Searching for Optimisation

Maastricht University

Searching for Optimisation

inaugural lecture

delivered on Wednesday 25 May 2022, by

Prof. dr. Frank Phillipson,

appointed as professor of 'computational operations research' at the school of business and economics

Abstract

Operations Research (OR) can be defined as a scientific approach to the solution of problems in the management of complex systems. It uses quantitative methods to help decision makers design, analyse, and improve the performance or operation of these systems. An important part of this, is defining and solving optimisation problems. One can only be interested in finding the best solution, but often (in practice) a combination of, or interaction between, the quality of the solution and the computation time is important. Model-based algorithms and heuristics have traditionally played a role in this, however, data-based methods such as machine learning are increasingly used. Hardware improvements are also not to be underestimated. A major improvement in the computing power of conventional computers has made problem solving methods more efficient over the past 40 years and is still ongoing. This is currently being reinforced by the use of GPU processors, the emergence of quantum computers and other technological solutions such as photonic accelerators. This playing field of OR problems and the use of machine learning and existing and emerging hardware accelerations to define and solve them is the topic of the field of 'Computational Operations Research', which is discussed further in this article.

Rector Magnificus, ladies and gentlemen,

1 Introduction

Operations Research (OR) can be defined[1] as a scientific approach to the solution of problems in the management of complex systems. It uses quantitative methods to help decision makers design, analyse, and improve the performance or operation of systems. Organisations may be looking for a very wide range of operational improvements, for example greater efficiency, better customer service, higher quality or lower costs. Consider, for example, parcel delivery. Many of these improvements are of interest here: how do we route the vehicles, not only minimising the kilometres driven but also increasing the chance of a successful delivery, what does the layout of a distribution point look like, how do you use a package wall efficiently? The goal of the OR professional is, together with clients, to find practical and pragmatic solutions for operational, but also tactical and strategic, problems, often within tight time limits.

The OR field was developed during and just after the Second World War. During this War, research was conducted in England and the United States into the efficiency of certain military operations. Many applications and methodologies were developed in this environment that still form the basis of OR. After World War II, OR grew in many other areas as well, as scientists learned to apply its principles to the civilian sector, such as manufacturing, logistics, and telecommunications. The development of the simplex linear programming algorithm in 1947 [10] and the development of computers over the next three decades, among other things, enabled OR to solve problems with more and more variables and constraints. In addition, the large amount of data required for such problems could be stored and manipulated efficiently. But actually, this development has never stopped and is now more timely than ever. Increasing computing power continues to open up new possibilities for us, both in the use of new techniques and in the size of the problems we can solve.

What kind of problems do we mean here? These are problems that can be formulated as decision or as optimisation problems. In the example of parcel delivery, the order of the delivery addresses has to be determined, such that the travelling distance is minimal. An answer or decision must be

[1]According to the Association of European Operational Research Societies.

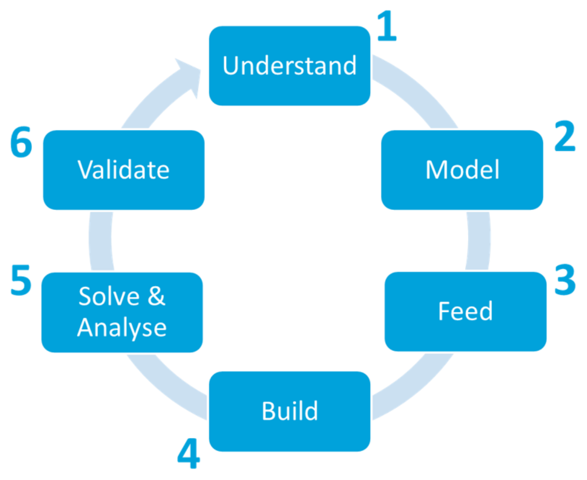

Figure 1: Standard approach to OR problems in 6 steps.

found, that is the best over all possible answers or solutions. Often there are many solutions possible. Another important element in these systems, that potentially makes the problem difficult, is whether there is uncertainty. If the environment is fixed and known, we speak of deterministic problems, where combinatorial optimisation is the subject of research: find the best of many possible solutions. Purely finding the optimal route of the parcel deliverer is an example of this. If there is uncertainty in parts of the problem, we speak of stochastic problems. In the example of parcel delivery, realise that we now also take the traffic congestion into account in the planning. Here, simulation, probability calculation and statistics play an important role in solving the problem. We classify all these problems and techniques under OR.

A standard approach to OR problems contains the following steps, see Figure 1 (based on [40]):

1. Understand.

2. Model.

3. Feed.

4. Build.

5. Solve & Analyse.

6. Validate.

The first two steps are extremely important within OR: understanding the problem and translating the most important features into a model. This often leads to interesting discussions with domain experts, who find all details equally important.

The third step is often the most time consuming. In many cases, obtaining, collecting and processing data takes by far the most time in a project. Classically, an attempt is made here to provide certain parameters in the model with a specific value or a probability distribution.

The sequence of these first three steps is a classic fact that is developing rapidly. More and more often, data is used directly at the start of this process, and understanding and modelling is seen as a (semi-)automated process, based on this data, where the model is created by artificial intelligence techniques, also called AI. This process is adaptive in time by adding new data. That is, when the data that is added to the model changes over time, the model structure will automatically also change. The data is now increasingly used directly to discover and (dynamically) record structure and (interactive) behaviour of and within the model [22, 45].

The fourth and fifth step are the favourite part of many (colleague) scientists. Developing the mathematical solution techniques, applying them to the cases and analysing the outcome is the daily work of many OR colleagues. The fourth step has traditionally received the most attention: the problem solving methods. For mathematicians often also the most challenging: how do you efficiently find the best or, usually, a good solution for a problem with many possible solutions or with uncertainty. Here too, methods from artificial intelligence play an important role. Or, perhaps it is actually true that many techniques that the OR has been using for years are now classified under AI.

The last step is the most forgotten step. Often one is very satisfied with the solution they have found and forget to see whether it actually works in reality. Also, it should not be forgotten to connect the different steps with each other. If the model or the data in the model contains uncertainties or does not contain all the details, then it makes no sense to have a problem solving methodology that takes a lot of time to arrive at the optimal answer. The optimal solution based on incomplete or incorrect data is not optimal. Often it still requires iteration steps, with input from human experts, to arrive at a good answer. The expert, or problem owner, then receives the answer from the model, sees from his experience where issues are, feeds them back to the model and asks for a new solution.

An important development within this whole process, is that the complexity of the underlying systems that need to be optimised is increasing, both in size and in mutual dependence, and that behaviour (of people) is increasingly an underlying phenomenon that is difficult to model. Also, the objectives of a system are not always unambiguous, which leads to multi-objective problems. Where, as an example, you previously simply minimised costs, whereby a lower limit was set on the performance, we are now increasingly interested in (all) balanced combinations in which trade-offs between costs and performance for multiple underlying systems are balanced, the so-called Pareto frontiers.

This complexity forces the OR professional to make choices: Can this problem still be modelled? Should I cut it into pieces? Should I let an AI system model it? Can it still be calculated in a reasonable time?

In this article I discuss the observation that OR has traditionally encompassed analytical tools from many different disciplines and is still expanding. The development of the possibilities in problem solving methods is closely related to the development of computers that make calculations possible. Both hardware developments (better processors, CPU, GPU, quantum computing), and software developments (solvers, AI, machine learning), have peaked and are partly intertwined with developments in OR. In order to explain the scope of the field of Computational Operation Research, the research directions and the ambitions, I will further discuss the three building blocks: (1) the methodologies and the underlying building blocks of these methodologies, (2) the hardware and its development and (3) the measurement of efficiency of the whole. In the last section, these field will come together to define the field of research.

2 Measuring Efficiency

First, we discuss the efficiency aspect: how do you determine the efficiency of a problem solving methodology, including the underlying hardware? Suppose you have a problem that you want to solve. Solving here means finding the solution to the problem. For problems that are formulated mathematically, it often involves finding an algorithm ($\mathcal{A}$), a recipe or step-by-step plan to use for the problem ($\mathcal{P}$) to find the (best) solution ($\mathcal{S}^*$):

$$\mathcal{P} \mapsto \mathcal{A} \mapsto \mathcal{S}^*.$$

You also do this when you try to solve the Sudoku in the newspaper in the morning. Even then you will probably follow an (implicit) step-by-step approach. The better this recipe, the faster you will solve the Sudoku. There is a branch of mathematics that deals with the question of how expensive an algorithm is, or how many steps does this algorithm take to arrive at the desired solution for a specific instance of the problem. This branch of mathematics is called 'computational complexity theory'. In addition to the number of steps required, also computation time and amount of memory required to solve the problem on a computer are important. For a problem of size n, you can determine the number of steps for a deterministic problem, for example $f_{\mathcal{A}}(n)$, or indicate in which order of magnitude the computation time falls, via the large $\mathcal{O}$ notation. For example, the number of steps can be linear in n, $\mathcal{O}(n)$, or exponential in n, $\mathcal{O}(2^n)$. It is then interesting for these mathematicians to search for algorithms that find the answer in as few steps as possible, so, find that $\mathcal{A}^*$ that minimises $f_{\mathcal{A}}(n)$:

$$\mathcal{P}_n \mapsto \mathcal{A}^* \mapsto \mathcal{S}^*.$$

For some problems, however, the number of steps to find the best answer is so large that it is not feasible in practice. Consider, for example, the travelling salesman problem, which is a problem that the parcel deliverer wants to solve. A travelling salesman must visit n cities and return home and wants to make the distance he has to travel as short as possible. One possible algorithm to find the shortest tour, is to calculate the distance for all possible routes and to choose the shortest one. How often should a distance be calculated? There are $n! = n \cdot (n-1) \cdot (n-2) \cdot \cdots \cdot 1$ possible routes. For n=20 that means 5109094217170940000 ($\approx 5.1 \cdot 10^{19}$) possible routes. If you have a computer that can calculate 1 million solutions every second, you are calculating for 1.6 million years. For 70 cities, the number of possibilities is about 10^{100}, which is 1 Googol.

In OR, we are therefore more interested in algorithms that find a good solution in reasonable time, rather than the right or best solution. These kinds of algorithms are also called heuristics (from the Greek find, εὑρίσκω, also think of Aristotle with his eureka, 'I found it'). We could note this with

$$\mathcal{P}_n \mapsto \mathcal{H} \mapsto \approx \mathcal{S}^*.$$

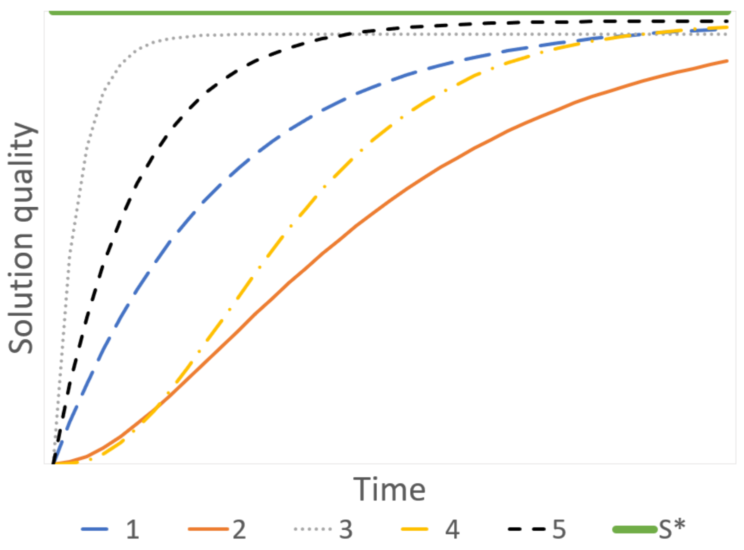

Figure 2: TS paths for various algorithms.

This means that we have a problem of magnitude n and a heuristic that yields an approximation of the solution. Where it was previously important to look for an algorithm in which the number of steps is minimal, here we will look for a heuristic within a framework of three performance indicators: how good is the answer, how much time is needed, or how many steps are needed and how do the first two interact? In Figure 2, for a number of (fictional) heuristics the course over time through the solution space, so called Time-Solution quality (TS) paths, are shown. The best solution would yield the value indicated by the green line (S^*). This value is usually unknown. All other colours are various different heuristics or heuristic-hardware combinations. The difference between black (5) and yellow (4) is obvious: black gives much faster, much better solutions, although they eventually converge to the same solution, which is not the optimal solution. Gray (3) initially performs even better, but converges to a worse solution. One question now might be, which of these heuristics is preferable. Blue (1) and Orange (2) seem clear losers and Black seems to be preferable to Yellow. The difference between Gray and Black is more subtle and depends, for example, on the time scale on the horizontal axis. If the point where black and grey intersect is after a few seconds or minutes, then Black is probably preferable. If this intersection takes hours, days, or weeks of calculation, then Gray might be better. You may also wonder whether you can combine Gray and Black.

Of course, these TS-paths hold for a heuristic for specific input (x) and parameters (θ). It is interesting whether the course of the paths for the different input (x, θ) deviates strongly. Interesting for researchers with a more theoretical slant is to prove that a specific heuristic (i) never deviates more than a certain fraction (α) from the best possible answer.

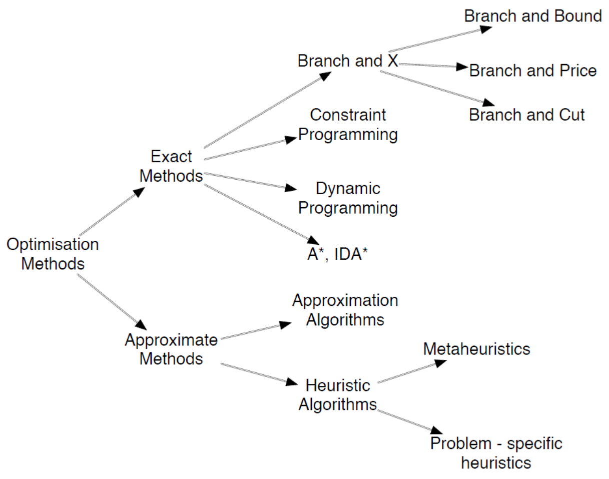

Figure 3: Classification of the Classical Optimisation Methods, from [8].

3 Methods and Computing Pipelines

The second part is about the methods and algorithms to solve a problem. Generic, we can distinguish a number of problem solving methods [8], as shown in Figure 3. At the highest level, there is the difference between exact methods and approximation methods. We have already indicated that most exact methods, which find an optimal solution and guarantee their optimality, are not suitable for solving large problems in most cases, although the scope of these methodologies is expanding due to the development of the computers. The approximation methods can be further divided into approximation algorithms and heuristic algorithms. The first give an approximation guarantee, in the worst case they will not deviate more than a percentage α from the optimal solution. Heuristic algorithms do not provide this guarantee. They are designed for a specific problem and perhaps even for a specific problem instance. This class again consists of two sub-fields, the meta-heuristics and the problem-specific heuristics. Meta-heuristics are general purpose methods and can therefore be used for multiple problems. They often use a specific search method to navigate a solution space. The cleverness here is in the way they move through this space, often based on natural processes. Mostly, these methods work in two steps [27]:

1. Diversification: Randomly search for areas of the search space that are promising;

2. Intensification: Take one or more of these areas and look there in more detail for a better solution.

Problem-specific heuristics use characteristics of the problem, identified by the modeller, to limit the search space and then search (further) for a good solution. Due to this targeted reduction of search space, these problem-specific heuristics are often faster to use, but more time-consuming to develop.

From the application point of view, there are good reasons to prefer problem-specific heuristic methods [44]:

1. Acceptance – "People would rather live with a problem they can not solve than accept a solution they can not understand" (Woolsey and Swanson). The adoption and use by decision makers of decision rules is likely to be facilitated by an understanding, at least intuitively, of how the rules work.

2. Show improvement compared to current practices - decision makers are often (reasonably) satisfied with a heuristic solution if it produces better results than the current approach. The 'best solution' is unknown and often out of sight.

3. Fast results - Sometimes fast, reasonable results are needed and heuristics can be developed and used faster than (complex) optimisation routines.

4. Robustness – Heuristics may be less sensitive to variations in problem characteristics and data quality. From [2] we learn: "Optimal solutions are vulnerable, meaning they can be extremely sensitive to changes in the data. If the problem description changes slightly, to restore an optimal solution, generally the entire problem (which was computationally

expensive to solve in the first place) must be solved again. On the other hand, heuristics often divide the problem and thus ignore inter-partition relationships. This allows updates to be restricted to only the affected partition. Recalculation can be done locally and therefore faster." In addition, some constraints are flexible in practice and a heuristic method can more easily accommodate this flexibility.

Where I indicated earlier that meta-heuristics can often be distinguished in two steps: diversification and intensification, we can also indicate a number of techniques that often serve as building blocks for meta- and problem-specific heuristics [44]:

- Generate random (feasible) solutions: randomly select a solution and evaluate the score of this solution;
- Problem decomposition: cut the problem into parts and solve each of these parts separately;
- Inductive methods: define a problem solving method for a simpler version of the problem and try to translate it to the original problem;
- Reduction: reduce the solution space;
- Approximation methods: manipulate the input in such a way that a simpler problem arises, for example by linearising a continuous function piece-wise;
- Constructive methods: generate a (good) solution step-by-step;
- Local improvement methods: search for a better solution close to a previously found solution through a few simple adjustments;
- Perturbation: slightly change a solution, so that another feasible (or not) solution ('neighbouring solution') is created.

For example, you can say that the method 'Iterated Local Search' [26] is a combination of (1) construct an initial solution, (2) perform a local improvement, (3) perform a perturbation, (4) construct a feasible solution, and then repeat steps (3) and (4). This is shown schematically in Figure 4. We also call this kind of schemes a 'computing pipeline': a process description of the steps you take to arrive at (an approximation of) the optimal or a good

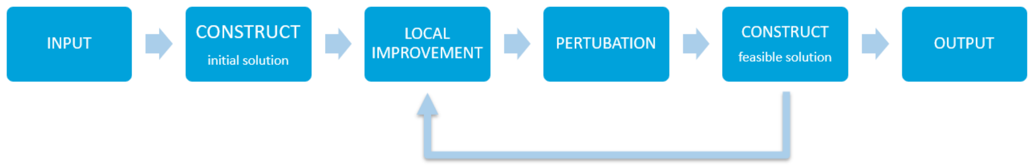

Figure 4: Overview Iterated Local Search steps in a computing pipeline.

solution to a problem. Another method, Grasp [15, 16], is even simpler, repeating the steps (1) generate an arbitrary solution and (2) construct a feasible solution from this solution and remember the best solution of these repetitions. Simulated annealing [48, 23] also starts with an initial solution and a local improvement methodology, where, with a decreasing probability during the process, also deteriorations are accepted to ensure diversification.

In the search for a solution, lessons can be learned from nature. Here also, creatures search or move in a space through a specific structure or movement. For constructive and local improvement methods, the process can be learned from evolutionary processes. If it is about learning methods, inspiration can be drawn from neural processes. Movement through space can be learned from swarms of animals and for repair and protection solutions one could look at immune systems of animals and plants. An overview of methods can be found in Figure 5.

In summary, heuristics are important for solving OR problems in practice. These heuristics can often be constructed from standard building blocks. With these building blocks you can then build computing pipelines at every level of detail, where specific parts of these pipelines can be mapped to different types of hardware. Which leads me to the next topic.

4 Supporting Hardware

Part three of this article is the computer side: what kind of computers do we have available and how does that affect the algorithms and the performance? Although it is not always clear to the end user, a lot of progress has been made at the hardware level. The classical CPU (Central Processing Unit) is much more powerful than before and, based on multiple cores and threats, increasingly capable of (virtual) parallelisation. The GPU (Graphics Processing Unit) is naturally ideally suited for parallelisation, but not for all

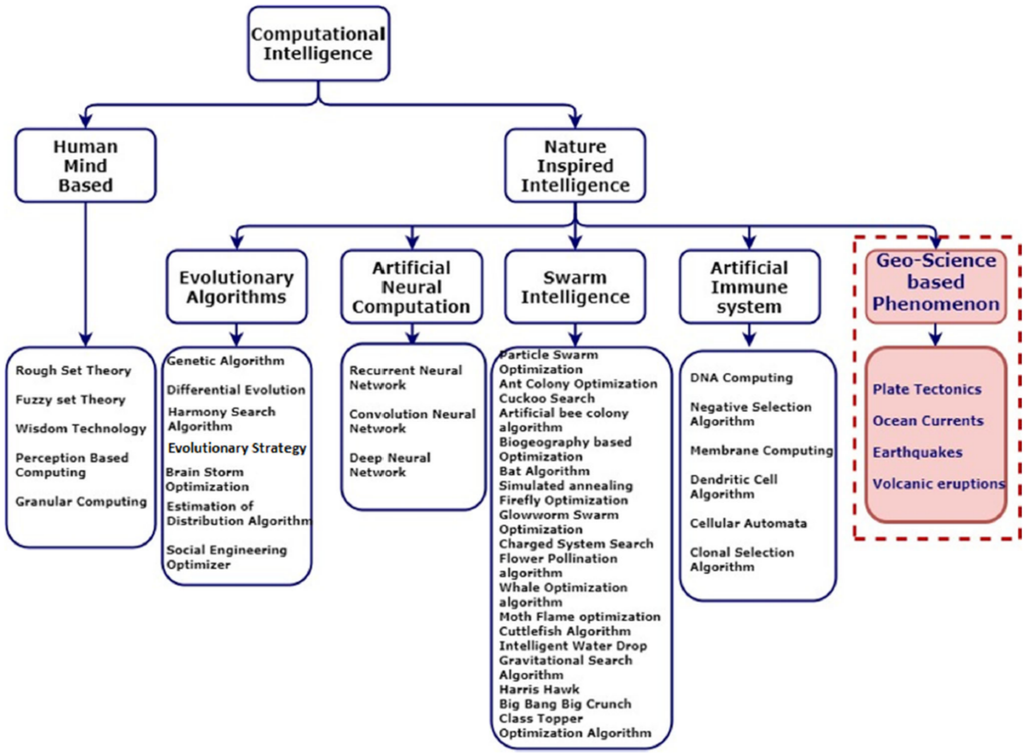

Figure 5: Overview of bio-inspired methods, from [18].

tasks and requires more power. We also have the rise of quantum computers, photonic hardware and the (further) development of classical supercomputers.

A CPU is the electronic heart of the computer, that executes instructions that are contained in a computer program. The CPU performs basic arithmetic, logic, control, and input/output (I/O) operations specified by the instructions in the program. CPUs are called general purpose processors because they can perform almost any type of computation, making them less efficient and less costly in terms of power and chip size. A basic CPU works sequentially. This sequential working is good for linear and complex calculations, but not for simpler and multiple calculations that require parallel calculations. Parallelisation here means that you put several CPU units to work side by side, but that has an upper limit of a few tens or hundreds. This means that there are still tasks that can be performed just fine on a CPU and may be preferable. Examples include machine learning algorithms that do not require parallel computing such as vector machine algorithms, algorithms that support time series data, recurring neural networks, and algorithms that include large-scale branching[2].

A GPU is a specialised processor designed to speed up the creation of graphics. A GPU can be present on a video card or embedded on the motherboard. Modern GPUs are very efficient at manipulating computer graphics and image processing. Their very large scale parallel structure makes them more efficient than CPUs for algorithms that need to process large blocks of data in parallel. They are used for Machine Learning applications that can take advantage of the parallel processing capabilities. GPUs are generally designed with specific algorithms in mind. It is therefore also quite difficult to program an algorithm for efficient execution on a general GPU.

The specific performance advantages for each of these hardware types mean that hybrid solutions can and should also be looked at, where different sub-tasks are assigned to the best processor [24].

In addition to machine learning, research is also being done on parallelisation of search algorithms, as presented in the previous section. For example, in [11] parallel high-performance versions of so-called adapted Antlion and Grasshopper meta-heuristic algorithms are examined to solve Travelling Salesman problems. The authors of [34] look at a GPU accelerated island-

[2]Based on article on ThinkML.ai: ‘CPU vs GPU in Machine Learning Algorithms: Which is Better?’

model genetic algorithm and in [12] they look at a 3D sensor placement problem, taking advantage of a GPU's parallelisation capabilities.

Especially for Neural Networks, more and more use is being made of new hardware accelerations such as FPGA (Field Programmable Gate Arrays), reprogrammable circuits, and ASIC (Application-specific integrated circuit), non-programmable but task-specific circuits [35]. The NPU (Neural Processing Unit) and TPU (Tensor Processing Unit) are again specific versions of this.

In this context you should also think of fully optical neural networks [42] and quantum photonic processors [46] that can perform Gaussian Boson Sampling as their only task for the time being. Gaussian Boson Sampling can be used for example to find dense sub-graphs, such as cliques and graph similarities [6].

Another problem-specific hardware is an optical Ising machine. Several researchers and labs around the world are developing these on a larger scale now [4, 5, 41]. Ising problems can be solved with these machines, defined by:

$$\min H_{\text{Ising}} = -\frac{1}{2}\sum_{ij,i<j}^{N} J_{ij}\sigma_i\sigma_j - \sum_{i}^{N} b_i\sigma_i,$$

where the spins $\sigma_i \in \{-1, 1\}$ must be found, given the interaction matrix J and the bias b. This Ising problem is an equivalent of the QUBO problem:

$$\min x^T Q x,$$

with $x \in \{0, 1\}^N$. The transformation $x_i = \frac{1+\sigma_i}{2}$ makes the two problems equivalent. There exists Ising or QUBO formulations for many well-known optimisation problems [17, 28] and applications of all kinds can be found in recent literature, from financial [36], to machine learning [32], logistics [31, 39] and network optimisation [37, 38].

Another hardware solution for Ising and QUBO problems is the quantum annealer, the best known version of which is offered by the Canadian D-Wave Systems. Here, quantum annealing, once presented by [21], is used to solve optimisation problems in the form of an Ising or QUBO formulation [30]. And with that we have arrived in the quantum domain.

Quantum Computer: a new computer that can solve specific problems in seconds, while the current generation of computers would take many years for solving that same problem. At least, that is what is promised by the scientists who are working on building the Quantum Computer (QC), among others in Delft, The Netherlands. This QC uses quantum mechanical phenomena, such as superposition and entanglement, to perform operations. Superposition is the most famous property used by the QC. Where classical computers require the data to be encoded in binary digits (bits), each of which is always in one of two states (0 or 1), the QC uses quantum bits, called qubits, which can be in both states at the same time, or actually, in a complex linear combination of the two. This state of a qubit is called a superposition. However, when the qubit is observed, it will appear, seemingly randomly, as a 0 or 1. The other property used is entanglement. Here, the states of two particles, each of which may be in a different location, are related to each other. The QC will not be the answer to all problems. In terms of complexity, a new group of problems will arise, BQP, Bounded-error Quantum Polynomial time [3], an extension of P, containing problems that can be solved by the Quantum Computer, with at most a 1/3 probability of error. One of the most well-known problems in BQP is integer factorisation, which is why different types of cryptography will no longer offer the promised security. In terms of development, we are just at the beginning with the QC. The current generation only has a limited number of qubits, that also have a limited lifespan and are of limited stability. Applications are already being worked on for this generation of hardware, the so-called Noisy Intermediate-Scale Quantum (NISQ) Computers. The most important areas of application where QC can already add value over classical computing are expected to be chemical simulations, machine learning and optimisation.

We can currently distinguish two paradigms in quantum computing: gate-based quantum computers (GBC) and the aforementioned quantum annealers (QA). GBC are most similar in operation to the current generation of computers; they are capable of performing operations (gate operations, such as AND, OR) on specific qubits or on multiple qubits at the same time. This allows for actual programming, which is often visualised via circuit diagrams, such as in Figure 6. Here each line represents a qubit and time runs from left to right, with the operations, called gates, represented as blocks.

Well-known algorithms for this paradigm are Grover's algorithm [19], which can find a specific element in an unsorted set, Shor's algorithm [43], which solves the discrete logarithm problem and the integer factorisation problem in polynomial time, HHL [20], for solving systems of linear equations, and VQE [14] and QAOA [13], both variational algorithms for, among other things, optimisation. In a variational algorithm, parameters are assessed for

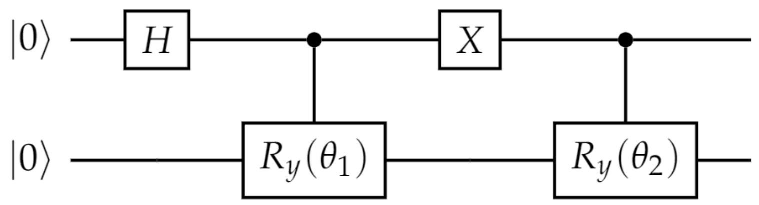

Figure 6: Example quantum circuit.

a quantum circuit in a collaboration between a quantum and a conventional computer, with which the best solution can be found.

Quantum annealing works very differently. The evolution of a quantum state on the quantum processor of D-Wave is described by a time-dependent Hamiltonian ($H(t)$), consisting of the original Hamiltonian (H_0), whose ground state is easy to create, the equal superposition, and the final Hamiltonian (H_1), whose ground state encodes the solution of the current problem, via the QUBO. This system is initialised in the ground state of the original Hamiltonian. The adiabatic theorem says that if the system evolves according to the Schrödinger equation, and the minimum spectral gap of $H(t)$ is not zero, then for T large enough $H(T)$ will converge to the ground state of H_1. Although we will not go into the technical details here, it is good to know that it is usually not possible to estimate a time T to ensure that the system always evolves into the desired state. For some classes of problems, the optimal annealing time has been experimentally determined [1]. There is in general no guarantee of optimality.

Also for QC it holds that each type can solve certain problems (probably) more efficient. However, in general hybrid solutions must be designed to solve practical problems. This applies more generically to the different hardware types discussed. The specific performance advantages for each of the hardware types mean that hybrid solutions can and should be considered, in which different sub-tasks are assigned to the best processor.

We are on the eve of a hardware revolution. Hardware that can solve specific problems much faster is approaching. It is becoming increasingly important to think about hybrid approaches: which part of the problem do we allow which hardware to solve?

5 Computational Operations Research

How does this all come together in the research field 'Computational Operations Research'? Before that, let us first talk about some current trends. As said before, OR uses quantitative methods to help decision makers design, analyse and improve the performance or operation of systems. However, these systems have been subject to changes in recent years. We are dealing with increasingly complex systems, which are also mutually dependent, so-called 'systems of systems'. Think, for example, of ICT and energy systems, ICT and automotive systems or logistics systems. Next, these systems can also have conflicting objectives and an ever-growing amount of data is available, introducing additional sources of complexity and uncertainty.

In addition, the systems must become increasingly independent and adaptive; they have to adapt themselves to changes and provide optimal settings and control, the so-called self-X or self-* methods and systems. Important from the control view of this, is the frequently asked question: do we prefer 'central control or completely decentralised, via autonomous agents'? Models themselves are also becoming more autonomous, as indicated earlier, where they are fed from the data directly to allow structures and (interactive) behaviour to be established. These trends on the demand side are putting a lot of pressure on computational efficiency.

On the supply side, we have another important trend, which is the development of the hardware, especially the emergence of photonic and quantum computing. It is important that OR professionals are aware of these developments, understand what these developments mean for their field and that students are already trained with an awareness of these developments. Not only the students in the technical fields, but also students in gamma fields who solve problems with computers.

The idea of the research field is that we bring together as many techniques as possible, starting from traditional mathematics and computer science, and linking them to hardware evolution and thus to physics, as depicted in Figure 7. The aim here is achieving practically applicable computational efficiency for solving optimisation or decision problems in the complex systems mentioned earlier, whether or not with uncertainty. In the near future it will no longer be sufficient to use (only) a heuristic method or solver with a conventional computer. We have to look at the most efficient combination of smart methods and available hardware.

We have summarised this more specifically under the name 'Quantum en-

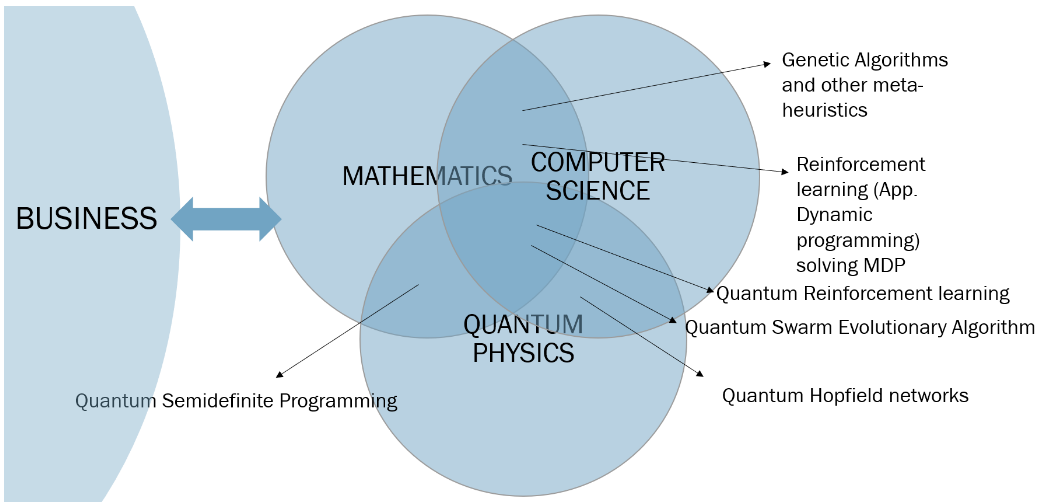
BUSINESS
MATHEMATICS
COMPUTER SCIENCE
QUANTUM PHYSICS
Genetic Algorithms and other meta-heuristics
Reinforcement learning (App. Dynamic programming) solving MDP
Quantum Reinforcement learning
Quantum Swarm Evolutionary Algorithm
Quantum Hopfield networks
Quantum Semidefinite Programming

Figure 7: Scope of Computational Operations Research.

hanced decision intelligence' as part of Computational Operations Research. The need for the field of 'Decision Intelligence' is also recognised elsewhere. Gartner names Decision Intelligence as the top trend of 2022[3] and describes it as 'a wide range of decision-making techniques bringing multiple traditional and advanced disciplines together to design, model, align, execute, monitor and tune decision models and processes.' We combine this with the right allocation of supporting (quantum) computing technology. It is worth mentioning here that we are not specifically looking for quantum supremacy, but use the quantum computer (and other hardware) wherever possible as a tool to find improvements for TS paths. To this end, we try to create new computing pipelines for all kinds of problems, improving the aforementioned TS path. This may require redesigning the entire pipeline, not only for the problem solving methodology, but also including data processing and modelling step, for existing problems, to get a formulation that might already have a better TS path on a classic computer or that is better suitable for quantum computers or other hardware and therefore gets a better TS path. Figure 8 shows this schematically, two alternative pipelines to facilitate applications of quantum computing.

An example: the problem of 'Sparse sensor placement for reconstruction' as described in [7], can be solved classically by performing a number

[3]Note that Autonomic Systems themselves are also one of their current top trends.

of successive steps: Singular value decomposition - QR factorisation - Column Pivoting - Solving a Linear System. We can now take an alternative pipeline: Determine mutual information all sensors - solve the Quadratic Integer Programming Problem as proposed by [33]. The latter is classically difficult to solve, but very suitable for a quantum computer, with which a more favourable TS path can still be found.

An interesting development related to the TS-paths is the need to compare different quantum hardware [25]. Purely the number of qubits does not say everything, which leads to, for example, Quantum Volume [9] and CLOPS [49] as measures, a combination of the number of qubits, degree of noise, number of gates per second, etc. Another way to quantify the performance of QC are the application-oriented benchmarks, in which a specific problem is identified, solved and checked how quickly or well that happens. An example of this is the Q-Score developed by ATOS [29]. This score is determined by the approximate solutions of the Max-Cut problem for specific graphs, $(N, \frac{1}{2})$ Erdös-Rényi graphs, for increasingly larger graphs. The largest N for which a solution can be found that is clearly better than a randomly obtained solution is called the Q-score. We have calculated this score for the D-Wave hardware [47] and introduced a time factor for this. Note that this score is therefore not only hardware dependent, but depends on the combination of methods, techniques and hardware.

This brings me to the end of this article. The title was 'Searching for Optimisation'. I hope you now see that this title can be interpreted in two ways. Search techniques are very important in optimisation as a means to find a good solution. In addition, within OR, together with our colleagues in adjacent, and sometimes partly overlapping, knowledge areas, we are looking for ways to improve our tool set, in order to be able to solve problems in business and society even more efficiently. This increasingly demands a multidisciplinary approach, connecting OR with computer science, physics and other fields.

References

[1] Tameem Albash and Daniel A Lidar. "Demonstration of a scaling advantage for a quantum annealer over simulated annealing". In: *Physical Review X* 8.3 (2018), p. 031016.

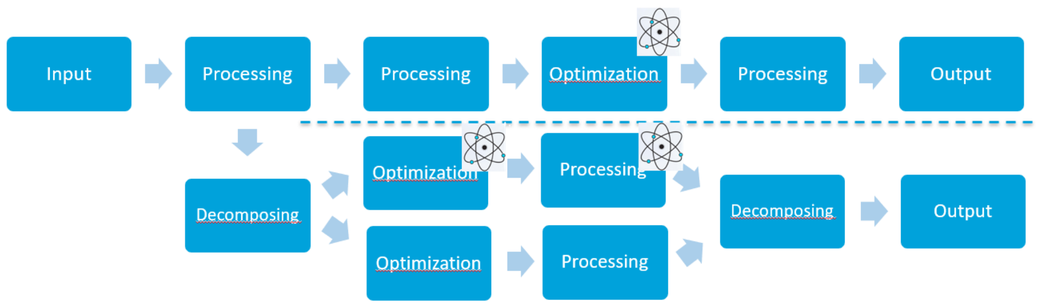

Figure 8: Example of alternative pipelines.

[2] John J Bartholdi III and Loren K Platzman. “Heuristics based on spacefilling curves for combinatorial problems in Euclidean space”. In: *Management Science* 34.3 (1988), pp. 291–305.

[3] Ethan Bernstein and Umesh Vazirani. “Quantum complexity theory”. In: *SIAM Journal on computing* 26.5 (1997), pp. 1411–1473.

[4] Fabian Böhm, Guy Verschaffelt, and Guy Van der Sande. “A poor man’s coherent Ising machine based on opto-electronic feedback systems for solving optimization problems”. In: *Nature communications* 10.1 (2019), pp. 1–9.

[5] Fabian Böhm et al. “Order-of-magnitude differences in computational performance of analog Ising machines induced by the choice of nonlinearity”. In: *Communications Physics* 4.1 (2021), pp. 1–11.

[6] Thomas R Bromley et al. “Applications of near-term photonic quantum computers: software and algorithms”. In: *Quantum Science and Technology* 5.3 (2020), p. 034010.

[7] Steven L Brunton and J Nathan Kutz. *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press, 2019.

[8] Jose Caceres-Cruz et al. “Rich vehicle routing problem: Survey”. In: *ACM Computing Surveys (CSUR)* 47.2 (2014), pp. 1–28.

[9] Andrew W Cross et al. “Validating quantum computers using randomized model circuits”. In: *Physical Review A* 100.3 (2019), p. 032328.

[10] George B Dantzig. “Origins of the simplex method”. In: *A history of scientific computing*. 1990, pp. 141–151.

[11] GR Dheemanth, VC Skanda, and Rahul Nagpal. “Parallel Antlion Optimisation (ALO) and Grasshopper Optimization (GOA) for Travelling Salesman Problem (TSP)”. In: *Innovations in Computer Science and Engineering*. Springer, 2021, pp. 787–793.

[12] Joacim Dybedal and Geir Hovland. “GPU-Based Optimisation of 3D Sensor Placement Considering Redundancy, Range and Field of View”. In: *2020 15th IEEE Conference on Industrial Electronics and Applications (ICIEA)*. IEEE. 2020, pp. 1484–1489.

[13] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. “A quantum approximate optimization algorithm”. In: *arXiv preprint arXiv:1411.4028* (2014).

[14] Dmitry A Fedorov et al. “VQE method: A short survey and recent developments”. In: *Materials Theory* 6.1 (2022), pp. 1–21.

[15] Thomas A Feo and Mauricio GC Resende. “A probabilistic heuristic for a computationally difficult set covering problem”. In: *Operations research letters* 8.2 (1989), pp. 67–71.

[16] Thomas A Feo and Mauricio GC Resende. “Greedy randomized adaptive search procedures”. In: *Journal of global optimization* 6.2 (1995), pp. 109–133.

[17] Fred Glover, Gary Kochenberger, and Yu Du. “A tutorial on formulating and using QUBO models”. In: *arXiv preprint arXiv:1811.11538* (2018).

[18] Lavika Goel. “An extensive review of computational intelligence-based optimization algorithms: trends and applications”. In: *Soft Computing* 24.21 (2020), pp. 16519–16549.

[19] Lov K Grover. “A fast quantum mechanical algorithm for database search”. In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 1996, pp. 212–219.

[20] Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. “Quantum algorithm for linear systems of equations”. In: *Physical review letters* 103.15 (2009), p. 150502.

[21] Tadashi Kadowaki and Hidetoshi Nishimori. “Quantum annealing in the transverse Ising model”. In: *Physical Review E* 58.5 (1998), p. 5355.

[22] Hamdi Kavak et al. "Big data, agents, and machine learning: towards a data-driven agent-based modeling approach". In: *Proceedings of the Annual Simulation Symposium.* 2018, pp. 1–12.

[23] Scott Kirkpatrick. "Optimization by simulated annealing: Quantitative studies". In: *Journal of statistical physics* 34.5 (1984), pp. 975–986.

[24] Kishore Kothapalli et al. "CPU and/or GPU: Revisiting the GPU vs. CPU myth". In: *arXiv preprint arXiv:1303.2171* (2013).

[25] Matt Langione et al. *The race to quantum advantage depends on benchmarking.* Boston Consulting Group, 2022.

[26] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. "Iterated local search". In: *Handbook of metaheuristics.* Springer, 2003, pp. 320–353.

[27] Manuel Lozano and Carlos Garcia-Martinez. "Hybrid metaheuristics with evolutionary algorithms specializing in intensification and diversification: Overview and progress report". In: *Computers & Operations Research* 37.3 (2010), pp. 481–497.

[28] Andrew Lucas. "Ising formulations of many NP problems". In: *Frontiers in physics* 2 (2014), p. 5.

[29] Simon Martiel, Thomas Ayral, and Cyril Allouche. "Benchmarking Quantum Coprocessors in an Application-Centric, Hardware-Agnostic, and Scalable Way". In: *IEEE Transactions on Quantum Engineering* 2 (2021), pp. 1–11.

[30] Catherine C McGeoch. "Adiabatic quantum computation and quantum annealing: Theory and practice". In: *Synthesis Lectures on Quantum Computing* 5.2 (2014), pp. 1–93.

[31] Florian Neukart et al. "Traffic flow optimization using a quantum annealer". In: *Frontiers in ICT* 4 (2017), p. 29.

[32] Niels MP Neumann et al. "Multi-agent reinforcement learning using simulated quantum annealing". In: *International Conference on Computational Science.* Springer. 2020, pp. 562–575.

[33] Xuan Vinh Nguyen et al. "Effective global approaches for mutual information based feature selection". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining.* 2014, pp. 512–521.

[34] Ryoma Ohira and Md Saiful Islam. "GPU accelerated genetic algorithm with sequence-based clustering for ordered problems". In: *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. 2020, pp. 1–8.

[35] Linus Pettersson. *Convolutional Neural Networks on FPGA and GPU on the Edge: A Comparison*. 2020.

[36] Frank Phillipson and Harshil Singh Bhatia. "Portfolio optimisation using the d-wave quantum annealer". In: *International Conference on Computational Science*. Springer. 2021, pp. 45–59.

[37] Frank Phillipson, Tariq Bontekoe, and Irina Chiscop. "Energy storage scheduling: a QUBO formulation for quantum computing". In: *International Conference on Innovations for Community Services*. Springer. 2021, pp. 251–261.

[38] Frank Phillipson and Irina Chiscop. "A Quantum Approach for Tactical Capacity Management of Distributed Electricity Generation". In: *International Conference on Innovations for Community Services*. Springer. 2022, pp. 323–333.

[39] Frank Phillipson and Irina Chiscop. "Multimodal container planning: a QUBO formulation and implementation on a quantum annealer". In: *International Conference on Computational Science*. Springer. 2021, pp. 30–44.

[40] Jayant Rajgopal. "Principles and applications of operations research". In: *Maynard's Industrial Engineering Handbook.–2004.–P* (2004), pp. 11–27.

[41] Vikram Ramesh, Vighnesh Natarajan, and Anil Prabhakar. "A spatial-photonic Ising machine to solve the two-way number-partitioning problem". In: *arXiv preprint arXiv:2110.01616* (2021).

[42] Bin Shi, Nicola Calabretta, and Ripalta Stabile. "InP photonic integrated multi-layer neural networks: Architecture and performance analysis". In: *APL Photonics* 7.1 (2022), p. 010801.

[43] Peter W Shor. "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer". In: *SIAM review* 41.2 (1999), pp. 303–332.

[44] Edward Allen Silver. "An overview of heuristic solution methods". In: *Journal of the operational research society* 55.9 (2004), pp. 936–956.

[45] Sergej Svorobej et al. "Towards automated data-driven model creation for cloud computing simulation." In: *SimuTools*. 2015, pp. 248–255.

[46] Caterina Taballione et al. "A universal fully reconfigurable 12-mode quantum photonic processor". In: *Materials for Quantum Technology* 1.3 (2021), p. 035002.

[47] Ward Van der Schoot et al. "Evaluating The Q-score of D-Wave's QUBO Solvers". In: *Proceedings of IEEE World Congress on SERVICES*. 2022, pp. 1–6.

[48] Vladimir Vcerny. "Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm". In: *Journal of optimization theory and applications* 45.1 (1985), pp. 41–51.

[49] Andrew Wack et al. "Quality, Speed, and Scale: three key attributes to measure the performance of near-term quantum computers". In: *arXiv preprint arXiv:2110.14108* (2021).