APPENDIX 2

DETERMINISTIC OR RANDOM SEARCH METHODS

The search space is the set of entities over which the search is conducted. In the case of optimisation, the search space is a set of possible solutions to the problem at hand. There is an important distinction between the search space itself and the representation space, which consists of data structures used to represent the points in search space. In the case of evolutionary methods, the search space is often known as the space of *phenotypes*, this term is borrowed from biology, where the phenotype is the organism itself, as distinct from its *genotype*, which is its genetic description (or representation).

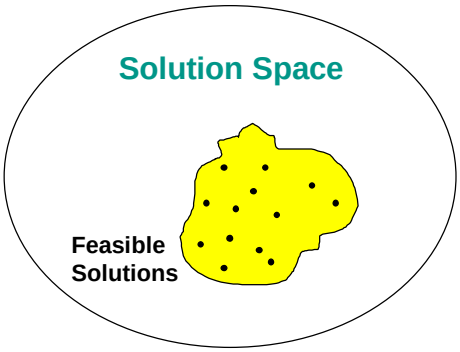

Figure A2.1. Feasibility space.

Many search (optimisation) methods are formulated by presenting their search space, together with an objective function (or a set of objectives) and a set of constraints. A constraint usually takes the form of a clause (an equation or an inequality) that must be satisfied by a solution in order to be considered feasible. For instance, in engineering design, the optimisation problem may be to minimise the cost of some products (i.e. an engine) subject to various constraints that ensure its stability, its

adequacy to its environment, sufficient capacity, and so forth. In certain cases, the object of the search may be simply to locate a solution that satisfies all these constraints, i.e. the objective function itself may be absent. Constraint satisfaction problems can, of course, be formally restated as optimisation problems by constructing an objective function that computes the number of constraints violated by a solution and by seeking to minimise them.

In constrained optimisation, the search space becomes decomposed into two regions, the feasible region, consisting of those solutions that satisfy all the constraints, and the infeasible region, in which some are violated (Figure A2.1). Depending on the form and complexity of the constraints, a number of approaches are available in the literature. The most generally applicable, but usually the least satisfactory, is to use an additive penalty function to degrade the measured quality of solutions that violate constraints.

Perhaps the most general mechanism for handling constraints in optimisation (*meta-heuristics methods*) is to employ a *penalty* function. It is a term that is added to the value of the given objective function when a solution violates one or more constraints, degrading its rated performance. The size (*value*) of the added penalty reflects in some way the degree of constraint violation. It is common practice to increase the size of penalties during the course of a run, so that while a degree of violation is tolerated in early stages (*generations*), this tolerance is reduced over time. Constraint satisfaction problems can, of course, be tackled as optimisation problems by using a penalty function in the absence of an objective function. The principal attraction of penalty functions is their more-or-less universal applicability. Nevertheless, they exhibit a number of drawbacks. First, they do not provide any problem specific information to the method. Secondly, the choice of weighting for the constraints is a somewhat subtle matter. This problem with penalty functions is rather like the problem that arises when a multi-objective problem is tackled by forming a weighted sum of the various objectives.

In certain cases, the procedure for *mapping* an infeasible solution to a feasible one is known. Such a procedure is known as a *repair mechanism*. There are various ways of incorporating repair mechanisms into evolutionary algorithms. While these approaches are generally more satisfactory than employing penalty functions, some care must be taken to ensure that the set of possible moves in the search space do not become drastically limited (Sedgewick, 1984).

When searching for the global optimum solution of complex problems, one is generally faced to a fundamental conflict between precision, reliability and computing time. Each optimisation method represents a particular compromise. Traditional hill-climbing *'exact'* methods (gradient descent, simplex method, branch & bound) mainly concentrate on local exploitation of the search space. They solely focus on the precision and computation time at the expense of reliability. They almost suffer from the rush to the first discovered local extrema. Genetic Algorithms (GAs) is a promising search method originally influenced by the *Darwinian* theory (Holland, 1975), present many advantages and original principles such as: population-based

search, recombination and stochastic mechanisms. These methods rough out a problem by finding the most promising regions of the entire search space. However, they often yield unsatisfactory results: indeed, they suffer from a certain inefficiency, characterised by a slow convergence and a lack of accuracy when a high quality solution is required.

The GAs focus on the global part of the search task and give less attention to the local part. This characteristic often prevents them from being a practical method for many real-world applications. So, w*hat is missing the GAs to be an efficient global search method?* The global exploitation capacities are often non-existent for hill-climbing methods, consequently, their utility can be greatly reduced if they are used as such for the search in multi-modal search space (*numerous local optima*). These considerations, highlighting the complementary properties of the meta-heuristics and particularly GAs and the hill-climbing, suggest that hybridisation between both approaches may lead to improve performances of search methods.

Almost, all solution methods can solve some problems, yet they cannot solve a significant fraction of other interesting instances–this what we call in computer science jargon the *method overfitting*. Indeed, search methods and problem knowledge must be taken together with a certain care, the more we use knowledge the more search space is limited. Thus, the space exploration's method can be more or less very restricted to small regions. In some cases, the random *choice of moves* significantly accelerates the search methods. For high constrained problems, if the problem have only one valid solution, *random search* can do better job than a well defined search method!

It is highly desirable to be able to find ways of incorporating knowledge about the domain into search methods. This knowledge must be used as much as possible by the method during its execution. All problems are more or less simple to solve, the main task is to well understand their structure. An instructive inspiration from the nature by observing some phenomena can help (to design a method) to solve design problems. Since computers have to do just the tedious part of designer job, semi-automatic computer-aided design methods must be preferred to the automatic ones. It is the human (decision maker) who has to decide about the proposed designs. Each designer or domain expert typically brings specialised knowledge or skills to the design process.

References

(Sedgewick, 1996) Sedgewick R. and Flajolet P., 'An introduction to the analysis of algorithms', Addison-Wesley, England, 1996.

(Holland, 1975) Holland J.H., 'Adaptation in natural and artificial systems', University of Michigan Press, Ann Arbor, 1975.