Universidade Federal de Ouro Preto - UFOP
Instituto de Ciências Exatas e Biológicas - ICEB
Departamento de Computação - DECOM

Algoritmo Heurístico Aplicado a Produção de Automóveis

Bolsista: João Vitor Mascarenhas dos Santos
Orientador: Marco Antonio Moreira de Carvalho – DECOM/UFOP

Nota: Relatório final referente ao período de 05/10/2016 a 28/02/2017, apresentado à Universidade Federal de Ouro Preto, como parte das exigências do Programa Institucional de Voluntários de Iniciação Científica - Primeiro Semestre

Local: Ouro Preto - Minas Gerais - Brasil
Data: 13 de fevereiro de 2017

Resumo

Atualmente, linhas de produção mistas (em que mais de um modelo de produto é produzido a partir de uma mesma base) são utilizadas em uma grande gama de indústrias que produzem produtos personalizáveis, com destaque para a indústria automotiva, na qual diferentes modelos de automóveis são produzidos a partir de um mesmo chassi. Um problema combinatório de tomada de decisão importante neste contexto que tem recebido atenção de pesquisadores recentemente é o problema de sequenciamento da linha de produção, que se refere a sequência de produção e personalização dos diferentes automóveis. Além disto, se há múltiplos departamentos envolvidos, cada um com um interesse diferente na produção de um mesmo veículo a serem considerados, ou devido à intempéries, como falta de estoque de material ou falha de máquinas, torna-se essencial replanejar a linha de produção, sequenciando somente automóveis que podem de fato ser produzidos e personalizados de acordo com cada cenário. Um caso particular de sequenciamento de linhas de produção automotivas é a definição de lotes de automóveis para pintura. Este problema possui alto impacto nos custos e também no tempo de produção no setor automotivo, uma vez que, tipicamente em uma linha de montagem, um carro está pronto para pintura a cada minuto e a pintura, considerando que nenhuma preparação adicional seja necessária, é realizada em 3 minutos. Um planejamento ineficiente desta etapa pode aumentar consideravelmente o tempo exigido para realização da tarefa e ocasionar gargalos na linha de produção. Este trabalho utiliza os métodos *Depth-First Search*, *Best Insertion* e *Iterated Local Search* para resolver o problema de sequenciamento de carros. *Depth-First Search* provou ser um método de rápida execução e bom em gerar soluções iniciais, *Best Insertion* é capaz de obter os melhores resultados, dentre os três métodos e *Best Insertion* demonstrou ser uma boa estratégia para ser combinada com o *Iterated Local Search*.

Assinatura do orientador(a): _____

Marco Antonio Moreira de Carvalho

Assinatura do bolsista: _____

João Vitor Mascarenhas dos Santos

Sumário

Capítulo 1

Introdução

Henry Ford desenvolveu uma nova forma de produzir carros em 1903, conhecida como "Fordismo", onde cada trabalhador teria um trabalho muito específico para executar. Esta inovação reduziu muito o tempo de produção e custos por cada unidade produzida. Devido ao tempo e custo de produção reduzidos, foi possível iniciar a produção em massa. Posteriormente, isso levou a linhas de produção mistas em que uma única linha poderia produzir mais do que um tipo de produto, por exemplo, uma indústria automóvel poderia construir vários tipos de carros a partir do mesmo chassi. De acordo com **?**, a customização em massa foi criada depois disso, levando a produtos altamente diferenciados. Por exemplo, em termos de personalização, a Mercedes produziu 1,1 milhões de carros na fábrica de Rasstatt na Alemanha entre os anos de 2003 e 2005. De todos esses carros, apenas dois eram idênticos.

De acordo com **?**, no início de cada dia de trabalho, o planejamento da produção de uma sequência de carros precisa ser realizada. Muitas vezes, isto exige que as peças necessárias estejam em estoque e o tempo de cada estação de trabalho precisa ser levado em conta para que o planejamento seja viável. Para as peças a serem entregues na linha de produção, há duas opções, eles podem ser trazidas de fornecedores, que são *Just-in-Time* (JIT) e *Just-in-Sequence* (JIS). JIT traz as partes no momento em que são necessárias, enquanto o JIS irá pré encomendá-las, ou alterar a ordem quando elas chegam, de modo que elas vão estar na ordem em que são necessárias na linha de produção. Outra questão é que JIS exige armazenagem temporária, quer na fábrica ou nos fornecedores. Também, o tempo necessário para classificar as peças pode aumentar o tempo de entrega.

Uma estação de trabalho é um trabalhador ou uma máquina que executa um trabalho o que pode ser a instalação de um item opcional. Um item opcional é algo que o comprador seleciona quando ele decide qual carro é o melhor para suas necessidades. Alguns exemplos de itens opcionais são assentos de couro, teto solar e ar condicionado. A linha de produção pode ter estações de trabalho móveis de duas maneiras: física e virtual. No modo físico, os carros são sequenciados a fim de acompanhar os tempos de trabalho das estações. No modo virtual as estações de trabalho se movem ao longo da linha de produção e os carros não precisam ser alterados a partir da sequência original.

Eventos como falta de peças ou máquinas quebradas podem levar a um cenário em que o sequenciamento dos carros precisa ser mudado. Isso pode levar a grandes perdas monetárias e deve ser evitado a todo custo. A falta de peças pode ser evitada criando uma sequência melhor para os carros serem produzidos, levando em consideração o estoque. Falhas nas máquinas podem ser evitadas fazendo uma manutenção regular e através da introdução de redundância na linha de produção, isto é, ter mais do que uma máquina, sempre que possível, que pode fazer o mesmo trabalho. Esta é uma linha de pesquisa conhecida como *production forecast*, que tenta

manter todos os equipamentos funcionando e cuidar do estoque das peças.

A *French Society of Operations Research and Decision Analysis* (ROADEF) é uma sociedade internacional que, entre outras atribuições, organiza um concurso internacional, bienal, com foco na interação da academia com as empresas, a fim de fazer as melhorias feitas pela academia sejam aplicadas em empresas para obter melhores resultados de produção. A *French Society of Operations Research and Decision Analysis* (ROADEF) também formaliza a descrição de problemas específicos e criam conjuntos de instâncias ao longo do ano de competição. Também desenvolvem verificadores de solução para que os participantes possam ver o custo final de sua solução,o formato correto e também validar as restrições. Esta sociedade, muitas vezes oferece prémios, sob a forma de dinheiro para os seus participantes, podendo chegar até onze mil euros. ROADEF começou a realizar competições em 1999. Cada ocorrência é focada em um tipo diferente de indústria:

- ROADEF'99: *Inventory Management Problem*;
- ROADEF'01: *Frequency Assignment Problem with Polarization constraints*;
- ROADEF'03: *Management of the Missions of the Earth Observation Satellites*;
- ROADEF'05: *Car Sequencing problem*;
- ROADEF'07: *Technicians and Interventions Scheduling for Telecommunications*;
- ROADEF'09: *Disruption Management for Commercial Aviation*;
- ROADEF'10: *A Large-Scale Energy Management Problem with varied constraints*;
- ROADEF'12: *Machine Reassignment*;
- ROADEF'14: *Rolling Stock Unit Management on Railways Sites*; e
- ROADEF'16: *Inventory Routing Problem*.

Este estudo focará especificamente no ROADEF'05 com o assunto Problema de Sequenciamento de Veículos.

? e **?** afirmam que o Problema de Sequenciamento de Veículos (CS) é um problema em que uma empresa tem de produzir uma determinada quantidade de carros em um dia de trabalho e os carros deste grupo têm itens opcionais a serem adicionados (por exemplo, rádio, teto solar, ar condicionado, etc.). Ao lidar com este tipo de problema, o objetivo é reduzir a carga de trabalho de cada estação de trabalho. A carga de trabalho destas estações são frequentemente definidas pela proporção de carros que elas podem processar, ou seja, instalar o item opcional. Por exemplo, uma determinada estação poderia instalar um determinado item opcional em três a cada sete carros. Isso significa que nenhuma sequência de carros deve ter mais de três carros com o item opcional a ser instalado, em um total de sete carros. Portanto, o custo de CS é geralmente associada com quantas vezes este tipo de restrição é violada.

O outro objetivo do Problema de Sequenciamento de Veículos é determinar a sequência de carros a serem produzidos, a fim de reduzir os custos de pintura, dado que cada mudança de cores implica em custos adicionais. De acordo com **?**, o custo pode variar de US$27 no preto até US$122 no âmbar. Outra preocupação é que cada personalização pode ser feita apenas em alguns carros em cada lote. Nota-se que neste caso algumas pausas podem ser necessárias a fim de remover a tinta que foi acumulando-se na ponta do bocal.

? e **?** mostram que o Problema de Sequenciamento de Veículos é um problema $\mathcal{NP}$-difícil. Este problema está sendo pesquisado ao longo dos últimos vinte anos, a fim de cortar custos na indústria automobilística. Os nove métodos a seguir, entre outros, têm sido utilizados para resolver este problema no passado:

- Satisfação de restrições;
- Métodos Guloso;
- Programação Inteira;
- *Backtracking*;
- *Branch and Bound*;
- *Large Neighbourhood Search*;
- Busca Local;
- Algorítmos Genéticos; e
- Colônia de Formigas.

A Figura 1.1 mostra o número de artigos publicados e registradas no *Web of Science*[1] têm aumentado desde que este problema foi proposto por **?** como uma variante do problema *job-shop schedule*. O pico em 2008 pode ser devido ao fato de que o desafio do ROADEF em 2005 atraiu a comunidade acadêmica para ele.

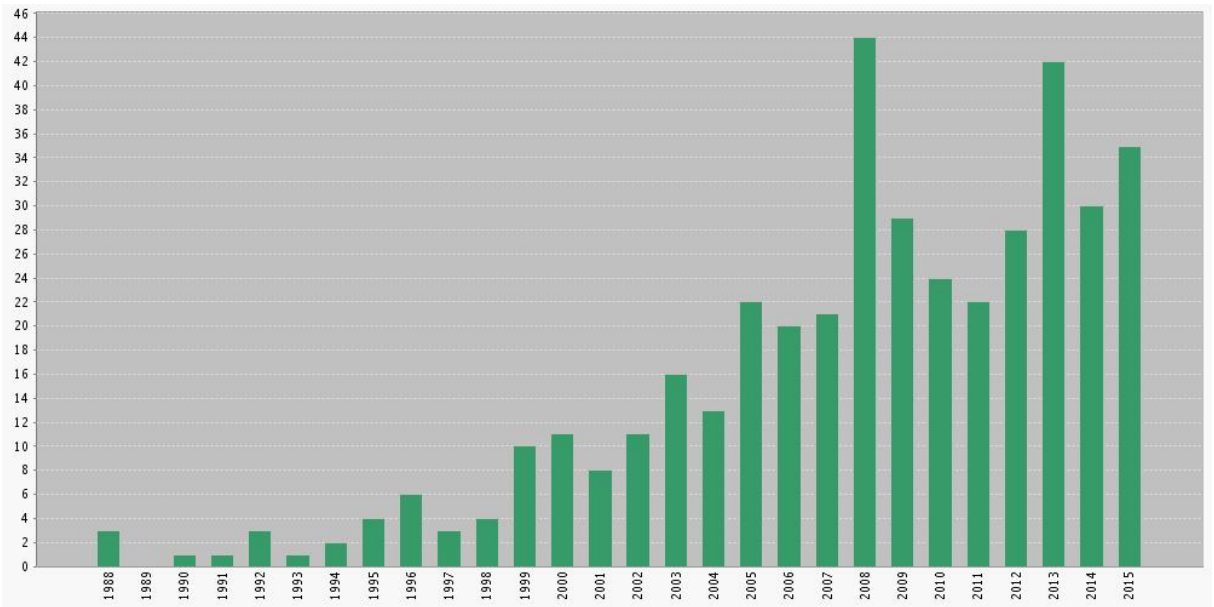

Figura 1.1: Artigos publicados sobre sequenciamento de veículos após 1987.

[1] https://webofknowledge.com/ Acessado em 26/01/2016

1.1 Motivação

Existem três principais motivações para este trabalho. Primeiro, trata-se um problema $\mathcal{NP}$-difícil. Em segundo lugar, este tem sido objeto de investigação por um grande sociedade, ROADEF. Em terceiro lugar, ele tem um grande impacto na economia do estado de Minas Gerais e do Brasil.

Como mencionado anteriormente, foi mostrado por **?** e **?** que este problema pertence à classe de problemas $\mathcal{NP}$-difícil, e em 2005, a *French Society of Operations Research and Decision Analysis* (ROADEF) junto com a fabricante de automóveis Renault, selecionaram este assunto como tema da competição. ROADEF pagou até sete mil euros como prêmio para os competidores. A competição contou com participantes de vários países, tais como França, Brasil, Holanda, Canadá, Alemanha, Suíça, Polónia e Bósnia e Herzegovina.

De acordo com a ANFAVEA[2] a participação da indústria automotiva no produto interno bruto brasileiro em geral foi de 5% e 23% sobre o setor industrial em 2013. A participação da indústria automotiva no produto interno bruto de Minas Gerais foi de 25,8%. Isso gerou em torno de R$ 178,5 bilhões em impostos. O Brasil é atualmente o sétimo maior produtor de automóveis do mundo.

[2]http://www.anfavea.com.br/50anos/60.pdf Acessado em 09/12/2015

Capítulo 2

Objetivos

Este estudo elaborou uma heurística, para solução do Problema de Sequenciamento da Produção de Automóveis. Este principal objetivo foi alcançado através dos seguintes passos a seguir descritos.

- Revisão da literatura sobre o problema de sequenciamento de veículos;
- Projeto de uma heurística que cumpre com os requerimentos do problema;
- Implementação da heurística mencionada previamente;
- Pesquisa por técnicas de polimento
- Avaliação do método com dados reais e casos de teste públicos; e
- Comparação do método com o estado da arte da literatura.

Capítulo 3

Revisão da Literatura

? foram os primeiros a descrever o problema de sequenciamento de carros em 1986. Em 2005 ROADEF utilizou o sequenciamento de carros como o problema do seu desafio bianual, constituindo um marco na história do problema. **?** descreveu-o como um estudo de caso do problema *job-shop scheduling*. Foi descrito passo a passo como reduzir o *job-shop* para mostrar o que é a seqüência de carros, os atributos e as restrições para cada item opcional. Eles também mostraram que ele pode não haver uma solução (isto é, mais carros que exigem a opção do que a fábrica pode lidar) para determinadas instâncias. Para estes casos, os autores criaram a ideia de penalidades para alterar o custo de uma solução. Seu principal objetivo era desenvolver um ambiente para permitir que outros pesquisadores resolvessem este problema usando o raciocínio automatizado, onde eles declaram um conjunto de regras e o *software* decide a sequência dos carros.

Por exemplo, considere que uma opção tem a restrição de proporcionalidade de 1 : 3 (isto é, por cada 3 carros dentro de uma sequência, apenas um pode ter o item opcional instalado), e que a entrada requer 2 carros com esta opção. A Tabela 3.1 mostra, no lado esquerdo, que, independentemente da sequência, a penalidade seria 1. Outro tipo de penalidade, aplicada quando há violação de restrição para carros próximos uns dos outros, é mostrada na terceira e quarta linha do lado direito da tabela. A tabela representa uma parte de uma sequência de carro, em que *S* significa um carro que tem a opção e *N* de outra forma.

Tabela 3.1: Ambos tipos de penalidade.

veículo	S	N	S
penalalidade	0	0	1
veículo	N	S	S
penalidade	0	0	1

veículo	S	N	S
penalidade	0	0	1
veículo	N	S	S
penalidade	0	0	3

3.1 Anos 90: Programação por Restrições

No início dos anos 90, os esforços para resolver problemas de sequenciamento de carros focaram-se principalmente sobre os métodos de satisfação de restrições. **?**, **?**, **?**, **?** e **?** desenvolveram novos algoritmos e técnicas para melhorar os *constraint satisfaction solvers* da época, tal como o *Constraint Handling in Prolog* (CHIP). Além disso, foram desenvolvidos operadores para acelerar a programação. Devido a limitações de *hardware* daquele tempo, a maioria dos experimentos realizados consideravam apenas casos pequenos de até 400 carros,

geralmente contendo apenas 100, e que levaram até 10 horas para processar. Os exemplos utilizados eram todos gerados aleatoriamente. **?** propôs operações de pré-processamento, a fim de remover possíveis valores ruins entre os dados.

Depois disso, **?** começou a desenvolver técnicas mais avançadas de satisfação de restrições, a fim de resolver o problema de sequenciamento de carros, mas percebeu que não era uma boa idéia. No entanto, este trabalho constituiu ideias interessantes:

- Carros com os mesmos opcionais seriam contados como uma classe de carros, em vez de um único carro. Se o número de opcionais crescesse esta ideia seria impraticável, porque o número de possíveis carros cresce exponencialmente com o número de opções (2^O, em que O é o número de itens opcionais);
- A ordem dos carros na entrada poderia ser alterada para que o *backtracking* feito pelo *solver* seja menos frequentes e menos pesada; e
- Uma técnica para a remoção de possíveis valores ruins antes de realizar operações. Um valor ruim é um valor que pode levar a uma solução de alto custo.

Instâncias com até cinco opções foram consideradas nas experiências relatadas, mas, como mencionado acima, os resultados foram desanimadores.

3.2 Anos 2000: Metaheurística, heurística e métodos exatos

? foram os primeiros a usar um *Genetic Algorithm with Hill Climbing* (GAwHC). O GAwHC foi comparado a uma busca tabu e os resultados apresentados mostraram que o GAwHC tinha um desempenho melhor. No entanto, o GAwHC apresentou algumas limitações, tais como o tamanho máximo da entrada e o tempo necessário para resolver uma instância.

No início dos anos 2000 foram empregadas uma maior diversidade de métodos para tratar o problema de sequenciamento de veículos. **?** utilizou vários tipos de busca gulosa, busca local, *backtracking*, *branch and bound*, também combinando-os. **?** misturou vários métodos também, mas incluiu a metaheurística *Ant Colony Optimization* (ACO). **?** utilizou *Large Neighbourhood Search* para resolvê-lo. **?** desenvolveram IDWALK, uma busca local que iria fazer algumas alterações na solução e verificar qual solução gerada depois daquela era a melhor. **?**, combinou *branch e bound* com busca tabu. Não é possível comparar estes métodos, uma vez que usam diferentes casos de teste do problema.

Em 2005 ROADEF escolheu o sequenciamento de carros como tema de seu concurso bianual. **?** escreveu um artigo para resumir o concurso e os resultados. ROADEF fez um detalhamento explícito de qual era o problema e como uma solução seria avaliada. É também demonstrada a dificuldade do problema, a sua complexidade, bem como alguns métodos utilizados pelos participantes para resolver o problema. No site [1] há a possibilidade de enviar novas soluções para avaliação e comparação com os resultados do desafio.

Após 2005, muito esforço foi tomado utilizando ACO. O livro **?** mostra várias maneiras de implementação, bem como problemas no qual tem sido usado no passado. Alguns autores como **?** utilizaram soluções aleatórias iniciais gulosas e depois disso aplicaram ACO para melhorar os resultados.

? compararam três métodos para resolver o problema de sequenciamento de carros, *Constraint Satisfaction programming*, programação inteira, e ACO. Foram utilizadas as instâncias da

[1] http://roadef.proj.info-ufr.univ-montp2.fr/2005/ Assessed on 27/01/2016

CSPLib e foi proposta uma nova forma de cálculo das penalidades para eliminar a dupla contagem de carros. No entanto, este novo cálculo não foi encontrado em qualquer outro trabalho.**?** utilizaram programação dinâmica e *simulated annealing*.

Nos últimos anos (a partir de 2009), houve uma ligeira diferença na formulação do problema. Autores iniciaram a usar mais de uma linha de produção ou removeram os custos da mudança da cor da otimização. **?**, **?** e **?**, seguiram esta linha de pesquisa.

? pesquisaram várias publicações anteriores que abordaram o problema de sequenciamento e veículos e apontaram o que eles pensavam ser uma agenda de pesquisa futura desta área. Esta agenda envolve abordar otimização multi-objetivo e resequenciamento da linha de produção.

Como sugerido por **?** artigos publicados após 2012 mostraram uma definição muito mais específica e local do problema de sequenciamento de carros. Tais como **?** e **?**, onde leva-se em conta o tempo de produção para produzir os veículos, ou **?**, onde é otimizada mais do que uma linha de produção ao mesmo tempo.

3.3 Complexidade Computacional

Em relação à dificuldade do problema, **?** mostrou que é $\mathcal{NP}$-completo, reduzindo-o para o caminho Hamiltoniano. **?** e **?**, mais tarde, mostraram que é um problema $\mathcal{NP}$-difícil.

Capítulo 4

Fundamentação Teórica

O Problema de sequenciamento do veículos é comumente definido por uma tupla (*C, S, K, P, Q, R*) em que *C* é o conjunto de carros a serem produzidos, *O* é o conjunto de diferentes itens opcionais que podem ser adicionados a um carro na fábrica atual, *K* é o conjunto de carros do dia anterior e *p* limita a quantidade máxima de carros que podem ter um item opcional específico instalado dentro de uma sequência de tamanho *q* (referenciado como restrições $p:q$ a partir de agora). O último elemento *r* determina se um carro possui ou não um item opcional instalado:

$$r(c_i, o_j) = \begin{cases} 1 \text{ se o carro } i \text{ possui o item opcional } j \\ 0 \text{ caso contrário} \end{cases}$$

O ROADEF propôs uma fórmula para o cálculo do custo de uma solução que é uma versão simplificada da fórmula proposta por **?**. Para cada item opcional $o_i \in O$ qualquer subsequência de tamanho q_i deve ser verificada para ver se há mais de p_i carros nesta subsequência necessitando deste item opcional o_i, de acordo com a Equação 4.1.

$$\text{penalidade p:q} = \sum_{i=1}^{O} \sum_{j=|K|-q_i+1}^{N} \begin{cases} (\text{count}(j, j+q_i) - p_i) * w_i \text{ se count}(j, j+q_i) - p_i > 0 \\ 0 \text{ caso contrário} \end{cases} \quad (4.1)$$

Na Equação 4.1:

- *O* é o número de itens opcionais;
- *N* é o número de veículos na instância;
- *K* é o número de veículos do dia anterior;
- w_i é o peso associado ao item o_i;
- q_i é o tamanho da subsequência a ser analizado e p_i é o máximo de veículos que pode estar nessa subsequência sem que ocorrem pênaltis; e
- *count* é uma função que conta quantos carros na subsequência de j até $j+q_i$ possuem o item opcional instalado.

Quando se avalia o número de trocas de cor, apenas o último carro do dia anterior é levado em conta. A Equação 4.2 define o custo de trocas de cor em uma solução, onde w_c é o peso associado a troca de cor.

$$\text{custo das trocas de cores} = \sum_{j=|K|+1}^{N} \begin{cases} w_c \text{ se cor}(j) \neq \text{cor}(j-1) \\ 0 \text{ caso contrário} \end{cases} \tag{4.2}$$

O custo final da solução é dado pela Equação 4.3

$$Z = (4.1) + (4.2) \tag{4.3}$$

4.1 Entrada

A entrada é composta de quatro arquivos:

optimization_objectives: descreve a prioridade das restrições para a instância. Por exemplo, se o custo de trocar de cor é maior que o custo de violar uma restrição $p : q$;

paint_batch_limit: descreve quantos carros podem ser pintados utilizando a mesma cor numa solução factível;

ratios: definite as restrições $p : q$; e

vehicles: descreve cada veículo requerido no planejamento atual.

Em relação as penalidades para este problema, ROADEF'05 definiu três categorias de penalidades. As restrições do tipo p:q são divididas em dois grupos: *prioritárias* e *não prioritárias*. A diferença entre eles é que uma violação de uma restrição prioritária $p : q$ implica sempre em um pênalti maior. Esta diferença foi adicionado devido à forma como a planta de trabalho Renault realmente funciona. Cada mudança de cor também resultar em uma penalidade. O valor associado com uma penalidade é definido na entrada e representada por uma permutação de $\{10^0, 10^3, 10^6\}$.

Quanto ao arquivo *vehicles* é importante mencionar que há um conjunto K, com $|K| = max(q) - 1$, carros, do dia anterior. Este é o ponto de partida para a contagem das restrições.

4.1.1 Exemplo de entrada e saída

A Tabela 4.1 exemplifica como uma entrada é estruturada. No exemplo, quatro tipos de automóveis devem ser produzidos e podem ter até cinco itens opcionais. O número de veículos necessários de cada tipo está na parte inferior da tabela. Cada item opcional possui sua restrição $p : q$, que limita a quantidade máxima de carros, p, que pode ter esse item opcional em qualquer sequência de tamanho q. As cores foram adicionadas, a fim de tornar mais semelhante ao problema da ROADEF'05 e apenas duas cores foram adicionadas. Um tipo de carro é definido por quais itens opcionais este possui e sua cor. Este exemplo poderia ser os carros solicitados em um dia de planejamento em uma fábrica.

Neste exemplo o custo de violar uma restrição de prioridade é 10^6, uma restrição não prioritária é 10^3 e o custo de uma troca de cor é de R 10^0. O tamanho do lote para este exemplo é de 4. A última coluna da Tabela 4.1 é a prioridade de cada restrição $p : q$. Os valores 1 e 0, respectivamente, significam pesos 10^6 e 10^3 para as restrições, ou seja, definem restrições prioritárias e não prioritárias.

Tabela 4.1: Exemplo de entrada, adaptado do **?**.

Opc.	Tipo de Veículo				Restr.	
	1	2	3	4	$p:q$	Prioridade
1	✓			✓	3:5	1
2		✓			1:3	0
3	✓	✓			2:5	0
4			✓	✓	2:3	1
5			✓		1:4	1
Co r	Preto	Azul	Preto	Azul		
N° req.	2	2	2	2		

Como mencionado anteriormente, neste exemplo, considera-se $|K| = 4$ carros do dia anterior. A sequência destes carros dados na entrada é importante e deve ser mantida. A Tabela 4.2 é a descrição dos últimos quatro carros do dia anterior.

Tabela 4.2: Ultimos quatro carros do dia anterior.

Opc.	Tipo de Veículo			
	1	2	3	4
1	✓			✓
2		✓		
3	✓	✓		
4			✓	
5			✓	✓
Cor	Preto	Azul	Preto	Azul

Uma solução para este problema é uma sequência de carros que não violam o limite do lote. As soluções podem ter custos diferentes dependendo de qual conjunto de p:q foram violadas e o peso de cada p:q que foi violada.

Tabela 4.3: Resultado ótimo detalhado, citado previamente.

Opc.	Carro												Restr.
	1	2	3	4	2	4	3	1	4	2	3	1	$p:q$
1	✓	✓			✓			✓	✓			✓	3:5
2			✓				✓			✓			1:3
3		✓	✓		✓		✓			✓		✓	2:5
4	✓			✓		✓		✓	✓		✓		2:3
5				✓		✓					✓		1:3

Como pode ser visto na Tabela 4.3, a sub-sequência [1, 2, 3, 4, 2] não respeita o limite da terceira restrição. Nesta sub-sequência de tamanho 5, deve haver no máximo dois carros que requerem o terceiro item opcional, no entanto, existem três [2, 3, 2]. A outra violação de restrição neste exemplo é na sub-sequência [3, 4, 2, 4, 3], que também viola a terceira restrição. O custo associado a violar as restrições neste caso é 2×10^3.

A Tabela 4.4 mostra como o custo de trocas de cor é calculado. Note que apenas os carros envolvidos neste cálculo são mostrados, omitindo os três primeiros carros [1, 2, 3] do dia anterior. O custo associado a alterações de cor, neste caso, é de 3×10^0. Portanto, o custo final para esta solução é de 2003.

Tabela 4.4: Troca de Cor

Carro	4	2	4	3	1	4	2	3	1
Cor	Azul	Azul	Azul	Preto	Preto	Blue	Azul	Preto	Preto

4.2 Depth-First Search

O algoritmo *Depth-First Search* (DFS) é amplamente utilizada ao lidar com grafos. **?** define a DFS da seguinte forma "A estratégia seguida pela depth-first search é, como o nome sugere, ir a funs no grafo, sempre que possível" . A ordem de complexidade é de $O(V+E)$, em que V é o número de nós no grafo e E é o número de arestas.

A DFS usa um esquema de cores pra definir quais nós já foram visitados:

Branco: significa que um nó ainda não foi visitado;

Cinza: significa que um nó foi alcançado e os seus vizinhos estão sendo visitados; e

Preto: significa que o nó e todos os seus vizinhos foram visitados.

O Algoritmo 1 é a inicialização de variáveis para a DFS. O método $G.V$ retorna a lista de todos os vértices contidos no grafo G. O método $v.color$ altera ou retrona a cor do vértice v. O segundo laço no Algoritmo 1 é para iniciar a DFS a partir de de todos os vértices do grafo. Isso é necessário no caso de o grafo seja desconexo.

Algoritmo 1: DFS Adaptada de **?**.

```
   Entrada: Graph G
1  para v ∈ G.V hacer
2  |  v.color ← White;
3  |  v.π ← Null;
4  fin
5  para v ∈ G.V hacer
6  |  se v.color = White então
7  |  |  DFS-Visit(G, v);
8  |  fim
9  fin
```

O Algoritmo 2 marca um vértice v como cinza, uma vez que começa o procedimento. Depois para cada vértice que existe uma aresta de v e ainda são marcados como branco, o *DFS-Visit* é chamado de forma recursiva. Depois de todos os vértices da vizinhança do vértice v forem visitados, o vértice v é então marcada como preto.

A Figura 4.1 exemplifica uma execução da DFS: o grafo possui 5 vértices e 5 arestas. Conforme descrito antes, um vértice marcado em preto significa que todos os vizinhos do vértice foram visitados. A borda mais grossa significa que a aresta já foi atravessada. A partir do vértice 1 a DFS em seguida, vai para o vértice 2, e assim por diante. Vertex 1 só se torna preto após todos os outros vértices terem visitados.

Algoritmo 2: DFS-Visit Adaptado de **?**.

Entrada: Grafo G, Vértice v

```
1 v.color ← Gray;
2 para u ∈ G.ADJ[v] hacer
3     se u.color = White então
4         u.π ← v;
5         DFS-Visit(G, u);
6     fim
7 fin
8 v.color ← Black;
```

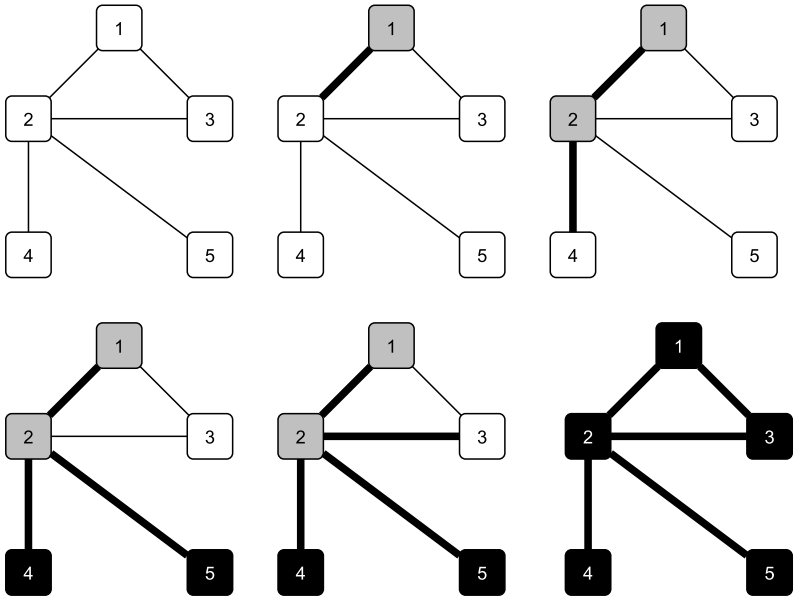

Figura 4.1: Execução da DFS.

4.2.1 Best Insertion

O algoritmo *Best Insertion* (BI) é uma técnica gulosa simples para resolver problemas permutacionais que normalmente escolhem um elemento de solução de forma aleatória e move este elemento para outra posição da solução. Esta nova posição é determinada pela posição onde o valor da solução é o melhor possível.

No contexto Problema de Sequenciamento de Veículos (CS) o *Best Insertion* remove um carro de uma posição calcula como as restrições estariam nas outras posições que este carro pode ser inserido. Este é um processo pesado e deve ser otimizado.

4.3 Iterated Local Search

? descreveram as principais etapas do *Iterated Local Search* (ILS). Esta técnica, o ILS, atua sobre uma solução inicial, normalmente uma solução gerada por um método guloso. Depois, o ILS executa um procedimento de busca local, a fim de encontrar um ótimo local. Este primeiro resultado é considerado como o melhor inicialmente. Em seguida, o ILS aplica uma perturbação

(isto é, uma modificação aleatória) à solução atual e a armazena como uma solução candidata. Esta solução candidata passar por diferentes procedimentos de busca local, a fim de conseguir um ótima local em cada busca local. A solução resultante candidata é comparada e, em caso de que o critério de aceitação aceite este candidato, o ILS atualiza a melhor solução. Este processo é repetido até que um critério de parada for satisfeito, retornando a melhor solução encontrada até o momento.

O Algoritmo 3 apresenta as operações básicas do ILS, onde S_{melhor} é a melhor solução encontrada até o momento, $S_{candidate}$ é a solução que está sendo alterada, a fim de tentar encontrar uma solução melhor e α é um número que indica o percentual da solução a ser alterado durante a perturbação.

Algoritmo 3: ILS

```
   Entrada: double α
1  S_best ← InitialSolution();
2  S_best ← LocalSearch(S_best);
3  repita
4  |  S_candidate ← Perturbation(S_best, α);
5  |  S_candidate ← LocalSearch(S_candidate);
6  |  se Critério de aceitação(S_candidate, S_best) então
7  |  |  S_best ← S_candidate;
8  |  fim
9  até critério de parada ser atingido;
10 retorna S_best
```

A busca local, perturbação e critério de aceitação são frequentemente tratados como algoritmos caixa preta e dependentes problema. Além disso, a intensidade da perturbação requer testes, a fim de definir o que é melhor para o problema.

O Algoritmo 4 retrata um exemplo de busca local. Até o critério de parada ser atingido, o valor da solução atual é armazenado na variável *S*', e aplica-se um movimento, isto é a posição de dois elementos são trocadas na solução. Se o resultado for pelo menos igual ao resultado anterior o movimento é aceito, caso contrário, o movimento é desfeito. Ao final, o procedimento retorna um mínimo local.

Algoritmo 4: Local Search

```
   Entrada: Solution s
1  repita
2  |  s' ← s;
3  |  Movement(s);
4  |  se evaluate(s) > evaluate(s') então
5  |  |  UndoMovement(s);
6  |  fim
7  até critério de parada ser atingido;
8  retorna s
```

A Figura 4.2 mostra o que o algoritmo de perturbação deve fazer em um problema de minimização. A perturbação deve diferenciar a solução o suficiente para que um ótimo local diferente possa ser explorado pelos algoritmos de busca local. A partir de uma solução de s^* , para

encontrar um ótimo local mínimo diferente de s^*', a perturbação deve modificar a solução atual e ir para outra região do espaço de busca, digamos, s'. Em seguida, um procedimento de busca local é responsável por refinar esta nova solução até que o novo ótimo local seja encontrado.

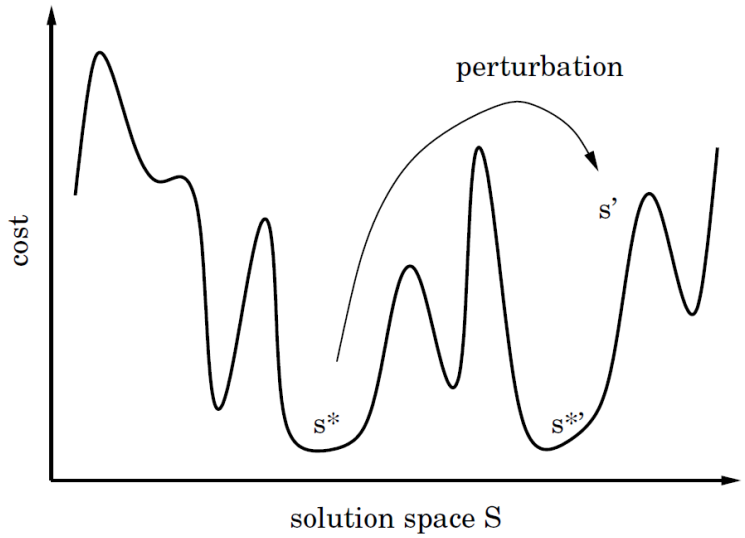

Figura 4.2: Gráfico de perturbação, retirado de **?**.

O critério de aceitação em um problema de *minimização* pode estar apenas avaliando se $S_{candidate}$ é válido e comparando os custos de ambos, $S_{candidate}$ e S_{melhor}, embora muitos outros critérios possam ser adotados para esse fim.

Analogamente, o critério de parada adotado pode variar. Por exemplo, pode ser o número máximo de iterações, o tempo máximo de funcionamento, etc.

Capítulo 5

Desenvolvimento

Este capítulo descreve as implementações feitas, em detalhes, como a modelagem computacional e de representação, a nova heurística construtiva e a aplicação do método ILS.

5.1 Representação Computacional

Existem duas estruturas principais para a representação computacional deste problema. O primeiro é a representação de dados e o segundo é a representação da solução. Há também estruturas auxiliares para representar pequenas entidades do problema.

A representação de dados é trivial. A representação da solução utiliza três matrizes para armazenar a lista de veículos, que veículo foi sequenciado, e os valores associados a cada restrição $p:q$.

A matriz de veículos contém os índices dos veículos dos dados de entrada atualmente na solução. A matriz de veículos sequenciados visitados contém as informações sobre se um veículo já foi incluído na solução ou não. A terceira matriz armazena os valores de cada restrição $p:q$ após o último veículo ter sido adicionado, ou não, à solução.

A representação da solução também usa três matrizes usadas para avaliar movimentos de troca mais rápidamente. **?** desenvolveu uma técnica para avaliar um movimento de troca no tempo $O(m)$, onde m é o número de restrições $p:q$. Esta técnica utiliza três matrizes, M_1, M_2 e M_3. Essas matrizes representam restrições $p:q$ como linhas e os veículos como colunas.

Matriz M_1: armazena na linha i e coluna j, o número de veículos, começando na posição j na sequência da solução terminando na posição $j+q$, que possuem o item opcional i.

Matriz M_2: armazena na linha i e coluna j, o número de sub-sequências começando na coluna 1 na matriz M_1 i-ésima linha e terminando na coluna j que têm mais de p veículos em uma sub-sequência de tamanho q. Isto é, o número de cada restrição $p:q$ que viola as sub-sequências.

Matriz M_3: que é semelhante a M_2, armazena na linha i, coluna j, o número de sub-sequências começando na coluna 1 na matriz M_1 i-ésima linha e terminando na coluna j que tem pelo menos p veículos em uma sub-sequência de tamanho q.

Por exemplo, considere que há dez veículos a serem produzidos e apenas uma restrição *p:q* com $p = 1$ e $q = 4$. Dado que cinco em cada dez veículos exigem esta opção Figura 5.1 mostra como os carros são sequenciados e como as matrizes seriam. A primeira posição da matriz M_1 conta os veículos da posição 1 para a posição 4, totalizando três carros. As outras posições na

matriz M_1 apenas desliza a janela de quatro carros. Uma vez que existem mais do que *p=1* veículos na primeira posição de M_1, a primeira posição de M_2 possui o valor um. Novamente, uma vez que existe pelo menos *p=1* carro na primeira posição de M_1, a primeira posição de M_3 possui o valor um.

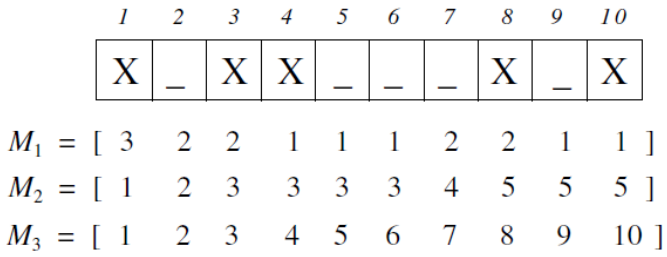

Figura 5.1: Matrizes do exemplo M_1, M_2 e M_3. Fonte: **?**

Vamos ilustrar a utilização destas matrizes em um movimento de troca, tal como descrito em **?**. Suponha que os carros nas posições a e b foram para trocar posições em uma solução. Ao utilizar matrizes M_2 e M_3 é possível avaliar rapidamente o custo do movimento. Equação 5.1 mostra a diminuição do número de violações associadas com a restrição de relação q_j.

$$\Delta_j^1 = \begin{cases} M_2[a][j] - M_2[a-q_j][j], & \text{se } a-q_j > 0 \\ M_2[a][j], & \text{caso contrário} \end{cases} \tag{5.1}$$

Equação 5.2 mostra o aumento do número de violações associadas com a restrição de relação q_j.

$$\Delta_j^2 = \begin{cases} M_2[b][j] - M_2[b-q_j][j], & \text{se } b-q_j > 0 \\ M_2[b][j], & \text{caso contrário} \end{cases} \tag{5.2}$$

Ambas as equações, 5.1 e 5.2, avaliam apenas um item opcional. O custo do movimento de troca é definido por $\Delta_j = \Delta_j^2 - \Delta_j^1$. Os custos totais de movimento (ou seja, considerando as restrições de relação de todos os itens opcionais) é então calculada pela Equação 5.3, onde m é o número de itens opcionais.

$$\sum_{j=1}^{m} \Delta_j \tag{5.3}$$

5.2 Modelagem do Problema

Neste trabalho, o problema foi modelado por meio de grafos, em que cada nó v representa um veículo e uma aresta conecta dois veículos v e w se o veículo w pode ser sequenciado após o veículo v, como pode ser visto na Figura 5.2. Este grafo corresponde ao exemplo dado na Tabela 4.1. O nó amarelo representa os veículos do dia anterior, enquanto os nós azuis representam os veículos sendo sequenciados.

Para sequenciar os veículos pelo do grafo, seria necessário determinar um caminho Hamiltoniano , uma vez que cada veículo precisa estar na sequência. No entanto, o problema de decisão associado é $\mathcal{NP}$-Completo. Acrescentar pesos às arestas modela o problema de forma mais realista. Por exemplo, isto pode representar o custo de mudança de cor entre os veículos.

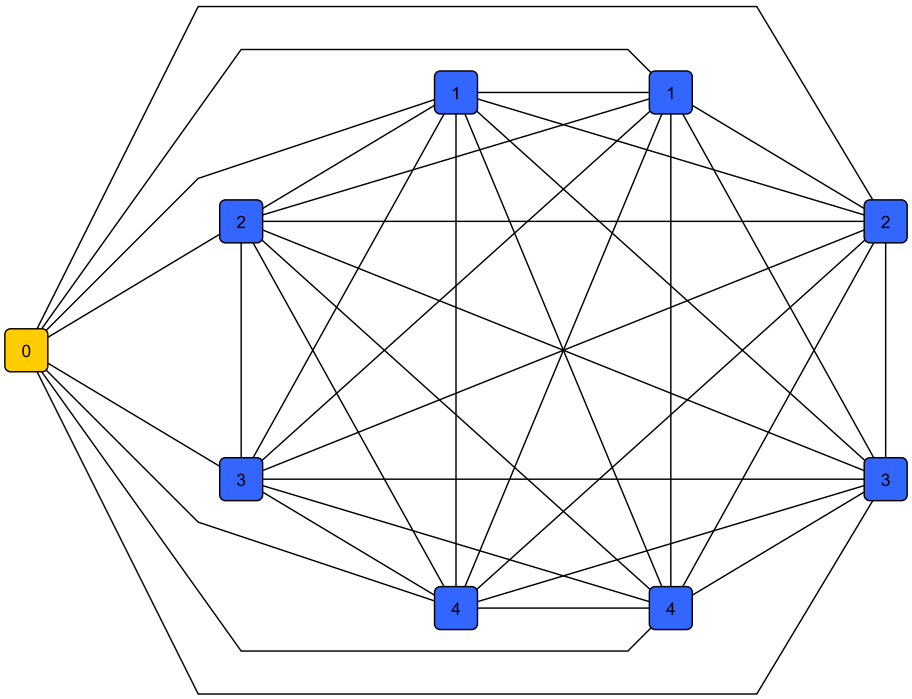

Figura 5.2: Grafo de exemplo a respeito da Tabela 4.1.

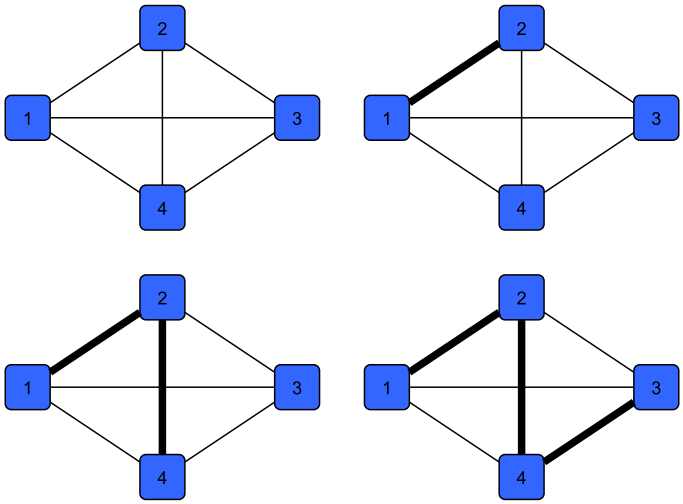

Figura 5.3: Caminho Hamiltonian.

Figura 5.3 é uma ilustração de um algoritmo de detecção de caminho Hamiltoniano. Neste caso o vértice 1 é inicial. O caminho hamiltoniano encontrado é [1, 2, 4, 3].

Uma maneira mais rápida, ainda que não ótima, para encontrar uma solução seria realizar um DFS, inserindo os vértices na solução à medida que são exploradas. Isso não garante encontrar uma boa solução, ou mesmo viável, desde que as arestas levam em consideração apenas se o próximo veículo pode ser sequenciado após o atual. No entanto, isso pode ser um bom começo para um método heurístico, dando bons *insights* sobre a solução.

5.3 Uma Nova Heurística Gulosa

O método heurístico desenvolvido é uma combinação do *Depth-First Search* (DFS) com o *Best Insertion* (BI), ambos descritos anteriormente no Capítulo 4.

A DFS em primeiro lugar produz uma lista de candidatos a cada iteração, carros os quais ainda não estão na solução. Depois que a lista é gerada a DFS seleciona um candidato aleatoriamente entre os carros com o menor custo. O método itera até que nenhum candidato possível esteja disponível: ou todos os carros estão na solução ou não há nenhum carro atual que é possível ser inserido.

O cálculo dos pesos e seleção de candidatos da DFS é descrito a seguir. Os pesos das arestas é dinâmico neste problema. A figura 5.4 é um *snapshot* de como este conceito funciona. O vértice 0 foi incluído na solução, que representa os carros do dia anterior. A lista de candidatos tem quatro carros: adicionando o carro 1 aumentaria o custo da solução em 1000, adicionando o carro 2 aumentaria o custo em 100 e assim por diante. Quando houver um empate entre mais de um carro, o algoritmo escolhe um aleatoriamente.

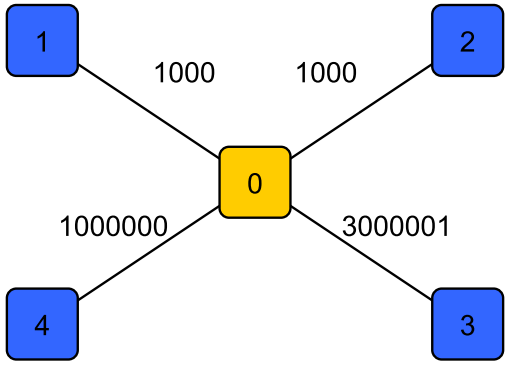

Figura 5.4: Peso antes da primeira inserção.

Porque isso implica no aumento o menor valor de solução, o carro 2 é escolhido aleatoriamente e adicionou-se à solução. Portanto, os pesos para inserir o restante dos carros mudam conforme pode ser visto na Figura 5.5. O custo para adicionar carro 4 para a solução agora é 0, o custo para adicionar carro 3 é 1000001, e assim por diante. A DFS iria continuar este processo até não haver mais carros a serem inseridos na solução.

A DFS é um algoritmo simples, uma vez que não realiza *backtrackings*. Algumas vezes é possível que ainda existam veículos restantes, mas a solução não seria viável, se qualquer um desses veículos fossem inseridos. Outro problema que pode ocorrer é a penalidade associada para inserir um veículo na solução é muito grande e, de modo a evitar este aumento no custo, este veículo deve ser inserido em outra posição da solução. Nestes casos, é interessante utilizar o algoritmo *Best Insertion*, conforme descrito no Capítulo 4.

A *Best Insertion* é executado cada vez que a adição de um carro na solução aumenta o custo da solução. A Figura 5.6 exemplifica como o algoritmo de BI funciona. Os quatro primeiros carros (em amarelo) são do dia anterior. Os três seguintes (em azul) são carros do dia atual e não tem uma penalidade associada. Ao adicionar o oitavo carro (vermelho) na solução gera-se um pênalti. Para evitar este pênalti, o algoritmo remove este carro e simula a sua inserção entre os carros anteriores, após do último veículo do dia anterior (i.e., ultimo carro amarelo). Em cada uma dessas posições, a solução é avaliada e a melhor posição é determinada com base no melhor

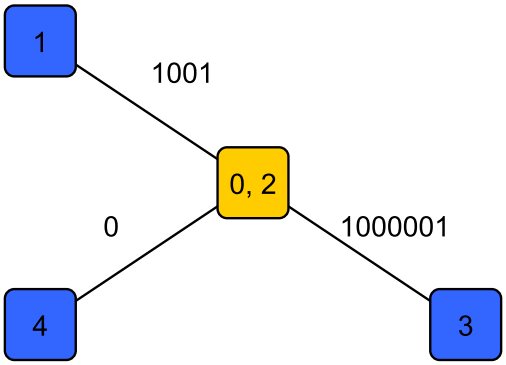

Figura 5.5: Pesos depois da primeira inserção.

valor da solução (isto é, nenhuma penalidade menor ou um número menor de penalidades). Na ilustração a seguir, a melhor posição é entre carros 202 e 203.

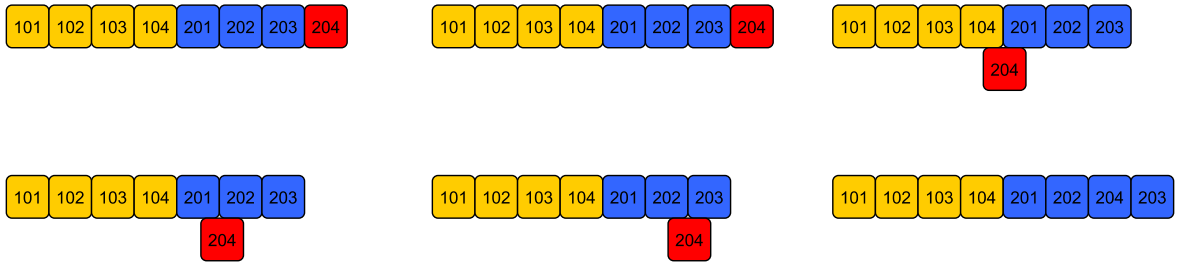

Figura 5.6: Exemplo de execução do BI.

Esta técnica utiliza a matriz descrita anteriormente, M_1, para pular alguns passos à frente e acelerar o processo. Por exemplo, se um carro seria inserido na posição i e isso implicaria uma penalidade relacionada a um $p:q$ com $q = 10$ e, em seguida, que a pena se estenderia até a posição $i+10$. Não há necessidade de verificar as posições de $i+1$ até $i+10$, de modo que o BI em vez de inserir o carro na próxima posição ele é inserido diretamente na posição $i+11$.

5.4 Iterated Local Search

? propôs as seguintes buscas locais para a solução do Problema de Sequenciamento de Veículos. Todos os carros (ou grupo de carros) são escolhidos aleatoriamente.

Car Exchange **:** troca a posição de dois veículos;

Car Insertion **:** remove um veículo de uma posição e insere em uma nova posição;

Reinsertion **:** remove um número aleatório de veículos, entre dois e oito, e insere-os novamente em outra posição usando o *Best Insertion*;

Group Exchange **:** troca um grupo contíguo de veículos com outro grupo; e

***Group Inversion* :** inverte a ordem dos veículos em uma sequência contígua desde que todos possuam a mesma cor.

No entanto, após alguns testes, constatamos que apenas *Car Exchange, Reinsertion* e *Group Inversion* produziram bons resultados. Então foi decidido que seriam usados apenas estes métodos como busca local.

Group Exchange pode não ser bom devido ao fato de que ele requer grupos de carros com a mesma cor, e nas instâncias do ROADEF, o número de cores é relativamente pequeno se comparado com o número de carros. Por isso, os grupos trocados ou eram muito diferente, ou muito similares. No caso de os grupos serem muito diferentes, este procedimento de busca local pode levar a um potencial aumento no número de violações. No caso de os grupos serem muito semelhantes, o custo da solução não seria alterado.

Car Insertion pode não gerar boas soluções devido ao fato de que a substituição de apenas um veículo pode não ser suficiente para realmente melhorar a solução. Além disso, o tempo de processamento deste movimento é muito alta quando comparada com os outros. *Reinsertion* também tem um elevado tempo computacional, no entanto, uma vez que altera mais veículos ele pode gerar soluções melhores.

O Algoritmo 5 mostra a implementação do *Iterated Local Search*. A DFS foi usada como a geração da solução inicial, devido ao seu tempo de processamento. Com base nos testes preliminares realizados, o melhor valor para a perturbação encontrado foi $\alpha = 0,2$, ou seja, 20% dos elementos da solução passará por movimentos de troca aleatórios.

Algoritmo 5: Implementação do ILS

Entrada: Graph G, double α

```
1  S_best ← DFS(G);
2  repita
3  |   S_candidate ← Perturbation(S_best, α);
4  |   S_candidate ← GroupInversion(G, S_candidate);
5  |   S_candidate ← Reinsertion(G, S_candidate);
6  |   S_candidate ← CarExchange(G, S_candidate);
7  |   se Critério de aceitação(S_candidate, S_best) então
8  |   |   S_best ← S_candidate;
9  |   fim
10 até critério de parada ser atingido;
11 retorna S_best
```

Capítulo 6

Resultados e Discussão

Neste capítulo são descritos o ambiente computacional, onde os experimentos foram executados, as instâncias e seus principais atributos. Os experimentos realizados e os resultados obtidos pelos métodos serão descritos no Capítulo 5.

6.1 Ambiente Computacional

O ambiente computacional utilizado nos experimentos é composto por um processador Intel® Core™ i7-4790 CPU @ 3.60 GHz, 16 GB de memória RAM e sistema operacional Ubuntu 14.04.4 LTS.

6.2 Instâncias

A tabela 6.1 mostra os detalhes para o conjunto de instâncias X retiradas do site do desafio ROADEF'05[1]. A tabela é ordenada, crescentemente, pelo número de carros em cada caso. N° Carros e o número de carros no caso; N° p:q é o número de restrições $p : q$ na determinada instância e *Color Batch* é o tamanho do lote de cor para a instância. As próximas três colunas mostram os valores da penalidade para cada instância. Em cima dessas colunas está o tipo de restrições: Restrição de Alta Prioridade (RAP), Restrição de Baixa Prioridade (RBP) e Troca de Cor (TC). Em cada linha, para estas colunas, é apresentado o peso associado, um valor do conjunto $\{10^6, 10^3, 10^0\}$, como mencionado no Capítulo 4.

[1] http://challenge.roadef.org/2005/en/sujet.php Acessado em 20/07/2016

Tabela 6.1: Características das instâncias do ROADEF.

Nome	N° Carros	N° Cores	N° p:q	Color Batch	RAP	RBP	TC
BATILLY_CH2	78	5	6	100	10^6	10^3	10^0
DACIA_CH1	91	6	1	1000	10^3	10^0	10^6
CORDOBA_CH2	234	7	4	10	10^6	10^0	10^3
CURITIBA_VU	252	7	8	30	10^6	10^0	10^3
CORDOBA_CH1	278	9	5	10	10^6	10^0	10^3
FLINS_CH2	372	12	6	15	10^6	10^0	10^3
DACIA_CH2	377	7	2	1000	10^3	10^0	10^6
BATILLY_CH1	460	15	26	100	10^6	10^3	10^0
SANDOUVILLE_CH2	508	13	20	12	10^6	10^0	10^3
REVOZ	718	13	12	200	10^3	10^0	10^6
SANDOUVILLE_CH1	792	13	22	12	10^6	10^0	10^3
BURSA	822	14	7	60	10^6	10^0	10^3
FLINS_CH1	974	13	11	15	10^6	10^3	10^0
CURITIBA_VP	985	10	8	400	10^6	10^0	10^3
MAUBEUGE	1071	20	20	60	10^6	10^3	10^0
VALLADOLID	1279	13	12	40	10^6	10^0	10^3
DOUAI_CH3	1283	17	12	20	10^6	10^0	10^3
PALENCIA	1338	15	18	10	10^6	10^0	10^3
DOUAI_CH1	1543	15	12	20	10^6	10^0	10^3

A Tabela 6.2 mostra os valores das restrições $p : q$ em cada caso. Estes são divididos em Restrição de Alta Prioridade e Restrição de Baixa Prioridade. A fim de resumir os dados, várias restrições foram agrupadas. Por exemplo, o RBP de exemplo Batilly_CH2 é de 1:(2, 3, 5, 14) significa que há pelo menos uma restrição de cada 1:2, 1:3, 1:5, 1:14. Para alguns casos, talvez haja mais do que uma restrição com o mesmo valor, no entanto, valores repetidos não são exibidos. O símbolo '-' denota a ausência do tipo de restrição.

Tabela 6.2: Características das instâncias do ROADEF, continuação.

Nome	RAP	RBP
BATILLY_CH2	1:6	1:(2, 3, 5, 14)
DACIA_CH1	1:2	-
CORDOBA_CH2	1:(6, 9, 16), 2:3	-
CURITIBA_VU	1:(2, 4, 17, 22), 3:5	1:(5, 13)
CORDOBA_CH1	1:(2, 15)	1:5
FLINS_CH2	1:(2, 5, 20, 100), 8:10	-
DACIA_CH2	1:2	-
BATILLY_CH1	1:(2, 6, 7, 8, 10) 1:(12, 20, 25)	1:(3, 4, 8, 10, 12, 13, 16, 18, 19), 1:(39, 91, 136), 3:(4, 5), 4:5
SANDOUVILLE_CH2	1:(2, 3, 4, 6, 8, 10)	1:(3, 4, 5, 6, 7, 9, 20, 25, 50)
REVOZ	1:3, 5:6	1:(3, 4, 5, 6, 10), 2:3, 10:15
SANDOUVILLE_CH1	1:(2, 3, 4), 2:3	1:(1, 3, 4, 5, 6, 7, 8, 9, 10), 1:(18, 137, 274), 2:3
BURSA	1:(6, 7), 3:4	1:(2, 3), 39:43
FLINS_CH1	1:(2, 4, 5, 8, 10, 30, 100), 2:3, 4:9	1:(5, 9)
CURITIBA_VP	1:(2, 5, 29)	1:(3, 6,7, 10, 65)
MAUBEUGE	1:(3, 4, 10), 2:3	1:(2, 3, 6, 7, 9, 18, 20, 28, 76), 2:3, 5:8, 17:30
VALLADOLID	1:(6, 7, 10, 20)	1:(2, 3, 5, 8, 10)
DOUAI_CH3	1:(3, 5)	1:(8, 14, 16, 19, 25, 38, 47, 247), 12:60, 41:60
PALENCIA	1:(2, 4, 13), 3:5, 6:7	1:(3, 6, 7, 10, 15, 20)
DOUAI_CH1	1:20	1:(2, 4, 6, 7, 8, 13, 19, 56, 297)

6.3 Resultados

As próximas três tabelas de dados são apresentadas na seguinte norma: a coluna *BKS* apresenta as melhores soluções conhecidas, tomadas a partir do site do ROADEF[2]; a coluna S^* apresenta a melhor solução encontrada pelos métodos propostos, S é o valor médio das soluções sobre todas as execuções independentes para cada método, σ é o desvio padrão entre os valores da

[2]http://roadef.proj.info-ufr.univ-montp2.fr/2005/ Acessado em 20/06/2016

solução e T é o tempo de execução, expresso em segundos. A coluna *gap* apresenta a distância percentual entre o BKS e S^*, calculado da seguinte forma $100 \times (S^* - BKS)/BKS$.

As tabelas 6.3 e 6.4 apresentam os resultados do *Depth-First Search* (DFS), *Depth-First Search* com *Best Insertion* (DFS+BI), para cada instância. Os métodos DFS e DFS+BI foram executados cada um, 10 vezes independentemente. A tabela 6.5 apresenta os resultados do *Iterated Local Search* (ILS), que foi executada 5 vezes independentemente.

Tabela 6.3: Resultados DFS.

Instância	BKS	gap(%)	S^*	S	σ	T
BATILLY_CH2	3.0	200033.33	6004	5306404.3	3552688.7	0.01
DACIA_CH1	5010000.0	0.22	5021000	5023300.0	1791.6	0.01
CORDOBA_CH2	153034000.0	56.20	239033000	267735000.0	20885487.9	0.06
CURITIBA_VU	55590.0	510859.12	284042175	329841165.3	26988456.6	0.07
CORDOBA_CH1	30000.0	0.00	30000	430000.0	800000.0	0.06
FLINS_CH2	37000.0	113508.11	42035000	60637300.0	12899288.8	0.08
DACIA_CH2	6056000.0	0.45	6083000	6094200.0	8121.6	0.13
BATILLY_CH1	36341495.4	831.23	338424075	400527277.4	30888441.1	0.25
SANDOUVILLE_CH2	31077118.0	913.88	315085345	364580895.2	26965211.7	0.29
REVOZ	12002003.0	0.33	12042122	12047848.2	5994.0	0.56
SANDOUVILLE_CH1	196971.0	126398.10	249164570	299750254.9	28211898.5	0.50
BURSA	110069.0	22.19	134488	31728868.6	32461050.8	0.82
FLINS_CH1	61183030.0	1077.03	720141436	773043309.1	45249267.7	1.00
CURITIBA_VP	8087035.0	3795.49	315029403	340428472.7	19540292.7	0.93
MAUBEUGE	160365.0	489239.55	784729372	866365470.4	60179793.2	2.02
VALLADOLID	189075.0	542058.59	1025086362	1048893996.6	15321570.7	1.91
DOUAI_CH3	231030.0	176534.32	408078274	429278735.8	13990969.8	1.25
PALENCIA	328006.0	261279.42	857340191	949623222.9	63726323.9	2.54
DOUAI_CH1	69237.0	2.41	70904	71052.9	133.7	1.82

Tabela 6.4: Resultados DFS+BI.

Instância	BKS	gap(%)	S^*	S	σ	T
BATILLY_CH2	3.0	133400.00	4005	5202404.4	2891062.5	0.00
DACIA_CH1	5010000.0	0.22	5021000	5023100.0	1445.7	0.01
CORDOBA_CH2	153034000.0	60.14	245074000	311771100.0	28382555.8	1.82
CURITIBA_VU	55590.0	518211.09	288129137	309630436.8	13529778.4	26.84
CORDOBA_CH1	30000.0	450.00	165000	979400.5	1328088.4	53.93
FLINS_CH2	37000.0	130140.54	48189000	55490200.0	6528131.2	249.74
DACIA_CH2	6056000.0	0.46	6084000	6091700.0	2968.2	0.10
BATILLY_CH1	36341495.4	1001.66	400358062	431977363.9	23728979.7	105.82
SANDOUVILLE_CH2	31077118.0	991.73	339279584	391048178.0	22606966.8	551.51
REVOZ	12002003.0	0.33	12042122	12547747.7	670093.6	0.46
SANDOUVILLE_CH1	196971.0	192019.78	378420254	407408561.5	16225939.1	4420.89
BURSA	110069.0	85784.88	94532631	144135738.4	28539632.9	904.29
FLINS_CH1	61183030.0	2043.79	1311633430	1350933969.7	26348570.9	3640.96
CURITIBA_VP	8087035.0	4475.91	370055478	387166314.0	13108359.2	1084.60
MAUBEUGE	160365.0	604873.25	970165358	1061728486.6	54671658.9	3218.19
VALLADOLID	189075.0	481655.12	910878484	959172872.3	29373981.6	8727.04
DOUAI_CH3	231030.0	157785.02	364761757	376963609.6	8634509.2	13679.40
PALENCIA	328006.0	439207.00	1440953323	1520660372.9	56856421.7	10415.80
DOUAI_CH1	69237.0	1371.27	1018664	1031958.3	8820.6	44145.30

Tabela 6.5: Resultados ILS.

Instância	BKS	gap(%)	S^*	S	σ	T
BATILLY_CH2	3.0	0.00	3	3.4	0.5	1800.00
DACIA_CH1	5010000.0	0.18	5019000	5023200.0	3059.4	1800.00
CORDOBA_CH2	153034000.0	0.01	153045000	153045600.0	489.9	1807.60
CURITIBA_VU	55590.0	37872.73	21109039	23312651.0	1940354.2	1812.60
CORDOBA_CH1	30000.0	0.00	30000	33800.0	4915.3	1819.00
FLINS_CH2	37000.0	191.89	108000	111200.0	2481.9	1990.00
DACIA_CH2	6056000.0	0.51	6087000	6093600.0	4176.1	1802.00
BATILLY_CH1	36341495.4	290.96	142080131	154500927.8	8269657.9	2030.59
SANDOUVILLE_CH2	31077118.0	187.28	89279583	95673764.0	3437417.2	2097.79
REVOZ	12002003.0	0.29	12037165	12046746.8	6619.1	2062.87
SANDOUVILLE_CH1	196971.0	79304.25	156403340	166027837.2	5758979.3	3914.80
BURSA	110069.0	23.05	135440	306836.6	86097.5	1977.59
FLINS_CH1	61183030.0	379.96	293652290	302445135.2	7138626.2	2604.59
CURITIBA_VP	8087035.0	648.22	60508933	65085308.0	3430398.2	2399.19
MAUBEUGE	160365.0	194055.62	311357658	330927890.6	10745414.1	2858.39
VALLADOLID	189075.0	76469.77	144774302	157970707.0	8631611.3	2808.59
DOUAI_CH3	231030.0	57824.35	133822636	139217630.6	5270829.9	7388.98
PALENCIA	328006.0	141652.35	464956229	477967860.8	7771664.2	3167.39
DOUAI_CH1	69237.0	2.57	71014	71089.4	65.5	12400.80

Com base nos resultados apresentados na Tabela 6.3, é possível ver que a DFS é um método rápido, normalmente não requer mais do que três segundos para executar e pode ser usado como um gerador de solução inicial. No entanto, o valor da solução é elevado e tem um grande desvio padrão. Por exemplo Batilly_CH2 o valor de *BKS* é 3 e o melhor resultado encontrado pela DFS é 6004. Apesar de o valor real ser 6004, isso realmente significa que apenas sete penalidades adicionais são geradas (seis de valor 1000 e uma do valor 1). É importante notar que, apesar de ser um método rápido, a DFS deve ser executada mais de uma vez, de modo a ter uma solução inicial mais interessante, devido ao fato de que contém aleatoriedade.

De acordo com a Tabela 6.4, o algoritmo de *best insertion*, algumas vezes não é tão eficiente como se espera: aproxima-se do melhor resultado conhecido, no entanto, o tempo de execução requerido para resolver uma instância pode ser muito maior. Por exemplo, na instância Sandouville_CH1 a DFS exige apenas 0,48 segundos de execução, em média, enquanto a DFS+BI exige 4420.89 segundos em média. Isto é devido ao fato de que ótimos locais deste problema, muitas vezes são piores do que o ótimo global, ou seja, DFS+BI pode inserir um carro em uma posição que, no contexto local parece ser a decisão certa, mas no contexto global este carro deve estar em posição completamente diferente.

A baixa qualidade das decisões locais é explicado pela natureza das restrições $p : q$. Estas restrições são cumulativas na solução, no entanto, eles também são renováveis, isto é, a cada q carros na solução o valor p é zero e começa a acumular novamente. Assim, em uma solução parcial $p : q$ valor de restrição não nos dá uma visão sobre a estrutura da solução completa. Quanto mais diferentes as restrições $p : q$, menos preciso são os movimentos locais.

O tempo de execução do método ILS é definido em 1800 segundos. Em alguns casos (DOUAI_CH1, DOUAI_CH3, Palencia, etc.) o tempo de execução é maior, isto se dá pelo fato de que o BI algumas vezes leva mais tempo para processar. Em relação ao valor da função objetivo, o custo ainda varia de execução a execução, mas menos do que a DFS e DFS+BI. O ILS também é capaz de obter resultados mais próximos dos melhores resultados conhecidos. Em alguns casos, ILS é capaz de atingir melhor resultado conhecido. Outros casos, como CURITIBA_VU, MAUBEUGE e Palencia podem haver altos valores de *gap* devido ao fato de que são altamente restritos e ILS não foi capaz de fazer mudanças suficientes na solução a cada etapa.

A Tabela 6.6 apresenta o melhor resultado de cada método para cada instância e o *ranking* de todos os três métodos. A coluna *Resultado* é o melhor resultado encontrado por cada método para cada instância e a coluna *Ranking* apresenta o *ranking* ordinal que cada método tem em cada caso, quando comparado com os outros métodos. O *ranking* ordinal atribui um valor para cada método de acordo com o seu desempenho: ao melhor método o valor 1 é atribuído; para o segundo melhor método, o valor 2 é atribuído e assim por diante. No caso de empates, o mesmo número é atribuído a ambos os métodos. Isto dá-nos uma maneira de comparar diretamente o desempenho dos métodos.

Como pode ser visto na Tabela 6.6, ILS claramente é melhor do que DFS e DFS+BI: em média, o ILS tem um *ranking* médio igual a 1,21. Nos casos em que o ILS não pôde obter a melhor solução, ele está bem perto da melhor. Por exemplo na instância DACIA_CH2, o ILS levou três pênaltis de 10^3 a mais do que a DFS. Outro exemplo na instância BURSA, ILS gerou 48 pênaltis de 10^0 a menos que o DFS mas gerou um pênalti de 10^3. Na instância DOUAI_CH1, o ILS gerou 890 pênaltis de 10^0 a menos do que a DFS mas gerou um pênalti de 10^3 a mais.

Tabela 6.6: Ranking dos três métodos, DFS, DFS+BI and ILS.

Instância	Resultado			Ranking		
	DFS	DFS+BI	ILS	DFS	DFS+BI	ILS
BATILLY_CH2	6004	4005	3	3	2	1
DACIA_CH1	5021000	5021000	5019000	3	3	1
CORDOBA_CH2	239033000	245074000	153045000	2	3	1
CURITIBA_VU	284042175	288129137	21109039	2	3	1
CORDOBA_CH1	30000	165000	30000	1	3	1
FLINS_CH2	42035000	48189000	108000	2	3	1
DACIA_CH2	6083000	6084000	6087000	1	2	3
BATILLY_CH1	338424075	400358062	142080131	2	3	1
SANDOUVILLE_CH2	315085345	339279584	89279583	2	3	1
REVOZ	12042122	12042122	12037165	3	3	1
SANDOUVILLE_CH1	249164570	378420254	156403340	2	3	1
BURSA	134488	94532631	135440	1	3	2
FLINS_CH1	720141436	1311633430	293652290	2	3	1
CURITIBA_VP	315029403	370055478	60508933	2	3	1
MAUBEUGE	784729372	970165358	311357658	2	3	1
VALLADOLID	1025086362	910878484	144774302	3	2	1
DOUAI_CH3	408078274	364761757	133822636	3	2	1
PALENCIA	857340191	1440953323	464956229	2	3	1
DOUAI_CH1	70904	1018664	71014	1	3	2
			Average Ranking:	2.05	2.79	1.21

Capítulo 7

Conclusões

O Problema de Sequenciamento de Veículos mostrou-se ser um problema interessante, em termos de dificuldade e do que pode ser desenvolvido. A revisão da literatura foi feita a partir do ano de 1986, quando o problema foi proposto pela primeira vez, até 2016. Foram desenvolvidos três métodos para resolver este problema: *Depth-First Search* (DFS), *Depth-First Search* com *Best Insertion* (DFS+BI) e *Iterated Local Search* (ILS). Em relação ao ILS, cinco principais buscas locais foram implementadas, mas apenas três mostraram que poderiam melhorar o resultado.

Uma surpresa que surge é que o desempenho do DFS+BI foi pior do que o desempenho da DFS, tanto em valor da solução quanto em tempo de execução. Em relação ao resultado, isto pode ser causado pelo fato de que, quando o BI está em execução, carros são inseridos no que parece ser melhor numa outra posição, no entanto, a posição original era, de fato, melhor. Quanto ao tempo de execução, o BI é comparativamente mais lento, porque precisa percorrer todos os carros até agora selecionados a partir do dia em curso e este é um processo computacionalmente pesado.

Ao comparar DFS, DFS+BI e ILS com outros autores usando o site[1] do ROADEF, esses três métodos não são capazes de alcançar uma boa classificação. Com uma classificação global de 33, com uma pontuação de -0.3666 enquanto o melhor autor tem uma pontuação de $1,0122$, é nítido que ainda há melhorias a serem feitas com o ILS.

Trabalhos futuros incluem considerar mais casos de teste, o desenvolvimento de novos procedimentos de busca local, paralelismo e experimentos computacionais adicionais.

[1] http://challenge.roadef.org/2005/en/revival2005 Acessado em 20/07/2016