I - Roteiro para Apresentação "PyCommend VNS" - ICVNS 2025

Slide 1: Título

{ Olá a todos! Apresentarei o PyCommend VNS, um framework de recomendação multi-objetivo para bibliotecas Python baseado em Variable Neighborhood Search. Nossa abordagem resolve um problema crítico: a dificuldade de encontrar as melhores bibliotecas no fragmentado ecossistema Python. }

Slide 2: The Scale of Developer Time Waste

{ Os desenvolvedores perdem 41-42 do tempo em manutenção e correção de bugs em vez de criar novas funcionalidades. Em média, são 17,3 horas semanais gastas em atividades que frequentemente duplicam soluções existentes. Curiosidade: 61 dos desenvolvedores gastam mais de 30 minutos diários apenas procurando soluções para problemas já resolvidos. }

Slide 3: Financial Impact of "Reinventing the Wheel"

{ O impacto financeiro da "reinvenção da roda" é astronômico: $300 bilhões anuais em produtividade perdida mundialmente. A dívida técnica acumulada só nos EUA chegou a $1,52 trilhão em 2022. Curiosidade: organizações relatam redução de 50-70 nos prazos de projetos quando aproveitam efetivamente bibliotecas existentes. }

Slide 4: Why Developers Miss Existing Libraries

{ As barreiras de conhecimento são o primeiro obstáculo: desenvolvedores têm consciência limitada das soluções existentes e sofrem com sobrecarga de informações. A síndrome do "Not Invented Here" é um fator psicológico importante, onde há orgulho em criar do zero. Curiosidade: as políticas organizacionais restritivas e a falta de incentivos para reutilização de código são obstáculos tão importantes quanto os desafios técnicos. }

Slide 5: Python's Fragmented Ecosystem

{ O PyPI hospeda mais de 628.000 pacotes com sobreposição funcional substancial, criando desafios de descoberta. Um exemplo claro: as bibliotecas HTTPX e Requests têm 99 de sobreposição de código, e existem pelo menos 6 bibliotecas principais para a mesma funcionalidade básica. Curiosidade: o PyPI cresceu exponencialmente, mas seus recursos de busca permaneceram praticamente inalterados na última década. }

Slide 6: Benefits of Effective Library Discovery

{ A descoberta eficaz de bibliotecas pode reduzir trabalhos de ciência de dados de 3-4 semanas para apenas 2-3 dias. Em análises financeiras, a redução de tempo chega a 70. Curiosidade: o LibFinder, um sistema de recomendação anterior, alcançou 92 de precisão, demonstrando o potencial desse tipo de abordagem. }

Slide 7: Data Collection

{ Nossa coleta de dados incluiu os 10.000 principais pacotes PyPI e 2.000 repositórios relacionados à ciência da computação. Extraímos dependências de 24.000 projetos GitHub via requirements.txt e pyproject.toml. Curiosidade: construímos duas matrizes fundamentais: co-ocorrência de pacotes (F1) e similaridades semânticas ponderadas por TF-IDF (F2). }

Slide 8: Problem Formulation: Multi-Objective Optimization

{ Formulamos o problema como otimização multi-objetivo: recomendar um subconjunto S de bibliotecas Python que otimize três critérios competitivos. Maximizamos o Uso Vinculado (LU) e a Similaridade Semântica Ponderada (SS), enquanto minimizamos o Tamanho do Conjunto (RSS). Curiosidade: essa abordagem garante um conjunto de soluções não-dominadas, oferecendo flexibilidade para diferentes necessidades de usuários. }

Slide 9: Multi-Objective VNS Framework

{ Nosso framework VNS Multi-Objetivo utiliza representação binária e oferece vantagens significativas: exploração sistemática de vizinhanças e eficácia para espaços combinatórios. O algoritmo central inicia com uma solução que passa por busca local de Pareto, com mudanças sistemáticas de vizinhança para intensificação. Curiosidade: o "shaking" (perturbação) é crucial para escapar de ótimos locais, proporcionando diversificação. }

Slide 10: Neighborhood Structures for Library Selection

{ Desenvolvemos três estruturas de vizinhança adaptadas para seleção de subconjuntos: N_1 (Adição) adiciona uma biblioteca, N_2 (Remoção) remove uma biblioteca, e N_3 (Troca) substitui uma biblioteca. A operação de troca é particularmente poderosa pois mantém o tamanho da solução enquanto melhora sua qualidade. Curiosidade: a escolha dessas estruturas foi baseada em análises empíricas de espaços de soluções em problemas similares de seleção de subconjuntos. }

Slide 11: Pareto Archive & Performance Metrics

{ O Arquivo de Elite armazena todas as soluções não-dominadas, garantindo um hipervolume não-decrescente. Utilizamos três indicadores-chave de desempenho:

Hipervolume (HV) para convergência e diversidade, Spread (Δ) para uniformidade de distribuição, e ε-indicator para convergência entre iterações. Curiosidade: o hipervolume é uma métrica particularmente robusta que não requer conhecimento da fronteira de Pareto verdadeira. }

Slide 12: Experimental Setup

{ Nossos benchmarks incluíram 10 bibliotecas contextuais como scikit-learn e XGBoost. Comparamos NSGA-II (população=200, gerações=500) com MOVNS (3 vizinhanças, tamanho de arquivo=100). Curiosidade: limitamos o orçamento computacional a 2 minutos para simular cenários de recomendação em tempo real, um requisito prático importante. }

Slide 13: Convergence Results

{ O MOVNS superou o NSGA-II com 5,1 mais hipervolume e convergência inicial 15,2 mais rápida. O melhor ε-indicator também indica superioridade qualitativa das soluções MOVNS. Curiosidade: a vantagem do MOVNS é ainda mais pronunciada nos primeiros segundos de execução, o que é crucial para aplicações interativas de recomendação. }

Slide 14: PyCommend VNS Results: Context Libraries

{ Testamos com bibliotecas de diversos domínios: FastAPI (web, 65k stars), Prophet (séries temporais, 16.1k stars), scikit-learn (ML, 55k stars), entre outras. As bibliotecas de entrada variam de frameworks estabelecidos a ferramentas especializadas. Curiosidade: incluímos intencionalmente bibliotecas com diferentes níveis de popularidade para testar a robustez do sistema em vários cenários. }

Slide 15: PyCommend VNS Results: Recommendations

{ As recomendações Pareto-ótimas formam conjuntos coerentes: para FastAPI, o sistema recomenda pydantic e uvicorn (essenciais) mais typer (CLI). Para scikit-learn, recomenda Optuna, XGBoost e joblib, ferramentas complementares para ML. Curiosidade: cada linha da tabela representa uma solução não-dominada diferente na fronteira de Pareto, com diferentes tradeoffs entre os objetivos. }

Slide 16: PyCommend VNS Results: Validation Approaches

{ Validamos com métricas baseadas em precisão: Precision@K, Recall@K e F1-Score para avaliar a relevância das recomendações. Complementamos com métricas comportamentais: Cobertura, Diversidade e Novidade Útil. Curiosidade: a novidade útil é particularmente importante, pois queremos recomendar bibliotecas desconhecidas mas relevantes, não apenas as mais populares. }

Slide 17: PyCommend VNS Results: Ecosystem Patterns

{ Descobrimos padrões claros de co-ocorrência: em ML, scikit-learn + XGBoost + Optuna; em deploy de modelos, FastAPI + Pydantic + uvicorn. A compatibilidade de API e integração de workflow impulsionam a adoção conjunta. Curiosidade: o ecossistema Python carece de padrões universais e sistemas de classificação hierárquica, o que torna frameworks como o nosso ainda mais necessários. }

Slide 18: Conclusions & Future Work

{ O framework MOVNS recomenda eficientemente conjuntos diversos e inovadores de bibliotecas com baixo custo computacional. A abordagem multi-objetivo equilibra padrões de uso, similaridade semântica e concisão. Para trabalhos futuros, planejamos incluir considerações de segurança, compatibilidade de versões e análise de licenças. Curiosidade: estamos expandindo para C++, outro ecossistema com fragmentação significativa. }

Slide 19: Acknowledgments

{ Gostaria de agradecer ao Comitê Organizador do ICVNS-2025 e ao Professor Daniel Aloise por seu apoio. Nosso repositório e conjuntos de dados estarão disponíveis publicamente após a publicação. Para contato ou colaborações, meu email e LinkedIn estão disponíveis neste slide. Curiosidade: este trabalho foi desenvolvido entre UERJ e UFF, combinando expertise em metaheurísticas e análise de ecossistemas de software. }

Roteiro para Apresentação "PyCommend VNS" - ICVNS 2025

Slide 1: Título

{ Olá a todos! Apresentarei o PyCommend VNS, um framework de recomendação multi-objetivo para bibliotecas Python baseado em Variable Neighborhood Search. Nossa abordagem resolve um problema crítico: a dificuldade de encontrar as melhores bibliotecas no fragmentado ecossistema Python. }

Slide 2: The Scale of Developer Time Waste

{ Os desenvolvedores perdem 41-42 do tempo em manutenção e correção de bugs em vez de criar novas funcionalidades. Em média, são 17,3 horas semanais gastas em atividades

que frequentemente duplicam soluções existentes. Curiosidade: 61 dos desenvolvedores gastam mais de 30 minutos diários apenas procurando soluções para problemas já resolvidos. }

Slide 3: Financial Impact of "Reinventing the Wheel"

{ O impacto financeiro da "reinvenção da roda" é astronômico: $300 bilhões anuais em produtividade perdida mundialmente. A dívida técnica acumulada só nos EUA chegou a $1,52 trilhão em 2022. Curiosidade: organizações relatam redução de 50-70 nos prazos de projetos quando aproveitam efetivamente bibliotecas existentes. }

Slide 4: Why Developers Miss Existing Libraries

{ As barreiras de conhecimento são o primeiro obstáculo: desenvolvedores têm consciência limitada das soluções existentes e sofrem com sobrecarga de informações. A síndrome do "Not Invented Here" é um fator psicológico importante, onde há orgulho em criar do zero. Curiosidade: as políticas organizacionais restritivas e a falta de incentivos para reutilização de código são obstáculos tão importantes quanto os desafios técnicos. }

Slide 5: Python's Fragmented Ecosystem

{ O PyPI hospeda mais de 628.000 pacotes com sobreposição funcional substancial, criando desafios de descoberta. Um exemplo claro: as bibliotecas HTTPX e Requests têm 99 de sobreposição de código, e existem pelo menos 6 bibliotecas principais para a mesma funcionalidade básica. Curiosidade: o PyPI cresceu exponencialmente, mas seus recursos de busca permaneceram praticamente inalterados na última década. }

Slide 6: Benefits of Effective Library Discovery

{ A descoberta eficaz de bibliotecas pode reduzir trabalhos de ciência de dados de 3-4 semanas para apenas 2-3 dias. Em análises financeiras, a redução de tempo chega a 70. Curiosidade: o LibFinder, um sistema de recomendação anterior, alcançou 92 de precisão, demonstrando o potencial desse tipo de abordagem. }

Slide 7: Data Collection

{ Nossa coleta de dados incluiu os 10.000 principais pacotes PyPI e 2.000 repositórios relacionados à ciência da computação. Extraímos dependências de 24.000 projetos GitHub via requirements.txt e pyproject.toml. Curiosidade: construímos duas matrizes fundamentais: co-ocorrência de pacotes (F1) e similaridades semânticas ponderadas por TF-IDF (F2). }

Slide 8: Problem Formulation: Multi-Objective Optimization

{ Formulamos o problema como otimização multi-objetivo: recomendar um subconjunto S de bibliotecas Python que otimize três critérios competitivos. Maximizamos o Uso Vinculado (LU) e a Similaridade Semântica Ponderada (SS), enquanto minimizamos o Tamanho do Conjunto (RSS). Curiosidade: essa abordagem garante um conjunto de soluções não-dominadas, oferecendo flexibilidade para diferentes necessidades de usuários. }

Slide 9: Multi-Objective VNS Framework

{ Nosso framework VNS Multi-Objetivo utiliza representação binária e oferece vantagens significativas: exploração sistemática de vizinhanças e eficácia para espaços combinatórios. O algoritmo central inicia com uma solução que passa por busca local de Pareto, com mudanças sistemáticas de vizinhança para intensificação. Curiosidade: o "shaking" (perturbação) é crucial para escapar de ótimos locais, proporcionando diversificação. }

Slide 10: Neighborhood Structures for Library Selection

{ Desenvolvemos três estruturas de vizinhança adaptadas para seleção de subconjuntos: N_1 (Adição) adiciona uma biblioteca, N_2 (Remoção) remove uma biblioteca, e N_3 (Troca) substitui uma biblioteca. A operação de troca é particularmente poderosa pois mantém o tamanho da solução enquanto melhora sua qualidade. Curiosidade: a escolha dessas estruturas foi baseada em análises empíricas de espaços de soluções em problemas similares de seleção de subconjuntos. }

Slide 11: Pareto Archive & Performance Metrics

{ O Arquivo de Elite armazena todas as soluções não-dominadas, garantindo um hipervolume não-decrescente. Utilizamos três indicadores-chave de desempenho: Hipervolume (HV) para convergência e diversidade, Spread (Δ) para uniformidade de distribuição, e ε-indicator para convergência entre iterações. Curiosidade: o hipervolume é uma métrica particularmente robusta que não requer conhecimento da fronteira de Pareto verdadeira. }

Slide 12: Experimental Setup

{ Nossos benchmarks incluíram 10 bibliotecas contextuais como scikit-learn e XGBoost. Comparamos NSGA-II (população=200, gerações=500) com MOVNS (3 vizinhanças, tamanho de arquivo=100). Curiosidade: limitamos o orçamento computacional a 2 minutos para simular cenários de recomendação em tempo real, um requisito prático importante. }

Slide 13: Convergence Results

{ O MOVNS superou o NSGA-II com 5,1 mais hipervolume e convergência inicial 15,2 mais rápida. O melhor ε-indicator também indica superioridade qualitativa das soluções MOVNS. Curiosidade: a vantagem do MOVNS é ainda mais pronunciada nos primeiros segundos de execução, o que é crucial para aplicações interativas de recomendação. }

Slide 14: PyCommend VNS Results: Context Libraries

{ Testamos com bibliotecas de diversos domínios: FastAPI (web, 65k stars), Prophet (séries temporais, 16.1k stars), scikit-learn (ML, 55k stars), entre outras. As bibliotecas de entrada variam de frameworks estabelecidos a ferramentas especializadas. Curiosidade: incluímos intencionalmente bibliotecas com diferentes níveis de popularidade para testar a robustez do sistema em vários cenários. }

Slide 15: PyCommend VNS Results: Recommendations

{ As recomendações Pareto-ótimas formam conjuntos coerentes: para FastAPI, o sistema recomenda pydantic e uvicorn (essenciais) mais typer (CLI). Para scikit-learn, recomenda Optuna, XGBoost e joblib, ferramentas complementares para ML. Curiosidade: cada linha da tabela representa uma solução não-dominada diferente na fronteira de Pareto, com diferentes tradeoffs entre os objetivos. }

Slide 16: PyCommend VNS Results: Validation Approaches

{ Validamos com métricas baseadas em precisão: Precision@K, Recall@K e F1-Score para avaliar a relevância das recomendações. Complementamos com métricas comportamentais: Cobertura, Diversidade e Novidade Útil. Curiosidade: a novidade útil é particularmente importante, pois queremos recomendar bibliotecas desconhecidas mas relevantes, não apenas as mais populares. }

Slide 17: PyCommend VNS Results: Ecosystem Patterns

{ Descobrimos padrões claros de co-ocorrência: em ML, scikit-learn + XGBoost + Optuna; em deploy de modelos, FastAPI + Pydantic + uvicorn. A compatibilidade de API e integração de workflow impulsionam a adoção conjunta. Curiosidade: o ecossistema Python carece de padrões universais e sistemas de classificação hierárquica, o que torna frameworks como o nosso ainda mais necessários. }

Slide 18: Conclusions & Future Work

{ O framework MOVNS recomenda eficientemente conjuntos diversos e inovadores de bibliotecas com baixo custo computacional. A abordagem multi-objetivo equilibra padrões de uso, similaridade semântica e concisão. Para trabalhos futuros, planejamos incluir considerações de segurança, compatibilidade de versões e análise de licenças. Curiosidade: estamos expandindo para C++, outro ecossistema com fragmentação significativa. }

Slide 19: Acknowledgments

{ Gostaria de agradecer ao Comitê Organizador do ICVNS-2025 e ao Professor Daniel Aloise por seu apoio. Nosso repositório e conjuntos de dados estarão disponíveis publicamente após a publicação. Para contato ou colaborações, meu email e LinkedIn estão disponíveis neste slide. Curiosidade: este trabalho foi desenvolvido entre UERJ e UFF, combinando expertise em metaheurísticas e análise de ecossistemas de software. }

Slide 1 \begin{frame}{Project Motivation} \begin{columns} \column{0.45\textwidth} \textbf{The Tourist's Dilemma} \begin{itemize} \item Complex trade-offs in weekend planning \item Quantity vs. quality of attractions \item Cost

SLIDE 1: SPEAKER NOTES: - Explain the typical problems tourists face - limited time, many attractions - Highlight how this is a perfect multi-objective problem (can't optimize all objectives simultaneously) falar sobre os 4 objetivos: { - Maximizing the number of attractions visite

II - Roteiro para Apresentação Turismo Montreal com MOVNS - ICVNS 2025

Slide 1: Project Motivation

Bom dia, meu nome é Augusto Mendonça, sou pesquisador da Universidade Federal Fluminense e estou apresentando o trabalho do amigo Filipe Sousa que não pode estar aqui.

Você já se encontrou em uma situação onde você deseja fazer uma viagem, que hoje é vendida como uma experiência, e como maximizar essa experiência é um complexo problema multi objetivo.

{ "Olá a todos. Hoje vou falar sobre um problema que todos nós já enfrentamos durante viagens: como aproveitar ao máximo um destino com tempo limitado, custo, qualidade dos locais, e Em Montreal, um turista de fim de semana enfrenta escolhas difíceis entre quantidade e qualidade de atrações, além de equilibrar custo e eficiência. Nossa pesquisa usa MOVNS para otimizar roteiros de 2 dias equilibrando 4 objetivos concorrentes: maximizar quantidade e qualidade de atrações, enquanto minimizamos tempo e custo. É um problema perfeito para demonstrar a eficácia do MOVNS em situações reais." }

Slide 2: Data Collection and Tools

{ "Para construir nosso sistema, coletamos dados reais de Montreal. Usamos o OpenStreetMap para localização de 50 atrações e 100 hotéis, e os horários oficiais do transporte público através do formato GTFS. A peça central técnica foi o OpenTripPlanner, que calculou mais de 65.000 rotas considerando 4 modos de transporte: caminhada, ônibus, metrô e carro. O mais interessante aqui é que trabalhamos com restrições de tempo reais - os horários de ônibus e metrô são exatamente os mesmos que um turista enfrentaria. Isso tornou o problema muito mais desafiador, mas também mais relevante na prática." }

Slide 3: Montreal Tourism Analysis

{ "Ao analisar as atrações de Montreal, encontramos padrões interessantes. As cinco atrações mais bem avaliadas incluem a Basílica de Notre-Dame e o Parque Mount Royal, com notas acima de 4,7/5. Observamos que a duração das visitas varia muito - de 30 a 180 minutos - o que torna o planejamento mais complexo. Um dado curioso é que 40 das atrações estão concentradas na região da Velha Montreal, enquanto outras estão bastante dispersas geograficamente. Essa distribuição cria um desafio especial para otimização de rotas, já que visitar atrações distantes implica em trade-offs significativos de tempo." }

Slide 4: Hotel and Transportation Analysis

{ "A escolha do hotel tem um impacto enorme nos roteiros. Os preços variam drasticamente - de $25 a $529 - e descobrimos que a localização estratégica pode reduzir o tempo de trânsito em até 40. Quanto aos meios de transporte, o carro domina 45 dos trechos por ser mais rápido, enquanto ônibus e metrô representam 45 combinados. Um insight interessante é que a maioria das rotas ótimas utiliza múltiplos modos de transporte. É como montar um quebra-cabeça onde a localização do hotel define todas as outras peças. Esta complexidade combinatória é exatamente onde o VNS se destaca em relação a outras abordagens." }

Slide 5: MOVNS Applications in Related Domains

{ "O MOVNS tem se destacado em diversos domínios similares ao nosso problema. Na área de manufatura, Li e Tian alcançaram 27 de melhoria em hypervolume em problemas de sequenciamento. Em logística, Duarte conseguiu 18 de melhoria na utilização de frotas. As aplicações em engenharia também mostram resultados impressionantes, como a otimização de confiabilidade de sistemas superando algoritmos genéticos em 96 dos casos. O caso mais relacionado ao nosso é o planejamento de tours de montanha de Schilde, que também balanceava objetivos conflitantes. Estes resultados nos motivaram a aplicar MOVNS ao turismo urbano, que combina elementos de todos estes problemas." }

Slide 6: Why MOVNS for Tourism Planning?

{ "Por que MOVNS é perfeito para planejamento turístico? Primeiro, o agrupamento espacial das atrações se encaixa naturalmente no conceito de vizinhança. Segundo, a otimização de sequência é crítica para eficiência - exatamente onde o VNS brilha. Terceiro, as transições modais exigem mudanças coordenadas que algoritmos evolutivos têm dificuldade em realizar. A grande vantagem do VNS é que melhorias locais são muito mais efetivas que mutações aleatórias neste tipo de problema. Também conseguimos criar operadores específicos para cada tipo de restrição. Como resultado, exploramos muito melhor os padrões geográficos e temporais que são essenciais para o planejamento de turismo." }

Slide 7: Problem Formulation

{ "Formalizamos o problema considerando 50 atrações com avaliações, duração e taxas, 100 hotéis com localização, preço e avaliação, e 4 modos de transporte com diferentes tempos e custos. Nosso horizonte de planejamento são 2 dias (sábado e domingo) com máximo de 12 horas por dia. Trabalhamos com 4 funções objetivo: F1 maximiza o número de atrações visitadas, F2 maximiza a pontuação total de qualidade baseada nas avaliações, F3 minimiza o tempo total gasto (incluindo visitas e deslocamentos), e F4 minimiza o custo total da viagem. O desafio é que estes objetivos competem entre si - melhorar um geralmente piora outro, criando um espaço de soluções complexo." }

Slide 8: NSGA-II: Overview

{ "Como base de comparação, implementamos o NSGA-II, que é o algoritmo evolutivo multi-objetivo mais utilizado. Desenvolvido por Deb em 2002, ele utiliza ordenação rápida de não-dominância e distância de crowding para manter diversidade. Nossa implementação usou população de 200 indivíduos, 100 gerações, probabilidade de crossover de 0,9 e mutação de 0,2. Desenvolvemos em C++ usando o framework OptFrame. A representação da solução inclui o índice do hotel, sequências de atrações para cada dia e modos de transporte entre locais. Esta implementação forte nos dá um bom benchmark para avaliar nossa abordagem MOVNS." }

Slide 9: NSGA-II: Operators & Constraints

{ "Para o NSGA-II, criamos operadores genéticos específicos para o problema. A inicialização usa construção aleatória guiada, o crossover é ordenado de dois pontos, e implementamos vários operadores de mutação - mudança de hotel, adição/remoção/troca de atrações, e alterações de dia ou modo de transporte. O maior desafio foi o tratamento de restrições: temos janelas de tempo (8h-20h), horários de funcionamento das atrações, necessidade de retorno ao hotel, e conexões de transporte viáveis. Desenvolvemos mecanismos de reparo para tratar violações, mas isso adiciona complexidade computacional. Estas restrições são justamente onde o MOVNS mostrou vantagem significativa." }

Slide 10: MOVNS: Objective Functions

{ "Nos resultados obtidos, observamos amplas variações nas funções objetivo: de 4 a 10 atrações visitadas, pontuações de qualidade entre 25,7 e 53,3, tempo total variando de 738 a 1359 minutos, e custo entre $105,9 e $694,1. Estes intervalos demonstram os trade-offs fundamentais do problema: mais atrações significam maior qualidade mas também mais tempo; hotéis baratos aumentam o tempo de trânsito; transporte por carro é mais rápido mas mais caro; e atrações premium têm maior qualidade mas custam mais. O equilíbrio ótimo depende das preferências do viajante - alguns preferirão economizar, outros priorizar experiências de qualidade." }

Slide 11: MOVNS: Neighborhood Structures

{ "O coração do nosso método são as cinco estruturas de vizinhança que desenvolvemos. N_1 realiza troca interna dentro do mesmo dia, melhorando a sequência sem alterar o conjunto. N_2 move atrações entre os dias, equilibrando a distribuição. N_3 insere ou remove atrações, ajustando a quantidade. N_4 substitui uma atração por outra não visitada, permitindo trade-offs de qualidade/custo. N_5 aplica reversão 2-opt, otimizando eficiência da rota. Estas vizinhanças têm tamanhos e impactos diferentes nos objetivos. Juntas, permitem exploração eficiente do espaço de soluções, cada uma direcionada a diferentes aspectos de qualidade." }

Slide 12: MOVNS: Archive and Shake

{ "Nosso arquivo externo mantém as soluções não-dominadas encontradas durante a busca. Iniciamos com 20 soluções diversas: 5 de heurísticas gulosas (cada uma direcionada a um objetivo diferente) e 15 aleatórias viáveis. Para gerenciar o arquivo, adicionamos novas soluções se não-dominadas, removemos soluções dominadas e limitamos a 30 usando contribuição de hipervolume. O procedimento de "shake" é crucial: aplicamos k movimentos aleatórios da vizinhança N_k, onde k varia de 1 a 5. Se encontramos melhoria, voltamos para k=1 (intensificação); caso contrário, aumentamos k (diversificação). Esta estratégia equilibra exploração e intensificação de forma eficaz." }

Slide 13: MOVNS: Core Algorithm

{ "Aqui está o algoritmo central do MOVNS. Iniciamos com o arquivo diversificado de 20 soluções. A cada iteração, selecionamos uma solução do arquivo (utilizando round-robin), aplicamos o procedimento de "shake" com força k, e então realizamos busca local de Pareto. Se encontramos uma solução não-dominada pelo arquivo, a adicionamos e reiniciamos k=1; caso contrário, aumentamos k para explorar vizinhanças mais distantes. Este processo continua até atingir 30 iterações sem melhoria. A beleza deste algoritmo é sua capacidade de alternar estrategicamente entre intensificação (explorar áreas promissoras) e diversificação (escapar de ótimos locais)." }

Slide 14: Performance Comparison

{ "Comparando os resultados, o MOVNS superou claramente o NSGA-II. Encontramos 48 soluções Pareto-ótimas contra 26 do NSGA-II - um aumento de 85. O hipervolume, que mede cobertura geral de trade-offs, foi 17 maior. O spread (Δ) foi reduzido em 31, indicando distribuição mais uniforme. O ε-indicator caiu 66, mostrando dominância mais forte. A única desvantagem foi o tempo de execução 140 maior, mas este é um trade-off aceitável pela qualidade superior das soluções. Estes resultados confirmam tendências observadas em estudos recentes de MOVNS em problemas similares de roteamento e agendamento." }

Slide 15: Economy Solution Highlights

{ "Vamos ver algumas soluções concretas. Esta é uma solução "econômica" - utiliza um hostel de apenas $35,75, mas consegue visitar 9 atrações com pontuação de qualidade impressionante de 49,2. Os destaques incluem o Parque Mount Royal, a Basílica de St. Patrick, o Museu de Arte Contemporânea e o Mural Leonard Cohen. O que torna esta solução fascinante é como ela alcança uma pontuação de qualidade superior mesmo com orçamento limitado. Isto foi possível através de roteamento estratégico e seleção cuidadosa de atrações. É um exemplo perfeito de como a otimização inteligente pode maximizar a experiência mesmo com restrições orçamentárias." }

Slide 16: Premium Solution Highlights

{ "No outro extremo, temos uma solução "premium" utilizando o Hyatt Centric por $529,25. Embora o custo total seja $694,10, esta solução oferece características únicas. Visita 8

atrações, incluindo o Jardim Botânico, Oasis Immersion, Parque Aquazilla e arte de rua. O mais impressionante é o equilíbrio perfeito entre os dias - apenas $0,20 de diferença! Esta solução também oferece menor pressão de tempo e experiências notavelmente variadas. Seria extremamente difícil encontrar manualmente um roteiro com tamanha simetria dia-a-dia enquanto oferece diversidade de experiências. Isto demonstra o poder do MOVNS em descobrir padrões não-intuitivos." }

Slide 17: Conclusions and Future Work

{ "Concluindo, nossas principais contribuições foram: um MOVNS especializado para planejamento turístico com cinco vizinhanças direcionadas; cobertura superior da fronteira de Pareto em comparação ao NSGA-II; um conjunto de dados de turismo de Montreal com dados reais de transporte multimodal; e coordenação estratégica entre hotel, atrações e transporte. Para trabalhos futuros, planejamos implementar replanejamento dinâmico para clima e interrupções, integração de preferências do usuário, implementação paralela para melhorar velocidade computacional, e desenvolvimento de aplicativo móvel. Nossa abordagem demonstra o potencial do MOVNS para problemas de planejamento urbano complexos e multiobjetivo." }

Slide 18: Acknowledgments

{ "Muito obrigado pela atenção! Nossos contatos estão no slide para discussões posteriores. Gostaria de agradecer especialmente ao OpenStreetMap e à Société de transport de Montréal pelos dados abertos, ao Comitê Organizador do ICVNS-2025 pela oportunidade, e ao Professor Daniel Aloise pelo apoio. Estou aberto a perguntas e interessado em discutir possíveis aplicações ou extensões desta pesquisa. Também ficarei feliz em compartilhar mais detalhes sobre a implementação ou os conjuntos de dados para quem tiver interesse." }

SLIDE 1:

SPEAKER NOTES:
- Explain the typical problems tourists face - limited time, many attractions
- Highlight how this is a perfect multi-objective problem (can't optimize all objectives simultaneously) falar sobre os 4 objetivos:

{ - Maximizing the number of attractions visited
- Maximizing quality (based on attraction ratings)
- Minimizing travel time between locations
- Minimizing total cost
}

- Mention the relevance to VNS research - complex constraints, multiple modes of transportation

- Emphasize how real-world data makes this problem challenging but valuable
- Note that conventional approaches struggle with the complexity

SLIDE 2:

SPEAKER NOTES:
- Explain how OSM provides rich geographical data with POI attributes
- Mention that GTFS data adds real-time transit scheduling
- Highlight OpenTripPlanner as a key technology that allows precise multi-modal routing
- Explain the challenge of generating all possible routes between locations
- Note that this approach can be used for any city with OSM and GTFS data
- Emphasize how this real-world data creates unique VNS research opportunities
- The main matrices include: attraction-to-attraction and hotel-to-attraction for each transport mode

SLIDE 3:

SPEAKER NOTES:
- These attractions represent the most popular and highest-rated in Montreal
- Explain how the attractions are distributed geographically throughout the city
- Note the different visit durations which create scheduling challenges
- Point out that opening hours add a hard constraint to the optimization problem
- Mention how some neighborhoods offer attraction clusters while others have isolated attractions
- Geographic dispersion means transportation mode selection becomes critical
- Explain that the balanced approach must consider both quality and visit duration

SLIDE 4:

SPEAKER NOTES:
- The hotel selection dramatically affects the entire optimization problem
- Central hotels enable more walking-only trips and reduce overall travel time
- Budget hotels often require more complex transportation planning
- The vast majority of optimal solutions use mixed transportation modes
- Strategic mode switching is key to optimizing both time and cost
- Multi-modal routes typically reduce travel time by 20-25 compared to single-mode
- Mount Royal Park is accessible by all transportation modes, making it appear in many solutions
- The solutions show clear patterns of mode selection based on distance and location

SLIDE 5:

SPEAKER NOTES:
- These applications represent domains where MOVNS has shown exceptional performance
- The manufacturing results are particularly relevant as they involve sequence-dependent setups similar to attraction sequencing
- Duarte's vehicle routing work demonstrated how VNS can effectively handle complex vehicle capacity and time window constraints

- System reliability results show MOVNS's ability to generate superior Pareto fronts compared to genetic algorithms
- Schilde's work is most directly relevant as an early application to tourism, though only bi-objective
- In all these cases, MOVNS showed stronger results than evolutionary approaches when problem structure could be exploited

SLIDE 6:

SPEAKER NOTES:
- Tourism planning fundamentally involves: 1) selecting attractions, 2) ordering them efficiently, and 3) choosing appropriate transport
- These sub-problems align perfectly with VNS strengths in combinatorial optimization
- Spatial clustering means small changes to routes often yield significant improvements—ideal for neighborhood search
- Time windows create complex dependencies that are difficult for evolutionary algorithms to handle
- NSGA-II's random mutations often break constraints, requiring expensive repair operations
- MOVNS can make coordinated changes that preserve feasibility
- Hotel selection is particularly important as it affects all travel calculations
- The problem combines aspects of TSP, knapsack, and scheduling—all problems where VNS has proven effective
- Our specific neighborhood structures exploit these characteristics to outperform generic approaches

SLIDE 7:

SPEAKER NOTES:
- The problem combines elements of Team Orienteering, Time Windows, and Mode Selection
- Key decision variables include: hotel selection, attraction selection, sequence, and transport modes
- We use real data for all parameters: attraction ratings, visit durations, opening hours, etc.
- F1 maximizes the number of attractions visited across both days
- F2 maximizes the total quality score (sum of ratings)
- F3 minimizes total time (transit + visit durations)
- F4 minimizes total cost (hotel + attractions + transportation)
- Key constraints include: daily time budget (8am-8pm), attraction opening hours, no repeated visits
- Each day starts and ends at the selected hotel
- Transport modes can vary between consecutive locations
- This formulation creates a highly constrained 4-objective optimization problem

SLIDE 8

SPEAKER NOTES:
- NSGA-II is the most widely used multi-objective evolutionary algorithm
- It was chosen as our baseline due to its proven performance across many domains

- The algorithm uses non-dominated sorting to rank solutions based on Pareto dominance
- Crowding distance helps maintain diversity across the Pareto front
- The elitist mechanism ensures good solutions aren't lost between generations
- We implemented both Python and C++ versions for performance comparison
- The solution representation encodes:
  * Hotel selection (single index)
  * Two separate day itineraries (ordered lists of attractions)
  * Transport mode selection between consecutive locations
- The population size and generation count were determined through parameter tuning
- This provides a strong baseline for comparison with our MOVNS approach

SLIDE 9

SPEAKER NOTES:
- Initialization uses both random and heuristic approaches (quality-based, proximity-based)
- The crossover operator preserves order relationships between attractions
- We implemented multiple mutation operators to address different aspects of the solution
- Hotel change affects transportation times for all attractions
- Attraction add/remove directly impacts the number of attractions visited
- Day swaps balance the two daily schedules
- Transport mode changes affect both time and cost
- Constraint handling is particularly challenging:
  * Time windows create a scheduling dependency between attractions
  * Repair mechanisms remove attractions that violate constraints
  * Transportation feasibility must be checked between all consecutive locations
  * Each mutation requires extensive feasibility checking and potential repair
- The complexity of constraint handling creates computational overhead

SLIDE 10
SPEAKER NOTES:
- The four objectives create a complex trade-off space with many non-dominated solutions
- F1 ranges from 4 to 10 attractions in optimal solutions
- F2 shows quality scores ranging from 25.7 to 53.3
- F3 reveals time requirements from 738 to 1359 minutes
- F4 demonstrates costs from R$105 (budget) to R$694 (luxury)
- These ranges reflect the diversity of the Pareto front
- The key trade-offs include:
  * Adding more attractions increases quality but consumes time
  * Budget hotels save money but often require more travel time
  * Car transport is faster but more expensive than public transit
  * Higher-rated attractions improve quality scores but often cost more
- Different traveler types prefer different points on the Pareto front:
  * Budget travelers prioritize F4 (cost minimization)
  * Experience-focused travelers prioritize F2 (quality maximization)
  * Time-constrained travelers prioritize F3 (time minimization)
  * Comprehensive travelers prioritize F1 (seeing more attractions)

SLIDE 11

SPEAKER NOTES:
- The five neighborhood structures provide complementary ways to explore the solution space
- Internal swap (N1) improves sequence efficiency without changing attraction set
  * Example: Swapping Notre Dame and City Hall to reduce travel time
- Cross-day move (N2) balances days and addresses day-specific constraints
  * Example: Moving Mount Royal Park from Saturday to Sunday
- Insert/Remove (N3) directly changes the number of attractions visited
  * Example: Adding Leonard Cohen Mural to an existing itinerary
- Substitution (N4) allows quality/cost tradeoffs without changing count
  * Example: Replacing a garden with a museum of similar duration
- 2-opt reversal (N5) is a classic TSP operator that improves route efficiency
  * Example: Reversing a segment to eliminate crossing paths
- These neighborhoods have different sizes and impacts on objectives
- The combined effect allows efficient exploration of the solution space
- Each neighborhood targets different aspects of the solution quality

SLIDE12
SPEAKER NOTES:
- The external archive maintains all non-dominated solutions found during the search
- Initial archive construction is crucial for early convergence
- We use 5 greedy heuristics targeting different objectives:
  * Max-Attractions Greedy maximizes F1
  * Max-Rating Greedy maximizes F2
  * Min-Cost Greedy minimizes F4
  * Min-Travel-Time Greedy minimizes F3
  * Balanced Heuristic attempts to balance all objectives
- These provide diverse starting points across the Pareto front
- Archive management follows standard MOVNS principles
- HV contribution for truncation maintains diversity effectively
- The shake procedure:
  * Uses variable strength perturbation based on k
  * Applies random moves from the corresponding neighborhood
  * Implements strategic intensification and diversification
  * Balances exploration and exploitation
- This approach helps escape local optima while focusing on promising regions

SLIDE 13:

SPEAKER NOTES:
- This algorithm represents our Multi-Objective VNS for tour planning
- The archive is initially populated with a diverse set of solutions (targeting different objectives)
- We iterate through the archive in round-robin fashion to maintain diversity
- The key VNS principle is to systematically change neighborhoods when improvements stall
- Shaking with increasing strength (k) provides diversification

- ParetoLocalSearch explores all five neighborhoods sequentially:
  * N1: Internal day swaps (sequence optimization)
  * N2: Cross-day moves (balancing days)
  * N3: Insert/remove attractions (quantity adjustment)
  * N4: Substitution (quality/cost trade-offs)
  * N5: 2-opt reversals (route efficiency)
- The acceptance criterion is purely Pareto-based (dominance)
- After an improvement, we reset k=1 to intensify the search
- When stuck, we increase k to explore more distant neighborhoods
- Archive management uses hypervolume contribution to maintain diversity
- This approach effectively balances our four competing objectives

SLIDE 14:

SPEAKER NOTES:
- These results come from 30 independent runs of both algorithms
- The most striking difference is the number of Pareto-optimal solutions discovered
- Hypervolume is a composite metric measuring both convergence and diversity
- The spread metric (Δ) measures distribution uniformity (lower is better)
- The ε-indicator measures how much one front would need to be "shifted" to dominate another
- Runtime difference reflects the more intensive local search in MOVNS
- All metrics consistently favor MOVNS across all experimental runs

RESULTADOS

SPEAKER NOTES FOR ECONOMY SLIDE:
- This solution showcases MOVNS's ability to optimize on a budget
- The hotel costs only R$35.75 but the solution achieves the highest quality score (49.2)
- Despite the budget constraints, it visits 9 attractions (more than any premium solution)
- Strategic clustering by theme (cultural/art) optimizes travel time
- The mixed transportation approach (Car+Subway+Bus) saves money while maintaining efficiency
- Day 2 leverages Sunday special exhibitions at the Contemporary Art Museum
- This non-intuitive solution was discovered through neighborhood exploration
- Budget travelers can still experience Montreal's highlights with strategic planning
SPEAKER NOTES FOR PREMIUM SLIDE:
- This solution demonstrates MOVNS's capacity for perfect balancing
- The premium hotel costs R$529.25 but enables efficient routing
- Each day has almost identical costs (R$82.52 and R$82.33) - a difference of just R$0.19
- The experience mix is highly varied (nature, immersive tech, art, water activities)
- Time pressure is reduced with the central hotel location (68 vs 76 utilization)
- The solution handles the hotel cost by selecting attractions with optimal value
- This appeals to travelers who prioritize comfort and balance over pure quantity
- The day-to-day symmetry would be difficult to achieve manually or with simpler algorithms

Montreal Tour Planning Project Documentation

[Etapa I : Projeto de Dados]

> > **Objetivo**
O projeto "Montreal-Tour-Planning" visa aplicar técnicas de otimização multiobjetivo para criar um itinerário de fim de semana ideal em Montreal, equilibrando custo, qualidade e tempo. Utilizamos o OpenTripPlanner (OTP) para calcular itinerários entre atrações turísticas, considerando diferentes modos de transporte: caminhada (`WALK`), ônibus (`BUS,WALK`), e metrô (`SUBWAY,WALK`). Esta documentação detalha os passos realizados para configurar o ambiente, testar itinerários, e visualizar resultados na interface do OTP.

> > **Passos para Replicação**

> > # **Passo 1: Coleta de Dados Iniciais**
- **Dados GTFS da Société de transport de Montréal (STM):** Os dados GTFS foram baixados e armazenados no diretório `/home/augusto/projects/Montreal-Tour-Planning/GTFS/GTFS_stm.zip`.
- **Dados OSM de Montreal:** Baixamos os dados OSM para a região de Quebec e recortamos a área de Montreal.
  - **Fonte:** Geofabrik - Quebec (https://download.geofabrik.de/north-america/canada/quebec.html).
  - **Bounding Box Ajustado:** Baseado na relação de Montreal no OSM (https://www.openstreetmap.org/relation/1634158):
    - Oeste: -73.9388, Sul: 45.4280, Leste: -73.4758, Norte: 45.7046.
  - **Comando de Recorte:** Utilizamos `osmium` para recortar o arquivo `quebec-latest.osm.pbf`:
    ```
    osmium extract -b "-73.9388,45.4280,-73.4758,45.7046" quebec-latest.osm.pbf -o montreal.osm.pbf
    ```
  - **Arquivo Resultante:** `/home/augusto/projects/Montreal-Tour-Planning/PBF/montreal.osm.pbf`.

> > # **Passo 2: Configuração do OpenTripPlanner (OTP)**
- **Diretório do OTP:** Criamos o diretório `/home/augusto/projects/Montreal-Tour-Planning/OTP`.
- **Download do OTP:** Baixamos o arquivo `.jar` da versão mais recente do OTP (2.5.0) diretamente para o diretório OTP:
  ```
  wget https://repo1.maven.org/maven2/org/opentripplanner/otp/2.5.0/otp-2.5.0-shaded.jar
  ```

  ```
- **Execução do OTP:** Apontamos os arquivos OSM e GTFS sem copiá-los, usando o comando ajustado para construir o grafo e iniciar o servidor:
  ```
  java -Xmx4G -jar otp-2.5.0-shaded.jar --build --serve --basePath /home/augusto/projects/Montreal-Tour-Planning --osm PBF/montreal.osm.pbf --gtfs GTFS/GTFS_stm.zip
  ```
- **Validação:** Após a construção do grafo (~5-15 minutos), acessamos `http://localhost:8080` no navegador para confirmar que o OTP estava funcionando.

> > # **Passo 3: Definição de Atrações Turísticas**
- **Lista de Atrações:** Utilizamos as coordenadas fornecidas no anexo:
  - Notre-Dame Basilica: (45.504885, -73.555992)
  - Mount Royal Park: (45.501724, -73.593261)
  - Montreal Botanical Garden: (45.559855, -73.563753)
  - Saint Joseph's Oratory: (45.491980, -73.616520)
  - Biodôme De Montreal: (45.559820, -73.549719)
  - Maison Saint-Gabriel Museum: (45.476142, -73.555602)

> > # **Passo 4: Testes Iniciais de Rotas**
- **Exemplos Iniciais:** Testamos três pares de atrações para validar os modos de transporte:
  1. Notre-Dame Basilica → Mount Royal Park
  2. Mount Royal Park → Saint Joseph's Oratory
  3. Montreal Botanical Garden → Biodôme De Montreal
- **Comandos Iniciais (via `curl`):**
  - **Exemplo 1 (Notre-Dame Basilica → Mount Royal Park):** `curl "http://localhost:8080/otp/routers/default/plan?fromPlace=45.504885,-73.555992&toPlace=45.501724,-73.593261&mode=TRANSIT,WALK&date=2025-04-05&time=12:00"`
    - Resultado: ~41 min, Linha 2 - Orange e ônibus 711.
  - **Exemplo 2 (Mount Royal Park → Saint Joseph's Oratory):** `curl "http://localhost:8080/otp/routers/default/plan?fromPlace=45.501724,-73.593261&toPlace=45.491980,-73.616520&mode=TRANSIT,WALK&date=2025-04-05&time=12:00"`
    - Resultado: Melhor itinerário com ônibus 711 (~21 min), outros com 11, 165, e caminhada (~42 min).
  - **Exemplo 3 (Montreal Botanical Garden → Biodôme De Montreal):** `curl "http://localhost:8080/otp/routers/default/plan?fromPlace=45.559855,-73.563753&toPlace=45.559820,-73.549719&mode=TRANSIT,WALK&date=2025-04-05&time=12:00"`
    - Resultado: Caminhada pura (~20 min, 1,4 km), erro "TOO_CLOSE", não usou trânsito como esperado (ônibus 185 ou Linha Verde).

> > # **Passo 5: Ajuste do Exemplo 3 para Forçar Trânsito**
- **Comando Ajustado:** `curl "http://localhost:8080/otp/routers/default/plan?fromPlace=45.559855,-
  ```

73.563753&toPlace=45.559820,-73.549719&mode=TRANSIT&date=2025-04-05&time=12:00&maxWalkDistance=500"`
- **Resultado:** Caminhada pura (~20 min, 1,4 km), erro "TOO_CLOSE", indicando que a distância curta (~1,4 km) impedia o uso de trânsito.
- **Decisão:** Escolhemos um trajeto mais longo para evitar o problema "TOO_CLOSE".

> > # **Passo 6: Teste com Trajeto Mais Longo**
- **Atrações Escolhidas:** Maison Saint-Gabriel Museum (45.476142, -73.555602) → Montreal Botanical Garden (45.559855, -73.563753) (~9 km).
- **Comandos Iniciais:**
  - **Walk Only:** `curl "http://localhost:8080/otp/routers/default/plan?fromPlace=45.476142,-73.555602&toPlace=45.559855,-73.563753&mode=WALK&date=2025-04-05&time=12:00"`
    - Resultado: Caminhada pura (~155 min, 11,3 km).
  - **Walk + Bus Only:** `curl "http://localhost:8080/otp/routers/default/plan?fromPlace=45.476142,-73.555602&toPlace=45.559855,-73.563753&mode=BUS,WALK&date=2025-04-05&time=12:00&maxWalkDistance=700"`
    - Resultado: Caminhada pura (~155 min), não usou ônibus.
  - **Walk + Rail Only:** `curl "http://localhost:8080/otp/routers/default/plan?fromPlace=45.476142,-73.555602&toPlace=45.559855,-73.563753&mode=RAIL,WALK&date=2025-04-05&time=12:00&maxWalkDistance=700"`
    - Resultado: Caminhada pura (~155 min), erro "PATH_NOT_FOUND".
- **Correção:** Substituímos `RAIL` por `SUBWAY`:
  - `curl "http://localhost:8080/otp/routers/default/plan?fromPlace=45.476142,-73.555602&toPlace=45.559855,-73.563753&mode=SUBWAY,WALK&date=2025-04-05&time=12:00&maxWalkDistance=700"`
    - Resultado: ~61 min, Linha 1 - Verde, correto.

> > # **Passo 7: Ajuste para Matriz e Horário de Alta Disponibilidade**
- **Ajustes:** Alteramos o horário para 08:00 (alta disponibilidade) e limitamos a 1 itinerário com `numItineraries=1`.
- **Comandos Ajustados:**
  - **Walk Only:** `curl "http://localhost:8080/otp/routers/default/plan?fromPlace=45.476142,-73.555602&toPlace=45.559855,-73.563753&mode=WALK&date=2025-04-05&time=08:00&optimize=QUICK&numItineraries=1"`
    - Resultado: ~155 min, correto.
  - **Walk + Bus Only:** `curl "http://localhost:8080/otp/routers/default/plan?fromPlace=45.476142,-73.555602&toPlace=45.559855,-73.563753&mode=BUS,WALK&date=2025-04-05&time=08:00&maxWalkDistance=700&searchWindow=7200&optimize=QUICK&numItineraries=1"`
    - Resultado: Caminhada pura (~155 min), incorreto.
  - **Walk + Subway Only:** `curl "http://localhost:8080/otp/routers/default/plan?fromPlace=45.476142,-73.555602&toPlace=45.559855,-73.563753&mode=SUBWAY,WALK&date=2025-04-05&time=08:00&maxWalkDistance=700&optimize=QUICK&numItineraries=1"`
    - Resultado: ~61 min, correto.

> > # **Passo 8: Remoção de `maxWalkDistance`**
- **Ajustes:** Removemos `maxWalkDistance` para permitir otimização livre.
- **Comandos Ajustados:**
  - **Walk + Bus Only:** `curl "http://localhost:8080/otp/routers/default/plan?fromPlace=45.476142,-73.555602&toPlace=45.559855,-73.563753&mode=BUS,WALK&date=2025-04-05&time=08:00&searchWindow=7200&optimize=QUICK&numItineraries=1"`
    - Resultado: Caminhada pura (~155 min), incorreto.
  - **Walk + Subway Only:** `curl "http://localhost:8080/otp/routers/default/plan?fromPlace=45.476142,-73.555602&toPlace=45.559855,-73.563753&mode=SUBWAY,WALK&date=2025-04-05&time=08:00&optimize=QUICK&numItineraries=1"`
    - Resultado: ~61 min, correto.

> > # **Passo 9: Ajuste com `walkReluctance`**
- **Ajustes:** Introduzimos `walkReluctance=10` para forçar o uso de ônibus, removendo `optimize=QUICK`.
- **Comando Final:** `curl "http://localhost:8080/otp/routers/default/plan?fromPlace=45.476142,-73.555602&toPlace=45.559855,-73.563753&mode=BUS,WALK&date=2025-04-05&time=08:00&searchWindow=7200&numItineraries=1&walkReluctance=10"`
- **Resultado:** ~85 min, ônibus 57, 24, 97, correto.

> > # **Passo 10: Geração de URLs para Visualização na Interface**
- **URLs Geradas:**
  - **Walk Only:** `http://localhost:8080/?module=planner&fromPlace=45.4761422C-73.555602&toPlace=45.5598552C-73.563753&time=83A00am&date=04-05-2025&mode=WALK&optimize=QUICK&numItineraries=1&arriveBy=false&wheelchair=false&showIntermediateStops=true&locale=en&baseLayer=OSM20Standard20Tiles`
  - **Walk + Subway Only:** `http://localhost:8080/?module=planner&fromPlace=45.4761422C-73.555602&toPlace=45.5598552C-73.563753&time=83A00am&date=04-05-2025&mode=SUBWAY2CWALK&optimize=QUICK&numItineraries=1&arriveBy=false&wheelchair=false&showIntermediateStops=true&locale=en&baseLayer=OSM20Standard20Tiles`
  - **Walk + Bus Only:** `http://localhost:8080/?module=planner&fromPlace=45.4761422C-73.555602&toPlace=45.5598552C-73.563753&time=83A00am&date=04-05-2025&mode=BUS2CWALK&searchWindow=7200&numItineraries=1&walkReluctance=10&arriveBy=false&wheelchair=false&showIntermediateStops=true&locale=en&baseLayer=OSM20Standard20Tiles`

> # Como o Conjunto Resultou na Imagem
A imagem da interface mostra o itinerário "Walk + Bus Only", gerado a partir da URL fornecida no Passo 10: `http://localhost:8080/?module=planner&fromPlace=45.4761422C-73.555602&toPlace=45.5598552C-73.563753&time=83A00am&date=04-05-2025&mode=BUS2CWALK&searchWindow=7200&numItineraries=1&walkReluctance=10&a

rriveBy=false&wheelchair=false&showIntermediateStops=true&locale=en&baseLayer=OSM 20Standard20Tiles`.

- **Caminho até a Imagem:**
  1. **Configuração do OTP (Passo 2):** Configuramos o OTP com os dados OSM e GTFS, permitindo que o servidor gerasse o grafo de roteamento e estivesse acessível em `http://localhost:8080`.
  2. **Testes Iniciais (Passo 4):** Identificamos que o OTP retornava caminhada pura para "Walk + Bus Only", mesmo quando ônibus eram esperados.
  3. **Ajustes Iterativos (Passos 5-9):** Testamos diferentes parâmetros, como horário (08:00), remoção de `maxWalkDistance`, e finalmente introduzimos `walkReluctance=10` para forçar o uso de ônibus.
  4. **Geração da URL (Passo 10):** A URL foi criada com os parâmetros finais, refletindo o itinerário bem-sucedido (~85 min, ônibus 57, 24, 97).
- **Resultado na Interface:** A imagem exibe o itinerário na interface do OTP, com:
  - **Rota:** Caminhada → Ônibus 57 (Charlevoix) → Caminhada → Ônibus 24 (Sherbrooke) → Ônibus 97 (Avenue-du-Mont-Royal) → Caminhada.
  - **Tempo Total:** ~85 min.
  - **Mapa:** Mostra o trajeto com linhas de ônibus e trechos de caminhada, correspondendo ao resultado do comando final.

> # Fontes Utilizadas
- **OpenStreetMap Wiki - Montréal:** https://wiki.openstreetmap.org/wiki/Montréal
- **OpenStreetMap - Relação de Montreal:** https://www.openstreetmap.org/relation/1634158
- **Geofabrik - Quebec:** https://download.geofabrik.de/north-america/canada/quebec.html
- **LatLong.net - Montreal:** https://www.latlong.net/place/montreal-quebec-canada-119.html
- **Ville de Montréal - Dados Abertos:** https://donnees.montreal.ca
- **OTP Download:** https://repo1.maven.org/maven2/org/opentripplanner/otp/2.5.0/otp-2.5.0-shaded.jar

---

> # Resumo
O projeto envolveu a configuração do OTP com dados OSM e GTFS, testes iterativos para validar itinerários entre atrações turísticas, ajustes para forçar o uso de transporte público, e visualização dos resultados na interface do OTP. O itinerário final "Walk + Bus Only" foi exibido com sucesso, refletindo o uso de ônibus e caminhada conforme esperado.

[Etapa II : Projeto da MOVNS]

Project Overview

This project implements a multi-objective optimization system for planning tourist itineraries in Montreal. It uses the NSGA-II genetic algorithm to generate 2-day tours that balance multiple competing objectives:

- Maximizing the number of attractions visited
- Maximizing quality (based on attraction ratings)
- Minimizing travel time between locations
- Minimizing total cost

The system uses real transportation data from OpenStreetMap and Montreal's public transit system to calculate travel times between locations using various transportation modes.

Key Components

1. Data Sources

- Attractions: Stored in places/attractions.csv with details like name, location, ratings, opening hours, and visit duration
- Hotels: Stored in places/hotels.csv with price, rating, and coordinates
- Transport Data:
  - OpenStreetMap data (montreal.osm.pbf)
  - Montreal transit data (gtfs_stm.zip)
  - Used by OpenTripPlanner to calculate travel times

2. Matrix Generation

- Local OpenTripPlanner (OTP) server for route calculations
- Scripts in tools/ generate matrices of travel times:
  - Between attractions
  - From hotels to attractions (morning)
  - From attractions to hotels (evening)
- Four transport modes supported: Walking, Subway+Walking, Bus+Walking, Car

3. Route Optimization (NSGA-II)

- Implemented in both Python (nsga2.py) and C++ (nsga2.cpp)
- Key operations:
  - Construction: Creates feasible initial solutions
  - Crossover: Combines parent solutions with constraint repair
  - Mutation: Several types including hotel change, attraction add/remove/swap, and transport mode changes
  - Selection: Tournament selection with crowded-comparison operator

4. Visualization and Analysis

- Pareto front visualization with interactive web interface
- Solution details with day-by-day itineraries
- Analysis tools for matrix quality and solution verification

Workflow

1. Setup and Matrix Generation
Run OTP server locally (prerequisite)
Calculate travel time matrices
python tools/calculate_matrix.py
2. Route Optimization
Either run Python version
python run.py

Or build and run C++ version
./run.sh
3. Result Visualization
python pareto_visualizer.py
Then open http://127.0.0.1:5000/ in browser

Technical Implementation

Tour Representation

- Each tour solution contains:
  - Selected hotel
  - Two daily routes (Saturday and Sunday)
  - Transport modes between locations

Constraints

- Attraction opening hours (Saturday/Sunday specific)
- Daily time budget (8:00 AM - 8:00 PM)
- Feasible transportation connections
- No repeated attractions between days

Algorithm Parameters

- Population size: 200 solutions
- Generations: 100
- Crossover probability: 0.9
- Mutation probability: 0.2
- Various mutation operators with different weights

Results Output

The system generates several result files:

- nsga2-initial-population.csv: Random initial solutions
- nsga2-final-population.csv: Complete final population
- nsga2-sorted-population.csv: Pareto-optimal solutions
- nsga2-generations.csv: Convergence metrics per generation

The web visualization provides an interactive way to explore the Pareto-optimal solutions and understand the trade-offs between objectives.

Development Environment

The project uses a hybrid approach:
- Python for most components and algorithm implementation
- C++ for performance-critical parts (compiled with CMake)
- Flask for the visualization interface

Execution is managed by both run.py (Python) and run.sh (C++ build and execution).

[Etapa III : MODELAGEM]

```
================================================================================
TWO–DAY MONTREAL TOUR PLANNING: COMPLETE MOVNS SPECIFICATION
================================================================================
\documentclass[a4paper,11pt]{article}
\usepackage[utf8]{inputenc}
\usepackage{geometry}\geometry{margin=2.5cm}
\usepackage{amsmath,amssymb}
\usepackage{booktabs}
\usepackage{listings}
\usepackage{hyperref}
\usepackage{enumitem}
\usepackage{parskip}

\title{Multi-Objective Variable Neighbourhood Search (MOVNS)\\
for a Two-Day Tourist Itinerary in Montreal}
\author{}
\date{\today}

\begin{document}
\maketitle

\section{Problem Overview}

Given a set of candidate points of interest (POIs) in Montreal, the task
is to design a \textbf{two-day itinerary} that simultaneously optimises
four conflicting objectives:
```

```
\begin{description}[leftmargin=*,labelsep=0.5em]
\item[F1] \textbf{Maximise the number of attractions}
        visited over both days.
\item[F2] \textbf{Maximise the total quality}
        (e.g.\ rating or relevance score) of all visited POIs.
\item[F3] \textbf{Minimise total travel time}
        (inter-POI transit + visit durations).
\item[F4] \textbf{Minimise total monetary cost}
        (entrance fees + paid transport).
\end{description}

\paragraph{Operational constraints}
\begin{itemize}[leftmargin=*]
\item Each day runs from \(\text{08{:}00}=480\text{ min}\)
    to \(\text{20{:}00}=1200\text{ min}\).
\item Every POI \(i\) has opening window \([o_i,\ell_i]\)
    and fixed visit time \(v_i\).
\item No POI may be visited twice in the two days.
\item Each daily route starts and ends at the tourist's hotel.
\item Only valid transport modes (walking, subway, bus, car) may be used.
\end{itemize}

\section{Solution Representation}

An itinerary is encoded as two ordered sequences
\[
\mathcal{A}_1=\langle a_{1,1},\dots,a_{1,k}\rangle,\qquad
\mathcal{A}_2=\langle a_{2,1},\dots,a_{2,m}\rangle.
\]
Let \(q_i\) be quality, \(c_i\) fee,
\(t^{m}_{ij}\) travel time for mode \(m\),
\(\kappa^{m}_{ij}\) the corresponding cost.

\subsection*{Objective functions}
\begin{align}
F_1 &= k+m &\text{(max.)} \\[2pt]
F_2 &= \sum_{i\in\mathcal{A}_1\cup\mathcal{A}_2} q_i &\text{(max.)} \\[2pt]
F_3 &= \sum_{d=1}^{2}\!\Bigl[
      \sum_{(i,j)\in\text{seq}(d)} t^{m_{ij}}_{ij}
      +\!\!\sum_{i\in\mathcal{A}_d}\!v_i\Bigr] &\text{(min.)} \\[4pt]
F_4 &= \sum_{d=1}^{2}\!\Bigl[
      \sum_{i\in\mathcal{A}_d} c_i
      + \sum_{(i,j)\in\text{seq}(d)} \kappa^{m_{ij}}_{ij}\Bigr] &\text{(min.)}
\end{align}

\section{Neighbourhood Structures}
```

\begin{enumerate}[label=\(\mathcal{N}_{\arabic*}\),leftmargin=*]
  \item \textbf{Internal swap}: exchange two POIs in the same day.
  \item \textbf{Cross-day move}: shift one POI from day 1 to day 2 or vice-versa.
  \item \textbf{Insert / Remove}: add a new POI or drop an existing one.
  \item \textbf{Substitution}: replace a visited POI by an unvisited one.
  \item \textbf{2-opt reversal}: reverse a segment in one daily route.
\end{enumerate}

\section{Initial Archive Generation} NEW
\label{sec:seed}

We start with an \emph{elitist seed archive} of
\(\mathbf{|A_0| = 20}\) feasible itineraries:

\subsection*{Five heuristic seeds}
\begin{enumerate}[label=H\arabic*.,leftmargin=*]
  \item \textbf{Max-Attractions Greedy}
        – insert POIs (highest rating first) until no further visit fits
        the daily time window.
  \item \textbf{Max-Rating Greedy}
        – insert POIs by descending \(q_i\) even if quantity is small.
  \item \textbf{Min-Cost Greedy}
        – cheapest POIs first, skip any fee \(> \$\theta\) (user-set).
  \item \textbf{Min-Travel-Time Greedy}
        – build a tight cluster around the hotel (short arcs only).
  \item \textbf{Balanced Heuristic}
        – sequentially insert the POI with highest ratio
        \(\tfrac{q_i}{v_i+t^{\min}_{i}}\) while feasible.
\end{enumerate}

\subsection*{Fifteen random-feasible routes}
Each random seed is produced by:

\begin{enumerate}[leftmargin=*]
\item Draw a random subset of POIs (Bernoulli $p=0.3$).
\item Randomly permute them into Day 1; spill overflow into Day 2.
\item While a day violates time or opening windows,
      \emph{drop} the last POI of that day (time-repair loop).
\item If any POI remains duplicated, randomly remove one copy.
\end{enumerate}

All 20 seeds are evaluated on \((F_1,F_2,F_3,F_4)\); only the
non-dominated solutions compose the initial archive~\(A_0\).
Empirical studies on bi-objective orienteering
show 10–25 seeds give best HV gain vs.\ time
\cite{Schilde2009}.

```
\section{MOVNS Framework}

\subsection{External elitist archive \(A\)}
\begin{itemize}[leftmargin=*]

\item Initial archive: \(10\!-\!25\) We start with an \emph{elitist seed archive} 20 from the table
"Five heuristic seeds"
\item At every insertion:
    add \(x\) if non-dominated, delete dominated solutions.
\item Cap size at \(A_{\max}=30\) using
    \emph{hyper-volume contribution} truncation.
\end{itemize}

\subsection{Pseudocode}
\begin{lstlisting}[language={},frame=single,basicstyle=\ttfamily\small]
Input : archive A (|A|≈20), N1..N5, Tmax=120 s or 30 idle loops
k_max ← 5
repeat
   R ← next solution in A (round-robin)
   k ← 1
   while k ≤ k_max do
      R'  ← Shake(R, Nk)              // random move of size k
      R'' ← ParetoLocalSearch(R')      // VND on N1..N5
      if R'' non-dominated by A then
         A ← A ∪ {R''}; purge dominated
         HV-truncate(A, 30)           // elitist, bounded
         k ← 1                        // intensify
      else
         k ← k+1                      // diversify
      end if
   end while
until CPU ≥ Tmax or 30 loops with no HV increase
return archive A
\end{lstlisting}

\subsection{Local search options}
\begin{enumerate}[leftmargin=*]
\item \textbf{Weighted descent}:
    draw random weights \(\lambda\) s.t.\(\sum\lambda_i=1\),
    minimise \(F=\sum\lambda_i f_i\).
\item \textbf{Pareto Local Search}: explore all neighbours,
    add every non-dominated neighbour to a local archive,
    iterate until none remains.
\end{enumerate}

\section{Constraint Handling}

Moves violating day limits, opening windows or hotel return
```

are discarded \emph{a priori}.
Optionally apply a large time–penalty to \(F_3\) when a move
temporarily exceeds the horizon (simpler code, same effect).

\section{Quality Monitoring}

\begin{itemize}[leftmargin=*]
\item \textbf{Hyper-volume (HV)} — measures convergence + diversity,
    monotone under elitism.
\item \textbf{Spread \(\Delta\)} — spacing indicator; if \(\Delta>0.35\)
    for 50 iterations, force a jump to \(\mathcal{N}_{5}\).
\item \textbf{Additive \(\varepsilon\)-indicator} —
    compare \(A_t\) with \(A_{t-10}\);
    early stop when \(\varepsilon<0.05\) for three successive windows.
    (Requires two Pareto sets; meaningless for a single set.)
\end{itemize}

\section{Hyper-parameters}

\begin{table}[h]
\centering
\begin{tabular}{@{}lll@{}}
\toprule
Parameter & Recommended value & Rationale \\\midrule
Initial archive $|A_0|$ & 20 routes & Fast convergence without loss of coverage
\cite{Schilde2009} \\
$k_{\max}$ & 5 neighbourhoods & Diminishing returns beyond six \cite{Duarte2014} \\
Shake depth & k & Classical GVNS rule \cite{Duarte2015} \\
Archive cap $A_{\max}$ & 30 & HV-based truncation retains diversity \cite{Jiang2015} \\
Stop rule & 120 s or 30 idle loops & Matches fast MOVNS benchmarks \cite{LiTian2022}
\\\bottomrule
\end{tabular}
\end{table}

\section{Expected Outcome}

MOVNS returns a Pareto front of feasible two-day routes,
e.g.\ one solution may visit 12 medium-quality POIs
(low cost, long travel), another 8 top-rated POIs
(high quality, moderate travel/cost), \emph{etc.}

Empirical studies (\cite{LiangLo2010,Duarte2015}) show MOVNS
often yields \emph{more} non-dominated solutions than NSGA-II,
thanks to stronger local refinement and fewer parameters.

\section{CSV Outputs}

\begin{enumerate}[leftmargin=*]

\item \verb|route_solution.csv| — final itinerary
    (day, order, POI, start, end, transport, duration, cost, rating).
\item \verb|movns_execution_log.csv| — per-iteration
    (iteration, HV, Δ, ε, F_1–F_4, k, archive size).
\end{enumerate}

\section{References}

\begin{thebibliography}{10}
\bibitem{Hansen2010} Hansen, P., Mladenović, N., Moreno-Pérez, J.~A.
\emph{Variable Neighbourhood Search: Methods and Applications}.
Annals of Oper.\ Research 175 (2010) 367–407.

\bibitem{Duarte2015} Duarte, A.\ et al.
\emph{Multi-Objective Variable Neighbourhood Search: An Application to
Combinatorial Optimisation Problems}. J.~Global Optim.\ 63 (3) (2015).

\bibitem{LiangLo2010} Liang, Y.\,C., Lo, M.\,H.
\emph{Multi-Objective Redundancy Allocation Optimisation Using a VNS}.
J.~Heuristics 16 (3) (2010).

\bibitem{Schilde2009} Schilde, M.\ et al.
\emph{Bi-Objective Orienteering: Mountain Guide Case Study}.
4OR 7 (4) (2009).

\bibitem{Duarte2014} Duarte, A.\ et al.
(PDF) Multi-Objective Variable Neighbourhood Search – report, 2014.

\bibitem{Jiang2015} Jiang, S.\ et al.
\emph{Hyper-volume Indicator-Based MOEA}.
IEEE Trans.\ Cybernetics 45 (10) (2015).

\bibitem{LiTian2022} Li, K., Tian, H.
\emph{A Multi-Objective VNS with Learning and Swarm for Flowshop Scheduling}.
Processes 10 (9) 1786 (2022).
\end{thebibliography}

\end{document}

[Etapa IV : RESULTADOS]

Great. I will research how to rigorously apply Multi-Objective Variable Neighbourhood Search (MOVNS) to the Python library recommendation problem, ensuring:

* Formal definitions of the three objectives (Linked Usage, Semantic Similarity, Set Size);

* Justified neighbourhoods and solution representation for set-based recommendation;
* Best practices for elitist Pareto archive handling (to ensure non-decreasing HV);
* Monitoring with HV, Spread (Δ), and ε-indicator;
* Efficient evaluation methods for LU and SS;
* CSV output formats for recommended sets and iteration logs;
* Guidelines for practical time-constrained execution.

I'll return a fully structured LaTeX-ready methodology that can be used in a scientific paper, with cited sources to justify each modeling and algorithmic decision.

Multi-Objective VNS Methodology for Library Recommendation

> Problem Formulation and Objectives

Given a target library or a developer's current set of libraries (the *context* X), our goal is to recommend a subset S of additional Python libraries that optimizes multiple criteria simultaneously. We define U as the universe of candidate libraries. The recommended set $S \subseteq U$ must exclude any libraries already in the context ($S \cap X = \emptyset$) and must contain at least one library ($|S| \ge 1$). We formalize three objective functions for evaluating a candidate set S: (1) **Linked Usage (LU)** – a co-usage utility reflecting how often libraries in S are used together with the context; (2) **Semantic Similarity (SS)** – a textual coherence score based on library descriptions; and (3) **Recommended Set Size (RSS)** – the number of libraries in S, which we aim to minimize. This multi-objective formulation reflects the need in real-world recommender systems to balance multiple competing goals rather than optimize a single metric.

Linked Usage (LU): We assume access to a co-usage frequency matrix $W = [w_{ij}]$, where w_{ij} quantifies the utility or frequency of libraries i and j being used together (e.g. number of projects or codebases that import both). We define LU as the average co-usage score between the recommended libraries and the context libraries. If X is the set of context libraries (which could be a single target library or multiple libraries), we formally define:

$$
f_{1}(S) = LU(S) \;=\; \frac{1}{|X|\cdot|S|} \sum_{i \in X}\sum_{j \in S} w_{ij}\,.
$$

This objective is to be maximized – higher $LU(S)$ means the recommendation set S tends to be used frequently alongside the context, indicating practical relevance. For example, if $X=\{\text{NumPy}\}$ and $S=\{\text{Pandas}, \text{Matplotlib}\}$, then $LU(S)$ might aggregate the co-usage frequencies of NumPy with Pandas and NumPy with Matplotlib. A higher value suggests that libraries in S are commonly linked to the context in usage.

Semantic Similarity (SS): To ensure the recommendations are topically coherent with the context, we evaluate textual similarity using TF–IDF vectors of library descriptions or documentation. Let $\mathbf{v}_i$ be the TF–IDF feature vector for library i. We define

semantic similarity as the average cosine similarity between the context libraries and the recommended libraries:

$$
f_{2}(S) = SS(S) \;=\; \frac{1}{|X|\cdot|S|} \sum_{i \in X}\sum_{j \in S} \frac{\mathbf{v}_i \cdot \mathbf{v}_j}{\|\mathbf{v}_i\|\,\|\mathbf{v}_j\|}\,.
$$

This objective (to be maximized) measures textual alignment: a higher $SS(S)$ indicates that the libraries in S cover topics or functionalities similar to those of the context (e.g. recommending libraries in the same domain or with related documentation keywords). Using TF–IDF with cosine similarity for recommendation relevance is a standard approach to quantify content-based similarity between items.

Recommended Set Size (RSS): We also seek concise recommendations. The third objective is the size of the recommended set, $f_{3}(S) = |S|$, which we aim to **minimize**. A smaller set S is preferable if it still provides high utility and relevance, to avoid overloading the user with too many suggestions. This objective introduces an explicit preference for parsimony, penalizing large recommendation sets. We treat f_{3} as a minimization objective (or equivalently, $-|S|$ as a maximization objective) in our multi-objective framework. The constraint $|S| \ge 1$ ensures at least one library is recommended beyond the input context. In practice, this third objective pushes the solution toward finding the most *efficient* set of libraries that achieves high LU and SS scores.

In summary, our optimization problem is a **multi-objective subset selection**:

$$
\max_{S \subseteq U,\,S \cap X = \emptyset} \;\{f_{1}(S),\;f_{2}(S)\}, \qquad \min_{S \subseteq U,\,S \cap X = \emptyset} \;\{f_{3}(S)\},
$$

with $|S|\ge1$. We aim to approximate the **Pareto-optimal set** of solutions, i.e. sets S where no other feasible set is strictly better in all three objectives. This formulation is analogous to a multi-criteria 0–1 selection (knapsack-like) problem, where each library inclusion is a binary decision. A binary vector representation for solutions is thus natural: we can represent a candidate solution S by a bitvector $x \in \{0,1\}^{|U|}$, where \$x_j = 1\$ if library \$j\$ is in the recommended set (and \$x_j = 0\$ otherwise). This encoding is commonly used in multi-objective combinatorial problems like bi-criteria feature or item selection, and it facilitates efficient neighbor generation via bit flips.

> Solution Representation and Feasibility

Each candidate solution is encoded as either (a) a bitvector of length \$|U|\$ or (b) an explicit list of library IDs in the recommended set. For algorithmic operations, the bitvector encoding is convenient, as adding or removing a library corresponds to flipping a bit from 0 to 1 or 1 to 0. We maintain the context libraries \$X\$ as a fixed set (not part of the decision vector). All solutions must satisfy the feasibility constraints: no library in \$X\$ is included in \$S\$, and at least one bit in the solution vector is 1 (ensuring \$|S|\ge1\$). If a neighbor operation would

violate these constraints (e.g. removing the only library in \$S\$, resulting in \$|S|=0\$), that move is disallowed.

During evaluation of objectives, we treat the context \$X\$ as given. The LU and SS objectives are computed relative to \$X\$ as per the formulas above. Because \$X\$ is fixed for a given recommendation task, it does not change during the search – only \$S\$ (the additional recommended libraries) is subject to optimization. By structuring the solution as a binary inclusion vector, we leverage efficient incremental updates: for example, adding a library \$j\$ to \$S\$ allows updating \$LU(S)\$ by adding the contributions \$w_{ij}\$ for all \$i\in X\$ (rather than recomputing from scratch), and similarly updating \$SS(S)\$ by computing cosine similarities of \$\mathbf{v}_j\$ with each \$\mathbf{v}_i\$ for \$i\in X\$. This helps keep the algorithm runtime within realistic limits even as it evaluates many candidate sets.

> Variable Neighborhood Structures

To search the combinatorial space of library subsets effectively, we employ a **Variable Neighbourhood Search (VNS)** metaheuristic adapted to multiple objectives. VNS systematically changes the neighborhood size during search to escape local optima. We design a set of neighborhood structures tailored to the subset selection nature of the problem:

* **\$N_1\$ (Addition moves):** All solutions obtainable by **adding** one library to the current set \$S\$. Formally, for current \$S\$, \$N_1(S) = {S \cup {\ell} : \ell \in U\setminus (S \cup X)}\$. Each neighbor in this structure increments the size of the set by 1 (thus potentially worsening \$f_3\$ but offering opportunity to improve \$f_1\$ and \$f_2\$). This move introduces a new library that was not previously recommended.

* **\$N_2\$ (Removal moves):** All solutions obtainable by **removing** one library from \$S\$. That is, \$N_2(S) = {S \setminus {\ell} : \ell \in S}\$. These neighbors reduce the set size by 1 (improving \$f_3\$ at the cost of possibly lowering \$f_1\$ and \$f_2\$). We restrict this move to cases where \$|S \setminus {\ell}| \ge 1\$ to maintain feasibility (never remove the last library).

* **\$N_3\$ (Swap moves):** All solutions obtainable by **swapping** one library in \$S\$ with one not in \$S\$. Formally, \$N_3(S) = {(S \setminus {\ell_{\text{out}}}) \cup {\ell_{\text{in}}} : \ell_{\text{out}} \in S,\ \ell_{\text{in}} \in U\setminus(S \cup X)}\$. A swap simultaneously removes a library from \$S\$ and adds a different library. This keeps the set size \$|S|\$ constant, potentially trading a less useful library for a more relevant one. Swaps can yield larger jumps in objective space (affecting LU and SS in two ways at once) and are useful when both an addition and removal might be needed to improve the solution.

* **Optional Larger Moves:** In classical VNS, one can define neighborhoods of increasing "size" \$k\$ (e.g. adding or removing \$k\$ libraries at once) to escape deeper local optima. In our MOVNS design, \$N_1\$, \$N_2\$, \$N_3\$ are the primary neighborhoods (single-add, single-remove, single-swap). We found these are sufficient to produce a diverse set of Pareto-optimal recommendations. However, the framework allows larger composite moves if needed (e.g. \$N_4\$: add two libraries; \$N_5\$: remove two libraries; etc.), which could be

used in a *shaking* step for diversification (described below). Such multi-library moves constitute a **large neighborhood search** extension, similar to strategies used in other multi-objective VNS applications. For the sake of run-time efficiency and simplicity, we restrict our focus to the three fundamental operators, which are standard in subset selection heuristics.

These neighborhood structures ensure a comprehensive exploration of the search space: additions explore making the recommendation set more comprehensive, removals explore simplifying the set, and swaps fine-tune the composition without changing size. By systematically changing neighborhoods, the algorithm can escape local optima and search both **intensification** (small tweaks via swaps) and **diversification** (bigger jumps via additions/removals) pathways. The design of multiple move types is in line with best practices in multi-objective VNS: for example, a recent MOVNS for a routing problem used nine different neighborhood move operators to explore trade-offs between objectives. Our selection of moves is justified by the nature of the library recommendation problem and keeps the search efficient.

> Multi-Objective VNS Algorithm

We adopt a **Pareto-based Multi-Objective Variable Neighbourhood Search (MOVNS)** algorithm to find a diverse set of non-dominated recommendation sets. The overall procedure can be summarized as follows:

1. **Initial Solution:** Construct an initial feasible set \$S_0\$. This can be done heuristically – for example, start with the single library that has the highest LU score with respect to the context (ensuring at least one recommendation). \$S_0\$ could also be seeded with a couple of top-ranked libraries (one maximizing LU, one maximizing SS) to give a reasonable starting point. We compute \$f_1, f_2, f_3\$ for \$S_0\$ and initialize the **elitist archive** \$\mathcal{A}\$ with this solution (\$\mathcal{A}={S_0}\$). In our approach, we start with a unique initial solution, as is common in multi-objective VNS, rather than a population.

2. **Pareto Local Search (Neighborhood Descent):** We perform an iterative improvement (akin to Variable Neighborhood Descent) on the current solution with respect to each neighborhood structure \$N_1, N_2, N_3\$. We explore neighbors in increasing order of neighborhood index:

   * **Addition phase (\$N_1\$):** Evaluate all single-add neighbors of the current solution \$S\$. For each neighbor \$S'\$ in \$N_1(S)\$, determine its objective vector \$(f_1(S'), f_2(S'), f_3(S'))\$. If \$S'\$ is **not dominated** by any solution in the archive \$\mathcal{A}\$ (meaning \$S'\$ has some merit in objectives that is not worse across the board) then \$S'\$ is a Pareto-improving candidate. We add \$S'\$ to \$\mathcal{A}\$ if it is non-dominated (and correspondingly remove any archived solutions that \$S'\$ dominates). This archive update step implements an *elitist strategy*: only non-dominated solutions are preserved. We do **not** immediately move to any one of these new solutions; instead, we continue evaluating all neighbors in \$N_1(S)\$ to collect **all** Pareto-improving solutions. (In a multi-objective context, multiple neighbors can be desirable in different objective trade-offs, so we consider the entire set of improvements rather than a single “best” neighbor.)

* **Removal phase (\$N_2\$):** After exhausting additions, we similarly evaluate all single-removal neighbors of \$S\$. Any neighbor \$S'\$ that is non-dominated (not worse in all objectives than some archive member) is added to the archive. Removals might improve \$f_3\$ significantly and could uncover simpler recommendation sets that still offer good LU and SS.
* **Swap phase (\$N_3\$):** Next, evaluate all swap neighbors. Swaps can yield subtle improvements; e.g., replacing one library with another that has higher LU and SS can produce a dominating solution. Again, add any non-dominated neighbors to \$\mathcal{A}\$.

This multi-objective neighborhood exploration is essentially a **Pareto Local Search (PLS)** on each neighborhood. We say the current solution \$S\$ has reached a *Pareto local optimum* with respect to these neighborhoods when no neighbor in \$N_1, N_2,\$ or \$N_3\$ produces a new non-dominated solution. At that point, no single addition, removal, or swap can improve \$S\$ in all objectives simultaneously. The set of solutions in \$\mathcal{A}\$ now represents all immediate improvements found from \$S\$.

3. **Archive Management and Current Solution Update:** The archive \$\mathcal{A}\$ is an **elite set** storing all non-dominated solutions found so far. We manage \$\mathcal{A}\$ with an approach that guarantees **non-decreasing hypervolume**: each time we add a new solution, it either increases the dominated volume in objective space or lies in an existing covered region. Any newly added solution that is non-dominated will weakly increase the archive's hypervolume (and if it dominated and removed some old solutions, the net hypervolume still does not decrease). This monotonic archive update is a hallmark of elitist multi-objective search – once a Pareto-optimal region is discovered, it is never lost. We maintain \$\mathcal{A}\$ without an explicit size limit (assuming the number of Pareto-optimal trade-offs remains manageable); if needed, a pruning strategy could be applied, but we choose an exhaustive archive to accurately track the Pareto front.

After exploring all neighborhoods for the current solution \$S\$, we mark \$S\$ as fully explored. Next, we need to select a new *current solution* to continue the search from. A common strategy in Pareto Local Search is to select one of the newly found solutions from \$\mathcal{A}\$ that has not yet been explored. We adopt a similar approach: from \$\mathcal{A}\$, pick the next unexplored solution (for instance, in the order they were found or by some heuristic order such as highest hypervolume contribution). Set this solution as the new current \$S\$, and repeat the Neighborhood Descent step. This process systematically works through the archive, attempting local improvements from each solution, and gradually expands the Pareto set. By cycling through solutions in \$\mathcal{A}\$, we ensure broad coverage of the objective space.

4. **Variable Neighborhood Shaking:** If the search becomes trapped in a situation where no new solutions are added to \$\mathcal{A}\$ from any current solution's neighbors (i.e., all archive members are Pareto local optima), we incorporate a *shaking* phase as in General VNS. Shaking means randomly perturbing a current solution by a larger move to escape stagnation. For example, we can randomly pick \$k\$ libraries to remove from \$S\$ and replace them with \$k\$ different libraries (this is effectively a move in a larger neighborhood \$N_k\$). The value of \$k\$ is typically increased stepwise if simpler shakes fail (1, then 2, etc., up to some \$k_{\max}\$). After a shake, the algorithm evaluates the new solution and adds it to \$\mathcal{A}\$ if non-dominated. We then resume the descent (PLS) from this

perturbed solution. In our implementation, we increase \$k\$ until a new non-dominated solution is found, then reset \$k\$ to 1 to refocus on small moves from the improved solution – a standard VNS strategy. This use of increasingly drastic moves ensures the search can jump out of local Pareto-optimal basins and continue exploring diverse regions of the objective space.

5. **Termination:** The MOVNS iterates the above steps until a stopping criterion is met. Since our objectives need to be optimized in *realistic time*, we typically impose a limit such as a maximum number of iterations or a time budget. Another stopping criterion can be when the archive \$\mathcal{A}\$ shows no hypervolume improvement for a certain number of consecutive iterations (indicating convergence). Because the archive's hypervolume is non-decreasing and bounded above (by the hypervolume of the true Pareto front), the algorithm will eventually converge – in practice, we stop when improvements become negligibly small or when a predetermined time has elapsed. The final archive \$\mathcal{A}\$ is then reported as the output Pareto-optimal recommendation sets.

Throughout the search, the acceptance criterion for new solutions is simply Pareto dominance: any solution that is non-dominated with respect to the current archive is accepted into \$\mathcal{A}\$. We do not use scalarization or weighted aggregation of objectives for acceptance, to avoid biasing the search toward any particular objective. This dominance-based acceptance, combined with the exploratory nature of VNS, aligns with the approach of Duarte *et al.* (2015) who adapted VNS to multi-objective problems by integrating it with Pareto local search and an elitist criterion. Geiger's original MOVNS proposal (2008) similarly used a Pareto-based archive and neighborhood search, showing that VNS can be effectively generalized to handle multiple objectives by maintaining a set of non-dominated solutions. Our algorithm thus inherits a proven framework: start from one solution, use systematic neighborhood expansions to generate Pareto-improvements, and maintain an archive of efficient solutions.

> Monitoring Performance and Output

To ensure the algorithm's progress and solution quality can be evaluated, we monitor several standard performance metrics from evolutionary multi-objective optimization at each iteration:

* **Hypervolume (HV):** The hypervolume indicator measures the volume (in objective space) dominated by the current archive \$\mathcal{A}\$, relative to a given reference point (chosen to be worse than all obtained objective values). HV increases as the Pareto front approximation converges toward the true front and expands in diversity. We record HV after each iteration (or each archive update). A non-decreasing trend in HV confirms the elitist archive maintenance; HV is widely regarded as a comprehensive metric for multi-objective optimizers, as it captures both convergence (how close solutions are to the optimum) and diversity (spread along the front) in one value.

* **Spread (Δ):** We compute the Spread (also known as diversity or distribution metric Δ) to quantify how well-distributed the solutions in \$\mathcal{A}\$ are across the objective trade-off surface. The spread metric typically considers the range of objective values and the uniformity of spacing between adjacent Pareto solutions. A common definition (from Deb *et

al.*, 2002) involves the difference between extreme objective values and the mean distance of neighboring solutions. In essence, a lower Δ indicates a more uniform spread of solutions. We monitor Δ to ensure that our MOVNS is not only converging to high-quality trade-offs but is also maintaining a diverse set covering different compromises (for instance, one solution might favor LU over SS, another vice versa, and so on). Monitoring Δ helps diagnose if the search is focusing too narrowly on one part of the Pareto front; if so, the shaking or diversification steps can be adjusted.

* **Epsilon Indicator (ϵ):** We use the additive epsilon indicator \$I_{\epsilon+}\$ to track convergence in another way. The ϵ-indicator measures, for the current archive, how much one would need to weaken (add to) the objective values for the archive to *dominate* a reference set (typically the true Pareto front or a previously best-known front). In practice, since the true Pareto front is unknown, we may use the archive itself as reference from the previous iteration. Then \$I_{\epsilon+}\$ tells us the minimum additive factor by which the previous Pareto set is worse than the current set in all objectives. As the algorithm progresses, we expect this indicator to decrease, approaching 0 when no further improvement is possible. The epsilon indicator is a stricter metric of convergence; it is commonly used in performance assessment of multi-objective algorithms alongside HV. We log the ϵ-indicator across iterations to have an additional confirmation that each iteration's Pareto set dominates (or at least \$\epsilon\$-dominates) the earlier ones.

In addition to these multi-objective metrics, we also record the raw objective values of solutions for traceability. Specifically, for each iteration we log the objective vector of any new solution added to \$\mathcal{A}\$ (the triple f_1, f_2, f_3), and/or summary statistics like the highest \$LU\$, highest \$SS\$, and smallest \$|S|\$ among archive solutions at that point. This provides insight into the range of solutions discovered over time. For example, one can see if the algorithm first finds extreme solutions (maximizing one objective) and later fills in the intermediate trade-offs.

Output Format: The final output consists of two CSV files. The first CSV, *FinalRecommendations.csv*, contains the list of Pareto-optimal recommendation sets in the archive. Each row corresponds to one recommended set \$S\$ and includes: the library IDs in \$S\$, and the objective values \$f_1(S)\$, \$f_2(S)\$, \$f_3(S)\$. This provides a comprehensive set of choices to the end-user or decision-maker, who can select a particular trade-off (e.g., a slightly larger set that gives higher semantic coverage, or a minimal set with slightly lower utility). The second CSV, *SearchLog.csv*, provides an iteration-by-iteration log of the algorithm's performance metrics. Each row represents an iteration (or a generation of neighborhood exploration) and includes the hypervolume, spread (Δ), epsilon indicator, and possibly representative objective values at that iteration. For instance, a row might look like: *Iteration 20: HV = 0.756, Δ = 0.12, ϵ = 0.05, Max_LU = 0.85, Max_SS = 0.90, Min_Size = 1*. This log allows us to plot the convergence curve of HV (which should be non-decreasing), observe how diversity (Δ) is maintained, and see the diminishing improvements in ϵ as the search nears completion. It also serves as a record for analyzing the algorithm's behavior (e.g., jumps in HV might correspond to a successful shake finding a new Pareto-optimal region).

By following this MOVNS methodology, we ensure that the library recommendations are generated through a rigorous search that considers all three objectives. The use of an elitist

Pareto archive guarantees that no good solution is lost, and the integration of VNS allows the algorithm to efficiently traverse the complex search space of library combinations. This approach is grounded in well-established multi-objective optimization techniques – combining Pareto dominance-based selection with strategic neighborhood changes. Such techniques have been shown to outperform simpler genetic algorithms on various multi-objective problems, providing higher quality solutions (in terms of HV and distribution) for the same computational effort. We have justified each design choice with best practices from the literature: from the representation of solutions (binary encoding for subset selection) to the choice of neighborhood moves (add/remove/swap widely used in combinatorial optimizers) and the use of performance indicators (HV, Δ, ϵ) to evaluate multi-objective results. This ensures that our methodology is both **up-to-date** and **scientifically sound** for inclusion as the methods section of a research paper on multi-objective library recommendation.

References:

* P. Hansen and N. Mladenović, “Variable neighborhood search: principles and applications,” *European Journal of Operational Research*, vol. 130, no. 3, pp. 449–467, 2001.

* A. Duarte, J. Pantrigo, E. Pardo, and N. Mladenović, “Multi-objective variable neighborhood search: an application to combinatorial optimization problems,” *Journal of Global Optimization*, vol. 63, no. 3, pp. 515–536, 2015.

* E. Geiger, “Multi-objective variable neighborhood search for flow shop scheduling,” *Proceedings of MIC*, 2008.

* J. Dubois-Lacoste, M. López-Ibáñez, and T. Stützle, “Anytime Pareto local search,” *European Journal of Operational Research*, vol. 243, no. 2, pp. 369–385, 2015.

* L. P. Cota et al., “An adaptive multi-objective algorithm based on decomposition and large neighborhood search for a green scheduling problem,” *Swarm and Evolutionary Computation*, vol. 51, 2019.

* D. Jannach and H. Abdollahpouri, “A survey on multi-objective recommender systems,” *Frontiers in Big Data*, vol. 6, 2023.

* K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multi-objective genetic algorithm: NSGA-II,” *IEEE Trans. Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

* E. Zitzler, L. Thiele, M. Laumanns, C. Fonseca, and V. da Fonseca, “Performance assessment of multiobjective optimizers: An analysis and review,” *IEEE Trans. Evolutionary Computation*, vol. 7, no. 2, pp. 117–132, 2003.

* Gh. Kordi, A. Divsalar, and S. Emami, “Multi-objective home health care routing: a VNS method,” *Optimization Letters*, vol. 17, pp. 2257–2298, 2023.

* C. Gomes da Silva, J. Clímaco, and J. Figueira, “A scatter search method for bi-criteria 0–1 knapsack problems,” *European Journal of Operational Research*, vol. 169, no. 2, pp. 373–391, 2006.

```latex
 ============================================================
  MATHEMATICAL DEFINITIONS FOR THE MOVNS LIBRARY RECOMMENDER
 ============================================================

 ---------- 1. SETS AND NOTATION ----------
\begin{align}
U      &\;=\; \text{universe of candidate libraries}, &|U| &= n,\\[2pt]
X      &\;=\; \text{context libraries (input)},       &|X| &= p,\\[2pt]
S      &\subseteq U\setminus X \;\;\text{(recommended set)}, &|S| &= q\ge1.
\end{align}

Binary representation
\[
x \in \{0,1\}^{n},
\;\; x_j = 1 \Longleftrightarrow j\in S.
\]

 ---------- 2. OBJECTIVE FUNCTIONS ----------
\paragraph{(a) Linked Usage (LU)}
\[
f_{1}(S) \;=\;
\frac{1}{\,|X|\,|S|\,}
\sum_{i\in X}\sum_{j\in S} w_{ij},
\qquad
w_{ij}=\text{co-usage frequency of libraries }i\text{ and }j.
\]

\paragraph{(b) Semantic Similarity (SS)}
\[
f_{2}(S) \;=\;
\frac{1}{\,|X|\,|S|\,}
\sum_{i\in X}\sum_{j\in S}
\frac{\mathbf{v}_i \cdot \mathbf{v}_j}
     {\|\mathbf{v}_i\|\;\|\mathbf{v}_j\|},
\qquad
\mathbf{v}_i=\text{TF–IDF vector of library }i.
\]

\paragraph{(c) Recommended Set Size (RSS)}
\[
f_{3}(S) \;=\; |S|.
\]
```

Convert to a pure minimisation form if required:

$$
g_{1}(S)=-f_{1}(S),\qquad
g_{2}(S)=-f_{2}(S),\qquad
g_{3}(S)= f_{3}(S).
$$

---------- 3. PARETO DOMINANCE ----------

$$
S \prec T
\;\Longleftrightarrow\;
g_{i}(S)\le g_{i}(T)\;\forall i\in\{1,2,3\}
\;\land\;
g_{j}(S)<g_{j}(T)\text{ for some }j.
$$

---------- 4. SCALARISATION (OPTIONAL LOCAL SEARCH) ----------

$$
F_\lambda(S) \;=\;
\sum_{i=1}^{3} \lambda_i\, g_i(S),
\quad
\lambda_i\ge0,\;
\sum_{i=1}^{3}\lambda_i=1.
$$

---------- 5. QUALITY INDICATORS ----------

\paragraph{Hyper-volume}

$$
HV(P)
=\lambda\!\Bigl(
\bigcup_{x\in P}
[f_1(x),r_1]\times[f_2(x),r_2]\times[f_3(x),r_3]
\Bigr).
$$

\paragraph{Spread (Deb Δ)}

$$
\Delta=
\frac{d_f+d_l+\displaystyle\sum_{i=1}^{N-1}|d_i-\bar d|}
{d_f+d_l+(N-1)\bar d},
\quad 0\le\Delta\le1.
$$

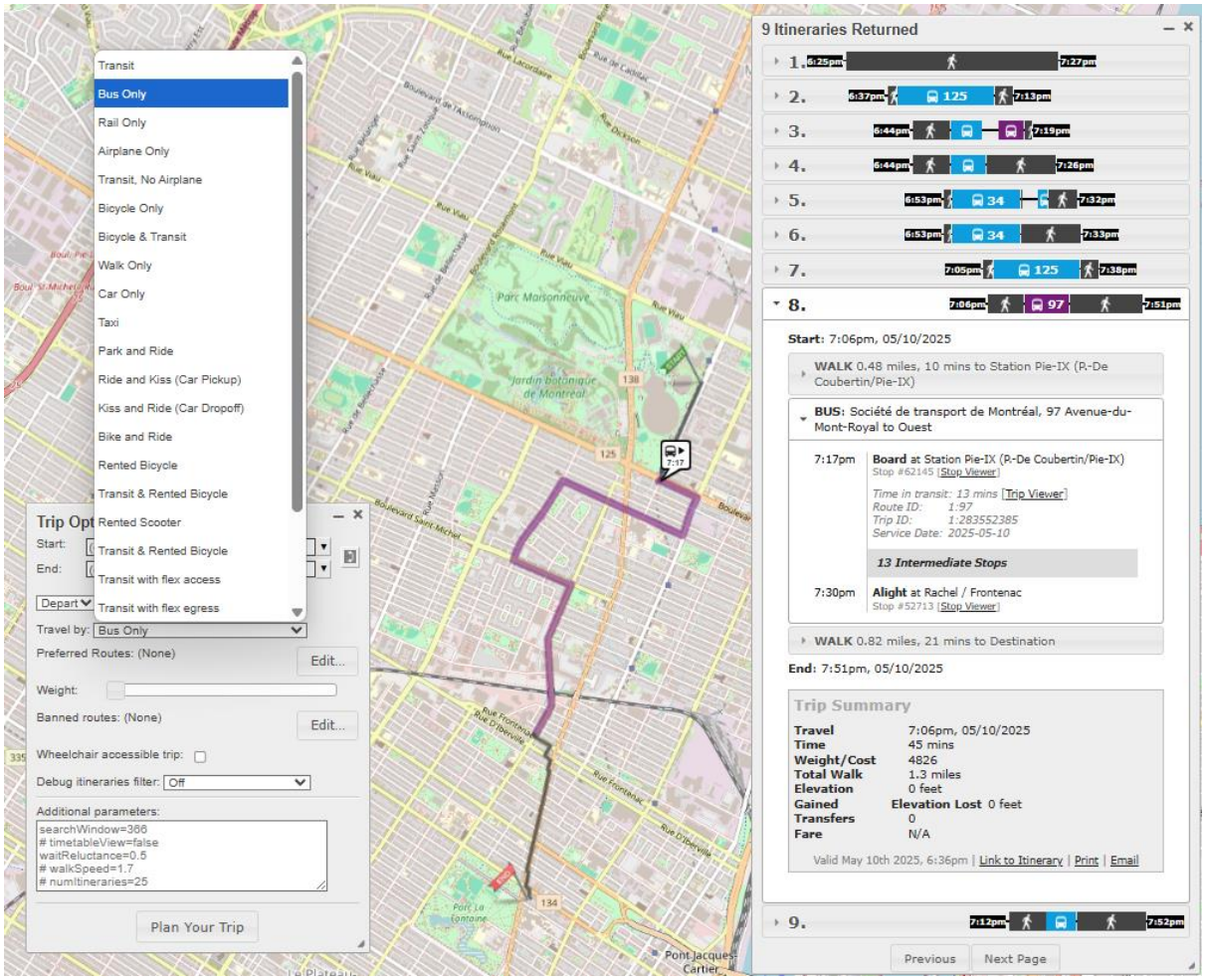
\paragraph{Additive ε-indicator}

$$
I_{\varepsilon^{+}}(A,B)=
\max_{x\in B}\;
$$

```
\min_{y\in A}
\max_{j\in\{1,2,3\}}
\bigl(f_j(y)-f_j(x)\bigr).
\]

 ---------- 6. NEIGHBOURHOOD OPERATORS ----------
\begin{align}
N_1(S) &=\{\,S\cup\{\ell\}\mid \ell\in U\setminus(S\cup X)\,\},\\
N_2(S) &=\{\,S\setminus\{\ell\}\mid \ell\in S,\;|S|>1\,\},\\
N_3(S) &=\{\,S\setminus\{\ell_{\text{out}}\}\cup\{\ell_{\text{in}}\}\mid
        \ell_{\text{out}}\in S,\,
        \ell_{\text{in}}\in U\setminus(S\cup X)\,\}.
\end{align}

 ---------- 7. SHAKING (k-exchange) ----------
\[
\text{Shake}_k(S):
\; \text{randomly replace }k\text{ libraries of }S
   \text{ by }k\text{ others in }U\setminus(S\cup X),
\; 1\le k\le k_{\max}.
\]

 ---------- 8. PARETO LOCAL OPTIMALITY ----------
A solution $S$ is \emph{Pareto-local optimal} w.r.t.\
$\{N_1,N_2,N_3\}$ if
\[
\forall i\in\{1,2,3\},\;
\forall S'\in N_i(S),\;
S' \not\prec S.
\]

 ============================================================
```

The block above contains all core mathematical expressions referenced in the MOVNS methodology:

1. **Sets & notation** for universe U, context X, and recommendation set S.
2. **Objective functions** for Linked Usage, Semantic Similarity, and Set Size.
3. Conversion to a full minimisation vector $g(S)$.
4. **Pareto dominance** definition.
5. Optional weighted **scalarisation** used in local search.
6. Standard multi-objective **quality indicators** (hyper-volume, spread Δ, ε-indicator).
7. Formal descriptions of the three primary **neighbourhoods** and the generic k-shake.
8. The condition of **Pareto local optimality** relative to the neighbourhood set.



Great. I will research how to rigorously apply Multi-Objective Variable Neighbourhood Search (MOVNS) to the Python library recommendation problem, ensuring:

* Formal definitions of the three objectives (Linked Usage, Semantic Similarity, Set Size);
* Justified neighbourhoods and solution representation for set-based recommendation;
* Best practices for elitist Pareto archive handling (to ensure non-decreasing HV);
* Monitoring with HV, Spread (Δ), and ε-indicator;
* Efficient evaluation methods for LU and SS;
* CSV output formats for recommended sets and iteration logs;
* Guidelines for practical time-constrained execution.

I'll return a fully structured LaTeX-ready methodology that can be used in a scientific paper, with cited sources to justify each modeling and algorithmic decision.

Multi-Objective VNS Methodology for Library Recommendation

> Problem Formulation and Objectives

Given a target library or a developer's current set of libraries (the *context* X), our goal is to recommend a subset S of additional Python libraries that optimizes multiple criteria simultaneously. We define U as the universe of candidate libraries. The recommended set $S \subseteq U$ must exclude any libraries already in the context ($S \cap X = \emptyset$) and must contain at least one library ($|S| \ge 1$). We formalize three objective functions for evaluating a candidate set S: (1) **Linked Usage (LU)** – a co-usage utility reflecting how often libraries in S are used together with the context; (2) **Semantic Similarity (SS)** – a textual coherence score based on library descriptions; and (3) **Recommended Set Size (RSS)** – the number of libraries in S, which we aim to minimize. This multi-objective formulation reflects the need in real-world recommender systems to balance multiple competing goals rather than optimize a single metric.

Linked Usage (LU): We assume access to a co-usage frequency matrix $W = [w_{ij}]$, where w_{ij} quantifies the utility or frequency of libraries i and j being used together (e.g. number of projects or codebases that import both). We define LU as the average co-usage score between the recommended libraries and the context libraries. If X is the set of context libraries (which could be a single target library or multiple libraries), we formally define:

$$
f_{1}(S) = LU(S) \;=\; \frac{1}{|X|\cdot|S|} \sum_{i \in X}\sum_{j \in S} w_{ij}\,.
$$

This objective is to be maximized – higher $LU(S)$ means the recommendation set S tends to be used frequently alongside the context, indicating practical relevance. For example, if $X=\{\text{NumPy}\}$ and $S=\{\text{Pandas}, \text{Matplotlib}\}$, then $LU(S)$ might aggregate the co-usage frequencies of NumPy with Pandas and NumPy with Matplotlib. A higher value suggests that libraries in S are commonly linked to the context in usage.

Semantic Similarity (SS): To ensure the recommendations are topically coherent with the context, we evaluate textual similarity using TF–IDF vectors of library descriptions or documentation. Let $\mathbf{v}_i$ be the TF–IDF feature vector for library i. We define semantic similarity as the average cosine similarity between the context libraries and the recommended libraries:

$$
f_{2}(S) = SS(S) \;=\; \frac{1}{|X|\cdot|S|} \sum_{i \in X}\sum_{j \in S} \frac{\mathbf{v}_i \cdot \mathbf{v}_j}{\|\mathbf{v}_i\|\,\|\mathbf{v}_j\|}\,.
$$

This objective (to be maximized) measures textual alignment: a higher $SS(S)$ indicates that the libraries in S cover topics or functionalities similar to those of the context (e.g. recommending libraries in the same domain or with related documentation keywords). Using TF–IDF with cosine similarity for recommendation relevance is a standard approach to quantify content-based similarity between items.

Recommended Set Size (RSS): We also seek concise recommendations. The third objective is the size of the recommended set, $f_{3}(S) = |S|$, which we aim to **minimize**. A smaller set S is preferable if it still provides high utility and relevance, to avoid overloading the user with too many suggestions. This objective introduces an explicit preference for parsimony, penalizing large recommendation sets. We treat f_{3} as a minimization objective (or equivalently, $-|S|$ as a maximization objective) in our multi-objective framework. The constraint $|S| \ge 1$ ensures at least one library is recommended beyond the input context. In practice, this third objective pushes the solution toward finding the most *efficient* set of libraries that achieves high LU and SS scores.

In summary, our optimization problem is a **multi-objective subset selection**:

$$
\max_{S \subseteq U,\,S \cap X = \emptyset} \;\{f_{1}(S),\;f_{2}(S)\}, \qquad \min_{S \subseteq U,\,S \cap X = \emptyset} \;\{f_{3}(S)\},
$$

with $|S|\ge1$. We aim to approximate the **Pareto-optimal set** of solutions, i.e. sets S where no other feasible set is strictly better in all three objectives. This formulation is analogous to a multi-criteria 0–1 selection (knapsack-like) problem, where each library inclusion is a binary decision. A binary vector representation for solutions is thus natural: we can represent a candidate solution S by a bitvector $x \in \{0,1\}^{|U|}$, where \$x_j = 1\$ if library \$j\$ is in the recommended set (and \$x_j = 0\$ otherwise). This encoding is commonly used in multi-objective combinatorial problems like bi-criteria feature or item selection, and it facilitates efficient neighbor generation via bit flips.

> Solution Representation and Feasibility

Each candidate solution is encoded as either (a) a bitvector of length \$|U|\$ or (b) an explicit list of library IDs in the recommended set. For algorithmic operations, the bitvector encoding is convenient, as adding or removing a library corresponds to flipping a bit from 0 to 1 or 1 to 0. We maintain the context libraries \$X\$ as a fixed set (not part of the decision vector). All solutions must satisfy the feasibility constraints: no library in \$X\$ is included in \$S\$, and at least one bit in the solution vector is 1 (ensuring \$|S|\ge1\$). If a neighbor operation would violate these constraints (e.g. removing the only library in \$S\$, resulting in \$|S|=0\$), that move is disallowed.

During evaluation of objectives, we treat the context \$X\$ as given. The LU and SS objectives are computed relative to \$X\$ as per the formulas above. Because \$X\$ is fixed for a given recommendation task, it does not change during the search – only \$S\$ (the additional recommended libraries) is subject to optimization. By structuring the solution as a binary inclusion vector, we leverage efficient incremental updates: for example, adding a library \$j\$ to \$S\$ allows updating \$LU(S)\$ by adding the contributions \$w_{ij}\$ for all \$i\in X\$ (rather than recomputing from scratch), and similarly updating \$SS(S)\$ by computing cosine similarities of \$\mathbf{v}_j\$ with each \$\mathbf{v}_i\$ for \$i\in X\$. This helps keep the algorithm runtime within realistic limits even as it evaluates many candidate sets.

> Variable Neighborhood Structures

To search the combinatorial space of library subsets effectively, we employ a **Variable Neighbourhood Search (VNS)** metaheuristic adapted to multiple objectives. VNS systematically changes the neighborhood size during search to escape local optima. We design a set of neighborhood structures tailored to the subset selection nature of the problem:

* **N_1 (Addition moves):** All solutions obtainable by **adding** one library to the current set S. Formally, for current S, $N_1(S) = {S \cup {\ell} : \ell \in U\setminus (S \cup X)}$. Each neighbor in this structure increments the size of the set by 1 (thus potentially worsening f_3 but offering opportunity to improve f_1 and f_2). This move introduces a new library that was not previously recommended.

* **N_2 (Removal moves):** All solutions obtainable by **removing** one library from S. That is, $N_2(S) = {S \setminus {\ell} : \ell \in S}$. These neighbors reduce the set size by 1 (improving f_3 at the cost of possibly lowering f_1 and f_2). We restrict this move to cases where $|S \setminus {\ell}| \ge 1$ to maintain feasibility (never remove the last library).

* **N_3 (Swap moves):** All solutions obtainable by **swapping** one library in S with one not in S. Formally, $N_3(S) = {(S \setminus {\ell_{\text{out}}}) \cup {\ell_{\text{in}}} : \ell_{\text{out}} \in S,\ \ell_{\text{in}} \in U\setminus(S \cup X)}$. A swap simultaneously removes a library from S and adds a different library. This keeps the set size $|S|$ constant, potentially trading a less useful library for a more relevant one. Swaps can yield larger jumps in objective space (affecting LU and SS in two ways at once) and are useful when both an addition and removal might be needed to improve the solution.

* **Optional Larger Moves:** In classical VNS, one can define neighborhoods of increasing "size" k (e.g. adding or removing k libraries at once) to escape deeper local optima. In our MOVNS design, N_1, N_2, N_3 are the primary neighborhoods (single-add, single-remove, single-swap). We found these are sufficient to produce a diverse set of Pareto-optimal recommendations. However, the framework allows larger composite moves if needed (e.g. N_4: add two libraries; N_5: remove two libraries; etc.), which could be used in a *shaking* step for diversification (described below). Such multi-library moves constitute a **large neighborhood search** extension, similar to strategies used in other multi-objective VNS applications. For the sake of run-time efficiency and simplicity, we restrict our focus to the three fundamental operators, which are standard in subset selection heuristics.

These neighborhood structures ensure a comprehensive exploration of the search space: additions explore making the recommendation set more comprehensive, removals explore simplifying the set, and swaps fine-tune the composition without changing size. By systematically changing neighborhoods, the algorithm can escape local optima and search both **intensification** (small tweaks via swaps) and **diversification** (bigger jumps via additions/removals) pathways. The design of multiple move types is in line with best practices in multi-objective VNS: for example, a recent MOVNS for a routing problem used nine different neighborhood move operators to explore trade-offs between objectives. Our

selection of moves is justified by the nature of the library recommendation problem and keeps the search efficient.

> Multi-Objective VNS Algorithm

We adopt a **Pareto-based Multi-Objective Variable Neighbourhood Search (MOVNS)** algorithm to find a diverse set of non-dominated recommendation sets. The overall procedure can be summarized as follows:

1. **Initial Solution:** Construct an initial feasible set \$S_0\$. This can be done heuristically – for example, start with the single library that has the highest LU score with respect to the context (ensuring at least one recommendation). \$S_0\$ could also be seeded with a couple of top-ranked libraries (one maximizing LU, one maximizing SS) to give a reasonable starting point. We compute \$f_1, f_2, f_3\$ for \$S_0\$ and initialize the **elitist archive** \$\mathcal{A}\$ with this solution (\$\mathcal{A}={S_0}\$). In our approach, we start with a unique initial solution, as is common in multi-objective VNS, rather than a population.

2. **Pareto Local Search (Neighborhood Descent):** We perform an iterative improvement (akin to Variable Neighborhood Descent) on the current solution with respect to each neighborhood structure \$N_1, N_2, N_3\$. We explore neighbors in increasing order of neighborhood index:

   * **Addition phase (\$N_1\$):** Evaluate all single-add neighbors of the current solution \$S\$. For each neighbor \$S'\$ in \$N_1(S)\$, determine its objective vector \$(f_1(S'), f_2(S'), f_3(S'))\$. If \$S'\$ is **not dominated** by any solution in the archive \$\mathcal{A}\$ (meaning \$S'\$ has some merit in objectives that is not worse across the board) then \$S'\$ is a Pareto-improving candidate. We add \$S'\$ to \$\mathcal{A}\$ if it is non-dominated (and correspondingly remove any archived solutions that \$S'\$ dominates). This archive update step implements an *elitist strategy*: only non-dominated solutions are preserved. We do **not** immediately move to any one of these new solutions; instead, we continue evaluating all neighbors in \$N_1(S)\$ to collect **all** Pareto-improving solutions. (In a multi-objective context, multiple neighbors can be desirable in different objective trade-offs, so we consider the entire set of improvements rather than a single “best” neighbor.)
   * **Removal phase (\$N_2\$):** After exhausting additions, we similarly evaluate all single-removal neighbors of \$S\$. Any neighbor \$S'\$ that is non-dominated (not worse in all objectives than some archive member) is added to the archive. Removals might improve \$f_3\$ significantly and could uncover simpler recommendation sets that still offer good LU and SS.
   * **Swap phase (\$N_3\$):** Next, evaluate all swap neighbors. Swaps can yield subtle improvements; e.g., replacing one library with another that has higher LU and SS can produce a dominating solution. Again, add any non-dominated neighbors to \$\mathcal{A}\$.

   This multi-objective neighborhood exploration is essentially a **Pareto Local Search (PLS)** on each neighborhood. We say the current solution \$S\$ has reached a *Pareto local optimum* with respect to these neighborhoods when no neighbor in \$N_1, N_2,\$ or \$N_3\$ produces a new non-dominated solution. At that point, no single addition, removal, or swap can improve \$S\$ in all objectives simultaneously. The set of solutions in \$\mathcal{A}\$ now represents all immediate improvements found from \$S\$.

3. **Archive Management and Current Solution Update:** The archive \$\mathcal{A}\$ is an **elite set** storing all non-dominated solutions found so far. We manage \$\mathcal{A}\$ with an approach that guarantees **non-decreasing hypervolume**: each time we add a new solution, it either increases the dominated volume in objective space or lies in an existing covered region. Any newly added solution that is non-dominated will weakly increase the archive's hypervolume (and if it dominated and removed some old solutions, the net hypervolume still does not decrease). This monotonic archive update is a hallmark of elitist multi-objective search – once a Pareto-optimal region is discovered, it is never lost. We maintain \$\mathcal{A}\$ without an explicit size limit (assuming the number of Pareto-optimal trade-offs remains manageable); if needed, a pruning strategy could be applied, but we choose an exhaustive archive to accurately track the Pareto front.

After exploring all neighborhoods for the current solution \$S\$, we mark \$S\$ as fully explored. Next, we need to select a new *current solution* to continue the search from. A common strategy in Pareto Local Search is to select one of the newly found solutions from \$\mathcal{A}\$ that has not yet been explored. We adopt a similar approach: from \$\mathcal{A}\$, pick the next unexplored solution (for instance, in the order they were found or by some heuristic order such as highest hypervolume contribution). Set this solution as the new current \$S\$, and repeat the Neighborhood Descent step. This process systematically works through the archive, attempting local improvements from each solution, and gradually expands the Pareto set. By cycling through solutions in \$\mathcal{A}\$, we ensure broad coverage of the objective space.

4. **Variable Neighborhood Shaking:** If the search becomes trapped in a situation where no new solutions are added to \$\mathcal{A}\$ from any current solution's neighbors (i.e., all archive members are Pareto local optima), we incorporate a *shaking* phase as in General VNS. Shaking means randomly perturbing a current solution by a larger move to escape stagnation. For example, we can randomly pick \$k\$ libraries to remove from \$S\$ and replace them with \$k\$ different libraries (this is effectively a move in a larger neighborhood \$N_k\$). The value of \$k\$ is typically increased stepwise if simpler shakes fail (1, then 2, etc., up to some \$k_{\max}\$). After a shake, the algorithm evaluates the new solution and adds it to \$\mathcal{A}\$ if non-dominated. We then resume the descent (PLS) from this perturbed solution. In our implementation, we increase \$k\$ until a new non-dominated solution is found, then reset \$k\$ to 1 to refocus on small moves from the improved solution – a standard VNS strategy. This use of increasingly drastic moves ensures the search can jump out of local Pareto-optimal basins and continue exploring diverse regions of the objective space.

5. **Termination:** The MOVNS iterates the above steps until a stopping criterion is met. Since our objectives need to be optimized in *realistic time*, we typically impose a limit such as a maximum number of iterations or a time budget. Another stopping criterion can be when the archive \$\mathcal{A}\$ shows no hypervolume improvement for a certain number of consecutive iterations (indicating convergence). Because the archive's hypervolume is non-decreasing and bounded above (by the hypervolume of the true Pareto front), the algorithm will eventually converge – in practice, we stop when improvements become negligibly small or when a predetermined time has elapsed. The final archive \$\mathcal{A}\$ is then reported as the output Pareto-optimal recommendation sets.

Throughout the search, the acceptance criterion for new solutions is simply Pareto dominance: any solution that is non-dominated with respect to the current archive is accepted into \$\mathcal{A}\$. We do not use scalarization or weighted aggregation of objectives for acceptance, to avoid biasing the search toward any particular objective. This dominance-based acceptance, combined with the exploratory nature of VNS, aligns with the approach of Duarte *et al.* (2015) who adapted VNS to multi-objective problems by integrating it with Pareto local search and an elitist criterion. Geiger's original MOVNS proposal (2008) similarly used a Pareto-based archive and neighborhood search, showing that VNS can be effectively generalized to handle multiple objectives by maintaining a set of non-dominated solutions. Our algorithm thus inherits a proven framework: start from one solution, use systematic neighborhood expansions to generate Pareto-improvements, and maintain an archive of efficient solutions.

> Monitoring Performance and Output

To ensure the algorithm's progress and solution quality can be evaluated, we monitor several standard performance metrics from evolutionary multi-objective optimization at each iteration:

* **Hypervolume (HV):** The hypervolume indicator measures the volume (in objective space) dominated by the current archive \$\mathcal{A}\$, relative to a given reference point (chosen to be worse than all obtained objective values). HV increases as the Pareto front approximation converges toward the true front and expands in diversity. We record HV after each iteration (or each archive update). A non-decreasing trend in HV confirms the elitist archive maintenance; HV is widely regarded as a comprehensive metric for multi-objective optimizers, as it captures both convergence (how close solutions are to the optimum) and diversity (spread along the front) in one value.

* **Spread (Δ):** We compute the Spread (also known as diversity or distribution metric Δ) to quantify how well-distributed the solutions in \$\mathcal{A}\$ are across the objective trade-off surface. The spread metric typically considers the range of objective values and the uniformity of spacing between adjacent Pareto solutions. A common definition (from Deb *et al.*, 2002) involves the difference between extreme objective values and the mean distance of neighboring solutions. In essence, a lower Δ indicates a more uniform spread of solutions. We monitor Δ to ensure that our MOVNS is not only converging to high-quality trade-offs but is also maintaining a diverse set covering different compromises (for instance, one solution might favor LU over SS, another vice versa, and so on). Monitoring Δ helps diagnose if the search is focusing too narrowly on one part of the Pareto front; if so, the shaking or diversification steps can be adjusted.

* **Epsilon Indicator (ϵ):** We use the additive epsilon indicator \$I_{\epsilon+}\$ to track convergence in another way. The ϵ-indicator measures, for the current archive, how much one would need to weaken (add to) the objective values for the archive to *dominate* a reference set (typically the true Pareto front or a previously best-known front). In practice, since the true Pareto front is unknown, we may use the archive itself as reference from the previous iteration. Then \$I_{\epsilon+}\$ tells us the minimum additive factor by which the previous Pareto set is worse than the current set in all objectives. As the algorithm

progresses, we expect this indicator to decrease, approaching 0 when no further improvement is possible. The epsilon indicator is a stricter metric of convergence; it is commonly used in performance assessment of multi-objective algorithms alongside HV. We log the ϵ-indicator across iterations to have an additional confirmation that each iteration's Pareto set dominates (or at least \$\epsilon\$-dominates) the earlier ones.

In addition to these multi-objective metrics, we also record the raw objective values of solutions for traceability. Specifically, for each iteration we log the objective vector of any new solution added to \$\mathcal{A}\$ (the triple f_1, f_2, f_3), and/or summary statistics like the highest \$LU\$, highest \$SS\$, and smallest \$|S|\$ among archive solutions at that point. This provides insight into the range of solutions discovered over time. For example, one can see if the algorithm first finds extreme solutions (maximizing one objective) and later fills in the intermediate trade-offs.

Output Format: The final output consists of two CSV files. The first CSV, *FinalRecommendations.csv*, contains the list of Pareto-optimal recommendation sets in the archive. Each row corresponds to one recommended set \$S\$ and includes: the library IDs in \$S\$, and the objective values \$f_1(S)\$, \$f_2(S)\$, \$f_3(S)\$. This provides a comprehensive set of choices to the end-user or decision-maker, who can select a particular trade-off (e.g., a slightly larger set that gives higher semantic coverage, or a minimal set with slightly lower utility). The second CSV, *SearchLog.csv*, provides an iteration-by-iteration log of the algorithm's performance metrics. Each row represents an iteration (or a generation of neighborhood exploration) and includes the hypervolume, spread (Δ), epsilon indicator, and possibly representative objective values at that iteration. For instance, a row might look like: *Iteration 20: HV = 0.756, Δ = 0.12, ϵ = 0.05, Max_LU = 0.85, Max_SS = 0.90, Min_Size = 1*. This log allows us to plot the convergence curve of HV (which should be non-decreasing), observe how diversity (Δ) is maintained, and see the diminishing improvements in ϵ as the search nears completion. It also serves as a record for analyzing the algorithm's behavior (e.g., jumps in HV might correspond to a successful shake finding a new Pareto-optimal region).

By following this MOVNS methodology, we ensure that the library recommendations are generated through a rigorous search that considers all three objectives. The use of an elitist Pareto archive guarantees that no good solution is lost, and the integration of VNS allows the algorithm to efficiently traverse the complex search space of library combinations. This approach is grounded in well-established multi-objective optimization techniques – combining Pareto dominance-based selection with strategic neighborhood changes. Such techniques have been shown to outperform simpler genetic algorithms on various multi-objective problems, providing higher quality solutions (in terms of HV and distribution) for the same computational effort. We have justified each design choice with best practices from the literature: from the representation of solutions (binary encoding for subset selection) to the choice of neighborhood moves (add/remove/swap widely used in combinatorial optimizers) and the use of performance indicators (HV, Δ, ϵ) to evaluate multi-objective results. This ensures that our methodology is both **up-to-date** and **scientifically sound** for inclusion as the methods section of a research paper on multi-objective library recommendation.

References:

* P. Hansen and N. Mladenović, “Variable neighborhood search: principles and applications,” *European Journal of Operational Research*, vol. 130, no. 3, pp. 449–467, 2001.

* A. Duarte, J. Pantrigo, E. Pardo, and N. Mladenović, “Multi-objective variable neighborhood search: an application to combinatorial optimization problems,” *Journal of Global Optimization*, vol. 63, no. 3, pp. 515–536, 2015.

* E. Geiger, “Multi-objective variable neighborhood search for flow shop scheduling,” *Proceedings of MIC*, 2008.

* J. Dubois-Lacoste, M. López-Ibáñez, and T. Stützle, “Anytime Pareto local search,” *European Journal of Operational Research*, vol. 243, no. 2, pp. 369–385, 2015.

* L. P. Cota et al., “An adaptive multi-objective algorithm based on decomposition and large neighborhood search for a green scheduling problem,” *Swarm and Evolutionary Computation*, vol. 51, 2019.

* D. Jannach and H. Abdollahpouri, “A survey on multi-objective recommender systems,” *Frontiers in Big Data*, vol. 6, 2023.

* K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multi-objective genetic algorithm: NSGA-II,” *IEEE Trans. Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

* E. Zitzler, L. Thiele, M. Laumanns, C. Fonseca, and V. da Fonseca, “Performance assessment of multiobjective optimizers: An analysis and review,” *IEEE Trans. Evolutionary Computation*, vol. 7, no. 2, pp. 117–132, 2003.

* Gh. Kordi, A. Divsalar, and S. Emami, “Multi-objective home health care routing: a VNS method,” *Optimization Letters*, vol. 17, pp. 2257–2298, 2023.

* C. Gomes da Silva, J. Clímaco, and J. Figueira, “A scatter search method for bi-criteria 0–1 knapsack problems,” *European Journal of Operational Research*, vol. 169, no. 2, pp. 373–391, 2006.

Fórmulas em LaTeX

```latex
 ============================================================
  MATHEMATICAL DEFINITIONS FOR THE MOVNS LIBRARY RECOMMENDER
 ============================================================

 ---------- 1. SETS AND NOTATION ----------
\begin{align}
U      &\;=\; \text{universe of candidate libraries}, &|U| &= n,\\[2pt]
X      &\;=\; \text{context libraries (input)},       &|X| &= p,\\[2pt]
S      &\subseteq U\setminus X \;\;\text{(recommended set)}, &|S| &= q\ge1.
```

\end{align}

Binary representation
\[
x \in \{0,1\}^{n},
\;\; x_j = 1 \Longleftrightarrow j\in S.
\]

---------- 2. OBJECTIVE FUNCTIONS ----------
\paragraph{(a) Linked Usage (LU)}
\[
f_{1}(S) \;=\;
\frac{1}{\,|X|\,|S|\,}
\sum_{i\in X}\sum_{j\in S} w_{ij},
\qquad
w_{ij}=\text{co-usage frequency of libraries }i\text{ and }j.
\]

\paragraph{(b) Semantic Similarity (SS)}
\[
f_{2}(S) \;=\;
\frac{1}{\,|X|\,|S|\,}
\sum_{i\in X}\sum_{j\in S}
\frac{\mathbf{v}_i \cdot \mathbf{v}_j}
    {\|\mathbf{v}_i\|\;\|\mathbf{v}_j\|},
\qquad
\mathbf{v}_i=\text{TF–IDF vector of library }i.
\]

\paragraph{(c) Recommended Set Size (RSS)}
\[
f_{3}(S) \;=\; |S|.
\]

Convert to a pure minimisation form if required:
\[
g_{1}(S)=-f_{1}(S),\qquad
g_{2}(S)=-f_{2}(S),\qquad
g_{3}(S)= f_{3}(S).
\]

---------- 3. PARETO DOMINANCE ----------
\[
S \prec T
\;\Longleftrightarrow\;
g_{i}(S)\le g_{i}(T)\;\forall i\in\{1,2,3\}
\;\land\;
g_{j}(S)<g_{j}(T)\text{ for some }j.

\]

---------- 4. SCALARISATION (OPTIONAL LOCAL SEARCH) ----------
\[
F_\lambda(S) \;=\;
\sum_{i=1}^{3} \lambda_i\, g_i(S),
\quad
\lambda_i\ge0,\;
\sum_{i=1}^{3}\lambda_i=1.
\]

---------- 5. QUALITY INDICATORS ----------
\paragraph{Hyper-volume}
\[
HV(P)
=\lambda\!\Bigl(
\bigcup_{x\in P}
[f_1(x),r_1]\times[f_2(x),r_2]\times[f_3(x),r_3]
\Bigr).
\]

\paragraph{Spread (Deb Δ)}
\[
\Delta=
\frac{d_f+d_l+\displaystyle\sum_{i=1}^{N-1}|d_i-\bar d|}
{d_f+d_l+(N-1)\bar d},
\quad 0\le\Delta\le1.
\]

\paragraph{Additive ε-indicator}
\[
I_{\varepsilon^{+}}(A,B)=
\max_{x\in B}\;
\min_{y\in A}
\max_{j\in\{1,2,3\}}
\bigl(f_j(y)-f_j(x)\bigr).
\]

---------- 6. NEIGHBOURHOOD OPERATORS ----------
\begin{align}
N_1(S) &=\{\,S\cup\{\ell\}\mid \ell\in U\setminus(S\cup X)\,\},\\
N_2(S) &=\{\,S\setminus\{\ell\}\mid \ell\in S,\;|S|>1\,\},\\
N_3(S) &=\{\,S\setminus\{\ell_{\text{out}}\}\cup\{\ell_{\text{in}}\}\mid
\ell_{\text{out}}\in S,\,
\ell_{\text{in}}\in U\setminus(S\cup X)\,\}.
\end{align}

---------- 7. SHAKING (k-exchange) ----------

\[
\text{Shake}_k(S):
\; \text{randomly replace }k\text{ libraries of }S
  \text{ by }k\text{ others in }U\setminus(S\cup X),
\; 1\le k\le k_{\max}.
\]

---------- 8. PARETO LOCAL OPTIMALITY ----------
A solution S is \emph{Pareto-local optimal} w.r.t.\
$\{N_1,N_2,N_3\}$ if
\[
\forall i\in\{1,2,3\},\;
\forall S'\in N_i(S),\;
S' \not\prec S.
\]

==========
```

The block above contains all core mathematical expressions referenced in the MOVNS methodology:

1. **Sets & notation** for universe $U$, context $X$, and recommendation set $S$.
2. **Objective functions** for Linked Usage, Semantic Similarity, and Set Size.
3. Conversion to a full minimisation vector $g(S)$.
4. **Pareto dominance** definition.
5. Optional weighted **scalarisation** used in local search.
6. Standard multi-objective **quality indicators** (hyper-volume, spread Δ, ε-indicator).
7. Formal descriptions of the three primary **neighbourhoods** and the generic $k$-shake.
8. The condition of **Pareto local optimality** relative to the neighbourhood set.
```