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

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 key type of strategy used in problem-solving, known for focusing on immediate benefits at each step of the process. In each decision-making moment, a greedy algorithm picks 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." This 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 sticks to it, trusting that these decisions will lead to the best end result.

Problems suited for greedy algorithms often have an "optimal substructure," meaning a 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.

Greedy algorithms are used across various fields, such as:

1. Scheduling: They are useful in organizing activities, sequencing jobs, and managing time intervals.
2. Graph Algorithms: Examples include Dijkstra's algorithm for shortest paths and Kruskal's and Prim's algorithms for creating minimum spanning trees.
3. Data Compression: Huffman coding is a greedy method that creates efficient codes for data based on frequency, optimizing storage use.

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.
2. **Prim's Algorithm**: Similar to Kruskal's algorithm, Prim's algorithm also 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.
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.
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.
5. **Nearest Neighbor Algorithm**: 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.

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.

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