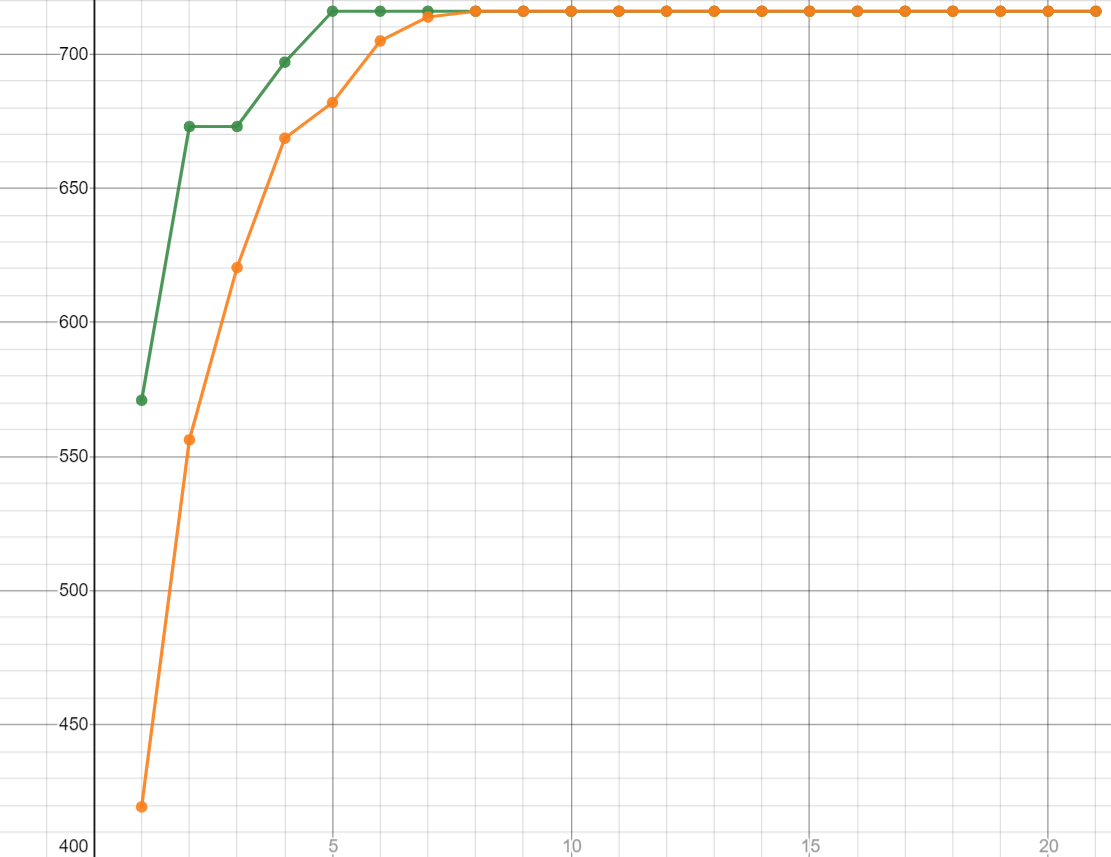
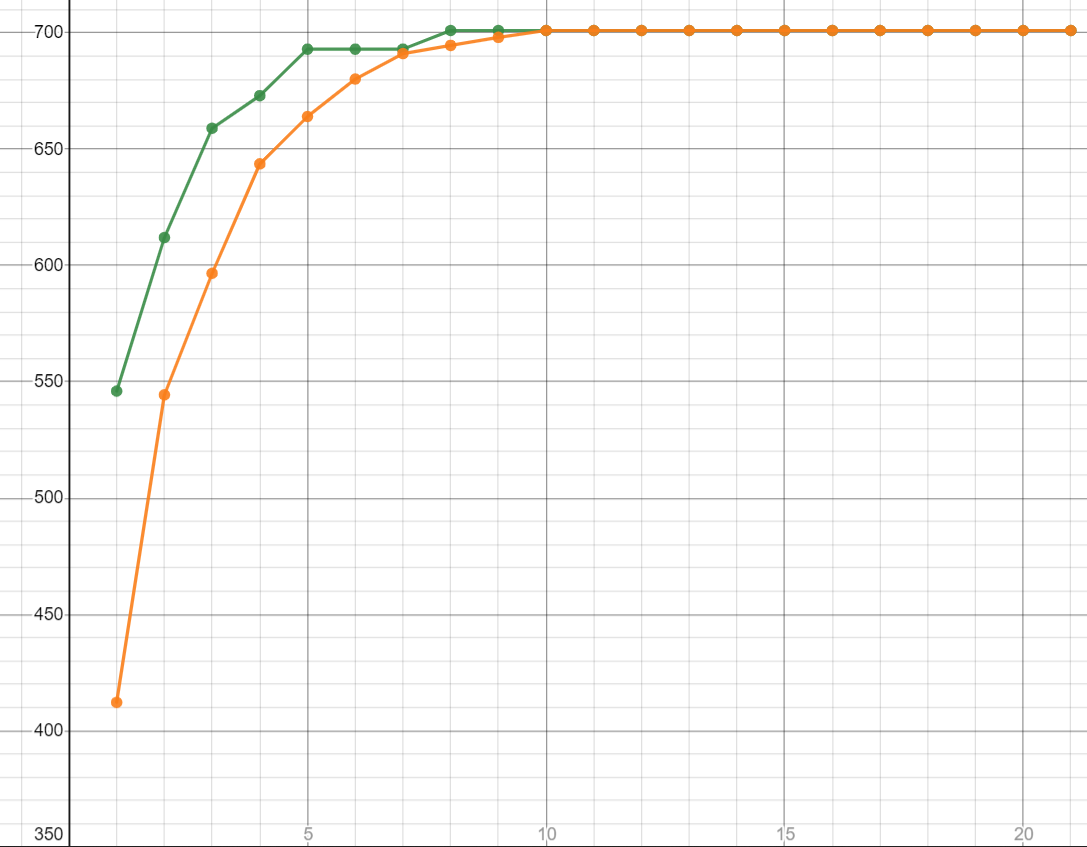
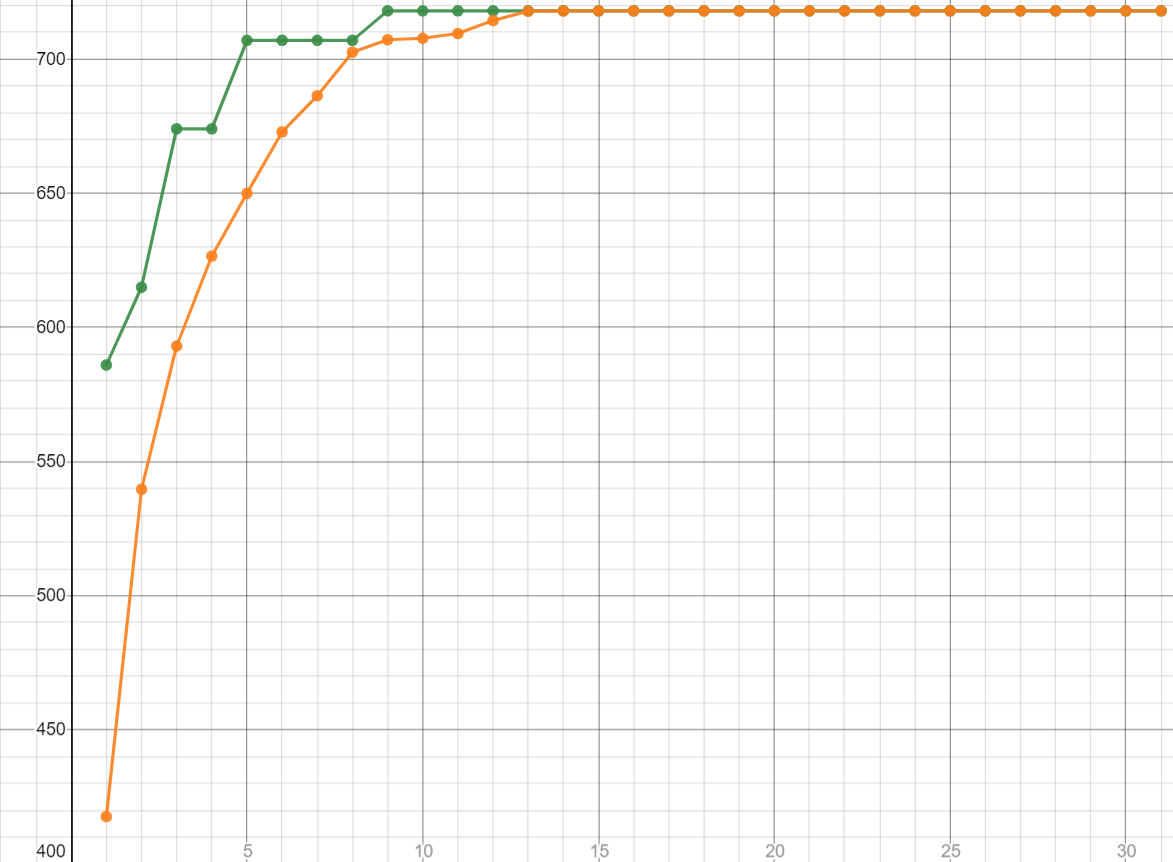
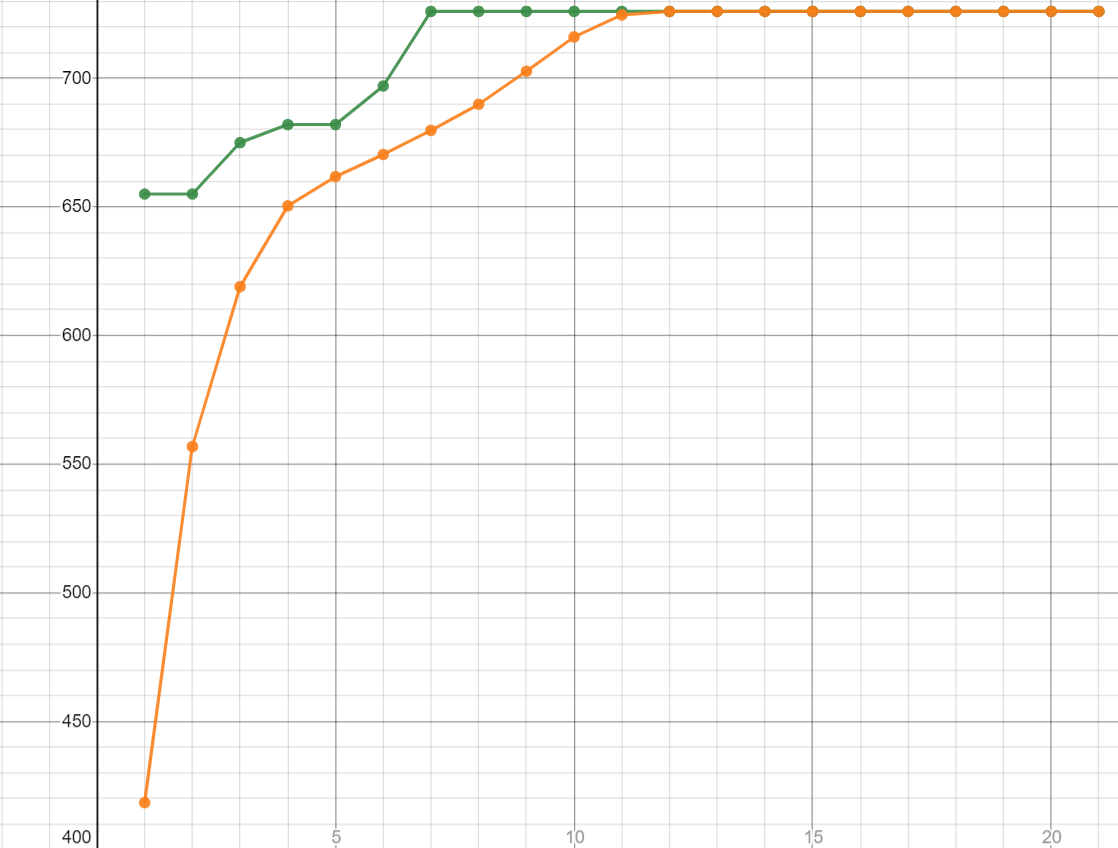
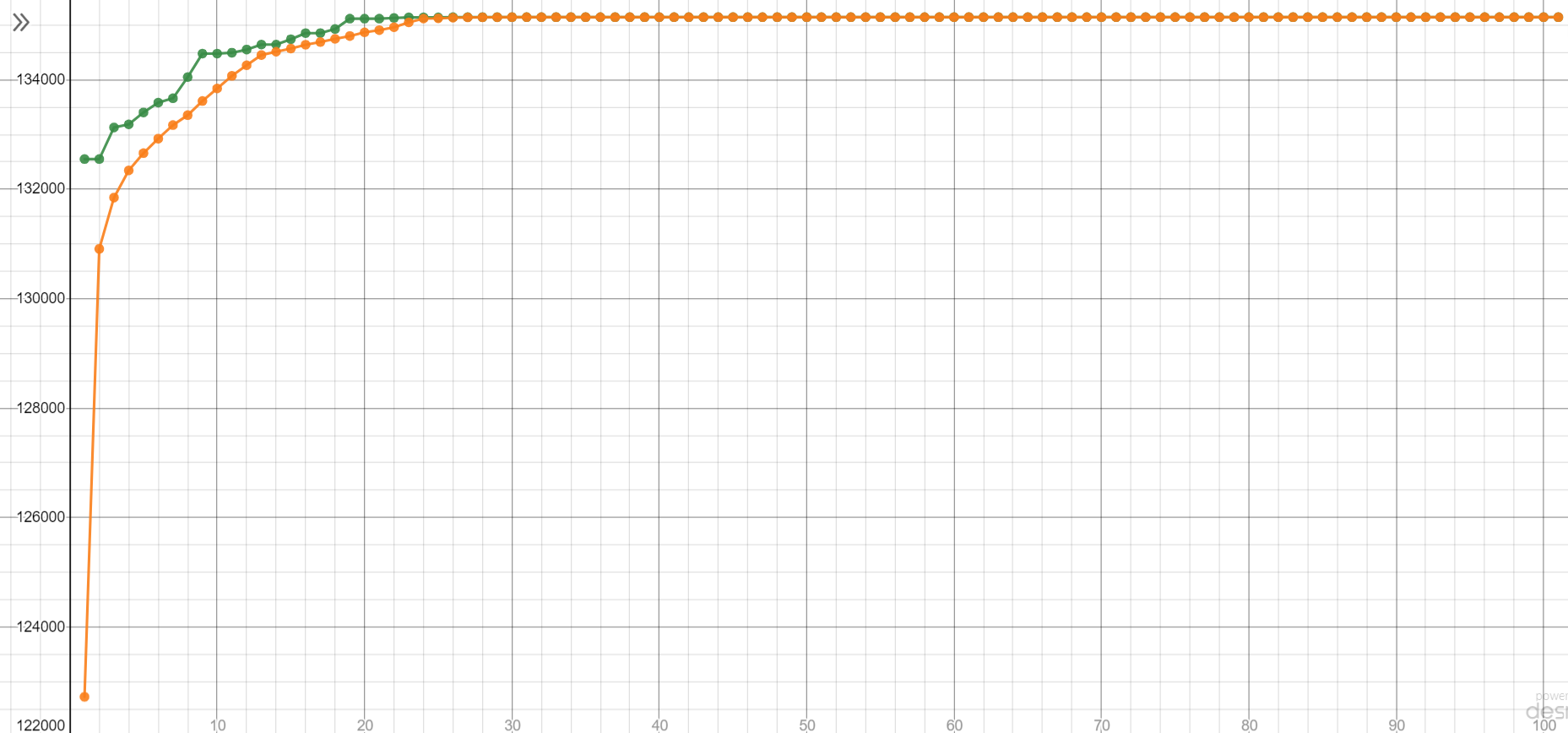
Documentatie Lab 3

require 'csv'  
  
*# file\_path - string, file with input data  
# @return - array*def read\_config\_file(*file\_path*)  
 *lines* = []  
 File.open(*file\_path*, 'r') do |*file*|  
 *lines* = *file*.readlines  
 end  
 *objects* = []  
 *n* = *lines*[0].strip.to\_i  
 (1..*n*).each do |*i*|  
 *line* = *lines*[*i*].strip  
 *parts* = *line*.split(' ')  
 *objects*[*i*-1] = {}  
 *objects*[*i*-1]['value'] = *parts*[1].to\_i  
 *objects*[*i*-1]['weight'] = *parts*[2].to\_i  
 end  
 *max\_sum* = *lines*[*n*+1].strip.to\_i  
 [*n*, *max\_sum*, *objects*]  
end  
  
*# n - integer, total number of objects  
# rand\_arr - array, a possible solution  
# objects - array of hashes, contains value and weight for each object  
# max\_sum - integer, maximum weight that fits in the backpack  
# @return - boolean*def is\_solution(*n*, *rand\_arr*, *objects*, *max\_sum*)  
 *sum* = 0  
 (0..*n*-1).each do |*i*|  
 *sum* += *objects*[*i*]['weight']\**rand\_arr*[*i*]  
 end  
 return true if *sum* <= *max\_sum* false  
end  
  
*# n - integer, total number of objects  
# @return - array*def generate\_random\_array(*n*)  
 *rand\_solution* = []  
 (0..*n*-1).each do |*i*|  
 *rand\_solution*[*i*] = rand(0..1)  
 end  
 *rand\_solution*.dup  
end  
  
*# n - integer, total number of objects  
# objects - array of hashes, contains value and weight for each object  
# max\_sum - integer, maximum weight that fits in the backpack  
# @return - array*def generate\_random\_solution(*n*, *objects*, *max\_sum*)  
 *rand\_arr* = generate\_random\_array(*n*)  
 while is\_solution(*n*, *rand\_arr*, *objects*, *max\_sum*) == false  
 *rand\_arr* = generate\_random\_array(*n*)  
 end  
 *rand\_arr*.dup  
end  
  
*# n - integer, total number of objects  
# objects - array of hashes, contains value and weight for each object  
# solution - array, a solution  
# @return - integer*def eval(*n*, *solution*, *objects*)  
 *sum* = 0  
 (0..*n*-1).each do |*i*|  
 *sum* += *objects*[*i*]['value']\**solution*[*i*]  
 end  
 *sum*end  
  
def greedy(*n*, *objects*, *max\_sum*)  
 *solution* = []  
 (0..*n*-1).each do |*i*|  
 *solution*[*i*] = 1  
 end  
 *obj* = []  
 *objects*.each{|*e*| *obj* << *e*.dup}  
 *obj\_cpy* = []  
 *obj*.each { |*e*| *obj\_cpy* << *e*.dup}  
 *obj\_cpy*.sort\_by! { |*e*| *e*['value']}  
 *i* = 0  
 while is\_solution(*n*, *solution*, *objects*, *max\_sum*) == false  
 *a* = *obj*.index { |*e*| *e*['value'] == *obj\_cpy*[*i*]['value'] }  
 *solution*[*a*] = 0  
 *i* += 1  
 end  
 [eval(*n*, *solution*, *objects*), *solution*.dup]  
end  
  
def generate\_population(*num*, *n*, *objects*, *max\_sum*)  
 *population* = []  
 *i* = 0  
 while *i* < *num  
 population* << generate\_random\_solution(*n*, *objects*, *max\_sum*)  
 *i* += 1  
 end  
 *pop\_cpy* = []  
 *population*.each { |*e*| *pop\_cpy* << *e*.dup}  
 *pop\_cpy*end  
  
def generate\_parents(*num*, *k*, *population*, *objects*, *n*)  
 *parents* = []  
 *i* = 0  
 *number\_of\_pop* = *population*.length  
 while *i* < *num  
 rand\_pos\_i* = rand(*number\_of\_pop* - 1)  
 *chosen\_parent* = *population*[*rand\_pos\_i*]  
 *i\_fit* = eval(*n*, *population*[*rand\_pos\_i*], *objects*)  
 *j* = 0  
 while *j* < *k  
 rand\_pos* = rand(*number\_of\_pop* - 1)  
 *new\_fit* = eval(*n*, *population*[*rand\_pos*], *objects*)  
 if *new\_fit* > *i\_fit  
 chosen\_parent* = *population*[*rand\_pos*].dup  
 *i\_fit* = *new\_fit* end  
 *j* += 1  
 end  
 *parents* << *chosen\_parent  
 i* += 1  
 end  
 *par\_cpy* = []  
 *parents*.each { |*e*| *par\_cpy* << *e*.dup}  
 *par\_cpy*end  
  
def cross\_parents(*n*, *objects*, *max\_sum*, *parents*)  
 *children* = []  
 *i* = 0  
 while *i* < *parents*.length  
 *p1* = *parents*[*i*]  
 *p2* = *parents*[*i* + 1]  
 *i* += 2  
 *c* = get\_children(*p1*, *p2*, *n*, *objects*, *max\_sum*)  
 *children* << *c*[0]  
 *children* << *c*[1]  
 end  
 *c\_cpy* = []  
 *children*.each { |*e*| *c\_cpy* << *e*.dup}  
 *c\_cpy*end  
  
def fix\_arr(*n*, *rand\_arr*, *objects*, *max\_sum*)  
 *i* = -1  
 until is\_solution(*n*, *rand\_arr*, *objects*, *max\_sum*)  
 while *i* < *rand\_arr*.length  
 *i* += 1  
 if *rand\_arr*[*i*] == 1  
 *rand\_arr*[*i*] = 0  
 break  
 end  
 end  
 end  
 *rand\_arr*.dup  
end  
  
def get\_children(*p1*, *p2*, *n*, *objects*, *max\_sum*)  
 *sol\_length* = *p1*.length  
 *cut\_position* = rand(1..*sol\_length* - 2)  
 *i* = 0  
 *c1* = []  
 *c2* = []  
 while *i* < *sol\_length* if *i* <= *cut\_position  
 c1*[*i*] = *p1*[*i*]  
 *c2*[*i*] = *p2*[*i*]  
 else  
 *c1*[*i*] = *p2*[*i*]  
 *c2*[*i*] = *p1*[*i*]  
 end  
 *i* += 1  
 end  
 *c1* = fix\_arr(*n*, *c1*, *objects*, *max\_sum*)  
 *c2* = fix\_arr(*n*, *c2*, *objects*, *max\_sum*)  
 [*c1*.dup, *c2*.dup]  
end  
  
def mutate\_children(*n*, *objects*, *max\_sum*, *children*)  
 *mutations* = []  
 *children*.each do |*c*|  
 *rand\_flip* = rand(*n*-1)  
 *c*[*rand\_flip*] = 1 - *c*[*rand\_flip*]  
 *c* = fix\_arr(*n*, *c*, *objects*, *max\_sum*)  
 *mutations* << *c*.dup  
 end  
 *m\_cpy* = []  
 *mutations*.each { |*e*| *m\_cpy* << *e*.dup}  
 *m\_cpy*end  
  
*# miu + lambda*def select\_survivors(*num*, *parents*, *children*, *mutated\_children*, *n*, *objects*)  
 *potential\_survivors* = []  
 *survivors* = []  
 *parents*.each do |*p*|  
 *potential\_survivors* << *p*.dup  
 end  
 *children*.each do |*c*|  
 *potential\_survivors* << *c*.dup  
 end  
 *mutated\_children*.each do |*m*|  
 *potential\_survivors* << *m*.dup  
 end  
 *potential\_survivors*.sort\_by!{ |*e*| -eval(*n*, *e*, *objects*) }  
 (0..*num*-1).each do |*i*|  
 *survivors* << *potential\_survivors*[*i*].dup  
 end  
 *survivors*end  
  
def best\_and\_average(*solutions*, *n*, *objects*)  
 *best\_fit* = eval(*n*, *solutions*[0], *objects*)  
 *average\_fit* = 0  
 *solutions*.each do |*s*|  
 *average\_fit* += eval(*n*, *s*, *objects*)  
 end  
 *average\_fit* = *average\_fit*\*1.0/*solutions*.length  
 [*best\_fit*, *average\_fit*]  
end  
  
def best\_and\_average\_unordered(*solutions*, *n*, *objects*)  
 *best\_fit* = 0  
 *average\_fit* = 0  
 *solutions*.each do |*s*|  
 *fit* = eval(*n*, *s*, *objects*)  
 *best\_fit* = *fit* if *best\_fit* < *fit  
 average\_fit* += *fit* end  
 *average\_fit* = *average\_fit*\*1.0/*solutions*.length  
 [*best\_fit*, *average\_fit*]  
end  
  
def alg(*n\_param*, *m\_param*, *objects*, *n*, *max\_sum*)  
 *best* = []  
 *values* = []  
 *t* = 0  
 *population* = generate\_population(*n\_param*, *n*, *objects*, *max\_sum*)  
 *data* = best\_and\_average\_unordered(*population*, *n*, *objects*)  
 *best* << *data*[0]  
 *values* << *data* while *t* < *m\_param  
 parents* = generate\_parents(*n\_param*, 3, *population*, *objects*, *n*)  
 *children* = cross\_parents(*n*, *objects*, *max\_sum*, *parents*)  
 *mutated\_children* = mutate\_children(*n*, *objects*, *max\_sum*, *children*)  
 *population* = select\_survivors(*n\_param*, *parents*, *children*, *mutated\_children*, *n*, *objects*)  
 *t* += 1  
 *data* = best\_and\_average(*population*, *n*, *objects*)  
 *best* << *data*[0]  
 *values* << *data* end  
 [*best*.max, *values*]  
end  
  
def write\_data(*n\_param*, *m\_param*, *n*, *objects*, *max\_sum*, *repeat*: 10)  
 *i* = 1  
 *all\_fit* = []  
 *all* = []  
 *t0* = Time.now  
 while *i* <= *repeat  
 values* = alg(*n\_param*, *m\_param*, *objects*, *n*, *max\_sum*)  
 *best\_fit* = *values*[0]  
 *all\_fit*.push(*best\_fit*)  
 *j* = 1  
 *plot\_best* = []  
 *plot\_avg* = []  
 *values*[1].each do |*v*|  
 *plot\_best* << "(#{*j*}, #{*v*[0]})"  
 *plot\_avg* << "(#{*j*}, #{*v*[1]})"  
 *j* += 1  
 end  
 *all*.push([*plot\_best*.join(", ")])  
 *all*.push([*plot\_avg*.join(", ")])  
 *i* += 1  
 end  
 *t1* = Time.now - *t0  
 worst* = *all\_fit*.min  
 *best* = *all\_fit*.max  
 *avg* = *all\_fit*.sum(0.0)/*all\_fit*.size  
 *all*.push(['Worst', *worst*])  
 *all*.push(['Best', *best*])  
 *all*.push(['Average', *avg*])  
 *all*.push(['Runtime', *t1*])  
 File.write("kp\_#{*n*}\_population\_#{*n\_param*}\_generations\_#{*m\_param*}.csv", *all*.map(&:to\_csv).join)  
end  
  
def main(*n\_param*, *m\_param*)  
 *values* = read\_config\_file('input\_file\_200.txt')  
 *n* = *values*[0]  
 *max\_sum* = *values*[1]  
 *objects* = *values*[2]  
 write\_data(*n\_param*, *m\_param*, *n*, *objects*, *max\_sum*)  
end  
  
main(200, 400)









N = 500 si M = 200

|  |  |
| --- | --- |
| Worst | 134321 |
| Best | 135145 |
| Average | 134773.5 |
| Runtime | 81.35608 |