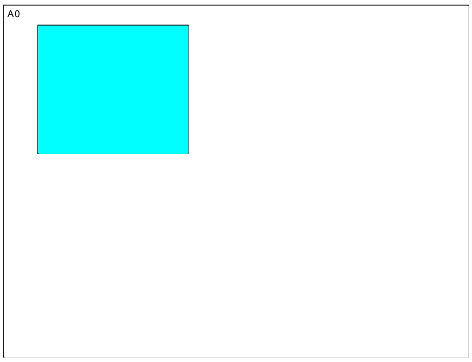
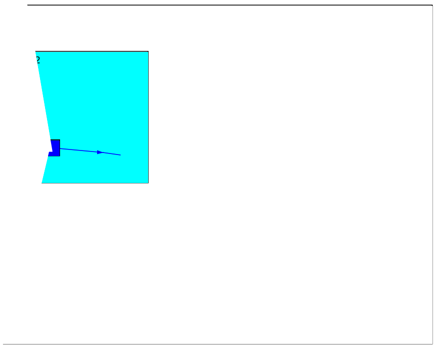
Evolving Good Hierarchical Decompositions of Complex Systems

ud̀ Lutz


A0

A0


2

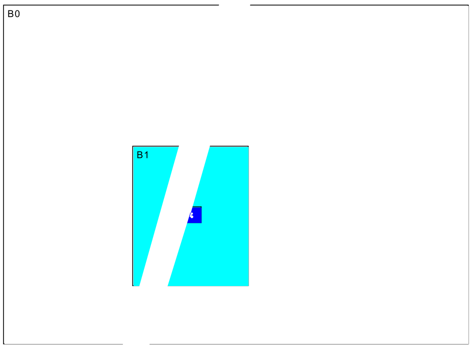
- prefer designs with greater cohesion in the modules
- prefer designs with less coupling between modules

These guidelines do not however say anything at all about what to do when they

Turing machine. One can therefore think of the encoding as being a message that is sent to some recipient (the decoder), whose task is then to decode the message. Given that the message will consist of string of bits, it is necessary for the decoder to be able to recognise which substrings of these bits represent which parts of the message. Therefore all the apparatus of Information Theory[11], and in particular the theory of optimal uniquely decodeable codes, can be brought to bear.

Given some data D, one is often faced with a choice of several competing theories which can account for the data. An often used principle in making such deciwill take. Only by understanding exactly what needs to be sent, and how often various symbols are used in such a message can one understand the complexity formula. We will begin by describing the ***high-level format*** of the encoding of a HMD, and then discuss encoding this as a string of bits whose length will be taken as the complexity of the HMD. It should also be noted that we will follow the usual not turn up in the messages), “empty” denotes the empty string, and |entitySeq| denotes the length of the following entitySeq (and similarly for |moduleSeq| and |connectionSeq|). |unconnected| represents the number of unconnected basic entities (i.e. with no links) in a module. The meaning of |modulesUp| will be explained in the next section where we give details of how links are described. It should be noted that we have been very careful about the ordering of information in these messages (links are described after details of which modules there are, and names have been specified for modules and basic entities) to enable a recipient of the message to decode them. By the time they get to link descriptions they will already know all names (codes) for entities and modules. It should also be noted that if a system has some isolated entities (entities with no connections to others), then we do not need to name these, but can simply say how many there are. This is a consequence of the fact that we are only trying to communicate the *structure* of the system, and not any other information such as the arbitrary names a designer might have used for the various components.

- Describing Links


B0
B1

2 0 n3 1 B2 n5   *(entity n9 info: 2 outgoing links)*

does not have prior knowledge of the relative frequencies of the various symbols occurring in the message. So how often is each name used in the message. A little thought will show that the names of a connected basic entity n will occur in the message once for each incoming edge of n, and once to specify the name prior to the edge descriptions, giving a total of

$$f(n) = \mathit{indegree}(n) + 1$$

Similarly, the name of each module m will occur

$$f(m) = \mathit{indegree}(m) + 1$$

times. Additionally, the first occurrence of each of these names (essentially telling the recipient what name (code) is being used, so they can recognise it in the future, has to be preceded by its length, so the recipient can tell when the name ends.

Information theory tells us that when transmitting a message containing symbols s_i occurring with frequencies f_i, then we need roughly

$$-\log_2\left(\frac{f_i}{f}\right)$$

bits (where $f = \sum_i f_i$) for the code for s_i, and that the message length will be minimised if we use codes of that length. Furthermore, information theory tells us that such codes exist. Furthermore, because of our use of “route descriptions” to name entities connected to a node, we need only guarantee unique names for the basic entities and submodules *within their containing module.* Accordingly, we can take the length of the code for a name to be

$$-\log_2\left(\frac{f(n)}{\sum_x f(x)}\right)$$

where x ranges over the named basic entities and sub-modules of the module n is in. In other words, the code lengths for names are determined by the *relative frequencies* of the names. The actual frequencies are determined globally, but the relative frequencies are determined by using these global frequencies locally within a module to compute the lengths of the local names.

- The Complexity Formula

In the light of the above discussion the formula for the complexity $\psi(X)$

where $\mathcal{M}$ is the set of all the modules in the HMD X, l represents the code lengths for integers discussed earlier, U_m is the set of unconnected basic entities in module m, C_m is the set of connected basic entities in module m, M_m

Another distinction often made in the GA literature is between the "classiuals in a neighbourhood can never be replaced until there is another fitter individual in the neighbourhood.

-   Crossover

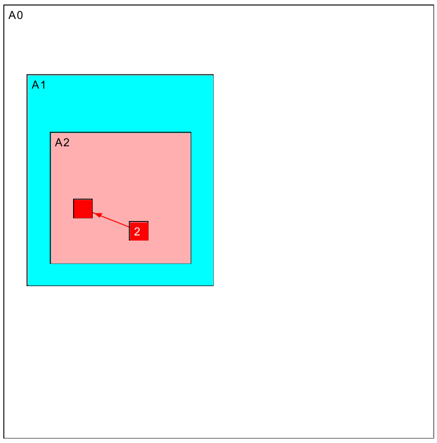
Defining a suitable crossover operation which produces a new HMD from two "parent" HMDs is not entirely trivial. Given a fixed underlying system S we can think of a HMD as simply being a tree structure over the nodes of S. We do not need to take the links into consideration as they are the same for all HMDs over S (of course we need to consider the links when computing the *fitness* of the HMD their description lengths form the dominant part of ψ). Since each genome can therefore be encoded by a tree structure, we have based our crossover operation on the tree-based crossover operation used by Koza[5] in his Genetic Programming work. However, using Koza's original tree crossover is not guaranteed to give a legal HMD of S, and we have to repair the result so that it is legal. This is easily done using the concatenation operator described below.

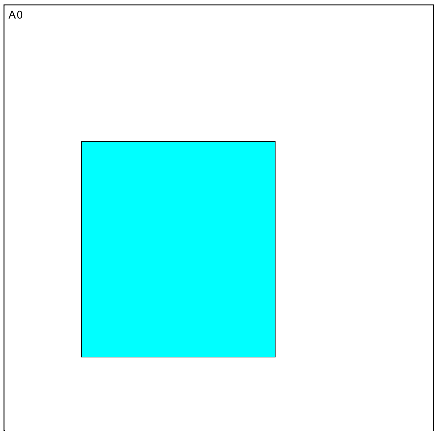
- -   Concatenat n   wo HMDs

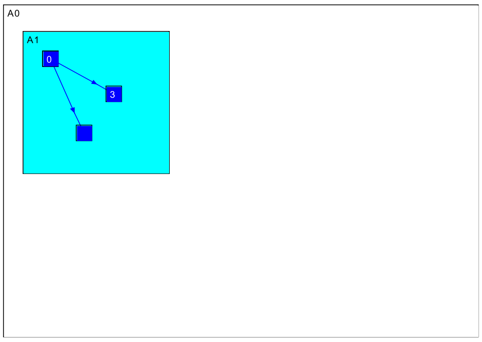
Given two HMDs H_1 and H_2 for two underlying systems S_1 and S_2 one can define a family of concatenation operators $\circ_M$ (one for each module M occurring in H_1) for "gluing together" the two HMDs to form a new HMD $H_1 \circ_M H_2$ for the system $S = S$T0, 7TL21.67 6 Td.[f100-1134.4410.14Tm(=)Tj61Tm[(S)-4.10 14]TJ in tui77.31 21.6 0 10 16(h)-31A.205(H)-1.10 14(l)-6.64422

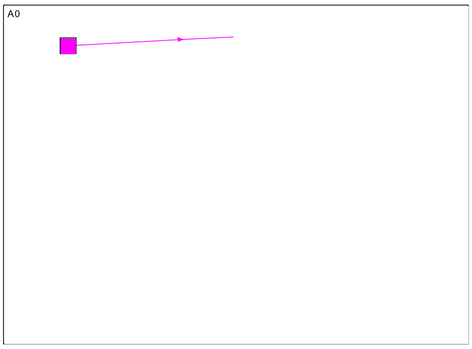
T_1. Define $S_1 \circ_M S_2$ (read *the concatenation of* S_2 *onto* S_1 *at* M) by the following process

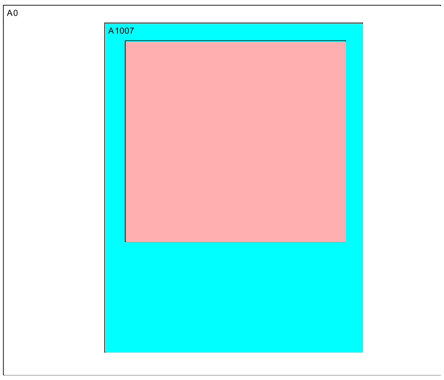
1. Let T be a copy of T_1, and T_2' a copy of T_2.
2. Delete from T_2'


A0
A1
A2
2


A0


A0
A1
0
3


A0


A0
A1007

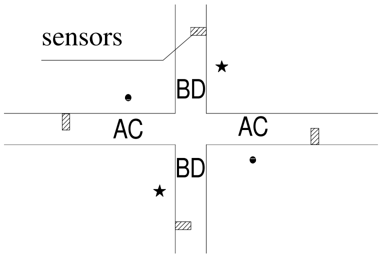


Figure 26 The traffic lights example

techniques. In this last example we will look at one (fairly small) example of a real software design, published in [10]. The requirement is to control the flow of traffic over a cross-roads, see Figure 26. The sensors provide information about waiting traffic, and the lights are only changed if traffic is waiting and the current pair of

[10] Robinson, P.J. (1 2) *HOOD: Hierarchical Object-Oriented Design*. Prentice-Hall Object-Oriented Series, Prentice Hall.

[11] Shannon, C.E. (1 48) The mathematical theory of communications. *Bell System Technical Journal* 27 37 –423, 623–656.

[12] Thornton, C.J. and du Boulay, B. (1 2) *Artificial Intelligence Through Search*. Intellect, Oxford, England.

[13] Weyuker, E.J. (1 88) Evaluating software complexity measures. *IEEE Transactions on Software Engineering*, SE-14( ), pp. 1357-65.

[14] Witten, I.H., Neal, R.M., and Clear, J.G. (1 87). Arithmetic coding for data compression. *Communications of the ACM*, 30(6) 520–540.

[15] Wood, J.A. (1 8) Improving Software Designs via the Minimum Description Length Principle. Ph.D. Thesis, University of Sussex.