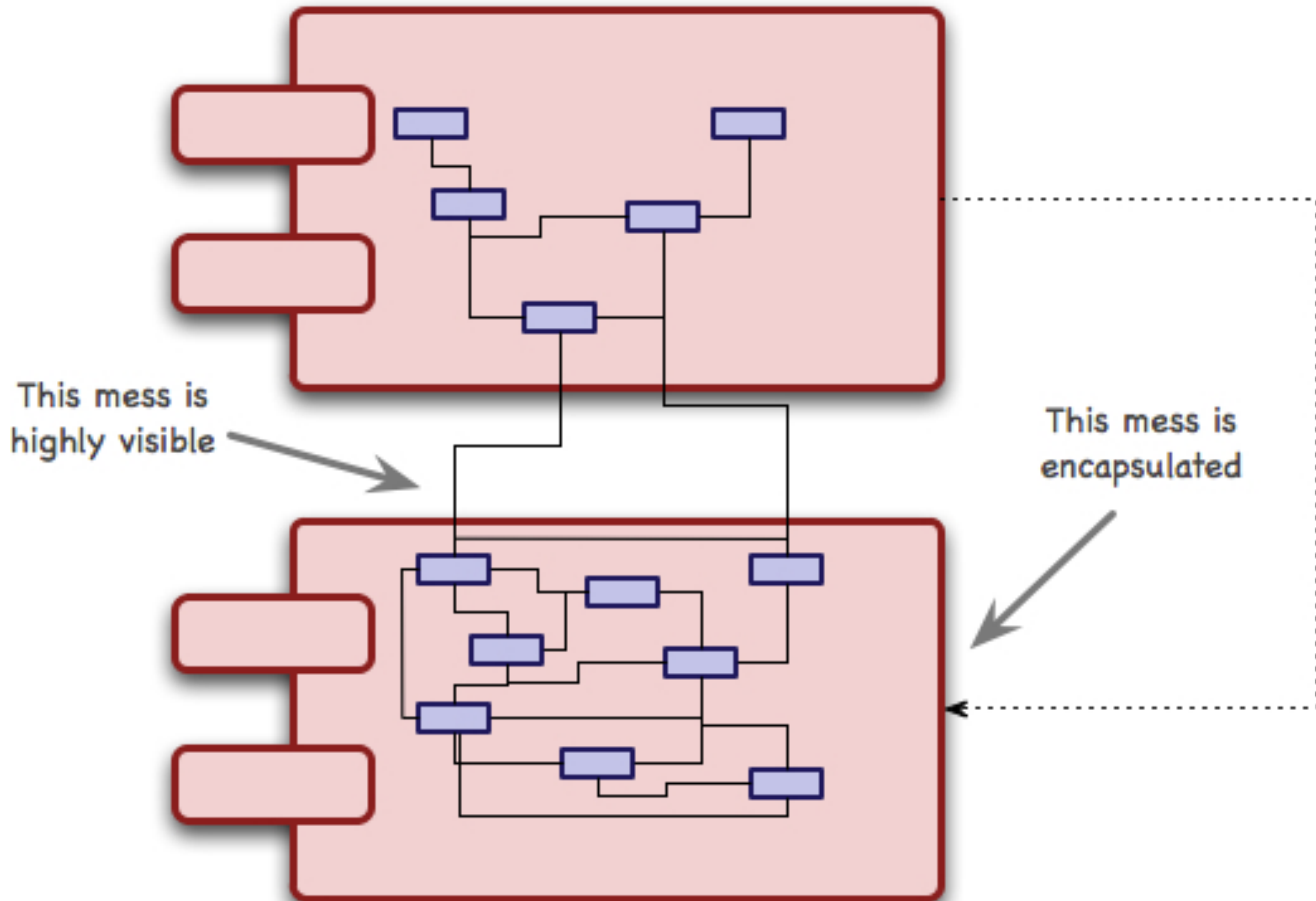


Image from: http://www.kirkk.com/modularity/wp-content/uploads/2009/12/EncapsulatingDesign1.jpg

Example of Hill Climbing Application: Software Module Clustering (Algorithmic Design)

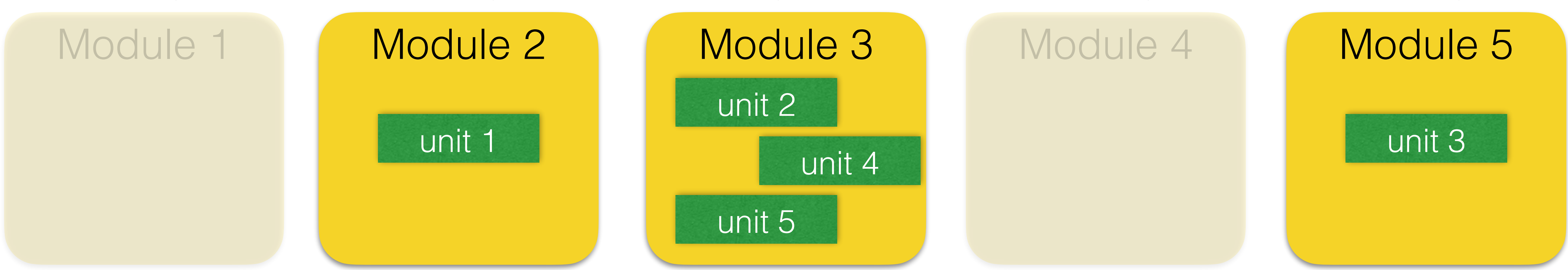
Leandro L. Minku

Design Variable

Design variable: allocation of units into modules.

- Consider that we have N units, identified by natural numbers in $\{1,2,\ldots,N\}$.
- This means that we have at most N modules.
- Our design variable is a list L of N modules, where each module L_i, $i \in \{1,2,\ldots,N\}$, is a set containing a minimum of 0 and a maximum of N units.

$L_1 = \{\}$ $\quad L_2 = \{1\}$ $\quad L_3 = \{2,4,5\}$ $\quad L_4 = \{\}$ $\quad L_5 = \{3\}$



Constraints and Objective Function

Constraints: N/A

Objective function: quality of modularisation (to be maximised).

$$\text{Quality}(L) = \sum_{\substack{i \in \{1,2,\ldots,N\} \mid \\ L_i \neq \{\}}} \text{Quality}(L_i)$$

(maximise)

$$\text{Quality}(L_i) = \frac{\#\text{IntraEdges}_i}{\#\text{IntraEdges}_i + 1/2 * \#\text{InterEdges}_i}$$

(maximise)

Problem Formulation

Hill-Climbing (assuming maximisation)

1. current_solution = generate initial solution randomly

2. Repeat:

    2.1 generate neighbour solutions (differ from current solution by a single element)

    2.2 best_neighbour = get highest quality neighbour of current_solution

    2.3 If quality(best_neighbour) <= quality(current_solution)

        2.3.1 Return current_solution

    2.4 current_solution = best_neighbour

Until a maximum number of iterations

Design variable —>
what is a candidate solution for us?

Objective —>
what is quality for us?

Are there any constraints that need to be satisfied?

Designing Representation, Initialisation and Neighbourhood Operators

Hill-Climbing (assuming maximisation)

1. current_solution = generate initial solution randomly

2. Repeat:

    2.1 generate neighbour solutions (differ from current solution by a single element)

    2.2 best_neighbour = get highest quality neighbour of current_solution

    2.3 If quality(best_neighbour) <= quality(current_solution)

        2.3.1 Return current_solution

    2.4 current_solution = best_neighbour

Until a maximum number of iterations

- Representation:
    - How to store the design variable.
    - E.g., boolean, integer or float variable or array.

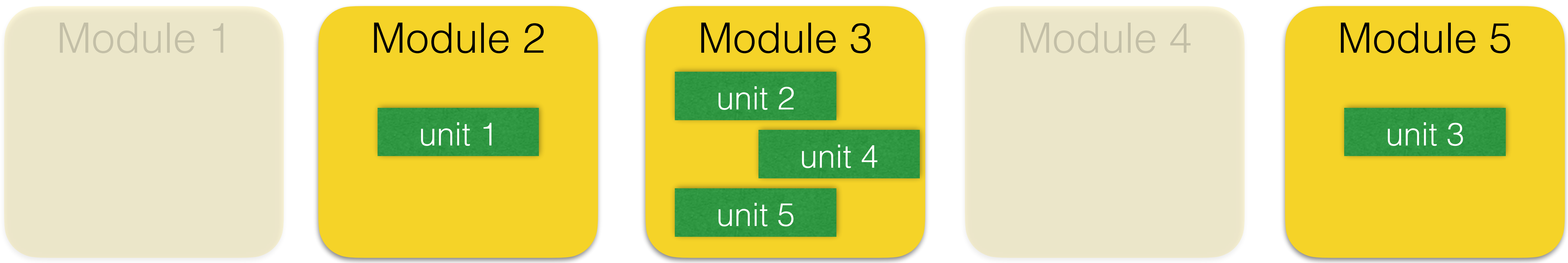
- Initialisation:
    - Usually involve randomness.

- Neighbourhood operator:
    - How to generate neighbour solutions.

Representation

How to represent the design variable internally in the implementation?

- E.g., list of N modules, where each module is a list of integers in {1,2,…,N} identifying the existing units.

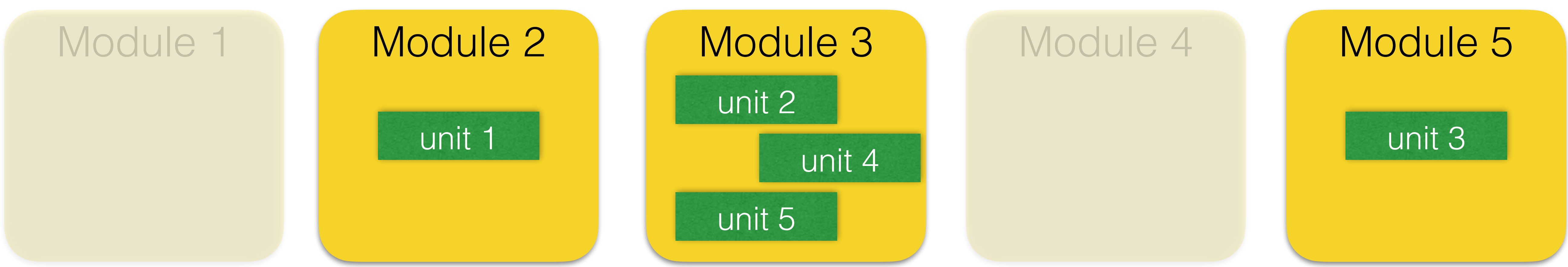


- E.g., if we have N=5, a possible allocation is L = {{},{1},{2,4,5},{},{3}}.

Representation

How to represent the design variable internally in the implementation?

- E.g., matrix A_{NxN}, where Aij = 1 if unit j is in module i, and 0 otherwise.



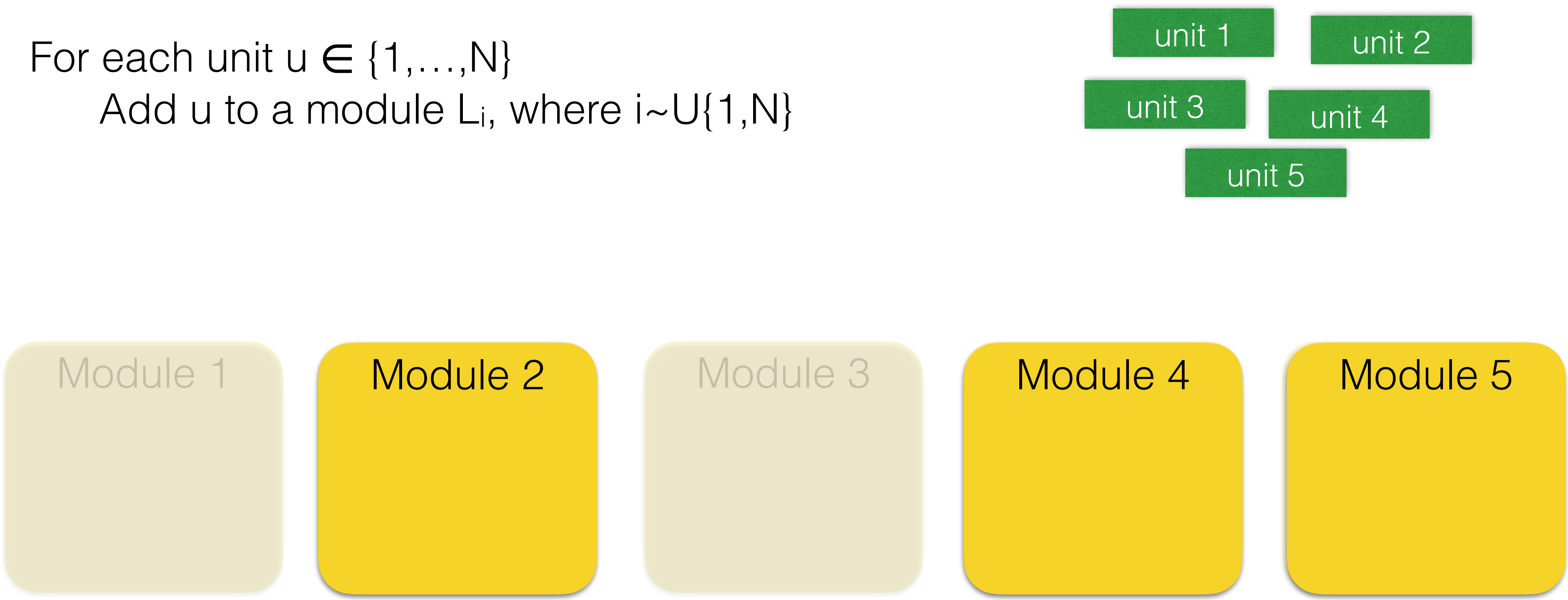
$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Initialisation

E.g.: place each unit into a randomly picked module.

For each unit $u \in \{1,\ldots,N\}$

Add u to a module L_i, where $i \sim U\{1,N\}$

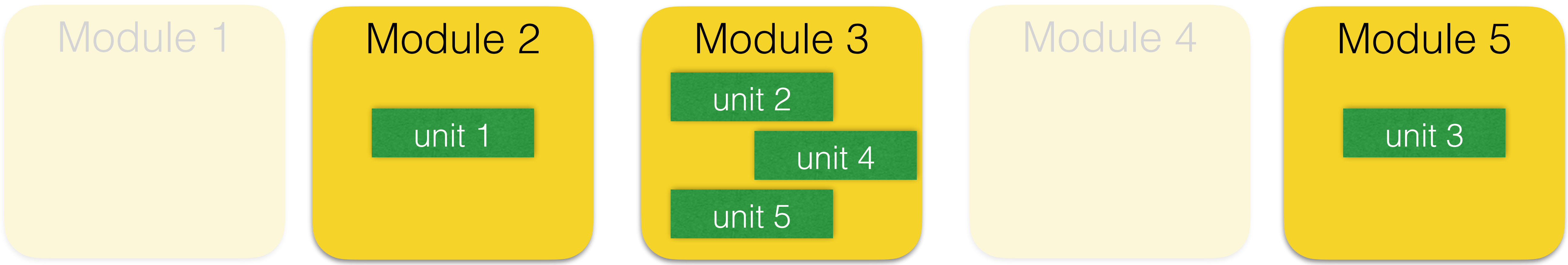


$L = \{\{\},\{1,3\},\{\},\{2\},\{4,5\}\}$

Neighbourhood Operator

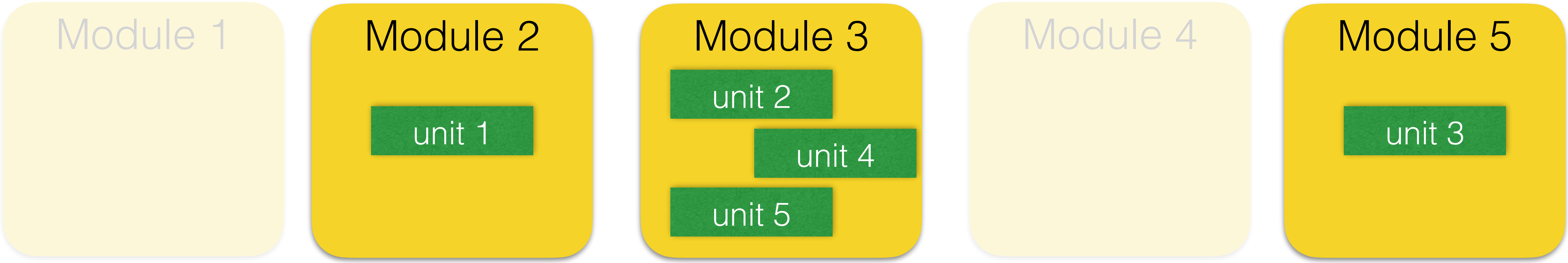
- What would be a possible neighbourhood operator for the software clustering problem?
  - A neighbour in the software module clustering problem would be a solution where a single unit moves from one module to another. E.g.:

$$L = \{\{\},\{1\},\{2,4,5\},\{\},\{3\}\} \longrightarrow L = \{\{\},\{1,5\},\{2,4\},\{\},\{3\}\}$$



Neighbourhood

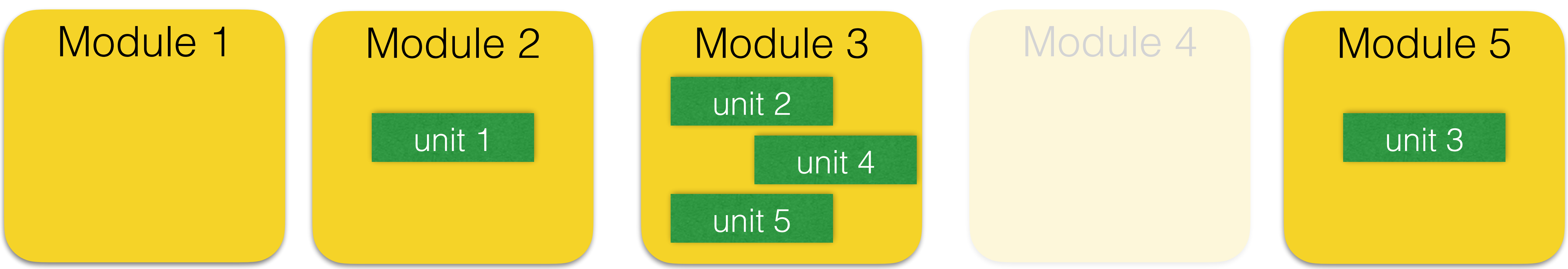
- Real world problems will frequently have more than two neighbours for each candidate solution.
- How many neighbours do we have for the candidate solution below, if we allow for equivalent neighbours?



5 units * 4 possible modules to move to = 20

Neighbourhood

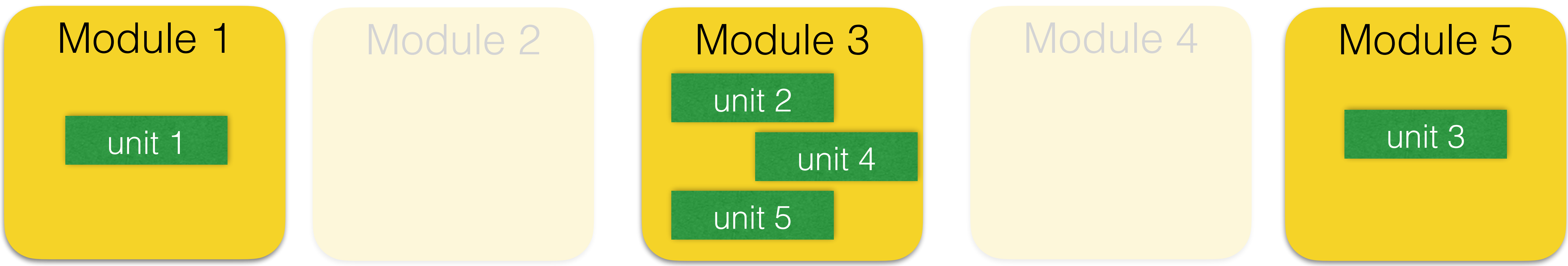
- How many neighbours do we have for the candidate solution below, if we allow for equivalent neighbours?



Some neighbours will be equivalent.
Duplicates could be eliminated.

Neighbourhood

- How many neighbours do we have for the candidate solution below, if we allow for equivalent neighbours?



For $i \in \{1,\ldots,N\}$ // module

For $j \in \{1,\ldots,\text{size}(L_i)\}$ // unit within module

For $i' \in \{1,\ldots,N\} \setminus i$ // another module

L' = clone of L

Move unit L'_{ij} to module $L'_{i'}$

Yield L' as a neighbour

Hill Climbing

Hill-Climbing (assuming maximisation)

1. current_solution = generate initial solution randomly

2. Repeat:

    2.1 generate neighbour solutions (differ from current solution by a single element)

    2.2 best_neighbour = get highest quality neighbour of current_solution

    2.3 If quality(best_neighbour) <= quality(current_solution)

        2.3.1 Return current_solution

    2.4 current_solution = best_neighbour

Until a maximum number of iterations

Simulated Annealing would also require a representation, initialisation procedure, and neighbourhood operator to solve a problem.

Summary

- Software Module Clustering problem formulation.
- Representation, initialisation and neighbourhood operators.

Next

- Application of Simulated Annealing.