# Evolutionary Techniques for Constrained Optimization Problems

Fernando Jiménez[1], José L. Verdegay[2]
[1]Dept. Informática, Inteligencia Artificial y Electrónica
Universidad de Murcia
Campus de Espinardo, 30071-Espinardo, Murcia, Spain
Phone: +34-68-364630, Fax: +34-68-364151
e-mail: fernan@dif.um.es
[2]Dept. Ciencias de la Computación e Inteligencia Artificial
Universidad de Granada
Avda. Andalucía, 38, 18071-Granada, Spain
Phone: +34-58-244019, Fax: +34-58-243317
e-mail: verdegay@goliat.ugr.es

ABSTRACT: An Evolutionary Algorithm to solve general constrained optimization problems is proposed in this paper. Mathematical programming problems such as linear, nonlinear, integer, boolean and mixed programming problems can be solved by using this technique. Some important characteristics of the Evolutionary Algorithm are a natural representation of solutions, a problem-independent technique for constraint satisfaction, tournament selection, complete generational replacement, and elitism strategy. Simulation results show a good performance by this method for several test problems with different structure.

KEYWORDS: Evolutionary algorithms; mathematical programming; constraint satisfaction.

## 1 INTRODUCTION

*Constrained optimization* deals with the determination of solutions to problems that maximize or minimize a certain function and which are feasible within the limits of given constraints. *Mathematical Programming* is a broad discipline that has been concerned for many decades with constrained optimization. On the one hand, this area has grown over the years and yielded very powerful theories and algorithms to find solutions to models of different mathematical structures, such as *linear programming*, *nonlinear programming*, *integer programming*, *fuzzy programming*, etc. On the other hand, Mathematical Programming has been applied to a great variety of theoretical and applied problems in a number of different areas such as Operations Research, Engineering, Artificial Intelligence, etc.

Here we consider an important subset of mathematical programming problems which include linear, nonlinear, integer, boolean and mixed programming problems. These problems can be represented by the following general mathematical formulation:

$$\begin{aligned} &Minimize && f(x,y,z) \\ &subject\ to && \\ & && g_j(x,y,z) \leq 0, \quad j = 1,\ldots,m \end{aligned}$$

where $x = (x_1,\ldots,x_p) \in R^p$ is a $p$ dimensional real-valued parameter vector, with $l_i^x \leq x_i \leq u_i^x$ ($l_i^x, u_i^x \in R$, $i = 1,\ldots,p$, $p \geq 0$), $R$ stands for the real line, and no real-valued parameter is assumed when $p = 0$; $y = (y_1,\ldots,y_q) \in Z^q$ is a $q$ dimensional integer-valued parameter vector, with $l_i^y \leq y_i \leq u_i^y$ ($l_i^y, u_i^y \in Z$, $i = 1,\ldots,q$, $q \geq 0$), $Z$ represents the integer number set, and no integer-valued parameter is assumed when $q = 0$; $z = (z_1,\ldots,z_r) \in \{0,1\}^r$ is an $r$ dimensional boolean-valued parameter vector ($r \geq 0$), and no boolean-valued parameter is assumed when $r = 0$; $f(x,y,z)$, $g_j(x,y,z)$ ($j = 1,\ldots,m$, $m \geq 0$) are linear or nonlinear arbitrary functions, and no constraint is assumed when $m = 0$.

Note that we have not restricted ourselves to only considering minimization problems subject to less-than-or-equal-to-zero constraints, since maximization and minimization problems can be equivalently solved, a less-than-or-equal-to constraint $g_j(x,y,z) \leq b_j$ can be rewritten as $g_j(x,y,z) - b_j \leq 0$, a greater-than-or-equal-to

constraint $g_j(x,y,z) \geq b_j$ can be rewritten as $-g_j(x,y,z) + b_j \leq 0$; and an equality constraint $g_j(x,y,z) = b_j$ can be represented by two inequality constraints $g_j(x,y,z) - b_j \leq 0$ and $-g_j(x,y,z) + b_j \leq 0$.

If all the functions $f$ and $g_j$ $(j = 1,\ldots,m)$ are linear, then the problem simplifies to a linear program, which is the classical problem of mathematical programming and extremely efficient algorithms exist to obtain the optimal solution (simplex method and derived, for real-valued parameter optimization, and branch-and-bound technique, cutting methods, dynamic programming, etc., for integer, boolean and mixed programming). If one of the functions $f$ or $g_j$ $(j = 1,\ldots,m)$ is nonlinear then we have a nonlinear programming problem. A nonlinear programming problem in which the objective function $f$ and functions $g_j$ $(j = 1,\ldots,m)$ are arbitrary ones is, in general, intractable. It is unrealistic to expect to find a deterministic method for the general nonlinear programming problem in the global optimization category which would be better than an exhaustive search. Many optimization techniques based on gradient methods aim at local optimization only.

As it is very well known, *Evolutionary Algorithms* (EA) (Biethahn, 1995; Goldberg, 1989; Michalewicz, 1992) are global optimization methods that aim at complex objective functions and constraints. Most research into applications of EA to nonlinear programming problems has been concerned with complex objective functions but not with constraints, and only recently several approaches (Michalewicz and Schoenauer, 1996) have extended evolutionary techniques by some constraint-handling methods. For particular constrained optimization problems, specialized EA have been developed by incorporating problem-specific knowledge into the EA, e.g. the *Transportation Problem* (Jiménez and Cadenas, 1995; Jiménez and Verdegay, 1998b; Michalewicz, 1992), or the *Traveling Salesman Problem* (Grefenstette et al., 1985). However, handling constraints by EA is not an easy task for general constrained optimization problems. The most usual technique in EA-based constrained optimization is the *penalty method* (Homaifar et al., 1994), in which a constrained problem is transformed into a unconstrained one by associating a cost or penalty with all the constraint violations. The success of this approach depends on the way in which the penalties are dealt with. The penalties, in order to prevent the premature convergence and divergence from the optimum result, must not be chosen either too large or too modest. Thus, the existence of nontrivial constraints produce a strong dependency between the problem and the EA. Other approaches such as decoders or repair algorithms (Michalewicz, 1992) also suffer from the disadvantage of being tailored to the specific problem and are not sufficiently general to handle a variety of problems. An overview of EA for constrained parameter optimization problems can be found in (Michalewicz and Schoenauer, 1996).

With this background, we are interested in problem-independent evolutionary techniques to solve general constrained optimization problems such as linear programming problems, nonlinear programming ones, integer programming ones, boolean programming ones, and mixed programming ones. Obviously, the main interest is in solving problems with which existing methods have difficulties (complex nonlinear programming problems). Consequently, the paper is organized as follows: Section 2 describes the structure and components of an EA to solve the considered mathematical programming problems. Simulation results for some interesting test problems are shown in section 3. Finally, section 4 indicates the main conclusions and some current working ways.

## 2 AN EVOLUTIONARY ALGORITHM FOR CONSTRAINED OPTIMIZATION

Evolutionary Algorithms are adaptive procedures of optimization and search that find solutions to problems inspired by the mechanisms of natural evolution. They imitate, on an abstract level, biological principles such as a population based approach, the inheritance of information, the variation of information via crossover/mutation, and the selection of individuals based on fitness. EA start with an initial set (population) of alternative solutions (individuals) for the given problem which are evaluated in terms of solution quality (fitness). Then, the operators of selection, replication and variation are applied to obtain new individuals (offspring) that constitute a new population. The interplay of selection, replication and variation of the fittest, leads to solutions of increasing quality over the course of many iterations (generations). When finally a termination criterion is met, such as a maximum number of generations, the search process is terminated and the final solution is shown as output. Moreover, the algorithm uses a parameter set, such as population size, number of generations, crossing and mutation probabilities, to guide the evolutionary process. The structure of an EA can be as shown in figure 1.

Obviously, an appropriate *representation* of solutions to the problem is necessary. The most well-known class of EA is the *Genetic Algorithm* (GA) (Holland, 1975), which has received a lot of attention in the last few years. The classical GA use fixed-length binary strings to represent individuals and two basic genetic operators (binary mutation and binary crossover). Other variants of EA, such as *Genetic Programming* (Koza, 1992), *Evolution Strategies* (Rechenberg, 1973), or *Evolutionary Programming* (Fogel et al., 1966) are less popular, though very powerful too, and differences arise mainly in solution representation and importance of crossover/mutation. Nevertheless, the field of EA is characterised by high dynamics, and modifications and extensions of the technology

```
procedure EA
begin
    initialize_population
    evaluate_population
    while (not termination-condition) do
    begin
        generate_new_population {selection, replication, variation
                                 and generational replacement}
        evaluate_population
    end
end
```

Figure 1: Structure of an Evolutionary Algorithm.

are continuously being developed. Our EA for mathematical programming problems is basically a GA because of overall sequence of operations, but a "natural" representation of solutions is used (floating-point numbers to represent real-valued parameters, integer numbers to represent integer-valued parameters, and boolean values to represent boolean-valued parameters). Thus, an individual $V$ of the population is represented as follows:

$$V = (x, y, z)$$

where $x = (x_1, \ldots, x_p)$, with $x_i \in R$ $(i = 1, \ldots, p)$, $y = (y_1, \ldots, y_q)$, with $y_i \in Z$ $(i = 1, \ldots, q)$, and $z = (z_1, \ldots, z_r)$, with $z_i \in \{0, 1\}$ $(i = 1, \ldots, r)$.

To handle constraints we consider the following general assumptions: (1) populations are composed of feasible and unfeasible individuals; (2) feasible individuals evolve towards optimality guided by an *optimality evaluation function*; (3) unfeasible individuals evolve towards feasibility guided by an *unfeasibility evaluation function*; (4) feasible individuals have greater probability of selection than unfeasible individuals.

According to these criteria, initialize_population, evaluate_population and generate_ new_population procedures can be designed as figures 2, 3 and 4 respectively show. In these algorithms POP and NEW_POP represent the current population and the new population respectively, POP($s$).IND represents the individual which is placed at the position $s$ in the current population, NEW_POP($s$).IND represents the individual which is placed at the position $s$ in the new population, POP($s$).OPTIMALITY represents the evaluation of the feasible individual which is placed at the position $s$ in the current population, POP($s$).UNFEASIBILITY represents the evaluation of the unfeasible individual which is placed at the position $s$ in the current population, and *popsize* is the population size.

```
procedure initialize_population
begin
    s ← 0
    while s ≤ popsize do
    begin
        s ← s + 1
        x_i ← random real value ∈ [l_i^x, u_i^x], for i = 1, ..., p
        y_i ← random integer value ∈ [l_i^y, u_i^y], for i = 1, ..., q
        z_i ← random value ∈ {0, 1}, for i = 1, ..., r
        V ← (x, y, z)
        POP(s).IND ← V
    end
end
```

Figure 2: A procedure to generate an initial population.

Both optimality and unfeasibility evaluation functions have to be minimized. An individual $V = (x, y, z)$ is feasible if $g_j(x, y, z) \leq 0$ for all $j = 1, \ldots, m$. However, we allow a violation $\Delta \geq 0$ and then an individual $V = (x, y, z)$ is feasible if $g_j(x, y, z) \leq \Delta$ for all $j = 1, \ldots, m$. Note that there is a similarity between the unfeasibility evaluation function and the method of min-max formulation used in multiobjective optimization (Chankong and Haimes, 1983). This method attempts to minimize the relative deviations of the single objective functions from the individual optimum, and it can yield the best possible compromise solution when objectives with equal priority are required to be optimized. Since constraints and objectives can be treated in a similar way, and equal priority is assumed for all constraints, the min-max formulation is also appropriate for constraint satisfaction.

```
procedure evaluate_population
begin
   s ← 0
   while s ≤ popsize do
   begin
      s ← s + 1
      V ← POP(s).IND
      if feasible(V) then
         POP(s).OPTIMALITY ← f(x, y, z)
         {optimality evaluation function}
      else
         POP(s).UNFEASIBILITY ← max{g_j(x, y, z), j = 1 ..., m}
         {unfeasibility evaluation function}
   end
end
```

Figure 3: Evaluation of individuals in a population.

```
procedure generate_new_population
begin
   s ← 1
   I ← {1, ..., popsize}
   NEW_POP(s).IND ← best(I) {elitism strategy}
   while s < popsize do
   begin
      mate1 ← tournament_selection
      mate2 ← tournament_selection
      crossover(mate1,mate2,child1,child2)
      offspring1 ← mutation(child1)
      offspring2 ← mutation(child2)
      s ← s + 1
      NEW_POP(s).IND ← offspring1
      s ← s + 1
      if s ≤ popsize then
         NEW_POP(s).IND ← offspring2
   end
   POP ← NEW_POP
end
```

Figure 4: A procedure to obtain a new population.

The generate_new_population procedure can be designed in multiple ways. In our EA, *tournament selection*, *complete generational replacement* and *elitism strategy* are used. With the tournament selection (see figure 5), a group of *tourn* individuals is randomly sampled from the population and the best individual in the group is chosen for reproduction. Variation operators are applied to the selected individuals (with some probability) and the offspring are copied into the new population. This process is repeated until the whole new population is generated (complete generational replacement). Moreover, with the elitism strategy the best member of a population is always copied into the new population. Note that replication of individuals is achieved when no variation operator is applied.

To obtain the best individual of a collection (see figure 6), the following criteria are assumed: (1) a feasible individual is better than another feasible individual if the evaluation of the first is smaller than the evaluation of the second; (2) a feasible individual is better than a unfeasible individual; (3) an unfeasible individual is better than another unfeasible individual if the evaluation of the first is smaller than the evaluation of the second.

Many variation operators have been proposed during the last years (a discussion of these variation operators is shown in (Michalewicz and Schoenauer, 1996)). In the context in which we are here concerned, and after a

```
function tournament_selection
begin
   Set randomly J = {j_1, ..., j_tourn} ⊆ {1, ..., popsize}
   return(best(J))
end
```

Figure 5: Tournament selection.

```
function best
input: index set K = {k_1, ..., k_l} ⊆ {1, ..., popsize}
begin
   s ← 1
   V ← POP(k_s).IND
   while s ≤ l do
   begin
      s ← s + 1
      if (feasible(POP(k_s).IND) and feasible(V) and
      POP(k_s).OPTIMALITY<POP(V).OPTIMALITY) or
      (feasible(POP(k_s).IND) and unfeasible(V)) or
      (unfeasible(POP(k_s).IND) and unfeasible(V) and
      (POP(k_s).UNFEASIBILITY<POP(V).UNFEASIBILITY) then
            V ← POP(k_s).IND
   end
   return(V)
end
```

Figure 6: A procedure to obtain the best individual of a collection.

long experimentation process, we finally decided to use *uniform crossover*, *nonuniform mutation* and *uniform mutation*.

Uniform crossover works as follows. With probability $p_c$, two parents $V_1 = (x_1, y_1, z_1)$, with $x_1 = (x_1^1, \ldots, x_p^1)$, $y_1 = (y_1^1, \ldots, y_q^1)$, $z_1 = (z_1^1, \ldots, z_r^1)$ and $V_2 = (x_2, y_2, z_2)$, with $x_2 = (x_1^2, \ldots, x_p^2)$, $y_2 = (y_1^2, \ldots, y_q^2)$, $z_2 = (z_1^2, \ldots, z_r^2)$, produce two children $V_3 = (x_3, y_3, z_3)$, with $x_3 = (x_1^3, \ldots, x_p^3)$, $y_3 = (y_1^3, \ldots, y_q^3)$, $z_3 = (z_1^3, \ldots, z_r^3)$ and $V_4 = (x_4, y_4, z_4)$, with $x_4 = (x_1^4, \ldots, x_p^4)$, $y_4 = (y_1^4, \ldots, y_q^4)$, $z_4 = (z_1^4, \ldots, z_r^4)$, where $x_i^3 = x_i^1$ or $x_i^3 = x_i^2$, with equal probability for all $i = 1, \ldots, p$, $y_i^3 = y_i^1$ or $y_i^3 = y_i^2$, with equal probability for all $i = 1, \ldots, q$, and $z_i^3 = z_i^1$ or $z_i^3 = z_i^2$, with equal probability for all $i = 1, \ldots, r$. Child $V_4$ is created by reversing decisions for all components.

The action of the nonuniform mutation, which is only applied to the components with floating-point representation, depends on the age of the population, and its effect is a fine local tuning in the last generations of the EA. Given an individual $V = (x, y, z)$ with $x = (x_1, \ldots, x_p)$, nonuniform mutation generates an offspring $V' = (x', y, z)$ with $x' = (x'_1, \ldots, x'_p)$ where $x'_i$ $(i = 1, \ldots, p)$ has been mutated with probability $p_m$ as follows:

$$x'_i = \begin{cases} x_i + (u_i^x - x_i) \cdot r \cdot (1 - \frac{t}{T})^c, \text{ if a random digit is 0} \\ x_i - (x_i - l_i^x) \cdot r \cdot (1 - \frac{t}{T})^c, \text{ if a random digit is 1} \end{cases}$$

As usual $[l_i^x, u_i^x]$ is the domain of the variable $x_i$, and $r$ is a random number from $[0, 1]$, $T$ is the maximal generation number, $t$ is the present generation, and $c$ is a system parameter determining the degree of nonuniformity.

Uniform mutation is applied to the components with integer and boolean representation. Given an individual $V = (x, y, z)$ with $y = (y_1, \ldots, y_q)$, $z = (z_1, \ldots, z_r)$, uniform mutation generates an offspring $V' = (x, y', z')$ with $y' = (y'_1, \ldots, y'_q)$, $z' = (z'_1, \ldots, z'_r)$, where $y'_i$ $(i = 1, \ldots, q)$ and $z'_j$ $(j = 1, \ldots, r)$ have been mutated with probability $p_m$ as follows:

$$y'_i = \text{random integer value} \in [l_i^y, u_i^y]$$
$$z'_j = \text{random value} \in \{0, 1\}$$

## 3 SIMULATION RESULTS

Although the EA have been tested by a large number of test problems which include integer, boolean and mixed mathematical programming problems, for the sake of comparison with other evolutionary computation techniques we show in this section simulation results for the following mathematical programming problems reported by Michalewicz and Schoenauer (1996): (1) test case $G1$, with 13 real-valued parameters, quadratic objective function, and 9 linear inequalities; (2) test case $G7$, with 10 real-valued parameters, quadratic objective function, 3 linear inequalities, and 5 nonlinear inequalities; (3) test case $G9$, with 7 real-valued parameters, polinomial objective function, and 4 nonlinear inequalities; (4) test case $G10$, with 8 real-valued parameters, linear objective function, 3 linear inequalities, and 3 nonlinear inequalities. The values of the EA parameters used in the experiments are shown in Table 1, and Table 2 shows simulation results for the considered test problems. The global optimum, the best fitness obtained with the proposed EA, the best fitness of solutions without

| Parameter | Value |
|---|---|
| Population size ($popsize$) | 20 |
| Maximal generation number ($T$) | problem-dependent (1000 – 50000) |
| Crossing probability ($p_c$) | 0.4 |
| Mutation probability ($p_m$) | 0.4 |
| Degree of non-uniformity ($c$) | 2 |
| Number of individuals in the comparison set for tournament selection ($tourn$) | 4 |

Table 1: EA parameters used in the experiments.

| Test case | Global optimum | Best fitness | Best fitness with other evolutionary techniques | Maximal violation allowed | Maximal violation obtained |
|---|---|---|---|---|---|
| G1 | $-15$ | $-15.000000032$ | $-15.000$ | $10^{-7}$ | $5.7618918525 \cdot 10^{-8}$ |
| G7 | 24.3062091 | 25.012657345 | 25.486 | 0.0001 | $9.9527851766 \cdot 10^{-5}$ |
| G9 | 680.6300573 | 680.63319807 | 680.640 | 0.001 | $9.9999148776 \cdot 10^{-4}$ |
| G10 | 7049.330923 | 7088.9652637 | 7286.650 | 0.001 | $9.9998753935 \cdot 10^{-4}$ |

Table 2: Simulation results.

violated constraints by more than 0.001 obtained with other constraint-handling evolutionary techniques[1] reported in literature, the maximal violation of the constraints allowed for solutions, and the maximal violation of the constraints in the solution, are shown in Table 2 for each test problem. Results show a good performance by this evolutionary technique obtaining best results than those found with other evolutionary techniques for all test cases.

## 4 CONCLUSIONS

The main focus of this work has been to design a problem-independent evolutionary constraint-handling technique to find acceptable solutions to general constrained optimization problems. An important set of mathematical programming problems such as linear, nonlinear, integer, boolean and mixed programming problems can be solved by using this evolutionary computation technique. Simulation results for four test cases showed a good performance by the proposed EA. As main current working ways the two following ones are to be pointed out: 1) a modality of the proposed constraint-handling technique in which the nondomination concept of multiobjective optimization is applied to the constraints to "push" the unfeasible individuals toward the feasible region, and 2) integration of the proposed technique with other multiobjective optimization techniques (Fonseca and Fleming, 1993; Fonseca and Fleming, 1995; Horn and Nafpliotis, 1993; Jiménez and Verdegay, 1998a; Srinivas and Deb, 1995) to find multiple nondominated solutions to constrained multiobjective optimization problems. In this extended EA, a niche formation technique to maintain an appropiate diversity is incorporated.

## ACKNOWLEDGEMENT

The authors thank the Comisión Interministerial de Ciencia y Tecnología (CICyT) for the support given to this work under the projects TIC97-1343-C02-02, 1FD97-0255-C03-01 (02) and ESP97-1518-E.

## REFERENCES

Biethahn, J., Nissen, V. (1995), *Evolutionary Algorithms in Management Applications.* Springer-Verlag Berlin Heidelberg.

Chankong, V., Haimes, Y.Y. (1983). *Multiobjective Decision Making: Theory and Methodology.* North-Holland series in Systems Science and Engineering, Andrew P. Sage (Ed.).

Fogel, L.J., Owens, A.J., Walsh, M.J. (1966). *Artificial Intelligence through Simulated Evolution.* Wiley, New York.

Fonseca, C.M, Fleming, P.J. (1993). Genetic algorithms for multi-objective optimization: formulation, discussion and generalization. In S. Forrest (Ed.), *Proc. of the Fifth Intern. Conf. on Genetic Algorithms* (pp. 416-423). Morgan Kaufmann, San Mateo.

[1] Static penalties, dynamic penalties, annealing penalties, death penalty, behavioral memory, superiority of feasible points, and repair of unfeasible individuals (Michalewicz and Shoenauer, 1996).

Fonseca, C.M., Fleming, P.J. (1995). An overview of avolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, vol. 3, no. 1, pp. 1-16.
Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.
Grefenstette, J.J., Gopal, R., Rosmaita, B.J., Van Gucht, D. (1985). Genetic algorithms for the traveling salesman problem. *Proc. of an International Conference on Genetic Algorithms and Their Applications*, pp. 160-168.
Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor.
Homaifar, A., Qi, C.X., Lai, S.H. (1994). Constrained optimization via Genetic Algorithms. *Simulation*, vol. 62, no. 4, pp. 242-254.
Horn, J., Nafpliotis, N. (1993). Multiobjective optimization using the niched Pareto Genetic Algorithm. IlliEAL Report no. 93005.
Jiménez, F., Cadenas, J.M. (1995). An evolutionary program for the multiobjective solid transportation problem with fuzzy goals. *Operations Research and Decision*, vol. 2, pp. 5-20.
Jiménez, F., Verdegay, J.L. (1998a). Constrained multiobjective optimization by evolutionary algorithms. *Procs. of the International ICSC Symposium on Engineering of Intelligent Systems (EIS'98)*, pp. 266-271. University of La Laguna, Tenerife, Spain.
Jiménez, F., Verdegay, J.L. (1998b). An evolutionary algorithm for interval solid transportation problems. *Evolutionary Computation*, vol. 7, no. 1, pp. 103-107.
Koza, J.R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, Cambridge, Massachusetts.
Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag.
Michalewicz, Z., Schoenauer, M. (1996). Evolutionary Algorithms for constrained parameter optimization problems. *Evolutionary Computation*, vol. 4, no. 1, pp. 1-32.
Rechenberg, I. (1973). *Evolutionary Strategy: Optimization of Technical Systems According to the Priciples of Biological Evolution*. Frommann-Holzboog.
Srinivas, N., Deb, K. (1995). Multiobjective optimization using nondominated sorting in Genetic Algorithms. *Evolutionary Computation*, vol. 2, no. 3, pp. 221-248.