Para replicar o experimento de Ochelska-Mierzejewska et al. (2021), nossa equipe realizou algumas adaptações importantes. Essas mudanças foram necessárias para garantir que o código funcionasse sem erros e explicam por que nossos resultados foram diferentes dos do artigo em alguns pontos.

1. Mudança na Representação do Cromossomo (Técnica de Divisão) O artigo original tenta incluir os retornos ao depósito (zeros) dentro da sequência genética. O próprio autor admite que isso cria problemas, gerando rotas impossíveis (ex: caminhão com carga acima do limite).

- **Nossa Solução:** Usamos uma lista única com a ordem dos clientes (chamada de *Giant Tour*) e aplicamos uma função automática de **divisão de capacidade**. Essa função percorre a lista e, sempre que o caminhão enche, cria automaticamente um retorno ao depósito.
- **Vantagem:** Isso garantiu que 100% das soluções geradas fossem válidas. Isso ajudou muito o operador **ERX**, que funcionou bem melhor no nosso teste do que no artigo, pois não precisou lidar com rotas "quebradas".

2. Simplificação da Mutação (Custo-Benefício) O artigo usa dois tipos de mutação. Nós optamos por implementar apenas a **Troca Simples (Swap)**.

- **Impacto (O Trade-off):** Ao simplificar a mutação, ganhamos facilidade na implementação, mas perdemos "diversidade". Isso significa que nosso algoritmo às vezes ficava preso em uma solução razoável e não conseguia achar a ótima. Isso ficou claro no método **Rank** nas instâncias grandes: como a pressão para escolher os melhores era alta e a nossa mutação era "tímida", o algoritmo convergiu rápido demais para uma solução que não era a ideal.

3. Instâncias Utilizadas Como nem todos os mapas usados pelo autor estão disponíveis publicamente, usamos um conjunto de mapas equivalentes das bibliotecas padrão (CVRPLIB).

- **Validação:** Mesmo mudando os mapas, o comportamento dos operadores se manteve (ex: o ERX continuou sendo muito bom), o que prova que nosso código está consistente.

4. Implementação Própria Não usamos bibliotecas prontas de IA. Programamos toda a lógica dos cruzamentos (PMX, OX, AEX, ERX) do zero em Python.

- **Vantagem:** Isso nos deu controle total para entender como cada operador funciona passo a passo, garantindo uma comparação justa.

Comparação **pequenas** instâncias AcMsF

Média dos resultados agrupados por métodos de seleção

	Artigo	Nosso resultado
elitismo	16,85%	24,38%
rank	18,97%	35,73%
roleta	16,87%	15,34%
torneio	17,14%	16,53%

Nesta fase inicial, os resultados validam a correção da nossa implementação do algoritmo. Os métodos **Roleta** (15,34%) e **Torneio** (16,53%) apresentaram desvios muito próximos ou até ligeiramente inferiores aos relatados no artigo original. Isso indica que a lógica base de pressão seletiva está funcionando conforme o esperado. O método **Rank** já começa a demonstrar uma leve tendência de desvio superior (35,73%), sinalizando uma sensibilidade maior à nossa simplificação do operador de mutação.

Média dos resultados agrupados por crossover

	Artigo	Nosso resultado
alternating_edges	14,69%	25,69%
cycle	20,41%	26,09%
edge_recombination	12,88%	15,46%
order	20,26%	22,72%
partially_mapped	19,04%	24,78%

O operador **Edge Recombination (ERX)** confirmou ser o mais eficiente para este problema, apresentando o menor desvio tanto no artigo original (12,88%) quanto na nossa replicação (15,46%). Isso demonstra que, em instâncias menores, preservar as conexões (arestas) entre as cidades é a estratégia mais eficaz para encontrar rotas curtas rapidamente. Os resultados dos outros operadores (AEX, CX, OX, PMX) mantiveram a mesma ordem de eficiência observada na literatura.

Comparação **médias** instâncias AcMsF

Média dos resultados agrupados por métodos de seleção

	Artigo	Nosso resultado
elitismo	63,87%	37,48%
rank	58,05%	78,89%
roleta	66,18%	31,63%
torneio	58,05%	29,73%

Neste cenário, observamos o impacto positivo da nossa adaptação na representação do cromossomo (técnica de *Giant Tour* com divisão automática). Enquanto o artigo relata um aumento drástico no erro para **Roleta** e **Torneio** (acima de 58-66%), nossos resultados mantiveram-se na faixa de 29-31%. Isso sugere que a abordagem do autor sofreu com a geração de indivíduos inválidos, desperdiçando processamento, enquanto nossa abordagem garantiu que 100% da população fosse aproveitável, acelerando a busca

Média dos resultados agrupados por crossover

	Artigo	Nosso resultado
alternating_edges	74,92%	51,30%
cycle	49,72%	50,61%
edge_recombination	82,42%	32,82%

order	50,68%	43,70%
partially_mapped	49,95%	43,73%

O contraste no desempenho do **ERX** é notável: obtivemos 32,82% de desvio contra 82,42% do artigo. O autor sugere que operadores baseados em arestas demoram para convergir, mas nossos dados refutam isso para nossa implementação. Ao garantir a validade das rotas via *Split*, o ERX mostrou-se extremamente robusto e capaz de superar a implementação original, provando que a preservação de arestas continua eficiente em média escala quando bem implementada.

Comparação **grandes** instâncias AcMsF

Média dos resultados agrupados por métodos de seleção

	Artigo	Nosso resultado
elitismo	111,74%	58,19%
rank	104,95%	142,55%
roleta	109,40%	65,68%
torneio	103,31%	54,95%

Aqui fica evidente o *trade-off* (custo-benefício) da nossa simplificação na mutação. O método **Rank** apresentou o pior desempenho (142,55% de desvio), superando muito o erro do artigo. Como o Rank exerce alta pressão para escolher apenas os melhores, e nossa mutação (apenas *Swap*) é pouco agressiva para gerar diversidade, o algoritmo convergiu prematuramente para soluções ruins (ótimos locais). Por outro lado, **Torneio** e **Elitismo** mantiveram-se muito mais estáveis que no estudo original, beneficiando-se da representação robusta.

Média dos resultados agrupados por crossover

	Artigo	Nosso resultado
alternating_edges	142,79%	95,59%
cycle	83,13%	85,42%
edge_recombination	157%	68,94%
order	81,43%	74,13%
partially_mapped	71,40%	77,65%

Este é o resultado mais divergente e positivo da replicação. Enquanto o artigo conclui que o **ERX** tem o pior desempenho em grandes instâncias (157% de erro), nossa replicação obteve o **melhor desempenho geral** com ele (68,94%). Isso indica que a falha relatada no artigo original provavelmente se deve à dificuldade de manter a integridade das arestas na representação com 'zeros'. Nossa implementação demonstra que o ERX é, na verdade, altamente escalável e eficiente para grandes instâncias do VRP quando a viabilidade da solução é garantida.