desvio padrão igual a zero em todas as instâncias.

Tabela 5.2: Instâncias propostas por **?** com taxas de utilização de 60% e 65%.

Instância	BRKGA	Tempo(s)	Melhor Resultado
60-01	0	328,2	0
60-02	0	328,3	0
60-03	0	328,4	0
60-04	0	330,4	0
60-05	0	328,6	0
60-06	0	327,3	0
60-07	0	330,1	0
60-08	0	328,8	0
60-09	0	327,5	0
60-10	0	330,1	0
65-01	0	328,9	0
65-02	0	325,0	0
65-03	0	330,8	0
65-04	0	329,2	0
65-05	0	329,0	0
65-06	0	327,6	0
65-07	0	329,5	0
65-08	0	329,4	0
65-09	0	327,6	0
65-10	0	330,2	0

Tabela 5.3: Instâncias propostas por **?** com taxas de utilização de 70% e 75%.

Instância	BRKGA	Tempo(s)	Melhor Resultado
70-01	0	330,2	0
70-02	0	327,5	0
70-03	0	329,1	0
70-04	0	327,8	0
70-05	0	329,5	0
70-06	0	328,5	0
70-07	0	330,5	0
70-08	0	327,3	0
70-09	0	325,2	0
70-10	0	329,9	0
75-01	0	327,9	0
75-02	1	324,2	0
75-03	0	327,5	0
75-04	3	322,6	0
75-05	0	322,8	0
75-06	1	324,2	0
75-07	1	327,9	0
75-08	0	323,1	0
75-09	1	326,0	0
75-10	0	327,2	0

Tabela 5.4: Instâncias propostas por **?** com taxas de utilização de 80% e 85%.

Instância	BRKGA	Tempo(s)	Melhor Resultado
80-01	0	324,3	0
80-02	4	339,7	0
80-03	0	342,1	0
80-04	3	334,5	0
80-05	2	336,9	0
80-06	3	332,7	0
80-07	3	340,2	0
80-08	0	332,1	0
80-09	4	331,7	0
80-10	4	336,9	0
85-01	3	335,5	0
85-02	3	334,0	0
85-03	0	340,9	0
85-04	9	337,2	0
85-05	10	323,2	0
85-06	7	331,3	0
85-07	3	329,2	0
85-08	4	323,3	0
85-09	6	327,9	0
85-10	1	337,1	0

Tabela 5.5: Instâncias propostas por **?** com taxas de utilização de 90%.

Instância	BRKGA	Tempo(s)	Melhor Resultado
90-01	16	344,0	0
90-02	9	343,2	0
90-03	11	321,1	0
90-04	6	322,1	0
90-05	19	317,1	0
90-06	0	328,2	0
90-07	16	318,9	0
90-08	13	318,9	0
90-09	4	319,8	0
90-10	7	316,0	0

Apesar deste conjunto possuir instâncias de fácil solução, diferentemente do método ACO, o método proposto não obteve solução ótima em todas as instâncias. O tempo de execução se manteve alto e constante em todo o conjunto. As instâncias que exigem menor taxa de utilização por opções 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 75%.

5.3.2 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 **?**, as codificações da Forma Normal Conjuntiva (CNF) propostas por **?**, a *Iterated Beam Search* (IBS) proposta por **?** e a metaheurística *Ant Colony Optimization* (ACO) de **?** que usou desvio padrão como critério de desempenho, portanto não é apresentado na tabela, pois não possui tempo em segundos para comparar com os outros métodos, obtiveram os melhores resultados entre todos os métodos presentes na literatura que consideraram estas instâncias. A Tabela 5.6 apresenta as instâncias com 200, 300 e 400 carros.

Tabela 5.6: Instâncias *hard* propostas por **?**.

Instância	BRKGA	Tempo(s)	Melhor Resultado	Tempo(s)	Métodos
Pb-200-01	47	332,3	0	3,40	VFLS
Pb-200-02	30	334,9	2	0,86	CNF
Pb-200-03	47	329,5	3	93,3	VFLS
Pb-200-04	53	318,8	7	1,04	CNF
Pb-200-05	31	311,9	3	1,89	VFLS
Pb-200-06	42	318,8	6	0,52	VFLS
Pb-200-07	30	314,9	0	0,57	VFLS
Pb-200-08	29	316,8	8	0,18	VFLS
Pb-200-09	37	321,2	10	0,32	VFLS
Pb-200-10	33	316,2	17	0,18	VFLS
Pb-300-01	58	738,1	0	5,55	IBS
Pb-300-02	62	738,0	6	0,57	VFLS
Pb-300-03	72	743,5	13	0,68	VFLS
Pb-300-04	61	739,9	7	1,46	VFLS
Pb-300-05	100	734,9	16	141,61	VFLS
Pb-300-06	71	737,7	2	13,93	VFLS
Pb-300-07	65	734,7	0	4,26	VFLS
Pb-300-08	52	737,9	8	0,53	VFLS
Pb-300-09	60	738,2	7	1,16	VFLS
Pb-300-10	71	738,2	13	3,13	VFLS
Pb-400-01	74	1139,9	1	55,5	VFLS
Pb-400-02	108	1343,6	15	16,59	VFLS
Pb-400-03	79	1342,6	9	0,53	VFLS
Pb-400-04	86	1340,8	19	0,47	VFLS
Pb-400-05	72	1341,0	0	0,38	IBS
Pb-400-06	76	1338,8	0	1,57	VFLS
Pb-400-07	87	1348,6	4	8,26	VFLS
Pb-400-08	87	1338,6	4	8,31	VFLS
Pb-400-09	92	1341,8	5	3,40	VFLS
Pb-400-10	76	1344,1	0	3,05	VFLS

Ao comparar os resultados obtidos pelo método proposto com os melhores da literatura, percebe-se a dificuldade que a metaheurística proposta encontra para obter resultados razoavelmente próximos. Além do alto tempo de execução para cada instância, o número de violações é extremamente alto. O tempo de execução se manteve constante para cada conjunto de instâncias e o número de violações aumentou à medida em que aumentou o número de carros.

5.3.3 Instâncias *harder*

O último conjunto é composto por 9 tradicionais instâncias propostas por **?**, 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 5.7 apresenta os resultados para este conjunto de instâncias. Os métodos *Very Fast Local Search* (VFLS) proposto por **?**, as codificações da Forma Normal Conjuntiva (CNF) de **?** e a metaheurística *Ant Colony Optimization* (ACO) de **?** obtiveram os melhores resultados para este conjunto de instâncias e são apresentados na referida tabela com exceção do método ACO que usou desvio padrão como forma de medir seu desempenho.

Tabela 5.7: Instâncias propostas por **?**.

Instância	BRKGA	Tempo(s)	Melhor Resultado	Tempo(s)	Métodos
4/72	13	77,3	0	0,07	CNF
6/76	13	78,4	6	0,00	VFLS
10/93	28	77,3	3	0,53	CNF
16/81	21	77,9	0	0,07	CNF
19/71	17	78,2	2	0,51	VFLS
21/90	13	78,0	2	0,11	VFLS
26/82	14	78,0	0	0,14	CNF
36/92	19	77,8	1	0,05	CNF
41/66	8	77,6	0	0,03	VFLS

Para este conjunto de instâncias, verificou-se novamente um alto tempo de execução comparado ao melhor método presente na literatura. Nota-se que o tempo se manteve constante durante a execução das instâncias, ficando por volta de 78 segundos. O número de violações obtido pelo método proposto também é alto quando comparado com o melhor resultado reportado na literatura.

5.4 Análise Geral

Após a realização dos experimentos, houve a análise dos dados de acordo com critérios de desempenho. O sumário dos resultados encontra-se na Tabela 5.8. Na primeira coluna, há a descrição de cada critério considerado e na segunda coluna, o valor correspondente.

Tabela 5.8: Sumário dos resultados.

Tempo máximo de execução	29.576,2 s
Tempo médio de execução	271,3 s
Número médio de violações	20,2
Número de soluções sem violações	40
Número de resultados iguais aos da literatura	40

Os resultados obtidos não são satisfatórios. O tempo máximo de execução e o tempo médio foram extremamente altos. Apenas para 40 das 109 instâncias consideradas nos experimentos foram obtidos resultados iguais aos da literatura e, coincidentemente, foram os resultados em

que a solução ótima foi obtida. De acordo com os resultados, há uma demanda de implementação de métodos adicionais para refinamento da metaheurística proposta a fim de se obter melhores resultados.

Capítulo 6

Plano de Atividades Restantes

Neste capítulo é apresentado o planejamento para a conclusão da monografia como um todo. A Tabela 6.1 descreve as seis etapas a serem desenvolvidas a fim de concluir este trabalho.

Tabela 6.1: Planejamento de desenvolvimento para a monografia 2.

Atividades	Mês 1	Mês 2	Mês 3	Mês 4
Pesquisar sobre métodos de refinamento	√			
Implementar métodos de refinamento da heurística		√	√	
Executar experimentos			√	
Analisar experimentos realizados			√	√
Descrever os experimentos				√
Conclusão do texto da monografia				√

As atividades restantes se concentram basicamente em pesquisar métodos de refinamento que possam ser adaptados à metaheurística, como métodos de busca local, implementando e analisando-os quanto aos critérios de melhoria para ao Algoritmo Genético de Chaves Aleatórias proposto. As novas versões da metaheurística serão submetidas a novos experimentos computacionais e análises adicionais baseadas nestes resultados. Por fim, o trabalho será concluído pela escrita do texto final da monografia.

Capítulo 7

Conclusões

Este trabalho considera parte do processo de produção de carros em uma indústria, denominada Problema de Sequenciamente 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. 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 insatisfatórios comparado aos resultados obtidos por outros métodos da literatura. Entretanto, os trabalhos futuros incluem o aprimoramento da metaheurística proposta através de métodos de busca local, como *2-swap*, *2-opt*, e *best insertion* a fim de aprimorar os resultados obtidos pelo método evolutivo, em busca da solução ótima para o CSP. Consequentemente, a ciclo de validação das implementações, ajuste de parâmetros e experimentos computacionais serão novamente realizados para as novas versões do método proposto.

Referências Bibliográficas

Anuário da Indústria Automobilística Brasileira, A. . (2016). http://www.anfavea.com.br/anuario.html.

Artigues, C.; Hebrard, E.; Mayer-Eichberger, V.; Siala, M. e Walsh, T. (2014). SAT and hybrid models of the car sequencing problem. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*.

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

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/.

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.

Dincbas, M.; Simonis, H. e Van Hentenryck, P. (1988). Solving the car-sequencing problem in constraint logic programming. In *ECAI*, volume 88, pp. 290–295.

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. (1998). Two results on car-sequencing problems. *Research Reports of the APES Group, APES-02-1998, Available from Http://www. dcs. stand. ac. uk/˜ Apes/apesreports. html*.

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. (2011a). Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, 17(5):487–525.

Gonçalves, J. F. e Resende, M. G. (2011b). A parallel multi-population genetic algorithm for a constrained two-dimensional orthogonal packing problem. *Journal of Combinatorial Optimization*, 22(2):180–201.

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.

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.

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.

Mazur, M. e Niederliński, A. (2015a). A two-stage approach for an optimum solution of the car assembly scheduling problem part 1. problem statement, solution outline and tutorial example. *Archives of Control Sciences*, 25(3):355–365.

Mazur, M. e Niederliński, A. (2015b). A two-stage approach for an optimum solution of the car assembly scheduling problem. part 2. clp solution and real-world example. *Archives of Control Sciences*, 25(3):367–375.

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://openmp.org/wp/,%20acessado%20em%2011%20de%20maio%20de%202013.

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.

Régin, J.-C. e Puget, J.-F. (1997). A filtering algorithm for global sequencing constraints. In *International Conference on Principles and Practice of Constraint Programming*, pp. 32–46. Springer.

Siala, M.; Hebrard, E. e Huguet, M. J. (2015). A study of constraint programming heuristics for the car-sequencing problem. *Engineering Applications of Artificial Intelligence*.

Siala, M.; Hebrard, E.; Huguet, M.-J.; Siala, M.; Hebrard, E.; Huguet, M.-J. e Huguet, ·. M.-J. (2014). An optimal arc consistency algorithm for a particular case of sequence constraint. *Constraints*, 19:30–56.

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.

Zinflou, A.; Gagné, C. e Gravel, M. (2007). Crossover operators for the car sequencing problem. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pp. 229–239. Springer.