CO3091 - Computational Intelligence and Software Engineering

Lecture 09

Image from: https://media.licdn.com/mpr/mpr/shrinknp_800_800/AAEAAQAAAAAAAAb4AAAAJDRmMWEzMzA3LWMxMWEtNDljNS05N2NhLTIwMjBlNjYwZjQxMg.jpg

Requirements Selection

Leandro L. Minku

Overview

- What are requirements?
- What is requirements selection?
- How to formulate it as an optimisation problem?
- How to design an optimisation algorithm for this problem?
- Variants of the problem formulation.
- [Revision of some genetic operators]

What Are [Software] Requirements?

- There are different definitions.
- According to the IEEE Standard Glossary of Software Engineering Technology, a requirement is:

1. A condition or capability needed by a user to solve a problem or achieve an objective.
2. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.
3. A documented representation of a condition or capability as in 1 or 2.

What Are [Software] Requirements?

- More loosely, requirements can be seen as:
  - [Descriptions of] services that a software system must provide and the constraints under which it must operate.
  - [Descriptions of] needs of stakeholders that are to be solved by software.
- Examples:
  - Every order should be assigned a unique identifier.
  - User should be able to search the database.
  - Information from the database should be retrieved to the user in less than 1s.

Types of Requirements

- Requirements can be categorised according to different perspectives. E.g.:
  - Functional requirements: services to be provided by the system.
  - Non-functional requirements: constraints on the services to be provided by the system.
  - User requirements: statements in natural language and diagrams of the services to be provided.
  - System requirements: detailed descriptions of the system services.

Each requirement has a cost.

Requirements

[Descriptions of] needs of stakeholders that are to be solved by software.

Who Are the Stakeholders?

- “The people and organisations affected by the application.”

Conger, S. (1994) The New Software Engineering, International Thomson Publishing.

- “System stakeholders are people or organisations who will be affected by the system and who have a direct or indirect influence on the system requirements.”

Kotonya, G. and Sommerville, I. (1998) Requirements Engineering: processes and techniques, John Wiley.

- “Stakeholders are people who have a stake or interest in the project.”

Cotterell, M. and Hughes, B. (1995) Software Project Management, International Thomson Publishing.

Examples of Stakeholders

- Users.
- Developers.
- Legislators, e.g.:
  - professional bodies, government agencies, legal representatives, safety executives.
- Decision-makers, e.g.:
  - managers of the software development team, user managers, financial controllers.

Different stakeholders may have different importance to a company.

Different stakeholders may see different value in different requirements.

The level of satisfaction of a stakeholder depends on the requirements that are satisfied.

Requirements Selection

- We need to decide which possible requirements to implement, considering (potentially among others):
  - their cost,
  - their value from different stakeholders perspectives,
  - the importance of the stakeholder who wants the requirement.

Requirements Selection Problem

- Select requirements so as to:
  - Maximise the value of the requirements selected.
  - Considering the stakeholders' importances.
  - Subject to a maximum cost (budget).

- Problem gets difficult to solve manually when there are many requirements and many stakeholders involved.

- Requirements need to be selected not only in the first release of a software, but also in subsequent releases.
  - The next release problem.

Requirements Selection Problem Formulation

- Consider that you have:
  - A set of stakeholders for a software system:
    Stakeholders = $\{s_1, s_2, \ldots, s_m\}$

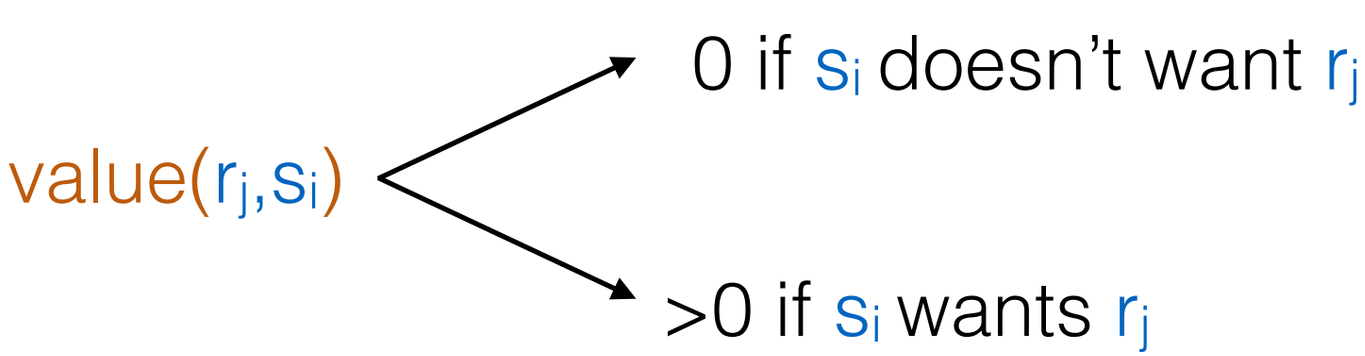
  - Weight reflecting the importance of each stakeholder:
    Importance = $\{imp(s_1), imp(s_2), \ldots, imp(s_m)\}$
    $imp(s_i) \in [0,1]$
    sum of the importances = 1

  - A set of possible requirements:
    Requirements = $\{r_1, r_2, \ldots, r_n\}$
    (assuming the requirements are independent)

  - Cost of each requirement:
    Costs = $\{cost(r_1), cost(r_2), \ldots, cost(r_n)\}$

Requirements Selection Problem Formulation

Each stakeholder s_i assigns a value to each requirement r_j.



The higher the value(r_j,s_i), the more important r_j is to s_i.

Requirements Selection Problem Formulation

sum of values of the requirement from different stakeholders' perspectives

Score of requirement r_j: $score(r_j) = \sum_{i=1}^{m} value(r_j, s_i)\ imp(s_i)$

Requirements Selection Problem Formulation

Score of requirement r_j: $\text{score}(r_j) = \sum_{i=1}^{m} \text{value}(r_j, s_i)\ \text{imp}(s_i)$

weighted sum of values of the requirement

Design variable: $\mathbf{x} = \{0,1\}^n$

- xj = 0 if requirement r_j is not included
- xj = 1 if requirement r_j is included

Objective: Maximise $\sum_{j=1}^{n} \text{score}(r_j)\ x_j$

Requirements Selection Problem Formulation

Selected requirements must be within budget.

Constraint: $\sum_{j=1}^{n} \text{cost}(r_j)\, x_j \leq C$

total cost of selected requirements

Example of Evolutionary Algorithm Design

Representation: binary $\mathbf{x} = \{0,1\}^n$, where

- n is the number of potential requirements,
- each position corresponds to a specific potential requirement,
- 0 represents that the corresponding requirement is not included,
- 1 represents that the corresponding requirement is included.

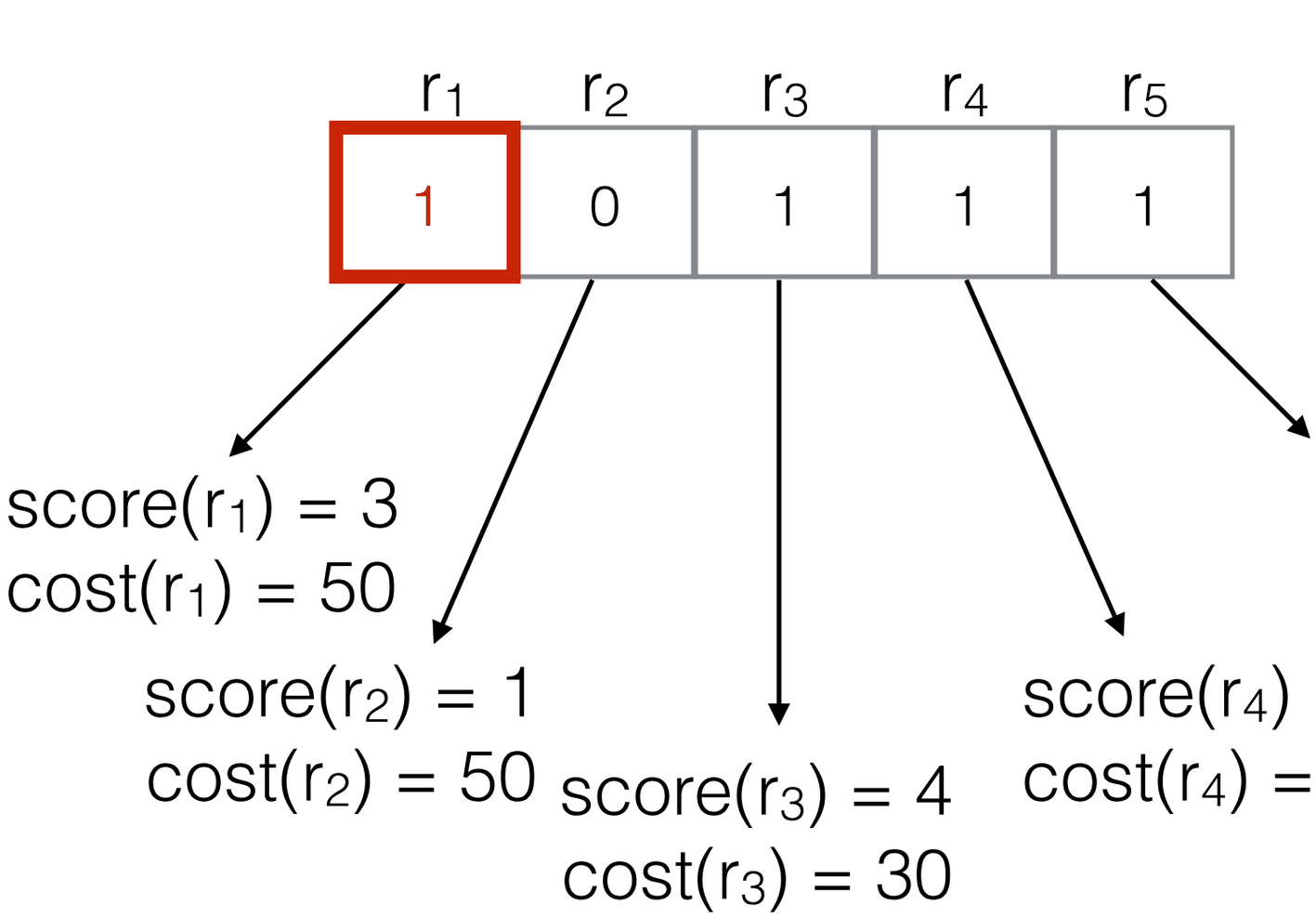
Fitness function: $\sum_{j=1}^{n} \text{score}(r_j)\ x_j$ (to be maximised)

total score of selected requirements

Example of Evolutionary Algorithm Design

Dealing with the budget constraint:

- Repair operator:
  while total cost > C
    delete (flip to zero) the currently included requirement with lowest score



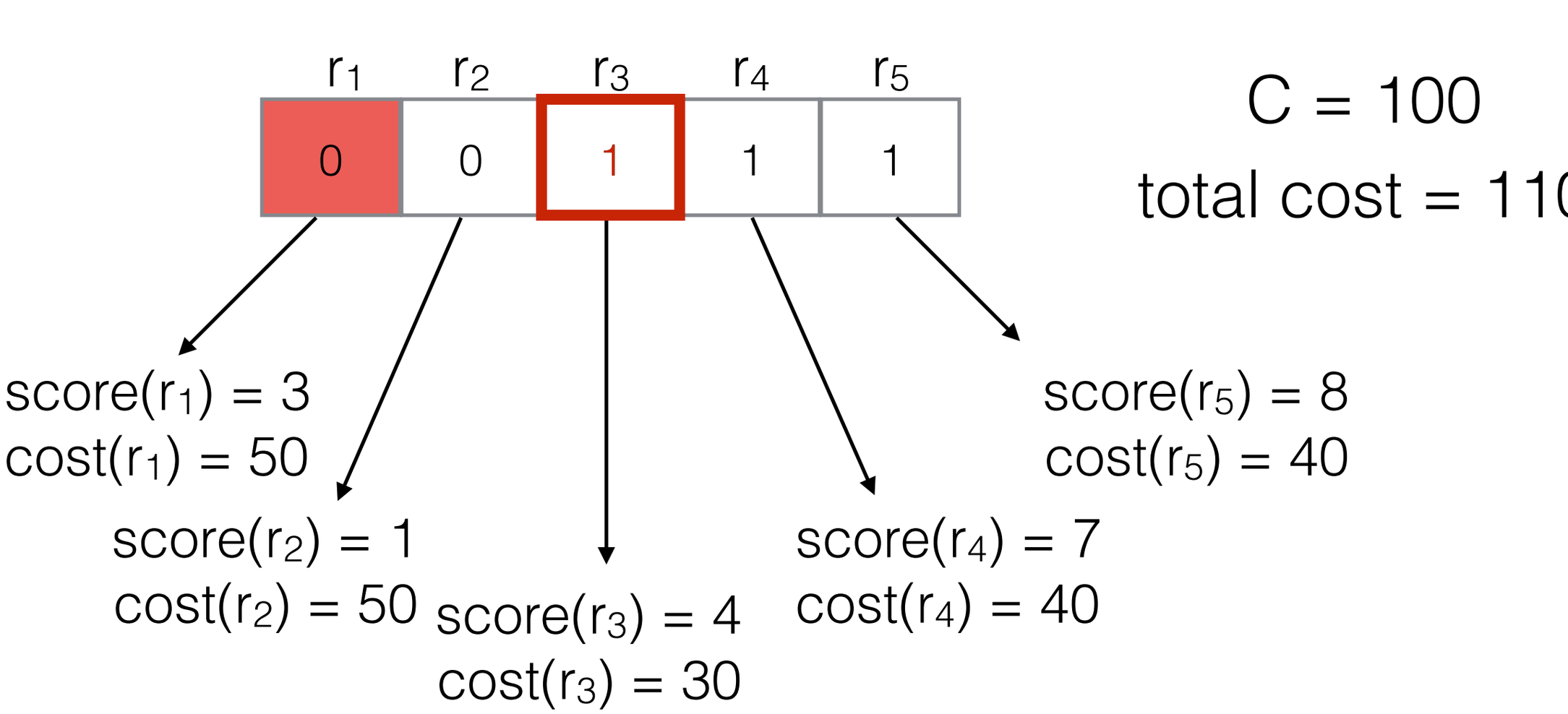
C = 100

total cost = 160

Example of Evolutionary Algorithm Design

Dealing with the budget constraint:

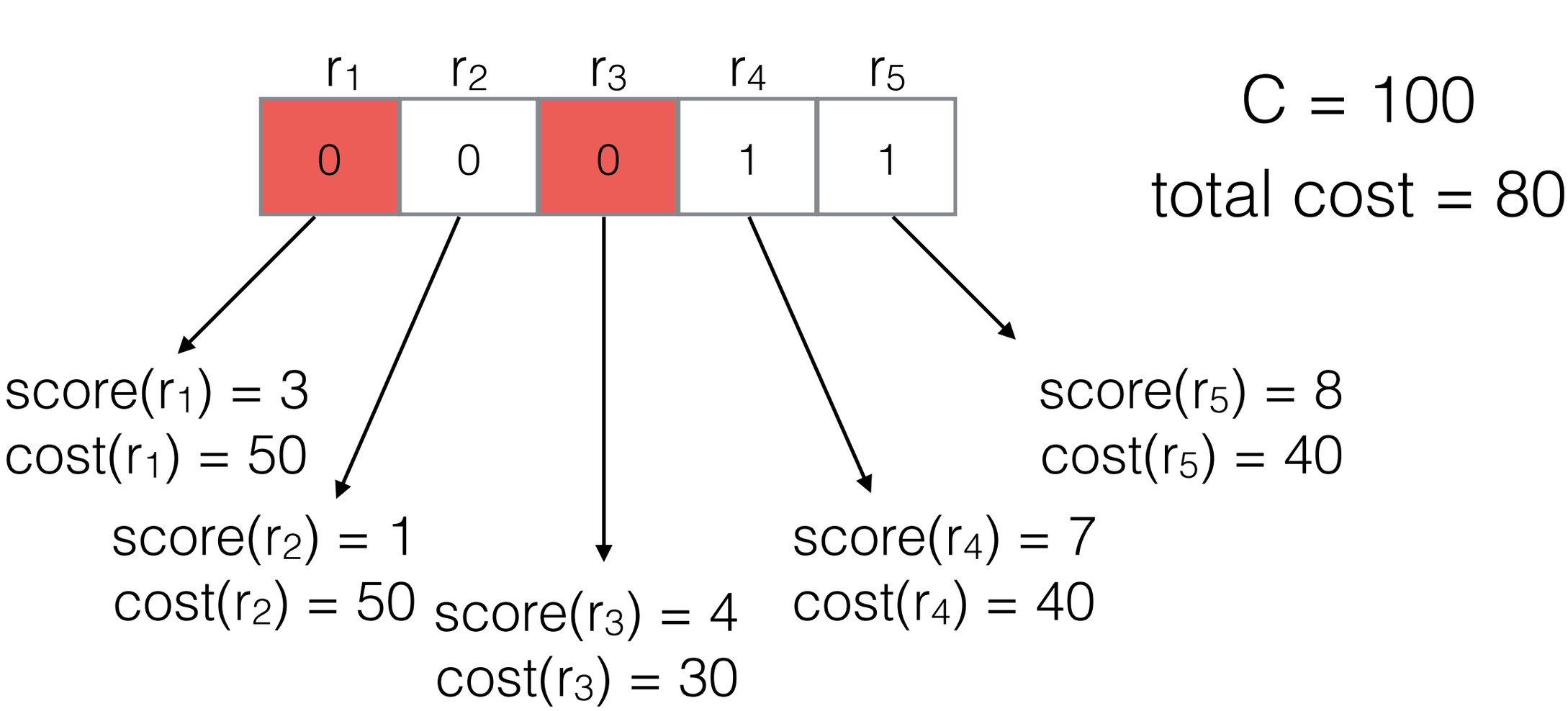
- Repair operator:
  while total cost > C
    delete (flip to zero) the currently included requirement with lowest score



Example of Evolutionary Algorithm Design

Dealing with the budget constraint:

- Repair operator:
  while total cost > C
    delete (flip to zero) the currently included requirement with lowest score



Example of Evolutionary Algorithm Design

Crossover: uniform crossover

Mutation: bitwise bit-flipping

Parents selection: 2-tournament selection

Survivor selection: elitist + age-based

Stoping criterion: maximum number of generations

Uniform Crossover

- For each gene of offspring 1:
  - Give 50% chance for gene to come from parent 1 and 50% to come from parent 2.
- Make an inverse copy of the gene for the offspring 2.
- Inheritance is independent of position.

Parent 1 1 0 0 1 0 0 0 1
Parent 2 0 1 0 1 1 0 1 0

Child 1 1 1 0 1 1 0 0 0
Child 2 0 0 0 1 0 0 1 1

heads = from parent 1
tails = from parent 2
heads = from parent 1

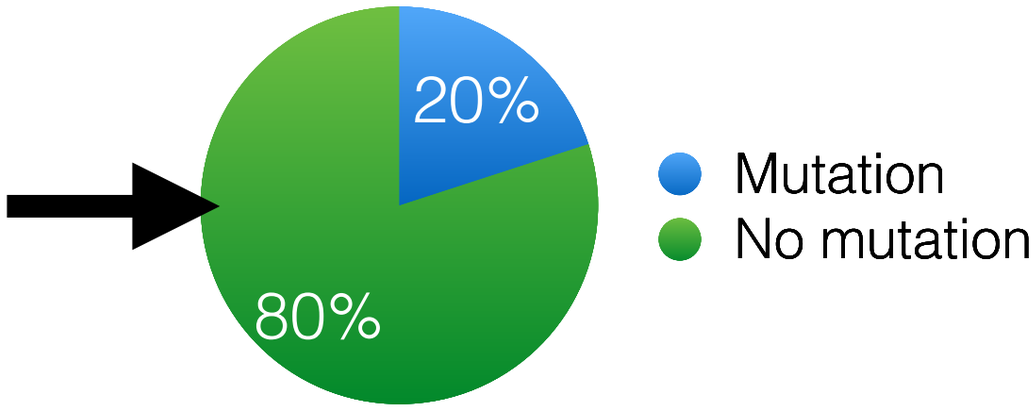
Image from: http://cacm.acm.org/system/assets/0001/3918/112013_CACMpg126_Puzzled.large.jpg?1384869700&1384869700

Bitwise Bit-Flipping Mutation

- Flip each gene (from 0 to 1 or vice-versa) with probability *Pm*.
- *Pm* is called mutation rate, and is typically in [1/pop_size, 1/chromosome_length].
- Example:

Before mutation 0 1 1 0 1

After mutation 0 1 1 **1** 1



Tournament Parents Selection

- Informal Procedure:
  - Pick *k* members at random then select the best of these.
  - Repeat to select more individuals.

E.g.: k = 2, assuming maximisation

Genotypes	Fitnesses
00011	9
01000	64
10101	25
01001	81

Parent: 01001

Elitist + Age-Based Survivor Selection

1. set of survivors = all children

2. Find the best individual among children and previous generation.

3. If the best individual is an individual from the previous generation
    1. Replace the worst child in the set of survivors by this best individual.

4. Make set of survivors survive!

EA Pseudocode

Evolutionary Algorithm

1. Initialise population **(random, using binary representation)**
2. Evaluate each individual **(repair and determine the total score of selected requirements)**
3. Repeat **(until a maximum number of generations)**
   - 3.1 Select parents **(using tournament selection)**
   - 3.2 Recombine parents with probability Pc **(using uniform crossover)**
   - 3.3 Mutate resulting offspring with probability Pm **(using bitwise bit-flipping)**
   - 3.4 Evaluate offspring **(repair and determine the total score of selected requirements)**
   - 3.5 Select survivors for the next generation **(elitist + age-based)**

Alternative Problem Formulation

- There is no constraint on the total cost.
- We want to maximise the total score and minimise the total cost.
- Fitness function can be the weighted average between total score and total cost of selected requirements.

Alternative Problem Formulation

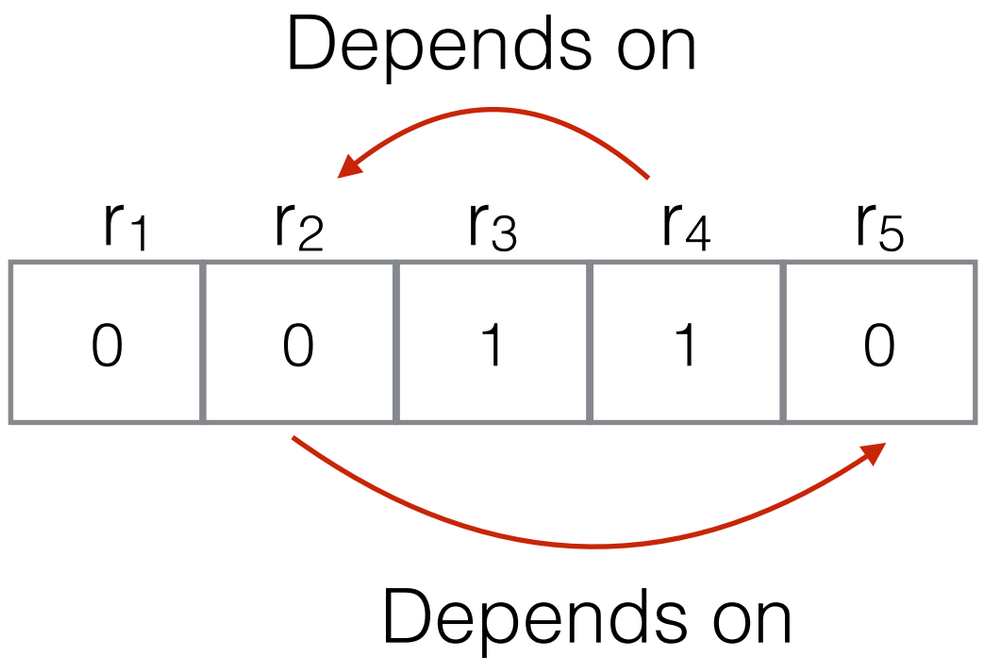
- There are relationships among requirements.
  - Dependency: some requirements can only be included if some other requirements are also included.
  - Conflict: some requirements cannot co-exist.

Example of How to Deal with Constraints

- Repair operator for dealing with dependency constraint:
  - For each included requirement with dependencies,

    Set all genes on which the requirement depends to 1.

    Ensure that the dependencies of the newly added requirements are satisfied too.

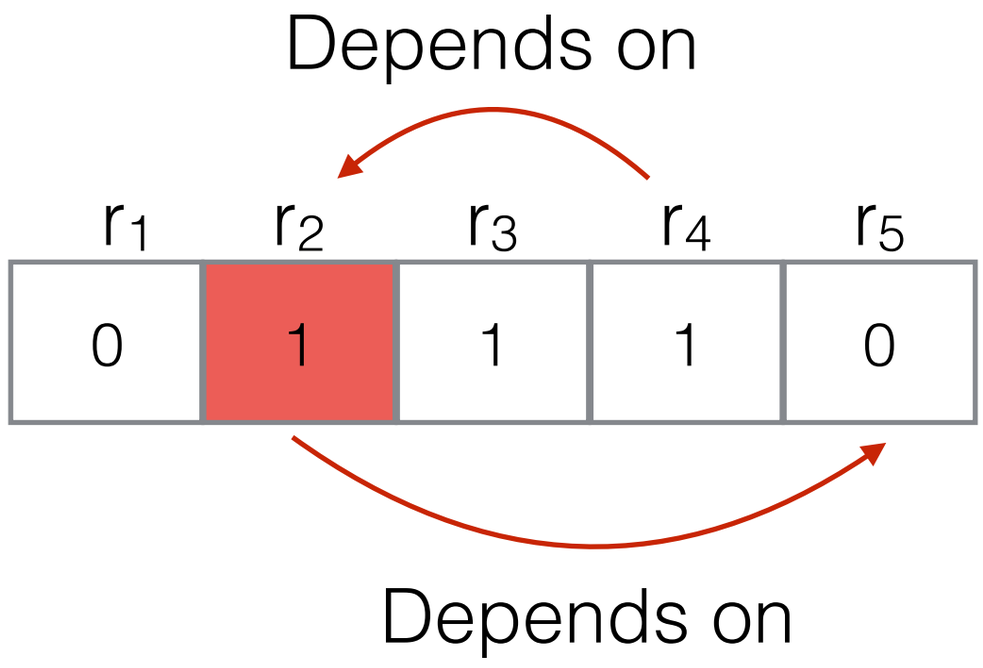


Example of How to Deal with Constraints

- Repair operator for dealing with dependency constraint:
  - For each included requirement with dependencies,

    Set all genes on which the requirement depends to 1.

    Ensure that the dependencies of the newly added requirements are satisfied too.

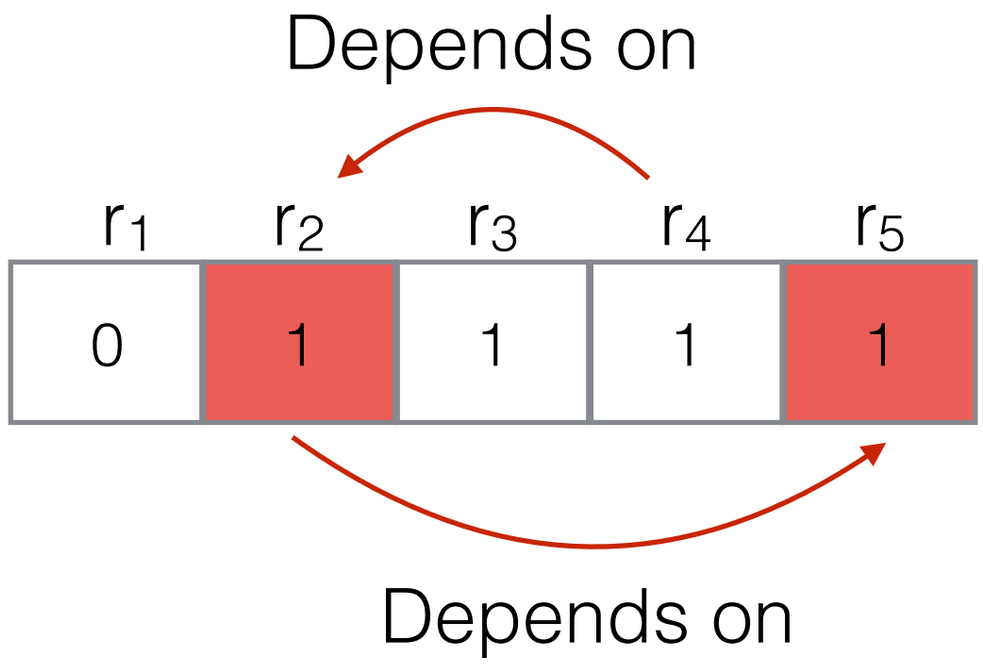


Example of How to Deal with Constraints

- Repair operator for dealing with dependency constraint:
  - For each included requirement with dependencies,

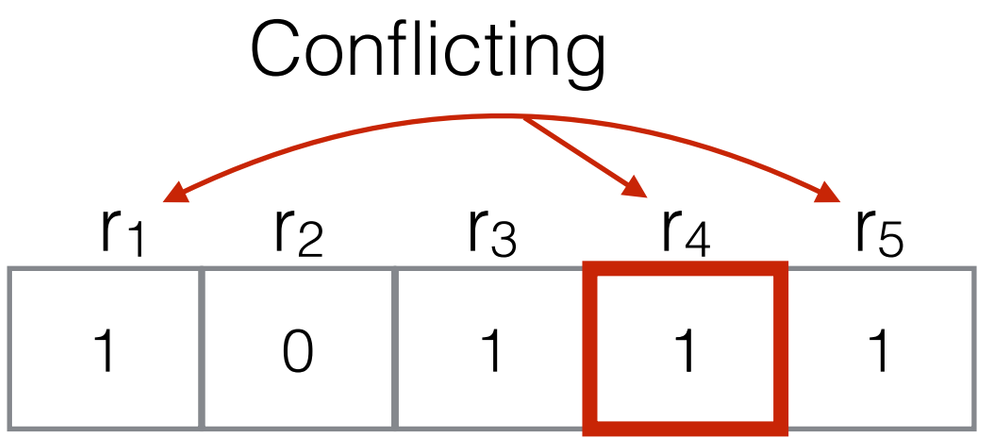
    Set all genes on which the requirement depends to 1.

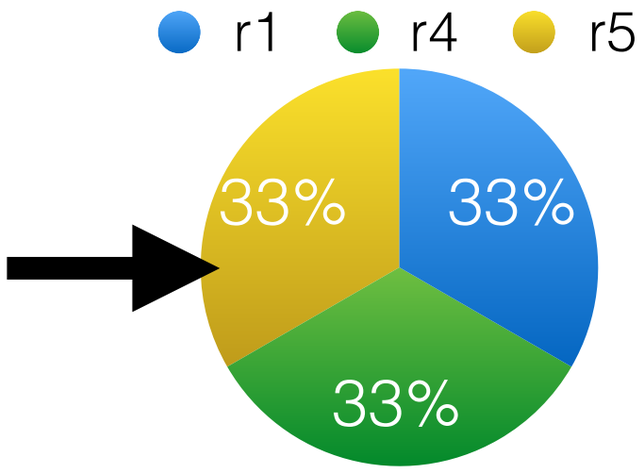
    Ensure that the dependencies of the newly added requirements are satisfied too.



Example of How to Deal with Constraints

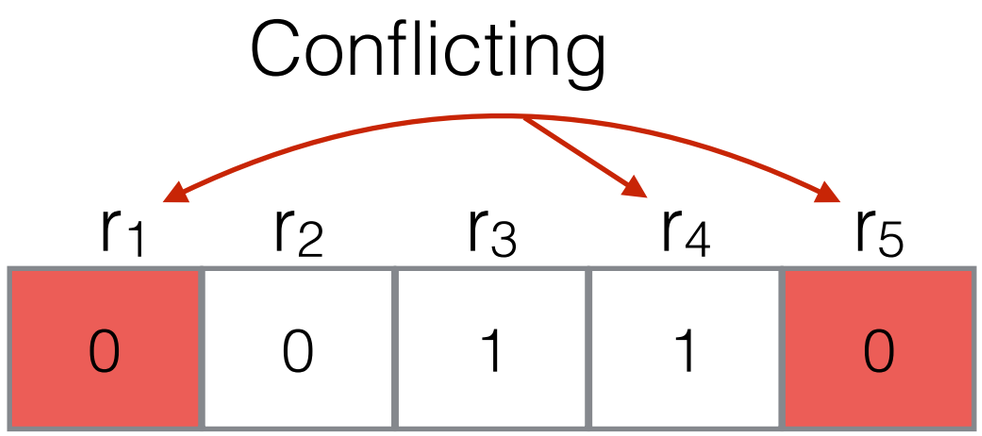
- Repair operator for dealing with conflict constraint:
  - For each included requirement with conflicts,
    - Randomly select only one of the conflicting requirements to be kept as 1.





Example of How to Deal with Constraints

- Repair operator for dealing with conflict constraint:
  - For each included requirement with conflicts,
    - Randomly select only one of the conflicting requirements to be kept as 1.



Summary

- Requirements selection is difficult to perform manually when the number of potential requirements and stakeholders is large.
- We can formulate it as an optimisation problem.
- Requirements selection is quite similar to the lorry problem.
- There are different variants of the requirements selection problem.
- Different variants need different algorithm designs.

Further Reading

Requirements Interaction in the Next Release Problem

José del Sagrado, Isabel M. Águila, Francisco J. Orellana

Proceedings of the 13th annual conference companion on Genetic and evolutionary computation, Pages 241-242

http://dl.acm.org/citation.cfm?id=2001858.2001994