Heurística híbrida para o problema de sequenciamento de carros

Antonio Augusto Chaves
Universidade Federal de São Paulo - UNIFESP
São José dos Campos, SP, Brasil
antonio.chaves@unifesp.br

Edson Luiz França Senne
Universidade Estadual Paulista - UNESP
Guaratinguetá, SP, Brasil
elfsenne@unesp.br

RESUMO

O problema de sequenciamento de carros é um importante problema de programação industrial. O problema consiste em determinar a ordem em que um conjunto de veículos deve passar por vários estágios de uma linha de montagem, a fim de tornar o processo de fabricação o mais econômico possível. É um problema de otimização combinatória NP-difícil e neste artigo considera-se apenas a fase final da linha de montagem, em que os veículos se diferem pelos diferentes dispositivos opcionais que devem ser instalados. Apresenta-se um algoritmo heurístico baseado na metaheurística *Clustering Search* para resolver o problema. A eficiência deste algoritmo é testada para instâncias de referência obtidas do repositório CSPLib

PALAVRAS CHAVE. Clustering Search, Sequenciamento, Heurísticas.

Área Principal: Metaheurísticas

ABSTRACT

The car sequencing problem is an important industrial scheduling problem. The problem consists in to determine the order in which a set of vehicles must go through various stages of an assembly line in order to make the manufacturing process as economical as possible. It is an NP-hard combinatorial optimization problem and in this paper we consider only the final stage of the assembly line, in which vehicles are distinguished by different optional devices that have to be installed on them. A heuristic algorithm for solving the problem which is based on Clustering Search metaheuristic is presented. The efficiency of this heuristic algorithm is tested on benchmark instances drawn from CSPLib repository.

KEYWORDS. Clustering Search. Sequencing. Heuristics.

Main Area: Metaheuristics

1. Introdução

Um importante problema de programação industrial é conhecido como problema de sequenciamento de carros (do inglês, CSP - *Car Sequencing Problem*). A moderna indústria automobilística precisa oferecer uma elevada variedade de veículos, permitindo aos consumidores escolher entre uma ampla gama de produtos. Ao mesmo tempo, a indústria necessita minimizar os custos de produção. O desafio de fornecer uma ampla gama de produtos a baixo custo requer um processo de planejamento e otimização. Um passo importante neste processo é planejar a sequência diária em que os veículos devem ser dispostos na linha de montagem.

A produção moderna de veículos compreende basicamente três fases: a montagem do corpo do veículo (*body shop*), a pintura do veículo (*paint shop*) e a instalação de dispositivos opcionais no veículo (*assembly shop*). O CSP consiste em determinar a ordem em que um conjunto de veículos deve passar por essas etapas, a fim de tornar todo o processo de fabricação o mais econômico possível. Cada uma destas fases de fabricação é submetida a um conjunto específico de restrições, que tendem a ser conflitantes. Para o processo de pintura, por exemplo, é interessante que os carros similares estejam agrupados, uma vez que o custo desta fase está relacionada principalmente com a mudança de cor. No entanto, este agrupamento de veículos pode não ser conveniente para a instalação de dispositivos opcionais, pois existirão custos adicionais se muitos veículos do mesmo tipo tiverem de ser processados consecutivamente devido às capacidades limitadas das unidades de instalação de dispositivos.

O problema é bem descrito por [Knausz, 2008]. Uma competição internacional de algoritmos para resolver este problema foi proposta em 2005 pela fabricante de automóveis Renault e organizada pela ROADEF, a Sociedade Francesa de Pesquisa Operacional e Análise de Decisão (ver *http://challenge.roadef.org/2005/en/*). Para esta competição foram consideradas as restrições de pintura e instalação de opcionais, mas o objetivo de minimizar o número de violações de restrições de instalação de opcionais (divididas em restrições de alto nível de prioridade e restrições de baixo nível de prioridade) tinha um peso maior do que a minimização do número de alterações de cores da pintura. O objetivo era construir uma sequência de veículos otimizando os requisitos de pintura e de instalação de opcionais, o que levava à minimização de três funções-objetivo possivelmente contraditórias: o número de violações de restrições de alta prioridade de instalação de opcionais (com peso de 10^6), o número violações de restrições de baixa prioridade de instalação de opcionais (com peso de 10^3), e o número de alterações de cor (com peso 1). Todos os algoritmos submetidos para esta competição deveriam retornar a melhor solução encontrada após 600 segundos de tempo de computação.

Neste artigo, considera-se apenas a fase final da linha de montagem (*assembly shop*). Assim, o problema é determinar em que sequência um conjunto de diferentes tipos de carros devem ser dispostos ao longo de uma linha de montagem, a fim de instalar equipamentos opcionais (como ar-condicionado, teto solar, sistema de som, GPS, etc). Cada item é instalado por uma estação de trabalho específica, que tem uma capacidade limitada de atendimento de automóveis por unidade de tempo. Por conseguinte, os veículos que requerem um determinado item devem ser espaçadas de modo a não exceder a capacidade de atendimento da estação de trabalho correspondente. Esta exigência é modelada por uma restrição racional do tipo N_j/P_j, o que significa que em cada sequência de P_j carros consecutivos, no máximo N_j carros devem exigir o item j. Considera-se que o plano de produção especifica quantos carros de cada tipo deve ser montado. O objetivo é encontrar uma sequência, que não viola qualquer dessas restrições racionais ou, se tal sequência não existir, minimizar o número de violações dessas restrições.

Uma abordagem alternativa para modelar este requisito é considerado por [Golle et al., 2010]. Esta abordagem avalia uma sequência considerando explicitamente os tempos de operação, os movimentos dos trabalhadores, as fronteiras da estação de trabalho e outras características operacionais da linha de montagem. Desta forma, sobrecargas de trabalho podem ser quantificadas exatamente e, por conseguinte, serem minimizadas. Uma sequência é considerada viável se

não houver sobrecarga de trabalho durante a execução da sequência; caso contrário, a sequência é inviável.

Neste artigo serão consideradas as restrições racionais. O problema a ser abordado neste trabalho pode ser formalmente apresentado considerando as seguintes definições:

- $C = \{1, ..., n\}$ é o conjunto de tipos de carros a serem produzidos;
- $q_i (i = 1, ..., n)$ é a quantidade de carros do tipo i no plano de produção;
- $E = \{1, ..., m\}$ é o conjunto de dispositivos a serem instalados;
- Cada dispositivo $j \in E$ está sujeito a uma restrição N_j/P_j e, se esta restrição não for respeitada, existirá um custo adicional a_j para a instalação do dispositivo j;
- $Q : \Pi \times C \rightarrow N$, onde Π é o conjunto de sequências de elementos de C e N é o conjunto de números naturais, tal que $Q(s, i) = n$ se existem n ocorrências de $i \in C$ na sequência $s \in \Pi$;
- $sub(s)$ é o conjunto de subsequências da sequência $s \in \Pi$, onde π é uma subsequência de s (denotado como $\pi \subseteq s$) se existem duas (possivelmente vazias) sequências π_1 e $\pi_2 \in \Pi$ tal que $s = \pi_1 \pi \pi_2$;
- $viola : sub(s) \times E \rightarrow \{0, 1\}$, tal que para cada subsequência π de s e dispositivo $j \in E$, $viola(\pi, j) = 0$ se $Q(\pi, j) \leq N_j$, e $viola(\pi, j) = 1$, caso contrário;
- $custo : \Pi \rightarrow R$, onde R é o conjunto de números reais, tal que $custo(s)$ para qualquer sequência s é definido por:

$$custo(s) = \sum_{j \in E} \sum_{\pi \subseteq s}^{|\pi| = P_j} viola(\pi, j) \times a_j$$

Dadas estas definições, o problema considerado neste trabalho pode ser definido formalmente como: determinar uma sequência s de menor custo tal que, para cada tipo de carro $i \in C$, $Q(s, i) = q_i$. Em outras palavras, o problema é buscar uma sequência de custo mínimo composta pelos tipos de veículos que devem ser produzidos.

Neste trabalho propõe-se um algoritmo heurístico híbrido baseado no método *Clustering Search* (CS) para resolver eficientemente o problema. Este algoritmo utiliza a metaheurística *Iterated Local Search* (ILS) como método de geração de soluções para o CS. Testes com instâncias disponíveis na literatura são realizados para analisar a eficiência do CS.

O restante deste trabalho está organizado da forma descrita a seguir. A Seção 2 apresenta uma breve revisão da literatura sobre o CSP, com os principais trabalhos e as técnicas usadas para resolver o problema. A Seção 3 discute a heurística proposta, apresentando em detalhes as metaheurísticas CS e ILS. A Seção 4 apresenta os resultados obtidos pela heurística proposta, considerando exemplares do problema disponíveis na biblioteca CSPLib. A Seção 5 apresenta as conclusões do trabalho e aponta direções para trabalhos futuros.

2. Revisão da literatura

O CSP pertence à classe de problemas *NP-hard* [Kis, 2004], mesmo quando apenas a fase de instalação de dispositivos (*assembly shop*) é considerada. Existem várias propostas para resolver o CSP. Poucas destas propostas usam métodos exatos, por exemplo, [Prandtstetter e Raidl, 2008], baseada em Programação Linear Inteira (PLI), e [Fliedner e Boysen, 2008], baseada em um algoritmo *branch & bound* especializado que explora a estrutura do CSP a fim de reduzir a complexidade combinatória. No entanto, devido à complexidade do problema, a aplicação destas propostas está limitada a casos relativamente pequenos.

Para exemplares maiores do problema, métodos baseados em busca local têm sido particularmente bem sucedidos. [Gottlieb et al., 2003] apresentam algoritmos gulosos, algoritmos de busca local e algoritmos de otimização por colônia de formigas (ACO), e compara estes algoritmos para exemplares disponíveis na biblioteca de referência CSPLib. Os autores concluem que os algoritmos gulosos são bons para instâncias relativamente fáceis e que algoritmos de busca local executam muito bem, mas são ligeiramente inferiores à abordagem ACO. [Jaszkiewicz et al., 2004] usam um algoritmo genético de busca local (GLS), que inclui uma heurística eficiente para geração de soluções iniciais e um rápido procedimento de busca local. Os autores concluem que o algoritmo GLS gera melhores soluções do que um algoritmo de busca local *multiple-start*, dado o mesmo tempo de computação.

[Solnon et al., 2008] apresentam o contexto industrial e as especificidades do CSP, descrevem o processo do desafio ROADEF 2005, apresentam os métodos propostos pelas equipes concorrentes e analisam os resultados desses métodos nos casos de sequenciamento de carros fornecidos pela Renault. Esta competição foi vencida por [Estellon et al., 2008], que utilizou uma abordagem denominada busca local muito rápida (VFLS) que se baseia em explorações muito rápidas de pequena vizinhança. Os autores apontam, após um extenso estudo computacional, que metaheurísticas sofisticadas são inúteis para resolver o CSP e que os aspectos algorítmicos, por vezes negligenciados, continuam a ser os principais ingredientes para a concepção e engenharia de heurísticas de busca local de alto desempenho.

Apesar da opinião de [Estellon et al., 2008], metaheurísticas têm sido utilizadas com sucesso para resolver o CSP. [Ribeiro et al., 2008a,b] apresentam e descrevem em detalhes uma abordagem heurística para o CSP baseada nos paradigmas das metaheurísticas busca local iterativa (ILS) e busca em vizinhança variável (VNS). A heurística baseada em ILS é usada para otimizar o objetivo de montagem de alto nível de prioridade e a heurística baseada em VNS é usada para otimizar os objetivos de montagem de baixo nível de prioridade e de mudança de cor de pintura. Os experimentos computacionais mostraram que a heurística proposta obtém resultados muito bons, merecendo o segundo prêmio no desafio ROADEF 2005.

As metaheurísticas ILS e VNS também são usadas em alguns outros trabalhos. [Prandtstetter e Raidl, 2008] propõem uma abordagem híbrida que usa técnicas de PLI dentro de um esquema VNS para examinar grandes vizinhanças. Resultados computacionais demonstraram que grandes vizinhanças examinadas utilizando técnicas de PLI melhoram o desempenho global significativamente, de modo que a abordagem híbrida supera claramente variantes que consideram apenas vizinhanças definidas de forma tradicional.

Outras abordagens metaheurísticas que têm sido utilizados para resolver CSP são: busca tabu [Zufferey et al., 2006], recozimento simulado [Briant et al., 2008], busca tabu iterativa [Cordeau et al., 2008], algoritmo evolutivo [Joly e Frein, 2008], GRASP [Rizzo e Urrutia, 2011] e *beam search* [Golle et al., 2015].

3. *Clustering Search*

Neste trabalho propõe-se um novo método híbrido para o CSP, baseado no método *Clustering Search* (CS) [Oliveira et al., 2013]. O CS é um método híbrido que busca combinar metaheurísticas e heurísticas de busca local, em que a busca é intensificada somente em regiões do espaço de busca que merecem atenção especial (regiões promissoras). O objetivo é introduzir inteligência e prioridade para a escolha de soluções onde aplicar a busca local, ao invés de escolher aleatoriamente ou aplicar busca local em todas as soluções.

O CS procura localizar áreas de busca promissora construindo-as em *clusters*. Um *cluster* é definido por um centro, c, que é, geralmente, inicializado aleatoriamente e, posteriormente, tende progressivamente a pontos promissores no espaço de busca. O número de *clusters*, NC, deve ser fixado a priori.

O CS pode ser explicitado em quatro partes conceitualmente independentes: a metaheurística (SM), o componente de agrupamento (IC), um módulo de análise (AM) e a busca local (LS).

A Figura 1 apresenta o fluxograma do CS.

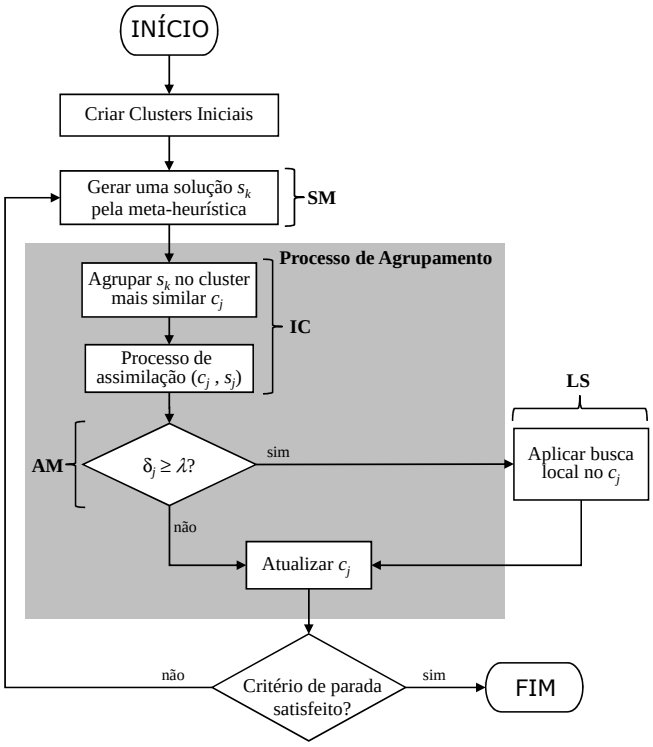

Figura 1: Fluxograma do CS

O componente SM pode ser implementado por qualquer algoritmo de otimização que gera soluções diversificadas do espaço de busca. Ele trabalha como um gerador de soluções, explorando o espaço de busca através da manipulação de um conjunto de soluções, de acordo com sua estratégia de busca específica.

O componente IC procura reunir soluções similares em *clusters*, mantendo, para cada *cluster*, uma solução representativa deste grupo, denominada centro do *cluster*. Uma métrica de distância, Δ, é definida, a priori, permitindo uma medida de similaridade para o processo de agrupamento. Por exemplo, em um problema de otimização combinatória, a similaridade pode ser definida como o número de movimentos necessários para alterar uma solução qualquer até tornar-se igual ao centro de um *cluster* [Oliveira e Lorena, 2007].

O processo de assimilação é aplicado a cada nova solução gerada s_k, considerando o centro c_j do *cluster* mais próximo de s_k. A idéia do processo de assimilação é mesclar as características das duas soluções s_k e c_j de modo a produzir uma nova solução representativa que substituirá o centro deste *cluster*. O método *Path-relinking* [Glover et al., 2000] pode ser usado para gerar uma série de soluções intermediárias entre s_k e c_j de modo que o novo centro do *cluster* seja a melhor solução obtida neste processo.

O componente AM examina cada *cluster*, em intervalos regulares, indicando um provável *cluster* promissor. A densidade de um *cluster* j (δ_j) é uma medida que indica o nível de atividade dentro deste *cluster*. Por exemplo, δ_j pode ser definida como o número de soluções geradas pelo componente SM e agrupadas no *cluster* j. Quando δ_j atinge um limitante λ, indicando que um certo padrão de soluções está sendo gerado por SM, a região deste *cluster* é melhor investigada para acelerar o processo de convergência.

Por último, o componente LS explora uma região supostamente promissora, representada por um *cluster*, e intensifica a busca nessa região, aplicando heurísticas no centro deste *cluster*.

3.1. CS aplicado ao CSP

Uma solução do CSP pode ser representada por um vetor de valores inteiros formado pela sequência de carros a serem produzidos. O valor armazenado em cada posição i deste vetor representa o tipo de carro a ser produzido na sequência. A Figura 2 ilustra uma solução com 7 carros de 3 tipos (produção de 2 carros do tipo 1, 3 carros do tipo 2 e 2 carros do tipo 3). A solução representada neste exemplo corresponde à sequência: {1, 1, 2, 2, 3, 2, 3}.

	1	2	3	4	5	6	7
Tipo de carro	1	1	2	2	3	2	3

Figura 2: Representação de solução do CSP

Para calcular a função objetivo do CSP utiliza-se o seguinte algoritmo:

Algoritmo 1: FUNÇÃOOBJETIVO

Entrada: s
Saída: número de violações das restrições racionais N_j/P_j

```
1  início
2      Seja m o número de dispositivos a serem instalados
3      custo = 0
4      j = 0
5      enquanto j < m faça
6          Sejam N e D, respectivamente, o numerador e o denominador da restrição de
             capacidade referente ao item j
7          L ← lista de sequências de tamanho D em s
8          para i = 1 até |L| faça
9              soma ← número de vezes que o item j é usado nesta i-ésima sequência de
                 carros
10             se soma > N então
11                 custo = custo + (soma − N) ∗ a_j;
12             fim
13         fim
14         j = j + 1;
15     fim
16 fim
17 retorna custo
```

O método CS precisa de uma métrica de distância para calcular a similaridade entre soluções. Neste trabalho define-se uma medida de distância entre duas soluções do CSP como sendo o número de tipos de carros atribuídos em posições diferentes nestas soluções.

O CS utiliza a metaheurística *Iterated Local Search* (ILS) como componente SM para gerar soluções para o processo de agrupamento. O ILS [Lourenco et al., 2003] baseia-se na ideia de que um algoritmo de busca local pode obter melhores soluções gerando-se novas soluções de partida, as quais são obtidas por meio de perturbações na solução ótima local.

O algoritmo ILS compreende, basicamente, três componentes: (a) um procedimento que gera uma solução inicial s_0 para o problema; (b) um procedimento de busca local; e (c) um procedimento de perturbação, que modifica a solução corrente s, gerando uma nova solução s'. O desempenho do algoritmo ILS depende do mecanismo de perturbação, que deve ser forte o suficiente para permitir a diversificação (e assim escapar de um ótimo local) e fraca o suficiente para permitir a intensificação nas vizinhanças da melhor solução encontrada ou do ótimo local corrente.

Neste trabalho, o ILS parte de uma solução viável aleatória. Para tal, inicializa-se o vetor que representa a solução com a quantidade de carros a serem produzidos de cada tipo em posições escolhidas aleatoriamente. O procedimento de perturbação da heurística ILS consiste em

uma simples troca de posição entre dois tipos de carro da sequência. Essa troca é executada mesmo que a solução piore (ou seja, mesmo que o custo da nova solução seja maior do que o custo da solução existente antes da troca). Os tipos de carro da sequência a serem trocados de posição são escolhidos aleatoriamente. O procedimento de perturbação utiliza o parâmetro β, que indica quantas vezes o movimento de perturbação será aplicado.

A busca local do ILS considera todas as possíveis trocas entre dois tipos de carro da sequência, mas a troca somente é efetivada se o custo da sequência é minimizado, ou seja, há uma melhora na solução.

O pseudocódigo do ILS é apresentado a seguir:

Algoritmo 2: ILS

```
Entrada: β
Saída: melhor solução encontrada
1  início
2      s_0 = GeraSolucaoInicial()
3      s = BuscaLocal(s_0)
4      enquanto critérios de parada não estão satisfeitos faça
5          s' = Perturbacao(s, β)
6          s'' = BuscaLocal(s')
7          se s''.fo < s.fo então
8              s = s'';
9          fim
10     fim
11 fim
12 retorna s
```

Os *clusters* iniciais do CS são gerados da mesma maneira que a solução inicial do ILS. Desta forma, gera-se NC centros de *clusters* com soluções aleatórias viáveis.

A cada iteração do CS, uma solução s_k é agrupada ao *cluster* j mais próximo (*cluster* com a menor distância entre seu centro c_j e a solução s_k). O volume deste *cluster* é incrementado em uma unidade e o centro do *cluster* precisa ser atualizado com os novos atributos de s_k. Neste processo de assimilação é utilizado o método *Path-Relinking*: o novo centro será a melhor solução encontrada no caminho entre o centro c_j e a solução s_k. Para o CSP, um movimento no *Path-Relinking* consiste em trocar um tipo de carro de uma posição para outra no vetor. A Figura 3 apresenta um exemplo do *Path-Relinking* aplicado ao CSP.

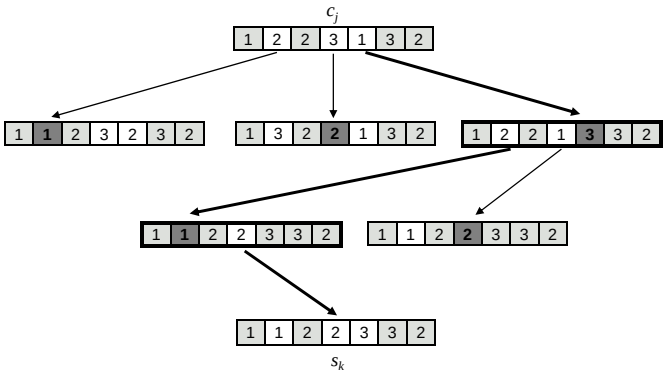

Figura 3: Exemplo do PR aplicado ao CSP

A busca local é acionada sempre que um *cluster* j for considerado promissor, intensificando a busca no centro c_j. [Chaves e Lorena, 2010] propõem a utilização do método Descida em Vizinhança Variável (VND, do inglês *Variable Neighborhood Descent*) [Mladenović e Hansen,

1997] como método de busca local. Nesse trabalho são utilizadas quatro heurísticas de descida:

1. Troca Simples: considera todas as possíveis trocas de posição entre dois tipos de carro da sequencia;
2. Transferência: considera a remoção de cada tipo de carro da sequência e todas as possíveis posições para inserção do mesmo;
3. Lin2Opt: considera a inversão da ordem dos tipos de carro em subsequências escolhidas aleatoriamente;
4. Troca Dupla: considera todas as possíveis trocas de dois tipos de carro consecutivos por outros dois tipos de carro consecutivos;

Se uma solução melhor for encontrada retorna-se para a primeira heurística e continua-se a busca a partir da nova solução. O VND se encerra quando nenhuma melhora na solução corrente puder ser obtida por meio destas heurísticas. O centro do *cluster* é atualizado com a melhor solução obtida pelo VND.

O pseudocódigo do CS é apresentado a seguir:

Algoritmo 3: CS

Entrada: NC, λ
Saída: melhor solução encontrada
início
  crie NC clusters iniciais
  {geração de soluções - SM}
  enquanto *critérios de parada não estão satisfeitos* **faça**
    gere uma solução (s_k) pelo ILS
    { processo de agrupamento - IC }
    encontre o *cluster* j mais similar a s_k
    agrupe s_k no *cluster* j ($\delta_j \leftarrow \delta_j + 1$)
    atualize o centro do *cluster* j ($c_j \leftarrow PR(c_j, s_k)$)
    { modulo de análise - AM }
    se $\delta_j > \lambda$ **então**
      reduza o volume $\delta_j \leftarrow 0$
      { heurística de busca local - LS }
      encontre o melhor vizinho ($\hat{c}_j$) de c_j
      se $\hat{c}_j.fo < c_j.fo$ **então**
        atualize o centro $c_j \leftarrow \hat{c}_j$
        se $c_j.fo < c^*.fo$ **então**
          $c^* \leftarrow c_j$
        fim
      fim
    fim
  fim
fim
retorna c^*

4. Resultados Computacionais

O método heurístico foi codificado na linguagem C e os testes computacionais foram executados em um PC com processador Intel core i5, 2.5 GHz e memória de 6 GB. Os experimentos foram empreendidos com o objetivo de evidenciar a qualidade dos resultados do CS, mostrando que este método pode ser competitivo para resolver o CSP.

Os experimentos computacionais foram conduzidos com exemplares disponíveis na CSPLib [Smith, 2004], disponível em *http://www.csplib.org/Problems/prob001/*. Estes problemas estão agrupados em três conjuntos e caracterizam-se por possuírem resultados conhecidos, alguns

com soluções factíveis e outros com solução factível ainda não encontrada, que requerem soluções com 100, 200, 300 e 400 carros, todas com as mesmas restrições de capacidade de 5 opcionais, que diferenciam-se pelo número de classes e a relação de pertinência dos opcionais nestas classes.

Os parâmetros do CS foram calibrados através de testes empíricos realizados sobre um subconjunto de instâncias. Os parâmetros adotados neste trabalho são:

- Número de *clusters* do CS (NC) = 15;
- Limitante de *cluster* promissor (λ) = 10;
- Número máximo de iterações do ILS = 1000;
- Intensidade da perturbação do ILS ($0,2 \leq \beta \leq 0,5$).

Os resultados computacionais obtidos encontram-se nas Tabelas 1, 2 e 3. A primeira coluna destas tabelas contêm nome da instância (Instância), a segunda coluna apresenta a melhor solução conhecida na literatura (Melhor), na terceira e quarta coluna têm-se a melhor solução encontrada pelo CS (Sol*) e a solução média em 20 execuções do CS (Sol), respectivamente. Por fim, na quinta coluna tem-se tempo médio de execução do CS (Tempo, em segundos).

Na Tabela 1 cada linha apresenta o resultado da resolução de 10 exemplares. Para cada exemplar o método CS foi executado 20 vezes.

Tabela 1: Resultados Computacionais para as instâncias [Lee et al., 1998]

		CS		
Instância	Melhor	Sol*	Sol	Tempo
problem_60	0,00	0,00	0,00	0,88
problem_65	0,00	0,00	0,00	1,10
problem_70	0,00	0,00	0,00	1,38
problem_75	0,00	0,00	0,00	2,09
problem_80	0,00	0,00	0,00	3,31
problem_85	0,00	0,00	0,00	4,02
problem_90	0,00	0,00	0,03	14,32
média	0,00	0,00	0,00	3,87

Tabela 2: Resultados Computacionais para as instâncias [Gent e Walsh, 1999]

		CS		
Instância	Melhor	Sol*	Sol	Tempo
problem_04_72	0	0	2,33	196,65
problem_06_76	6	6	6,00	190,23
problem_10_93	3	3	5,87	394,59
problem_16_81	0	0	2,23	353,66
problem_19_71	2	2	3,97	317,00
problem_21_90	2	2	3,60	306,41
problem_36_92	2	2	3,97	297,78
problem_41_66	0	0	0,57	159,07
problem_26_82	0	0	2,57	273,49
média	1,67	1,67	3,46	276,54

Tabela 3: Resultados Computacionais para instâncias [Gagné et al., 2006]

		CS		
Instância	Melhor	Sol*	Sol	Tempo
pb_200_01	0	2	5,40	1001,53
pb_200_02	2	4	5,97	1001,33
pb_200_03	4	9	11,00	1001,26
pb_200_04	7	9	10,93	1001,46
pb_200_05	6	6	7,83	1001,27
pb_200_06	6	6	7,07	1001,37
pb_200_07	0	0	1,30	928,03
pb_200_08	8	8	9,03	1001,08
pb_200_09	10	10	11,57	1001,21
pb_200_10	19	19	20,30	1001,35
pb_300_01	0	6	9,93	1005,03
pb_300_02	12	15	17,37	1004,00
pb_300_03	13	15	17,93	1004,27
pb_300_04	7	10	13,23	1005,93
pb_300_05	29	38	41,93	1004,99
pb_300_06	2	9	12,93	1004,27
pb_300_07	0	7	10,00	1004,83
pb_300_08	8	8	11,27	1005,72
pb_300_09	7	11	14,07	1004,96
pb_300_10	21	25	28,10	1003,23
pb_400_01	1	11	14,97	1013,11
pb_400_02	16	27	31,17	1011,67
pb_400_03	9	15	17,90	1013,52
pb_400_04	19	21	24,27	1011,13
pb_400_05	0	5	10,90	1012,02
pb_400_06	0	6	10,67	1009,84
pb_400_07	4	12	16,43	1012,11
pb_400_08	4	12	16,97	1010,71
pb_400_09	5	15	18,90	1011,20
pb_400_10	0	8	11,60	1011,36
média	4,78	6,00	7,79	993,17

Pelos resultados mostrados nas Tabelas 1 a 3 pode-se observar que o método CS proposto obtêm boas soluções para o CSP em um tempo computacional baixo. O CS obtêm a solução ótima para todos os exemplares das instâncias de [Lee et al., 1998]. Para o segundo conjunto de instâncias [Gent e Walsh, 1999], consideradas mais difíceis, o CS encontra todas a melhores soluções conhecidas na literatura. No conjunto de instâncias com maior número de carros [Gagné et al., 2006], o CS encontra a melhor solução conhecida na literatura em 7 das 30 instâncias testadas e soluções próximas das melhores conhecidas nas demais instâncias.

O método CS encontra boas soluções para todos os exemplares testados. Além disso, o método mostrou ser robusto, uma vez que os valores das soluções médias estão próximos das melhores soluções encontradas pelo CS.

Para as instâncias maiores, o tempo computacional do CS foi limitado em 1000 segundos. Apesar disso, o CS possui uma boa convergência, encontrando boas soluções em poucos segundos de execução.

5. Conclusão

Este trabalho propõe um novo método híbrido para gerar boas soluções para o problema de sequenciamento de carros (CSP) em uma linha de produção de veículos considerando a instalação de dispositivos opcionais. O método híbrido *Clustering Search* (CS) proposto faz uso da metaheurística *Iterative Local Search* (ILS) para explorar o espaço de soluções problema.

A ideia do CS é evitar a aplicação de heurísticas de busca local em todas as soluções geradas por uma metaheurística. O CS detecta regiões promissoras no espaço de soluções e aplica busca local somente nessas regiões. Detectar regiões promissoras é uma alternativa interessante, prevenindo a aplicação indiscriminada de heurísticas.

Os experimentos computacionais mostram que o CS é competitivo para resolver esse problema em um tempo computacional razoável. O CS encontrou boas soluções para os exemplares testados em pouco tempo de execução e mostrou ser um método robusto. Portanto, estes resultados validam a aplicação do CS ao CSP.

Um possível alvo de estudos futuros é a utilização de outras heurísticas de busca local. Pretende-se também utilizar outras metaheurísticas para gerar soluções para o processo de agrupamento do CS. Além disso, a atualização do valor da função objetivo do CSP sem a necessidade de recalcular toda a função será objeto de investigação futura.

Agradecimentos

Este trabalho é financiado pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq (Processo nº 482170/2013-1, nº 307330/2015-0, nº 303339/2013-6).

Referências

Briant, O., Naddef, D., e Mounié, G. (2008). Greedy approach and multi-criteria simulated annealing for the car sequencing problem. *European Journal of Operat. Research*, 191(3):993–1003.

Chaves, A. A. e Lorena, L. A. N. (2010). Clustering search algorithm for the capacitated centered clustering problem. *Computers & Operations Research*, 37(3):552–558.

Cordeau, J.-F., Laporte, G., e Pasin, F. (2008). Iterated tabu search for the car sequencing problem. *European Journal of Operational Research*, 191(3):945–956.

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*, 191(3):1023–1042.

Gagné, C., Gravel, M., e Price, W. L. (2006). Solving real car sequencing problems with ant colony optimization. *European Journal of Operational Research*, 174(3):1427–1448.

Gent, I. P. e Walsh, T. (1999). Csplib: a benchmark library for constraints. In *Principles and Practice of Constraint Programming–CP'99*, p. 480–481. Springer.

Glover, F., Laguna, M., e Martí, R. (2000). Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 29(3):653–684.

Golle, U., Boysen, N., e Rothlauf, F. (2010). Analysis and design of sequencing rules for car sequencing. *European Journal of Operational Research*, 206(3):579–585.

Golle, U., Rothlauf, F., e Boysen, N. (2015). Iterative beam search for car sequencing. *Annals of Operations Research*, 226(1):239–254.

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

Jaszkiewicz, A., Kominek, P., e Kubiak, M. (2004). Adaptation of the genetic local search algorithm to a car sequencing problem. In *7th National Conference on Evolutionary Algorithms and Global Optimization, Kazimierz Dolny, Poland*, p. 67–74.

Joly, A. e Frein, Y. (2008). Heuristics for an industrial car sequencing problem considering paint and assembly shop objectives. *Computers & Industrial Engineering*, 55(2):295–310.

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

Knausz, M. (2008). *Parallel variable neighboorhood search for the car sequencing problem*. PhD Thesis, Fakultät fur Informatik der Technischen Universität Wien.

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 *Advanced Topics in A.I.*, p. 191–202. Springer.

Lourenco, H., Martin, O., e Stützle, T. (2003). Iterated local search. In Glover, F. e Kochenberger, G. A., editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, p. 320–353. Springer, New York.

Mladenović, N. e Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100.

Oliveira, A. C. M., Chaves, A. A., e Lorena, L. A. N. (2013). Clustering search. *Pesquisa Operacional*, 33(1):105–121.

Oliveira, A. C. e Lorena, L. A. (2007). Hybrid evolutionary algorithms and clustering search. In *Hybrid Evolutionary Algorithms*, p. 77–99. Springer.

Prandtstetter, M. e Raidl, G. R. (2008). An integer linear programming approach and a hybrid variable neighborhood search for the car sequencing problem. *European Journal of Operational Research*, 191(3):1004–1022.

Ribeiro, C. C., Aloise, D., Noronha, T. F., Rocha, C., e Urrutia, S. (2008a). An efficient implementation of a vns/ils heuristic for a real-life car sequencing problem. *European Journal of Operational Research*, 191(3):596–611.

Ribeiro, C. C., Aloise, D., Noronha, T. F., Rocha, C., e Urrutia, S. (2008b). A hybrid heuristic for a multi-objective real-life car sequencing problem with painting and assembly line constraints. *European Journal of Operational Research*, 191(3):981–992.

Rizzo, L. e Urrutia, S. (2011). Uma heuristica grasp para o problema estendido de sequenciamento de carros. In *XLIII Simpósio Brasileiro de Pesquisa Operacional*, p. 1745–1752.

Smith, B. (2004). Csplib problem 001: Car sequencing. In *CSPLib: A problem library for constraints*. URL `http://www.csplib.org/Problems/prob001`.

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.

Zufferey, N., Studer, M., Silver, E. A., et al. (2006). Tabu search for a car sequencing problem. In *FLAIRS Conference*, p. 457–462.