

OPTIMIZATION TECHNOLOGY WHITE PAPER

A comparative study of optimization techniques



ILOG, Inc.
1901 Landings Drive
Mountain View, California 94043
1-800-FOR-ILOG
www.ilog.com



© 1997, ILOG, Inc.

All rights reserved.

All trademarks and registered trademarks
are the property of their respective holders.

Table of Contents

Introduction	4
Why optimize?	4
When to optimize?	4
What optimization method to use?	5
Optimization	6
Problem formulation	6
Decision variables	6
Objective function	7
Constraint functions	7
Solution approaches	8
Heuristics	8
Optimization techniques	8
Rule based systems	8
Linear and integer programming	9
Domain reduction and constraint propagation	9
Genetic algorithms	10
Simulated annealing	11
Examples	13
The mixing problem	13
The scheduling problem	15
The resource allocation problem	16
The design problem	20
Conclusion	21

Introduction

Why optimize?

The search for improved profits never ends. Many of the obvious areas for increased profit have been exhausted during the past decade. Both data gathering and management have been automated, even by smaller firms; optimization, however, can be used to extract detailed information from a system to yield new business opportunities. Optimization can fine tune operations to create further improvements in process efficiency. Systems based on optimization tools can promote flexibility so systems remain efficient under many scenarios.

Many companies are now focusing on delivering new, custom designed services that rely on optimization. A delivery company may want to provide up to the minute information about the delivery times of customers' packages. This information can be made available for individual packages by means of data from the delivery network and optimization tools.

The existence of repetitive processes in many operations means improving the efficiency of a single process, even by small amounts, can lead to substantial gains for the overall operation. Many production assembly lines have already been automated for improved efficiency, but further gains may be achieved by considering the order in which products are sent along the line. The gain for each product may be minuscule, but the overall gain for the assembly system increases with each product sent along the line. When the magnitude of the operation is large, the overall gain can be very large.

Systems that adapt quickly to changes in a process are of greater value than those that do not. Reacting to a change quickly allows a system to maintain efficiency. Airport gate allocation is dependent on flight arrival and departure times. If a flight is delayed, the most efficient gate allocation may need to be changed. Finding efficient new allocations quickly lets gate allocations be implemented on the fly, which lets the allocation system run as efficiently as the information flow allows.

Many companies are looking to the future with ideas that initially present complex problems. Optimization tools can transform many of these ideas into reality. New technology is opening up satellite possibilities for companies dealing in telecommunications. Using optimization tools allows them efficiently to plan the launch and coordination of satellite missions, thus reducing costs and making the missions feasible. Flexible employee scheduling lets companies utilize employees efficiently, while still meeting customer and employee requirements. Optimization allows myriad possibilities to be assessed and the most effective schedule found in a timely manner.

When to optimize?

There is a special set of problems that have been decisive in creating competitive advantages for businesses. Solving these problems has been difficult, but has lead to significant cost reductions, cycle time improvements and revenue opportunities. When business situations involve costly assets, high sensitivity to tradeoffs among decision factors, or narrow margins for error, they can often benefit from optimized resource allocation. Resources, including capital, equipment, labor, time, bandwidth and even executive attention span, must be carefully assigned in the right amounts, at the right times, and in the right order to obtain the best possible result. When the set of alternatives is extremely large, optimization selects the best one, such as choosing the combination of cards to include in a computer box to meet the specifications at lowest cost.

There are business situations for which optimization would be inappropriate. Consider calculating the largest oval table that fits into a rectangular room. This may appear to be an optimization problem, but it may be solved directly using calculus. Whenever there are simpler mechanisms, they should be applied first; however, many seemingly simple problems are in fact enormously complex:

- How to cut fabric patterns from a bolt of cloth to minimize waste
- How to route delivery trucks to minimize mileage
- How to order construction tasks to lower building costs

Optimization should be used when there is no easy, direct solution. This usually occurs when the problem structure is complex or there are millions of possible solutions. In such cases, there may be no efficient, direct solution approach, so optimization techniques can search for the best solution. In some cases, when no solution can be found, the problem is relaxed (some restrictions on the alternatives are lifted), and optimization is used to find the best solution for the relaxed problem. The relaxed solution may provide the basis for accurately solving the original problem.

What optimization method to use?

There are numerous approaches to optimization. The approach used depends on the problem. Several optimization techniques are considered in this paper. Rules based systems are commonly used in control applications, and involve predetermined rules to generate solutions. These are useful when the problem has a set of rules that can be known in advance. Linear programming and its extension to integer programming are among the better known approaches, and are widely used to optimize economic goals when rules are unstable or unknowable. Domain reduction and constraint propagation are combined in a relatively new technique that has proved useful with scheduling problems. Genetic algorithms and simulated annealing are two recently developed approaches that improve solutions over time.

This paper presents an overview of optimization, including problem formulation, optimization theory and the use of heuristics. Summaries of these techniques are followed by example applications, with a comparison of the merits of each approach to the application. The purpose of this paper is to help readers assess approaches to their own problems.

Optimization

Optimization is the process of finding the best solution (or optimum) from a set of solutions. Optimization is divided into two classes: global and local. Global optimization finds the best solution from the set of all solutions (the global optimum). Local optimization finds the best solution from a set of solutions that are close to one another (a local optimum). In local optimization, the solution found depends on the starting point for the optimization. Global optimization will always find the same solution regardless of the starting point, but usually will require more computational power.

It may be nearly impossible to find the global optimum in some applications, or there may be no way to check whether a solution is the global optimum; even local optima, however, may prove beneficial. Indeed, in many cases finding the global optimum may not be necessary. Finding a good solution (a local optimum) quickly may be more desirable than finding the best solution (the global optimum) slowly. In fact, there are times when finding *any* feasible solution quickly has business value. For example, a customer service representative may want to give a delivery time to a customer. The new order is entered into the system and a good schedule generated by local optimization. Although this may not be the best schedule, the representative can give a delivery time to the customer quickly and know the actual delivery time will be the same or earlier, depending whether global optimization is used later.

The type of optimization required for a problem depends on the structure of the problem. If all the decision variables are real and the objective and constraints are linear functions, linear programming is usually the best approach. This is a simplistic formulation, and real world applications often require non-linear functions, set variables, or logical variables and constraints. The ability of approaches to handle such complications is the primary factor to be considered when deciding among the approaches.

Problem formulation

The key to optimization is problem formulation. Decision variables, an objective function and some constraint functions describe problems. The structure of these variables and functions frequently determines the best approach for solving the problem.

Decision variables

Decision variables are used to represent decisions that need to be made. For example, a cereal company may want to know what proportion of its new cereal should be made from shredded wheat. This decision is represented by a real variable in the range $[0, 1]$. A machine scheduler may want to know which component to process first among three components (A, B and C). This decision is represented by a discrete variable with possible values in $\{A, B, C\}$. There are many types of variables:

- Real variables have a lower and upper bound (l and u respectively) and may take any value in the range $[l, u]$.
- Integer variables have a lower and upper bound (l and u respectively) and may take any integer value in the range $[l, u]$.
- Logical variables may be either true or false. These variables may be represented by an integer variable with a lower bound of 0 (for false) and an upper bound of 1 (for true).
- Choice variables have a set of possible values (e.g., the bar codes of different products) and must take one of these values. These variables are often represented by an integer variable with a lower bound of

1 and an upper bound of the number of possible values. The integers in this range correspond to different choices.

- Set variables have different sets as possible values (e.g., the set of products to store in a container) and must be one of these sets. These variables may be represented as choice variables, but their interactions with other variables are often more complex.

Objective function

The objective function describes the goal, economic or otherwise, of a specific choice of decision variables. For example, a cereal company may wish to define the proportions of ingredients to make a new cereal and minimize costs. The economic goal of a specific mix of ingredients would be the cost of purchasing and processing the ingredients in the mix. Objective functions are classified into two types:

- Linear objective functions involve a linear combination of the decision variables, (i.e., each decision variable is multiplied by a constant, and these values are added together). These are common for many cost functions:

$$c(x_1, x_2, \dots, x_n) = a_1 x_1 + a_2 x_2 + \dots + a_n x_n.$$

- Non-linear objective functions allow for any function of the decision variables, which covers a wider range of interactions and tradeoffs that may occur in real world situations:

$$c(x_1, x_2) = x_1^2 + x_1 x_2 \text{ or } c(x_1, x_2) = \log(x_1) + \sin(x_2).$$

Constraint functions

Constraint functions describe the logical or physical conditions the decision variables must obey. For example, the cereal company mentioned above may want its mix of ingredients to have a specified level of fiber. As another example, a construction company may want to know when to begin each activity in building a house, but the roof cannot be put on until the frame is constructed.

Constraint functions may be linear or non-linear; another type is the logical constraint, e.g., **if A, then not B**. For example, if an air compressor is being used for sandblasting, it can not be used for painting at the same time. These constraints may be represented using logical variables and constraint functions, but in some approaches, these constraints are dealt with in a more direct way.

There is never a unique formulation for each problem. The formulation plays a major role in determining which approach to use. The formulation will often be altered so it lends itself to a desired approach; however, choosing an approach before formulating a problem is usually not the best way to proceed. Selecting an approach first frequently leads one to ignore more effective approaches. Some problems have good formulations for a certain approach, but that approach may be very inefficient for a particular problem. Considering the formulation of the problem and the structure of the formulation enables one to consider the pros and cons of several solution approaches before deciding which to use.

Commercial optimization problems can usually be classified into several categories that can then be matched with optimization techniques to deliver the best problem formulation and solution approach. For example, scheduling problems are usually formulated with integer variables and logical constraints, and should usually be solved using constraint propagation and domain reduction. By looking at similar problems with existing efficient solutions, managers can make good decisions about the optimization tools to apply.

Solution approaches

Heuristics

In some cases, optimization may not be possible or simply not efficient enough to generate solutions in the required time, so a heuristic may be used instead. Heuristics are algorithms that do not attempt optimality, but generate acceptable solutions in a majority of cases. Heuristics are used because they are computationally efficient and/or easy to implement. They are not very precise or predictable; further, they occasionally fall apart due to scalability problems and/or unrealistic assumptions. If a problem is solved repetitively and the parameters change often, heuristics are more likely to fall apart. Answers can always be generated, but some may be wasteful. Heuristic performance may be improved by incorporating optimization algorithms. Better solutions are generated for a broader range of parameters by optimizing inside some steps of the heuristic; however, pure optimization is the only way to guarantee the best solution for all problem parameters.

Optimization techniques

This section presents a summary of the main optimization techniques currently used in industry. The details of each technique are not included; rather, an overview of each technique is given.

Rule based systems

Rule based systems are not optimization tools, but are commonly used in control systems, heuristics and adaptive systems that may provide a type of optimization. Rule based systems consist of a process controlled by a fixed set of hundreds or even thousands of rules. In practice, the rules and their interactions are very complex. If the rules do not change as the system changes, optimality is not guaranteed. In fact, static rule based systems are heuristics. They generally generate good decisions, but there may be cases in which the rules perform badly. Currently, rule based systems are successful primarily for alarm and information correlation, and data filtering, particularly in telecommunication network monitoring applications.

Adaptive rule based systems may provide some optimization. These systems change the rules depending on the current state of the process or the value of the previous decision, which allows the rules to adapt with the process. Although this is an improvement on static rule based systems, the way in which the rules adapt is still static. Thus, adaptive rule based systems provide a type of optimization, since the rules still provide good solutions for a large range of situations; however, it is still a heuristic approach.

There is one case in which rule based systems always provide optimality. If it has been *proved* that the rules used always provide an optimal solution for any state of the process, the rule based system will provide optimal control of the system; unfortunately, these proofs usually pose a more complex problem than generating the rules in the first place. Typically, the biggest problem with rule systems is that no one knows the rules to apply.

Vendors of rule based systems include Neuron Data, The Haley Enterprise and ILOG.

Linear and integer programming

Linear programming is probably the best known and most widely used optimization technique. It is commonly used to make managerial decisions about the allocation of scarce resources to products. The

costs of the resources and profits from the products are used to determine the best solution. Any problem that can be formulated with real decision variables, a linear objective function, and linear constraint functions (referred to as a linear program), may be solved using linear programming.

Linear programs were originally solved using the Simplex method; more recently, interior point methods have come to the forefront. Problems with 10,000 constraints and 100,000 variables may be solved in a reasonable time frame using interior point methods. Industrial problems with over 12,000,000 variables are being solved today.

Although linear programming is very efficient for solving linear programs, problems arise if the objective or constraints are not linear functions. In some cases, non-linear functions may be approximated by piecewise linear functions, and linear programming is still used; however, this is often an inefficient representation of the problem, and a very limited number of functions may be represented this way. These formulations may cause explosively large matrices that are solved very slowly. For example, logical constraints usually require at least two linear constraints to be represented correctly, so a large number of logical constraints creates a huge linear program. This is a common difficulty in scheduling and sequencing problems.

Similarly, other types of variables cannot be dealt with directly using linear programming. Integer programming uses linear programming to solve problems with some integer variables (all other variable types may be represented by a combination of integer variables and linear constraints), but still having linear objective and constraint functions. The integer variables are represented as real variables, and the resulting linear program is solved; then a repetitive process is used in which an integer variable is bounded above or below in an attempt to force it to an integer value by adding constraints, and the modified linear program is resolved. This method (branch and bound) terminates when all the integer variables take integer values. When the number of integer variables is small, integer programming solves problems fairly quickly; unfortunately, this process can be too time consuming for problems with large numbers of integer variables. Some problems require millions of iterations to be solved.

Vendors of linear and integer programming packages include ILOG CPLEX Division and OSL.

Domain reduction and constraint propagation

Domain reduction and constraint propagation is often the appropriate approach for problems with very different variables and constraints, such as those with integer, logical and choice variables and/or linear and logical constraints. Domain reduction and constraint propagation is especially appropriate for problems with integer variables and logical constraints, such as scheduling problems. The combination of domain reduction and constraint propagation with linear programming can be very effective when solving problems with both real and integer variables and linear constraints.

This technique starts with the premise that it may be more efficient to identify where the solution *is not* before trying to quantify what it *is*. Every decision has a domain of possible values, e.g., a real variable can take any value in the range $[l, u]$; an integer variable can take any integer value in the range $[l, u]$, and a logical variable can take the value **0** or **1**. Domain reduction systematically removes values that are no longer possible from each variable's domain. If there are enough constraints, the domains may be dramatically reduced. When a variable has an empty domain, the problem becomes infeasible, which is to say, there is no possible value for that variable, so the problem has no solution.

Constraint propagation works by systematically reducing all variable domains. Each constraint has an associated set of variables that are affected by the constraint. Each variable belongs to the associated set of at least one constraint; otherwise, it is an unconstrained variable. When the domain of that variable is altered, the associated set of variables for each of these constraints is affected. The change to their domains in turn affects other variables via other constraints. The effect of a change to one variable is propagated via constraints throughout all the decision variables. The effectiveness of this approach increases as the network of constraints expands.

Domain reduction and constraint propagation allows a compact representation of feasible solutions, those that satisfy all constraints. This approach enables all the types of variables and constraints to be represented in an intuitively obvious way. As the variables are related to one another via constraints, the feasible region (represented by a domain for each variable) is refined using constraint propagation. When the value of a variable is set, the effect on all the other variables is known instantly. If setting a variable to a certain value causes another variable to become infeasible, constraint propagation reveals this instantly. Thus, the feasible region is always represented, even when some of the variables have set values. If the feasible region becomes empty, the current set values are infeasible and need to be changed.

A search of all feasible solutions is needed to find a global optimum. This search defines the order in which variables should be set and a way to choose a value from a variable's domain. The internal search algorithm is crucial for efficient optimization. When the only requirement is a feasible solution, the first solution found by the search algorithm is sufficient, but if an objective function is being optimized, the search algorithm needs to find the objective function value for all feasible solutions. If the search algorithm is inefficient, this process may be too time consuming for practical use.

Vendors of domain reduction and constraint propagation packages include ILOG and Cosytech.

Genetic algorithms

Genetic algorithms find the solution to problems by using evolution from an acceptable solution to a better one. In contrast to the previous approaches, genetic algorithms focus on local optimization; however, the evolution is controlled in an attempt to search the entire solution space, which may make the optimization global. A population of solutions exists at each iteration of the algorithm. This population may be said to "evolve" into a new population at the next iteration. The evolution is achieved by using specific operators: reproduction, crossover and mutation.

- Reproduction copies a solution from the old population to the new population with a probability depending on the fitness of the solution. The fitness of a solution is the value of the objective function for that solution. The better the objective function, the higher the probability that solution will survive to the next iteration.
- Crossover swaps a section of two solutions. Each new solution contains part of the two old solutions. For example, consider a problem finding the value of five logical variables. Two solutions for this problem may be:

1 0 0 1 0

↑↓

1 1 0 1 1

and the crossover may swap the middle three values to give:

1 1 0 1 0

1 0 0 1 1

in the new population. This operator attempts to create solutions that have the best properties of the original solutions.

- Mutation mutates a solution, which is to say, randomly changes a small part of it. Very little mutation takes place in practice, but this step is necessary for global optimization. With reproduction and crossover, genetic algorithms might get stuck in an area where there is a local optimum. Mutation

allows the algorithm to jump to a new solution. Thus, if there is sufficient mutation, the algorithm will always sample the entire solution space and the optimum will be global.

Other, more specialized, operators may also be used, but reproduction, crossover and mutation generally prove sufficient for most practical problems.

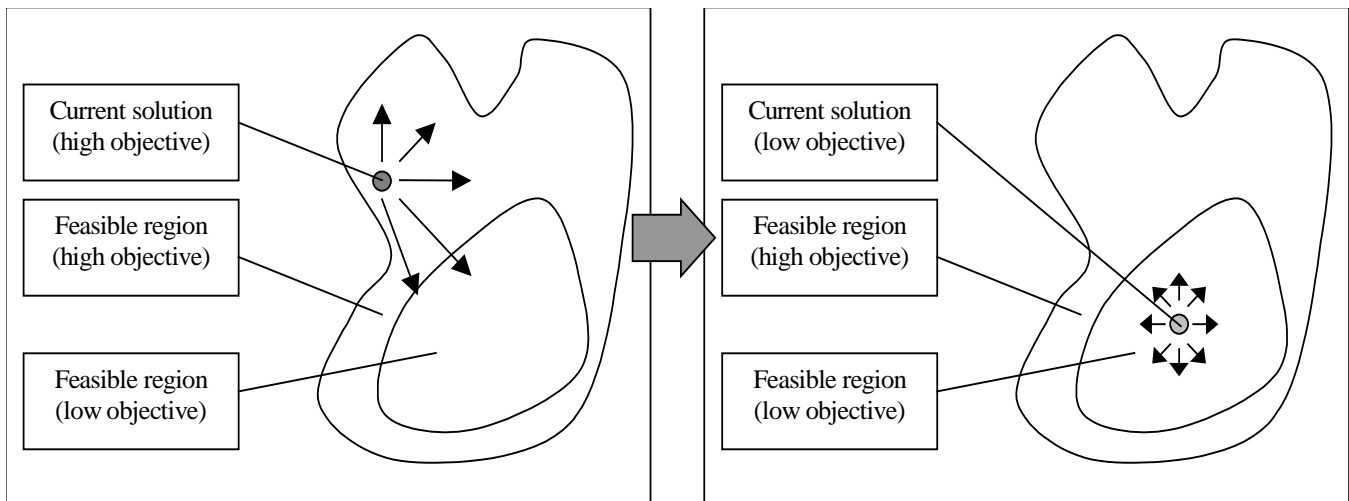
Of course, genetic algorithms may take an enormous number of iterations to succeed, but the simplicity of the process supports reasonable execution speed. Genetic algorithms have proved useful for problems with constraint and objective functions that are difficult to deal with, such as highly non-linear functions. Genetic algorithms are easy to implement if feasible solutions can be generated easily, but care needs to be taken when designing the operators to ensure the algorithm does not optimize only locally. Additional concerns are the need to generate and maintain the population of solutions, as well as the uncertainty when to terminate the algorithm. Genetic algorithms work better with a large population of solutions, since this means the chances of a good solution evolving are increased as a result of a larger genetic pool; unfortunately, the space needed to store a single solution may be large, so the algorithm uses a large amount of memory to maintain the entire population. Generating a large initial population may be time consuming, and there is no way to know when a genetic algorithm has found the optimal solution. As in natural evolution, the longer the algorithm is allowed to run, the better the solution becomes, but there is no way to know whether the “species” is perfect. The uncertainty when to terminate the algorithm, combined with the overhead from generating the initial population of solutions, can lead to undesirably long execution times.

Vendors of genetic algorithms include Optimax.

Simulated annealing

Simulated annealing is similar to a genetic algorithm, since it improves solutions over time. The term “annealing” refers to the way liquid metals are cooled slowly to ensure a low energy, highly structured solid form. Simulated annealing may be said to “cool” the solution slowly to ensure that it achieves the lowest possible objective. By initially allowing a high level of movement throughout the solution space, simulated annealing attempts to search the entire space, thus providing global optimization. Later in the process, the cooling allows only small movements in the solution space, and the process converges on a final solution. The nature of the movements throughout the process means that as the process cools, the solution moves to an area with low objective values (see Figure 1).

Figure 1 Annealing in a minimization problem Initial movements are large and directed toward low objective regions. As the process converges, the movements are small and the solution lies at a low objective.



Simulated annealing requires four elements:

- 1) A description of possible solutions $\mathbf{x} \in \mathbf{X}$
- 2) A generator of random changes between solutions
- 3) An objective function $\mathbf{E}(\mathbf{x})$ for the solutions
- 4) A control parameter T and an annealing schedule that describes how the control parameter varies with time

Starting from a solution \mathbf{x}_0 the solution moves from $\mathbf{x}_1 \rightarrow \mathbf{x}_2$ with probability

where k is a constant of nature relating temperature to energy (Boltzmann's constant).

$$p = e^{-\frac{E(\mathbf{x}_2) - E(\mathbf{x}_1)}{kT}}$$

Simulated annealing has the same advantages and disadvantages as genetic algorithms. The approach can solve problems with difficult functions, but the optimization may be local, and there is no way to know whether the optimal solution has been found. Again, the longer the algorithm is allowed to run, the better the solution will be; however, there is no way to be certain whether the final solution achieved has the lowest energy.

Examples

The mixing problem

Morning, Inc. mixes cereal from various ingredients (wheat, oats, nuts, etc.) The company wants its cereal to have certain qualities (low in fat, high in fiber, etc.) The properties of the ingredients are known, as are their prices. Morning, Inc. wants to know what proportion of each ingredient to include in the final cereal mix to meet quality requirements and minimize production costs. Table 1 contains some sample data for such a mixing problem.

Amount per Serving	Product Requirements	Shredded Wheat	Oat Bran	Corn Flour	Nuts	Salt
Cost (\$)	Lowest	0.01	0.02	0.01	0.08	0.01
Fat (%)	≤ 3	1	1	4	8	5
Fiber (%)	≥ 14	12	18	8	2	1
Protein (g)	≥ 5	2	1	4	10	3

Table 1 Cereal ingredients

The decision variables for this problem are the proportions of each ingredient in the cereal mix. The variables are x_{SW} , x_{OB} , x_{CF} , x_N and x_S , where x_{SW} = proportion of shredded wheat in the cereal mix, x_{OB} = proportion of oat bran in the cereal mix, etc. The objective function is the linear function $c(\mathbf{x}) = 0.01x_{SW} + 0.02x_{OB} + 0.01x_{CF} + 0.08x_N + 0.01x_S$ and needs to be minimized. There are three linear constraints, one for each quality requirement, and one constraint ensuring the proportions add up to 1. For example, the constraint for fat is $x_{SW} + x_{OB} + 4x_{CF} + 8x_N + 5x_S \leq 3$.

This formulation is a linear program, and so may be solved efficiently by a linear programming interior point method; in fact, this small problem is rapidly solved with the Simplex method. The set of rules needed to solve even this simple linear program would be far too complex to use a rule based approach. Domain reduction and constraint propagation would provide reduced domains for the variables; however, all the domains are real intervals, so the search for an optimal solution would need to consider a very large number of solutions. Both a genetic algorithm and simulated annealing could be used here, but the efficiency of these approaches does not compare to linear programming in this case. Thus, linear programming is the best approach for this problem, which is precisely what it was designed to solve more than 50 years ago.

Now consider the following change: for each different ingredient, Morning, Inc. must spend considerable time setting up machinery, and if an ingredient is less than 10% of the mix, the company would rather remove it and save its machinery setup. This introduces five logical variables z_{SW} , z_{OB} , z_{CF} , z_N and z_S , where z_{SW} = true if shredded wheat is included in the mix, false if it is not, z_{OB} = true if oat bran is included in the mix, false if it is not, etc. There are ten new logical constraints, two for each ingredient. The constraints for shredded wheat are if z_{SW} is true then $x_{SW} \geq 0.1$ and if z_{SW} is false then $x_{SW} = 0$. These constraints can be represented as linear constraints and the logical variables as 0-1 integer variables. Thus, the constraint system is still linear, but the presence of integer variables mean the formulation is now an integer program.

Integer programming can now be used, but does not have the same efficiency as linear programming. In fact, the branch and bound procedure for integer programming may require $2^5 = 32$ runs of a linear solver. The rules needed to solve this new problem will be even more complex than for the original problem, so a

rule based approach is an even less attractive option. Domain reduction and constraint propagation does not require the conversion of the new variables and constraints. Logical variables and logical constraints can be dealt with directly by this approach. The constraint propagation allows the effect these logical variables have on the feasible region to be seen immediately, but the feasible sets are still composed of real intervals, which present a problem when searching for the optimal solution. Genetic algorithms have a problem finding the best point in a real interval. Points in such intervals are generated at random, so the solution may never fall on the best point. The longer the algorithm is allowed to run, the greater the chance of finding this point. Simulated annealing also becomes more complicated when optimizing over real variables. The combination of integer and real variables means the generator for random solution changes is difficult to define. One method that could be considered is a combination of linear programming and domain reduction and constraint propagation. The logical variables and constraints could be dealt with by domain reduction and constraint propagation. This would yield a set of feasible regions, each of which could be used as the feasible region for a linear program with the original objective. This is similar to the branch and bound approach, but the representation of the feasible regions is more compact, and should result in reduced solution times. Some of the linear runs could be eliminated altogether as a result of domain reduction. On large problems, this combination of the two approaches reduces the number of variables and constraints introduced by the branch and bound process, and delivers good results.

It can be seen from this example that solving a problem with a combination of different variables and constraints is inherently difficult for all the approaches considered here. In many cases, the specifics of the problem will be the deciding factor for the approach that should be used. In this particular example, the linearity of the objective and constraints mean linear programming is the method of choice for the problem as originally posed. The introduction of logical variables and constraints means techniques to deal with these complications are needed; however, the presence of real variables requires an intelligent search engine such as the interior point or Simplex methods for linear programming to find the best solution. This example demonstrates the possibility of combining solution techniques.

The scheduling problem

Housing Construction is building a residential complex. There are several activities to be completed during construction, some with time links to other activities (i.e., roofing may not be started until the walls are done). Some activities require common resources, such as labor. Housing Construction wants to know when to start each activity, which resources to assign to an activity and how long construction will take.

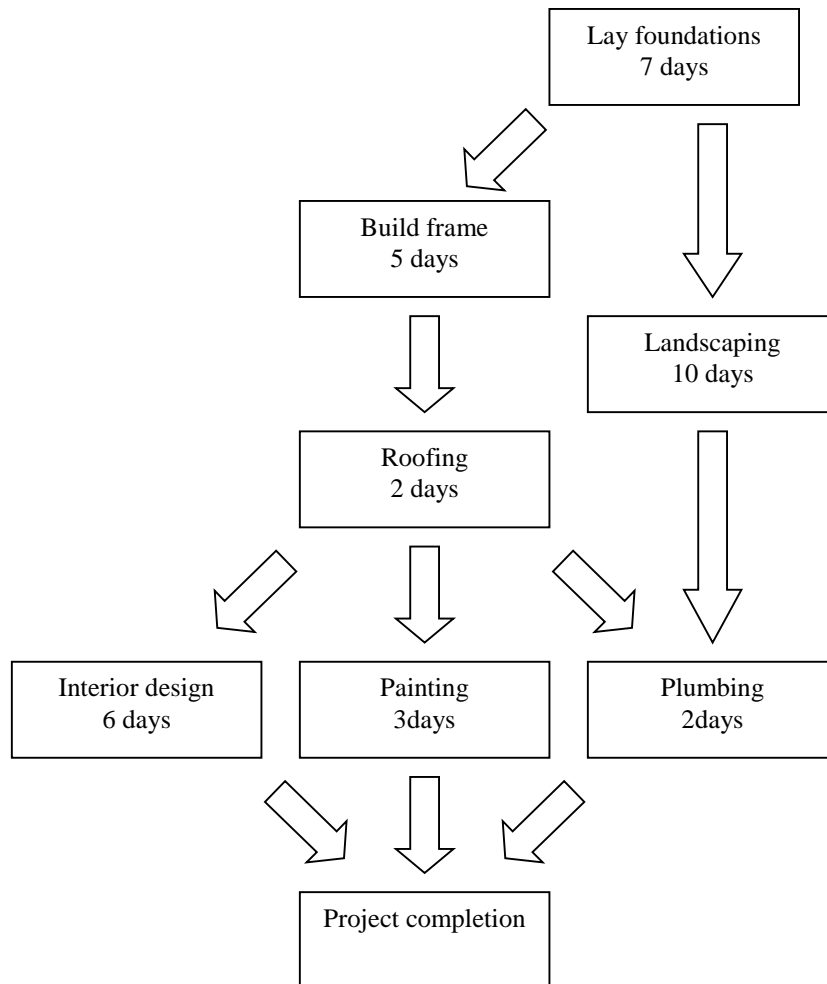


Figure 2 Activity summary

As in the previous example, a rule based system would require a very complex set of rules to guarantee the best solution for varying data (activity duration and resource requirements) in this scheduling problem; conversely, a fairly simple set of rules could provide a heuristic to generate a good solution for a majority of the data.

The best formulation for linear programming has one decision variable for the start time of each activity: x_{LF} = start time for laying foundations, x_{ID} = start time for interior design, etc. The objective is to minimize the start time of the project completion activity. There is one constraint for each time link, e.g., the constraint that building the frame cannot be started until laying the foundations is finished is expressed $x_{BF} \geq x_{LF} + 7$. The restrictions on common resources are very difficult to express as linear constraints; in fact, the only way

to deal with these restrictions is through 0-1 integer variables and several constraints, which transforms the problem into an integer program. As stated in the previous example, integer programming requires many runs of a linear solver, and may not be efficient enough to be the approach of choice.

Domain reduction and constraint propagation may be the best approach for this problem. Using the same decision variables as the linear programming formulation (x_{LF} = start time for laying foundations, x_{ID} = start time for interior design, etc.) and the same constraints for the time links ($x_{BF} \geq x_{LF} + 7$ expresses the fact building the frame can not be done until laying the foundation is finished), domain reduction and constraint propagation can use logical constraints explicitly to express the restrictions on common resources. By giving priority of resources to the activities with the longest duration and starting activities as early as possible, domain reduction and constraint propagation can search the feasible region efficiently and solve scheduling problems quickly. The restrictions on common resources are adhered to by eliminating intervals of the available start times for an activity if resources required by that activity are being used during the intervals. As in the previous example, domain reduction and constraint propagation may be used to improve the efficiency of integer programming by eliminating some of the linear solver runs. Domain reduction and constraint propagation with a good search engine should be an efficient solver for most scheduling problems.

Both genetic algorithms and simulated annealing deal with the common resource restrictions better than linear programming; unfortunately, as in the previous example, both genetic algorithms and simulated annealing have trouble dealing with continuous variables. Specialized operators for a genetic algorithm may provide a good solver, but these operators are very problem dependent. Therefore, this specialized genetic algorithm may be limited in the scope of problems it solves efficiently.

The resource allocation problem

Petro Co. owns and operates a fleet of oil tankers carrying an assortment of petroleum products. The loads vary depending on the time of year. The configuration of the tanks in the oil tankers determines which products may be stored in each tank. Petro Co. needs to know which product to put in each tank to minimize the total amount of product remaining to load. Table 2 contains sample tank configurations and product loads.

Product	Capacity (tons)					Load (tons)
	Tank 1	Tank 2	Tank 3	Tank 4	Tank 5	
Premium	1136	1117	1120	2166	2147	5708
Regular	1102	1084	1087	2101	2083	2182
Gas Oil	1277	1256	1259	2434	2412	1289
Fuel Oil	1460	1435	1439	2782	2758	1435

Table 2 Sample tank configuration and product load

Again, guaranteeing the best solution for differing data (product loads and tank capacities) with a rule based system would involve a complex set of rules; by contrast, a simple set of rules can provide a good heuristic for this problem. The following rule would provide a good solution for most data:

- For the product with the largest remaining load, use the remaining tank with the largest capacity for that product

This rule would yield the following solution:

- Premium loaded in tanks 2, 4 and 5, and 278 tons are left to load
- Regular loaded in tank 1, and 1080 tons are left to load
- Gas Oil is not loaded, and 1289 tons are left to load
- Fuel Oil is loaded in tank 3, and no more is left to load

One better solution loads the Fuel Oil in tank 2 and the Premium in tank 3. This would still load all the Fuel Oil, but leave only 275 tons of Premium. Clearly, this rule based approach is only a heuristic.

The linear formulation of this problem is an integer program with 0-1 integer variables, one for each product tank pair, e.g., $z_{p_1} = 1$ if Premium is loaded in tank 1 and 0 otherwise. There are also two real variables for each product, s_p = the shortage of tank capacity for Premium (i.e., the remaining Premium left to load) and e_p = the excess of tank capacity for Premium (i.e., the difference between the tank capacity for Premium and the amount of Premium loaded). The objective is to minimize the sum of the shortage variables. There is one constraint for each product (to check how much of that product is loaded, e.g., the constraint for Premium is $1136z_{p_1} + 1117z_{p_2} + 1120z_{p_3} + 2166z_{p_4} + 2147z_{p_5} + s_p - e_p = 5708$) and one constraint for each tank (to ensure the tank holds only one product, e.g., the constraint for tank 1 is $z_{p_1} + z_{r_1} + z_{g_1} + z_{f_1} \leq 1$). There are 20 integer variables, each taking the value 0 or 1, possibly requiring 2^{20} runs of a linear solver. Although this problem is manageable by today's standards, integer programming is probably not the best approach.

Domain reduction and constraint propagation formulates the problem in a more compact form. There is a choice variable for each tank. The choices are the different products that may be stored in the tank, e.g., p_1 = the product stored in tank 1 and $p_i \in \{\text{Premium, Regular, Gas Oil, Fuel Oil}\}$. This formulation implicitly allows only one product in each tank. The objective of a solution can be calculated using logical constraints and real variables. For example, the variable l_{p_1} = the amount of load space for Premium provided by tank 1 and the constraints **If $p_1 = \text{Premium}$ then $l_{p_1} = 1136$ otherwise $l_{p_1} = 0$, If $p_2 = \text{Premium}$ then $l_{p_2} = 1117$ otherwise $l_{p_2} = 0$** , etc. determine the tank capacity for Premium. The constraint **If $p_1 + p_2 + p_3 + p_4 + p_5 \geq 5708$ then $c_p = 0$ otherwise $c_p = 5708 - (p_1 + p_2 + p_3 + p_4 + p_5)$** determines the cost for Premium. The objective is to minimize the total cost for all the products. All the real variables are determined by the choice variables, so these variables should be set first by the internal search. There are now only $4^5 = 2^{10}$ different solutions, each of which requires only an objective calculation, not a linear solver run. By using the diverse representations supported by domain reduction and constraint propagation, the work required to solve this problem is greatly reduced.

Both genetic algorithms and simulated annealing are appropriate for this problem. The only decision variables needed for these techniques are the 0-1 variables from the linear programming formulation, e.g., $z_{p_1} = 1$ if Premium is loaded in tank 1 and 0 otherwise. The real variables are not needed, since they are used only to keep the objective linear. Since neither genetic algorithms nor simulated annealing require a linear objective, the contribution to the objective both approaches is $c(z) = \max(0, 5708 - (1136z_{p_1} + 1117z_{p_2} + 1120z_{p_3} + 2166z_{p_4} + 2147z_{p_5})) + \max(0, 2182 - (1102z_{r_1} + 1084z_{r_2} + 1087z_{r_3} + 2101z_{r_4} + 2083z_{r_5})) + \max(0, 1289 - (1277z_{g_1} + 1256z_{g_2} + 1259z_{g_3} + 2434z_{g_4} + 2412z_{g_5})) + \max(0, 1435 - (1460z_{f_1} + 1435z_{f_2} + 1439z_{f_3} + 2782z_{f_4} + 2758z_{f_5}))$. The generic genetic algorithm is sufficient to provide global optimization for this problem. The crossover operator swaps all the decision variables for a random set of tanks, given two solutions:

i	P					R					G					F				
	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
z¹	0	0	1	0	0	0	0	0	0	1	1	0	0	1	0	0	1	0	0	0
z²	1	1	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

with $c(z^1) = 4687$ and $c(z^2) = 2823$, that crossover at tanks 1, 2 and 5 to give two new solutions:

I	P					R					G					F				
	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
z¹	1	1	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0
z²	0	0	1	1	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0

with $c(z^1) = 3869$ and $c(z^2) = 2533$. The mutation operator swaps the allocation of a random tank to another random product, given the solution:

i	P					R					G					F				
	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
z	0	0	1	0	0	0	0	0	0	1	1	0	0	1	0	0	1	0	0	0

with $c(z) = 4687$, a mutation occurs that switches tank 1 to Premium to give the solution:

i	P					R					G					F				
	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
z	1	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0

with $c(z) = 3551$. These operators allow the genetic algorithm to span the entire feasible region without generating infeasible points. Along with the standard reproduction operator, these operators specify an efficient genetic algorithm for solving the problem. Simulated annealing can also provide global optimization for this problem. The generator for random changes simply chooses a tank at random and switches it to another random product exactly as the mutation operator specified for genetic algorithms. This generator allows simulated annealing to move throughout the entire feasible region, finally settling at a global optimum. Although both these techniques may be slower than integer programming or domain reduction and constraint propagation for a problem this small, the simplicity and efficiency of their implementations, along with the guarantee of global optimization, mean that as problems become larger (more tanks and products) these algorithms become very competitive.

The design problem

Big Construction is building a stadium. The architect must decide where to support the main 100 m cross beam in one of the stands. He can add a support every 10 m along the beam, and the supports vary in strength from 1 to 10. The cost of the supports increases with strength, but also varies depending where the support is added. The total stability of the beam depends on the strength and placement of the supports, but must reach a safety requirement of being able to hold 50 tons. The architect must minimize the cost of supporting the beam. The cost of adding a support is $(3000 - 100x + x^2) \log(s + 1)$ dollars, and a support is able to hold tons, where s = the strength of the support and x = the position of the beam (between 0 and 100 m along the beam).

$$\left(\frac{(50 - x)^4}{62500} - \frac{(50 - x)^2}{25} + 100 \right) \frac{s^2}{10}$$

As in many of the other examples, finding the rules needed to guarantee optimality is as complicated as solving the problem itself, so a rule based approach is not an option. The problem is also highly non-linear, and cannot be solved by either linear or integer programming. The decision variables for this problem are $s_{10}, s_{20}, \dots, s_{100}$, where s_x = the strength of the support at x (0 means there is no support). The objective is to minimize the cost of the supports, and the only constraint is that the total support of the beams is greater than 50 tons. Domain reduction and constraint propagation can be used to solve this problem; unfortunately, there is only one constraint to propagate and a large feasible region. Therefore, even with a efficient search, this approach will be slow to find the global optimum.

The best approach is to use either a genetic algorithm or simulated annealing. As in the previous example, the crossover operator would swap the strengths of random supports between two solutions, and the mutation operator would randomly change the strength of a randomly chosen support. This would guarantee the entire feasible region is considered; unlike the previous example, infeasible solutions may be generated. This leads to an inefficient step that slows the genetic algorithm, but once the algorithm moves to a good region, this will occur infrequently. Simulated annealing evidences the problem of generating infeasible points, since the random generator is the same as mutation for the genetic algorithm. As in the genetic algorithm, once simulated annealing moves into a good region and cools (allowing only small steps) these inefficient steps will occur rarely. This example shows the power of these two approaches when dealing with integer decision variables in a highly non-linear problem.

Conclusion

Several factors need to be considered when using optimization to solve business problems. First, the formulation of the problem needs to be carefully structured. In most cases, more than one formulation makes sense; however, finding the best formulation is not always easy. Indeed, the merit of a formulation often depends on the solution approach used to solve the problem. If the solution technique is inefficient for a given problem formulation, a more appropriate formulation may be required. It may be that the formulation can be solved efficiently by another solution approach. This is the balance managers need to consider when selecting optimization software. By taking several views of the problem with different solution approaches in mind, management can make an informed decision on both the best problem formulation and the appropriate optimization software.

This paper summarizes several optimization techniques widely used in the commercial sector. The advantages and disadvantages of each technique indicate that some problems lend themselves to certain techniques. The examples in this paper show when some approaches are preferable to others.

Rule based systems do not provide optimization, although a semblance of optimization may be achieved with adaptive systems. Rule based systems may also be used to create heuristics that provide good solutions for a majority of problem parameters. For most problems, the guarantee of optimality cannot be achieved without a complex system of rules. Creating these rules is often as difficult as solving the original problem.

Linear programming is a very powerful tool for solving linear formulations, but its limitations become clear when different variables (integer, logical) or functions (logical constraints, non-linear objectives) are required to formulate problems correctly. Integer programming compensates in many cases, but is less efficient, and still does not cope with non-linear functions.

Domain reduction and constraint propagation provides a platform for diverse problem formulations. The ability of this technique to incorporate many types of variables and functions means it may be applied to most problems. It supports a compact representation of the feasible solution space in most cases. The weakness of this approach becomes apparent when searching the feasible region for an optimal solution. If the feasible region is large, this approach may become inefficient at finding the best solution. The need for efficient search engines is critical when applying domain reduction and constraint propagation. When both linear and non-linear constraints are involved, using linear programming solvers for the linear constraints and domain reduction solvers for the non-linear constraints provides efficient search engines for constraint propagation. Combining these techniques provides an approach that may be applied to a broad range of problems.

Both genetic algorithms and simulated annealing are techniques based on natural processes. Genetic algorithms allow a solution to evolve and improve over time, while simulated annealing cools a solution slowly to achieve the lowest possible objective (when minimizing). These approaches use random behavior to try to include all possible solutions. Unfortunately, this proves disadvantageous, since the random net is not always cast wide enough, with the result that the optimization is only local. Problems occur in continuous spaces where random selection has an infinite number of possibilities. When the decision variables are more discrete in nature, the power of these algorithms becomes clear. There are no restrictions on the objective or constraint functions. The algorithms inevitably move closer to an optimal solution, and provide better and better solutions as they progress.

The example applications examined here demonstrate that domain reduction and constraint propagation is the most robust solution technique, although it can be inefficient in some cases. Linear programming, genetic algorithms and simulated annealing are the most advantageous techniques when the problem formulation is suited to their particular strengths. In many real world applications, a mix of decision variables is present. Many of these applications reduce to a linear formulation with both real and integer

variables. For these problems, a mix of linear programming with domain reduction and constraint propagation can be the most efficient way to represent and search the solution space.

The final decision on an optimization package depends on a balance of several factors. The modeling of the real world application with a problem formulation is the key step. This formulation depends not only on the problem, but the available optimization techniques. The optimization technique used also depends on the formulation, and the structure of the formulation is crucial in determining the most efficient solution approach. Managers need to consider the problem being solved in depth and the applicability of different solution approaches before making decisions.