بسم اللّه الرحمن الرحیم

- تذکر۱: لطفاً یادداشت‌برداری کنید. آنچه در کلاس مطرح می‌شود ممکن است فراتر از محتوای اسلایدها باشد.
- تذکر ۲: لطفاً در اسرع وقت ایمیل خود را در سامانهٔ درس (lms.iut.ac.ir) وارد کنید.
- تذکر ۳: اگر احیاناً در ترجمه یا فهم اسلایدها مشکل داشتید، لطفاً در ساعات رفع اشکال به دفتر بنده مراجعه کنید.
- تذکر ۴: اسلایدها و دیگر محتواهای مرتبط با درس روی سامانهٔ درس (lms.iut.ac.ir) آپلود می‌شوند.

The 0-1 knapsack problem

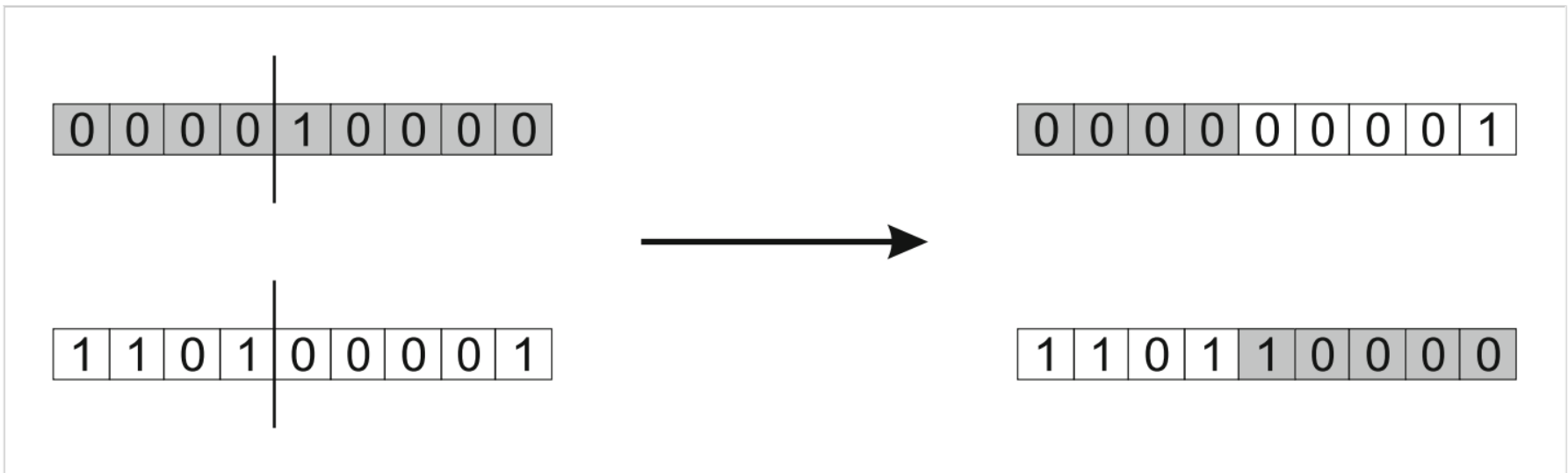
We are given a set of n items, each of which has attached to it some value v_i, and some cost (weight) c_i. The task is to select a subset of those items that maximises the sum of the values, while keeping the summed cost (weight) within some capacity $C_{\max}$.

Representation

It is a natural idea to represent candidate solutions for this problem as binary strings of length n, where a 1 in a given position indicates that an item is included and a 0 that it is omitted.

Recombination for Binary Representation

One-Point Crossover: It works by choosing a random number r in the range $[1, l-1]$ (with l the length of the encoding), and then splitting both parents at this point and creating the two children by exchanging the tails.



Mutation for Binary Representation

Bitwise mutation for binary encodings: Although a few other schemes have been occasionally used, the most common mutation operator for binary encodings considers each gene separately and allows each bit to flip (i.e., from 1 to 0 or 0 to 1) with a small probability p_m.

Evaluation function (or fitness function)

Item	A	B	C	D	E	F	G	H	I
Value	13	8	11	16	4	1	1	9	10
Cost (i.e., weight)	340	210	190	450	120	20	60	120	220

Capacity = 1000

می‌توانیم وزن‌های نظیر آیتم‌هایی که نظیر آن‌ها در رشتهٔ ما عدد ۱ ظاهر شده است را با یکدیگر جمع کنیم. اما این‌کار چه ایرادی دارد؟ برای مواجهه با این مشکل چه می‌توان کرد؟

$$f(101010101) = 39$$
$$f(010101010) = 34$$
$$f(100011011) = 37$$
$$f(111111111) = 32$$
$$f(111110000) = 32$$
$$f(111111000) = 32$$

(با تشکر از آقای نوحی!)

Minimum Balanced Cut

INSTANCE: Graph $G = (V, E)$, with $|V| = 2n$, edge weight function $w : E \mapsto Z^+$.

SOLUTION: A partition of V into subsets V_1, V_2, with $|V_1| = |V_2| = n$.

MEASURE: $\sum_{\substack{(u,v)\in E:\\ u\in V_1, v\in V_2}} w(u,v)$

In this case, a solution (i.e., a partition (V_1, V_2) of the set of nodes $\{v_1, v_2, \ldots, v_{2n}\}$) can be encoded by a binary string σ of $2n$ bits such that the i-th bit σ_i is 0 (respectively, 1) if $v_i \in V_1$ (respectively, $v_i \in V_2$).

در یک جواب قابل‌قبول برای این مسئله، تعداد صفرها باید برابر با تعداد یک‌ها باشد. چطور می‌توان برقراری این شرط را تضمین کرد؟

☜ راه اول: در جمعیت اولیه، همهٔ افراد باید واجد شرط فوق باشند، و عملگرهای mutation و crossover باید به‌گونه‌ای تعریف شوند که برقراری این شرط را مخدوش نشود. در این صورت، تابع هدف می‌تواند به‌شکل زیر تعریف شود:

The fitness evaluation function can be defined as follows. Let c_{ij} be the weight associated to edge (v_i, v_j), and let us assume that $c_{ij} = 0$ if there is no edge (v_i, v_j). Then, the evaluation function ϕ is defined as

$$C - \sum_{i=1}^{2n} \sum_{j=1}^{2n} c_{ij} \sigma_i (1 - \sigma_j)$$

(The fitness has to be maximized.)

☜ راه دوم: می‌توان شرط برابر بودن تعداد صفرها و یک‌ها در هر فرد از جمعیت را برداشت، اما برای نقض این شرط باید جریمهٔ سنگینی وضع کرد:

$$\phi(\sigma) = C - \left(\sum_{i=1}^{2n} \sum_{j=1}^{2n} c_{ij} \sigma_i (1 - \sigma_j) + K \left| n - \sum_{i=1}^{2n} \sigma_i \right| \right)$$

The expression $K \left| n - \sum_{i=1}^{2n} \sigma_i \right|$, where K is a suitably large constant, is introduced to significantly decrease the fitness of unfeasible solutions. Finally, C is another suitably large constant. (The fitness has to be maximized.)

Real-Valued or Floating-Point Representation

Example: A nonlinear programming problem with simple bound constraints (or box constraints):

$$\text{Maximize} \qquad f(x_1, \ldots, x_n)$$

$$\text{subject to}$$

$$L_j \leq x_j \leq U_j, \quad \text{for } j = 1, \ldots, n,$$

Example: A small nonlinear programming problem (only a single variable):

$$\text{Maximize} \qquad f(x) = 12x^5 - 975x^4 + 28{,}000x^3 - 345{,}000x^2 + 1{,}800{,}000x,$$

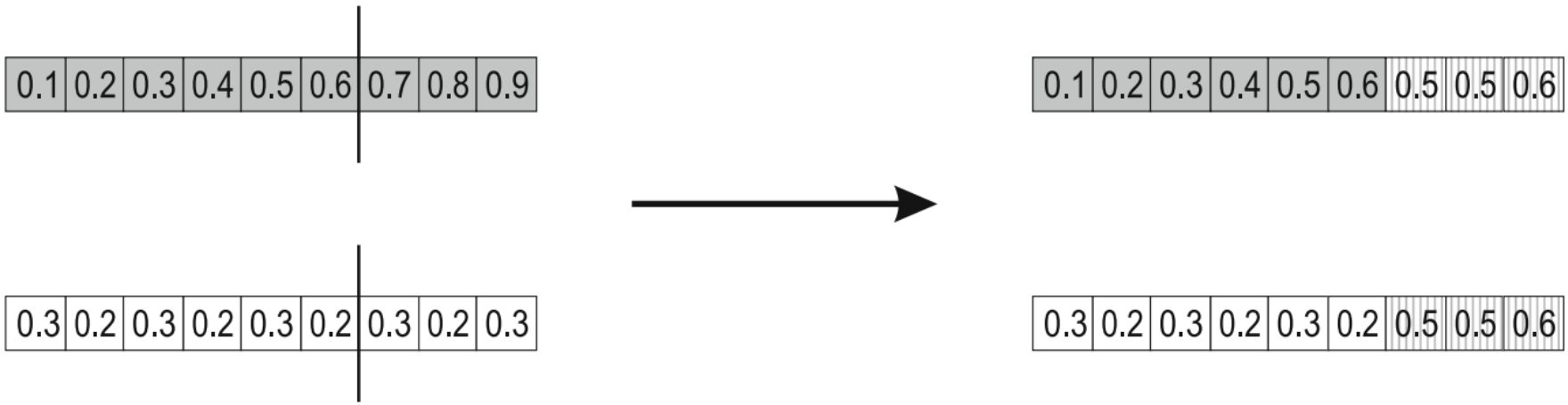
$$\text{subject to}$$

$$0 \leq x \leq 31.$$

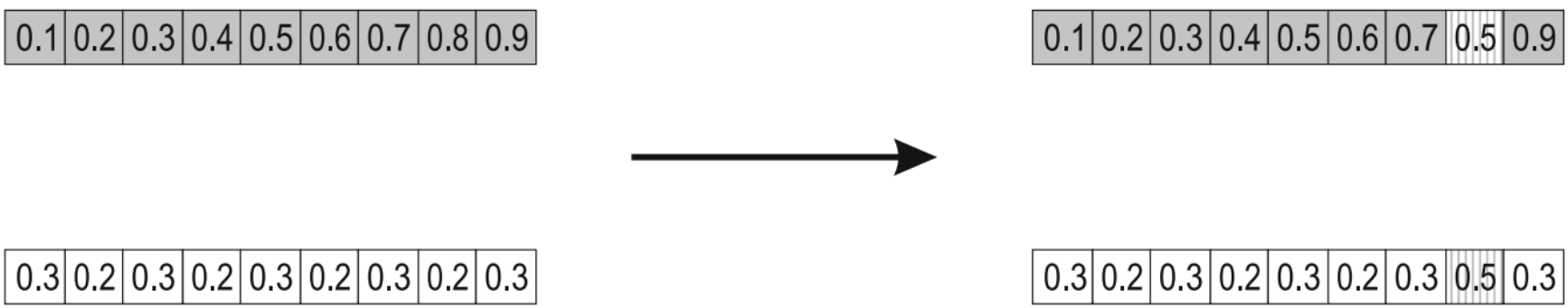
The genotype is now a vector $\langle x_1, x_2, \ldots, x_n \rangle$ **with** $x_i \in [L_i, U_i]$.

Recombination Operators for Real-Valued Representation

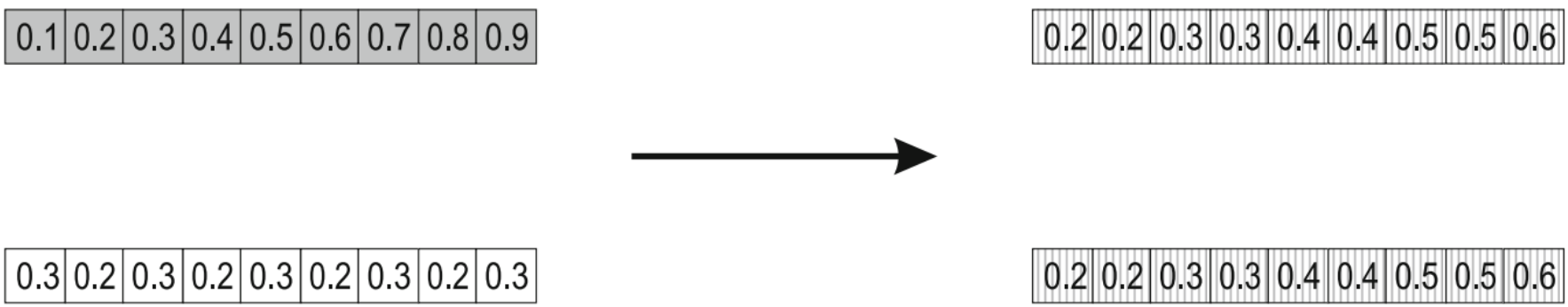
1. Simple arithmetic recombination with $k = 6, \alpha = \frac{1}{2}$:



2. Single arithmetic recombination with $k = 8, \alpha = \frac{1}{2}$:



3. Whole arithmetic recombination with $\alpha = \frac{1}{2}$:



Mutation for Real-Valued Representation

It is common to change the allele value of each gene randomly within its domain given by a lower L_i and upper U_i bound, resulting in the following transformation:

$$\langle x_1, x_2, \ldots, x_n \rangle \rightarrow \langle x'_1, x'_2, \ldots, x'_n \rangle, \text{ where } x_i, x'_i \in [L_i, U_i].$$

Uniform Mutation: For this operator the values of x'_i are drawn uniformly randomly from $[L_i, U_i]$.

WalkSAT

We have seen several local search algorithms so far in this book, including HC and SA. These algorithms can be applied directly to satisfiability problems, provided that we choose the right evaluation function.

Because the goal is to find an assignment that satisfies every clause, an evaluation function that counts the number of unsatisfied clauses will do the job.

WalkSAT

In recent years, there has been a great deal of experimentation to find a good balance between greediness and randomness.

One of the simplest and most effective algorithms to emerge from all this work is called WalkSAT. On every iteration, the algorithm picks an unsatisfied clause and picks a symbol in the clause to flip. It chooses randomly between two ways to pick which symbol to flip: (1) a “min-conflicts” step that minimizes the number of unsatisfied clauses in the new state and (2) a “random walk” step that picks the symbol randomly.

WalkSAT

function WALKSAT(*clauses*, p, *max_flips*) **returns** a satisfying model or *failure*
  inputs: *clauses*, a set of clauses in propositional logic
    p, the probability of choosing to do a "random walk" move, typically around 0.5
    max_flips, number of flips allowed before giving up

  model ← a random assignment of *true*/*false* to the symbols in *clauses*
  for $i = 1$ **to** *max_flips* **do**
    if *model* satisfies *clauses* **then return** *model*
    clause ← a randomly selected clause from *clauses* that is false in *model*
    with probability p flip the value in *model* of a randomly selected symbol from *clause*
    else flip whichever symbol in *clause* maximizes the number of satisfied clauses
  return *failure*

When WalkSAT returns a model, the input sentence is indeed satisfiable, but when it returns failure, there are two possible causes: either the sentence is unsatisfiable or we need to give the algorithm more time.