INF05010 Otimização Combinatória

Trabalho Final:

Variable Neighborhood Search para o problema do trabalho balanceado

Giovanni Milanez e Gleydson Campos

21 de junho de 2025

1 Introdução

O objetivo deste trabalho foi implementar uma meta-heurística para resolver o problema do *trabalho balanceado*. A meta-heurística escolhida foi o *Variable Neighborhood Search* (VNS), cujos detalhes serão apresentados na Seção 3.

O problema do trabalho balanceado é definido da seguinte forma: dado um conjunto de n tarefas que devem ser executadas em ordem e m operadores, cada operador $j \in [m]$ leva um tempo p_{ij} para executar a tarefa $i \in [n]$. Deseja-se particionar essas n tarefas em m Intervalos contíguos $[b_k, e_k]$, $k \in [m]$, e determinar uma permutação dos operadores π, de forma que o tempo total necessário para cada operador seja o menor possível.

Mais formalmente, o tempo total de execução do j-ésimo operador é dado por:

$$T_j = \sum_{t \in [b_j, e_j]} p_{t,\pi_j}$$

e o objetivo é minimizar o tempo máximo de execução entre os operadores, ou seja:

$$T = \max_{j \in [m]} T_j$$

2 Formulação matemática

Variáveis:

$$y_{ik} \in \{0, 1\} \quad \text{para } i = 1, \ldots, n;\ k = 1, \ldots, m$$

$$T \geq 0$$

Parâmetros:

- p_{ik}: tempo para o operador k executar a tarefa i
- n: número total de tarefas
- m: número de operadores

Função Objetivo:

$$\min T$$

Restrições:

$$\sum_{k=1}^{m} y_{ik} = 1 \qquad \forall i = 1, \ldots, n \tag{1}$$

$$\sum_{i=1}^{n} p_{ik} y_{ik} \leq T \qquad \forall k = 1, \ldots, m \tag{2}$$

$$y_{ik} - y_{(i+1)k} + y_{jk} \leq 1 \qquad \forall k = 1, \ldots, m;\ \forall i = 1, \ldots, n-1;\ \forall j = i+2, \ldots, n \tag{3}$$

A restrição (1) garante que cada tarefa seja atribuída a exatamente um operador. A restrição (2) assegura que o tempo total de cada operador não exceda o makespan T. A restrição (3) garante que as tarefas atribuídas a cada operador formem segmentos contíguos.

3 Variable Neighborhood Search

Algorithm 1 Variable Neighborhood Search (VNS)

```
1:  function VNS(p::Matrix{Float64}; iter_max=5000000)
2:      n, m ← size(p)
3:      (borders, π) ← solucao_inicial(p, n, m)
4:      f_best ← avalia(p, borders, π)
5:      k_max ← length(NEIGHS)
6:      for it ← 1 to iter_max do
7:          k ← 1
8:          while k ≤ k_max do
9:              candidate ← NEIGHS[k](p, borders, π, n)
10:             (new_b, new_π) ← busca_local(p, candidate, n)
11:             f_new ← avalia(p, new_b, new_π)
12:             if f_new < f_best then
13:                 (f_best, borders, π) ← (f_new, new_b, new_π)
14:                 k ← 1
15:             else
16:                 k ← k + 1
17:             end if
18:         end while
19:     end for
20:     return (f_best, borders, π)
21: end function
```

3.1 Componentes do Algoritmo

3.1.1 Geração da solução inicial

A solução inicial é gerada através de uma heurística gulosa:

1. Geram-se $m - 1$ pontos de corte aleatórios distintos entre as posições 2 e n
2. Ordena-se estes pontos para definir m segmentos contíguos de tarefas
3. Atribui-se uma permutação aleatória dos operadores aos segmentos

3.1.2 Função de avaliação

Para uma dada segmentação e atribuição de operadores, calcula-se o tempo total de cada operador j como a soma dos tempos das tarefas de seu segmento: $T_j = \sum_{i \in \text{segmento}_j} p_{i,\pi_j}$. O makespan é $\max_j T_j$.

3.1.3 Estruturas de vizinhança

O VNS utiliza 5 vizinhanças ordenadas por grau crescente de perturbação:

1. **Troca de operadores (swap):** Troca aleatoriamente dois operadores entre seus segmentos. Mantém a segmentação, alterando apenas a atribuição.
2. **Shift ± 1:** Move uma fronteira entre segmentos em uma posição (para frente ou para trás), respeitando os limites dos segmentos adjacentes.
3. **Shift $\pm\delta$:** Generalização do anterior, movendo uma fronteira em até $\delta = 4$ posições aleatoriamente.
4. **Expand-and-reassign ± 1:** Combina o shift ± 1 com reatribuição ótima: após mover a fronteira, encontra o melhor operador para o segmento modificado e o troca com o operador atual.
5. **Expand-and-reassign $\pm\delta$:** Versão mais agressiva que combina shift $\pm\delta$ com reatribuição ótima do operador.

Esta ordem permite ao VNS explorar primeiro modificações locais (vizinhanças 1-2), depois estruturais (3), e finalmente otimizações mais complexas (4-5).

3.1.4 Busca Local

Aplica-se busca local *first-improvement* percorrendo todas as 5 vizinhanças na ordem definida. Para cada vizinhança, avalia-se sistematicamente os vizinhos até encontrar uma melhoria, quando se move para a nova solução e reinicia o processo. A busca termina quando nenhuma vizinhança produz melhoria.

3.1.5 Critério de terminação

O algoritmo termina após 5 milhões de iterações, garantindo tempo computacional adequado para convergência.

3.1.6 Representação da solução

Uma solução é representada por:

- $borders$: vetor definindo o início de cada segmento, onde $borders[i]$ é a primeira tarefa do segmento i
- π: permutação dos operadores, onde $\pi[i]$ é o operador atribuído ao segmento i

Exemplo: Para $n = 8$ tarefas e $m = 3$ operadores:

$$borders = [1, 4, 7] \tag{1}$$

$$\pi = [2, 1, 3] \tag{2}$$

$$\text{Segmentos} = \{(1, 2, 3) \rightarrow \text{Op. } 2, (4, 5, 6) \rightarrow \text{Op. } 1, (7, 8) \rightarrow \text{Op. } 3\} \tag{3}$$

4 Configuração Experimental

Para avaliar o desempenho do algoritmo VNS proposto, foram realizados experimentos computacionais comparando-o com diferentes abordagens: heurísticas construtivas e formulações de programação linear inteira mista (MILP) resolvidas via solver exato. Esta seção detalha a configuração experimental utilizada.

4.1 Instâncias de Teste

Os experimentos foram conduzidos utilizando 10 instâncias do problema do trabalho balanceado, denominadas tba1 a tba10. A Tabela 1 apresenta as características de cada instância.

Instância	**Tarefas (n)**	**Operadores (m)**	**Razão (n/m)**
tba1	37	13	2.85
tba2	32	15	2.13
tba3	27	13	2.08
tba4	22	15	1.47
tba5	42	10	4.20
tba6	37	13	2.85
tba7	32	14	2.29
tba8	27	11	2.45
tba9	22	10	2.20
tba10	42	10	4.20

Tabela 1: Características das instâncias de teste

As instâncias apresentam variação tanto no número de tarefas (22 a 42) quanto no número de operadores (10 a 15), permitindo avaliar o comportamento dos algoritmos em diferentes configurações do problema. As instâncias apresentam tempos de processamento p_{ij} variados para cada par tarefa-operador, com valores mínimos padronizados em 0.1.

4.2 Ambiente Computacional

Todos os experimentos foram executados em ambiente computacional com as seguintes especificações:

- **Linguagem de programação:** Julia 1.11.5
- **Solver exato:** GLPK (GNU Linear Programming Kit)
- **Sistema operacional:** Linux (x86_64-linux-gnu) e Windows (x86_64-w64-mingw32)
- **Bibliotecas auxiliares:** JuMP.jl para modelagem matemática, Random.jl para geração de números aleatórios

O algoritmo VNS foi implementado integralmente em Julia, aproveitando sua eficiência computacional para operações numéricas intensivas. As formulações MILP foram modeladas utilizando a biblioteca JuMP.jl e resolvidas através do solver GLPK, escolhido por sua disponibilidade gratuita e capacidade de lidar com problemas de programação inteira mista de médio porte.

Por se tratar de um método estocástico, foram realizadas 30 execuções independentes por instância, inicialmente, e depois mais 10, ambas com sementes aleatórias distintas, permitindo análise estatística robusta dos resultados.

4.3 Algoritmos de Comparação

Para avaliar a eficácia do VNS proposto, foram implementados e comparados quatro algoritmos adicionais:

4.3.1 Heurísticas Construtivas

Heurística Gulosa: Algoritmo construtivo que atribui cada tarefa sequencialmente ao operador que resulta no menor makespan naquele momento. Para cada tarefa i, calcula-se o tempo total resultante $T_k + p_{ik}$ para cada operador k e escolhe-se o operador com menor valor.

Heurística com Contiguidade: Extensão da heurística gulosa que reorganiza a solução obtida para garantir a propriedade de contiguidade das tarefas. Após a atribuição inicial, as tarefas são reagrupadas em segmentos contíguos mantendo aproximadamente o mesmo número de tarefas por operador.

4.3.2 Formulações MILP

Solver Simples: Implementação da formulação MILP básica apresentada na Seção 2, sem restrições de contiguidade (3). Permite atribuições não-contíguas de tarefas aos operadores, executado com timeout de 20 minutos.

Solver Melhorado: Versão estendida que inclui restrições de contiguidade do tipo:

$$y_{ik} - y_{(i+1)k} + y_{jk} \leq 1 \quad \forall k, \forall i < j - 1$$

garantindo que tarefas atribuídas ao mesmo operador formem segmentos contíguos. Executado com timeout de 3 horas devido à maior complexidade computacional.

Ambas as formulações MILP foram resolvidas utilizando o solver GLPK com configurações padrão, buscando soluções ótimas dentro dos limites de tempo estabelecidos.

4.4 Parâmetros e Critérios

4.4.1 Parâmetros do VNS

O algoritmo VNS foi configurado com os seguintes parâmetros:

- **Critério de parada:** 5.000.000 de iterações, determinado através de testes que mostraram convergência adequada dentro deste limite
- **Número de vizinhanças:** 5 estruturas ordenadas por agressividade crescente, permitindo exploração sistemática do espaço de soluções
- **Parâmetro δ:** valor máximo de 4 para movimentações de fronteira, balanceando diversificação e intensificação na busca
- **Estratégia de busca local:** first-improvement para garantir eficiência computacional sem comprometer qualidade das soluções
- **Número de execuções:** 30 execuções independentes por instância com sementes aleatórias distintas, assegurando significância estatística dos resultados

4.4.2 Configurações dos Solvers

Os algoritmos exatos foram configurados com diferentes limites de tempo baseados na complexidade computacional de cada formulação:

- **Solver Simples:** timeout de 1.200 segundos (20 minutos), suficiente para a formulação relaxada sem restrições de contiguidade encontrar a solução ótima (limite inferior)
- **Solver Melhorado:** timeout de 10.800 segundos (3 horas), necessário devido à complexidade exponencial introduzida pelas restrições de contiguidade
- **Configurações GLPK:** parâmetros padrão, modo silencioso ativado para reduzir saída de log

4.4.3 Métricas de Avaliação

Para cada algoritmo, foram coletadas as seguintes métricas:

- **Qualidade da solução:** valor do makespan obtido
- **Gap relativo:** distância percentual em relação ao melhor método conhecido
- **Tempo computacional:** tempo de execução até convergência ou timeout
- **Robustez (VNS):** estatísticas descritivas das 30 execuções
- **Tempo de convergência (VNS):** tempo até encontrar a melhor solução
- **Contiguidade:** verificação da propriedade de contiguidade nas soluções

Os resultados são apresentados em termos de qualidade absoluta das soluções e comparações relativas entre os métodos.

5 Resultados Experimentais

Esta seção apresenta os resultados obtidos pelos algoritmos implementados nas 10 instâncias de teste. Os experimentos foram conduzidos seguindo a configuração descrita na Seção 4, com análises comparativas de desempenho, tempo computacional e qualidade das soluções.

5.1 Desempenho do VNS

O algoritmo VNS foi executado 30 vezes para cada instância, permitindo análise estatística da robustez do método. A Tabela 2 apresenta um resumo dos resultados obtidos.

Instância	n×m	Melhor	Pior	Média	Desvio	Tempo (s)
tba1	37×13	0.564	1.384	0.674	0.163	74.6
tba2	32×15	0.523	0.721	0.670	0.050	70.8
tba3	27×13	0.475	0.751	0.558	0.107	68.1
tba4	22×15	0.307	0.307	0.307	0.000	66.2
tba5	42×10	1.491	1.812	1.547	0.107	69.7
tba6	37×13	0.567	0.752	0.642	0.053	71.7
tba7	32×14	0.594	0.882	0.684	0.063	97.5
tba8	27×11	0.846	0.929	0.886	0.028	68.3
tba9	22×10	0.584	0.896	0.632	0.056	68.2
tba10	42×10	1.326	1.883	1.427	0.170	69.9

Tabela 2: Resultados do VNS (30 execuções por instância)

O VNS demonstrou boa consistência, com baixo desvio padrão na maioria das instâncias. Um resultado particularmente notável é o da instância tba4, onde o VNS convergiu para

o valor ótimo (0.307) em todas as 30 execuções. Esta convergência consistente é explicada pelo fato de que, para esta instância específica, a solução ótima global (confirmada pelo Solver Simples) coincide com a solução ótima que respeita contiguidade (confirmada pelo Solver Melhorado), veremos claramente isso nos experimentos das seções seguintes. Isso demonstra que nem sempre a imposição de contiguidade resulta em degradação da qualidade da solução, dependendo da estrutura particular dos tempos de processamento da instância. O tempo médio de execução manteve-se estável entre 66-98 segundos, com exceção da instância tba7 que apresentou maior complexidade.

5.2 VNS com heurística sequencial determinística

Para esta análise comparativa, adotou-se um protocolo experimental complementar com 10 execuções independentes por instância. Crucialmente, a solução inicial foi redefinida como uma **heurística sequencial determinística**, substituindo a heurística gulosa aleatória utilizada anteriormente. Esta mudança foi necessária porque **a heurística gulosa produz soluções iniciais diferentes a cada execução** devido aos cortes aleatórios e permutações de operadores, impossibilitando o cálculo consistente de gaps de melhoria.

A nova heurística sequencial divide as tarefas uniformemente entre os operadores em ordem (Operador 1 recebe tarefas 1 a $\lfloor n/m \rfloor$, Operador 2 recebe as próximas $\lfloor n/m \rfloor$ tarefas, etc.), garantindo que **todas as execuções partam do mesmo baseline**. Esta abordagem oferece: (1) **consistência experimental**, mesma SI em todas as execuções, permitindo cálculo direto do gap SI→SF; e (2) **interpretabilidade prática**, representa uma estratégia de divisão de trabalho naturalmente intuitiva, correspondendo a como um gestor dividiria tarefas sem otimização.

Instância	**SI**	**SF**	**Gap SI→SF (%)**	**Tempo VNS (s)**
tba1	2.446	0.564	77.0	82.8
tba2	1.572	0.523	66.8	69.3
tba3	2.490	0.489	80.4	66.0
tba4	2.290	0.307	86.6	65.5
tba5	3.440	1.500	56.4	69.3
tba6	3.556	0.575	83.8	75.7
tba7	2.142	0.594	72.3	74.9
tba8	2.920	0.846	71.0	78.1
tba9	1.984	0.584	70.6	82.2
tba10	2.921	1.326	54.6	73.4

Tabela 3: Análise do VNS com hurística determinística

A Tabela 3 revela mais uma vez aspectos fundamentais sobre a eficácia do VNS. A análise dos gaps de melhoria SI→SF demonstra que o algoritmo consegue melhorias substanciais em todas as instâncias, variando de 54.6% (tba10) a 86.6% (tba4). Estes

resultados evidenciam que, embora a divisão sequencial uniforme forneça um ponto de partida factível, o processo de busca estruturado do VNS é fundamental para explorar eficientemente o espaço de soluções e atingir configurações de alta qualidade que respeitam a restrição de contiguidade.

A consistência dos tempos de execução, mantendo-se na faixa de 65-83 segundos independentemente do tamanho da instância, o torna altamente competitivo, pois demonstra a estabilidade computacional do algoritmo e sua adequação para aplicações práticas onde previsibilidade temporal é importante. Esta característica é especialmente valiosa em ambientes produtivos que requerem replanejamento frequente ou otimização em tempo real.

5.3 Comparação com Heurísticas Construtivas

A Tabela 4 compara o desempenho do VNS (melhor resultado das 30 execuções) com as heurísticas construtivas implementadas.

Instância	**VNS**	**Heur. Gulosa**	**Heur. Contig.**	**Gap Gulosa (%)**	**Gap Contig. (%)**
tba1	0.564	0.327	3.195	-42.0	+466.5
tba2	0.523	0.539	2.201	+3.1	+320.8
tba3	0.475	0.308	2.942	-35.2	+519.4
tba4	0.307	0.363	1.685	+18.2	+448.5
tba5	1.491	1.034	3.911	-30.7	+162.3
tba6	0.567	0.541	3.034	-4.6	+435.3
tba7	0.594	0.292	2.236	-50.8	+276.4
tba8	0.846	0.846	2.463	0.0	+191.1
tba9	0.584	0.351	1.671	-39.9	+186.3
tba10	1.326	0.788	3.737	-40.6	+181.9

Tabela 4: Comparação entre VNS e heurísticas construtivas

É importante observar que a heurística gulosa, apesar de não respeitar a restrição de contiguidade, não representa um limite inferior teórico para o problema. Embora a heurística tome decisões localmente ótimas para cada tarefa individual (escolhendo sempre o operador que minimiza o makespan naquele momento), essas escolhas míopes podem levar a soluções subótimas globalmente. Consequentemente, o VNS pode encontrar soluções contíguas superiores à heurística gulosa através de sua capacidade de reavaliar e reorganizar as atribuições globalmente via busca local e múltiplas estruturas de vizinhança.

Os resultados revelam um padrão interessante: a heurística gulosa, que não respeita a restrição de contiguidade, frequentemente obtém soluções com makespan inferior ao VNS (gaps negativos), mas viola a estrutura sequencial exigida pelo problema. Em contraste, a heurística com contiguidade, que força o reagrupamento das tarefas em segmentos contíguos, produz soluções significativamente piores, com degradação média superior

a 300% em relação ao VNS.

Este resultado evidencia o trade-off fundamental entre qualidade da solução e satisfação da restrição de contiguidade. O VNS consegue explorar eficientemente este espaço de soluções através de suas vizinhanças estruturadas, obtendo soluções que respeitam a contiguidade com qualidade superior às heurísticas construtivas que também satisfazem esta restrição.

5.4 Comparação com Solvers Exatos

A Tabela 5 apresenta a comparação entre o VNS e as formulações MILP resolvidas via GLPK, considerando os diferentes limites de tempo estabelecidos.

Instância	**VNS**	**Sol. Simples**	**Sol. Melhorado**	**Status Simples**	**Status Melhorado**
tba1	0.564	0.299	1.027	ÓTIMO (20min)	TIMEOUT (3h)
tba2	0.523	0.386	0.809	ÓTIMO (0.1s)	TIMEOUT (3h)
tba3	0.490	0.308	0.481	ÓTIMO (0.1s)	TIMEOUT (3h)
tba4	0.307	0.307	0.307	ÓTIMO (0.01s)	ÓTIMO (0.6s)
tba5	1.491	0.568	2.253	ÓTIMO (25s)	TIMEOUT (3h)
tba6	0.567	0.335	0.652	ÓTIMO (22s)	TIMEOUT (3h)
tba7	0.594	0.200	0.871	ÓTIMO (0.4s)	TIMEOUT (3h)
tba8	0.846	0.846	0.974	ÓTIMO (0.1s)	TIMEOUT (3h)
tba9	0.584	0.282	0.584	ÓTIMO (0.02s)	ÓTIMO (71s)
tba10	1.326	0.567	2.291	ÓTIMO (6s)	TIMEOUT (3h)

Tabela 5: Comparação entre VNS e solvers exatos

Instância	**Violações Solver Simples**	**Violações Solver Melhorado**
tba1	23	0
tba2	16	0
tba3	13	0
tba4	9	0
tba5	29	0
tba6	21	0
tba7	16	0
tba8	15	0
tba9	7	0
tba10	30	0
Média	**17.9**	**0.0**

Tabela 6: Violações de contiguidade nas soluções dos solvers

A Tabela 6 quantifica as violações de contiguidade nas soluções obtidas pelos solvers. Uma violação ocorre quando tarefas atribuídas ao mesmo operador não formam um segmento contíguo (por exemplo, operador 1 recebe tarefas 2, 5 e 7). O Solver Simples, que

não inclui restrições de contiguidade, produz soluções com múltiplas violações, variando de 7 a 30 por instância. Em contraste, o Solver Melhorado, que incorpora explicitamente as restrições de contiguidade na formulação MILP, garante matematicamente que todas as soluções respeitem esta propriedade, resultando em zero violações em todos os casos.

Os resultados evidenciam comportamentos distintos entre as duas formulações MILP:

Solver Simples (sem contiguidade): Encontrou soluções ótimas para todas as instâncias dentro do limite de 20 minutos, geralmente em poucos segundos. Como esperado, essas soluções apresentam makespans inferiores ao VNS, mas violam sistematicamente a restrição de contiguidade, conforme detalhado na Tabela 6.

Solver Melhorado (com contiguidade): A inclusão das restrições de contiguidade aumentou drasticamente a complexidade computacional. O solver encontrou soluções ótimas apenas para as instâncias menores (tba4 e tba9), atingindo timeout de 3 horas nas demais. Nas instâncias onde convergiu, as soluções respeitam perfeitamente a contiguidade.

Comparando soluções que respeitam contiguidade, o VNS demonstrou desempenho superior ao Solver Melhorado em 8 das 10 instâncias. Nos casos tba4 e tba9, onde o solver encontrou soluções ótimas, o VNS também convergiu para o mesmo valor ótimo (tba4, tba9). Este resultado destaca a eficácia do VNS para resolver instâncias de tamanho prático onde métodos exatos tornam-se computacionalmente complicados.

5.5 Comparação metódos que respeitam contiguidade

Para uma comparação direta entre métodos que respeitam a restrição de contiguidade, a Tabela 7 consolida os resultados dos três algoritmos factíveis para implementação prática.

Instância	VNS	Heur. Contig.	Solver Melhor.	Melhor	Status Solver
tba1	0.564	3.195	1.027	VNS	TIMEOUT
tba2	0.523	2.201	0.809	VNS	TIMEOUT
tba3	0.490	2.942	0.491	VNS	TIMEOUT
tba4	0.307	1.685	0.307	Empate	ÓTIMO
tba5	1.491	3.911	2.253	VNS	TIMEOUT
tba6	0.567	3.034	0.652	VNS	TIMEOUT
tba7	0.594	2.236	0.871	VNS	TIMEOUT
tba8	0.846	2.463	0.974	VNS	TIMEOUT
tba9	0.584	1.671	0.584	Empate	ÓTIMO
tba10	1.326	3.737	2.291	VNS	TIMEOUT
Vitórias	**8**	**0**	**0**	**2 empates**	**2/10 ótimos**

Tabela 7: Comparação entre métodos que respeitam contiguidade

Esta análise evidencia que o VNS obtém soluções superiores em 8 das 10 instâncias, empatando nas outras 2 onde o Solver Melhorado conseguiu convergir para a solução

ótima. O resultado confirma a eficácia do VNS para problemas de tamanho prático, especialmente considerando que o método exato conseguiu convergir em apenas 2 das 10 instâncias testadas.

6 Análise e Discussão

Os resultados experimentais revelam aspectos importantes sobre o comportamento dos diferentes algoritmos no problema do trabalho balanceado. Esta seção analisa os principais achados e suas implicações.

6.1 Eficácia do VNS

O algoritmo VNS demonstrou robustez e consistência notáveis nos experimentos realizados. A análise das 30 execuções independentes por instância revela baixa variabilidade nos resultados, com desvios padrão inferiores a 0.17 em todas as instâncias. Particularmente impressionante é o caso da instância tba4, onde todas as execuções convergiram para o valor ótimo (0.307), confirmado posteriormente pelo Solver Melhorado.

O VNS demonstra sua utilidade através da consistência dos resultados obtidos. O algoritmo garante exploração sistemática do espaço de soluções e oferece robustez estatística através das múltiplas execuções, fornecendo confiabilidade na qualidade das soluções finais.

6.2 Trade-off entre Qualidade e Contiguidade

Os resultados experimentais evidenciam claramente o trade-off fundamental entre qualidade da solução e satisfação da restrição de contiguidade. Este fenômeno manifesta-se de forma consistente em três níveis distintos de análise.

No nível das heurísticas construtivas, observa-se que a heurística gulosa simples, que ignora contiguidade, frequentemente produz soluções superiores ao VNS em termos de makespan puro. Contudo, ao forçar a contiguidade através do reagrupamento (heurística com contiguidade), a degradação na qualidade é dramática, com piora média de 300% em relação ao VNS. Este resultado quantifica o custo computacional de impor contiguidade de forma ingênua.

No nível dos métodos exatos, o Solver Simples (sem contiguidade) encontra consistentemente soluções ótimas com makespans inferiores tanto ao VNS quanto ao Solver Melhorado. A inclusão das restrições de contiguidade no Solver Melhorado não apenas degrada a qualidade das soluções obtidas, mas também torna o problema computacionalmente complicado para instâncias de tamanho moderado.

O VNS posiciona-se estrategicamente neste trade-off, conseguindo soluções que respeitam contiguidade com qualidade significativamente superior às heurísticas construtivas que também satisfazem esta restrição. Em particular, nas instâncias onde o Solver Melhorado conseguiu convergir (tba4 e tba9), o VNS alcançou soluções igualmente satisfatórias, demonstrando sua capacidade de aproximar soluções ótimas factíveis em tempo computacional razoável.

Este comportamento sugere que o VNS explora eficientemente o subespaço de soluções contíguas, evitando as armadilhas das decisões míopes das heurísticas construtivas enquanto mantém tratabilidade computacional superior aos métodos exatos.

6.3 Limitações dos Métodos Exatos

Os experimentos com formulações MILP revelaram limitações significativas dos métodos exatos para o problema do trabalho balanceado, particularmente quando a restrição de contiguidade é considerada.

O Solver Simples, embora encontre soluções ótimas rapidamente, produz resultados impraticáveis para aplicações reais devido às múltiplas violações de contiguidade. Com uma média de 18 violações por instância, as soluções obtidas requerem reorganização substancial para implementação prática, eliminando as garantias de otimalidade.

A formulação completa (Solver Melhorado) demonstra as limitações fundamentais da abordagem exata para este problema. A inclusão das restrições de contiguidade aumenta significativamente a complexidade computacional: das 10 instâncias testadas, apenas 2 convergiram para soluções ótimas dentro do limite de 3 horas.

Instância	**Variáveis**	**Rest. Simp.**	**Rest. Melh.**	**Restr. Contiguidade**	**Melhorado**
tba1	482	50	8.240	8.190	TIMEOUT
tba2	481	47	7.022	6.975	TIMEOUT
tba3	352	40	4.265	4.225	TIMEOUT
tba4	331	37	3.187	3.150	ÓTIMO
tba5	421	52	8.252	8.200	TIMEOUT
tba6	482	50	8.240	8.190	TIMEOUT
tba7	449	46	6.556	6.510	TIMEOUT
tba8	298	38	3.613	3.575	TIMEOUT
tba9	221	32	2.132	2.100	ÓTIMO
tba10	421	52	8.252	8.200	TIMEOUT

Tabela 8: Complexidade computacional das formulações MILP

A Tabela 8 ilustra claramente a explosão combinatorial: enquanto as formulações simples possuem dezenas de restrições, a inclusão da contiguidade resulta em milhares. A análise da estrutura do problema revela que o número de restrições de contiguidade cresce quadraticamente com o tamanho da instância ($O(m \cdot n^2)$) (será provado na subseção

seguinte), explicando por que apenas as menores instâncias (tba4 e tba9) conseguiram convergir dentro do limite de tempo.

Estes resultados confirmam que, para instâncias de tamanho prático, métodos exatos tornam-se computacionalmente complicadas quando a contiguidade deve ser respeitada. Esta limitação fundamenta a necessidade de abordagens heurísticas eficazes como o VNS proposto, que consegue produzir soluções de alta qualidade em tempo computacional razoável para toda a gama de instâncias testadas.

6.4 Análise de Complexidade das Restrições

A formulação MILP do problema de trabalho balanceado apresenta duas variantes que diferem drasticamente em termos de complexidade computacional.

6.4.1 Formulação Básica (sem contiguidade)

A formulação básica possui:

- **Variáveis**: $nm + 1$ (onde $y_{ik} \in \{0, 1\}$ e $T \geq 0$)
- **Restrições de atribuição**: $\sum_{k=1}^{m} y_{ik} = 1$ para $i = 1, \ldots, n$ (n restrições)
- **Restrições de makespan**: $\sum_{i=1}^{n} p_{ik} \cdot y_{ik} \leq T$ para $k = 1, \ldots, m$ (m restrições)

Complexidade total: $O(n + m)$ restrições.

6.4.2 Formulação com Contiguidade

Para garantir que as tarefas atribuídas a cada operador sejam contíguas, adiciona-se a restrição:

$$y_{ik} - y_{(i+1)k} + y_{jk} \leq 1 \tag{4}$$

para cada operador $k \in \{1, \ldots, m\}$, cada tarefa $i \in \{1, \ldots, n-1\}$ e cada tarefa não consecutiva $j \in \{i+2, \ldots, n\}$.

6.4.3 Derivação Matemática da Complexidade

O número de restrições de contiguidade por operador é calculado como:

$$\text{Restrições por operador} = \sum_{i=1}^{n-1} \sum_{j=i+2}^{n} 1 \tag{5}$$

$$= \sum_{i=1}^{n-1} (n - i - 1) \tag{6}$$

$$= \sum_{i=1}^{n-1} (n - 1 - i) \tag{7}$$

$$= \frac{(n-1)(n-2)}{2} \tag{8}$$

Para m operadores, o total de restrições de contiguidade é:

$$\text{Restrições de contiguidade} = m \cdot \frac{(n-1)(n-2)}{2} = \frac{m(n-1)(n-2)}{2} \tag{9}$$

Complexidade total: $O(n + m + mn^2) = O(mn^2)$ restrições.

6.4.4 Exemplo Numérico: Instância tba1

Para a instância tba1 com $n = 37$ tarefas e $m = 13$ operadores:

- **Restrições básicas**: $37 + 13 = 50$
- **Restrições de contiguidade**: $13 \cdot \frac{(37-1)(37-2)}{2} = 13 \cdot \frac{36 \cdot 35}{2} = 13 \cdot 630 = 8.190$
- **Total**: $50 + 8.190 = 8.240$ restrições
- **Fator de explosão**: $\frac{8.240}{50} = 164,8\times$

6.4.5 Implicações Computacionais

A transformação de uma complexidade $O(n + m)$ para $O(mn^2)$ resulta em uma explosão combinatória que torna a formulação com contiguidade complicada para instâncias de tamanho prático. Esta análise justifica matematicamente:

1. A eficiência do *solver* simples (poucas restrições)
2. Os timeouts do *solver* melhorado (milhares de restrições)
3. A competitividade de meta-heurísticas como VNS para este problema

A complexidade quadrática em n das restrições de contiguidade constitui o principal gargalo computacional, explicando por que abordagens exatas tornam-se complicadas para instâncias reais do problema de trabalho balanceado.

6.5 Implicações Práticas

Os resultados obtidos possuem implicações significativas para a aplicação prática do problema do trabalho balanceado em ambientes produtivos reais.

Viabilidade Computacional: O VNS demonstra ser a única abordagem viável para instâncias de tamanho prático quando a contiguidade é obrigatória. Com tempo de execução consistente entre 66-98 segundos independentemente do tamanho da instância, o algoritmo oferece previsibilidade temporal essencial para sistemas de planejamento em tempo real. Em contraste, os métodos exatos falham sistematicamente para instâncias com mais de 25 tarefas.

Qualidade vs. Flexibilidade: O trade-off identificado entre qualidade da solução e satisfação da contiguidade reflete decisões estratégicas em ambientes produtivos. Organizações podem optar por: (1) soluções de menor makespan que requerem reorganização posterior (heurística gulosa), (2) soluções automaticamente implementáveis com qualidade razoável (VNS), ou (3) soluções subótimas mas de implementação trivial (heurística com contiguidade). O VNS posiciona-se como o melhor compromisso entre estes extremos.

Robustez Operacional: A consistência do VNS (baixo desvio padrão) é crucial para ambientes produtivos onde variabilidade nas soluções pode gerar instabilidade operacional. A capacidade de obter soluções similares em múltiplas execuções oferece confiabilidade para tomada de decisões gerenciais.

Escalabilidade: Os resultados sugerem que o VNS mantém eficácia mesmo em instâncias maiores, enquanto métodos exatos degradam rapidamente. Esta característica é fundamental para aplicações industriais onde o número de tarefas e operadores pode crescer significativamente.

Implementação Imediata: Diferentemente das soluções do Solver Simples, que requerem pós-processamento para eliminar violações de contiguidade, as soluções do VNS são diretamente implementáveis, reduzindo custos operacionais e riscos de implementação.

7 Conclusões

Este trabalho apresentou a implementação e avaliação de um algoritmo Variable Neighborhood Search (VNS) para o problema do trabalho balanceado com restrições de contiguidade. Os experimentos realizados em 10 instâncias de teste demonstraram a eficácia da abordagem proposta em comparação com heurísticas construtivas e métodos exatos.

7.1 Principais Contribuições

O VNS desenvolvido oferece uma solução prática e eficiente para o problema estudado, apresentando as seguintes características principais:

- **Robustez:** Baixa variabilidade nos resultados (desvio padrão $< 0{,}17$) em 30 execuções independentes por instância
- **Eficiência computacional:** Tempo de execução consistente entre 66-98 segundos, independente do tamanho da instância
- **Qualidade das soluções:** Superiores às heurísticas construtivas que respeitam contiguidade, com gaps superiores a 300% em favor do VNS
- **Viabilidade prática:** Soluções diretamente implementáveis sem necessidade de pós-processamento

7.2 Insights sobre o Problema

Os experimentos revelaram características importantes sobre a natureza do problema do trabalho balanceado:

- O trade-off entre qualidade da solução e satisfação da restrição de contiguidade é significativo, com degradação de até 519% quando contiguidade é imposta ingenuamente
- Métodos exatos tornam-se computacionalmente complicados para instâncias com mais de 25 tarefas quando contiguidade é obrigatória

7.3 Considerações Finais

O algoritmo VNS proposto demonstrou ser uma abordagem eficaz para o problema do trabalho balanceado, oferecendo o melhor compromisso entre qualidade da solução, tempo computacional e satisfação das restrições práticas. Os resultados confirmam a superioridade das meta-heurísticas sobre métodos exatos para problemas de otimização combinatória de complexidade prática, especialmente quando restrições estruturais como contiguidade devem ser respeitadas.

Para aplicações industriais, o VNS oferece uma ferramenta confiável e eficiente para otimização do balanceamento de cargas de trabalho, contribuindo para melhoria da eficiência operacional em ambientes produtivos.

Referências

[1] Souza, M.J.F. *Visão do Algoritmo VNS*. Departamento de Computação, Universidade Federal de Ouro Preto. Disponível em: http://www.decom.ufop.br/marcone/Disciplinas/InteligenciaComputacional/VNS.pdf. Acesso em: 19 jun. 2025.

[2] Mladenovic, N. *ICVNS2021: Tutorial by Prof. Mladenovic - Introduction to Variable Neighborhood Search metaheuristic*. [Vídeo]. Tutorial apresentado no ICVNS-21, Khalifa University, Abu Dhabi, UAE, 21 mar. 2021. Disponível em: https://www.youtube.com/watch?v=yL3jIX3XnWY. Acesso em: 20 jun. 2025.