UNIVERSITY OF INFORMATICS OF MADRID - UPM
ARTIFICIAL INTELLIGENCE DEPARMENT

INTELLIGENT SERCHING BASED ON METHAHEURISTICS
MASTER IN ARTIFICIAL INTELLIGENCE

Solving the magic square problem with the NSGA II genetic algorithm

Alumni: Petra Franjic, Gustavo Puig
Professor: D. Alfonso Mateos Caballero
Date: 2th November 2015

Abstract:

The aim of this practical assignment was to implement and analyze a multi-objective genetic algorithm with the goal of obtaining the solution of the magic square problem, which is a combinatorial optimization problem. The algorithm that was chosen for this task is the Non-dominated Sorting Genetic Algorithm II (NSGA-II). Simulations with different parameter values were conducted and the performance of the algorithm was measured. The first part of the documentation explains different aspects of the workings of the algorithm and in the second part the results of the carried out simulations are presented and discussed.

Key words List:

Multi-objective optimization, NSGA-II, Evolutionary Computation, Genetic Algorithm, Combinatorial Optimization.

Contents

Introduction

The magic square is a square matrix of size nXn with integer elements between 1 and n^2 where the sums of the elements of all rows, columns and the main diagonal are equal. There are $(n2)!$ ways to fill the square. The magic square belongs to the set of combinatorial optimization problems that can be formulated as constraint satisfaction problems (CSPs) [1]. In this practical assignment we have studied whether the magic square problem of size 44 can be optimized effectively with the Non-dominated Sorting Genetic Algorithm II (NSGA-II) genetic algorithm.

Genetic algorithms (GAs) belong to a family of stochastic search methods covered by the generic term Evolutionary Computation (EC). The main characteristic of EC techniques is the computational simulation of the natural evolutionary process, as modeled by the Neo-Darwinian paradigm. These stochastic approaches were developed as an alternative to solving high-dimensional, discontinuous and multimodal problems, where traditional deterministic search techniques often proved ineffective [2].

NSGA (Non-Dominated Sorting in Genetic Algorithms) [3] is a popular non-domination based genetic algorithm for multi-objective optimization. It has been proved as very effective but still received criticism for its computational complexity, lack of elitism, and its diversity preservation techniques. A modified version of this algorithm based on its original design is the NSGA-II [4] algorithm. It incorporates elitism, has a better sorting algorithm and less bias in preserving diversity.

Chapter 1

Problem design

To use NSGA-II to solve the problem of the magic square, it was required to set up the magic square problem as a constraint satisfaction multi-objective problem. We imposed two constraints on the problem and 9 optimization goals which followed proposed solutions for similar problems, like Sudoku puzzle solving [7].

By the nature of the magic square each number between 1 and n^2 has to appear in the square exactly once. This rule was imposed with the two mentioned constraints. The first constraint maintained that the sum of all of the numbers in the square is equal to $0.5 * n^2 * (n^2 + 1)$, or, in the particular case of a $4x4$ magic square, to the sum of all numbers from 1 to 16. The second constraint maintained that the product of all numbers in the square equals to $(n^2)!$, or in the particular case of a $4x4$ magic square, to 16!. The constraints in algebraic form:

$$\frac{n^2(n^2+1)}{2} - \sum_{i}^{n}\sum_{j}^{n} x_{i,j} = 0 \tag{1.1}$$

$$(n^2)! - \prod_{i}^{n}\prod_{j}^{n} x_{i,j} = 0 \tag{1.2}$$

To guide the algorithm towards optimal solutions, it was necessary to use additional knowledge about the problem. The additional knowledge that we took into account was that considering a magic square with dimension n, the sum in each of the sub-blocks (rows, columns, diagonal) equals $0.5 * n * (n^2 + 1)$ [8]. This consideration was necessary because only those sub-blocks which sum to this value can contribute to an optimal solution. It is not enough, for an example, to consider as better those solutions for which can be said

only that they have more of these sub- blocks summing to the same value. The functions that were being minimized in algebraic form:

$$f_1(x) = |\frac{n(n^2+1)}{2} - \sum_{j}^{n} x_{1,j}| \tag{1.3}$$

...

$$f_5(x) = |\frac{n(n^2+1)}{2} - \sum_{i}^{n} x_{i,1}| \tag{1.4}$$

...

$$f_9(x) = |\frac{n(n^2+1)}{2} - \sum_{i}^{n} x_{i,j}| \tag{1.5}$$

Chapter 2

NSGA-II algorithm

2.1 Outline of genetic algorithms

2.2 Search components of the NSGA-II algorithm

In addition to the common concepts of single-objective metaheuristics, a multi-objective metaheuristics has to deal with three additional main search problems. Here these components are identified and in the following sections a detailed explanation will be given of the methods NSGA-II uses to address each of the problems. The first problem is the fitness assignment which has the role of guiding the search algorithm towards Pareto optimal solutions for a better convergence. This procedure assigns a scalar-valued fitness to a vector objective function [5]. NSGA-II employs a dominance based approach for this component of the search algorithm. The second search problem is the preservation of diversity where emphasis is on generating a diverse set of Pareto solutions in the objective space and helping the exploration of the fitness space [2]. The NSGA-II here employs a nearest-neighbor approach through calculating the crowding distance. Lastly, elitism guides the search towards preservation and use of elite solutions which allows a robust, fast, and monotonically improving performance of tje metaheuristic.

2.2.1 Dominance based ranking

The dominance-based approaches use the concept of dominance for fitness assignment, contrary to other approaches that use a scalarization function or treat the various objectives separately. The main advantage of dominance-based approaches is that they don't need the transformation of the MOP into a single-objective problem.

There are several fitness assignment procedures based on dominance that can guide the search towards the Pareto border. The NSGA-II algorithm uses dominance depth where the population of solutions is decomposed into several fronts [5]. The non-dominated solutions of the population receive rank 1 and form the first front E_1. The solutions that are not dominated except by solutions of E_1 receive rank 2 and they form the second front E_2. In a general way, a solution receives the row k if it is only dominated by individuals of the population belonging to the unit $E_1 \cup ...E_{k-1}$. Then, the depth of the solution corresponds to the depth of the front to which it belongs. Since a single value fitness (rank) is assigned to every solution in the population, any search component of a single- objective metaheuristic can be used to solve MOPs. The bene t of using a Pareto-based fitness assignment such as this, compared to scalar methods, is that it evaluates the quality of a solution in relation to the whole population. No absolute values are assigned to solutions. Dominance depth rank assignment is shown on Figure 1.

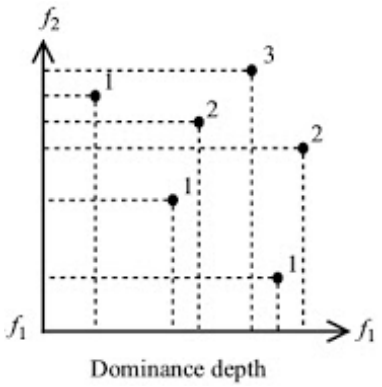

Figure 2.1: Dominance depth [5]

2.2.2 Crowding distance

There are several nearest-neighbor approaches that can be employed to evaluate the distribution of solutions and the crowding distance has proved to be one of the more eficient [2]. The crowding distance of a solution is de ned as the circumference of the rectangle de ned by its left and right neighbors, and infinity if there is no neighbor. Figure 2 illustrates the concept of crowding. Solutions with high crowding distance are considered better solutions, as they introduce more diversity in the population. In Figure 1 the solutions a and d belonging to the rank 1 in terms of non-dominance have the best score in terms of crowding. Then follows the solution b and then c. The rectangle associated with c is the smallest one [5]. NSGA-II uses crowding distance in its selection operator to keep a diverse front by making sure each member stays a crowding distance apart.

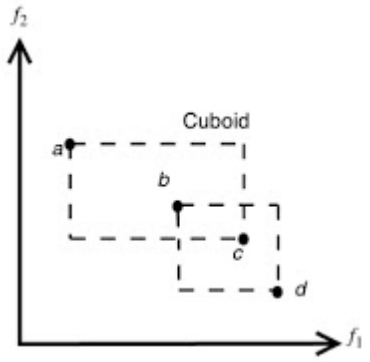

Figure 2.2: Crowding distance [5]

2.2.3 Elitism

Elitism consists of archiving the best solutions generated during the search. Elitism in NSGA-II is evident in the general outline of the algorithm. The key phase here is the replacement of individuals in the old generation. Before the replacement phase individuals in the old generation are sorted according to their ?goodness? together with the newly generated o spring. This allows the best individuals of the older generation to be passed in the next generation. The mechanism can be seen in Figure 3. P(sub)t is the parents population and Q(sub)t is the o spring population at generation t. F1 are the best solutions from the combined populations (parents and o spring). F2 are the second best solutions and so on.

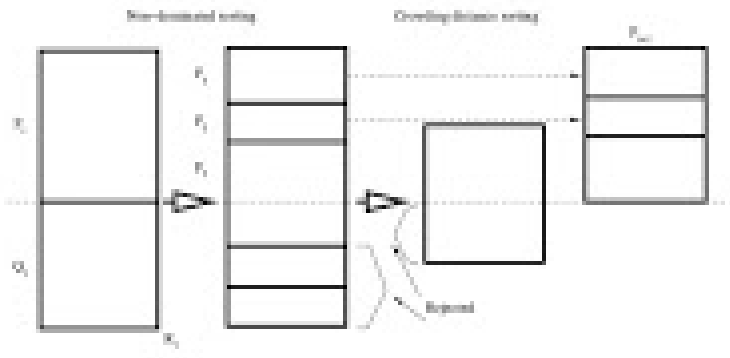

Figure 2.3: Flow diagram that shows the way in which the NSGA-II elitism works.

2.3 Outline of the NSGA-II algorithm

First the population is initialized as usual. Once the population is initialized it is sorted into fronts based on non-dominance. Each individual is assigned a fitness value corresponding to the front to which they belong to. In addition to fitness the crowding distance is calculated for each individual. If the non-dominance ranking of two individuals is equal, the crowding criteria is used to select the individual that gives the best crowding distance. Binary tournament selection is used to select parents from the population based on these two criteria. The selected population generates o spring from crossover and mutation operators. The population consisting of the current population and the current o spring is sorted again based on non-domination and crowding distance and the best N individuals are taken to form the new population, where N is the size of the population.

2.4 Codification

Two types of codification can be used to represent points in the solution space, binary and non-binary codification. Generally it can?t be said that one codification is better than the other, as it depends on the problem that is being dealt with.

Considering the design of the assigned problem, we have decided to use non- binary integer codification as the more simple and straightforward of the two possible codification. An additional reason against binary coded solutions was the fact that the magic square is a combinatorial optimization problem, which imposes a set of constraints on the possible solutions. By applying common crossover and mutation operators on bit strings it would be very difficult to respect the given constraint that the numbers should stay in the 1 to n^2 range, or that each of the numbers in the square should appear only once.

It is interesting to note the duality in the viewpoint of variables and values for this particular problem as was expressed in [6]. The array of numbers for codifying the square could be considered as either codifying the problem of finding the number to go in each cell of the square, or deciding which cell to put each number in. We have decided to go with the first approach since the constraints on the row and column sums are much easier to express.

2.5 Operators

The design of the algorithm requires a specification of the mutation and crossover operators. The selection operator has already been described as the binary tournament operator where two individuals are picked randomly from the population and the better one is given the chance to reproduce.

2.5.1 The crossover operator

The crossover operator that was used can be described as a larger scale uniform crossover. Two versions of the crossover were used, one that worked by recombining the rows of the parents, and the other that recombined the columns. Each was applied with equal probability. This alternation was necessary because passing on rows or columns from the parents to the children could be equally beneficial.

The workings of the crossover can be described as follows, for the case of rows, and the explanation is analogous for the columns. First a mask of zeros and ones is formed, with the same length as is the number of rows. For each position in the mask, the symbol defines from which parent the row will be taken to form the child. Next a child is generated according to the scheme. For an example, mask 1010 means that the first and third row will be taken from the first parent and the second and fourth row from the second parent. Each of the symbols has an equal probability of being generated which means that the child will inherit approximately half of the rows from the first parent and half from the second.

Although a gene is considered a single position in the square, we have decided against applying the crossover operator on this scale. The reason is that the sub-blocks (rows and columns) are the smallest units where beneficial changes could occur that we would like to get passed on to the children. In other words, if the sum of the numbers in one of the rows is equal to the target value, we wouldn?t want a crossover operator to break the row.

The chosen crossover operator can generate infeasible solutions, where any of the numbers from 1 to n^2 is present more than once in the square, and some of them do not appear at all. As was discussed with the design of the problem, these solutions are marked as violating the constraints of the problem and are not further considered.

2.5.2 The mutation operator

The mutation operators for combinatorial optimization problems like the magic square have to respect certain constraints. They can never generate numbers out of the 1 to n^2 range and each of the numbers in the range may be present only once. This limits the mutation to various permutation operators.

The permutation operator that was used in the algorithm was a swap mutation. It randomly selects a position in the square and then randomly selects another position in the same row or column as the first position that was selected. It then exchanges values held at the two positions.

Chapter 3

Implementation

3.1 MOEA Framework

The problem was implemented in the Java programming language. The study used the MOEA Framework free and open source Java library, version 2.1, available from http://www.moeaframework.org/. The MOEA Framework contains a comprehensive suite of tools for analyzing the performance of algorithms. It supports both run-time dynamics and end-of-run-analysis. Run-time dynamics capture the behavior of an algorithm throughout the duration of a run, recording how its solution quality and other elements change. End-of-run analysis, on the other hand, focuses on the result of a complete run and comparing the relative performance of various algorithms.

An adjustment had to be made to the existing library to weaken the constraints on the Permutation subclass of the Variable class to permit the crossover operator that was explained in the operators section. Three additional classes were created, one to specify the problem and two classes for the modi ed crossover and mutation operators.

3.2 Evolution Algorithm Performance Indicators

The performance of the NSGA-II evolutionary algorithm is estimated using quantitative performance measures implemented in the MOEA Framework Java Library. The ve performance indicators are: Hypervolume (HV), Generational Distance (GD), Inverted Generational Distance (IGD), Additive epsilon Indicator (AEI) and Maximum Pareto Front Error (MPFE). The performance is measured both in terms of convergence and diversity. First a reference set is computed which will be used for comparison. Multiple simulations are run and the best solution obtained is used as the reference set.

Chapter 4

Conclusion

Bibliograpy:

[1] Crawford, B. et al., Solving Constraint Satisfaction Puzzles with Constraint Programming, Proceedings of the Third International Conference on Convergence and Hybrid Information Technology, 2008., vol. 2, 926-931.

[2] Coello Coello, C. A., Lamont, G. B., Van Veldhuizen, D. A. Evolutionary Algorithms for Solving Multi-Objective Problems, 2nd edition, Springer

[3] N. Srinivas and Kalyanmoy Deb, Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms, Evolutionary Computation 2 (1994), no. 3, 221 ? 248.

[4] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan, A Fast Elitist Multiobjective Genetic Algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation 6 (2002), no. 2, 182 - 197.

[5] Talbi, E., Metaheuristics: From Design to Implementation, Wiley Publishing, 2009.

[6] Rossi, F., van Beek, P., Walsh, T. Handbook of Constraint Programming, Elsvier, 2006.

[7] Mantere, T., Solving, rating and generating Sudoku puzzles with GA, IEEE Congress on Evolutionary Computation, 1382-1389, 2007.

[8] https://en.wikipedia.org/wiki/Magic$_s quare$.