Otimização de Escalas de Tarefas Domésticas em Repúblicas Estudantis: Uma Abordagem Exata Baseada em Restrições

Fernanda Alves Andrade[1], **Hugo Augusto Silva de Faria**[1], **Luiz Henrique de Carvalho**[1]
Marcos Vinício Euzébio[1], **Nicole Bertolino Lamounier Santos**[1]

[1]Departamento de Computação – Universidade Federal de Ouro Preto (UFOP)
Ouro Preto – Minas Gerais – Brasil

Abstract. *Shared housing management presents complex scheduling challenges, particularly regarding the assignment of cleaning duties. This work proposes a mathematical model and a computational solution to optimize the allocation of residents to household chores. We formulate the problem as a cost minimization problem, considering task difficulty and individual unavailability. An exact solution using a backtracking algorithm with pruning is implemented and analyzed. The results demonstrate the efficacy of the model in finding optimal schedules that respect capacity constraints and minimize the overall impact on residents.*

Resumo. *A gestão de moradias compartilhadas apresenta desafios complexos de agendamento, particularmente na atribuição de tarefas de limpeza. Este trabalho propõe um modelo matemático e uma solução computacional para otimizar a alocação de moradores a tarefas domésticas. O problema é formulado como uma minimização de custos, considerando a dificuldade das tarefas e a indisponibilidade individual. Uma solução exata utilizando um algoritmo de backtracking com poda é implementada e analisada. Os resultados demonstram a eficácia do modelo em encontrar escalas ótimas que respeitam restrições de capacidade e minimizam o impacto geral sobre os moradores.*

1. Introdução

A convivência em repúblicas estudantis exige uma organização rigorosa para garantir a manutenção do ambiente e a harmonia entre os moradores. Um dos pontos mais críticos dessa convivência é a divisão de tarefas domésticas (limpeza de cômodos). Distribuições manuais frequentemente resultam em ineficiências ou percepções de injustiça, onde a carga de trabalho não condiz com a disponibilidade de tempo ou a capacidade física dos indivíduos.

Do ponto de vista acadêmico, este desafio de alocação de recursos sob restrições assemelha-se aos clássicos problemas de *Nurse Rostering* (escalas de enfermeiras), amplamente estudados na literatura de Pesquisa Operacional por envolverem a necessidade de conciliar demandas organizacionais com preferências pessoais [Burke et al. 2004].

Neste contexto, a computação pode oferecer ferramentas para automatizar e otimizar essa distribuição. O objetivo deste trabalho é apresentar um modelo formal e uma implementação algorítmica capaz de gerar escalas de limpeza justas e viáveis. Para solucionar o problema, optou-se por modelá-lo como um Problema de Satisfação de Restrições (CSP - *Constraint Satisfaction Problem*). Essa abordagem é ideal para cenários onde é

mandatório encontrar um estado que satisfaça um conjunto rígido de regras lógicas antes de buscar qualquer otimização [Russell and Norvig 2010].

O trabalho está organizado da seguinte forma: a Seção 2 detalha a definição formal do problema e o modelo matemático adotado, incluindo parâmetros e restrições obrigatórias. A Seção 3 descreve a metodologia de solução via CSP e a implementação do algoritmo em Python. A Seção 4 discute os resultados obtidos e a complexidade da solução. Por fim, a Seção 5 apresenta as conclusões.

2. Definição do Problema e Modelo Matemático

O problema central consiste em alocar um conjunto finito de moradores para realizar a limpeza de diferentes cômodos ao longo de um horizonte de planejamento (conjunto de semanas). O objetivo é minimizar o impacto da alocação, ponderando a dificuldade intrínseca de cada tarefa com a indisponibilidade declarada de cada morador.

2.1. Conjuntos e Parâmetros

Para a modelagem matemática, definem-se os seguintes conjuntos:

- S: Conjunto de semanas do horizonte de planejamento ($s \in S$).
- M: Conjunto de moradores disponíveis ($m \in M$).
- C: Conjunto de cômodos da casa ($c \in C$).

Os parâmetros de entrada que alimentam o modelo são:

- D_c: Grau de dificuldade associado à limpeza do cômodo c.
- $U_{s,m}$: Grau de indisponibilidade do morador m na semana s.
- Q_c: Quantidade de moradores necessários para limpar o cômodo c (demanda da tarefa).

Para garantir a viabilidade da solução, assume-se a condição de existência onde a oferta total de mão-de-obra deve suprir a demanda dos cômodos, expressa por:

$$\sum_{c \in C} Q_c \leq |M| \tag{1}$$

Não somente, é necessário resguardar o modelo da situação em que pelo menos um cômodo não possa ser limpo em alguma semana, condicionado à restrição de que um morador não deve limpar o mesmo cômodo mais de uma vez durante todo o horizonte de planejamento (restrição três na seção 2.3)

$$Q_c \cdot |S| \leq |M| \quad \forall c \in C \tag{2}$$

2.2. Variáveis de Decisão e Função Objetivo

A decisão de alocação é modelada por uma variável binária $x_{s,m,c}$:

$$x_{s,m,c} = \begin{cases} 1 & \text{se o morador } m \text{ for alocado ao cômodo } c \text{ na semana } s \\ 0 & \text{caso contrário} \end{cases} \tag{3}$$

A função objetivo busca minimizar o "custo ponderado" total Z da escala:

$$\text{Minimizar } Z = \sum_{s \in S} \sum_{m \in M} \sum_{c \in C} (D_c \cdot U_{s,m} \cdot x_{s,m,c}) \tag{4}$$

Desta forma, o algoritmo penaliza alocações onde um morador com alta indisponibilidade recebe uma tarefa de alta dificuldade.

2.3. Restrições do Modelo

O modelo está sujeito a três restrições fundamentais que definem a viabilidade da escala:

1. Atendimento da Demanda: Em cada semana, a soma dos moradores alocados a um cômodo deve igualar a demanda Q_c:

$$\sum_{m \in M} x_{s,m,c} = Q_c, \quad \forall s \in S, \forall c \in C \tag{5}$$

2. Capacidade Individual: Cada morador pode assumir no máximo uma tarefa por semana. Isso permite folgas caso o número de moradores exceda a demanda:

$$\sum_{c \in C} x_{s,m,c} \leq 1, \quad \forall s \in S, \forall m \in M \tag{6}$$

3. Não Repetição de Tarefas: Para garantir a rotatividade e equidade a longo prazo, um morador não pode ser alocado ao mesmo cômodo mais de uma vez durante todo o horizonte S:

$$\sum_{s \in S} x_{s,m,c} \leq 1, \quad \forall m \in M, \forall c \in C \tag{7}$$

3. Implementação e Solução Proposta

Para resolver o modelo proposto, foi desenvolvida uma solução em linguagem Python, encapsulada na classe `RepublicaSolver`. A abordagem escolhida foi um algoritmo exato de busca em profundidade (*Backtracking*), capaz de explorar o espaço de estados e garantir a otimalidade global da solução, dado o tamanho restrito das instâncias típicas de uma república [Kumar 1992].

3.1. Estrutura de Dados e Verificações Iniciais

A classe recebe como entrada os dicionários de dificuldade, vagas e matrizes de indisponibilidade. Antes de iniciar a busca, o método `resolver` executa verificações matemáticas críticas (*Math Check*):

1. Verifica se o total de vagas semanais excede o número total de moradores.
2. Verifica se a demanda acumulada de um cômodo específico ao longo de todas as semanas excede o número total de moradores distintos disponíveis (devido à restrição de não repetição).

Caso qualquer verificação falhe, o algoritmo encerra prematuramente, evitando processamento inútil em instâncias inviáveis.

3.2. Algoritmo de Backtracking com Poda

O núcleo da solução é o método recursivo `_backtrack`. A estratégia de ramificação ocorre iterando sobre as semanas e, dentro destas, sobre os cômodos.

Para cada cômodo, o algoritmo identifica os candidatos válidos filtrando aqueles que:

- Já trabalham na semana atual (respeitando a restrição 2).
- Já realizaram a tarefa específica em semanas anteriores (respeitando a restrição 3).

Utilizando a biblioteca `itertools`, o algoritmo gera todas as combinações possíveis de candidatos para preencher as Q_c vagas do cômodo.

A escolha por um algoritmo de busca exata (Backtracking), modelado como um Problema de Satisfação de Restrições (CSP - *Constraint Satisfaction Problem*), justifica-se pela natureza do problema. Diferente de técnicas aproximativas (como Algoritmos Genéticos) que não garantem a solução ótima, ou de Aprendizado de Máquina que requerem grandes bases de dados para treinamento, a abordagem baseada em restrições e busca em árvore garante matematicamente o cumprimento de todas as regras rígidas (*Hard Constraints*) da república. Dado que o espaço de busca para o número de moradores (N=9) é tratável computacionalmente, a busca exata oferece a melhor relação custo-benefício entre tempo de processamento e qualidade da solução.

3.2.1. Cálculo de Custo e Heurística de Penalidade

Durante a avaliação de uma equipe candidata, o custo é calculado conforme a função objetivo apresentada na Seção 2. No entanto, a implementação introduz uma regra de negócio adicional (*Soft Constraint*) para qualidade de vida, não presente no modelo matemático estrito, mas essencial na prática:

> Se um morador trabalhou na semana imediatamente anterior (em qualquer cômodo), seu custo individual é multiplicado por um fator de penalidade `PENALIDADE_CONSECUTIVO` (valor fixado em 10.000).

Esta heurística força o algoritmo a priorizar, sempre que matematicamente possível, a alternância de semanas de folga para os moradores, elevando o custo de soluções que sobrecarregam sequencialmente o mesmo indivíduo.

3.2.2. Poda (Pruning)

Para lidar com a complexidade fatorial do problema, implementou-se uma poda baseada no custo. Se o `custo_atual` da alocação parcial já exceder o `melhor_custo` encontrado até o momento, o ramo é descartado imediatamente.

Para quantificar o ganho de desempenho proporcionado pela estratégia de *Branch and Bound*, foi realizado um experimento comparativo. O algoritmo foi executado em dois cenários distintos utilizando a mesma instância de entrada:

1. **Cenário A (Com Poda):** O algoritmo interrompe a exploração de um ramo assim que o custo parcial excede o melhor custo encontrado até o momento (linha 45 do código original).

2. **Cenário B (Sem Poda):** A verificação de otimalidade foi desabilitada, forçando o algoritmo a explorar todo o espaço de soluções viáveis, independentemente do custo acumulado.

Essa comparação visa demonstrar a importância da poda para a escalabilidade do sistema.

4. Resultados e Discussão

Os testes foram realizados utilizando diferentes instâncias representativas, variando o número de moradores, semanas e cômodos, com demandas de vagas heterogêneas e matrizes de indisponibilidade aleatórias. O ambiente de testes consistiu em um computador MacBook Air com processador Apple Silicon M1 e sistema operacional macOS.

4.1. Análise Experimental de Diferentes Entradas

Foram criados três cenários distintos, descritos na tabela abaixo. A utilização de uma instância de teste, não compromete a generalidade da solução. A adoção de múltiplas instâncias de entrada alteraria somente os parâmetros, sem exigir modificação no algoritmo.

Instância	Moradores	Semanas	Cômodos	Resultado	Observação
I	9	3	4	Viável	Alta penalização
II	9	4	3	Viável	Baixa penalização
III	9	5	3	Inviável	Violação estrutural

Na Instância I (9 moradores, 3 semanas e 4 cômodos), o algoritmo encontrou uma solução que o custo final foi elevado (40243). Esse valor não decorre necessariamente da complexidade combinatória mas sim da ativação da penalidade por trabalho consecutivo.

```
MELHOR ESCALA ENCONTRADA (Custo: 40243)

--- Semana1 ---
Sala      : Marcos
Cozinha   : Pedro
Cozinha   : Ana
Banheiro  : Lucas
Quintal   : Nicole

--- Semana2 ---
Sala      : Ana
Cozinha   : Joao
Cozinha   : Carla
Banheiro  : Sophia
Quintal   : Maria

--- Semana3 ---
Sala      : Carla
Cozinha   : Lucas
Cozinha   : Nicole
Banheiro  : Marcos
Quintal   : Pedro
```

Na Instância II (9 moradores, 4 semanas e 3 cômodos), o algoritmo encontrou uma solução viável, com custo de 166, concluindo que a distribuição de tarefas teve maior alternância, reduzindo a incidência da penalidade e o valor da função objetivo.

```
MELHOR ESCALA ENCONTRADA (Custo: 166)

--- Semana1 ---
Sala      : Pedro
Cozinha   : Joao
Cozinha   : Nicole
Banheiro  : Lucas

--- Semana2 ---
Sala      : Carla
Cozinha   : Ana
Cozinha   : Sophia
Banheiro  : Maria

--- Semana3 ---
Sala      : Nicole
Cozinha   : Lucas
Cozinha   : Marcos
Banheiro  : Pedro

--- Semana4 ---
Sala      : Joao
Cozinha   : Maria
Cozinha   : Carla
Banheiro  : Ana
```

Na Instância III (9 moradores, 5 semanas e 3 cômodos), o algoritmo encontrou uma solução inviável na etapa inicial de verificação matemática. A condição $Q_c \cdot |S| \leq |M|$ foi violada.

```
--- Verificação Inicial ---
Vagas por semana: 4
Moradores totais: 9
ERRO CRÍTICO: O cômodo 'Cozinha' precisa de 10 pessoas
diferentes ao todo, mas só existem 9.

Infeasible: Ajuste o número de moradores ou diminua as semanas.
```

4.2. Análise da Solução Gerada

A solução viável de custo 166 retornada pelo algoritmo demonstrou consistência matemática e aderência aos requisitos do problema. Observou-se o cumprimento integral das restrições rígidas (*Hard Constraints*):

- **Cobertura de Demanda:** Todos os cômodos receberam exatamente o número de moradores exigido (Q_c) em todas as semanas.

- **Capacidade Individual:** Nenhum morador foi alocado a mais de uma tarefa na mesma semana.
- **Rotatividade (Horizonte Finito):** Ao final do ciclo de 4 semanas, nenhum morador repetiu o mesmo cômodo, garantindo a diversidade de tarefas.

Além da viabilidade técnica, a qualidade da solução foi validada pela heurística de *Soft Constraint*. A introdução da penalidade (`PENALIDADE_CONSECUTIVO = 10000`) forçou o algoritmo a evitar, sempre que o espaço de soluções permitia, a alocação de um mesmo morador em semanas consecutivas. Na escala resultante, a alternância de folgas foi maximizada, elevando o custo apenas quando a combinação de indisponibilidades tornava o trabalho consecutivo inevitável.

4.3. Análise de Desempenho: Impacto da Poda

Como citado anteriormente, o algoritmo foi executado em dois cenários distintos para medir o tempo de convergência até a solução ótima:

1. **Com Poda (*Branch and Bound*):** O algoritmo utiliza a verificação de custo (linhas 45-46) para descartar ramos sub-ótimos.
2. **Sem Poda (Backtracking):** A verificação foi desabilitada, forçando a exploração completa da árvore de decisão.

Os resultados obtidos são apresentados na Tabela 1.

Tabela 1. Comparativo de Tempo de Execução

Métrica	Com Poda	Sem Poda
Tempo de Execução	**3,53 segundos**	$>$ **5 minutos (Interrompido)**
Status da Solução	Ótima Encontrada	Não Convergiu
Estados Explorados	Reduzido (Eficiente)	Explosão Combinatória

Discussão dos Resultados: Os dados evidenciam a criticidade da estratégia de otimização. Mesmo utilizando um processador de alta performance (Apple M1), a versão de força bruta mostrou-se inviável, ultrapassando 5 minutos de processamento contínuo sem concluir a varredura do espaço de estados. Em contrapartida, a versão com poda encontrou a melhor escala possível em apenas 3,53 segundos.

Esse comportamento confirma a natureza exponencial do problema (NP-Difícil). A poda, portanto, não atua apenas como um acelerador, mas como um requisito funcional obrigatório para a viabilidade da solução em tempo hábil.

4.4. Complexidade Teórica

A complexidade de pior caso permanece exponencial, dada a natureza NP-Difícil do problema de alocação. Contudo, a combinação das restrições de domínio (que reduzem os candidatos a cada nível da árvore) com a poda por custo torna a solução tratável para o escopo do projeto.

5. Conclusão

Este trabalho apresentou uma modelagem matemática robusta para o problema de escalas em repúblicas, traduzida em uma ferramenta computacional funcional. A implementação

em Python demonstrou que é possível conciliar restrições rígidas de capacidade com objetivos de minimização de desconforto (indisponibilidade e dificuldade).

A principal contribuição da implementação foi a adição da lógica de penalidade para trabalho consecutivo, refinando o modelo matemático puramente linear para um cenário mais humano e sustentável. Trabalhos futuros podem explorar meta-heurísticas (como Algoritmos Genéticos) para resolver instâncias maiores onde o método exato se torna computacionalmente inviável.

Disponibilidade do Código: A implementação completa, incluindo o código fonte, está disponível publicamente no repositório GitHub: `https://github.com/luizelemesmo/otimizacao-escala-republica`.

Referências

Burke, E. K., De Causmaecker, P., Vanden Berghe, G., and Van Landeghem, H. (2004). State of the art of nurse rostering. *Journal of scheduling*, 7(6):441–499.

Kumar, V. (1992). Algorithms for constraint-satisfaction problems: a survey. *AI magazine*, 13(1):32–32.

Russell, S. J. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition.

ANEXO I — Uso de Ferramentas de Inteligência Artificial

Este anexo descreve o uso pontual de ferramentas de Inteligência Artificial generativa como apoio ao desenvolvimento do trabalho, conforme solicitado na especificação da disciplina. Ressalta-se que todo o conteúdo final, decisões metodológicas, modelagem, implementação e análise crítica são de responsabilidade dos autores.

1. Tipo de solicitação: Esclarecimento conceitual
Plataforma utilizada: ChatGPT
Data aproximada: Janeiro de 2026
Descrição: Consulta sobre conceitos teóricos relacionados a Problemas de Satisfação de Restrições (CSP), incluindo definições de variáveis, domínios e restrições, com o objetivo de validar a adequação da técnica de Inteligência Artificial adotada.

2. Tipo de solicitação: Esclarecimento conceitual e modelagem de heurística
Plataforma utilizada: Gemini
Data aproximada: Janeiro de 2026
Descrição: Consulta sobre estratégias de penalização por trabalho consecutivo (*penalidade por fadiga*) em problemas de alocação, incluindo a ideia de aplicação de fatores de custo elevados para moradores alocados na semana anterior, bem como discussão conceitual sobre como modelar esse mecanismo em um algoritmo de busca com backtracking e em linguagens de modelagem declarativa. A ferramenta foi utilizada apenas como apoio conceitual, sem incorporação direta de código gerado.

3. Tipo de solicitação: Organização de conteúdo
Plataforma utilizada: Gemini
Data aproximada: Janeiro de 2026
Descrição: Solicitação de sugestões para organização estrutural de um relatório técnico no formato SBC, sem geração direta do texto final.

4. Tipo de solicitação: Busca e formatação de referências bibliográficas
Plataforma utilizada: ChatGPT
Data aproximada: Fevereiro de 2026
Descrição: Apoio na identificação de referências relevantes na literatura e orientações para formatação das referências bibliográficas em conformidade com as diretrizes acadêmicas solicitadas, preservando o template SBC.

5. Tipo de solicitação: Suporte técnico em LaTeX
Plataforma utilizada: Gemini
Data aproximada: Fevereiro de 2026
Descrição: Esclarecimento de dúvidas pontuais relacionadas ao uso de comandos LaTeX e adaptação do conteúdo ao template SBC.

6. Tipo de solicitação: Revisão de texto acadêmico
Plataforma utilizada: ChatGPT
Data aproximada: Fevereiro de 2026
Descrição: Utilização da ferramenta para revisão gramatical e estilística de trechos do relatório, com foco em clareza, coesão textual e adequação ao português acadêmico.

7. **Tipo de solicitação:** Verificação de coerência

Plataforma utilizada: ChatGPT

Data aproximada: Fevereiro de 2026

Descrição: Solicitação de verificação de coerência entre o texto do relatório e a especificação da atividade, com foco na aderência aos critérios de avaliação.