Object Oriented Programming 2023/24

Optimal patrol allocation of planetary systems using Evolutionary Programming

MEEC – IST

1 Problem

The empire has a set of n patrols $P = \{p_1, \ldots, p_n\}$ responsible for maintaining order in m planetary systems $A = \{a_1, \ldots, a_m\}$ to counter the movements of the rebels. Each patrol p_i requires $c_{ij} \in \mathbb{N}^+$ units of time to pacify the system a_j. The time it takes for patrols to move from one planetary system to another is considered negligible (as the patrol ships travel through hyperspace).

The goal is to find an allocation of the planetary systems among the empire's various patrols that minimizes the time required to pacify the entire empire. This is due to the emperor's concern that if an area is not quickly patrolled, the rebels become too powerful to control. An allocation of the planetary systems is essentially a partition of $A = A_1 \cup \ldots \cup A_n$ where A_i corresponds to the planetary systems patrolled by p_i. The time t_i that a patrol p_i spends monitoring the systems in A_i is calculated as follows:

$$t_i = \sum_{j : a_j \in A_i} c_{ij}.$$

The objective is to find a distribution of the planetary systems $A = A_1 \cup \ldots \cup A_n$ that minimizes the total patrolling time t, computed as:

$$t = \max_{1 \leq i \leq n} t_i.$$

For instance, assume the empire has three patrols $P = \{p_1, p_2, p_3\}$ and controls 6 planetary systems $A = \{a_1, a_2, a_3, a_4, a_5, a_6\}$. The time required for each system to be patrolled by each patrol is represented in the following 3×6 matrix:

$$C = \begin{pmatrix} 1 & 2 & 1 & 1 & 2 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}.$$

The objective is to identify an allocation of the planetary systems $A = A_1 \cup A_2 \cup A_3$ that minimizes the total time t required for patrolling the entire empire. In this scenario, one possible allocation is $A_1 = \{a_1, a_3, a_4, a_6\}$, $A_2 = \{a_2, a_5\}$, and $A_3 = \emptyset$, for which $t = \max\{1 + 1 + 1 + 1, 2 + 2, 0\} = 4$. It should be noted that there are alternative allocations that can patrol the empire within the same timeframe.

1.1 Police patrol as an open-shop scheduling problem

The Open Shop Scheduling Problem (OSSP) is a complex optimization challenge that arises in operations research and computer science, particularly within the context of manufacturing and production planning. The goal of the OSSP is to assign various tasks to machines or workstations, where each task requires processing on specific machines for certain durations and can be processed on any machine at any time, without a predefined sequence. The primary objective is to minimize the total completion time, known as the *makespan*, ensuring that no machine processes more than one task at a time and no task is processed by more than one machine simultaneously.

This problem is a notable example of NP-hardness, a classification in computational complexity theory that encompasses decision problems for which a solution can be verified in polynomial time, but no known algorithm can solve all instances of the problem in polynomial time. Specifically, the OSSP belongs to the subset of NP-complete problems, indicating that it is at least as hard as the most difficult problems in NP and that a polynomial-time algorithm solving one NP-complete problem could solve all NP problems.

Despite the relative simplicity of finding a solution for small instances of the OSSP, as in the given example, no polynomial-time algorithm is known for solving it in general. This lack of a polynomial-time solution is not just a minor inconvenience; it reflects a profound question in computer science related to the P vs NP problem, one of the seven Millennium Prize Problems. A solution to this problem, either by finding a polynomial-time algorithm for NP-complete problems or proving that no such algorithm exists, carries a reward of one million U.S. dollars.

Given the computational infeasibility of solving large instances of the OSSP exactly – where even a hypothetical 10GHz computer would take approximately 1200 centuries to find an optimal solution for a problem size of $n \times m = 75$ – researchers and practitioners often turn to heuristic and approximate methods. Among these, evolutionary programming stands out as a particularly effective approach.

Heuristic and metaheuristic approaches that can be applied to the OSSP include:

- Evolutionary Algorithms: Simulates the process of natural selection, wherein species that can adapt to environmental changes are able to survive and reproduce, thereby passing on successful adaptations to the next generation and progressively moving closer to an optimal solution. A relevant case of evolutionary algorithms is genetic algorithms, where the reproduction of individuals allows crossing over.
- Ant Colony Optimization: Mimics the foraging behavior of ants to find optimal paths through the solution space. This algorithm is based on the pheromone trail laid down by ants, which is used by other ants to find the path to food.
- Simulated Annealing: A probabilistic technique that explores the solution space by accepting both improvements and, with decreasing probability over time, certain degradations in solution quality. This method is inspired by the annealing process in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects.
- Tabu Search: Utilizes memory structures to avoid revisiting previously explored solutions, thereby encouraging the exploration of new areas of the solution space. This approach is particularly useful for solving optimization problems by guiding the search process away from less promising regions.

While these methods do not guarantee finding the optimal solution, especially for large and complex instances of the OSSP, they often yield sufficiently good solutions within a reasonable amount of computational time, making them practical for real-world applications.

1.2 Solving the OSSP through evolutionary programming

Evolutionary programming, inspired by the principles of biological evolution, employs iterative processes of mutation, selection, and reproduction to enhance a population of candidate solutions. In the context of challenges like the OSSP, evolutionary programming is adept at generating a variety of solutions across successive generations. This approach methodically advances towards more optimal schedules by effectively navigating through an extensive search space, a task that is considerably more efficient than methods based on exhaustive search.

One of the primary challenges in evolutionary programming is identifying the most fitting individual – or solution – within a potentially vast and complex search space. This difficulty arises from several factors:

- Vast Search Space: The search space in many problems, including the OSSP, can be exponentially large, making it impractical to evaluate every possible solution directly. Evolutionary programming aims to explore this space efficiently, but finding the absolute best solution without exhaustive search requires intelligent navigation strategies.
- Premature Convergence: There is a risk of the algorithm converging too early on suboptimal solutions. This happens when the population loses diversity too quickly, leading to stagnation where further iterations do not significantly improve the solutions.
- Balance between Exploration and Exploitation: Achieving the right balance between exploring new areas of the search space (exploration) and refining the best solutions found so far (exploitation) is critical. Too much exploration can lead to inefficiency and a lack of focus, while too much exploitation can cause premature convergence on suboptimal solutions.
- Evaluation Complexity: The fitness evaluation, which determines how well a solution meets the problem criteria, can be computationally expensive for complex problems. This makes the process of identifying the best-fitted individual time-consuming, especially when the population size is large or the number of generations is high.

1.3 Stochastic simulation

Stochastic simulation is a computational method used to model systems that have some degree of uncertainty or randomness. Unlike deterministic simulations, where the same set of initial conditions always produces the same result, stochastic simulations incorporate random variables and probabilistic behaviors to account for variability in outcomes. This approach is particularly valuable in analyzing and predicting the behavior of complex systems where uncertainty is inherent or where exact predictions are impossible due to the randomness involved.

Combining evolutionary programming with stochastic simulation offers a powerful approach to solving complex optimization and modeling problems. This synergy leverages the strengths of both methodologies, providing several advantages, such as handling uncertainty and variability, efficient exploration of complex search spaces, and improved solution quality and robustness.

Please refer to the slides from the lectures for the stochastic simulation of a toll highway.

2 Approach

This project aims to program a solution to the problem presented above in Java using evolutive programming modeled and implemented with objects.

2.1 Individual and population

The idea is to generate at instant zero a **population** of ν individuals, and make it evolve until the final instant τ. An **individual** is a distribution of the planetary systems among the patrols of the empire.

For individuals of the initial population the planetary systems are distributed randomly and uniformly among the patrols. The evolution of an individual modify its distribution in order to find one that minimizes the time to police the whole empire.

2.2 Comfort and best fitted individual

To each individual z it is associated a **comfort** $\varphi(z)$ given by:

$$\varphi(z) = \frac{t_{\min}}{t_z}$$

where:

- t_z is the time to patrol the empire given by the individual z;
- $t_{\min}$ it is a lower bound for the optimal time to patrol the empire, given by:

$$t_{\min} = \frac{\sum_{j=1}^{m} \min_{1 \le i \le n} c_{ij}}{n}.$$

Note that the comfort is a real value in (0,1] and that the most fitted individuals have a comfort value close to one or even one. The individual with greater comfort is termed as the **best fitted individual** (in the event of existing more than one individual under these circumstances it is considered one at random).

2.3 Random laws

Each individual z evolve according to its comfort, by the following random mechanisms:

- **Death**, exponential variable with mean value $(1 - \log(1 - \varphi(z)))\mu$ between events.
- **Reproduction**, exponential variable with mean value $(1 - \log(\varphi(z)))\rho$ between events. From reproduction, a new individual is born with a distribution of the planetary systems among the patrols equal to its parent up to $\lfloor (1 - \varphi(z))m \rfloor$ systems, i.e., the distribution of the planetary systems in the new individual is computed in the following way:
  (i) initially, it is given a distribution equal to the parent;
  (ii) from that initial distribution it is randomly and uniformly removed $\lfloor (1 - \varphi(z))m \rfloor$ planetary systems from the various patrols; and
  (iii) the planetary systems previously removed are redistributed randomly and uniformly among the patrols.

- **Mutation**, exponential variable with mean value $(1 - \log(\varphi(z)))\delta$ between events. The mutation of an individual removes randomly and uniformly a planetary system from one patrol p_i and places it randomly and uniformely into another patrol p_j, with $i \neq j$.

2.4 Epidemics

The population evolves in function of the individual evolution of its elements and also by the occurrence of **epidemics**. Whenever the number of individuals exceed a maximum ν_{max}, an epidemic occurs. To the epidemic will always survive the five individuals with greater comfort. For each of the remaining, the survival probability is $\frac{2}{3}\varphi(z)$.

2.5 Evolution

The evolution of the population is ruled by discrete stochastic simulation, that is, based on a pending event container. The simulation ends in the following situations:

- The time for the next event occurs after the final time τ.
- The population became extinct and there are no events to simulate.
- An individual of comfort one is found.