**CROSSOVER**

**Require:** parents, population

**Return:** population

**procedure** onePoint

crosspoint ← random(100)

**begin**

**for** j:= 0 **to** population.size **do**

**if** probability < 70 **then**

**if** j mod 2 ← 0 **then**

**for** i := 0 **to** child.size **do**

color1 ← parent1.color

color2 ← parent2.color

**if** newId < crosspoint **then**

child.vertex(i).color ← color1

**else**

child.vertex(i).color ← color2

**end if**

**end for**

**else**

**for i := 0 to child.size do**

color1 ← parent1.color

color2 ← parent2.color

**if** newId < crosspoint **then**

child.vertex(i).color ← color2

**else**

child.vertex(i).color ← color1

**end if**

**end for**

**end if**

**end if**

**end for**

**end**

**Require:** parents, population

**Return:** population

**procedure** twoPoint

crosspoint1 ← random(100)

crosspoint2 ← random(100)

**begin**

**for** j:= 0 **to** population.size **do**

**if** probability < 70 **then**

**if** j mod 2 ← 0 **then**

**for** i := 0 **to** child.size **do**

color1 ← parent1.color

color2 ← parent2.color

**if** newId < crosspoint1 **or** newId > crosspoint2 **then**

child.vertex(i).color ← color1

**else**

child.vertex(i).color ← color2

**end if**

**end for**

**else**

**for i := 0 to child.size do**

kolor1 ← parent1.color

kolor2 ← parent2.color

**if** newId < crosspoint1 **or** newId > crosspoint2 **then**

child.vertex(i).color ← color2

**else**

child.vertex(i).color ← color1

**end if**

**end for**

**end if**

**end if**

**end for**

**end**

**Require:** parents, population

**Return:** population

**procedure** uniform

**begin**

**for** j:= 0 **to** population.size **do**

**if** probability < 70 **then**

**if** j mod 2 ← 0 **then**

**for** i := 0 **to** child.size **do**

color1 ← parent1.color

color2 ← parent2.color

generator ← random(2)

**if** generator == 0 **then**

child.vertex(i).color ← color1

**else**

child.vertex(i).color ← color2

**end if**

**end for**

**else**

**for i := 0 to child.size do**

color1 ← parent1.color

color2 ← parent2.color

generator ← random(2)

**if** generator == 0 **then**

child.vertex(i).color ← color2

**else**

child.vertex(i).color ← color1

**end if**

**end for**

**end if**

**end if**

**end for**

**end**

**Require:** parents, population, chromosomColors

**Return:** population

**procedure** arithmetic

**begin**

**for** j:= 0 **to** population.size **do**

**if** probability < 70 **then**

**for** i := 0 **to** child.size **do**

color1 ← parent1.color

color2 ← parent2.color

child.vertex(i).color ← (color1 \* color2) % chromosomColors

**end for**

**end if**

**end for**

**end**

**MUTATION**

**Require:** chromosom, chromosomColors

**Return:** chromosom

**procedure** simple

**begin**

**foreach** vertex **in** chromosom **do**

**for** j:= 0 **to** chromosomColors **do**

colorsArray ← j

**end for**

**for** d:=0 **to** vertex.neighbours.size **do**

**if** vertex.neighbours(d).color == vertex.color **then**

**if** colorsArray contains a vertex.neighbours(d).color

remove this color from colorsArray

**end if**

**end if**

**end for**

vertex.color ← color.random(colorsArray.size)

**end foreach**

**end**

**Require:** chromosom, chromosomColors

**Return:** chromosom

**procedure** minMaxTransposition

**begin**

**for** i :=0 **to** chromosom.size **do**

badNeighbours := 0

**for** d:=0 **to**  vertex(i).neighbours.size **do**

**if** vertex(i).neighbours(d).color == vertex(i).color

badNeighbours ← badNeighbours + 1

**break**

**end if**

**end for**

badNeighboursArray(i) ← badNeighbours

**end for**

max ← badNeighboursArray.max

min ← badNeighboursArray.min

chromosom(max).color ← chromosom(min).kolor

**end**

**Require:** chromosom, chromosomColors

**Return:** chromosom

**procedure** randomTransposition

**begin**

**for** i :=0 **to** chromosom.size **do**

**for** d:=0 **to**  vertex(i).neighbours.size **do**

**if** vertex(i).neighbours(d).color == vertex(i).color

badNeighbour ← vertex(i).neighbours(d)

indexBadNeighbour ← i

**break**

**end if**

**end for**

**if** badNeighbours != null **then**

**break**

**end for**

**while** true **do**

newColor ← chromosom.random.color

**if** chromosom(indexBadNeighbour) != newColor **then**

chromosom(indexBadNeighbour) ← newColor

**break**

**end if**

**end while**

**end**