ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)
UDC/UDK 004.02:004.424.4

CHOICE FUNCTIONS FOR AUTONOMOUS SEARCH IN CONSTRAINT PROGRAMMING: GA VS. PSO

Ricardo Soto, Broderick Crawford, Sanjay Misra, Wenceslao Palma, Eric Monfroy, Carlos Castro, Fernando Paredes

Original scientific paper

The variable and value ordering heuristics are a key element in Constraint Programming. Known together as the enumeration strategy they may have important consequences on the solving process. However, a suitable selection of heuristics is quite hard as their behaviour is complicated to predict. Autonomous search has been recently proposed to handle this concern. The idea is to dynamically replace strategies that exhibit poor performances by more promising ones during the solving process. This replacement is carried out by a choice function, which evaluates a given strategy in a given amount of time via quality indicators. An important phase of this process is performed by an optimizer, which aims at finely tuning the choice function in order to guarantee a precise evaluation of strategies. In this paper we evaluate the performance of two powerful choice functions: the first one supported by a genetic algorithm and the second one by a particle swarm optimizer. We present interesting results and we demonstrate the feasibility of using those optimization techniques for Autonomous Search in a Constraint Programming context.

Keywords: *Artificial Intelligence, Autonomous Search, Constraint Programming*

Funkcije izbora za samostalno pretraživanje u ograničenom programiranju: genetski algoritam nasuprot optimizaciji roja čestica

Izvorni znanstveni članak

Heurističke metode nizanja vrijednosti i varijabli su ključni element u ograničenom programiranju. Poznate su kao strategija nabrajanja i mogu značajno utjecati na postupak rješavanja problema. Međutim, prilično je teško izabrati odgovarajući heuristički postupak jer je kompliciranо predvidjeti njihovo ponašanje. U zadnje je vrijeme za tu svrhu predloženo samostalno (autonomno) pretraživanje. Ideja je da se strategije koje su se pokazale lošima tijekom postupka rješavanja dinamički zamijene onima koje više obećavaju. Ta se zamjena izvodi korištenjem funkcije izbora, koja u zadanom vremenu procijenjuje ponuđenu strategiju preko indikatora kvalitete. Važnu ulogu u tom procesu ima optimizator kojemu je cilj fino podešavanje funkcije izbora kako bi se garantirala precizna procjena strategija. U ovom radu evaluiramo karakteristike dviju jakih funkcija izbora: prvu podržava genetski algoritam, a drugu optimizator roja čestica. Dajemo interesantne rezultate i demonstriramo mogućnost korištenja tih metoda optimiziranja za samostalno pretraživanje u kontekstu ograničenog programiranja.

Ključne riječi: *ograničeno programiranje, samostalno pretraživanje, umjetna inteligencija*

1 Introduction

Constraint Programming (CP) is a modern technology for solving constraint satisfaction and optimization problems. It inherits ideas and techniques from different domains such as artificial intelligence, operational research, and programming languages. Currently, several software solutions are supported by CP in a wide range of application areas, e.g., in manufacturing for minimizing the transport among production cells, in health care centres for nurse rostering, in engineering for the design of complex structures, in mathematics for solving cryptarithmetic puzzles, in games for agent simulation, and even for sequencing the DNA in molecular biology [16].

In CP, problems are formally stated in terms of variables and constraints. The variables hold a domain and represent the unknowns of the problem, while the relations among them are modelled as constraints. This formal representation is known as Constraint Satisfaction Problem (CSP). A solution to a CSP is found when each variable adopts, from its domain, values that completely satisfy the constraints. CSPs are solved by using a search engine, commonly called solver. This engine explores the potential solutions, which are distributed in a tree data structure, in order to reach a solution. This process considers enumeration and propagation phases. The enumeration is responsible for creating tree branches by assigning permitted values to the variables. The propagation aims at deleting from domains the unfeasible values by employing consistency techniques [16].

Within the enumeration process, the tree branches can be generated in different ways, depending on which values and variables are chosen. These choices are known as the variable and value ordering heuristics, and together form the enumeration strategy. The enumeration strategy is a major component of the process; in fact a wrong decision may cause dramatic effects on the resolution. However, taking the right decision is quite complicated as the behaviour of the strategy is hard to predict. Autonomous Search (AS) has been recently proposed to tackle this concern. The idea is to dynamically replace strategies exhibiting bad performances by more promising ones during the resolution. In this framework, enumeration strategies are classified in a quality rank, from which promising ones are chosen. Such a rank is generated by a choice function that incorporates indicators able to estimate the quality of strategies in a given interval of processing time. Those indicators and in turn the choice function are finely tuned by an optimizer in order to precisely evaluate the strategies. The tuning process is mandatory as the correct configuration of indicators has crucial effects on the ability of solving a specific problem. In previous works [6, 7, 8, 9] (see Sec. 2 for details), this tuning process is carried out by a genetic algorithm (GA) yielding excellent results for well-known CP benchmarks. In this paper, we introduce a new tuning-component based on particle swarm optimization (PSO).

We compare both optimization techniques and we illustrate interesting experimental results by demonstrating the feasibility of using those techniques for efficiently combining CP with AS.

This paper is structured as follows. The related work is presented in Section 2 followed by the preliminaries. The choice functions and the corresponding tuning process are presented in Section 4. Experiments are described in Section 5. Finally, we conclude and give some directions for future work.

2 Related work

As we have noted, predicting the behaviour of a strategy is quite hard in CP. To handle this concern, different approaches have been proposed. For instance, in [10] a framework that learns ordering heuristics is proposed. The idea is to examine the trace of solutions to problems that have been successfully solved. The approach handles a set of advisors that suggest actions to perform in the form of a comment. The reliability and utility of advisors is controlled by weights, which allows taking the final decision by computing a weighted combination of the comments done by the advisors. The weighted degree heuristic [4] follows a similar goal. Here, constraints are linked to weights to be augmented along the propagation process. Then, the weight sum for each variable associated to constraints is calculated in order to select the one with the largest value. An improvement to this method is proposed in [11, 19]. Authors argue that initial choices are most important than older ones, then sampling is carried out in a preliminary phase.

AS has been recently integrated to CP for tackling the aforementioned concern. The pioneer framework for AS in CP was proposed in [5]. This approach was implemented in OZ [17] and some preliminary experiments on the *n*-queens problem illustrated promising results. In [7, 8], the pioneer framework has been improved and re-implemented in Eclipse [1]. Here, a choice function supported by a genetic algorithm [14] was incorporated in order to perform a better rank generation. A more modular and extensible version of this framework has also been implemented, the benefits from a software engineering and architectural standpoint are described in [9]. In [15], this architecture is exploited in order to gather information about the search process. In fact, the focus is the combination of strategies generated by the dynamic replacement, which is called strategy blend. Some good blends are generated for the *n*-queens problem, the magic squares, and the knight tournament problem. Extended versions of such works considering more elaborated statistical analysis were reported in [13] for optimization problems and in [6] for constraint satisfaction problems.

Unlike previous work, in this paper we focus on the choice functions, we introduce a new PSO-based choice function and we illustrate interesting experimental results. Our goal is to compare different optimization approaches and to demonstrate the feasibility of using those evolutionary computing techniques for Autonomous Search in Constraint Programming.

3 Preliminaries

3.1 CSPs

A CSP is a formal representation of unknowns namely the variables, and relations among them called constraints. Formally, a CSP P is defined by a triple $P = \langle X, D, C \rangle$ where:

- X is a finite sequence of variables $X = \langle x_1, x_2, ..., x_n \rangle$.
- D is a finite set of domains $D = \langle d_1, d_2, ..., d_n \rangle$, such that $x_i \in d_i$ and d_i is a set of values for $i \in 1, ..., n$.
- C is a finite set of constraints $C = \langle c_1, c_2, ..., c_n \rangle$.

A solution to a CSP is an assignment $\{x_1 \rightarrow a_1, ..., x_n \rightarrow a_n\}$ that satisfies the whole set of constraints.

As an example, let us consider the *n*-queens problem, which consists in placing *n* chess queens on an *n x n* chessboard so that no two queens are able to attack each other by using the standard moves of chess queens. A solution requires that no two queens share the same column, row, or diagonal (see Fig. 1).

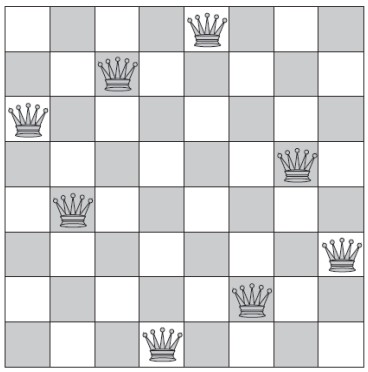

Figure 1 A solution to the *n*-queens problem (*n*=8)

Considering *n*=8, we identify eight variables, which can be expressed as $X_1, ..., X_8$, where X_i denotes the row position of the queen placed in the i^{th} column of the chessboard. The set of domains d_i for each of these variables is given by the integer interval [1,8], which represents the possible cells that queens can take. Then, we state the constraints of the problem as the following inequalities for $i \in [1,7]$ and $j \in [i+1,8]$:

- To ensure that no two queens share the same row: $X_i \neq X_j$.
- To ensure that no two queens share the same NW-SE diagonal: $X_i - i \neq X_j - j$.
- To ensure that no two queens share the same SW-NE diagonal: $X_i + i \neq X_j + j$.

A solution can be represented by the following assignment that satisfies the whole set of constraints:

$$\{x_1 \rightarrow 3, x_2 \rightarrow 5, x_3 \rightarrow 2, x_4 \rightarrow 8, x_5 \rightarrow 1,$$
$$x_6 \rightarrow 7, x_7 \rightarrow 4, x_8 \rightarrow 6\}.$$

3.2 CSP Solving

Fig. 2 depicts a classic algorithm for CSP solving. The idea is to generate partial solutions until a result is reached, applying backtracking when inconsistencies are found. The procedure begins with a loop including a set of instructions to be executed until a solution is found (i.e. all variables are fixed) or no solution is found (i.e. a failure is detected). Next, the corresponding variable and value are selected to create the branches of the tree. Then, the propagation is triggered in order to temporarily eliminate from domains unfeasible values. At the end, two conditional statements are responsible for backtracking. Backtrack allows the algorithm to come back to the previous instantiated variable that has still chance to reach a solution, while the shallow backtrack [3] instantiates the current variable with the following available value from its domain.

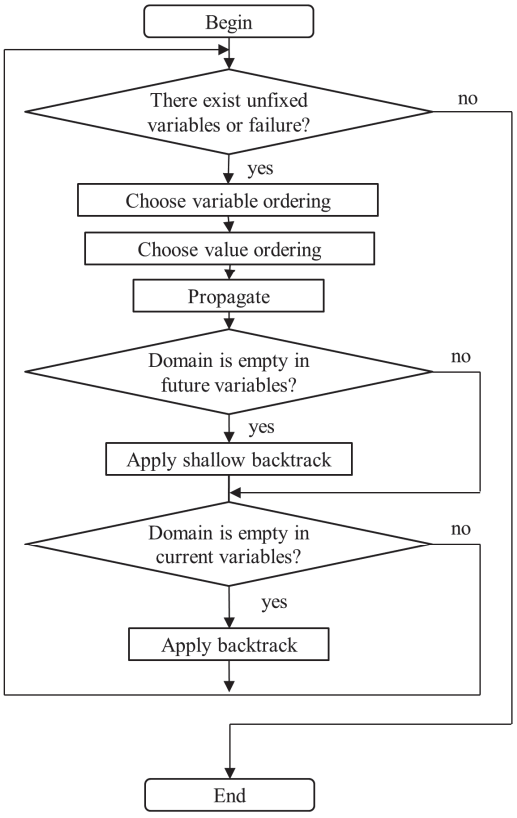

Figure 2 A flowchart describing a classic procedure for CSP solving

3.2 Integrating autonomous search in constraint programming

The framework for AS in CP can be seen as a 4-component architecture (see Fig. 3): SOLVE, OBSERVATION, ANALYSIS and UPDATE

- The SOLVE component carries out the CSP resolution by combining propagation and enumeration steps. The strategies employed in the process are selected from the quality rank, which is managed by the choice function.
- OBSERVATION is responsible for taking and recording snapshots, which correspond to relevant information of the solving process.
- ANALYSIS processes the snapshots captured by OBSERVATION. These snapshots are used to evaluate the strategies via indicators, which are stored in a database to be then gathered by UPDATE. Indicators used for the current experiments are described in Tab. 1; an extended list is presented in [6].
- UPDATE is responsible for organizing the strategy rank. This task is done by using the choice function, which through indicators determines the quality of strategies along the solving process. Those indicators and the choice function are finely tuned by an optimizer in order to precisely evaluate the strategies. In this work, the UPDATE component is supported by two choice functions: one based on a genetic algorithm and a second one based on particle swarm optimization.

Table 1 Indicators for Autonomous Search

Name	Description
n	Number of step; note that *n* increments each time a variable is fixed by enumeration.
$Tn(S_j)$	Number of steps from *X* to *Y*, where *X* is the last time that strategy S_j was activated and *Y* corresponds to step *n*-th
VFP	Number of variables fixed by the propagation phase
SB	Number of shallow backtracks
B	Number of backtracks
B-real	*SB*+*B*
dt	Current depth in step *t*
In1	A measure to evaluate a variation of the maximum depth, it is calculated as: *CurrentMaximumDepth* - *PreviousMaximumDepth*
In2	*dt* - *dt*-1
Thrash	*dt*-1 - *VFPt*-1

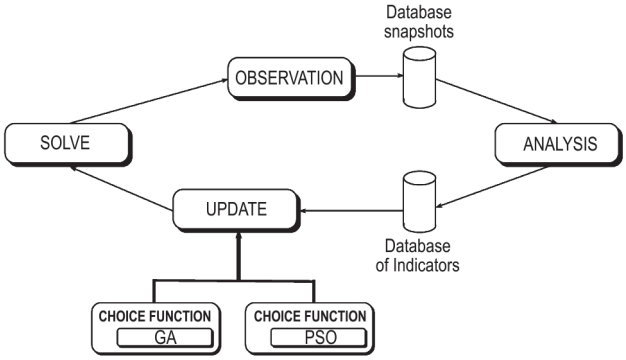

Figure 3 Architecture for Autonomous Search including choice functions

3.3 Choice function computation

The choice function [18] is a key component of the architecture. It is responsible for generating the rank of enumeration strategies and for choosing the corresponding promising strategy at each step. The strategy selected is the one with the highest choice function value. Steps -or decision points- refer to as when the solver is requested to instantiate a variable. Formally, a choice function *f* in step *n* for the strategy S_j is defined as follows:

$$f_n(S_j) = \sum_{i=1}^{l} \alpha_i f_{in}(S_j), \quad (1)$$

where l is the number of indicators, and α is the parameter for controlling the relevance of the indicator in the equation.

Now, the classic procedure for solving CSPs can be modified to be integrated with Autonomous Search. Fig. 4 depicts the new algorithm. At the end, three new instructions have been added: one for computing the indicators, another for computing the choice function; and the last one for selecting promising strategies. Let us note that AS functions are called after propagation since some indicators may be impacted by this phase. This allows one to capture the real effects of the strategy on the given processing time.

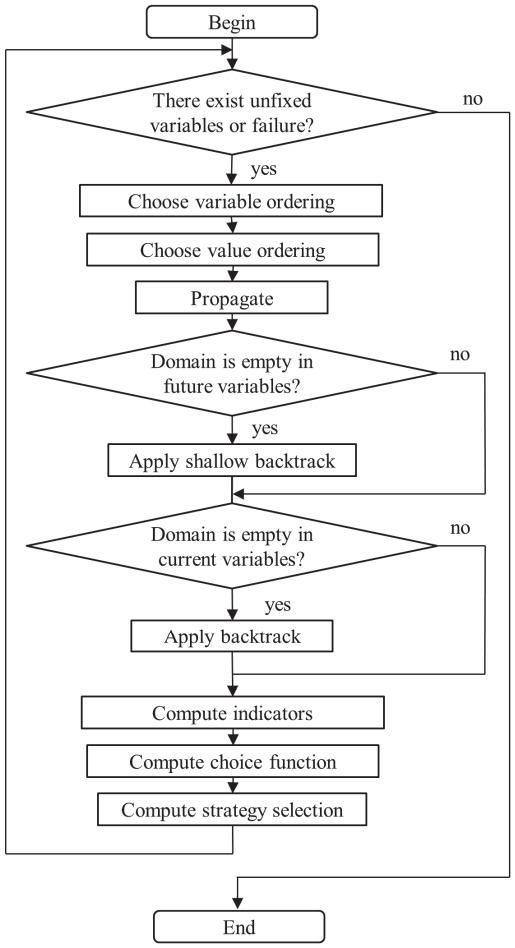

Figure 4 A flowchart describing the procedure for CSP solving including Autonomous Search

Then, as a result of applying this algorithm to any CSP, the resolution process is carried out by combining the most promising strategies. As an example, let us recall the 8-queens problem. It has been solved by using a portfolio of 8 enumerations strategies, which come from combining different variable ordering and value ordering heuristics (details about the portfolio are given in Table 2). The track of the resolution process by using AS and CP is depicted in Fig. 5. The *X*-axis represents the id of the strategy and the *Y*-axis represents the percentage of the complete solving time that the strategy works until being replaced. In the example, the strategy number 7 begins the process by working almost 8 % of the whole solving time; strategy 6 follows by slightly exceeding 2 %, then strategy 3 works about 2 %, and so on.

Table 2 Portfolio of 8 enumeration strategies

Id	Variable ordering	Value ordering
S1	The first variable in the list	min. value in domain
S2	The variable with the largest domain	min. value in domain
S3	The variable with the smallest domain	min. value in domain
S4	The most constrained variable	min. value in domain
S5	The first variable in the list	max. value in domain
S6	The variable with the largest domain	max. value in domain
S7	The variable with the smallest domain	max. value in domain
S8	The most constrained variable	max. value in domain

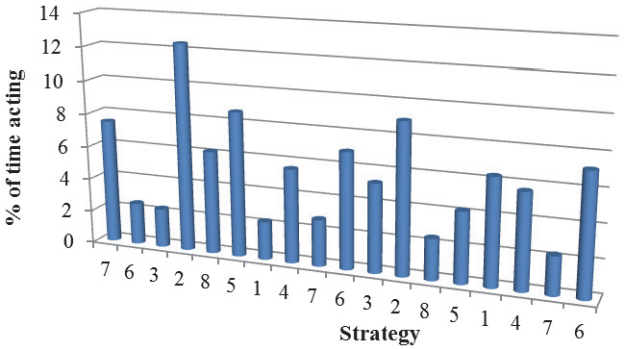

Figure 5 A chart illustrating the track of the solving process for the 8-queens problem by using AS+CP

4 Choice function tuning

As previously mentioned, the choice function must be tuned by correctly adjusting the α parameters in order to precisely evaluate the strategies. This process is mandatory since a correct parameter setting has positive incidence on the efficient solving of a given CSP. However, parameter tuning is quite complex to achieve due to the fact that most parameters are problem-dependent and their best values are not stable along the search. This tuning process can be seen as the resolution of an optimization problem whose solution should contain the optimal setting of the choice functions.

Thus, for determining the most appropriate configuration, the α parameters are fine-tuned by an optimizer. To this end, a sampling phase is performed where the CSP is solved to a given stop criterion such as a given number of variables instantiated, number of visited nodes, or number of backtracks. The sampling phase allows one to gather initial information and to train the choice function. After this phase, the CSP is solved by using the most successful set of parameter values for the choice function. Currently, the UPDATE component is supplied of choice functions based on two optimization techniques: genetic algorithms and particle swarm optimization. Both approaches are described in the following.

4.1 GA-based choice function

In order to avoid an error-prone manual parameterization, the first choice function is supported by a genetic algorithm (GA). In this approach, each chromosome of the population represents the α

parameters of a choice function. In a sampling phase, each chromosome is evaluated attempting to solve the problem to a given stop criterium. The number of backtracks, as indicator of process performance, is used to evaluate the chromosome. It is a standard measure of quality of both constraint propagation and enumeration, (solving time is commonly inaccurate). After evaluation, selection, crossover, and mutation operations are applied to the population so as to generate a new generation of choice functions. In the crossover phase, two chromosomes are randomly selected by the operator and then mated by swapping their genes. Both new chromosomes are then included in the list of candidate solutions. In the mutation phase, the operator acts over the genes of chromosomes mutating them according to the given rate. Finally, mutated individuals are also added to the list of potential solutions.

The GA was implemented using the Java Genetic Algorithm Package (JGAP) version 3.5[1]. The chromosome is composed of genes representing the choice function α's, where $-100 < \alpha_i < 100$ (an example of representation can be seen in Tab. 3).

Table 3 Example of chromosome representation

	α_1	α_2	α_3	…	α_l
Individual 1	50	−20	40	−35	90
Individual 2	30	70	−45	100	−20

A population of 10 individuals is randomly generated for the experiments. The parent selection used is the tournament selection considering 0,75 as the tournament selector parameter. The survival selection employed is handled by the JGAP tournament selector which acts until the population size is completed. Also, we implement uniform crossover and simple mutation, employing 0,5 as crossover rate and mask probability; and 0,1 as mutation rate.

4.2 PSO-based choice function

In PSO, each particle in the swarm represents a candidate solution to the optimization problem. Each particle moves with an adaptable velocity through the search space, adjusting its position according to its own experience and that of neighbouring particles. In the current implementation, each particle encodes the parameters of a choice function generating a choice function instance. A particle i is denoted as x_i where each dimension of the particle represents a α and l is the number of indicators of the choice function. Analogously to the previous choice function, particles are evaluated in a sampling phase attempting to solve the CSP partially to a fixed cut-off. An indicator of the process performance is used as a fitness value for the particle. At each iteration, the algorithm updates velocities and positions as described by Eq. 2 and 3, respectively [12].

$$v_i^{k+1} = wv_i^k + c_1 v_1^k (p_i^k - x_1^k) + c_2 v_2^k (p_g^k - x_i^k), \quad (2)$$

$$x_i^{k+1} = x_i^k + v_1^{k+1}, \quad (3)$$

where v_i is the velocity of particle x_i at iteration k, w is the inertia factor, c_1 and c_2 correspond to the cognitive and social parameter respectively, p_g is the best position of neighbours, p_i is the best position encountered by the particle, r_1 and r_2 are random numbers uniformly distributed in [0, 1], and N is the number of particles in the swarm.

The configuration employed for the experiments considers 30 particles, 0,9 as inertia factor, and 0,9 and 1,5 as c_1 and c_2, respectively. We have performed 100 iterations considering a cut-off of 30 %, using as fitness the number of backtracks. Let us note that a survey of both optimization techniques used can be seen in [2].

5 Experiments

We have performed a set of experiments in order to compare the performance of both choice functions. Experiments have been carried out on well-known benchmarks: N-Queens (N={8,10,20,50}) and Magic Squares (N={4,5}). The portfolio of strategies used is the one described in Tab. 2.

Tab. 4 depicts the number of backtracks for N-Queens (N={8,10,20,50}) and Magic Squares (N={4,5}) considering the 8 single strategies (S1 to S8), the GA-based choice function (GA-CF) and the PSO-based choice function (PSO-CF). As previously mentioned, due to the solving time being often inaccurate, we use the number of backtracks to measure the search effort. We use as stop criterion 65 535 steps, which is the maximum buffer size of our statistical tool. Let us recall that a step refer to as when the solver is requested to instantiate a variable by enumeration.

Table 4 Number of backtracks for single strategies (S1 to S8), and combinations of them by using a GA-based and a PSO-based choice function

	N-Queens				Magic Squares	
Strategy	N=8	N=10	N=20	N=50	N=4	N=5
S1	10	6	10 026	>27 406	12	910
S2	11	12	2539	>39 232	1191	>46 675
S3	10	4	11	177	3	185
S4	10	6	10 026	>26 405	10	5231
S5	10	6	10 026	>27 406	51	>46 299
S6	11	12	2539	>39 232	42	>44 157
S7	10	4	11	177	97	>29 416
S8	10	6	10 026	>26 405	29	>21 847
GA-CF	4	6	0	7	0	7
PSO-CF	6	2	11	3	3	53

Results show the performance of both choice functions. GA-CF and PSO-CF outperform single strategies in most problems. GA-CF is the best for all problems except for N-Queens (N=10), where the number of backtracks achieved is equivalent to the one of S1, S4, S5, and S8. PSO-CF also gains excellent position in the global ranking. It is able to outperform single strategies for N-Queens (N={8,10,50}) and Magic Squares (N=5). For N-Queens (N=20), PSO-CF ties with S3 and S7, and for Magic Squares (N=4) the performance is equivalent to the one of S3. Now, contrasting both choice functions, GA-CF exhibits better performance than PSO-CF for N-

[1] http://jgap.sourceforge.net/

Queens (N={8,20}) and Magic Squares (N={4,5}), but is outperformed by PSO-CF for N-Queens (N={10,50}). A graphical comparison of both choice functions can be seen in Fig. 6.

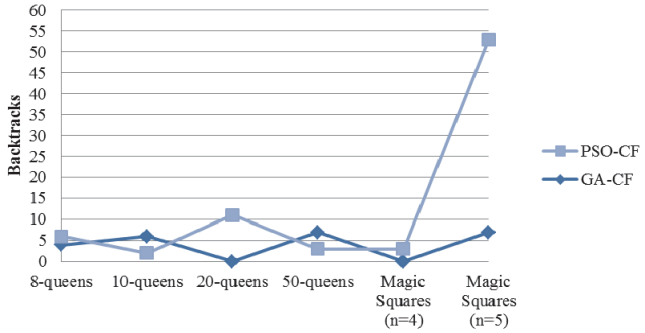

Figure 6 Comparing the GA-based and PSO-based choice function in terms of the number of backtracks

Table 5 Number of visited nodes for single strategies (S1 to S8), and combinations of them by using a GA-based and a PSO-based choice function

	N-Queens				Magic Squares	
Strategy	*N*=8	*N*=10	*N*=20	*N*=50	*N*=4	*N*=5
S1	24	19	23 893	>65 535	37	1901
S2	21	25	4331	>65 535	1826	>65 535
S3	25	16	51	591	22	546
S4	25	19	24 308	>65 535	31	13 364
S5	24	19	23 893	>65 535	110	>65 535
S6	21	25	4331	>65 535	69	>65 535
S7	25	16	51	591	230	>65 535
S8	25	19	24 308	>65 535	61	>65 535
GA-CF	14	22	20	66	16	40
PSO-CF	24	21	72	60	30	361

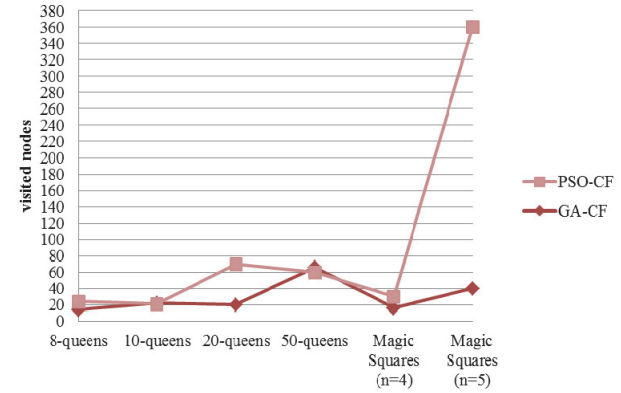

Figure 7 Comparing the GA-based and PSO-based choice function in terms of visited nodes

Tab. 5 depicts the number of visited nodes for the same benchmarks and strategies. It is also a useful measure of performance. Here, GA-CF and PSO-CF also exhibit good results. GA-CF is the best for all problems except for N-Queens (N=10), where it only outperforms S2 and S6. PSO-CF is able to keep close to the best values of N-Queens (N={8,10,20}) and Magic Square (N=4). However PSO-CF overcomes GA-CF for N-queens (N=10) as well as the 8 single strategies for Magic Square (N=4). Finally, it is the best one for the hardest instance of N-Queens. A graphical comparison of both choice functions for visited nodes can be seen in Fig. 7.

6 Conclusion

Choice functions are a key element of frameworks for Autonomous Search in Constraint Programming. They are responsible for ranking enumeration strategies in order to select the more promising ones for the solving process. The selection process is based on the evaluation of a set of indicators that measure the quality of strategies in a given amount of processing time. To guarantee a precise evaluation, the quality indicators as well as the choice functions are finely tuned by an optimizer. This process is quite hard as most choice function parameters are problem-dependent and their best values are not stable along the search. This tuning process can be seen as the resolution of an optimization problem whose solution should contain the optimal configuration of the choice functions. In this work we have introduced a new PSO-based choice function and we have compared it with a previously implemented GA-based choice function. Interesting and promising results on well-known benchmarks have been illustrated. Both choice functions are able to outperform the classical enumeration strategies used in CP.

A straightforward direction for future work is to design and implement new choice functions, for instance based on additional optimization techniques such as ant and bee colony optimization. Designing new combinations of enumeration strategies as well as the design of new statistical methods for enhancing the choice function will be another interesting direction to follow as well.

7 References

[1] Krzysztof, A.; Wallace, M. Constraint Logic Programming using Eclipse. Cambridge University Press, 2006.

[2] Balázs, K.; Horváth, Z.; Kóczy, L. T. Different Chromosome-based Evolutionary Approaches for the Permutation Flow Shop Problem. // Acta Polytechnica Hungarica, 9, 2 (2012), pp. 115-138.

[3] Barták, R.; Rudova, H. Limited assignments: A new cutoff strategy for incomplete depth-first search. // Proceedings of the 20th ACM Symposium on Applied Computing (SAC), pp. 388-392, 2005.

[4] Boussemart, F.; Hemery, F.; Lecoutre, C.; Sais, L. Boosting systematic search by weighting constraints. // Proceedings of the 16th Eureopean Conference on Artificial Intelligence (ECAI), pp. 146-150. IOS Press, 2004.

[5] Castro, C.; Monfroy, E.; Figueroa, C.; Meneses, R. An approach for dynamic split strategies in constraint solving. // In Proceedings of the 4th Mexican International Conference in Artificial Intelligence (MICAI), volume 3789 of LNCS, pp. 162-174. Springer, 2005.

[6] Crawford, B.; Castro, C.; Monfroy, E.; Soto, R.; Palma, W.; Paredes, F. Dynamic Selection of Enumeration Strategies for Solving Constraint Satisfaction Problems. // Rom. J. Inf. Sci. Tech., 15, 2 (2012), pp. 106-128.

[7] Crawford, B.; Soto, R.; Castro, C.; Monfroy, E. A Hyperheuristic Approach for Dynamic Enumeration Strategy Selection in Constraint Satisfaction. // Proceedings of the 4th International Work-conference on the Interplay between Natural and Artificial Computation (IWINAC), volume 6687 of LNCS, pp. 295-304. Springer, 2011.

[8] Crawford, B.; Soto, R.; Castro, C.; Monfroy, E.; Paredes, F. An Extensible Autonomous Search Framework for

Constraint Programming. // Int. J. Phys. Sci., 6, 14 (2010), pp. 3369-3376.
[9] Crawford, B; Soto, R.; Montecinos, M.; Castro, C; Monfroy, E. A Framework for Autonomous Search in the Eclipse Solver. // Proceedings of the 24th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE), volume 6703 of LNCS, pp. 79-84. Springer, 2011.
[10] Epstein, S. L.; Freuder, E. C.; Wallace, R. J.; Morozov, A.; Samuels, B. The adaptive constraint engine. // Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP), Lecture Notes in Computer Science, pp. 525-542. Springer, 2002.
[11] Grimes, D.; Wallace, R. J. Learning to identify global bottlenecks in constraint satisfaction search. // In Proceedings of the Twentieth International Florida Artificial Intelligence Research Society (FLAIRS) Conference, pp. 592-597. AAAI Press, 2007.
[12] Kennedy, J.; Eberhart, R. Particle swarm optimization. // In IEEE International Conference on Neural Networks, pp. 1942-1948, 1995.
[13] Monfroy, E.; Castro, C.; Crawford, B.; Soto R.; Paredes, F.; Figueroa, C. A Reactive and Hybrid Constraint Solver. // J. Exp. Theor. Artif. Intell., 25, 1 (2013), pp. 1-22.
[14] Oiso, M.; Matsumura, Y.; Yasuda, T.; Ohkura, K. Implementing genetic algorithms to CUDA environment using data parallelization. // Technical Gazette, 18, 4 (2011), pp. 511-517.
[15] Soto, R.; Crawford, B.; Monfroy, E.; Bustos, V. Using Autonomous Search for Generating Good Enumeration Strategy Blends in Constraint Programming. // Proceedings of the 12th International Conference on Computational Science and Its Applications (ICCSA), volume 7335 of LNCS, pp. 607-617. Springer, 2012.
[16] Rossi, F.; van Beek, P.; Walsh, T. Handbook of Constraint Programming. Elsevier, 2006.
[17] Schulte, C.; Smolka, G.; Würtz, J. Encapsulated search and constraint programming in OZ. // Proceedings of the 2nd International Workshop on Principles and Practice of Constraint Programming (PPCP), volume 874 of LNCS, pp. 134-150. Springer, 1994.
[18] Soubeiga, E. Development and Application of Hyperheuristics to Personnel Scheduling. PhD thesis, University of Nottingham School of Computer Science, 2009.
[19] Wallace, R.J.; Grimes, D. Experimental studies of variable selection strategies based on constraint weights. // J. Algorithms, 63, 1-3(2008), pp. 114-129.

Authors' addresses

Ph.D. Ricardo Soto, Professor
Pontificia Universidad Católica de Valparaíso
Avenida Brasil 2950, Valparaíso, Chile
and
Universidad Autónoma de Chile,
Pedro de Valdivia 641, Santiago, Chile
Ricardo.Soto@ucv.cl

Ph.D. Broderick Crawford, Professor
Pontificia Universidad Católica de Valparaíso
Avenida Brasil 2950, Valparaíso, Chile
and
Universidad Finis Terrae,
Av. Pedro de Valdivia 1509, Santiago, Chile
Broderick.Crawford@ucv.cl

Ph.D. Sanjay Misra, Professor
Atilim University, Ankara, Turkey
smisra@atilim.edu.ng

Ph.D. Wenceslao Palma, Professor
Pontificia Universidad Católica de Valparaíso
Avenida Brasil 2950, Valparaíso, Chile
Wenceslao.Palma@ucv.cl

Ph.D. Eric Monfroy, Professor
CNRS, LINA, Université de Nantes
2 rue de la Houssinière, Nantes, France
Eric.Monfroy@univ-nantes.fr

Ph.D. Carlos Castro, Professor
Universidad Técnica Federico Santa María
Avenida España 1680, Valparaíso, Chile
Carlos.castro@inf.utfsm.cl

Ph.D. Fernando Paredes, Professor
Escuela de Ingeniería Industrial, Universidad Diego Portales,
Manuel Rodríguez Sur 415, Santiago, Chile
Fernando.Paredes@udp.cl