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: $N = \{1, \ldots, n\}$ o conjunto de tarefas, $W = \{1, \ldots, k\}$ o conjunto de trabalhadores e $S = \{1, \ldots, m\}$ 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:

$$\min C_{\max}. \tag{1}$$

As principais restrições são:

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

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

Cada trabalhador é alocado a exatamente uma estação:

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

Cada estação possui exatamente um trabalhador:

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

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

$$u_{wis} \leq y_{si}, \qquad \forall i \in N,\ w \in W,\ s \in S, \tag{5}$$
$$u_{wis} \leq z_{ws}, \qquad \forall i \in N,\ w \in W,\ s \in S, \tag{6}$$
$$u_{wis} \geq y_{si} + z_{ws} - 1, \qquad \forall i \in N,\ w \in W,\ s \in S. \tag{7}$$

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

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

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

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

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

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

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	*Task Swap*	troca as estações de duas tarefas distintas;
2	*Task Reassignment*	move uma tarefa de sua estação atual para outra estação;
3	*Worker Swap*	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	*Task Reassignment*	tenta mover tarefas individualmente entre estações;
2	*Worker Swap*	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: $\{42, 101, 202, 303, 404, 505, 606, 707, 909, 1001, 1707, 1905, 1936\}$;
- $NUM\_REPLICATIONS = 13$ replicações máximas por instância (uma para cada *seed*);
- Limite de tempo por instância para a heurística: $TIME\_LIMIT = 900$ 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%*: desvio percentual da solução final em relação à inicial, calculado como
$$100 \times \frac{SI - SF}{SI};$$
- *Gap_to_Optimal%*: desvio percentual da solução final em relação ao valor ótimo, calculado como
$$100 \times \frac{SF - SO}{SF};$$

- *Total_ Time_ s*: tempo total consumido pela heurística para aquela instância (soma das replicações executadas até o limite de tempo);
- *SOL_ GUROBI*: valor objetivo retornado pelo Gurobi;
- *TIME_ GUROBI*: tempo de execução do Gurobi;
- *GAP_ GUROBI_ OPT*: *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

Nesta seção o aluno deve incluir a tabela consolidada e a análise dos resultados, comparando, para cada instância, o desempenho do VNS e do Gurobi em termos de:

- qualidade da solução (valores de SF, SO, SOL_GUROBI);
- melhoria da solução em relação à inicial (coluna *Improvement%*);
- *gap* da heurística e do solver em relação ao ótimo;
- tempo computacional da heurística versus tempo computacional do solver.

Também é interessante destacar: (i) em quais instâncias o VNS atingiu o ótimo, (ii) em quais instâncias o tempo limite impediu novas replicações e (iii) como a variabilidade introduzida pelas diferentes *seeds* afetou os resultados.

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.