

Trabalho Prático Final – GCC118/PCC540

Resolução do Problema ALWABP via Modelo MILP e Metaheurística VNS

Caio Bueno Finocchio Martins Tobias Maugus Bueno Cougo

Universidade Federal de Lavras (UFLA)
Departamento de Ciência da Computação

1 Introdução

O problema de *Assembly Line Worker Assignment and Balancing Problem* (ALWABP) consiste em designar tarefas a estações de trabalho e, simultaneamente, alocar trabalhadores a essas estações, levando em conta tempos de processamento heterogêneos e restrições de incapacidade. O objetivo é minimizar o tempo de ciclo da linha de montagem (C_{\max}), respeitando precedências tecnológicas e limitações de cada trabalhador.

O ALWABP generaliza o problema clássico de balanceamento de linhas, pois considera que trabalhadores distintos podem executar a mesma tarefa em tempos diferentes e, em alguns casos, podem ser completamente incapazes de realizá-la. Trata-se, portanto, de um problema de otimização combinatória de alta complexidade, para o qual modelos exatos podem se tornar inviáveis em instâncias maiores. Neste trabalho, são apresentados: (i) uma formulação do ALWABP como modelo de Programação Linear Inteira Mista (MILP), resolvida via solver Gurobi; e (ii) uma metaheurística *Variable Neighborhood Search* (VNS), desenvolvida para obter boas soluções em tempo computacional reduzido.

2 Formulação do problema como MILP

Sejam: $N = \{1, \dots, n\}$ o conjunto de tarefas, $W = \{1, \dots, k\}$ o conjunto de trabalhadores e $S = \{1, \dots, m\}$ o conjunto de estações, assumindo-se $m = k$, isto é, uma estação por trabalhador. O conjunto de precedências é denotado por $P \subseteq N \times N$, onde $(i, j) \in P$ indica que a tarefa i deve ser concluída antes de j . O tempo de processamento da tarefa i pelo trabalhador w é dado por t_{wi} , podendo assumir um valor muito grande (ou infinito) para modelar incapacidade. Seja, ainda, $I_w \subseteq N$ o subconjunto de tarefas que o trabalhador w não consegue executar.

As variáveis de decisão principais são:

- $C_{\max} \in \mathbb{R}_+$: tempo de ciclo da linha de produção (a ser minimizado);
- $y_{si} \in \{0, 1\}$: vale 1 se a tarefa i é atribuída à estação s ;
- $z_{ws} \in \{0, 1\}$: vale 1 se o trabalhador w é alocado à estação s ;

- $u_{wis} \in \{0, 1\}$: variável auxiliar para linearizar $y_{si} \cdot z_{ws}$, indicando que a tarefa i é executada na estação s pelo trabalhador w .

A função objetivo busca minimizar o tempo de ciclo:

$$\min C_{\max}. \quad (1)$$

As principais restrições são:

Atribuição de tarefas e trabalhadores Cada tarefa é atribuída a exatamente uma estação:

$$\sum_{s \in S} y_{si} = 1, \quad \forall i \in N. \quad (2)$$

Cada trabalhador é alocado a exatamente uma estação:

$$\sum_{s \in S} z_{ws} = 1, \quad \forall w \in W. \quad (3)$$

Cada estação possui exatamente um trabalhador:

$$\sum_{w \in W} z_{ws} = 1, \quad \forall s \in S. \quad (4)$$

Linearização e incapacidade A variável auxiliar u_{wis} lineariza o produto $y_{si}z_{ws}$:

$$u_{wis} \leq y_{si}, \quad \forall i \in N, w \in W, s \in S, \quad (5)$$

$$u_{wis} \leq z_{ws}, \quad \forall i \in N, w \in W, s \in S, \quad (6)$$

$$u_{wis} \geq y_{si} + z_{ws} - 1, \quad \forall i \in N, w \in W, s \in S. \quad (7)$$

Para tarefas em que o trabalhador é incapaz ($i \in I_w$), impõe-se:

$$u_{wis} = 0, \quad \forall i \in I_w, w \in W, s \in S. \quad (8)$$

Restrição de tempo de ciclo Para cada estação, o tempo total de processamento das tarefas atribuídas não pode exceder C_{\max} :

$$\sum_{i \in N} \sum_{w \in W} t_{wi} u_{wis} \leq C_{\max}, \quad \forall s \in S. \quad (9)$$

Precedências Se $(i, j) \in P$, a estação que executa i deve ser anterior ou igual à que executa j :

$$\sum_{s \in S} s y_{si} \leq \sum_{s \in S} s y_{sj}, \quad \forall (i, j) \in P. \quad (10)$$

O solver Gurobi é utilizado para resolver este modelo exato, com limite de tempo de 20 minutos por instância e coleta dos valores de solução, tempo computacional e *gap* final.

3 Algoritmo proposto: Variable Neighborhood Search (VNS)

Além do modelo MILP, foi implementada uma metaheurística *Variable Neighborhood Search* (VNS) para resolver o ALWABP. A VNS é baseada na exploração sistemática de diferentes estruturas de vizinhança, alternando fases de perturbação (*shaking*) e busca local.

3.1 Representação da solução

Uma solução S é representada por:

- um vetor de estações das tarefas, Y , em que $Y[i]$ indica a estação à qual a tarefa i está atribuída;
- um vetor de trabalhadores por estação, Z , em que $Z[s]$ indica qual trabalhador está alocado à estação s .

Dessa forma, cada par (estação, trabalhador) define a capacidade de execução das tarefas naquela posição da linha.

3.2 Função de avaliação

Dada uma solução (Y, Z) , o tempo de processamento de cada estação é calculado somando-se, para todas as tarefas atribuídas àquela estação, os tempos $t_{w,i}$ do trabalhador alocado $w = Z[s]$. O valor de $C_{\max}(S)$ é definido como o maior tempo entre todas as estações.

A solução é considerada factível se:

- todas as restrições de precedência são respeitadas, isto é, nenhuma tarefa é atribuída a uma estação posterior à estação de algum de seus predecessores;
- nenhuma tarefa é atribuída a uma estação cujo trabalhador alocado seja incapaz de executá-la.

Caso alguma dessas condições não seja satisfeita, a solução recebe uma penalização por infactibilidade (um valor de C_{\max} muito elevado), garantindo que soluções factíveis sejam sempre preferidas no processo de busca.

3.3 Geração da solução inicial

A solução inicial é construída de forma gulosa, em duas etapas:

1. **Alocação de trabalhadores às estações.** Gera-se uma permutação aleatória dos trabalhadores e cada trabalhador é atribuído a exatamente uma estação. Assim, a numeração das estações é determinada pela ordem dessa permutação.
2. **Atribuição gulosa de tarefas.** Constrói-se uma ordenação topológica das tarefas a partir do grafo de precedências. Em seguida, percorrem-se as tarefas nessa ordem e, para cada tarefa i , busca-se a primeira estação s (na ordem das estações) tal que: (i) o trabalhador $Z[s]$ seja capaz de executar a tarefa i , e (ii) todas as tarefas predecessoras de i tenham sido atribuídas a estações com índice menor ou igual a s . A tarefa é então fixada nessa estação.

Esse procedimento produz uma solução inicial factível (quando possível), que tende a respeitar precedências e capacidades de maneira simples e serve de ponto de partida para a VNS.

3.4 Estruturas de vizinhança

Na fase de *shaking*, são utilizadas $K_{\max} = 3$ vizinhanças:

k	Vizinhança	Descrição
1	<i>Task Swap</i>	troca as estações de duas tarefas distintas;
2	<i>Task Reassignment</i>	move uma tarefa de sua estação atual para outra estação;
3	<i>Worker Swap</i>	troca os trabalhadores alocados a duas estações.

A vizinhança é escolhida de acordo com o valor corrente de k , e a solução é aleatoriamente perturbada dentro dessa estrutura antes de aplicar a busca local.

3.5 Busca local (VND)

Após o *shaking*, aplica-se um *Variable Neighborhood Descent* (VND) com duas vizinhanças, exploradas em esquema de *first improvement*:

ℓ	Vizinhança de busca local	Descrição
1	<i>Task Reassignment</i>	tenta mover tarefas individualmente entre estações;
2	<i>Worker Swap</i>	tenta trocar trabalhadores entre estações.

Se um movimento que melhora C_{\max} é encontrado na vizinhança atual, a solução é atualizada e o VND retorna à vizinhança $\ell = 1$. Caso contrário, passa-se para a próxima vizinhança. O VND termina quando nenhuma das vizinhanças produz melhoria.

3.6 Critérios de parada

O algoritmo VNS interno utiliza um limite fixo de iterações $VNS_MAX_ITER = 500$: o laço principal é encerrado assim que esse número de iterações é atingido.

Além disso, no contexto experimental do trabalho, foi imposto um **limite de tempo por instância**:

- Para cada instância, são realizadas sucessivas replicações da VNS, cada uma com uma *seed* diferente, até um máximo de $NUM_REPLICATIONS = 13$ replicações.
- O tempo é cronometrado a partir do início da primeira replicação.
- Se o tempo acumulado das replicações para uma dada instância alcançar $TIME_LIMIT = 900$ segundos (15 minutos), novas replicações não são iniciadas para essa instância.

Dessa forma, o experimento combina um critério de parada por número de iterações dentro de cada execução da VNS e um critério adicional de tempo de CPU por instância quando se considera o conjunto de replicações.

4 Resultados computacionais e análise

4.1 Configuração experimental

Os experimentos foram conduzidos em um notebook com as seguintes características de hardware:

- Processador: Intel® Core™ Ultra 7 155H (3,80 GHz);
- Memória RAM instalada: 16 GB;
- Sistema operacional: 64 bits, arquitetura x64.

O modelo exato foi resolvido com o solver Gurobi, com limite de 20 minutos por instância. A metaheurística VNS foi implementada em Python e executada com os parâmetros resumidos a seguir:

- $VNS_MAX_ITER = 500$;
- $VNS_K_MAX = 3$;
- Conjunto de sementes para instâncias pequenas: $\{0, 1, 7, 13, 17, 23, 31, 42, 47, 73, 101, 202, 606\}$;
- Conjunto de sementes para instâncias grandes: $\{97, 131, 197, 223, 281, 313, 487, 557, 613, 733, 1707, 1905, 1936\}$;
- Limite de tempo por instância para a heurística: $TIME_LIMIT = 600$ segundos.

As instâncias do VNS são executadas em paralelo (*ProcessPoolExecutor*), mas, para cada instância, as replicações são executadas em série dentro de um único processo, respeitando o limite de tempo. O tempo de cada instância é registrado como o intervalo entre o início da primeira replicação e o término da última replicação de fato executada (ou o momento em que o limite de tempo foi atingido).

4.2 Métricas comparadas

Para cada instância, são reportados:

- SI : valor da solução inicial produzida pela heurística;
- SF : melhor valor de solução final obtido pela VNS, considerando todas as replicações executadas;
- SO : valor ótimo ou melhor *upper bound* conhecido, extraído de um arquivo de referência;
- $Improvement\%$ (coluna **Improvement%** na Tabela 1): desvio percentual da solução final em relação à inicial, calculado como

$$100 \times \frac{SI - SF}{SI};$$

- *Gap_to_Optimal%* (coluna **GT0%** na Tabela 1): desvio percentual da solução final em relação ao valor ótimo, calculado como

$$100 \times \frac{SF - SO}{SF};$$

- *Total_Time_s* (coluna **VnsTime**): tempo total consumido pela heurística para aquela instância (soma das replicações executadas até o limite de tempo);
- *SOL_GUROBI* (coluna **SG**): valor objetivo retornado pelo Gurobi;
- *TIME_GUROBI* (coluna **GurobiTime**): tempo de execução do Gurobi;
- *GAP_GUROBI_OPT* (coluna **GG0%**): *gap* percentual final do solver em relação ao ótimo.

Nas estatísticas agregadas (tempo médio, mínimo, máximo, total), o tempo total corresponde à soma dos tempos individuais das instâncias. No caso da VNS, como as instâncias são executadas em paralelo, esses tempos representam o esforço computacional somado, e não necessariamente o tempo de parede observado na execução completa.

4.3 Discussão qualitativa dos resultados

A Tabela 1 apresenta os resultados completos das 48 instâncias, incluindo, para cada uma, os valores de solução inicial (*SI*), solução final obtida pela VNS (*SF*), melhor valor conhecido ou ótimo (*SO*), solução retornada pelo Gurobi (*SOL_GUROBI*), tempos de execução e respectivos *gaps*. Esses dados constituem a base para as análises comparativas entre o modelo exato e a metaheurística.

De forma global, o experimento contempla 48 instâncias. Em todas elas o Gurobi produziu uma solução viável, atingindo o ótimo em 28 instâncias. A VNS alcançou exatamente o valor ótimo em 6 instâncias. Os tempos médios foram de aproximadamente 591,34 s para o Gurobi e 434,94 s para a VNS, com tempos totais acumulados de 28384,12 s (Gurobi) e 20877,05 s (VNS). Em média, a VNS melhora a solução inicial em cerca de 74,75%, mas ainda fica, em média, a 178,50% acima do ótimo na coluna *Gap_to_Optimal%*.

Famílias hes e ros (instâncias menores) Nas famílias **hes** e **ros** a VNS apresenta um comportamento bem mais robusto:

- Melhorias médias superiores a 80% (**hes**) e 65% (**ros**) em relação à solução inicial (*SI*);
- *Gap* médio em relação ao ótimo de apenas 11,41% para **hes** e 13,49% para **ros**;
- A VNS encontra o ótimo em 3 de 12 instâncias em cada família (por exemplo, 12_hes, 1_hes, 71_hes e 12_ros, 1_ros, 72_ros);
- O Gurobi é extremamente eficiente nesses casos, com tempos médios da ordem de poucos segundos (cerca de 3 s em **hes** e 1,4 s em **ros**), sempre com *gap* final zero.

Ou seja, para as instâncias menores, o modelo MILP resolvido pelo Gurobi é plenamente competitivo e, na prática, domina a heurística: encontra o ótimo rápido, enquanto a VNS gasta mais tempo, mas ainda assim produz soluções de boa qualidade e serve como alternativa heurística caso o solver exato não esteja disponível.

Tabela 1: Resultados completos por instância.

Instance	Seed	SI	SF	SO	SG	GurobiTime	GGO%	VnsTime	Improvement%	GTO%
11_hes	1	443.0	172.0	169.0	169.0	0.28	0.0	171.34	61.17	1.78
11_ros	7	50.0	31.0	30.0	30.0	0.09	0.0	60.91	38.00	3.33
11_ton	281	363.0	178.0	110.0	110.0	958.7	0.0	640.97	50.96	61.82
11_wee	223	955.0	67.0	29.0	35.0	1200.34	82.8571	703.00	92.98	131.03
12_hes	17	1024.0	107.0	107.0	107.0	0.47	0.0	157.45	89.55	0.00
12_ros	7	64.0	27.0	27.0	27.0	0.25	0.0	59.50	57.81	0.00
12_ton	557	465.0	182.0	108.0	108.0	1200.34	50.0	677.02	60.86	68.52
12_wee	223	800.0	71.0	30.0	33.0	1200.21	45.4545	712.28	91.12	136.67
1_hes	17	1024.0	94.0	94.0	94.0	0.55	0.0	196.51	90.82	0.00
1_ros	31	25.0	20.0	20.0	20.0	0.17	0.0	55.38	20.00	0.00
1_ton	557	496.0	150.0	87.0	87.0	964.16	0.0	639.00	69.76	72.41
1_wee	97	279.0	79.0	25.0	29.0	1200.23	44.8276	649.41	71.68	216.00
2_hes	47	457.0	96.0	95.0	95.0	1.76	0.0	200.18	78.99	1.05
2_ros	42	118.0	23.0	22.0	22.0	0.85	0.0	57.94	80.51	4.55
2_ton	487	1178.0	138.0	87.0	91.0	1200.35	36.2637	619.67	88.29	58.62
2_wee	131	270.0	84.0	26.0	28.0	1200.56	35.7143	702.73	68.89	223.08
41_hes	1	160.0	41.0	35.0	35.0	2.25	0.0	275.22	74.38	17.14
41_ros	47	46.0	13.0	10.0	10.0	1.49	0.0	112.04	71.74	30.00
41_ton	281	594.0	133.0	28.0	31.0	1200.34	70.9677	623.80	77.61	375.00
41_wee	97	608.0	79.0	10.0	10.0	1200.39	20.0	853.48	87.01	690.00
42_hes	202	302.0	46.0	40.0	40.0	4.96	0.0	330.26	84.77	15.00
42_ros	47	52.0	11.0	10.0	10.0	2.28	0.0	120.50	78.85	10.00
42_ton	131	492.0	105.0	32.0	35.0	1200.60	74.2857	645.97	78.66	228.12
42_wee	131	270.0	88.0	9.0	13.0	1200.67	92.3077	651.83	67.41	877.78
51_hes	7	256.0	57.0	51.0	51.0	6.12	0.0	372.98	77.73	11.76
51_ros	73	45.0	13.0	11.0	11.0	1.59	0.0	130.09	71.11	18.18
51_ton	97	503.0	79.0	35.0	39.0	1200.30	51.2821	689.09	84.29	125.71
51_wee	131	161.0	68.0	12.0	14.0	1200.61	42.8571	756.60	57.76	466.67
52_hes	0	269.0	55.0	50.0	50.0	5.85	0.0	346.87	79.55	10.00
52_ros	47	26.0	12.0	10.0	10.0	1.33	0.0	124.45	53.85	20.00
52_ton	197	504.0	106.0	43.0	45.0	1200.46	4.4444	657.18	78.97	146.51
52_wee	97	168.0	61.0	9.0	13.0	1200.44	84.6154	601.43	63.69	577.78
61_hes	42	979.0	77.0	66.0	66.0	3.32	0.0	433.80	92.13	16.67
61_ros	7	92.0	20.0	16.0	16.0	1.64	0.0	123.45	78.26	25.00
61_ton	223	873.0	338.0	61.0	73.0	1200.41	38.3562	602.12	61.28	454.10
61_wee	131	685.0	150.0	15.0	20.0	1200.44	45.0	646.85	78.10	900.00
62_hes	42	621.0	70.0	56.0	56.0	4.05	0.0	415.97	88.73	25.00
62_ros	0	61.0	17.0	13.0	13.0	2.95	0.0	147.21	72.13	30.77
62_ton	97	1290.0	269.0	66.0	71.0	1200.29	53.5211	618.77	79.15	307.58
62_wee	223	625.0	105.0	18.0	19.0	1200.36	5.2632	770.13	83.20	483.33
71_hes	0	1039.0	91.0	91.0	91.0	1.39	0.0	268.48	91.24	0.00
71_ros	31	65.0	18.0	15.0	15.0	1.53	0.0	119.79	72.31	20.00
71_ton	131	1349.0	285.0	54.0	73.0	1200.60	67.1233	651.45	78.87	427.78
71_wee	197	373.0	88.0	18.0	19.0	1200.65	52.6316	735.88	76.41	388.89
72_hes	202	643.0	90.0	65.0	65.0	4.94	0.0	282.56	86.00	38.46
72_ros	101	118.0	16.0	16.0	16.0	2.19	0.0	109.64	86.44	0.00
72_ton	313	1157.0	254.0	57.0	59.0	1200.19	45.7627	667.01	78.05	345.61
72_wee	223	747.0	97.0	16.0	21.0	1200.18	42.8571	688.86	87.01	506.25

Famílias *ton* e *wee* (instâncias maiores e mais difíceis) Nas famílias *ton* e *wee* a dificuldade do problema fica evidente. O Gurobi passa a explorar o limite de tempo (1200 s) na maior parte das instâncias:

- Tempos médios do Gurobi em torno de 1160,56 s (*ton*) e 1200,42 s (*wee*), com *gaps* médios finais de aproximadamente 41% (*ton*) e 49,5% (*wee*);
- Mesmo assim, o solver só alcança o ótimo em 3 instâncias *ton* (11_*ton*, 12_*ton*, 1_*ton*) e em apenas 1 instância *wee* (41_*wee*);
- A VNS, por outro lado, melhora bastante a solução inicial: melhoria média de 73,90% em *ton* e 77,11% em *wee*; porém, o *gap* médio em relação ao ótimo explode para 222,65% e 466,46%, respectivamente;
- Os tempos médios da VNS nessas famílias giram em torno de 644,34 s (*ton*) e 706,04 s (*wee*), isto é, cerca de metade a dois terços do tempo gasto pelo Gurobi.

Na prática, isso mostra que as instâncias *ton* e *wee* são de fato as mais difíceis do conjunto: nem o modelo exato atinge sistematicamente o ótimo (mesmo com 20 minutos), nem a heurística consegue se aproximar muito da solução ótima, apesar de melhorar fortemente a solução inicial. Ainda assim, em várias instâncias específicas (como 11_*ton*, 12_*ton*, 1_*ton* e 41_*wee*), o Gurobi serve como referência de qualidade ao fornecer o valor ótimo para comparação com a VNS.

Visão global A partir da Tabela 1, podem ser destacados alguns pontos:

- O uso de múltiplas *seeds* é fundamental: há instâncias em que sementes diferentes produzem valores finais *SF* sensivelmente distintos, permitindo que a VNS encontre, em algumas replicações, soluções muito próximas do ótimo.
- Em média, a VNS reduz o valor da solução em cerca de 75% em relação à construção inicial (*SI*), o que confirma a eficácia do esquema *shaking* + VND no refino da solução.
- Por outro lado, quando o objetivo é obter soluções muito próximas do ótimo em instâncias grandes, a combinação com o modelo exato continua importante: o Gurobi fornece limites inferiores e, em alguns casos, o próprio ótimo, enquanto a VNS explora o espaço de soluções produzindo bons *upper bounds* em tempos menores.

Em resumo, os resultados indicam que a formulação MILP é insubstituível como referência de qualidade para instâncias pequenas e médias, enquanto a VNS se mostra particularmente útil nas instâncias maiores, nas quais o solver exato atinge o limite de tempo com *gaps* elevados. A combinação das duas abordagens fornece um quadro completo: valores ótimos ou quase ótimos em instâncias simples e soluções heurísticas competitivas em instâncias mais difíceis.

5 Conclusão

Este trabalho apresentou uma abordagem híbrida para o ALWABP, combinando um modelo exato em MILP resolvido pelo Gurobi e uma metaheurística VNS implementada em Python. A formulação exata permite obter soluções ótimas ou limites superiores de alta qualidade, enquanto a VNS fornece soluções competitivas em tempos menores, especialmente em instâncias de maior porte.

A construção gulosa da solução inicial, aliada às estruturas de vizinhança definidas (trocas de tarefas, reatribuições e trocas de trabalhadores), mostrou-se adequada para explorar o espaço de soluções respeitando precedências e restrições de incapacidade. O uso de múltiplas *seeds* e de um limite de tempo por instância permitiu avaliar a robustez da heurística, ao mesmo tempo em que garante controle sobre o esforço computacional.

Como trabalhos futuros, podem ser considerados: (i) o uso de estratégias adaptativas para escolha de vizinhanças, (ii) a inclusão de mecanismos de intensificação/diversificação mais sofisticados (e.g., VNS reativo, *path relinking*), e (iii) a extensão do modelo para variações do ALWABP com múltiplas linhas ou objetivos adicionais.

Referências

- Miralles, C., Garcia-Sabater, J. P., Andres, C., & Cardos, M. (2007). Advantages of assembly lines in sheltered work centres for disabled. *International Journal of Production Economics*, 110(1–2), 187–197.
- Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11), 1097–1100.
- Hansen, P., & Mladenović, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3), 449–467.
- Gendreau, M., & Potvin, J. Y. (Eds.). (2019). *Handbook of Metaheuristics* (3rd ed.). Springer.
- Torres, P., & Lima, C. (2018). *Metaheuristics for Logistics*. Wiley.
- Knuth, D. E. (1997). *The Art of Computer Programming, Volume 2: Seminumerical Algorithms* (3rd ed.). Addison-Wesley.