



Constraint Handling

Leandro L. Minku

Overview

- Dealing with constraints:
 - By carefully designing algorithm operators.
 - By modifying the objective function.
- Completeness of Hill Climbing and Simulated Annealing.

Applying Simulated Annealing (and Hill-Climbing)

We need to specify:

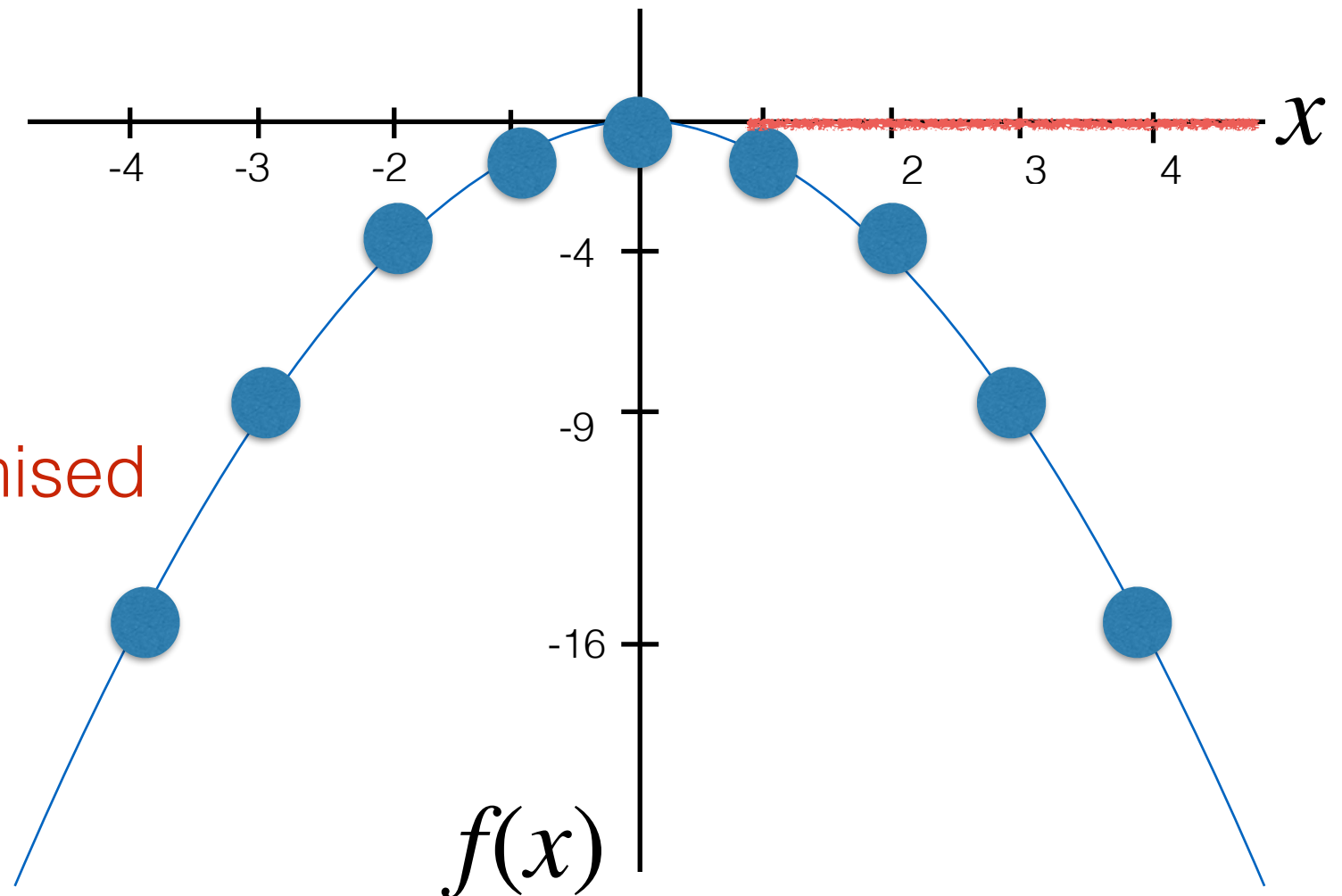
- Optimisation problem formulation:
 - Design variable and search space
 - Objective function
 - Constraints
- Algorithm operators:
 - Representation.
 - Initialisation procedure.
 - Neighbourhood operator.
- Strategy to deal with constraints.
 - Algorithm operators.
 - Objective function.

Illustrative Example — Problem Formulation

- Design variable:
 - $x \in \mathbb{Z}$
 - Search space: \mathbb{Z} .

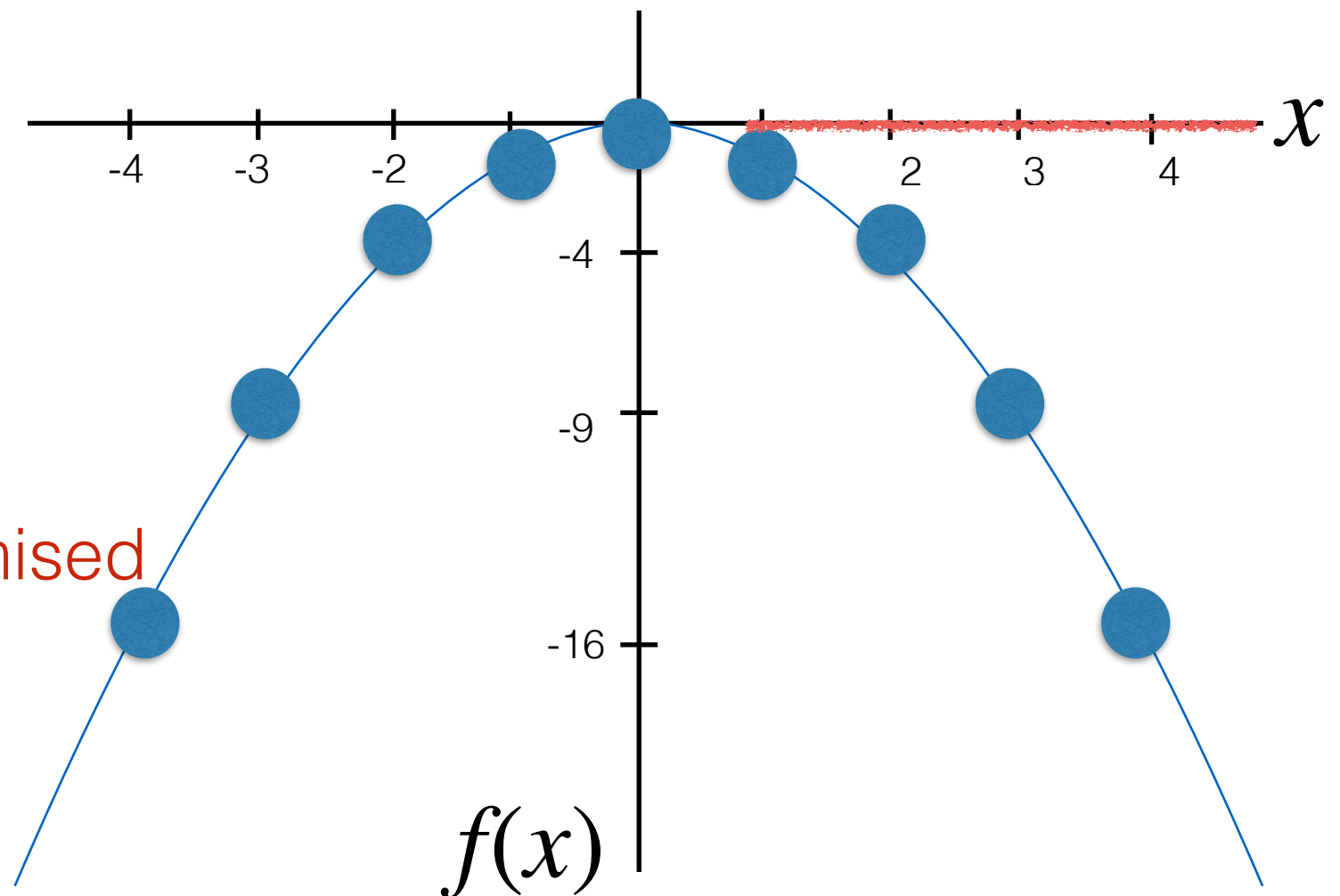
- Objective function:
 - $f(x) = -x^2$, to be minimised

- Constraints:
 - $g(x) = x \leq 0$.



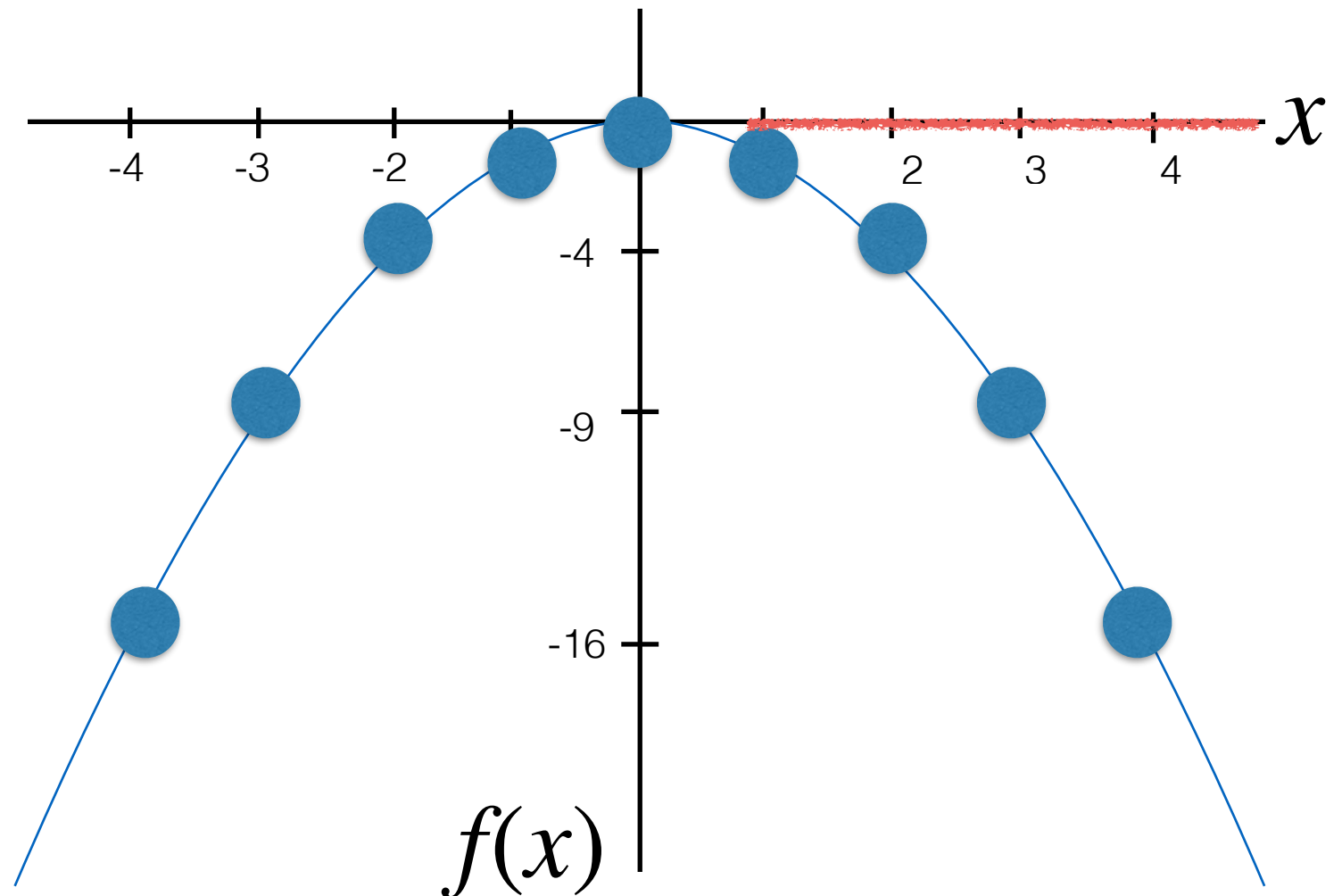
Illustrative Example — Problem Formulation

- Design variable:
 - $x \in \{\dots, -3, -2, -1, 0\}$
 - Search space:
 $\{\dots, -3, -2, -1, 0\}$.
- Objective function:
 - $f(x) = -x^2$, to be minimised
- Constraints:
 - No explicit constraint.



Illustrative Example — Algorithm Operators

- Representation:
 - Integer variable.
- Initialisation procedure:
 - Initialise with an integer value picked uniformly at random.
- Neighbourhood operator:
 - Add or subtract 1.



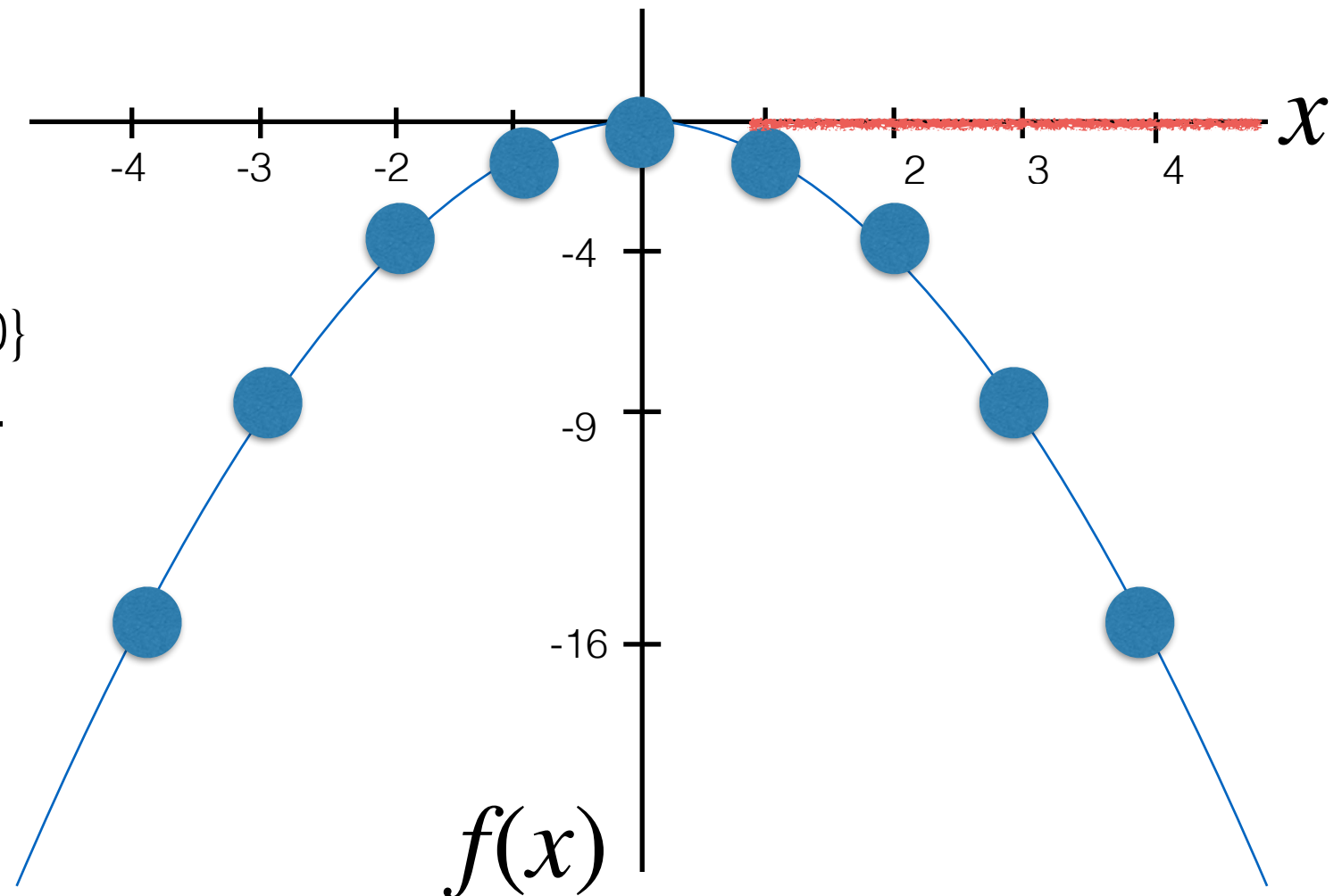
Hill Climbing and Simulated Annealing may return an infeasible solution.

How to Solve This Issue?

Dealing With Constraints Based on Algorithm Operators

Dealing With Constraints Based on Algorithm Operators

- Representation:
 - Integer variable x .
- Initialisation procedure:
 - Initialise with an integer value $x \in \{\dots, -4, -3, -2, -1, 0\}$ picked uniformly at random.
- Neighbourhood operator:
 - Add or subtract 1.
 - If a neighbour has $x > 0$, fix it by setting its x to 0.
- Constraint strategy:
 - Operators above



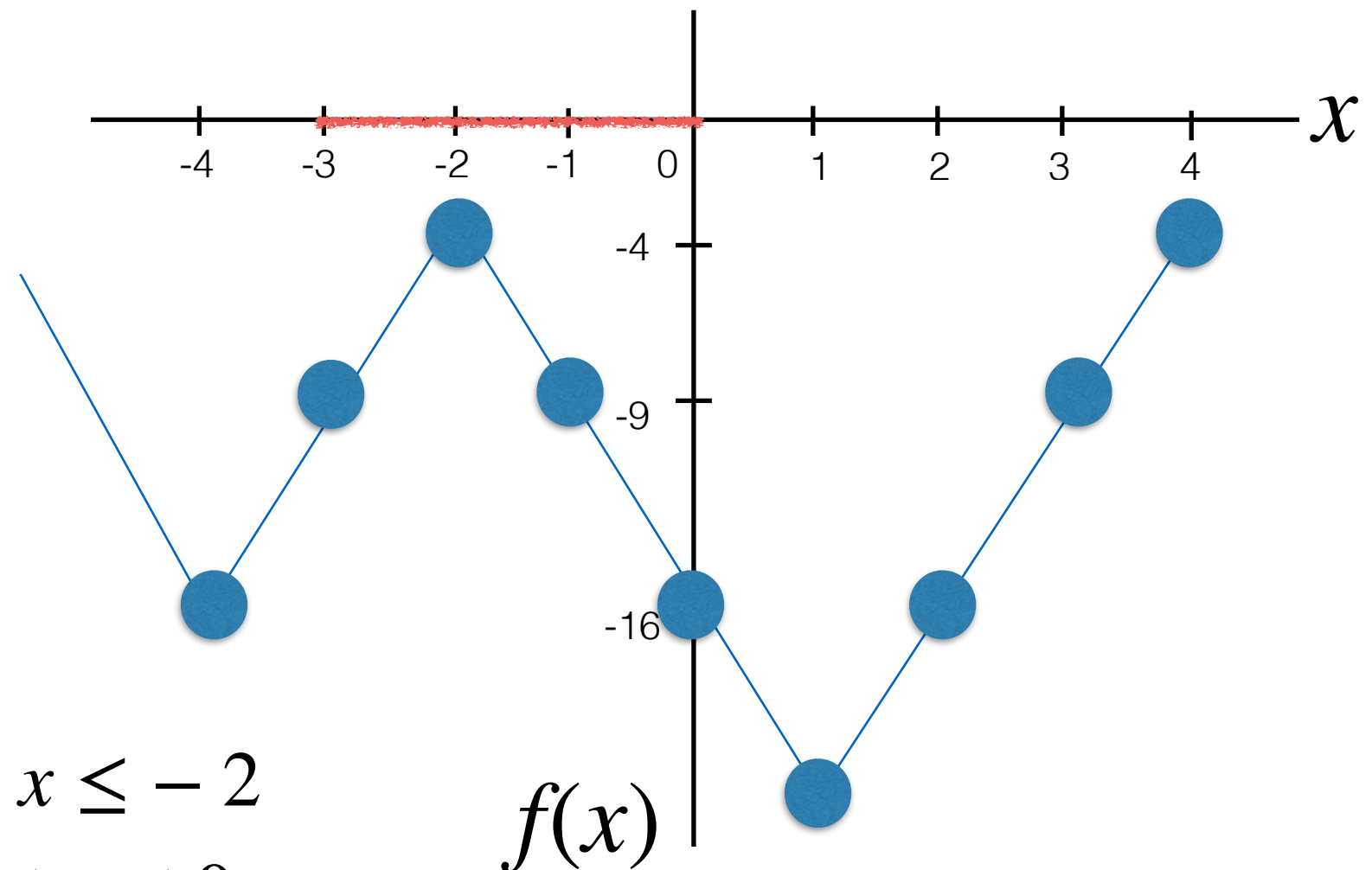
Dealing with Constraints Based on Algorithm Operators

- Advantage:
 - Ensure that no infeasible candidate solutions will be generated, facilitating the search for optimal solutions.
- Disadvantage:
 - May be difficult to design, and the design is problem-dependent.
 - Sometimes, it could restrict the search space too much, making it difficult to find the optimal solution.

Illustrative Example of Disadvantage

- Design variable:
 - $x \in \mathbb{Z}$
 - Search space: \mathbb{Z} .
- Objective function:
 - $f(x)$, to be minimised
- Constraints: $h(x) = 0$

$$h(x) = \begin{cases} x + 4, & \text{if } -3 \leq x \leq -2 \\ -x + 1, & \text{if } -1 \leq x \leq 0 \\ 0, & \text{otherwise} \end{cases}$$



Illustrative Example of Disadvantage

Cannot move to the region containing the optimum

- Representation:

- Integer variable x .

- Initialisation procedure:

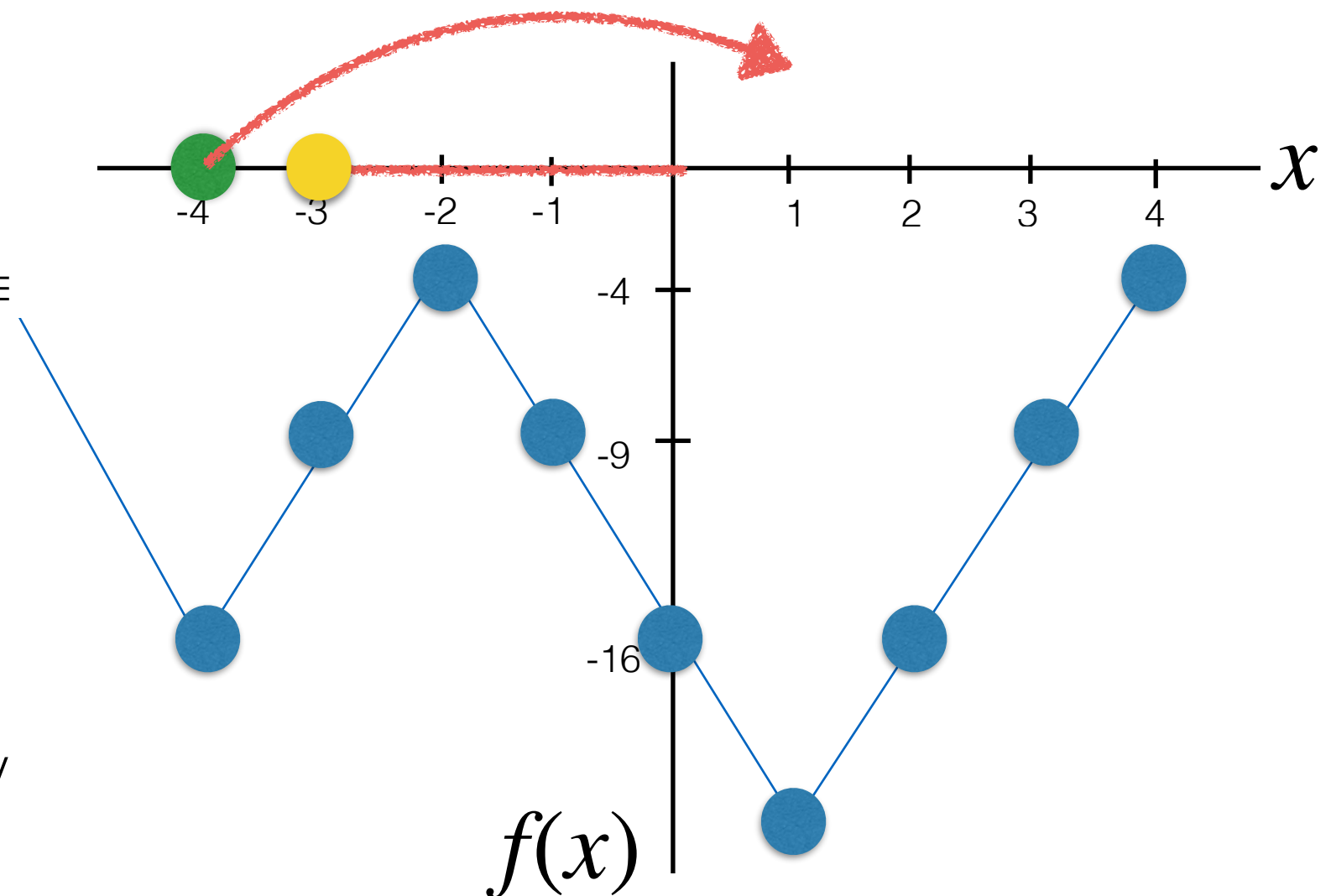
- Initialise with an integer value $x \in \{\dots, -6, -5, -4, 1, 2, 3, \dots\}$ picked uniformly at random.

- Neighbourhood operator:

- Add or subtract 1.
- If a neighbour is in $\{-3, -2\}$, fix it by setting its x to -4.
- If a neighbour is in $\{-1, 0\}$, fix it by setting its x to 1.

- Constraint strategy:

- Operators above



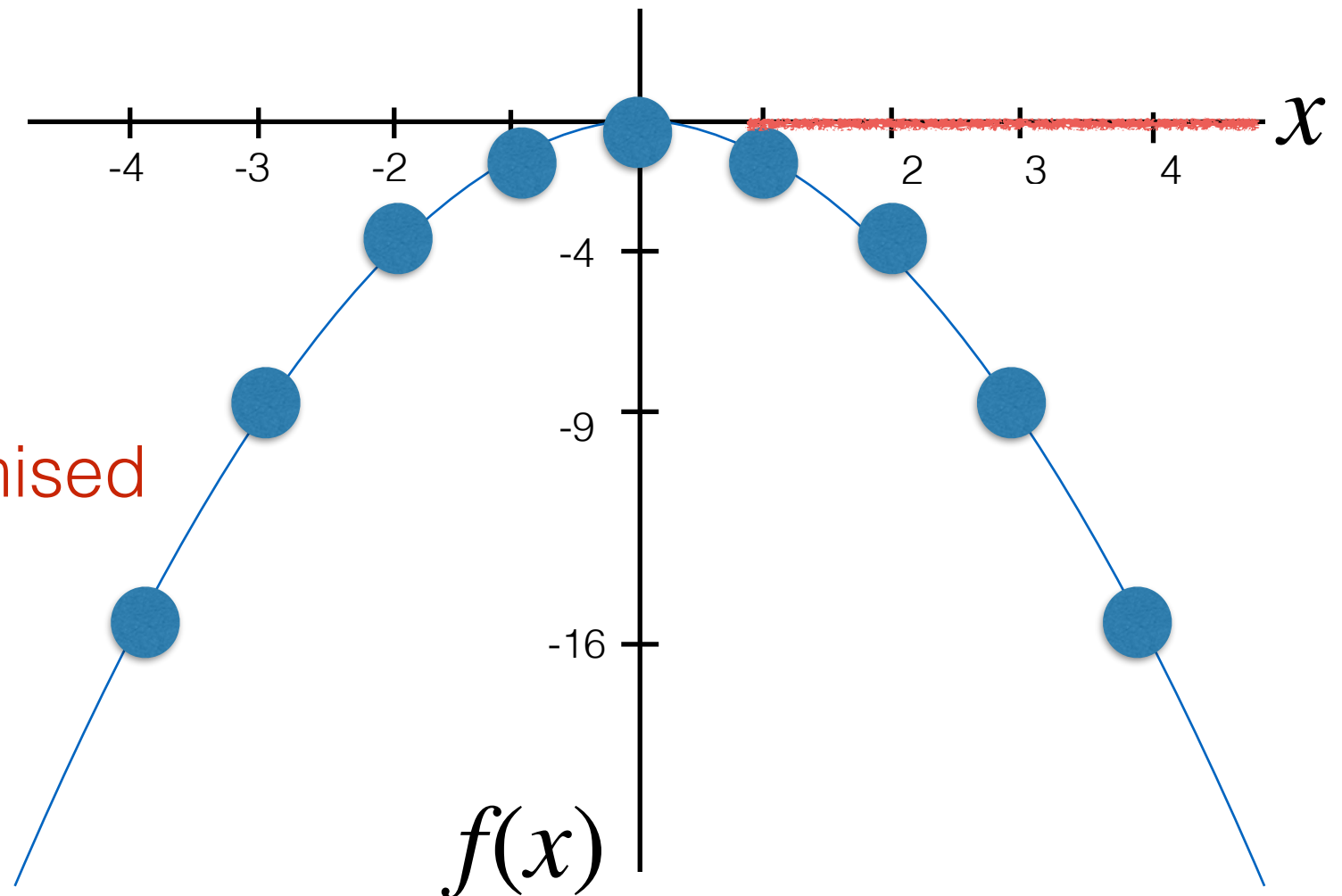
Dealing With Constraints By Modifying the Objective Function

Illustrative Example — Problem Formulation

- Design variable:
 - $x \in \mathbb{Z}$
 - Search space: \mathbb{Z} .

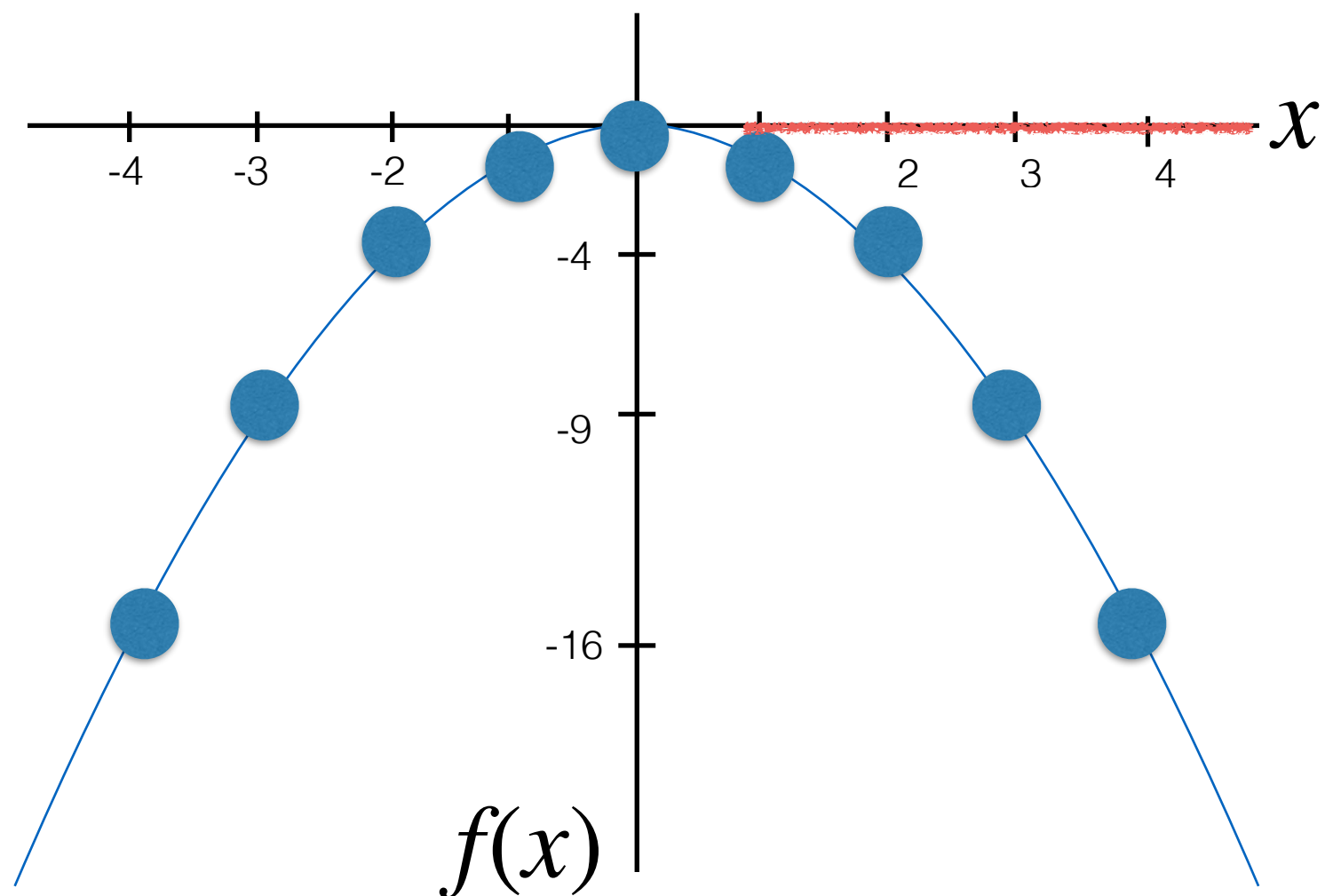
- Objective function:
 - $f(x) = -x^2$, to be minimised

- Constraints:
 - $g(x) = x \leq 0$.



Illustrative Example — Algorithm Operators

- Representation:
 - Integer variable.
- Initialisation procedure:
 - Initialise with an integer value picked uniformly at random.
- Neighbourhood operator:
 - Add or subtract 1.






Death Penalty

Death Penalty

Minimise $f(\mathbf{x})$

Subject to $g_i(\mathbf{x}) \leq 0, \quad i = 1, \dots, m$

$h_j(\mathbf{x}) = 0, \quad j = 1, \dots, n$

Minimise $f(\mathbf{x}) + Q(\mathbf{x})$ 

$$Q(\mathbf{x}) = \begin{cases} 0 & \text{if } \mathbf{x} \text{ is feasible} \\ C & \text{otherwise} \end{cases}$$

where C is a large positive constant ensuring that all infeasible solutions have worse objective value than any feasible solution.

Death Penalty

Minimise $f(\mathbf{x})$

Subject to $g_i(\mathbf{x}) \leq 0, \quad i = 1, \dots, m$

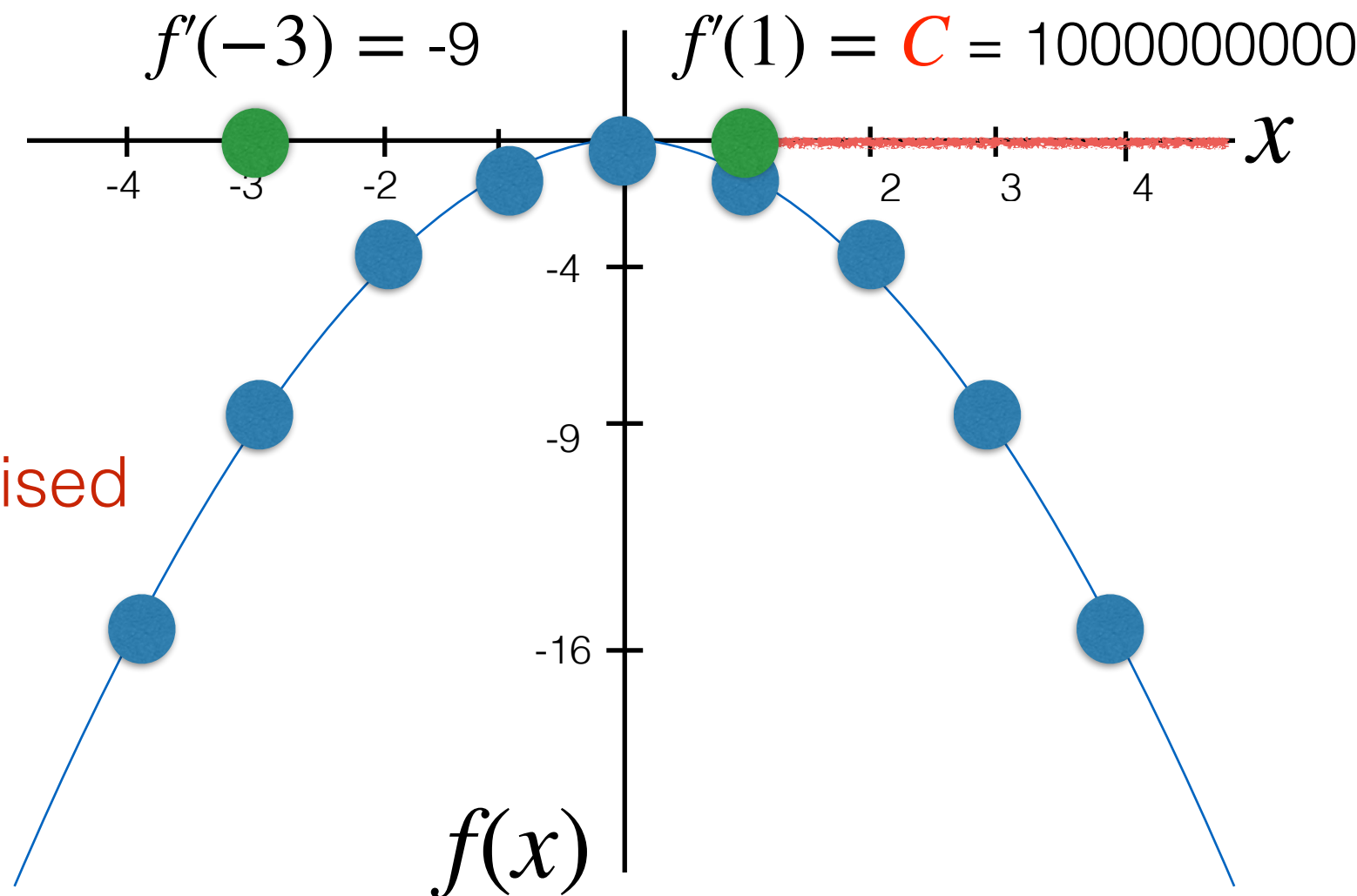
$h_j(\mathbf{x}) = 0, \quad j = 1, \dots, n$

$$f'(\mathbf{x}) = \begin{cases} f(\mathbf{x}) & \text{if } \mathbf{x} \text{ is feasible} \\ C & \text{otherwise} \end{cases}$$

where C is a large positive constant ensuring that all infeasible solutions have worse objective value than any feasible solution.

Illustrative Example — Problem Formulation

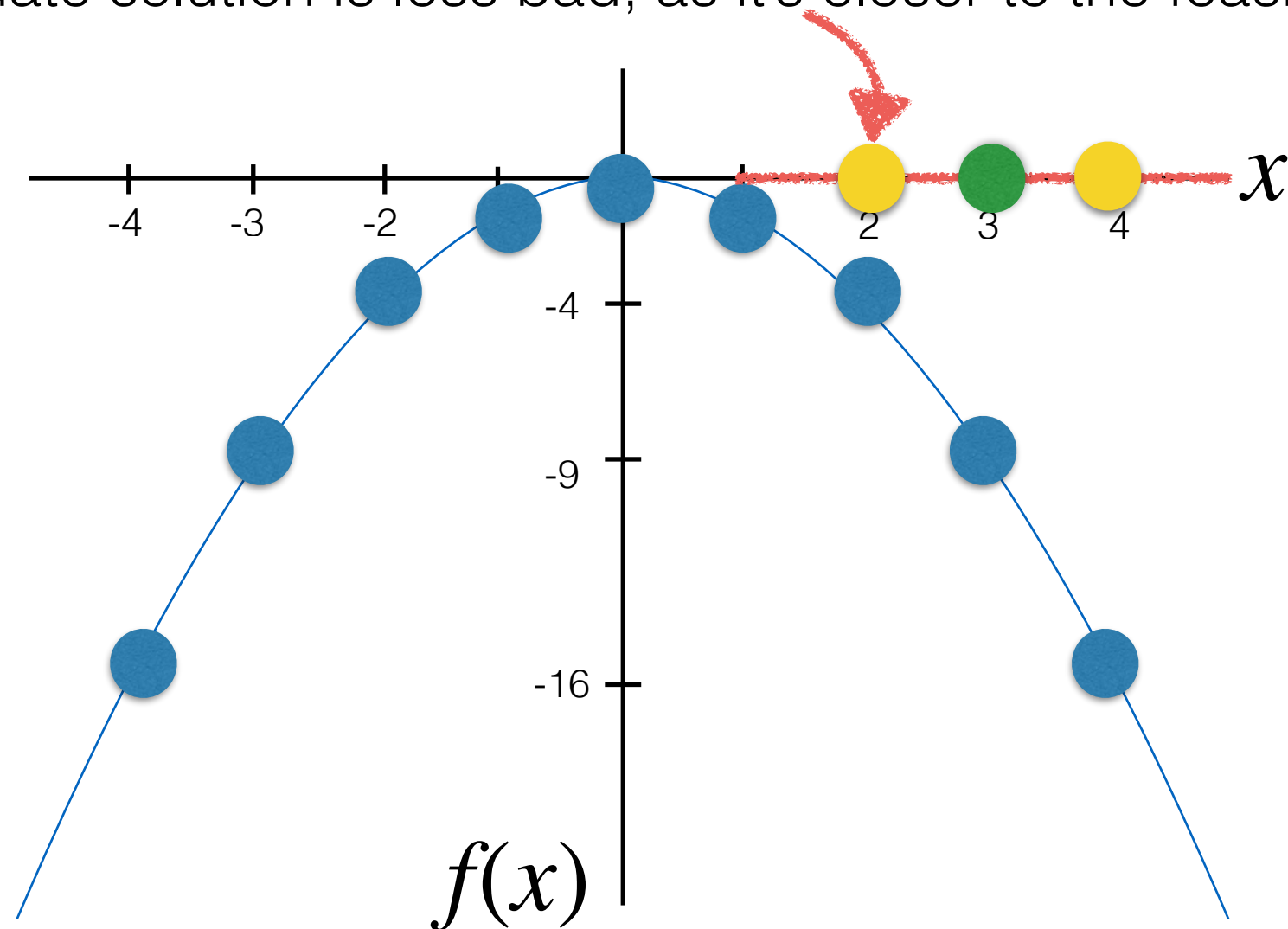
- Design variable:
 - $x \in \mathbb{Z}$
 - Search space: \mathbb{Z} .
- Objective function:
 - $f(x) = -x^2$, to be minimised
- Constraints:
 - $g(x) = x \leq 0$.



What value of $\textcolor{red}{C}$ would have been enough for this problem? $\textcolor{red}{C} = 1$

Weakness of Death Penalty

This candidate solution is less bad, as it's closer to the feasible region

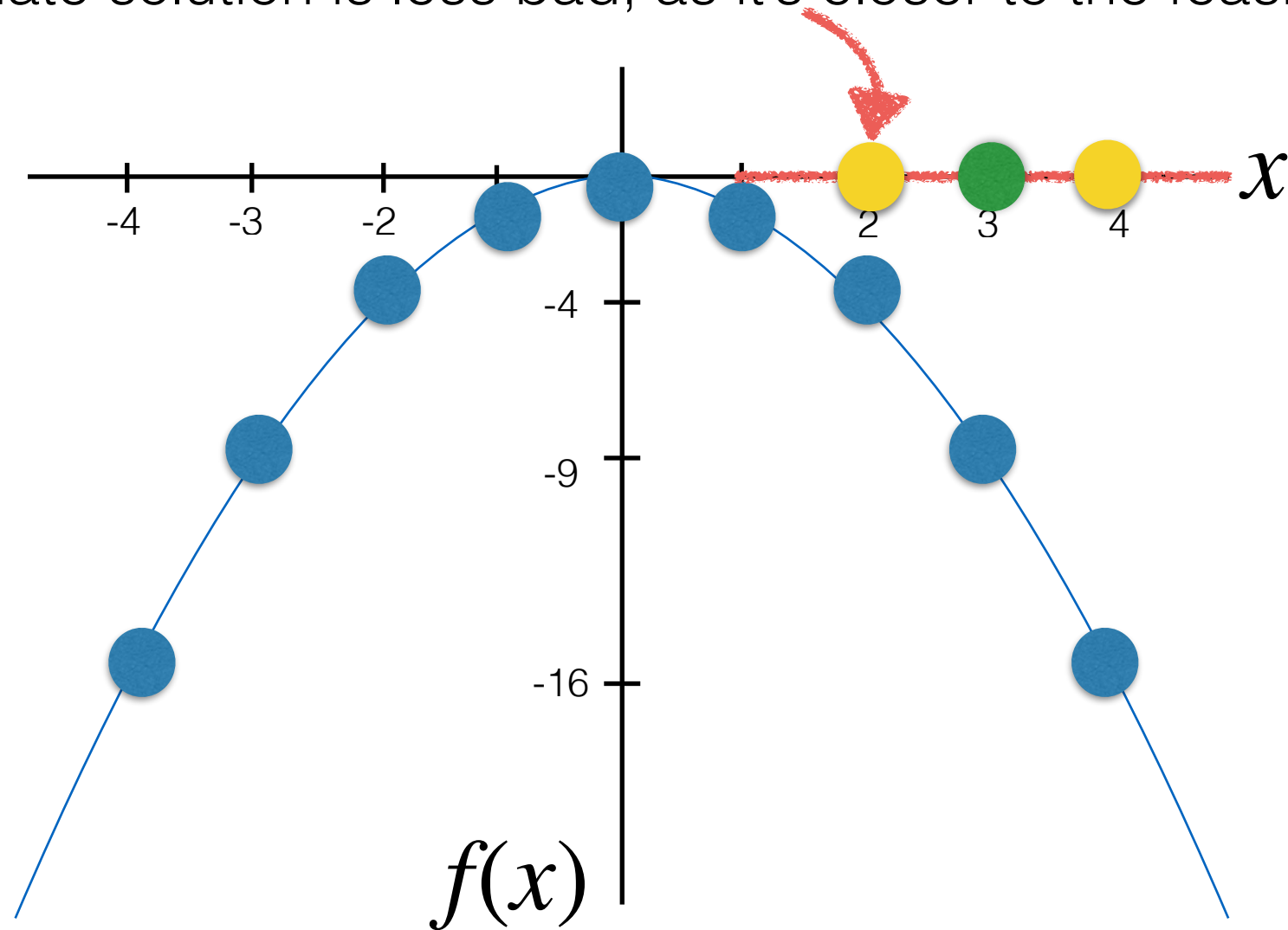


Problem: all infeasible solutions have the same penalty.

This will make it difficult for algorithms to find feasible solutions when the current solution is in regions of the space dominated by infeasible solutions.

Using the Level of Infeasibility

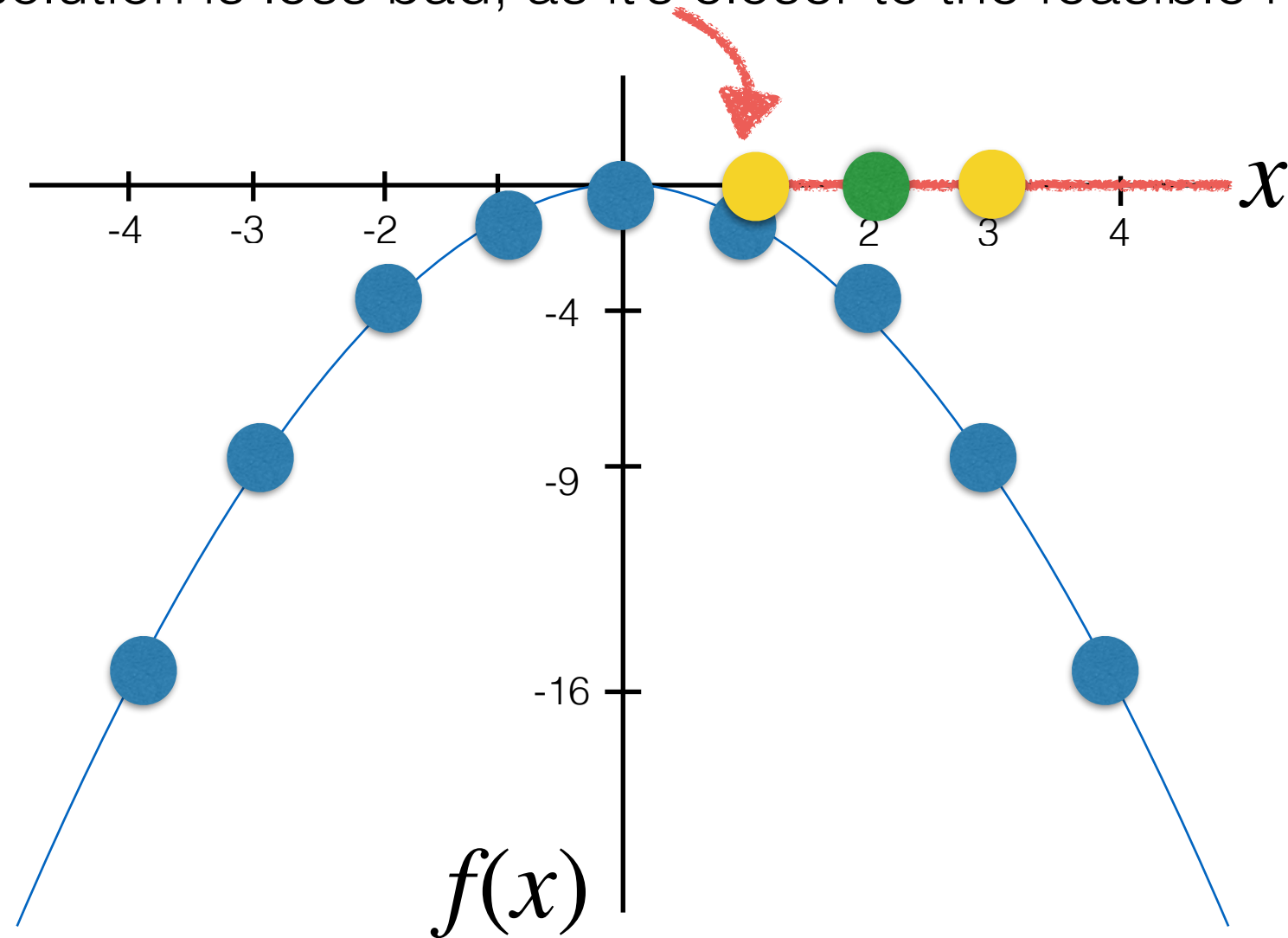
This candidate solution is less bad, as it's closer to the feasible region



Distinguishing the objective value of infeasible solutions could help the algorithm to reach a feasible region.

Using the Level of Infeasibility

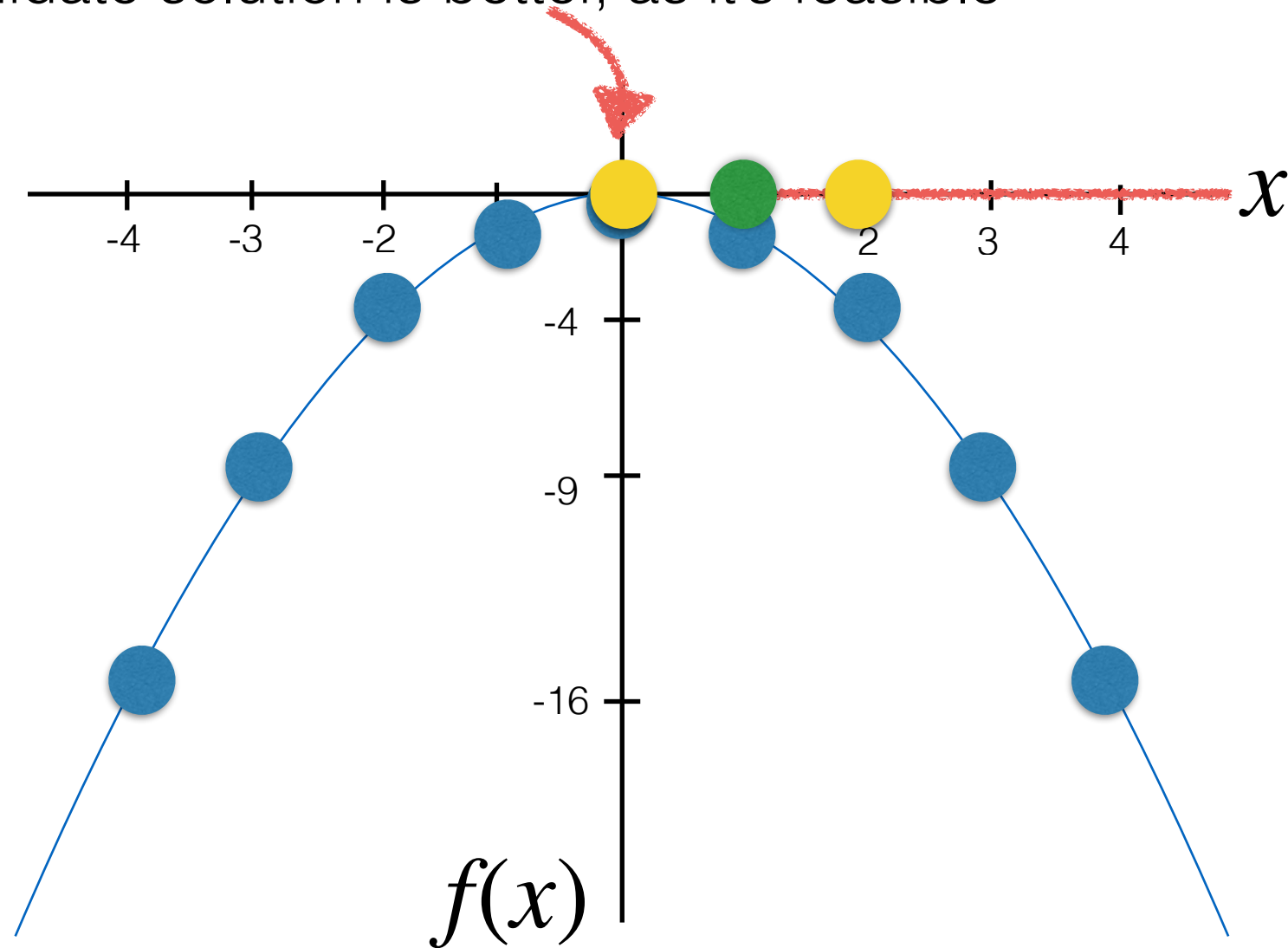
This candidate solution is less bad, as it's closer to the feasible region



Distinguishing the objective value of infeasible solutions could help the algorithm to reach a feasible region.

Using the Level of Infeasibility

This candidate solution is better, as it's feasible



Distinguishing the objective value of infeasible solutions could help the algorithm to reach a feasible region.

Using the Level of Infeasibility

Minimise $f(\mathbf{x}) + Q(\mathbf{x})$

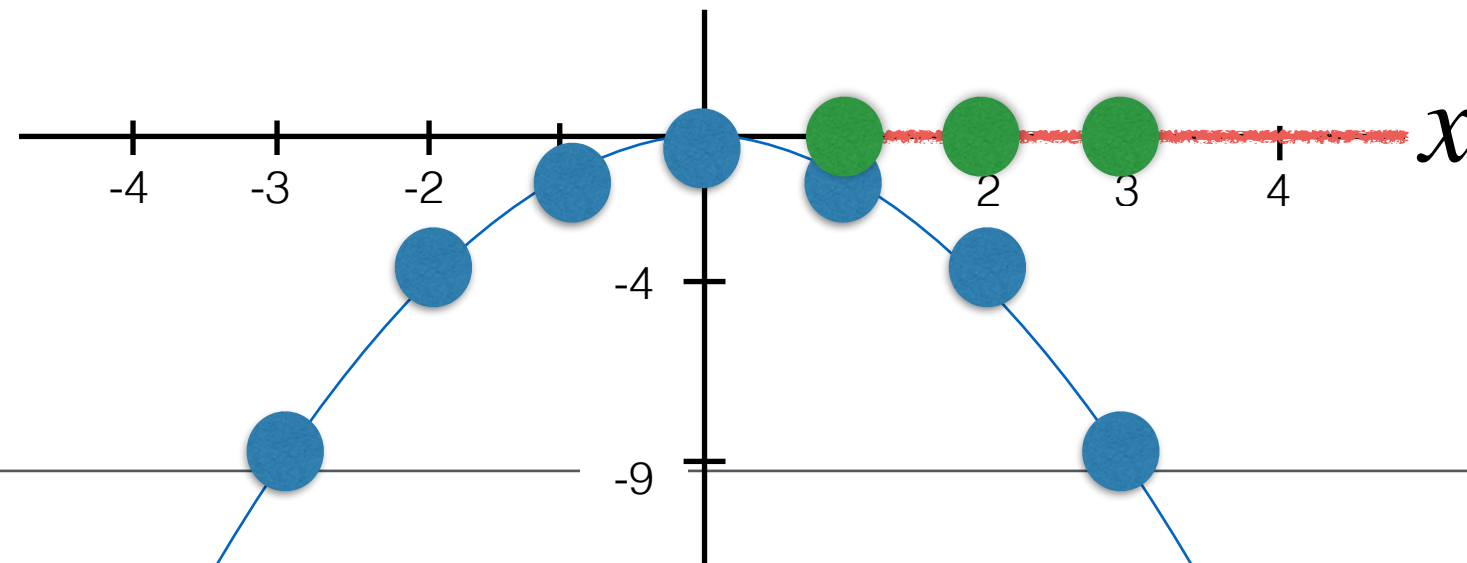
$x \leq 0$ $Q(x) = Cx$ where C is our large positive constant.

For example, if $x = 1$, $Q(x) = C * 1 = C$.


if $x = 2$, $Q(x) = C * 2 = 2C$.

if $x = 3$, $Q(x) = C * 3 = 3C$.

The more severely
the constraint is
violated, the
higher the penalty.



Using the Level of Infeasibility

Minimise $f(\mathbf{x}) + Q(\mathbf{x})$ 

$x \leq 0$ $Q(x) = Cx$ where C is our large positive constant.

For example, if $x = 1$, $Q(x) = C * 1 = C$.

if $x = 2$, $Q(x) = C * 2 = 2C$.

if $x = 3$, $Q(x) = C * 3 = 3C$.

The more severely
the constraint is
violated, the
higher the penalty.

So, the objective function itself will guide the algorithm towards finding feasible solutions, in the same way as it guides the algorithm towards finding (near-) optimal solutions.

Using the Level of Infeasibility

Minimise $f(\mathbf{x}) + Q(\mathbf{x})$

$$-x \leq 0$$

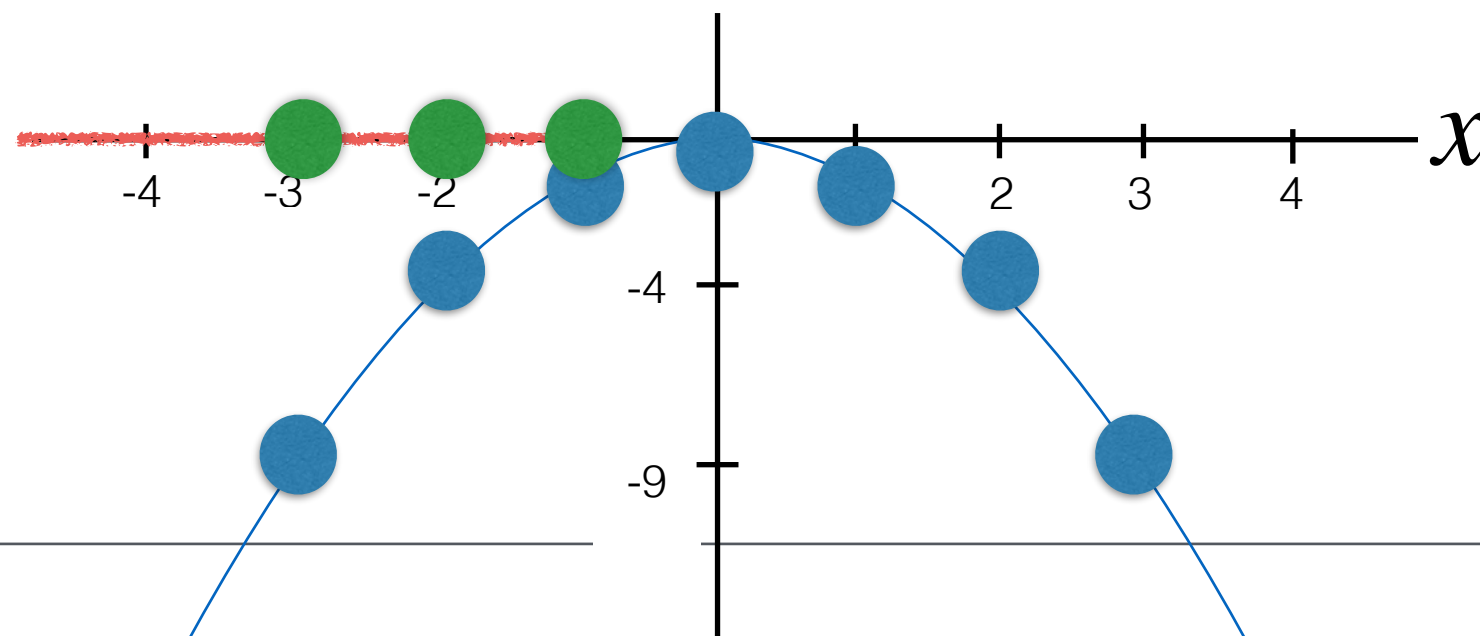
$$Q(x) = C(-x)$$

where C our large positive constant.

$$x \geq 0$$

For example, if $x = -1$, $Q(x) = C * (-(-1)) = C$.
if $x = -2$, $Q(x) = C * (-(-2)) = 2C$.
if $x = -3$, $Q(x) = C * (-(-3)) = 3C$.

The more severely the constraint is violated, the higher the penalty.



Using the Level of Infeasibility

Minimise $f(\mathbf{x}) + Q(\mathbf{x})$

$$Q(x) = Cx + C(x - 10) + \dots$$

$$x \leq 0 \quad x - 10 \leq 0$$

$$x \leq 10$$

$$Q(x) = Cg_1(\mathbf{x}) + Cg_2(\mathbf{x}) + \dots + Cg_m(\mathbf{x})$$

$$+ C|h_1(\mathbf{x})| + C|h_2(\mathbf{x})| + \dots + C|h_n(\mathbf{x})|$$

We should only add the penalties corresponding to violated constraints.

v_{g_i} is 1 if g_i is violated and 0 otherwise

v_{h_i} is 1 if h_i is violated and 0 otherwise

Using the Level of Infeasibility

Minimise $f(\mathbf{x}) + Q(\mathbf{x})$

$$Q(x) = Cx + C(x - 10) + \dots$$

$$x \leq 0 \quad x - 10 \leq 0$$

$$x \leq 10$$

$$Q(x) = v_{g_1} C g_1(\mathbf{x}) + v_{g_2} C g_2(\mathbf{x}) + \dots + v_{g_m} C g_m(\mathbf{x}) \\ + v_{h_1} C |h_1(\mathbf{x})| + v_{h_2} C |h_2(\mathbf{x})| + \dots + v_{h_n} C |h_n(\mathbf{x})|$$

We should only add the penalties corresponding to violated constraints.

v_{g_i} is 1 if g_i is violated and 0 otherwise

v_{h_i} is 1 if h_i is violated and 0 otherwise

Using the Level of Infeasibility

Minimise $f(\mathbf{x}) + Q(\mathbf{x})$

$$Q(x) = Cx + C(x - 10) + \dots$$

$$x \leq 0 \quad x - 10 \leq 0$$

$$x \leq 10$$

$$Q(x) = v_{g_1} C g_1(\mathbf{x})^2 + v_{g_2} C g_2(\mathbf{x})^2 + \dots + v_{g_m} C g_m(\mathbf{x})^2 \\ + v_{h_1} C h_1(\mathbf{x})^2 + v_{h_2} C h_2(\mathbf{x})^2 + \dots + v_{h_n} C h_n(\mathbf{x})^2$$

Squared values give larger distinction between the objective values of different infeasible solutions

Dealing with Constraints Based on Objective Functions

- Advantage:
 - May be easier to design.
- Disadvantage:
 - Algorithm has to search for feasible solutions.

Completeness

- If we use a strategy to deal with constraints that never enables any infeasible solution to be generated, algorithms such as Hill Climbing and Simulated Annealing are complete.
- Otherwise:
 - **Hill Climbing**: not complete if the objective function has local optima (a local optimum may be infeasible).
 - **Simulated Annealing**: not guaranteed to find a feasible solution within a reasonable amount of time (we may stop before finding a feasible solution).

Summary

- We need to design strategies to deal with the constraints.
- Examples of strategies:
 - Representation, initialisation and neighbourhood operators.
 - Objective function.

Next

- Example application.