Chapter 2
General Concepts in Metaheuristic Search

Metaheuristics have become a very popular family of solution methods for optimization problems because they are capable of finding "acceptable" solutions in a "reasonable" amount of time. Most optimization problems in practice are too complex to be approached by exact methods that can guarantee finding global optimal solutions. The time required to find and verify globally optimal solutions is impractical in most applications. An entire computational theory, which we will not discussed here, has been developed around problem complexity. It suffices to say that it is now known that the great majority of the optimization problems found in practice fall within a category that makes them "computationally intractable." Having accepted the reality that solution methods that yield verifiable globally optimal solutions are not practical, we must apply criteria derived from the problem context to determine what is an acceptable solution and what is reasonable amount of time. For instance, some timetabling problems (e.g., scheduling of courses at University) are notoriously difficult because they include many constraints. Therefore, an acceptable solution in this context could be one that violates the least number of constraints or one that improves a collective preference function value by a relatively small percentage over a solution found by a human scheduler. Comparisons against "manual" solutions are popular in practical problems and help define what is acceptable. Knowledge of the problem context is also necessary to define what is reasonable in terms of solution time. The range is wide. For instance, if an optimization problem is being solved to help a company make strategic decisions then reasonable solution times might be measured in weeks, or even months. These decisions are long range and involve budgets that are in the millions or billions of dollars. On the other end of the spectrum, there are optimization problems that need to be solved in "real time". For example, pricing problems in revenue management systems such as those used by airlines often need to be solved instantaneously in order to update airfares as demand changes. In fact, there are systems that uniquely identify and evaluate each passenger request to determine the optimal price based on revenue management and business strategies.

A. Duarte et al., *Metaheuristics for Business Analytics*, EURO Advanced Tutorials on Operational Research, https://doi.org/10.1007/978-3-319-68119-1_2

The choice of the metaheuristic to use is related to the solution time. It is well-known that, given enough solution time, well-conceived implementations of any metaheuristic methodology tend to perform at about the same level. That is, if competing well-conceived implementations are executed for an inordinate amount of time, it is very likely that the differences in the solutions that they find are not statistically significant. However, given that the strategies vary from one methodology to another, their performance profile also varies. The performance profile is a plot of the value of the objective function associated with the best solution found during the search against the solution time.

As stated by Laguna et al. [3], every metaheuristic procedure is built by considering that to be effective it must include strategies for both search diversification and intensification (also known as exploration and exploitation, respectively). Intensification refers to mechanisms and parameter settings that encourage the addition of solution features that historically (i.e., during the search) have been found to have merit. It also refers to strategies that focus the search on a particular region of attraction. These strategies must be balanced with diversification processes that expand the exploration of the solution space. The opportunity for search diversification depends on the "optimization horizon", that is the length of the optimization process (typically limited by a number of algorithmic iterations, total computational time, or number of objective function evaluations). The optimization horizon relates to the "reasonable" amount of time discussed above and therefore is associated with the problem context. The merit of exact optimization approaches that are expected to run to completion (i.e., that are configured to find and confirm optimal solutions) is measured exclusively by running time. In the realm of metaheuristics, merit is measured with a combination of the quality of the best solution found and the associated computational time to find it. Consider, for instance, the performance profile (i.e., the objective function value of the best solution found during the search) of two hypothetical metaheuristic procedures applied to a minimization problem (see Fig. 2.1).

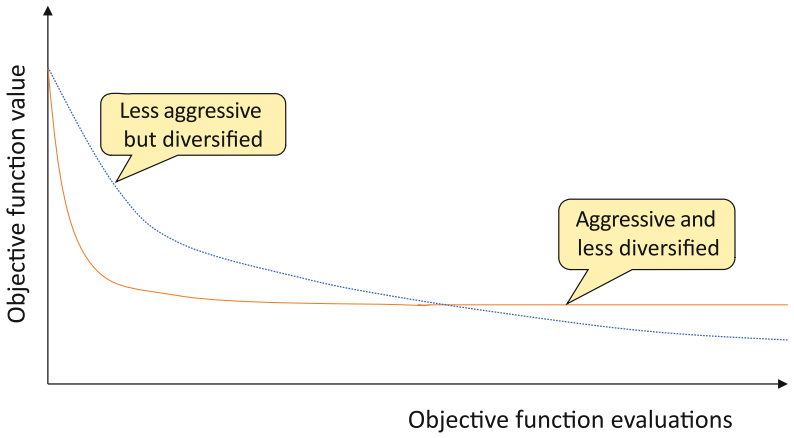

Fig. 2.1 Performance profiles of two hypothetical metaheuristic procedures

As depicted in Fig. 2.1, the search strategies and the parameter values of a particular procedure may be set to perform a search that initially is less aggressive (with respect to an early intensification process) but that, given enough computational time, it is able to steadily find improved outcomes (see dashed line). On the other hand, a procedure may intensify early in the search but may lack diversification strategies that would allow it to improve upon the incumbent solution later in the search. Ideally, a procedure should be able to do both, intensify and diversify in order to meet the goals of the analyst. Clearly, the best choice of metaheuristic design and the associated values of the search parameters to trigger intensification and diversification stages depend on the optimization horizon. In Fig. 2.1, the "Aggressive and less Diversified" procedure represented by the solid line is preferable for a short optimization horizon while the "Less Aggressive but Diversified" represented by the dashed line would be preferable in a problem context where the optimization horizon is relatively long.

The way metaheuristic methods balance search intensification and diversification varies. There are two main distinctions in the classification presented in Table 1.5: population-based procedures versus single-solution based procedures and the use of randomization. In general, population-based procedures maintain diversification by measuring dissimilarities among the solutions in the population. It is then said that a population "converges" when the solutions in the population become "too similar" to each other. If the similarity measure reaches a predetermined threshold, a mechanism is triggered to avoid what is called "premature convergence." This concept refers to a situation where the population has no diversity too early in the optimization horizon. The mechanisms that could be triggered to avoid premature convergence often include some form of randomization. That is, some totally random or semi-random solutions are added to a population in order to induce diversity. Other strategies include the use of memory, where solutions are constructed in such a way that are not only different from those in the current population but also avoid attributes that have been present in solution already explored during the search.

In this book, we focus on single-solution based metaheuristics. In these methodologies, intensification strategies are based on applying a local search to a diverse solution, modifying choice rules to encourage move combinations and solution features historically found to produce high quality solutions, or applying a variety of neighborhood structures. They may also initiate a return to attractive regions to search them more thoroughly. Attractive regions are associated with the concept of elite solutions. Membership in the elite set is often determined by setting a threshold that is connected to the objective function value of the best solution found during the search. Elite solutions can either be used to trigger intensification phases designed to examine their immediate neighborhoods or to expand the neighborhood of another solution. Hence, the main difference between intensification and diversification is that during an intensification stage the search focuses on examining neighborhoods while diversification focuses on moving the search to regions of the solution space that have largely remained unexplored.

During intensification stages, the term "neighbor" may have a broader meaning than in the usual context of neighborhood search. That is, in addition to considering solutions that are adjacent or close to the current solution by means of standard move mechanisms (e.g., flipping the values of a single binary variable as shown in Table 1.4), intensification strategies may generate neighbors by either grafting together components of elite solutions or by using modified evaluations that favor the introduction of such components. The diversification stage on the other hand encourages the search process to examine unvisited regions and to generate solutions that differ in various significant ways from those seen before. Again, such an approach can be based on generating subassemblies of solution components that are then "fleshed out" to produce full solutions, or can rely on modified evaluations as embodied, for example, in the use of penalty/incentive functions designed to discourage/encourage the addition of certain solution features.

2.1 Solution Representation

How to represent a solution is a fundamental design question in metaheuristic optimization. The solution representation is the main factor that determines the form and size of the search space. Therefore, the efficiency of the search is very much dependent on the choice of the solution representation. The representation also determines how the objective function will be evaluated and how the move mechanisms (also referred to as search operators) will create neighborhoods. Consider the following example adapted from [4].

Example 2.1 A pharmaceutical company is hiring five new salespeople to expand its sales in a western state. Pharmaceutical sales representatives do not sell directly to doctors because doctors do not purchase and distribute drugs. However, doctors do write prescriptions, and it is that activity the sales representatives try to influence. The pharmaceutical company is focusing its efforts on the 10 counties in the state and estimated the number of doctors in each county as shown in the last column of Table 2.1. This table also shows ten possible sales regions (comprising contiguous sets of counties) that the company has identified.

For instance, if a sales representative is assigned to region 1, he or she would be responsible for counties 1, 3, and 5. Each sales representative may be assigned to only a single sales region, so only half of the sales regions will be used. The company would like to assign the five sales representatives to the possible regions in such a way that at least one sales representative covers each county. If a county is covered by more than one sales representative, the doctors in the county are split equally among the sales representatives covering the county. The company is con-

cerned about fairness and therefore would like to find the set of regions that will result in a total number of doctors assigned to each sales representative be as equal as possible.

Table 2.1 Number of doctors per county and definition of sales region for Example 2.1

County	Sales region										Doctors
	1	2	3	4	5	6	7	8	9	10	
1	1		1		1						113
2		1		1		1		1			106
3	1		1				1			1	84
4		1		1							52
5	1				1			1			155
6		1				1				1	103
7			1				1		1		87
8		1			1			1			91
9				1			1		1		128
10			1			1				1	131

Solutions to selection problems such as the one described in Example 2.1 are typically represented with binary variables. In this case, we can define 10 binary variables, one for each region, where a value of one represents that the corresponding region has been selected and a value of zero represents that the region has not been selected. That is, the solution is represented by a binary vector of size 10. For instance, the selection of regions 1, 3, 4, 8, and 9 is represented as follows:

$$(1, 0, 1, 1, 0, 0, 0, 1, 1, 0)$$

The search space is then defined by all binary vectors of size 10. This means that the space consists of $2^{10} = 1,024$ solutions. However, not all of these solutions are feasible. Note that there are only 5 sales representatives and the selection must be such that all counties are covered. Therefore only those solutions (binary vectors) that include exactly 5 ones and 5 zeros are possible and within those only the ones that cover all counties are feasible. The number of possible region selections can be calculated as the total number of ways for choosing 5 regions out of 10:

$$\binom{10}{5} = \frac{10!}{5!(10-5)!} = 252$$

A complete examination of all these region selections reveals that only 114 are feasible. The remaining 138 selections fail to cover all the counties. The binary solution representation creates a search space for which only approximately 11% of the solutions are feasible. In other words, the solution space is a relatively small

fraction of the search space, when considering searches that are allowed to traverse both feasible and infeasible parts of the space.

Consider an alternative (discrete value) representation consisting of the identities of the selected regions. A solution is then represented by five discrete values in the range from 1 to 10. Therefore, the selection of regions 1, 3, 4, 8, and 9 is represented as follows:

$$(1, 3, 4, 8, 9)$$

This representation creates a search space with a size of $10^5 = 100,000$ solutions. The search space is much larger because it includes both infeasible solutions (i.e., all those for which a region is selected more than once and/or the selected regions do not cover all the counties) and duplicate solutions (i.e., solutions that only differ in the order in which the selected regions appear in the solution). This space can be significantly reduced by imposing a constraint requiring the selected regions to be all different. This all different constraint reduces the space to $10 \times 9 \times 8 \times 7 \times 6 = 30,240$ solutions. The space is still larger than the one induced by the binary representation because of all the duplicates. Note that, without imposing any other restriction, the solution representation allows for 120 (i.e., 5!) ways to represent the selection of the same set of regions. For instance, (1, 3, 4, 8, 9) is the same as (8, 3, 9, 1, 4), and it is also the same as any other permutation of the same numbers. If we divide 30,240 by 120, we reduce the size to 252, which is the same space defined by the binary representation.

The choice of a solution representation is guided by the notions of inclusion, connectivity, and efficiency. **Inclusion** refers to whether or not the resulting solution space includes all the feasible solutions to the original problem. In some cases, a solution representation may exclude (intentionally or unintentionally) some feasible solutions. This could simplify the search but may result in excluding some high quality solutions (including the global optimum). Indirect solution representations, described below, have the property of creating more manageable search spaces by sacrificing complete inclusion. **Connectivity** refers to whether or not there exists at least one path of moves that will connect all pairs of solutions in the solution space. For instance, if the representation is a binary vector and the move mechanism to explore the solution space consists of flipping the value of a single variable then it can be shown that all pairs of solutions are connected. In other words, any solution can be transformed into any other one by changing one value at a time. The length of the connecting path between the two solutions is equal to the number of variables that have different values and is called the Hamming distance. Search strategies may affect connectivity, for instance, when a search is not allowed to visit infeasible solutions. Finally, **efficiency** refers to the ease of transformation to move from one solution to another, the size of the induced search space, and the ease of evaluation of the objective function. A solution representation must be chosen in agreement with the move mechanisms that will be used to search the resulting space. In general, efficient representations are able to absorb some of the problem complexities. For instance, in Example 2.1, the binary representation does not limit the number of

selected regions to five. This means that a separate mechanism needs to be included in the search to distinguish between feasible and infeasible solutions. The discrete-value representation, on the other hand, implicitly enforces the constraint that limits the number of sales representatives to five. However, it results in a much larger solution space where only 0.25% of the solutions are unique and feasible.

Permutation vectors are popular solution representations in a variety of combinatorial optimization problems, including those related to job scheduling and vehicle routing. Consider the following example also adapted from [4].

Example 2.2 A printing shop must schedule ten jobs on a single machine. The processing times and due dates for each job are shown in Table 2.2. The jobs can be performed one after the other and there is no setup time involved. If jobs 1, 2, and 3 were scheduled first in that order, then job 1 would finish in 10 days, job 2 in 21 days, and job 3 in 28 days. Since the due dates for these jobs are 12, 35, and 20, respectively, only job 3 would be late (by 8 days) if the print shop were to use this sequence.

Table 2.2 Processing times and due dates for ten printing jobs

Job	Processing time (days)	Due date
1	10	12
2	11	35
3	7	20
4	5	27
5	3	23
6	7	36
7	5	40
8	5	40
9	12	55
10	11	47

A permutation is a natural and direct solution representation in this sequencing problem. A permutation creates a search space of $10! = 3,628,800$ solutions. In this case, the search space has a one-to-one mapping with the solution space. That is, the search space consists of all feasible solutions to the problem and it does not contain any duplicates.

Permutations are also used as indirect solution representations. For instance, suppose that instead of a single machine, the print shop in Example 2.1 would like to schedule the 10 jobs on two machines. A direct solution representation would need to account for the machine where jobs are assigned in addition to the order in which the jobs will be processed. Instead of using a complex solution representation to

account for these two pieces of information, a permutation could be employed as an indirect solution representation. Indirect representation require a mapping function or process to transform the representation into a solution to the problem. A simple transformation in the context of this assignment/sequencing problem may be to "assign the next job in the solution representation to the machine that becomes available first." Let us illustrate this rule to transform a permutation into a solution to the 2-machine problem.

Example 2.3 Suppose that a solution to the two-machine version of the problem described in Example 2.2 is given by (1, 3, 5, 4, 2, 6, 7, 10, 9, 8). Then, job 1 is arbitrarily assigned to machine 1 to start immediately. Job 3, is then assigned to machine 2, because this machine is available now and machine 1 will be available on day 10, after completing job 1. Machine 2 will be available first because job 3 requires 7 days. Therefore job 5 is assigned to machine 2. Job 4 is next in the sequence and it can be assigned to either machine because they both complete their current assignment on day 10. Suppose that machine 1 is arbitrarily selected and the process continues. At the end of the transformation process, the sequences are (1, 4, 6, 10, 8) and (3, 5, 2, 7, 9), for machines 1 and 2, respectively. Figure 2.2 shows a Gantt chart of this solution. In this case, the permutation results in a solution for which both machines are assigned the same number of jobs and both complete all the assigned jobs at the same time on day 38.

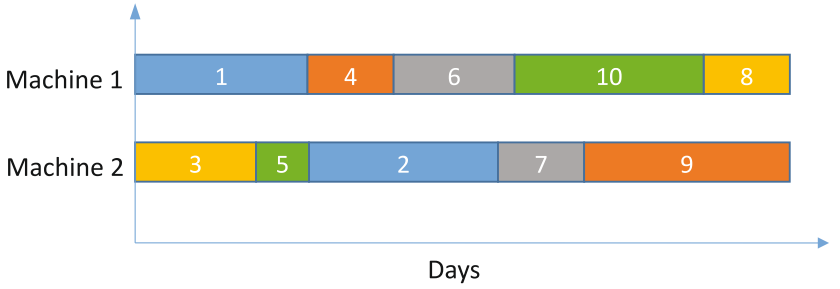

Fig. 2.2 Gantt chart of a 2-machine solution

An indirect solution representation is also referred to as encoding in some metaheuristic literature. The processes of transforming the representation into a solution of the problem is then referred to as decoding. In this nomenclature, the transformation rules are the decoder. The encoding in Example 2.3 is the ordering of the jobs (i.e., a permutation vector). The decoder is the assignment of the job to the first machine that becomes available. Implicit in this decoder is the notion that no job will wait for a machine to be available. Depending on the form of the objective

function value, this transformation has the potential of excluding high quality solutions. For instance, in some settings, schedulers are as concerned with jobs being late as they are of jobs being early. Tardy jobs may cause customer dissatisfaction, but early jobs incur in inventory cost, may become obsolete, and customers do not pay for them until past the deadline. This just-in-time philosophy translates in what is known in the scheduling literature as nonregular objective functions, because they are not always increasing with the completion time of each job. High quality solutions to a scheduling problem with a nonregular objective function may include idle times. Therefore, a decoder such as the one applied to Example 2.3 is likely to leave out of consideration a number of good solutions and limit the search to only those solutions that do not include idle times. The solutions with no idle time have low total tardiness cost but potentially large total earliness cost.

2.2 Objective Function

In an optimization problem, the objective function is the mathematical representation of the goal to achieve. The goal is stated as a minimization or a maximization of the objective function. The function associates a real value to each of the solutions in the solution space. This value indicates the quality of the solution. Clearly, larger values are associated with the best solution in a maximization problem and smaller values are an indication of higher solution quality in a minimization problem. The objective function plays an important role in guiding a metaheuristic search. When an indirect solution representation is used, the solution representation (encoding) must be transformed (decoded) into a solution of the problem in order to be evaluated by the objective function.

In many problems the objective function is relatively easy to formulate and its properties are such that it can be readily use to measure solution quality and serve as the main guide during the metaheuristic search process. In Example 2.1, the pharmaceutical company has the goal of selecting regions in order to be fair to all of its sales representatives. The company defines fairness as the uniform distribution of the total number of doctors among the sales representatives. More than one objective function may achieve this goal. Here are two possibilities for formulating an objective function to achieve the companys goal:

1. The variance of the number of doctors in each of the selected regions.
2. The difference between the maximum and the minimum number of doctors in the selected regions.

Both objective functions measure the dispersion of the number of doctors corresponding to the regions selected by a particular solution. Minimizing either one of these objective functions will guide an optimization process toward solutions that achieve the companys goal.

Regular objective functions associated with the sequencing problem in Example 2.2 are:

1. Number of tardy jobs.
2. Total number of days by which jobs are tardy.
3. Maximum number of days by which any job is tardy.

All these objective functions are to be minimized to meet customer satisfaction goals. The first one implies that clients would be equally dissatisfied if a job is late by one day or by multiple days. Using this objective function means that the print shop wants to minimize the number of dissatisfied clients. The second objective function assigns a "penalty" to each day that any job is tardy. So, the function attempts to minimize a total penalty. This function could be made client specific by adding a "tardiness penalty" to each job. Then the objective function would attempt to minimize the total weighted tardiness. The third objective function has a flavor of fairness. It attempts to minimize the worst case, i.e., the job that will be the tardiest. High quality solutions obtained with this objective function will tend to all jobs having about the same tardiness. A nonregular objective function could be formulated as an extension of the second objective function above by adding the total number of days by which jobs are early.

To analyze the effect that the objective function has on the search for an optimal solution, let us consider a reduced version of the scheduling problem in Example 2.2. Assume that instead of 10 jobs, the print shop has to schedule only the first 5 jobs. The search space defined by all permutations of these 5 jobs has a size of 120 solutions. We enumerate all these solutions and calculate the three regular objective function values and the nonregular objective function value described above. Table 2.3 shows the value of the optimal solution and the number of optimal solutions in the search space corresponding to each of the objective functions.

Table 2.3 shows that the first and the fourth objective functions are in the opposite end of the spectrum. That is, the solution space includes a fair number (17) of solutions that are optimal with respect to the number of tardy jobs. However, there is only one solution in the space that achieves the optimal value with respect to the total earliness and tardiness. In terms of metaheuristic search, it can be argued that it would be "easier" to find a solution that is optimal with respect to the first objective. Regarding the second and third objective functions, it can be conjectured that they both define equivalent objective value landscapes.

Table 2.3 Characteristics of optimal solutions for four objective functions

Objective Function	Optimal value	Number of optimal solutions
Number of tardy jobs	1	17
Maximum tardiness	1	3
Total tardiness	1	2
Total earliness and tardiness	11	1

The notion of objective function landscapes is important from the point of view of effectiveness of a metaheuristic search, as illustrated by the following example.

Example 2.4 Consider an optimization problem defined on a graph consisting of 6 nodes and 9 edges, as shown in Fig. 2.3. The numbers next to an edge represent a dissimilarity measure between the objects at each end of the edge. For instance, the dissimilarity between node A and node B is 6. The problem consists of selecting 3 nodes that are dissimilar. Two objective functions being considered are:

1. The sum of the dissimilarities between all the selected nodes.
2. The minimum node dissimilarity, where the dissimilarity of a node is calculated as the sum of the dissimilarities between itself and all the other selected nodes connected to it.

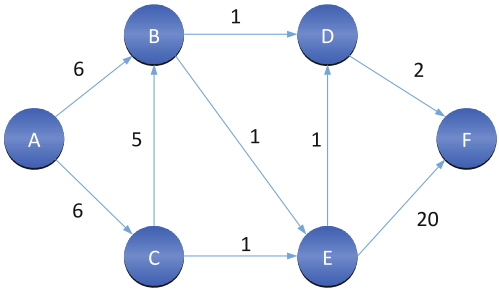

Fig. 2.3 Graph for Example 2.4

The first objective function is very common for these types of problems and it refers to a measure of dispersion efficiency. The second objective function is related to a class of problems for which equity is important and hence these problems belong to what is known as equitable dispersion. A solution to this problem can be represented as a discrete-value vector of size three, indicating the nodes that have been selected. For instance, the solution (A, B, C) represents the selections of nodes A, B, and C. The value of the first objective function (i.e., the sum of all the dissimilarities) is $6 + 5 + 6 = 17$. The value of the second objective function (i.e., the minimum node dissimilarity) is 11 because the dissimilarity of node A is $6 + 6 = 12$ and the dissimilarities of nodes B and C are $6 + 5 = 11$. These two objective functions create a significantly different objective function landscape, as shown in Figs. 2.4 and 2.5.

To create Fig. 2.4, we evaluated all 20 solutions to the problem employing the objective function that aggregates all the dissimilarities of the selected nodes. (Note that there are 20 unique ways of selecting 3 nodes out of 6.) The solutions in this

figure are lexicographically ordered. That is, the first solution is (A, B, C), the second is (A, B, D), the third is (A, B, E), and so on. Therefore, each solution in the figure differs from its immediate neighbor by exactly one value. Figure 2.5 is built in a similar way using the minimum node dissimilarity values.

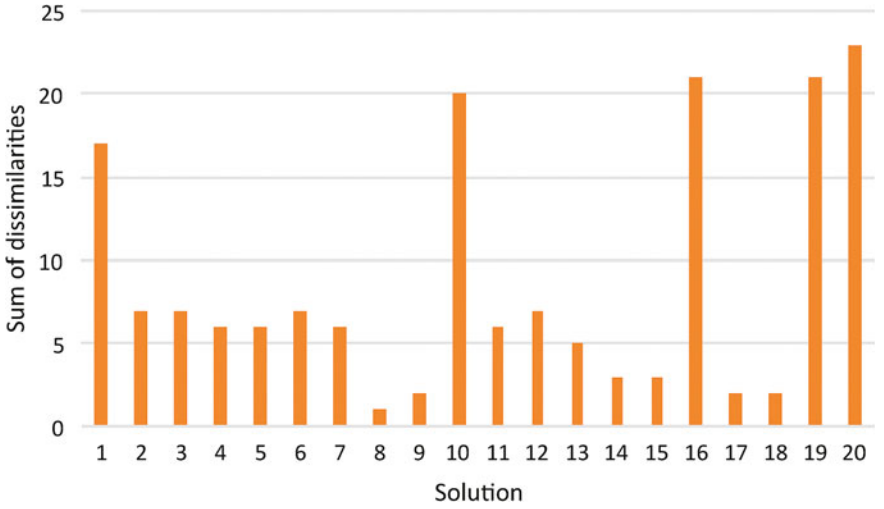

Fig. 2.4 Landscape of the sum of dissimilarities

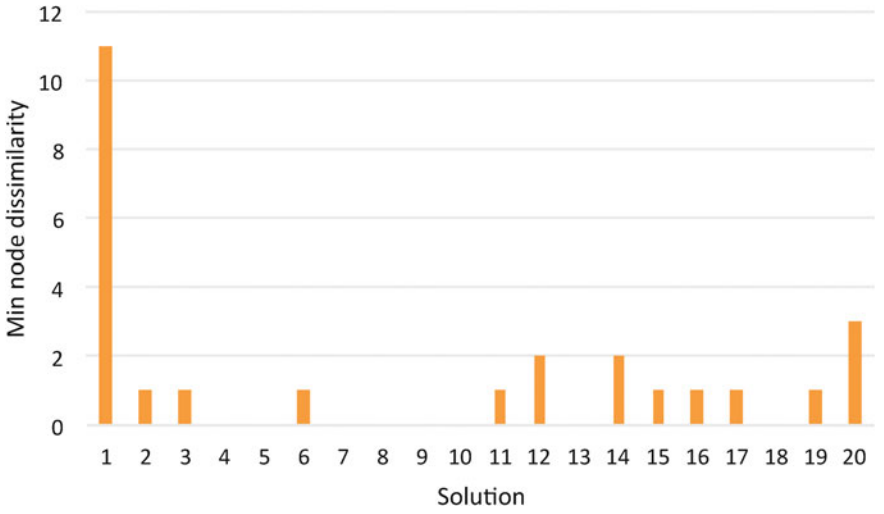

Fig. 2.5 Landscape of the minimum node dissimilarities

In the optimization jargon, the second objective function (see Fig. 2.5) is said to have a "flat" landscape. The flatness refers to a relatively large number of solutions with the same objective function value. In this case, 40% of the solutions (8 out of 20) have a value of zero. Spaces with flat landscapes are difficult to search because solutions are often surrounded by neighbors that have the same objective function value as the solution under consideration (also known as the current solution). That is, the objective function is not helpful for guiding the search and providing a direction where to move next. It is typical for a flat landscape to include one solution that clearly dominates all others, as in the case of Fig. 2.5, where the first solution (which is the global optimal) is significantly better than all the other ones.

The landscape in Fig. 2.4, on the other hand, is much more search-friendly. It includes 4 high quality solutions (10, 16, 19, and 20) and a relatively small percentage of solutions with the same value in a given region. While searching in a flat landscape is similar to trying to find a "needle in a haystack", searching in a space with an associated rugged landscape makes the use of the objective function a valuable tool for finding search directions. Landscape analysis (even a basic one) is recommended in order to identify effective search mechanisms and to select the right methods and strategies. Generally, flat landscapes suggest that a search method should favor construction of solutions over neighborhood explorations.

2.3 Constraint Handling

Constraints are common in optimization problems. Arguably, all optimization problems include constraints in one way or another. In this section, however, we focus on constraints that require special attention and that cannot be implicitly taken into consideration by a clever choice of a solution representation. Let us explore this notion by revisiting Example 2.1. As stated, the problem has two constraints:

1. There are five sales representatives and therefore exactly five regions must be selected.
2. The company wishes to visit the doctors in all counties, therefore all counties must be "covered" by at least one region.

The first is a so-called "equality" constraint because feasible solutions must choose exactly five regions. The second is a "greater than or equal" constraint because the number of regions covering a county must be "at least" one. The binary solution representation does not implicitly account for either one of these constraints and therefore the solution procedure must be designed to handle them. Let us define a binary decision variable x_i to represent the choice of whether or not region i is selected. That is, $x_i = 1$ if region i is selected and zero otherwise. A solution to the problem is then represented as follows:

$$(x_1, x_2, \ldots, x_{10})$$

Give the two constraints in the problem, a feasible solution must satisfy the following equations:

$$
\begin{aligned}
&x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} = 5 \\
&x_1 + x_3 + x_5 \geq 1 \\
&x_2 + x_4 + x_6 + x_8 \geq 1 \\
&x_1 + x_3 + x_7 + x_{10} \geq 1 \\
&x_2 + x_4 + x_9 \geq 1 \\
&x_1 + x_5 + x_8 \geq 1 \\
&x_2 + x_6 + x_{10} \geq 1 \\
&x_3 + x_7 + x_9 \geq 1 \\
&x_2 + x_5 + x_8 \geq 1 \\
&x_4 + x_7 + x_9 \geq 1 \\
&x_3 + x_6 + x_{10} \geq 1
\end{aligned}
$$

The first equation enforces exactly 5 of the binary variables to be one in any feasible solution. The following 10 equations enforce the coverage of the counties, with one equation corresponding to one county. For example, the first county can be covered by the selection of either region 1, 3, or 5. The second county can be covered by the selection of either one of four regions: 2, 4, 6, and 8. The interpretation of the other 8 equations is similar. If the problem is modeled this way, a solution procedure, when selecting regions, must verify that the selection satisfies all of these equations.

In the metaheuristic literature, the sequencing problem in Example 2.2 is considered unconstrained. Solutions are represented by a permutation vector and any permutation is a solution to the problem. However, if we model the problem by defining a decision variable x_i to indicate the identity of the job that will be processed in position i, the permutation representation implicitly handles the constraint that all x values must be different from each other. In other words, for the sequencing problem of Example 2.2, the permutation representation solves a problem that can be expressed as:

Decision variables: x_1, x_2, x_{10},
$1 \leq x_i \leq 10$ for $i = 1, \ldots, 10$,
All different.

There are three main constraint handling strategies used within metaheuristic methodologies: rejection, penalization, and repair. The **rejection** strategy consists of discarding any solution that does not meet all the constraints. For instance, suppose that in the problem of Example 2.1 a metaheuristic search is examining the solution (1, 2, 3, 5, 6), where the numbers represent the selected regions. The first constraint is clearly satisfied because the set consists of five distinct regions. The procedure

then must examine whether the solution meets all the covering constraints. This examination reveals that all counties are cover by at least one region, except the 9th county. This county can be covered by regions 4, 7, and 9, but none of these regions has been selected in the trial solution under examination. If a rejection strategy is in place, the solution is then discarded. The downside of rejection is that the procedure does not "learn" anything about the infeasibility related to the discarded solution. Also, since the solution is discarded and there is no record that the solution was already examined, the strategy could result in a situation where the search is just "spinning wheels" instead of making progress toward finding high quality solutions. In general, a rejection strategy is only used when the percentage of infeasible solutions in the search space is relatively small.

Penalization is the process by which the value of an infeasible solution is altered by a function that imposes a penalty for violating problem constraints. Let the objective function value for a solution x be given by $f(x)$. Then, a linear penalty function has the following form:

$$f'(x) = f(x) + p \times c(x)$$

In this penalized objective function value $f'(x)$, the value of p is the penalty factor and $c(x)$ represents a measure of constraint violation associated with solution x. Note that for feasible solution $c(x) = 0$ and therefore $f'(x) = f(x)$. The penalty factor must be such that no infeasible solution ends up with a $f'(x)$ value that is better (i.e., smaller in a minimization problem or larger in a maximization problem) than the objective function value of a feasible solution. A special form of penalization that is in a sense equivalent to the rejection strategy consists of assigning to all infeasible solutions in a minimization problem a very large $f'(x)$ value (denoted by `LARGE`) in the following way:

$$f'(x) = \begin{cases} f(x) & \text{if } x \text{ is feasible} \\ \texttt{LARGE} & \text{if } x \text{ is infeasible} \end{cases}$$

A negative large value is assigned when the goal is to maximize the objective function value. This special form does not distinguish among infeasible solutions and has no capacity to exploit information related to solutions that are "close" to being feasible.

An appropriate value of p is typically found through experimentation. Knowledge of the problem context also helps in the selection of penalty factors. Referring back to Example 2.1, let us assume that the search process is set up in such a way that all the solutions that it generates are feasible with respect to the selection of regions. That is, the process produces solutions for which there are always five different regions selected. Now, we would like to use a penalty function to distinguish between the solutions that are feasible with respect to county coverage and the ones that are not. Also assume that we are using the range of the number of doctors assigned to each selected region as the objective function value. To find a suitable value for p, we can calculate a value that we know no feasible solution will ever

reach. The problem data (see Table 2.1) show that county 4 has the smallest number of doctors (52). The data also show that there are 1,050 doctors in total. Therefore, the maximum possible range would be the result of assigning county 4 to one sales representative and all the other counties to another representative. The range value is the $1,050 - 52 = 998$. This is a very conservative value for the range, because we know that there are five sales representative and not just two. However, it is an easy calculation that does not involve additional analysis and refinements. If in addition, we define $c(x)$ as the number of uncovered counties, then the penalized objective function for Example 2.1 has the following form:

$$f'(x) = f(x) + 998 \times c(x)$$

This function guarantees that all infeasible solutions will have a penalized objective function value that is larger than the objective function value of any feasible solution. The smallest value of the penalty term is 998 and it occurs when $c(x) = 1$.

The constraints in Example 2.1 are of a special type that indicate whether or not a county is covered. Covering a county by more than one region does not provide information on the degree of feasibility of a solution. If a county is not covered, the left hand side of the equation is zero and this is true for all uncovered counties. Hence, there is no information that can be obtained from a single constraint on the degree of infeasibility of the solution being examined. The only meaningful information is the number of counties that are uncovered and this is why we defined $c(x)$ the way we did above.

There are constrained optimization problems for which a constraint that is violated can produce information on the degree of the violation. Consider a constraint for which the left hand side represents resources (e.g., money, people, machines, etc.) used by a solution x, denoted by $r(x)$, and the right hand side is a limit on the available resources, denoted by b. Then, the violation v of the constraint can be calculated as follows:

$$r(x) \leq b + v$$

In this case, v is an absolute violation value that is zero if the resources $r(x)$ needed by the solution x do not exceed the budget b. A relative measure could be more effective in situations where the problem contains multiple constraints with a ranges of values that are significantly different. A relative measure could have the following form:

$$r(x) \leq b(1 + v)$$

The advantage of a relative measure is that all the violation values are expressed as a fraction of the b values. This assumes that all violations are equally important unless individual p values are used for each constraint violation. A simple penalty function can be formulated by calculating $c(x)$ as the sum of the violations (either absolute or relative).

Repair strategies consist of the transformation of an infeasible solution into a feasible solution. The goal of a repair procedure should be to create a feasible solution out of an infeasible solution while keeping the feasible solution "as close as possible" as the originating infeasible solution. In other words, the idea is to make the least number of changes to the infeasible solution while transforming it into a feasible solution. Another goal, perhaps equally important as the first one, is that the transformation produces a feasible solution of the highest possible quality. Therefore, when changing elements or altering variable values to move an infeasible solution toward the feasibility region, the transformation rules should take advantage of problem context to make changes with the highest merit according to the objective function value. Greedy approaches that transform infeasible solutions into feasible solution one step at a time attempt to make changes that either improve the most or deteriorate the least the objective function value of the infeasible solution being transformed.

Example 2.5 Consider the region selection problem of Example 2.1. Assume that the objective function is the variance of the number of doctors in each of the selected regions. An optimization process produces the solution (1, 2, 3, 5, 6), which as stated above, it is infeasible because county 9 is not covered by any of the selected regions. Table 2.4 shows the characteristics of this solution.

Table 2.4 Characteristics of (1, 2, 3, 5, 6) solution

Region	Counties covered by selected region				Doctors
1	1	3	5		157
2	2	4	6	8	202
3	1	3	7		232
5	1	5	8		161
6	2	6	10		170

The objective function value of the solution in Table 2.4 is 820.74. Suppose that the repair procedure is such that it replaces, one at a time, selected regions with unselected regions until the solution becomes feasible. In order to repair this solution following such a process, we must identify candidate regions capable of covering the uncovered county 9. The candidates are regions 4, 7, and 9. We now could try all replacements and keep the one that repairs the solution and has the best objective function value. There are 15 possible replacement, as shown in Table 2.5. All but one of the resulting solutions are feasible. Solution 3 is not feasible because it does not cover county 7, which was covered only by region 3 in the original solution and this region is replaced by region 4 that does not include county 7 in the set of counties that it can cover.

Table 2.5 Solutions obtained by the repair procedure

Solution	Selected regions					Objective function value
1	2	3	4	5	6	3104.00
2	1	3	4	5	6	791.73
3	1	2	4	5	6	(*)527.67
4	1	2	3	4	6	1472.77
5	1	2	3	4	5	2645.17
6	2	3	5	6	7	778.30
7	1	3	5	6	7	1891.91
8	1	2	5	6	7	1001.90
9	1	2	3	6	7	851.80
10	1	2	3	5	7	3446.60
11	2	3	5	6	9	1336.30
12	1	3	5	6	9	1520.73
13	1	2	5	6	9	752.30
14	1	2	3	6	9	757.40
15	1	2	3	5	9	2460.47

*Infeasible solution, county 7 is not covered by any of the selected regions

The repair procedure, as performed in Table 2.5, identifies the replacement of region 5 with region 9 as the best change to transform the infeasible solution (1, 2, 3, 5, 6) with objective function value of 820.74 into the feasible solution (1, 2, 5, 6, 9) with objective function value of 752.30. In this case, a feasible solution was obtained by performing one step of the repair procedure. In larger problems, the repair procedure might need more than one step. In such a case, an infeasibility measure is needed to be able to choose moves that take the current solution closer to the feasibility region. In this example, we are able to select solution 13 in Table 2.5 solely based on its objective function value because there are several feasible solutions from which to choose. In the absence of a feasible solution, the rule should be such that the solution that is "least infeasible" should be chosen. We point out that repair mechanisms may fail even when the selection rule attempts to minimize the infeasibility of the current solution. In such a case, the solution may be discarded.

Some metaheuristic methods are designed to preserve feasibility. This means that once a feasible solution is found, the procedure makes only those moves that transform one feasible solution into another. For instance, in Example 2.5, once the search moves to solution (1, 2, 5, 6, 9), a feasibility preserving method would consider only those moves that replace regions in such a way that the resulting set covers all the counties. For efficiency purposes, the procedure would first verify the feasibility of the move (in this case, the replacement of a region) and then it would calculate the objective function value of the resulting solution as long as it is feasible.

2.4 Design Choices and Parameter Tuning

Metaheuristic methodologies provide a framework to create procedures to search for high quality solutions in spaces defined by the formulation of optimization problems. There are two main search spaces: discrete and continuous. Discrete spaces are typically associated with combinatorial optimization problems, for example, those for which the decision variables are represented by a set of discrete values, a binary, or a permutation vector. Continuous spaces are defined by a set of real (continuous) variables. There are also complex problems that involve mixed search spaces characterized by a combination of discrete and continuous variables.

Researchers have concluded that no single methodology is capable of dominating any other one in all problem settings and search spaces. While there is no scientific way of showing that one methodology is better than another one in general, the scientific process can be applied to show that a procedure emanating from a metaheuristic framework can perform better than another procedure, from the same metaheuristic framework or another one, in a particular class of problems. This means that when comparing solution methods, we are not comparing methodologies but rather particular implementations of methodological frameworks.

Each metaheuristic methodology includes a set of strategy and a general "search philosophy." As mentioned in Chap. 1, some methodologies are based on exploring the search space by moving from one solution to another. Other methodologies construct and maintain a population of solutions that continue to change (evolve) in stages. This evolution occurs according to rules and strategies to transform solutions and to include and discard them from the population. There are differences within single-solution based methodologies in the same way that not all the population-based methodologies are the same. For example, some rely more in randomization than others.

An instantiation of a methodology is its application to a particular problem or class of problems. Two applications of the same methodology to the same problem does not necessarily result in the same search procedure. The reason is that a metaheuristic implementation is the result of design choices and parameter tuning. The analyst decides which elements from the methodology to include in the implementation and how these elements will interact. The analyst also decides how to calibrate the performance of his/her implementation and therefore how to adjust the parameter values that ultimately determine the efficiency of the procedure.

Example 2.6 Consider the sequencing problem in Example 2.2 and assume that we want to optimize the nonregular objective function that penalizes both earliness and tardiness. Specifically, let p_j and d_j be the processing time and the due date, respectively, of job j. Also, let s be a solution to the problem represented as the ordering of the jobs. That is, s is a permutation for which $s(j)$ is the index of the job in position j. Then, the just-in-time objective function can be formulated as follows:

$$f(s) = \sum_{j=1}^{n} |c\,(s(j)) - d\,(s(j))|$$

In this formula, $c\,(s(j))$ represents the completion time of job in position j in the sequence, and $||$ is the absolute value function. This means that either positive or negative deviations from the due date count equally toward the total sum. The completion time of the job in position j is calculated with the following formula:

$$c\,(s(j)) = \sum_{i=1}^{j} p\,(s(i))$$

What this formula states is that the completion time for a job in position j is given by the sum of the processing time of all the jobs in positions before j plus the processing time of the job in position j. Table 2.6 shows the calculation of the objective function value of the solution given by ordering the jobs in Table 2.2 by increasing value of their due dates. This ordering is called the EDD (earliest due date) rule in the scheduling literature and in this case results in $s = (1, 3, 5, 4, 2, 6, 7, 8, 10, 9)$.

Table 2.6 Objective function value for EDD solution

Job	Processing	Due date	Completion	Deviation
1	10	12	10	2
3	7	20	17	3
5	3	23	20	3
4	5	27	25	2
2	11	35	36	1
6	7	36	43	7
7	5	40	48	8
8	5	40	53	13
10	11	47	64	17
9	12	55	76	21
Objective function value:				77

The jobs in Table 2.6 are EDD ordered, as seen by the increasing values in the due date column. The completion time is the accumulated processing time, including all previous jobs and the current job. The deviation is the absolution difference between the due date and the completion time. Finally, the objective function value is the sum of all completions.

One of the first design decisions that an analysis faces is how to initialize the search. For instance, Analyst A might decide that the search will start from a random solution, which in Example 2.6 would correspond to a random sequence. Analyst B, who could be basing his/her implementation on the same methodology as the Analyst A, might decide to initiate the search from the EDD solution shown in

Table 2.6. Both procedures, A and B, could be based on the same principles but the initialization of the search has been implanted in two different ways and therefore could lead to different results.

Then, the analysts must decide how solutions are going to be transformed during the search. This transformation process is referred to as a move mechanism, or simply a **move**. Two typical moves in permutation spaces are insertions and swaps. An **insertion** is the transferring of a single job from its current position to a new one. For instance, consider the solution in Table 2.6. Job 5 is currently in position 3, as shown in the following solution representation:

$$(1, 3, 5, 4, 2, 6, 7, 8, 10, 9)$$

An insertion move that takes job 5 from its current position and moves it to the sixth position results in the following solution:

$$(1, 3, 4, 2, 6, 5, 7, 8, 10, 9)$$

Note that this move caused jobs 4, 2, and 6 to "shift to the left" one position. These jobs used to be in position 4, 5, and, 6, but now they are in positions 3, 4, and 5. As intended, job 5 now occupies position 6. The objective function associated with the new solution is 95. As defined in Sect. 1.3, the move value is the difference between the objective function value before the move and the objective function value after the move. In this example, the move value is 18. Since the goal is to minimize the objective function value, a positive move value indicates a move to an inferior solution.

Now, let us consider a swap instead of an insert. A swap is an exchange of the positions of two elements (jobs, in our example). For instance, the swap of jobs 5 and 6 in the solution of Table 2.6 results in the following solution:

$$(1, 3, 6, 4, 2, 5, 7, 8, 10, 9)$$

The objective function associated with this solution is 103 for a move value of 26. A search procedure does not have to be limited to either swaps of inserts. For example, a procedure could be designed to perform inserts for a number of initial search steps and then perform swaps for the remainder of the search. Another design could be such that inserts and swaps alternate with certain periodicity. There are many different design choices even for a relatively simple search such as the one illustrated here.

When design choices involve parameters, a tuning process is necessary to configure a procedure. A **search parameter** is an input value to the procedure that is controlled by the analyst and that has an effect on the behavior of the procedure and ultimately on its efficiency and effectiveness. For instance, in the job sequencing problem, a parameter `INIT` could indicate the number of initial iterations (i.e., moves executed during the search) in which the procedure is going to perform inserts before permanently switching to performing swap as a mechanism

to search the solution space. The value of INIT might not be immediately obvious and careful experimentation is typically necessary to find effective search parameter values.

Referring back to our Chap. 1 discussion on model simplicity and elegance, as a general rule, the simpler models are those with fewer adjustable parameters. The more parameters the harder it is to find a combination of values that make the procedure work well across many instances of the problem that is designed to solve. It is not hard to begin with a simple deign and rapidly move to levels of complexity that require more and more parameters. For instance, one possibility in the sequencing problem is to limit the "jump" that any job makes in a single move. The jump is the difference between the current position of the job and the position that occupies after the move. Limiting the jump could improve the efficiency of the solution procedure, since it is unlikely that transferring a job too far from a position that is natural for its due date will result in a high-quality solution. While the rationale makes sense, the implementation adds a layer of complexity since we now will need to adjust a parameter JUMP indicating the maximum distance (measured in terms of number of positions) that a job is allowed to travel in a single move.

Compromises are often quite effective in implementing a sound idea and avoiding the added complexity. For instance, inserts and swaps in the sequencing problem could be limited to immediate positions. This is the same as setting JUMP equal to one. The effect of such a choice is twofold. On one hand, the JUMP parameter is no longer an adjustable value. And, on the other hand, the size of the neighborhood (i.e., the number of moves to explore around a given solution) decreases. When jobs are only allowed to move no more than one position away from its current position, inserts and swaps are equivalent, rendering the parameter INIT unnecessary. The number of moves is given by $n - 1$, where n is the number of jobs. However, when the distance is not limited at all (i.e., JUMP is set to n), the number of swaps is given by the following formula:

$$\frac{n^2 - n}{2}$$

Therefore, there are 45 possible swaps in a problem with 10 jobs, but there are only 9 swaps of jobs that are in immediate positions. The nine neighboring solutions that can be reached from the solution in Table 2.6 are shown in Table 2.7. The first row shows the current solution and the following nine rows show the neighbors that can be reached by performing swaps of jobs in immediate positions. The neighbors are labeled N1 to N9, assuming a so-called lexicographic ordering of the swap exploration.

As shown in Table 2.7, the swap of jobs 6 and 7 is the best move with a value of –2, because the objective function value (last column in the table, labeled *O.F.*) decreases from 77 to 75. This swap results in the solution labeled as N6 and highlighted in bold in Table 2.7. A design issue related to the exploration of a neighborhood relates to the order in which the moves are examined. This is relevant when the neighborhoods are very large (i.e., when they include many moves) and a

first-improving strategy is implemented. This strategy consists of executing a move as soon as an improvement is identified. For example, consider the neighborhood in Table 2.7. The procedure starts by swapping the jobs in positions 1 and 2 to examine the first neighbor (labeled N1). Then, it swaps the jobs in positions 2 and 3 to examine N2. In the first-improving strategy, the procedure will stop the neighborhood exploration as soon as it reaches N6 because this neighbor is better than the current solution. The move is executed and the search moves to solution N6. That is, N6 becomes the new current solution. The next exploration starts by attempting to swap the jobs in positions 7 and 8. Then, it continues with 8 and 9, 9 and 10, and it "wraps around" to 1 and 2. This creates a circular list for the ordering of the exploration. The process stops either when an improvement is found or when the entire 9 moves have been explored and no improvement has been identified. At this point, a decision, which depends on the methodology being applied, is made to move the search to another point.

Table 2.7 Reduced swap neighborhood

Solution	Sequence										O.F.
Current	1	3	5	4	2	6	7	8	10	9	77
N1	3	1	5	4	2	6	7	8	10	9	90
N2	1	5	3	4	2	6	7	8	10	9	81
N3	1	3	4	5	2	6	7	8	10	9	79
N4	1	3	5	2	4	6	7	8	10	9	87
N5	1	3	5	4	6	2	7	8	10	9	81
N6	**1**	**3**	**5**	**4**	**2**	**7**	**6**	**8**	**10**	**9**	**75**
N7	1	3	5	4	2	6	8	7	10	9	77
N8	1	3	5	4	2	6	7	10	8	9	83
N9	1	3	5	4	2	6	7	8	9	10	78

*Infeasible solution, county 7 is not covered by any of the selected regions

The best-improving strategy is an alternative to first-improving. In best improving, the order of the exploration is not important because all moves are evaluated and a decision where to go next is made after the quality of all neighbors has been determined. The advantage of a first-improving strategy is that the search has the potential to move faster to a new solution, without the need for evaluating the entire neighborhood. The disadvantage is that a large improvement might be missed due to the ordering in which the moves are explored. On the other hand, the advantage of the best-improving strategy is that the largest improvement is always identified, moving the search more rapidly to a local optimum. The disadvantage is that, for large neighborhoods, identifying the best improving move requires a considerable amount of computational resources.

Researchers have dealt and continue to deal with issues related to design choices and parameter tuning. Much of the effort has been applied to developing automated systems that could perform these tasks with little or no human involvement.

Automated tuning systems have been proposed based on the general area of design of experiments. These systems search for the combination of parameter values that make the procedure more likely to perform at its highest level when tackling a given class of problems. CALIBRA[1] is an example of a parameter tuning system based on experimental design procedures (see [1]). A step further in this line of research includes systems that not only search for the best combination of parameter values but also for the best combination of search elements. In other words, the analysts do not have to commit to a particular design or strategy (e.g., selecting between first-improving and best-improving) and instead they may include all choices and let an automated design engine select what is best for particular problem classes and at particular stages of the search process. Programming by Optimization, PbO,[2] is a prominent example of such a system (see [2]:

> "Premature commitment to design choices during software development often leads to loss of performance and limited flexibility. Programming by Optimization (PbO) is a design paradigm that aims to avoid such premature design choices and to actively develop promising alternatives for parts of the design. Rather than building a single program for a given purpose, software developers specify a rich and potentially large design space of programs. From this specification, programs that perform well in a given context are generated automatically through optimization techniques. PbO allows human experts to focus on the creative task of imagining possible mechanisms for solving given problems or subproblems, while the tedious job of determining what works best in a particular context is performed automatically, substituting human labor with computation."

Research in this area will continue, fueled by the promising results obtained by systems such as PbO. What seems to be clear at the moment is that the task of designing strategies and procedural elements that can be used to search solution spaces will still be the responsibility of a human analyst.

2.5 Exercises

1. The following questions relate to the problem described in Example 2.1:
    a. Develop a spreadsheet model of the problem using the binary representation of the decision variables and the variance of the number of doctors assigned to each region as the objective function. Use the `VAR.P` formula to calculate the variance.
    b. Verify that the spreadsheet model is correct by checking that the solution (0, 1, 1, 0, 1, 1, 1, 0, 0, 0) results in a variance of 778.3.
    c. Use the Evolutionary Solver in Excel to search for a high quality solution to this problem. Note that that an optimization model needs to be formulated in order to use the Evolutionary Solver. This entails declaring an objective

[1] http://coruxa.epsig.uniovi.es/~adenso/file_d.html.

[2] http://www.prog-by-opt.net/.

function, an optimization goal (maximize or minimize), a set of decision variables (changing cells), and constraints. What solution do you obtain?

d. On a new spreadsheet, develop a spreadsheet model using the discrete-value representation of the decision variables and the variance of the number of doctors assigned to each region as the objective function. Use the `VAR.P` formula to calculate the variance.
e. Verify that the spreadsheet model is correct by checking that the solution (2, 3, 5, 6, 7) results in a variance of 778.3.
f. Use the Evolutionary Solver in Excel to search for a high quality solution to this problem. Compare the solution obtained with this model and the solution obtained with the binary model.

2. The following questions relate to the problem in Example 2.2:

a. Develop a spreadsheet model of the problem by creating 10 cells for the processing ordering. That is, these cells represent a permutation of the jobs and therefore they represent the solution to the problem.
b. In three separate cells, calculate the following performance metrics: (1) number of tardy jobs, (2) total tardiness (i.e., the sum of all days by which jobs are tardy), and (3) total tardiness plus total earliness (i.e., the sum of all the days by which completion times deviate from the due dates).
c. Verify that the spreadsheet model is correct by checking that the solution (1, 3, 5, 4, 2, 6, 7, 8, 9, 10) results in 6 tardy jobs, a total tardiness of 68, and total deviation from the dues dates of 78.
d. Use the Evolutionary Solver in Excel to search for a high quality solution to this problem. Note that an "All different" (dif) constraint must be specified to declare that the decision variables represent a permutation. Run the solver three times, once for each of the objective functions. Compare the solutions found by these three executions of the solver.

3. Figure 2.6 shows ten elements represented by nodes that are labeled A to J. An edge between a pair of nodes indicates the degree of the dissimilarity between the nodes connected to the edge. The larger the value the more dissimilar the elements that are connected by the edge. The problem consists of selecting 5 elements in order to maximize a measure of dissimilarity. Two measures are being considered: (1) the sum of the dissimilarities of all the selected elements and (2) the minimum node dissimilarity, where the dissimilarity of a node is the sum of all the dissimilarities of the selected nodes connected to it. This situation may arise when selecting a team out of a pool of individuals. The goal is to select a diverse team, where diversity between two individuals may be a value that summarizes differences such as gender, ethnicity, level of education, or national origin. Selecting a team guided by the first diversity measure described above produces a set of individuals that collectively are the most diverse. However, the solution could result in a team where only a few individuals are very diverse with respect to each other, with the rest of them being fairly similar. Maximizing the

second diversity measure avoids this situation and creates some diversity equity among all team members.

a. Develop a spreadsheet model of the problem using a binary representation for the decision variables and creating objective function cells for both dissimilarity measures.
b. Verify that the spreadsheet model is correct by checking that the solution (A, B, G, I, J) results in a total dissimilarity of 11 and a minimum node dissimilarity of 2, corresponding to element J.
c. Use the Evolutionary Solver in Excel to search for a high quality solution to this problem. First find a high quality solution with respect to the first objective function and then find one with respect to the second objective function. Are these solutions the same?

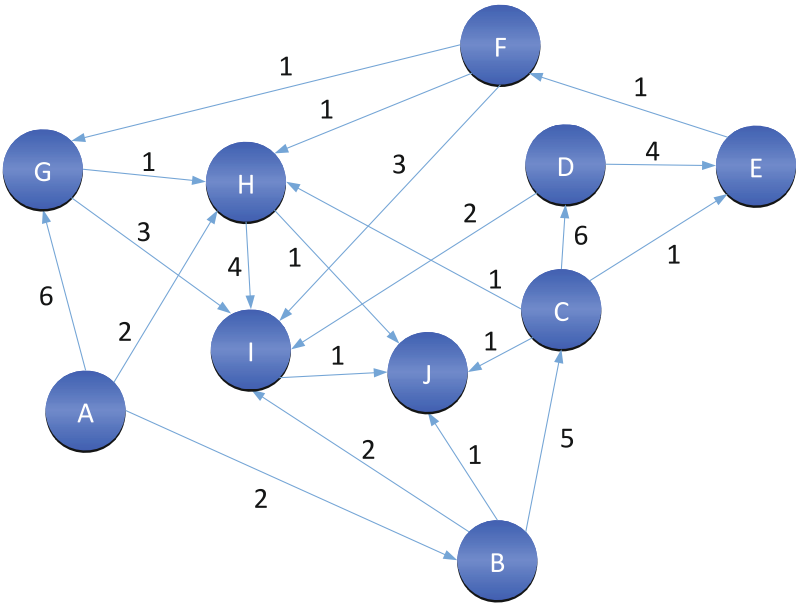

Fig. 2.6 Graph for Exercise 3

4. (Adapted from [4]). A small but growing company in the food industry is considering entering the breakfast market. In particular, it would like to add cold cereal products. Before developing its own line of cereals, the company wants to identify if there are different market segments. Specifically, the company would like to figure out if there is a health-conscious segment and a taste segment. The company has collected per-serving nutritional profiles for 74 different cold cereals currently in the market. The nutritional profile consists of the values of nine nutritional components: calories, protein, fat, sodium, fiber, carbs, sugar, potas-

sium, and vitamins. The data are in the *Exercise2-4.xlsx* file that accompanies this book. The problem of identifying market segments is typically approached with a technique known as cluster analysis, which is one of the data mining tools in business analytics. Cluster analysis consists of grouping observations into clusters and measuring the distance between the observations in each cluster. The goal is to find clusters of observations that share similar attributes. The merit of a particular clustering is typically measured by the total distance between each observation and its cluster centroid. The centroid of a cluster is the average value of each attribute for all the observations assigned to the cluster.

a. Develop a spreadsheet model of the problem using a binary representation for the decision variables and creating an objective function cell that calculates the total distance between each observation and the centroid of the cluster to which it is assigned. Note that there are 74 decision variables in this problem, one for each observation. A zero indicates that the observation is assigned to cluster 0 and a one indicates that the observation is assigned to cluster 1.
b. Verify that the spreadsheet model is correct by checking that the solution that assigns the first 37 observations to cluster 0 and the remaining observations to cluster 1 results in a total distance of 12,774.6.
c. Use the Evolutionary Solver in Excel to search for a high quality solution to this problem. Compare the Nutritional profiles of the centroids of the clusters in the best solution found by the solver. Can you identify two different market segments?

References

1. Adenso-Díaz, B., and M. Laguna. 2006. Fine-tuning of algorithms using partial experimental designs and local search. *Operations Research* 54 (1): 99–114.
2. Hoos, H.H. 2012. Programming by optimization. *Communications of the ACM* 55 (2): 70–80.
3. Laguna, M., J. Molina, F. Pérez, R. Caballero, and A. Hernández-Díaz, 2010. The challenge of optimizing expensive black boxes: a scatter search/rough set theory approach. *Journal of the Operational Research Society* 61 (1): 53–67.
4. Ragsdale, C. 2015. *Spreadsheet modeling and decision analysis: a practical introduction to business analytics*. Boston: Cengage Learning.
5. Talbi, E-G. 2009. *Metaheuristics: From design to implementation*. New Jersey: Wiley.

http://www.springer.com/978-3-319-68117-7

Metaheuristics for Business Analytics
A Decision Modeling Approach
Duarte, A.; Laguna, M.; Marti, R.
2018, X, 136 p. 25 illus., 2 illus. in color., Hardcover
ISBN: 978-3-319-68117-7