Each deme runs a sequential EA, and the individuals are able to interact only with other individuals in the deme.

An additional *migration* operator is defined: occasionally some selected individuals are exchanged among subpopulations, introducing a new source of diversity in the EA (see middle of Figure 2.14).

- The *cellular* model considers an underlying spatial structure for the individuals in the population, most usually a two-dimensional toroidal grid. The interactions in the evolutionary search are restricted only to neighboring solutions. The propagation of good characteristics in the solutions follows the *diffusion* model, gradually spreading through the grid.

  The limited interaction between individuals is useful to provide diversity in the population, often improving the efficacy of the evolutionary search (see right of Figure 2.14).

The evolutionary algorithm used in this work to solve the the problem presented in Section 3.1 follows the *distributed subpopulation model.*

2.1.3.2 Greedy Randomized Adaptive Search Procedure

The Greedy Randomized Adaptive Search Procedure (GRASP) is a well known metaheuristic, which has been applied for solving many hard combinatorial optimization problems with very good results, see: [de Aragão 2001, Festa 2004, Mavridou 1998, Martins 2000, Pardalos 1999, Rosseti 2001, Resende 1998a, Resende 1998b, Ribeiro 2002]. Extensive surveys of the associated literature are given in [Feo 1995, Resende 2003, Festa 2004].

Before describing the main ideas of GRASP, we formulate a generic combinatorial optimization problem based on the description introduced in [Resende 2003]. Let us consider:

i) $N = \{n_1, \dots, n_m\}$ is the finite basic set containing the potential elements which will be able to integrate a feasible solution.

ii) F denotes the set of feasible solutions satisfying: $F \subseteq 2^N$.

iii) $f : 2^N \to \mathbb{R}$ is the objective function. Without loss of generality, we assume the minimization version, i.e. the aim is to find a global optimal solution $S^* \in F$ such that $f(S^*) \leq f(S)$, $\forall S \in F$.

These points will be determined, when specifying the optimization problem to be studied. For example, in the case of the Minimum Vertex Covering Problem:

- $N = \{v_1, \dots, v_n\}$ is the set of nodes to be considered,
- E is the set of edges connecting the nodes of N,

- F is composed of all the subsets of N such that when $S \in F$ any edge in E has at least one endpoint in S,
- $f(S)$ is the number of nodes belonging to S.

A GRASP is an iterative process, where each iteration consists of two phases: *construction* and *local search*. The *construction phase* builds a feasible solution, whose neighborhood (in some sense to be defined when adapting the method to each specific problem) is explored during the second phase, looking for an improvement. The best solution over all GRASP iterations is returned as the result.

Algorithm 3 GRASP pseudo-code

```
Procedure GRASP($ListSize$,$MaxIter$,$Seed$):
1: $Read\_Input\_Instance()$;
2: for $k = 1$ to $MaxIter$ do
3:   $InitialSolution \leftarrow Construct\_GRSol(ListSize, Seed)$;
4:   $LocalSearchSolution \leftarrow Local\_Search(InitialSolution)$;
5:   if $cost(LocalSearchSolution) < cost(BestSolutionFound)$ then
6:     $Update\_Solution(BestSolutionFound, LocalSearchSolution)$;
7: return $BestSolutionFound$;
```

We describe now a generic GRASP implementation, whose pseudo-code can be seen in Algorithm 3. This generic implementation serves as a template to be mapped into the problems introduced in Section 4.2, where specific GRASP methods customized to our problems are proposed.

The GRASP heuristic has three main parameters: the number of iterations $MaxIter$, the candidate list size $ListSize$, and a third implicit parameter, the initial seed $Seed$ for the pseudo-random number generator. The first parameter corresponds to the number of iterations in the outer loop of the algorithm. The second parameter will be seen in more detail when explaining the construction phase, but roughly speaking, it is a measure of how many alternatives will be taken into account at each constructive step.

In some GRASP versions the size of the restricted candidate list is recomputed dynamically (i.e. the value of $ListSize$ is not fixed), being used in this case a threshold parameter denoted by α. Later on, we explain in detail both variants.

Looking again at the pseudo-code, it can be seen that GRASP iterations are carried out between lines 2 and 6. Each GRASP iteration consists of the construction phase (line 3), the local search phase (line 4) and, if necessary, the solution update (lines 5-6).

In the construction phase, a feasible solution is built. Algorithm 4 shows a generic pseudo-code for the construction phase. The solution is usually represented as a set of elements (the precise meaning of these elements depends on the specific problem); the construction phase starts from an empty set and iteratively adds an element until the set corresponds to a feasible solution.

At each step of the construction phase, a restricted candidate list (denoted by RCL) is determined by sorting all non already selected elements with respect to a greedy function that measures the (myopic) benefit of including them in the partial solution. In general, this function evaluates the incremental increase in the cost function $f(\cdot)$ when incorporating each new element into the solution under construction. Specifically, by applying this function, we build the RCL containing those elements whose addition to the current partial solution induce the smallest incremental costs (this is the greedy component of GRASP).

Algorithm 4 Pseudo-code for procedure Construct_GRSol (Construct Greedy Randomized Solution)

```
Procedure Construct_GRSol($ListSize$,$Seed$):
1: $Solution \leftarrow \emptyset$;
2: Incremental costs evaluation for the candidate elements;
3: while not_feasible($Solution$) do
4:     $RCL \leftarrow$ the restricted candidate list;
5:     $s \leftarrow$ select randomly an element from the $RCL$;
6:     $Solution \leftarrow Solution \cup \{s\}$;
7:     Incremental costs revaluation;
8: return $Solution$;
```

The next element to be included into the partial solution is randomly chosen (uniformly or in some biased form) from the RCL (this is the probabilistic component of GRASP). In this way, GRASP allows for different solutions to be obtained at each GRASP iteration. When the chosen element is added to the partial solution, the benefits associated with every element not yet added to the partial solution are updated in order to reflect the change induced by the insertion of the new element. Thus, the heuristic recomputes the RCL and reevaluates the incremental costs (this is the adaptive component of GRASP). Once the construction phase is finished, the solution built is returned.

The solutions generated by the construction phase are not guaranteed to be locally optimal with respect to simple neighborhood definitions. Hence, it can be beneficial to apply a local search to attempt to improve each constructed solution. A local search algorithm works in an iterative fashion by successively replacing the current solution by a better one taken from its neighborhood. It finalizes once there is no better solution found in the neighborhood.

Algorithm 5 shows a generic pseudo-code for the local search phase. It has as input a feasible solution *Solution* and searches for a better solution within a neighborhood $N(Solution)$ previously defined. In most of the cases, the local search phase takes as entry the feasible solution *Solution* delivered by the construction phase, but for certain applications, we could have several local search phases working in a combined form by exploring different neighborhoods, implying thus that their entries are not necessarily the solutions given by the construction phase.

Algorithm 5 Local_Search pseudo-code

```
Procedure Local_Search($Solution$):
1: while not_locally_optimal($Solution$) do
2:     Find $Neigh\_Sol \in N(Solution)$ satisfying $f(Neigh\_Sol) < f(Solution)$;
3:     $Solution \leftarrow Neigh\_Sol$;
4: return $Solution$;
```

The success when applying the local search phase is strongly related with the following points:

- the suitable choice of a neighborhood structure,
- efficient neighborhood search techniques,
- easy evaluation of the cost function when exploring the neighborhood,
- the quality of the starting solution.

The construction phase plays an important role with respect to this last point, since it must produce good starting solutions for this local search sub-procedure. Depending on the problem, the used neighborhoods are generally not complex. There exist two basic different strategies to explore a neighborhood, which are:

best-improvement: all neighbors are investigated and the current solution is (possibly) replaced by the best neighbor.

first-improvement: when finding the first better neighbor solution (i.e. whose cost value is smaller than that of the current solution), the current solution is replaced by this one.

In [Resende 2003], the authors mention that empirically (when applying both strategies on many applications), in most of the cases, both strategies reach the same final solution, but in general the *first-improvement* takes a smaller computational time. Besides, they observe that is more frequent the premature convergence to a non-global local optimum by using *best-improvement* than *first-improvement*.

One important characteristic of GRASP is its low parametrization; few parameters need to be set and tuned. This implies that the main effort can be focused on implementing efficient data structures to obtain fast iterations. Let us analyze the influence of the GRASP parameters and the RCL construction.

A GRASP algorithm finalizes once performed $MaxIter$ iterations. Clearly, the probability of finding a new solution improving the currently best one decreases with the number of iterations already computed, the quality of the best solution found may only improve with the latter. In general, the computation times from iteration to iteration are relatively similar, therefore the total computation time depends linearly on $MaxIter$. Thus, when increasing $MaxIter$, the global computation time will be increased as well as the probability of finding better solutions.

At any GRASP iteration, let us denote by $c(e)$ the incremental cost associated with the insertion of element $e \in E$ into the solution under construction and by c_{min} and c_{max} the smallest and the largest incremental costs respectively. There are two main variants to compute the RCL used in the construction phase. Next, we shall describe both approaches.

i) Given a positive integer $ListSize$, the RCL is composed of the $ListSize$ elements of E with the best (i.e. smallest) incremental costs. In this case, we say that the RCL is cardinality-based. The size of the RCL can be smaller than $ListSize$ since, depending on the instance, we could not get to compute exactly the $ListSize$ best elements.

ii) The second variant uses a threshold parameter denoted by $\alpha \in [0, 1]$. In this case the size of the RCL is dynamically adapted according to the quality of the elements to be added (we say that the RCL is value-based). Fixed α, the RCL is formed by all "feasible" elements $e \in E$ which can be inserted into the partial solution under construction without losing feasibility and whose quality is superior to the threshold value; that is to say:

$$e \in \text{RCL} \Leftrightarrow c(e) \in [c_{min}, c_{min} + \alpha(c_{max} - c_{min})].$$

If we set $\alpha = 0$ the resulting algorithm is purely greedy, and with $\alpha = 1$ we obtain a random construction. Hence, we can infer that α regulates the amounts of greediness and randomness in the construction phase.

For further details of GRASP the reader may consult the references [de Aragão 2001, Feo 1989, Feo 1995, Martins 2000, Resende 2003, Ribeiro 2002], which provide an extensive analysis of the GRASP metaheuristic based on many applications. Topics discussed include: successful implementation techniques, parameter tuning strategies, alternative solution construction mechanisms, techniques to speed up the local search, reactive GRASP, cost perturbations, bias functions, memory and learning, local search on partially constructed solutions, hashing, filtering, implementation strategies of memory-based intensification and post-optimization techniques using path-relinking, hybridizations with other metaheuristics and parallelization strategies.

2.2 Technical background

At this point we summarize several technical concepts and preconditions that were surveyed during first steps of the analysis and constitute the foundations of models described in Section 3.1 and Section 3.2.