APPENDIX 1

SEARCH AND SEARCH SPACE

A critical issue in solving search problems is the choice of suitable representation language for describing the states in the search space as well as the operators used to search in. It is well known for designers and optimisers that these choices can have a major impact on the efficiency of the search method.

In most general sense, knowledge representation means that an intelligent system has descriptions that correspond to the world. In computer science, these descriptions have to be clear enough such that an intelligent machine can conclude about its environment by formally manipulating them. In artificial intelligence's (AI) symbol manipulation, we can distinguish between the propositional (sentence like) representations, and complex representation models (schemata), like frames or scripts. The schemata allow representation of abstract information about objects, states, or events.

The notion of the world is carefully taken by the AI community, since what they represent in their program is knowledge about a strictly limited and artificial model (micro-world) of the world. The *good-sense* knowledge is the everyday practical knowledge used by people in mastering real life situations. While observing the human languages, a question is often raised about the representation of human good-sense knowledge. Is it simply a problem of scale and could therefore be solved by a large storage capacities, a lot of hard work, and ingenious search algorithms or whether is it a problem that lies in the nature of human capacities. It is quite hard to represent (*encode*) the human knowledge symbolically in formal languages like frames, scripts, etc. One of the big challenges is the incorporation of knowledge from the domain into the search methods.

Expert System also called knowledge-based systems are classical methods introduced to mimic the experts (humans) in their way of solving problems. Such systems can produce good results in areas where the human expertise is well developed and well represented by simple rules... Due to the success of expert systems in some fields, they have been integrated into the branch of computer science and artificial

intelligence. Some researchers have proclaimed that such systems can compete with human intelligence. However, such systems are domain depending and most of human knowledge are hard to integrate to computer programs. Until now, it is extremely hard to represent the human knowledge, since it is hard to count the type and the amount of knowledge he uses in his everyday experience...

The *encoding* is possibly one of the most difficult task in all solving methods. The problem is to find a representation that can be handled easily by the solving method at hand. Most of the methods use boolean, integer, real valued, logic expression, graph and many other kinds of search spaces. In order to make search algorithms more general and easier to interface, a simple representation is often seek. A problem comes from scalability, as the encoding sometimes includes some sort of compression of data (a kind of problem's knowledge). Furthermore, using a decoding process also distorts the search space in a manner that may (or may not) help the search algorithms.

The *representation* space is the set of *conceptual* data structures that are manipulated by an algorithm in the search space over which the search is conducted. In evolution strategies *strings* of real values are typically used to represent problem solutions, the genetic programming uses the *tree* of a program as its basic data type, while genetic algorithms use binary (or integer) string representations. Note that, in the latter, the representation space is defined not by the current data structures declared in a computer program when implementing the algorithm, rather by the sub-components of solutions the algorithm manipulates. One of the main challenges is how to design intelligently the representation of the search space?

In other words, the question is how to intelligently organise the search space to facilitate the job and reduce time to reach goals. How to organise (*represent*) the data in computer and how to manage them. As human, since having a body, we are spatially located creatures: we must always be facing some direction, have only certain objects in view, be within reach of some others. How we manage the spatial arrangement of items around us, is not an afterthought; it is an integral part of the way we think, plan and behave. The *spatial arrangements* of data, if it is well designed, can simplify the *choice* in case of uncertainty and can also simplify *perception* to well evaluate the decisions taken (moves). In a general search context, the search space is the set of entities over which the search is conducted. In the case of optimisation and decision problems, the search space is the 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 conceptual data structures used to represent the points in the search space.

A *global* optimisation is the task of finding the absolutely best solution to a given problem (an objective function, shape,...). In general, problems have a set of *local* optimal solutions which are not globally optimal. Consequently, global optimisation problems are typically quite difficult to solve exactly. In the context of combinatorial problems, they are often called NP-hard. Methods for global optimisation problems

can be categorised based on the properties of the problem and the types of guaranties provided to reach the 'optimal' solution.

Design is an activity that most humans are routinely engage in. Different designers of course operate with very different design objectives, design tools, and consequently, different design outcomes. Indeed, as pointed out by Herbert Simon–one of the founders of AI, *any entity, be it natural or artificial, that devises courses of action aimed at changing existing situations into preferred ones (whatever they might be), can be said to engage in design activity* (Simon, 1981).

The design process, from a computational perspective, essentially involves searching a design (search) space. The search space can be described by the 3-tuple (S, O, G) where S is the *start state* (which captures the existing states: inputs), O is the set of *operators* that can be used to transform a state into another, and G is a specification of the *desired goal state* (e.g., solution that meets the design objectives). For example, in a theorem proving task (which involves devising a proof of a theorem), the initial state consists of the axioms, the goal state is the theorem to be proved, and the operators are the logically sound rules of inference. Thus, the design activity is a sequence of operator applications that transforms the initial state into a goal state. It is also possible to think of the design task in terms of its decomposition into hierarchy of (hopefully simple) design sub-problems.

Like any problem that involves search in typically large spaces, design problems tend to be computationally hard in the absence of appropriate knowledge to guide search. Fortunately, search can be guided by knowledge that is at the disposal of the designer (human). This knowledge may be domain independent or domain specific and may take various forms. For example, such knowledge may be used to narrow the scope of the search by effectively *constraining* the search to a certain part of the search space, or *ranking* the alternatives available for exploration at a given stage of the design process in terms of their promise, etc. A special attention must be done to variability and uncertainty of the problem knowledge. This knowledge itself may be acquired and refined by the designer through its experience. While knowledge is useful in constraining the search, it should be noted that a blind *reliance* on the known can prevent exploration of the unknown and hence *hinder* the discovery of novel and truly creative designs. In many cases, random search can significantly help to explore the search space, and some times can do more better than *sophisticated* methods...

In general if there is sufficient knowledge about a problem domain, there is no need to search for a solution. However, if there is lack of knowledge about the search space, then if there is an information that help to detect areas within which a solution is expected to be found, a search method can be used to hunt for a solution within this area. We are concerned with design problems that do not lend themselves to a direct solution but that can be cast as search problems within a space of possible valid solutions.

Most search methods typically tend to oscillate among two trends which are *speed* (the number of candidate designs searched through before finding the correct one) and

flexibility (applicability to different problems). Most methods balance *exploration* of the search space with *exploitation* of areas of the space. Exploration points out new areas to search in, while exploitation concentrates search in a particular area. Uncontrollable exploration leads to wasted time in unpromising areas, while severe exploitation may miss the correct solution by concentrating on too small an area– there is a need for a balance among the two trends. That is, too much exploration means too much time, and too much exploitation means that the *valid* solution may not be reached in the attributed time. On the other hand, in the case of weak exploration, the valid solution may not be generated while insufficient exploitation may take too long. Striking a well balance between exploration and exploitation is by the way critical for search methods.

Wasting time in unpromising areas of the search space may seem a small price to pay for finding the correct solution. Unfortunately, in real-world design domains the size of the search spaces is so *large* that a human lifetime is short compared to the time needed by fastest super-computers to look at a significant fraction of search space. That why random or exhaustive search are infeasible since it explores too much.

Local search methods, are algorithms that do not execute any exploration and have knowledge available to 'solve' the problem quickly and directly. These algorithms rely on the existence of sufficient exploitable information and make strong assumptions about the search space to avoid long time search. However, since these strong assumptions do not usually hold across all kind of search spaces, these algorithms work only on the particular space they were designed for and do *miserably* on others. If a solution method can solve instances in a benchmark set, yet it cannot solve a significant fraction of other instances, then we have a case of method *over-fitting* (Falkenauer, 1998). Almost all search methods depend on domain-knowledge and favour speed over flexibility. Most are so brittle that they fail completely when domain knowledge is scarce.

In between these extremes of random or exhaustive search and almost no search, other search algorithms make assumptions about the search space. These assumptions correspond to *knowledge* about the space and may be correct, incorrect, or misleading. This knowledge is exploited to guide exploration–at least when to stop, thus speeding up the search.

Successful design in many domains requires substantial *creativity* from designers. Creativity involves not only exploration of domains, but modification, extension or transformation of domains by manipulating constraints.

References

(Falkenauer, 1998) Falkenauer E., 'On method overfitting', Journal of Heuristics, Vol. 4, pp. 281-287, 1998.

(Simon, 1981) Simon H.A., 'The sciences of the artificial', (3rd ed.), The MIT Press, Cambridge, MA, 1981.