Šablona závěrečné práce
studenta Unicorn Vysoká škola

Tato první stránka šablony není součástí bakalářské práce.

Tato šablona slouží jako vzorová šablona závěrečných prací studentům Unicorn Vysoké školy. Závěrečná práce musí obsahovat všechny náležitosti uvedené v této šabloně.

Nesplnění této podmínky může být považováno za důvod pro nepřipuštění závěrečné práce k obhajobě (nebo případně k vrácení práce od obhajoby k přepracování).

Další informace a pokyny k vypracování závěrečná práce naleznete na webových stránkách. Vše potřebné se také dozvíte v rámci předmětu Bakalářský seminář.

Při zpracování této šablony bylo použito písmo Cambria, 11pt. pro text a písmo Calibri pro nadpisy.

Vzor: ***PEVNÁ DESKA*** *závěrečné práce* ***není součástí elektronické verze*** **UNICORN VYSOKÁ ŠKOLA S.R.O.**

BAKALÁŘSKÁ/DIPLOMOVÁ PRÁCE (vyberte jednu možnost)

Rok Jméno a PŘÍJMENÍ autora ***(Jan NOVÁK)***

Vzor: ***TITULNÍ STRANA*** *závěrečné práce*

UNICORN VYSOKÁ ŠKOLA S.R.O.

Softwarové inženýrství a big data

DIPLOMOVÁ PRÁCE (vyberte jednu možnost)

Název práce (přesně podle zadání)

Autor BP: Jméno a příjmení autora/autorky (Jan Novák)

Vedoucí BP: Jméno a příjmení vedoucí/vedoucího práce i s tituly (prof. Ing. Jan Čadil, Ph.D.)

Vzor: ***ZADÁNÍ ZÁVĚREČNÉ PRÁCE*** *– originál, kopie originálu, naskenovaná podoba – dle jednotlivých forem (originál, 2 x kopie, elektronická verze)*

Vzor: ***ČESTNÉ PROHLÁŠENÍ*** *– prohlášení o samostatném vypracování závěrečné práce, datum a vlastnoruční podpis (v každém výtisku práce)*

Čestné prohlášení

Prohlašuji, že jsem svou bakalářskou práci na téma ..... vypracoval/a samostatně pod vedením vedoucího bakalářské práce a s použitím výhradně odborné literatury a dalších informačních zdrojů, které jsou v práci všechny citovány a jsou také uvedeny v seznamu použitých zdrojů.

Jako autor/ka této bakalářské práce dále prohlašuji, že v souvislosti s jejím vytvořením jsem neporušil/a autorská práva třetích osob a jsem si plně vědom/a následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb. Prohlašuji, že nástroje umělé inteligence byly využity pouze po podpůrné činnosti a v souladu s principem akademické etiky.

Dále prohlašuji, že odevzdaná tištěná verze bakalářské práce je shodná s verzí, která byla odevzdána elektronicky.

V..... dne ....

(Jan Novák)

Vzor: ***PODĚKOVÁNÍ*** *vedoucímu BP, konzultantům, odborníků, spolupracovníkům za poskytnuté rady a podkladové materiály apod.) –* ***není povinné***

Poděkování

Např: Děkuji vedoucímu bakalářské práce Jméno Příjmení (i s tituly) za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce...

Vzor: ***PRVNÍ ČÍSLOVANÁ STRANA*** *– číslice na první číslované straně se určí podle počtu předchozích stran, počínaje Titulní stranou, tzn. že pokud jsou řazené všechny dané strany – Titulní strana, Zadání (2 strany), Čestné prohlášení a Poděkování – je první číslovaná strana stranou 6.*

Název práce v českém/slovenském jazyce Název práce v anglickém jazyce

Vzor: ABSTRAKT A KLÍČOVÁ SLOVA

Abstrakt

Abstrakt česky. Abstrakt krátce a výstižně charakterizuje obsah závěrečné práce. Zpravidla obsahuje informace o stanovených cílech, použitých metodách, postupu řešení a výsledcích výzkumu. Může obsahovat krátkou informaci o použitých zdrojích. Délka abstraktu je zpravidla 100–500 slov.

Klíčová slova: klíčová slova práce, minimálně 5, maximálně 10

Abstract

Zde umístěte překlad abstraktu do anglického jazyka. Česky a anglicky psané abstrakty musí být totožné. Student/ka zodpovídá za jazykovou správnost anglického překladu. V případě, že se anglická a česká verze nevejdou na jednu stránku, umístěte celý překlad na samostatnou stránku.

Keywords: klíčová slova v anglickém jazyce

Vzor: ***OBSAH*** *– hierarchické uspořádání číslovaných názvů kapitol a podkapitol, včetně všech příloh, spolu s čísly jejich stran. Dále se uvádí Seznam obrázků/tabulek/grafů. Pozn.: počet a názvy kapitol samozřejmě odpovídají charakteru konkrétní práce.*

Obsah

Vzor: ***ÚVOD*** *(cca 5-10 normostran)*

Úvod

Vehicle Routing Problem (VRP)

Introduction to the Vehicle Routing Problem

The Vehicle Routing Problem (VRP) occupies a central place in operational research and logistics due to its direct applicability to optimizing the distribution of goods and services. First formulated by Dantzig and Ramser in 1959, the VRP has evolved significantly, encompassing various complex forms tailored to different industrial needs and constraints.

Historical Background and Significance

The VRP, initially introduced as the "Truck Dispatching Problem" by Dantzig and Ramser (1959), seeks to determine the most efficient routes for a fleet of delivery vehicles operating from a central depot. This problem is foundational in the field of combinatorial optimization and remains pivotal in supply chain management and logistics.

Problem Definition

At its core, the Vehicle Routing Problem (VRP) involves designing optimal routes for a fleet of vehicles that must deliver goods or services to a set of customers and return to the depot, minimizing total travel costs while satisfying various constraints. This classic problem not only focuses on reducing operational expenses but also on improving service efficiency and customer satisfaction.

Objective: The primary objective of the VRP is to determine a set of routes that minimizes the total cost of transportation. Costs typically include distance traveled, time spent, fuel consumption, and sometimes additional factors such as toll charges or vehicle maintenance. The total cost must be minimized while ensuring that all customer demands are met and all operational constraints are adhered to.

Constraints: Several constraints must be considered in the VRP:

- **Capacity Constraints**: Each vehicle has a maximum carrying capacity that must not be exceeded by the sum of the demands of the customers it serves. This ensures that the vehicle can physically carry the load assigned to it without overloading.
- **Route Length Constraints**: Often, there is a maximum allowable route length or time duration for each vehicle's tour, which might be dictated by legal driving hours, fuel limitations, or service level agreements.
- **Customer Constraints**: Each customer must be visited exactly once by one vehicle. In cases involving time windows, each customer must be serviced within a specific time frame.

- **Depot Constraints**: All routes must start and end at the depot, ensuring the cyclic nature of vehicle tours.

Formulation: Mathematically, the VRP can be represented using a graph $G = (V, E)$, where V is the set of vertices (nodes) including the depot and customers, and E is the set of edges (arcs) representing possible routes between vertices. Each edge $(i, j) \in E$ is associated with a cost c_{ij}, which typically represents the distance or travel time between vertices i and j.

The problem is often formulated as an integer programming model where decision variables indicate whether a route between two vertices is included in the solution. Let x_{ij} be a binary decision variable that equals 1 if edge (i, j) is included in the route and 0 otherwise. The objective function is to minimize the total cost associated with the selected routes, subject to various constraints.

$$\min \sum_{(i,j)\in E} c_{ij} x_{ij} \tag{1}$$

The objective function aims to minimize the total travel cost. The formulation includes several constraints to ensure the solution is feasible:

Degree Constraints: Each customer must be visited exactly once, which can be represented as:

$$\sum_{j\in V, j\neq i} x_{ij} = 1 \quad \forall i \in V \setminus \{0\} \tag{2}$$

$$\sum_{i\in V, i\neq j} x_{ij} = 1 \quad \forall j \in V \setminus \{0\} \tag{3}$$

These constraints ensure that each customer node i has exactly one incoming and one outgoing edge, forming a valid route.

Capacity Constraints: Each vehicle has a limited capacity Q. Let q_i be the demand at customer i. The total demand serviced by a vehicle on its route must not exceed its capacity. This can be formulated using flow variables u_i representing the load of the vehicle after visiting customer i:

$$u_i - u_j + Q x_{ij} \leq Q - q_j \quad \forall (i, j) \in E, i \neq j, i \neq 0 \tag{4}$$

$$q_i \leq u_i \leq Q \quad \forall i \in V \setminus \{0\} \tag{5}$$

Here, u_i is set to 0 at the depot.

Subtour Elimination Constraints: To prevent the formation of subtours (cycles that do not include the depot), we use subtour elimination constraints. For example, in the Miller-Tucker-Zemlin (MTZ) formulation:

$$u_i - u_j + Qx_{ij} \leq Q - q_i \quad \forall (i,j) \in E, i \neq j, i \neq 0 \tag{6}$$

$$q_i \leq u_i \leq Q \quad \forall i \in V \setminus \{0\} \tag{7}$$

Depot Constraints: All routes must start and end at the depot:

$$\sum_{j \in V \setminus \{0\}} x_{0j} = m \tag{8}$$

$$\sum_{i \in V \setminus \{0\}} x_{i0} = m \tag{9}$$

where m is the number of vehicles available.

Flow Conservation Constraints: These constraints ensure the flow conservation at each node, meaning the number of vehicles entering a customer node must equal the number leaving it:

$$\sum_{j \in V \setminus \{i\}} x_{ij} = \sum_{j \in V \setminus \{i\}} x_{ji} \quad \forall i \in V \setminus \{0\} \tag{10}$$

This comprehensive formulation ensures that all aspects of the VRP are covered, including minimizing travel costs, adhering to vehicle capacities, servicing all customers exactly once, starting and ending routes at the depot, and preventing subtours. The problem is complex due to the combination of these constraints, which necessitates advanced optimization techniques for finding feasible and near-optimal solutions in practical applications.

Applications: The VRP is highly relevant across various industries, including logistics, supply chain management, waste collection, and public transportation. Effective solutions to VRP can lead to significant cost savings, improved service levels, and enhanced operational efficiency.

Research and Development: The study of VRP has led to significant advancements in optimization theory and practice. Researchers have developed a wide range of solution methodologies, from classical operations research techniques to modern metaheuristics and machine learning approaches. Continuous improvement in computational power and algorithmic strategies promises ongoing advancements in solving VRP more efficiently and effectively.

This comprehensive overview of the VRP problem definition provides a solid foundation for understanding its complexities and importance, setting the stage for detailed exploration of its variants and solution methods in the subsequent sections.

Complexity and Relevance

The VRP is known to be NP-hard, implying that no efficient algorithm can solve all instances optimally in polynomial time. This complexity necessitates sophisticated solution approaches, both exact and heuristic, to find feasible and near-optimal solutions efficiently.

Variants and Extensions

There are numerous variants of the VRP, each introducing additional layers of complexity to address real-world logistics challenges. These variants reflect the diverse operational requirements and constraints encountered in practical applications. Key variants include:

- **Capacitated VRP (CVRP)**: The CVRP is the most widely studied variant of the VRP, focusing on the distribution of goods from a single depot to multiple customers using a fleet of vehicles with fixed capacities. Each vehicle must deliver goods without exceeding its capacity, and the objective is to minimize the total cost, typically the travel distance or time. This variant is fundamental in understanding more complex VRP forms and is used extensively in both academic research and practical applications.
- **VRP with Time Windows (VRPTW)**: The VRPTW adds the complexity of time constraints to the basic VRP. Each customer must be serviced within a specified time window, adding another layer of complexity to the routing problem. This variant is crucial for industries where delivery times are critical, such as perishable goods distribution and courier services. The goal is to minimize the total travel cost while ensuring that all deliveries or pickups occur within their designated time frames. Time windows make the problem significantly harder to solve, and specialized algorithms, often involving sophisticated scheduling techniques, are required.
- **Dynamic VRP (DVRP)**: In the DVRP, the problem parameters, such as customer requests or traffic conditions, change dynamically over time. This variant reflects real-world situations where information is not static and decisions must be adapted in real-time. For example, new customer orders might arrive throughout the day, or unexpected traffic jams might alter the planned routes. Solving the DVRP requires algorithms that can efficiently update solutions in response to new information, often leveraging real-time data and adaptive heuristics.

- **Stochastic VRP (SVRP)**: The SVRP deals with uncertainty in problem data, such as variable customer demand or unpredictable travel times. This variant introduces probabilistic elements into the routing problem, requiring solutions that are robust to variability and can handle unexpected changes efficiently. Approaches to solving the SVRP often involve stochastic modeling and optimization techniques that consider multiple scenarios or employ probabilistic constraints.
- **Green VRP**: The Green VRP focuses on minimizing environmental impacts, such as fuel consumption and CO2 emissions, in addition to traditional cost metrics. This variant is increasingly important due to growing environmental regulations and the need for sustainable logistics practices. Green VRP models incorporate factors like vehicle emissions, fuel efficiency, and alternative energy sources into the optimization process. Methods to address Green VRP include eco-friendly routing algorithms and multi-objective optimization techniques.

Methodological Approaches To tackle the VRP, both exact and heuristic methods have been developed, each with its strengths and weaknesses depending on the problem size and specific requirements.

Exact Methods: Exact methods guarantee finding an optimal solution by exhaustively exploring the solution space. These methods include:

- **Integer Programming (IP)**: IP formulations, such as the one presented earlier, model the VRP using binary variables and linear constraints. Solving IPs can yield optimal solutions, but the computational effort increases exponentially with problem size. Advanced techniques like Branch-and-Cut algorithms are often employed to handle larger instances by incorporating cutting planes to reduce the feasible region iteratively [**?**].
- **Branch and Bound (B&B)**: This method systematically explores branches of a tree representing subsets of the solution space. By calculating bounds on the best possible solution within each subset, B&B can prune large portions of the tree, thus reducing the number of solutions that need to be examined [**?**].
- **Dynamic Programming (DP)**: DP breaks down the VRP into simpler subproblems and solves them recursively. While highly effective for smaller instances, the state space grows exponentially with the number of customers, limiting its practicality for large-scale problems.

Heuristic and Metaheuristic Methods

Heuristic and metaheuristic methods provide near-optimal solutions more efficiently, making them suitable for larger instances where exact methods are impractical. These methods do not guarantee optimality but are designed to produce good quality solutions within reasonable computational times.

- **Genetic Algorithms (GA):** Inspired by natural selection, GAs evolve a population of solutions using selection, crossover, and mutation to improve solution quality.
- **Simulated Annealing (SA):** A probabilistic technique that mimics the annealing process, allowing occasional worse solutions to escape local optima.
- **Tabu Search (TS):** Enhances local search by using a memory structure to avoid revisiting recently explored solutions, promoting thorough exploration.
- **Ant Colony Optimization (ACO):** Inspired by ant foraging behavior, ACO uses pheromone trails to guide the search for optimal paths.
- **Particle Swarm Optimization (PSO):** Simulates the social behavior of bird flocks or fish schools, where solutions (particles) move through the solution space based on their own and their neighbors' experiences.

A detailed discussion of these methods will be provided in a later chapter.

Conclusion and Outlook

As industries continue to seek improvements in operational efficiencies and cost reductions, the relevance of the VRP is expected to grow. Advances in computational power, data analytics, and artificial intelligence are opening new avenues for research and application, making the VRP a vibrant and ever-evolving field of study.

This introduction provides a foundation for the comprehensive exploration of various aspects and complexities of the VRP, setting the stage for detailed discussions on its many variants and solving techniques in subsequent sections.

Overview of Heuristic and Meta-heuristic Optimization Algorithms

In the fields of computational science and mathematical programming, a heuristic algorithm is a method used to find a good enough solution when it's not feasible to find the perfect solution. These algorithms focus on working quickly and efficiently rather than achieving flawless accuracy or the best possible result. They are especially valuable in situations where solving the problem completely and precisely would require too much computing power, or when the details of the problem are not fully known.

Characteristics of Heuristic Algorithms:

1. **Approximation**: Heuristics do not guarantee that the solutions they provide will be optimal. Instead, they aim to deliver "good enough" solutions that are acceptable within practical constraints, such as limited processing time or resources.
2. **Efficiency**: By not exhaustively searching every possible solution, heuristic algorithms can provide solutions much faster than their exact counterparts. This makes them especially valuable in dealing with large datasets or complex problem spaces where an exact approach would be computationally infeasible.
3. **Adaptability**: Heuristic algorithms are often tailored to the specific characteristics of a problem, leveraging problem-specific insights and techniques to guide the search process towards more promising areas of the solution space.

Common Types of Heuristic Algorithms

Greedy Algorithms

Greedy algorithms are a fundamental strategy in problem-solving, known for their focus on immediate benefits at each step. In every decision-making moment, a greedy algorithm selects the best immediate option, hoping that these short-term optimal choices will result in the best overall solution. While not always perfect, greedy algorithms are highly effective for many optimization problems due to their simplicity, speed, and often excellent results.

The main feature of a greedy algorithm is the "greedy choice property," which means you can build an optimal solution step by step by making the best choice at each moment, without needing to rethink previous decisions. Once a decision is made, a greedy algorithm adheres to it, trusting that these decisions will lead to the best end result.

Problems suited for greedy algorithms often exhibit an "optimal substructure," meaning that the best solution contains within it the best solutions to smaller problems. For instance, finding the shortest path in a network can be seen as finding the shortest paths to points along the way. This characteristic allows

greedy algorithms to develop solutions progressively, focusing on the best choice at each step.

Application Areas Greedy algorithms are used across various fields due to their simplicity and efficiency. Some of the prominent application areas include:

1. **Scheduling:** Greedy algorithms are highly effective in scheduling tasks where immediate, optimal decisions can lead to efficient overall schedules. They are used to organize activities, sequence jobs, and manage time intervals. For instance, in job scheduling on machines, a greedy approach can be employed to minimize the total completion time by always assigning the next job to the machine that becomes available first. Additionally, in event scheduling, greedy algorithms help in assigning time slots to events to avoid conflicts, ensuring that no two events overlap. These algorithms are also used in task prioritization, where tasks are ordered based on deadlines or importance, allowing for efficient management of resources and time.

2. **Graph Algorithms:** Greedy strategies are fundamental in graph theory, with several key applications such as finding the shortest paths in networks, constructing minimum spanning trees, and ensuring efficient traversal of graphs. These applications are critical in network routing, geographical mapping, and various optimization problems where optimal paths or connections are required.

3. **Data Compression:** Greedy algorithms play a significant role in data compression techniques, optimizing storage and transmission efficiency. Huffman coding, a classic greedy algorithm, generates variable-length prefix codes based on the frequency of each data symbol. By assigning shorter codes to more frequent symbols and longer codes to less frequent symbols, Huffman coding minimizes the overall length of the encoded data. This approach is widely used in file compression formats such as ZIP and GZIP, as well as in multimedia codecs for compressing images, audio, and video. The efficiency of greedy algorithms in reducing data size without losing information makes them indispensable in modern data storage and communication.

4. **Resource Allocation:** In scenarios where resources need to be allocated efficiently, greedy algorithms are employed to make optimal choices at each step. This includes tasks such as assigning tasks to workers, distributing goods among locations, and allocating bandwidth in communication networks. By continually selecting the most beneficial allocation at each step, these algorithms help in achieving near-optimal utilization of resources.

5. **Financial Modeling:** Greedy algorithms are used in various financial models and decision-making processes. For example, in portfolio optimization, a greedy approach can be used to iteratively add assets to a portfolio based on their expected returns until the investment budget is

exhausted. Similarly, in real-time bidding for online advertising, greedy algorithms help in making immediate bidding decisions to maximize the return on investment.

6. **Route Planning:** In logistics and transportation, greedy algorithms are applied to route planning to minimize travel time or distance. This includes delivery route optimization, vehicle routing problems, and urban transit planning. By making locally optimal choices at each stage, these algorithms help in devising efficient routes that save time and fuel costs.

7. **Machine Learning:** In the field of machine learning, greedy algorithms are used in feature selection and decision tree construction. By iteratively selecting the most relevant features or the best split at each node of a decision tree, these algorithms help in building efficient and accurate models for classification and regression tasks.

Examples of Greedy Algorithms

1. **Dijkstra's Algorithm:** This algorithm finds the shortest path from a source node to all other nodes in a weighted graph. At each step, it selects the node with the smallest tentative distance from the source and updates the distances of its neighbors. The algorithm uses the following update rule:
$$d(v) = \min\{d(v), d(u) + w(u, v)\} \quad \text{for all } (u, v) \in E$$
where $d(v)$ is the tentative distance of vertex v, and $w(u, v)$ is the weight of the edge (u, v).

2. **Prim's Algorithm:** Similar to Kruskal's algorithm, Prim's algorithm constructs a minimum spanning tree. It starts with an arbitrary node and repeatedly adds the nearest node that hasn't been included in the tree yet. The inclusion criterion is:
$$\text{Include } v \text{ such that } (u, v) \text{ is the smallest edge connecting } T \text{ to } V \setminus T$$
where T is the set of nodes already included in the tree.

3. **Kruskal's Algorithm:** This algorithm constructs a minimum spanning tree for a connected weighted graph. It repeatedly selects the edge with the smallest weight that doesn't form a cycle with the previously selected edges. The algorithm uses the union-find structure to manage cycles.

4. **Huffman Coding:** This algorithm builds a variable-length prefix code for lossless data compression. It constructs a binary tree based on symbol frequencies, assigning shorter codes to more frequent symbols and longer codes to less frequent symbols. The cost function minimized by Huffman coding is:
$$\sum_{i=1}^{n} f_i \cdot l_i$$

where f_i is the frequency of symbol i and l_i is the length of the code assigned to symbol i.

5. **Nearest Neighbor Algorithm:** Commonly used in the traveling salesman problem (TSP), this algorithm selects the nearest unvisited city from the current city and adds it to the tour. The selection criterion is:

$$\text{Select } j \text{ such that } d(i,j) \text{ is minimum for all } j \in V \setminus \{i\}$$

where $d(i,j)$ is the distance between cities i and j.

Mathematical Formulation for Vehicle Routing Problems (VRP) In the context of VRPs, greedy algorithms can be used to construct initial solutions that can be further refined using more sophisticated methods. For instance, the Clarke-Wright Savings Algorithm is a greedy heuristic commonly used for VRPs. It starts by calculating the savings for combining routes and then iteratively merges routes with the highest savings.

Let s_{ij} denote the savings of combining routes i and j, calculated as:

$$s_{ij} = c_{i0} + c_{0j} - c_{ij}$$

where c_{i0} is the cost from the depot to customer i, c_{0j} is the cost from the depot to customer j, and c_{ij} is the cost between customers i and j. The algorithm merges routes with the highest s_{ij} until no further savings can be achieved.

Conclusion Greedy algorithms provide a straightforward approach to tackling optimization problems by making the locally optimal choice at each step. While they may not always yield the globally optimal solution, their efficiency and simplicity make them valuable tools, particularly as a starting point for more complex optimization procedures.

Local Search Algorithms

Local search algorithms are a class of heuristic methods used to solve optimization problems where the solution space is too large to explore exhaustively. These algorithms start with an initial solution and iteratively make small adjustments or "moves" to improve it. The process continues until no further improvements can be found or other termination conditions are met. Unlike global optimization methods, local search algorithms focus on finding a better solution in the neighborhood of the current solution rather than exploring the solution space as a whole.

The main characteristic of a local search algorithm is its focus on iterative improvement. By repeatedly making small changes to a solution and only accepting changes that improve the solution (or maintain the same level in some variations), local search can effectively refine a solution to a near-optimal state. However, a common challenge with these algorithms is their tendency to get

stuck in local optima—points in the solution space where no nearby solutions are better.

Local search algorithms are suitable for problems with a well-defined neighborhood structure, meaning there is a clear way to define small changes to a solution. These algorithms are often employed in areas like:

1. Combinatorial Optimization: Tasks such as vehicle routing, scheduling, and packing problems.
2. Machine Learning: Parameter tuning in models where small parameter adjustments can lead to improved predictions.
3. Engineering Design: Optimization of design parameters within given constraints to achieve optimal performance.

Examples of Local Search Algorithms:

1. **Hill Climbing**: This is a straightforward approach where the algorithm examines the neighboring solutions and moves to the neighbor with the highest value, repeating this process until no improvement can be made.
2. **Tabu Search**: Enhances the basic local search by using a memory structure that records recent moves or solutions and forbids or discourages revisiting them. This helps to avoid cycles and encourages exploration of new areas of the solution space.
3. **Variable Neighborhood Search (VNS)**: This algorithm systematically changes the neighborhood structure as it searches, helping to avoid local optima by shifting the search to different areas of the solution space.

Metaheuristic Algorithms

Metaheuristic algorithms represent a sophisticated class of heuristic methods that are designed to solve complex optimization problems where traditional methods are inefficient or infeasible. The term "metaheuristic" combines "meta," meaning beyond or at a higher level, and "heuristic," referring to a trial-and-error method for discovering solutions. These algorithms are known for their capability to guide and improve simpler heuristics, aiming to produce solutions that are superior to those typically found by pursuing local optimality alone.

Characteristics of Metaheuristic Algorithms:

- **Guidance of Search Process:** Metaheuristics provide high-level strategies that significantly influence the search process, making them effective at exploring complex solution spaces.
- **Exploration of Solution Space:** They are designed to efficiently navigate through vast solution spaces to find near-optimal solutions by balancing exploration and exploitation strategies.

- **Technique Variety:** The techniques used in metaheuristics range from simple local search procedures to more complex adaptive and learning processes.
- **Approximation and Non-determinism:** These algorithms are approximate and generally non-deterministic, often incorporating stochastic elements that enhance solution diversity.
- **Problem Agnosticism:** Unlike problem-specific algorithms, metaheuristics are versatile and can be applied to a wide range of problems without significant modifications.

Functionality and Application: Metaheuristics function as master strategies that adapt and modify existing heuristic approaches by incorporating local search and randomization to tackle optimization challenges effectively. This adaptability allows them to produce excellent solutions within a reasonable time frame, even for complex issues such as NP-hard problems or scenarios with limited or imperfect information.

Despite their strengths, it is important to note that metaheuristics do not guarantee the discovery of the optimal solution. Their effectiveness is often evidenced through empirical results from extensive computer simulations, although theoretical insights regarding their convergence and the potential to reach global optima are also available. However, the field is mixed with high-quality research alongside studies that may suffer from vagueness and poor experimental design.

Examples of Metaheuristic Algorithms:

1. **Simulated Annealing:** Utilizes a cooling schedule to probabilistically accept worse solutions, facilitating escape from local optima.
2. **Genetic Algorithms:** Mimic natural evolutionary processes such as selection, mutation, and crossover to evolve solutions over generations.
3. **Ant Colony Optimization:** Inspired by the foraging behavior of ants, this algorithm uses pheromone trails as a form of indirect communication to find optimal paths.
4. **Particle Swarm Optimization:** Based on social behavior patterns observed in flocks of birds or schools of fish, where the collective movement is adjusted according to the successful experiences of individuals.
5. **Tabu Search:** Employs a memory structure to keep track of the search history, helping to avoid revisiting previously explored solutions and encouraging the exploration of new areas.

Methodology for Evaluating Algorithms

Introduction to Algorithm Evaluation

Evaluating algorithms for the Vehicle Routing Problem (VRP) and its variants is crucial for understanding their efficiency, scalability, and practical applicability. This section outlines the methodologies and metrics commonly used to assess the performance of these algorithms.

Algorithm evaluation involves a combination of theoretical analysis and empirical testing. Theoretical analysis provides insights into the computational complexity and expected behavior of algorithms under various conditions, while empirical testing offers practical performance data based on real-world or simulated instances.

Goals of Algorithm Evaluation The primary goals of algorithm evaluation are to:

- **Assess Efficiency**: Determine how efficiently an algorithm utilizes computational resources such as time and memory.
- **Ensure Robustness**: Evaluate how well an algorithm performs under various conditions and with different data sets.
- **Measure Scalability**: Understand how the algorithm's performance scales with increasing problem size and complexity.
- **Validate Accuracy**: Ensure that the algorithm produces correct and reliable results.
- **Compare Alternatives**: Provide a basis for comparing different algorithms to select the most suitable one for a given problem.

Evaluation Criteria

Evaluating algorithms for the Vehicle Routing Problem (VRP) and its variants, such as the Capacitated Vehicle Routing Problem (CVRP), requires a comprehensive set of criteria to ensure a thorough assessment. The following criteria, derived from the principles outlined in the *Encyclopedia of Machine Learning and Data Mining* by Claude Sammut and Geoffrey I. Webb, are essential for evaluating these algorithms:

Time Complexity Time complexity refers to the amount of computational time an algorithm requires as a function of the input size. For VRP and CVRP, it is crucial to assess both the worst-case and average-case time complexities to understand the algorithm's efficiency. Algorithms with lower time complexities are generally preferred, especially for large-scale instances.

Space Complexity Space complexity measures the amount of memory an algorithm uses relative to the input size. Efficient use of memory is vital for handling large datasets typically encountered in VRP scenarios. Space complexity analysis helps in identifying algorithms that can process data without exhausting system memory resources.

Accuracy Accuracy is a measure of how close the algorithm's solution is to the optimal solution. In the context of VRP and CVRP, accuracy can be assessed by comparing the total route cost or distance with the known optimal values or best-known solutions. High accuracy is critical for ensuring cost-effective and reliable routing plans.

Robustness Robustness evaluates the algorithm's ability to perform well under various conditions, including changes in data, unexpected inputs, and different problem sizes. An algorithm that maintains high performance despite variations in input conditions is considered robust. This criterion is particularly important for real-world applications where data variability is common.

Scalability Scalability assesses how well an algorithm handles increasing problem sizes. For VRP and CVRP, scalability is essential because these problems often involve a large number of customers and vehicles. Algorithms that can scale efficiently without a significant increase in computational resources are highly desirable.

Flexibility Flexibility refers to the algorithm's ability to adapt to different variants of the VRP, such as VRP with Time Windows (VRPTW), Dynamic VRP (DVRP), and Stochastic VRP (SVRP). An algorithm that can be easily modified or extended to handle different constraints and objectives demonstrates high flexibility.

Computational Cost Computational cost involves both time and space complexities but also includes other resource expenditures such as energy consumption and processing power. Evaluating computational cost helps in understanding the practical feasibility of deploying an algorithm in real-world settings where resources may be limited.

Implementation Complexity Implementation complexity considers the ease of coding and integrating the algorithm into existing systems. Algorithms that are simpler to implement and require fewer lines of code are often more attractive for practical applications, even if they are slightly less efficient.

Interpretability Interpretability measures how easily the results produced by the algorithm can be understood and used by decision-makers. For VRP and CVRP, this involves generating solutions that are not only optimal but also easy to comprehend and justify to stakeholders.

Reliability Reliability evaluates the consistency of the algorithm's performance. An algorithm that consistently produces high-quality solutions and does not fail under different scenarios is considered reliable.

These evaluation criteria provide a holistic framework for assessing VRP and CVRP algorithms. In the next subchapter, we will delve into the **Experimental Design**, outlining the methodologies for setting up experiments to rigorously test and evaluate these algorithms.

Praktická část/Empirická část/Vlastní práce (není název kapitoly)

Autor/autorka uvedou vlastní název kapitoly vztahující se ke konkrétnímu tématu práce

Nadpis úrovně 2

*Obrázek se v textu značí následujícím způsobem: samotný obrázek se označí: „**Obrázek 1: Název obrázku**“ (11 nebo 12 pt, černě, tučně). Obrázky se označují názvem a číslováním nad obrázkem a zdrojovým dokumentem pod obrázkem, příp. informace o vlastním zpracování (11 nebo 12 pt, černě). K popisování doporučujeme využít nástroje textového editoru, který usnadní generování seznamu obrázků na konci práce.*

Obrázek 1: Logo

Zdroj: https://unicornuniversity.net/cs/

Obrázek 2: Obrázek jednorožce

Zdroj: Vlastní zpracování

Tabulky se označují názvem a číslováním nad tabulkou a zdrojovým textem pod tabulkou. Tabulky, obrázky a grafy se číslují zvlášť. Každá tabulka, obrázek nebo graf MUSÍ být v textu okomentován. Je nepřípustné, aby jednotlivé kapitoly (podkapitoly) tvořilo pouze grafické znázornění v podobě tabulek, grafů, obrázků, schémat atp. bez jejich okomentování.

Tabulka 1: Statistika vět zachovaných a vyřazených filtr. kritériem *FK1*

Sada	Celkem	Zachováno	Vyřazeno	Zachováno
dtest	5228	2384	2844	45,6 %
etest	5476	2419	3057	44,2 %
train-1	4709	2204	2505	46,8 %

Zdroj: Vlastní zpracování

Matematické rovnice, vzorce

Pokud jsou v práci rovnice, nezapomeňte je správně číslovat. Pro jejich zápis používejte MS Editor rovnic, případně jinou obdobnou aplikaci. Rovnice by měla vypadat například takto (nezapomeňte proměnné popisovat):

$$S = \pi r^2, \tag{1}$$

kde S je obsah kruhu o poloměru r .

Závěr

Rozsah je zpravidla 5-10 normostran.

Seznam použitých zdrojů

V seznamu zdrojů musí být uvedeny všechny v závěrečné práci citované zdroje. Zároveň nesmí seznam obsahovat zdroje, které nejsou v závěrečné práci použity.

Používáme citační normu ČNS ISO 690. Doporučujeme pro tvorbu citací některý z citačních nástrojů, které jsou v základní verzi zpravidla zdarma dostupné.

Seznam obrázků (existují-li)

Seznam tabulek (existují-li)

Seznam grafů (existují-li)

Seznam příloh (existují-li)

Každá příloha musí být alespoň jednou odkázána do vlastního textu práce. Přílohy se číslují. Každá příloha začíná na nové stránce.

Příloha A – Název přílohy

Příloha B – Název přílohy