Algorithm 1 Non-dominated Sorting

Require: A population of individuals P

for each individual $i \in P$ **do**
  Set $S_i := \emptyset$ and $n_i := 0$
end for
for all pairs $i, j \in P$, $i \neq j$ **do**
  if j dominates i **then**
    $S_j := S_j \cup \{i\}$
  else if i dominates j **then**
    $n_j := n_j + 1$
  end if
end for
for each $i \in P$ **do**
  If $n_i = 0$, keep i in the first non-dominated front P_1
end for
Set $k = 1$
while $P_k \neq \emptyset$ **do**
  for each $i \in P_k$ and $j \in S_i$ **do**
    Set $n_j := n_j - 1$
    if $n_j = 0$ **then**
      Update $Q := Q \cup \{j\}$
    end if
  end for
  Set $k = k + 1$ and $P_k = Q$ and update $Q := \emptyset$.
end while

Algorithm 2 Crowding Distance[1]

Require: A set $\mathcal{F} = \{(f_1^i, \dots, f_n^i)\}_{i \in [\ell]}$ of ℓ objective vectors

for each $i \in [\ell]$ **do**

  Set $d_i := 0$ ▷ initialise distances to 0

end for

for each objective $m \in [M]$ **do**

  Sort the set $\mathcal{F}$ according to objective f_m s.t.

$$f_m^{(I_1^m)} \leq f_m^{(I_2^m)} \leq \dots \leq f_m^{(I_\ell^m)}$$

  Set $d_{I_1^m} := \infty$ and $d_{I_\ell^m} := \infty$ ▷ so that boundary points are selected

  for $j \in \{2, \dots \ell - 1\}$ **do** ▷ for all other points

    $d_{I_j} := d_{I_j} + \frac{f_m^{(I_{j+1}^m)} - f_m^{(I_{j-1}^m)}}{f_m^{\max} - f_m^{\min}}$

  end for

end for

return the "crowding distances" $(d_1, \dots d_\ell)$

[1] We use the notation $[n] := \{1, 2, \dots, n\}$.

Algorithm 3 NSGA-II (one generation)

Require: A parent population P_t of size N
Require: An offspring population Q_t of size N

Sort $R_t := P_t \cup Q_t$ into non-dominated fronts $\mathcal{F}_1, \mathcal{F}_2, \ldots$
Set $i := 1$ and $P_{t+1} := \emptyset$.
while $|P_{t+1}| + |\mathcal{F}_i| < N$ **do**
    Set $P_{t+1} = P_{t+1} \cup \mathcal{F}_i$ ▷ add the i-th non-dominated front to the parent pop
    Set $i := i + 1$
end while
Perform "crowding sort" on the individuals in front $\mathcal{F}_i$
Add the $N - |P_{t+1}|$ most widely spread solutions by crowding distance to P_{t+1}
Create an offspring population Q_{t+1} from P_{t+1} using crowded tournament selection, crossover, and mutation operators.
Return P_{t+1} and Q_{t+1}