Information Sciences 308 (2015) 38–48

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Boosting autonomous search for CSPs via skylines

Ricardo Soto [a,b,c], Broderick Crawford [a,d,e], Wenceslao Palma [a,*], Karin Galleguillos [a], Carlos Castro [f], Eric Monfroy [g], Franklin Johnson [h], Fernando Paredes [i]

[a] Pontificia Universidad Católica de Valparaíso, Chile
[b] Universidad Autónoma de Chile, Chile
[c] Universidad Central de Chile, Chile
[d] Universidad Finis Terrae, Chile
[e] Facultad de Ingeniería y Tecnología, Universidad San Sebastián, Chile
[f] Universidad Técnica Federico Santa María, Chile
[g] CNRS, LINA, University of Nantes, France
[h] Universidad de Playa Ancha, Valparaíso, Chile
[i] Escuela de Ingeniería Industrial, Universidad Diego Portales, Santiago, Chile

ARTICLE INFO

Article history:
Received 16 April 2013
Received in revised form 27 November 2014
Accepted 31 January 2015
Available online 9 March 2015

Keywords:
Combinatorial optimization
Constraint satisfaction
Hyperheuristic
Skyline

ABSTRACT

Solving constraint satisfaction problems via constraint programming involves the exploration of a search tree where the potential solutions are distributed. The exploration phase is essentially controlled by an enumeration strategy that decides the order in which variables and values are selected to verify its feasibility. This process is known to be quite important, indeed perfect enumerations can reach a solution without useless explorations. However, selecting good strategies in advance is quite hard as the effects along the search are often unpredictable. Autonomous search addresses this concern by proposing to replace on the fly bad-performing strategies by more promising ones. Strategies are selected from a quality rank which is generated in function of their performance on the current solving process. However, the ranking computation is commonly tuned by an optimizer that negatively impacts the performance of the whole resolution. In this paper, we propose a faster autonomous search approach by integrating a powerful database technique called skyline. This technique allows us to avoid the use of costly rank functions and optimizers, accelerating as a consequence the solving process. We report results where the skyline-based approach clearly competes with previously reported autonomous search frameworks as well as with classic and more sophisticated heuristics such as impact-based search and *dom/wdeg*.

1. Introduction

Constraint programming (CP) is a widely employed technology for solving constraint satisfaction and optimization problems, that has successfully been employed to practical applications in various domains such as robotics [5,39], rostering [21,29], scheduling [36,7], manufacturing [38,2], supply chains [33,27], allocation [35,10] and bioinformatics [3,1]. This technology allow users to model a problem as a constraint satisfaction problem (CSP), which can be seen as a formal problem

* Corresponding author. Tel.: +56 32 2273761; fax: +56 32 2273859.
E-mail address: wenceslao.palma@ucv.cl (W. Palma).

http://dx.doi.org/10.1016/j.ins.2015.01.035
0020-0255/

representation mainly composed of an n-tuple of variables and an m-tuple of constraints. A CSP is solved once a n-tuple of values satisfying the constraints is found. The classical procedure for solving CSPs consists in employing a tree-data structure containing the possible solutions. Then, different algorithms can be used to explore and reach a solution, simple ones such as the well-known backtracking or more sophisticated ones such as the forward checking and maintaining arc consistency from the CP technology. In this context, two main phases are performed. The first one, called enumeration, assigns values to variables generating partial solutions that can be seen as branches of the tree. The second one, named propagation, tries to prune the tree by filtering from domains those values that do not lead to any solution.

The enumeration phase can be decomposed into two steps, the first one selects a variable from the problem and the second one temporarily assign a value to this variable. Both selections are controlled by the variable and value ordering heuristic, which together are known as the enumeration strategy. The enumeration strategy is an essential component of the solving process. A correct decision may dramatically reduce the solving time. In fact, perfect enumerations can reach a solution without performing backtracks on the search tree. However, much tuning can be done to choose the correct strategy, which in practice is a hard task as the performance of strategies is hard to predict. A modern proposal [23], named autonomous search (AS), addresses this concern. The idea is to let solvers adapt and control themselves during the solving phase in order to reach efficient solving processes but avoiding user involvement in tuning. In this way, simpler and powerful solvers will be available for non-expert users.

During the last two years, promising frameworks have incorporated AS in CP [15,17,30,16,37,14], the goal is to provide adaptive solvers able to activate the most appropriate strategy on each part of the search tree. Strategies are selected on the fly from a quality rank and depending on their performance they may be replaced by more promising ones. A choice function is responsible for measuring the performance of strategies during the resolution. The performance is computed through several indicators which attempt to reflect the real state of progress in the problem resolution. An optimizer is commonly introduced to tune the computation of the choice function for a better rank generation. However, the optimization process is commonly costly and negatively impacts the performance of the whole resolution.

Considering the aforementioned concern, our research challenge is to boost the CSPs solving process by providing a more efficient way to rank the enumeration strategies. The main contribution of this paper is the design of an improved AS framework for CP able to reach faster resolution phases. On the one hand, the support of AS allow us to alleviate the burden of user involvement in solver tuning. On the other hand, the resolution process is enhanced by the integration of a powerful ranking technique from the database domain named skyline [8]. The use of skyline allows us to obviate the need for costly rank functions and optimizers, as a consequence the whole resolution is accelerated. We report encouraging results where the skyline-based approach is able to compete with previously reported autonomous search frameworks as well as to classic and more sophisticated heuristics such as impact-based search and *dom/wdeg*.

The rest of this paper is organized as follows. Background information about CP and related works is presented in Section 2. The new approach based on skyline is described in Section 3. Section 4 presents the benchmark problems and the experimental results. Finally, in Section 5 we conclude.

2. Background information

In this section we briefly survey CSPs. Then, we present the related work.

2.1. Constraint satisfaction problems

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

- $\mathcal{X}$ is an n-tuple of variables $\mathcal{X} = \langle x_1, x_2, \ldots, x_n \rangle$.
- $\mathcal{D}$ is a corresponding n-tuple of domains $\mathcal{D} = \langle d_1, d_2, \ldots, d_n \rangle$ such that $x_i \in d_i$, and d_i is a set of values, for $i = 1, \ldots, n$.
- $\mathcal{C}$ is an m-tuple of constraints $\mathcal{C} = \langle c_1, c_2, \ldots, c_m \rangle$, and a constraint c_j is defined as a subset of the Cartesian product of domains $d_{j_1} \times \cdots \times d_{j_{n_j}}$, for $j = 1, \ldots, m$.

A solution to a CSP is an assignment $\{x_1 \to a_1, \ldots, x_n \to a_n\}$ such that $a_i \in d_i$ for $i = 1, \ldots, n$ and $(a_{j_1}, \ldots, a_{j_{n_j}}) \in c_j$, for $j = 1, \ldots, m$.

As an example, let us consider the magic squares problem, which consists in filling a $N \times N$ matrix with numbers from 1 to N^2 such that the sums of each row, each column and the two main diagonals are equal. Here, we identify N^2 variables representing the values of the matrix, each one denoted as X_{ij}, where i and j correspond to the row and column position of the value within the matrix, respectively, with $i,\ j \in \{1, N\}$. Each variable X_{ij} range over the domain $\{1, N^2\}$ and the magic sum is defined as $msum = N \times (N \times N + 1)/2$. Then, we can formulate the constraints of the problem as follows:

- The sum of rows are magic sum
  $\forall i \in N;\ msum = \sum_{j=1}^{n} X_{ij}$

11	24	7	20	3
4	12	25	8	16
17	5	13	21	9
10	18	1	14	22
23	6	19	2	15

Fig. 1. A solution of the magic squares problem ($N = 5$).

- The sum of columns are magic sum
  $\forall j \in N;\ msum = \sum_{i=1}^{n} X_{i,j}$
- The sum of the North-West–South-East diagonal is magic sum
  $msum = \sum_{i=1}^{n} X_{i,i}$
- The sum of the South-West–North-East diagonal is magic sum
  $msum = \sum_{i=1}^{n} X_{i,N-i+1}$
- The elements of the matrix are all different[1]
  $\texttt{alldifferent}(X_{1,2}, X_{1,3}, \ldots, X_{N,N})$.

A solution to this problem can be seen in Fig. 1.

2.1.1. Constraint solving

There exist different variations of CSPs such as dynamic CSP [13], composite CSP [34], weighted CSP [26], fuzzy CSP [18], and max-CSP [25], among others. In this paper, we focus on finite domain CSPs, which is the classic and most used one. Algorithm 1 depicts a classic procedure for finite domain 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 a chance to reach a solution, while the shallow backtrack instantiates the current variable with the following available value from its domain.

Algorithm 1. $solve(\mathcal{C}, \mathcal{D})$

```
1: while (¬success) or (failure) do
2:    Variable_Selection(D);
3:    Value_Selection(D);
4:    Propagate_C(D);
5:    if empty_domain_in_future_var then
6:       Shallow_Backtrack();
7:    end if
8:    if empty_domain_in_current_var then
9:       Backtrack();
10:   end if
11: end while
```

2.2. Related work

During the last years there is a trend to track the solving process in order to automatically identify good-performing strategies. Preliminary approaches proposed to sample and learn good strategies after solving a problem. Some examples in this context were reported in [20,19,40]. Reacting before waiting the entire resolution process is a more recent approach. For instance [31] proposes the impact-based search. The idea is to compute the impact of variables by measuring their relevance on the search space reduction. In this way, variables with highest impact should be selected first since they are more likely to reduce the search effort. Another related approach is reported in [9], which proposes to associate weights to constraints. Those weights are incremented once constraints lead to the deletion of a variables domain. Then, the weighted degree (*wdeg*) of a variable corresponds to the sum of weights of constraints where it is involved. Finally, *wdeg* can be used

[1] The $\texttt{alldifferent}(X_1, \ldots, X_n)$ constraint forces that values assigned to the variables $X_1, \ldots, X_n$ must be pairwise distinct [32].

in conjunction with the domain size to produce the *dom/wdeg* heuristic, that selects the variable with the smallest ratio current domain size to current *wdeg*.

A different framework but following a similar goal is proposed in [11]. Here, the idea is to select the most appropriate strategy from a given portfolio for each part of the problem. To this end, strategies are ranked and dynamically selected along the resolution depending on their performance. The performance is measured by a set of indicators associated to the quality of the search process. Such a framework has been used as core of different AS related works. For instance, in [15] a hyperheuristic approach and a choice function were incorporated in order to perform a better rank generation. The hyperheuristic operates at a higher level of abstraction than the solver. It decides which strategy is applied next during the search. This decision is guided by the choice function which evaluates the performance of strategies. Familiar AS-CP frameworks have also been used for solving optimization problems instead of pure CSPs [30].

The aforementioned AS frameworks, based on a hyperheuristic approach, adaptively rank the enumeration strategies (see Fig. 2) where the choice function attempts to capture the correspondence between the historical performance of each enumeration strategy based on information with regard to performance indicators of the solving process (see indicators used in Table 1). Then a rank of strategies is generated from which promising strategies are selected for the next step. Here, a decision point or step is performed every time the solver is invoked to fix a variable by enumeration.

The strategy selected is the one with highest choice function value. Formally, a choice function f in step n for the strategy S_j is defined as follows.

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

where l is the number of indicators, α_i is the parameter for controlling the relevance of the ith-indicator in the equation and $f_{in}(S_j)$ is the value of the ith-indicator for the strategy S_j at step n. In the current approach, the α parameters are finely tuned by an optimizer. A sampling phase is performed where the CSP is solved to a given stop criteria such as a given number of variables instantiated, number of visited nodes, or number of backtracks. Then, the information gathered in this sampling phase is used as input of the optimizer, which determines the most successful set of α values for the choice function.

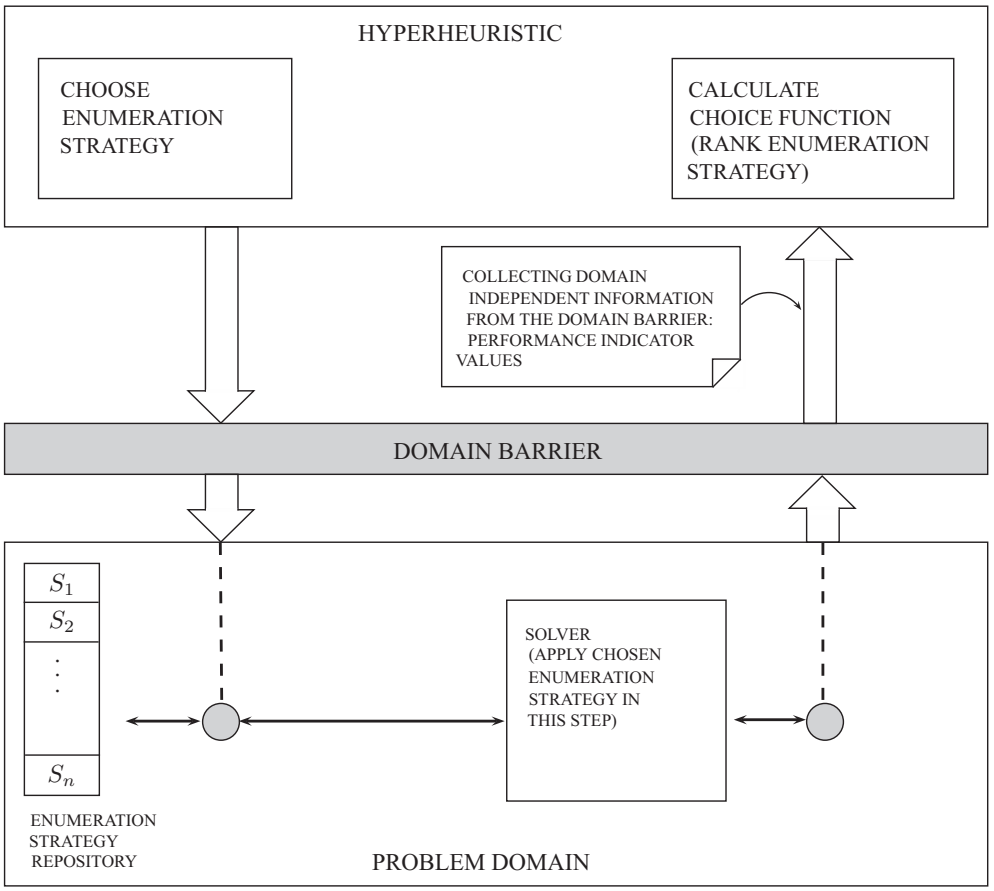

Fig. 2. AS framework [16].

Table 1
Search process indicators.

Name	Description
VFP	Number of variables fixed by propagation
n	Number of steps or decision points (n increments each time a variable is fixed during enumeration)
$T_n(S_j)$	Number of steps since the last time that an enumeration strategy S_j was used until step nth
SB	Number of Shallow Backtracks [4]
B	Number of Backtracks
In1	Represents a Variation of the Maximum Depth It is calculated as: $CurrentMaximumDepth - PreviousMaximumDepth$
In2	Calculated as: $CurrentDepth - PreviousDepth$. A positive value means that the current node is deeper than the one explored at the previous step
B-real	Number of backtracks considering also the number of shallow backtracks
d	Current depth in the search tree
Thrash	The solving process alternates enumerations and backtracks on a few variables without succeeding in having a strong orientation It is calculated as: $d_{t-1} - VFP_{t-1}$

Optimizers based on genetic algorithms [14,17,37] and particle swarm optimization [16] have been used to tune the computation of the choice function. This tuning process is very important because the correct tuning of the choice function weights has a crucial effect on the ability of the solver to properly solve specific problems. Parameter (choice function weights) tuning is difficult to achieve because the parameters are problem dependent and the best values of parameters are not stable along the search [28]. The optimal configuration of the choice function is a form of search too and it can be seen as the resolution of an optimization problem. In this work, we focus on improving the performance of the current AS approach by replacing optimizers by skylines.

3. The skyline approach

In this section we present our approach to boost AS solvers via skylines. We state that the use of skyline avoids the optimal configuration of the choice function leading to a faster solving process while preserving the quality of results. In the following, we survey briefly skyline and we present the details of our approach.

3.1. Skyline

Selecting the most interesting data with respect to user preferences plays an important role in many real applications such as multi-criteria decision making. In databases, this is reflected by the top-k retrieval paradigm where a scoring function f is used to retrieve the k tuples with the highest scores according to f. However, the difficulty in providing f leads to the adoption of the skyline paradigm [8]. Before the introduction of skyline queries into database research, this problem was known as the maximum vector problem or the Pareto optimum [24]. In skylines queries data are represented in a d-dimensional data space and a skyline query retrieves those data points that are not *dominated* by other points. A point dominates another point if it as good or better in all dimensions and better in at least one dimension. Without loss of generality, we assume that smaller values are better than larger ones on all attributes. More formally, a point $p = (p[1], p[2], \ldots, p[d])$ defined over the attribute set D, where $p[i]$ is a value on dimension d_i, dominates another point $q = (q[1], q[2], \ldots, q[d])$ iff $p[i] \leqslant q[i]$ where $1 \leqslant i \leqslant d$ and there is at least one dimension j such that $p[j] < q[j]$. Fig. 3 shows an example of skyline over a small

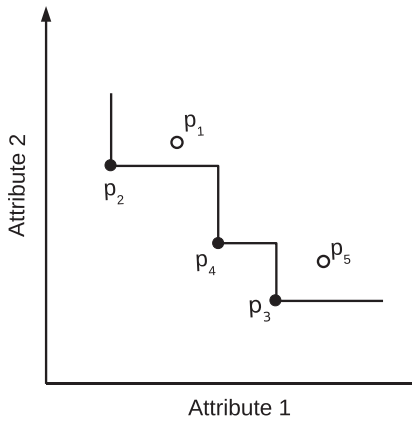

Fig. 3. A skyline example

set of points $P = \{p_1, p_2, p_3, p_4, p_5\}$ with $|D| = 2$. We can see that p_2 dominates p_1 since both coordinates of p_2 are smaller than that of p_1. Thus, p_1 cannot be a skyline point. Since p_2, p_3 and p_4 are not dominated by any other points, they are the skyline in this set of points.

The skyline operator was first introduced in [8], this work proposes the algorithms Block Nested Loop (BNL) and Divide-and-Conquer (D&C). In particular, BNL compares each tuple of the database with all the others and produces a skyline tuple only if it is not dominated. D&C partitions the data set into multiple subsets that can fit in memory. Skylines are computed separately in all partitions and then merged. The Sort-Filter-Skyline (SFS) algorithm [12] is proposed as a variant of BNL that presorts the data, w.r.t. a preference function, obtaining skyline tuples from the sorted list. A hybrid method named LESS [22] improves the aforementioned algorithms combining aspects of SFS and BNL. Roughly, LESS sorts the tuples as does SFS and uses an elimination-filter window acting similarly to the elimination window used by BNL having all of SFS's and BNL's benefits with no additional disadvantages.

There are many others algorithms to compute skylines and each of them assures that if a point p_i dominates a point p_j, p_i is preferable to p_j according to any scoring function which is monotone on all attributes [8]. Moreover, for any scoring function, the skyline always contains the best (the *top*-1) point that minimizes/maximizes the function.

3.2. Boosting AS for CSPs via skylines

Let us recall that in AS approaches, based on hyperheuristics, the enumeration strategies are evaluated by means of a choice function composed by l indicators which capture their performance during the solving process. Moreover, a simple choice function as $CF = \alpha_1 B + \alpha_2 Thrash$ must be optimized in order to find the best α's increasing the runtime of the solving process. Our proposal takes the l indicators, in this case B and *Thrash*, collected during the solving process and puts its values in a 2D space where each enumeration strategy can be seen as a $2D$ point and processed using a skyline algorithm. Fig. 4 shows the skyline of a set of points composed by enumeration strategies described using 2 indicators from Table 1 (B and *Thrash*). In this case, enumeration strategies with the lowest number of backtracks (B) and the lowest number of times that the solving process alternates enumeration and backtracks without succeeding (*Thrash*) are preferable.

Skyline result is composed of not dominated points which are equally preferable but in the case of AS we must choose an enumeration strategy to continue with the solving process. We know that in the skyline result we have the *top*-1 enumeration strategy but we do not know which one is the *top*-1. This sort of tie has been broken using the number of backtracks (B) because it is a good measure for the quality of both constraint propagation and enumeration [16]. Moreover, in [6] it has been shown that the runtime measure is consistent with B. Thus, we sort the skyline result w.r.t. B in increasing order and we choose the first one.

From all the algorithms proposed in the literature we choose BNL as the most straightforward to compute the skyline because in the context of AS at most there are dozens of enumerations strategies and our approach does not work with data stored in files. Thus, our approach does not deal with hard disk issues and the processing of large volumes of data in main memory. Basically, the algorithm compares each point p_i with every other point. If p_i is not dominated, then it belongs to the skyline.

Algorithm 2. $solveAS(\mathcal{C}, \mathcal{D})$

```
1: loadModel();
2: setUpPortfolio();
3: selectStrategyUsingSkyline();
4: while (¬success) or (failure) do
5:    Variable_Selection(D);
6:    Value_Selection(D);
7:    Propagate_C(D);
8:    if empty_domain_in_future_var then
9:       Shallow_Backtrack();
10:   end if
11:   if empty_domain_in_current_var then
12:      Backtrack();
13:   end if
14:   calculate_indicators();
15:   selectStrategyUsingSkyline();
16: end while
```

From a high level point of view, our approach that includes AS and skyline works as follows (see Algorithm 2). The first steps fix the solving options, a model containing the variables of the problem, its domains and the constraints are loaded. A portfolio of enumeration strategies, the indicators of the solving process and the cutoff value (i.e. percentage or number of fixed variables or number of visited nodes or number of backtracks or number of steps) are fixed. Then, the CSP problem

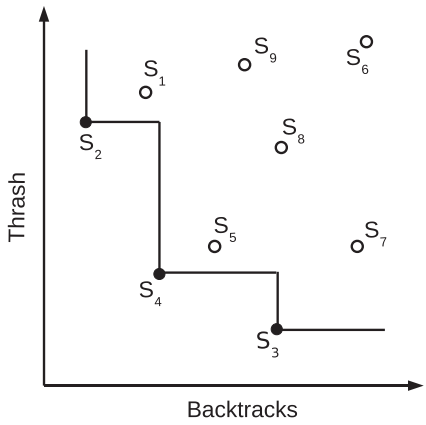

Fig. 4. Skyline of enumeration strategies.

is solved until the cutoff value and the best enumeration strategy is selected using the *selectStrategyUsingskyline*() function. Once the best enumeration strategy is selected, the CSP problem is solved and at each decision point or step (see Section 2.2) the performance indicators are captured, the enumeration strategies are evaluated using the *selectStrategyUsingskyline*() function and the best enumeration strategy is applied next during the solving process. Finally, if all the variables have been fixed, a solution is reported.

4. Experimental evaluation

In this Section we provide a performance evaluation of our approach. Experiments have been performed on a 2.33 GHz Intel Core2 Duo with 4 Gb RAM running Windows Vista, and the benchmarks employed are the following: *n*-queens (NQ) with $n = \{8, 10, 12, 15, 20, 50, 75\}$, magic squares (MS) with $n = \{3, 4, 5\}$, Sudoku, and knight tournament (Kn) with $N = \{5, 6\}$. The stop criterion is 65,535 steps, let us recall that a step refers to a request of the solver to instantiate a variable by enumeration. A portfolio of 8 enumerations strategies has been used, which is detailed in Table 2.

Tables 3 and 4 present the results measured in terms of number of backtracks, Tables 5 and 6 illustrate the runtime. We contrast the proposed approach with eight classic enumeration strategies (S_1–S_8) and three adaptive ones: the best performing hyperheuristic approach (HH) [16], the impact-based search (IBS) [31] and the *dom*/*wdeg* heuristic [9]. At the end, we also include a random selection strategy. Let us note that the portfolio of the skyline approach is composed of the same eight aforementioned classic enumeration strategies.

Results show that the skyline approach is able to compete with classic strategies as well as to the more sophisticated HH, IBS and *dom*/*wdeg*. Taking into account the backtracks, for small instances such as *n*-queens ($n = \{8, 10, 12\}$), skyline has a similar behavior than classic strategies and IBS, being outperformed by *dom*/*wdeg*. In the presence of bigger instances of the *n*-queens ($n = \{50, 75\}$), only two classic strategies are able to solve them (S_3 and S_7), while skyline succeed by requiring more backtracks than HH and *dom*/*wdeg*, but considerably less than IBS. When solving the magic squares problem with $n = 4$, skyline, HH, and *dom*/*wdeg* require a very few backtrack invocations compared to IBS. However, for $n = 5$, the opposite occurs, being IBS the fastest one. Now, considering the Sudoku and the hardest instance of the knight tournament, skyline is the one needing the minor number of backtracks. Finally, in the global backtrack ranking, it is also skyline the one requiring the minor (average) number of backtracks for solving the complete set of problems.

Considering solving times and small instances of the *n*-queens ($n = \{8, 10, 12\}$), IBS is the fastest one, being HH the slowest one by far. This can be explained by the cost of the optimizer within the hyperheuristic compared to the more lightweight

Table 2
Portfolio used.

Id	Strategy	Variable ordering	Value ordering
S_1	F + ID	First variable of the list	min. value in domain
S_2	AMRV + ID	The variable with the largest domain	min. value in domain
S_3	MRV + ID	The variable with the smallest domain	min. value in domain
S_4	O + ID	The variable with the largest number of attached constraints	min. value in domain
S_5	F + IDM	First variable of the list	max. value in domain
S_6	AMRV + IDM	The variable with the largest domain	max. value in domain
S_7	MRV + IDM	The variable with the smallest domain	max. value in domain
S_8	O + IDM	The variable with the largest number of attached constraints	max. value in domain

Table 3
Number of backtracks solving different instances of the N-Queens problem with different strategies.

Problem	Strategy							
	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
NQ($n = 8$)	10	11	10	10	10	11	10	10
NQ($n = 10$)	6	12	4	6	6	12	4	6
NQ($n = 12$)	15	11	16	15	15	11	16	15
NQ($n = 15$)	73	808	1	73	73	808	1	73
NQ($n = 20$)	10,026	2539	11	10,026	10,026	2539	11	10,026
NQ($n = 50$)	>27,406	>39,232	177	>26,405	>27,406	>39,232	177	>26,405
NQ($n = 75$)	>626,979	>36,672	818	>26,323	>26,979	>36,672	818	>26,323
MS($n = 4$)	12	1191	3	10	51	42	97	29
MS($n = 5$)	910	>46,675	185	5231	>46,299	>44,157	>29,416	>21,847
Sudoku	18	10,439	4	18	2	6541	9	2
Kn($n = 5$)	767	>42,889	767	>18,838	767	>42,889	767	>18,840
Kn($n = 6$)	>19,818	>43,098	>19,818	>19,716	>19,818	>43,098	>19,818	>19,716

Table 4
Number of backtracks solving different instances of the N-Queens problem with different strategies.

Problem	Strategy				
	Skyline	HH	*dom/wdeg*	IBS	Random
NQ($n = 8$)	9	6	5	2	5
NQ($n = 10$)	6	5	1	10	8
NQ($n = 12$)	14	16	6	14	18
NQ($n = 15$)	147	11	10	51	98
NQ($n = 20$)	89	11	10	12	32
NQ($n = 50$)	5025	252	25	166,837	>32,340
NQ($n = 75$)	1255	9255	22	266,686	>32,973
MS($n = 4$)	10	3	0	110	17
MS($n = 5$)	171	74,083	657	149	>39,742
Sudoku	0	826	546	45	250
Kn($n = 5$)	1470	667	4060	519,526	>40,022
Kn($n = 6$)	370	29,608	222,768	>670,340	>35,336
$\overline{X}$	714	9562	19,012	>86,677	>15,070

Table 5
Runtime in ms for different instances of the N-Queens problem with different strategies.

Problem	Strategy							
	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
NQ($n = 8$)	125	125	124	125	125	125	141	124
NQ($n = 10$)	125	124	125	125	124	125	125	125
NQ($n = 12$)	156	156	156	156	141	140	156	141
NQ($n = 15$)	296	2106	124	312	296	2137	125	312
NQ($n = 20$)	35,354	7785	156	35,179	34,246	7785	171	34,570
NQ($n = 50$)	t.o.	t.o.	1138	t.o.	t.o.	t.o.	1154	t.o.
NQ($n = 75$)	t.o.	t.o.	6645	t.o.	t.o.	t.o.	6505	t.o.
MS($n = 4$)	140	109	125	156	296	187	405	218
MS($n = 5$)	2919	t.o.	795	17,534	t.o.	t.o.	t.o.	t.o.
Sudoku	156	6271	141	140	156	3338	171	187
Kn($n = 5$)	4306	t.o.	4290	t.o.	4305	t.o.	4352	t.o.
Kn($n = 6$)	t.o.	t.o.	t.o.	t.o.	t.o.	t.o.	t.o.	t.o.

computations done by *dom/wdeg*, IBS, and skyline approaches. When solving, bigger instances of the n-queens ($n = \{50, 75\}$) the runtime required by skyline, HH and IBS increases, being *dom/wdeg* the more rapid one. Then, taking into account both instances of magic squares and Sudoku, IBS is again the one exhibiting the minor solving time. However, IBS fails in solving the hardest instance of the knight tournament before the stop criterion, while skyline is the fastest one. Finally, skyline is the strategy requiring the minor (average) runtime for solving the complete set of problems, followed by a short difference by *dom/wdeg*. Charts comparing adaptive strategies in terms of backtracks and solving time are illustrated in Figs. 5 and 6.

Table 6
Runtime in ms for different instances of the N-Queens problem with different strategies.

Problem	Strategy				
	Skyline	HH	*dom/wdeg*	IBS	Random
NQ(n = 8)	109	1379	213	26	6
NQ(n = 10)	109	1443	233	31	83
NQ(n = 12)	124	1468	246	33	30
NQ(n = 15)	312	1658	252	40	373
NQ(n = 20)	280	3288	277	39	5161
NQ(n = 50)	23,041	23,848	354	9588	t.o.
NQ(n = 75)	9486	37,950	487	23,759	t.o.
MS(n = 4)	124	5080	254	36	56
MS(n = 5)	593	9,849,363	7138	42	t.o.
Sudoku	109	1625	456	44	304
Kn(n = 5)	5093	607,599	695	2,091,627	t.o.
Kn(n = 6)	1763	114,906	36,868	t.o.	t.o.
$\overline{X}$	3429	887,467	3956	>193,206	>6007

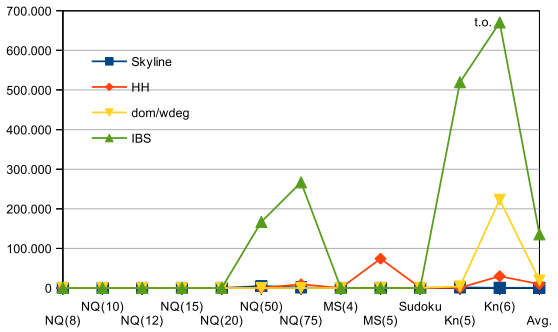

Fig. 5. A skyline example

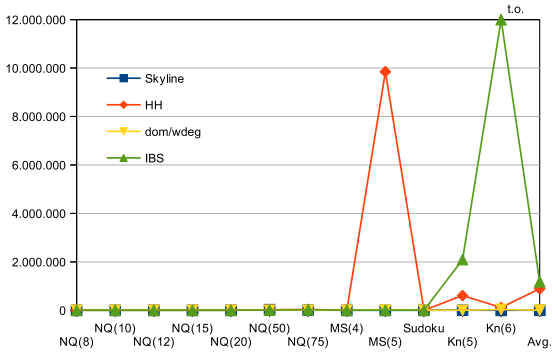

Fig. 6. Skyline of enumeration strategies.

5. Conclusions

In this work, we have proposed an improved AS framework that employs skylines to perform adaptive enumeration processes. The main goal of skylines is to provide a more efficient and lightweight way to rank the enumeration strategies compared to the costly hyperheuristic used in previous approaches. The proposed skyline-based strategy is able to detect efficiently bad decisions concerning enumeration strategies allowing the solver to adapt itself while converging to an efficient strategy for the problem being solved. The experimental results illustrates the effectiveness of the proposal, which combining a portfolio of simple/classic strategies it is able to compete with more sophisticated ones such as the *dom/wdeg*, IBS and a hyperheuristic-based one.

In the future, we plan to incorporate adaptive strategies to the portfolio, but finding the balance to avoid increasing the whole solving. In this way, we will be able to experiment with an interesting adaptive framework which is in turn built of

adaptive components. Another interesting idea is about the design of a similar adaptive framework for interleaving different propagation techniques.

Acknowledgements

Ricardo Soto is supported by Grant CONICYT/FONDECYT/INICIACION/ 11130459, Broderick Crawford is supported by Grant CONICYT/FONDECYT/ 1140897, and Fernando Paredes is supported by Grant CONICYT/FONDECYT/ 1130455.

References

[1] R. Backofen, S. Will, A constraint-based approach to fast and exact structure prediction in three-dimensional protein models, Constraints 11 (2006) 5–30.
[2] Z. Banaszak, M. Zaremba, W. Muszyski, Constraint programming for project-driven manufacturing, Int. J. Prod. Econ. 120 (2009) 463–475 (Special Issue on Introduction to Design and Analysis of Production Systems).
[3] P. Barahona, L. Krippahl, Constraint programming in structural bioinformatics, Constraints 13 (2008) 3–20.
[4] R. Barták, H. Rudová, Limited assignments: a new cutoff strategy for incomplete depth-first search, in: Proceedings of the 20th ACM Symposium on Applied Computing (SAC), 2005, pp. 388–392.
[5] N. Berger, R. Soto, A. Goldsztejn, S. Caro, P. Cardou, Finding the maximal pose error in robotic mechanical systems using constraint programming, in: Proceedings of the 23rd International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE, Lecture Notes in Computer Science, vol. 6096, Springer, 2010, pp. 82–91.
[6] C. Bessiere, B. Zanuttini, C. Fernandez, Measuring search trees, in: Proceedings ECAI'04 Workshop on Modelling and Solving Problems with Constraints, IOS Press, 2004, pp. 31–40.
[7] M. Bofill, J. Espasa, M. Garcia, M. Palahí, J. Suy, M. Villaret, Scheduling B2B meetings, in: Proceedings of 20th International Conference on Principles and Practice of Constraint Programming (CP), Lecture Notes in Computer Science, vol. 8656, Springer, 2014, pp. 781–796.
[8] S. Börzsönyi, D. Kossmann, K. Stocker, The skyline operator, in: Proceedings of the 17th International Conference on Data Engineering (ICDE), IEEE Computer Society, 2001, pp. 421–430.
[9] F. Boussemart, F. Hemery, C. Lecoutre, L. Sais, Boosting systematic search by weighting constraints, in: Proceedings of the 16th European Conference on Artificial Intelligence (ECAI), IOS Press, 2004, pp. 146–150.
[10] Q.T. Bui, Q.D. Pham, Y. Deville, Solving the agricultural land allocation problem by constraint-based local search, in: Proceedings of 20th International Conference on Principles and Practice of Constraint Programming (CP), Lecture Notes in Computer Science, vol. 8124, Springer, 2013, pp. 749–757.
[11] C. Castro, E. Monfroy, C. Figueroa, R. Meneses, An approach for dynamic split strategies in constraint solving, in: Proceedings of the 4th Mexican International Conference on Artificial Intelligence (MICAI), Lecture Notes in Computer Science, vol. 3789, Springer, 2005, pp. 162–174.
[12] J. Chomicki, P. Godfrey, J. Gryz, D. Liang, Skyline with presorting, in: Proceedings of the 19th International Conference on Data Engineering (ICDE), IEEE Computer Society, 2003, pp. 717–816.
[13] L. Climent, M.A. Salido, F. Barber, Robustness in dynamic constraint satisfaction problems, Int. J. Innovat. Comput. Inform. Control 8 (2012) 2513–2532.
[14] B. Crawford, C. Castro, E. Monfroy, R. Soto, W. Palma, F. Paredes, Dynamic selection of enumeration strategies for solving constraint satisfaction problems, Rom. J. Inf. Sci. Tech. 15 (2012) 106–128.
[15] B. Crawford, R. Soto, C. Castro, E. Monfroy, A hyperheuristic approach for dynamic enumeration strategy selection in constraint satisfaction, in: of the 4th International Work-conference on the Interplay Between Natural and Artificial Computation (IWINAC), Lecture Notes in Computer Science, vol. 6687, Springer, 2011, pp. 295–304.
[16] B. Crawford, R. Soto, E. Monfroy, W. Palma, C. Castro, F. Paredes, Parameter tuning of a choice-function based hyperheuristic using particle swarm optimization, Expert Syst. Appl. 40 (2013) 1690–1695.
[17] B. Crawford, R. Soto, M. Montecinos, C. Castro, E. Monfroy, A framework for autonomous search in the eclipse solver, in: Proceedings of the 24th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE), Lecture Notes in Computer Science, vol. 6703, Springer, 2011, pp. 79–84.
[18] D. Dubois, H. Fargier, H. Prade, The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction, in: Proceedings of Second IEEE International Conference on Fuzzy Systems (FUZZ), IEEE, 1993, pp. 1131–1136.
[19] S. Epstein, S. Petrovic, Learning to solve constraint problems, in: Proceedings of the Workshop on Planning and Learning (ICAPS).
[20] S.L. Epstein, E.C. Freuder, R.J. Wallace, A. Morozov, B. Samuels, The adaptive constraint engine, in: Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP), Lecture Notes in Computer Science, vol. 2470, Springer, 2002, pp. 525–542.
[21] F. He, R. Qu, A constraint programming based column generation approach to nurse rostering problems, Comput. OR 39 (2012) 3331–3343.
[22] P. Godfrey, R. Shipley, J. Gryz, Maximal vector computation in large data sets, in: Proceedings of the 31st International Conference on Very Large Data Bases (VLDB), ACM, 2005, pp. 229–240.
[23] Y. Hamadi, E. Monfroy, F. Saubion, Autonomous Search, Springer, 2012.
[24] H.T. Kung, F. Luccio, F.P. Preparata, On finding the maxima of a set of vectors, J. ACM 22 (1975) 469–476.
[25] J. Larrosa, P. Meseguer, Partition-based lower bound for max-CSP, Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP, vol. 1713, Springer, 1999, pp. 303–315.
[26] J. Larrosa, T. Schiex, Solving weighted CSP by maintaining arc consistency, Artif. Intell. 159 (2004) 1–26.
[27] J.E. Lee, K.D. Lee, Modeling and optimization of closed-loop supply chain considering order or next arrival of goods, Int. J. Innovat. Comput. Inform. Control 9 (2013) 3639–3654.
[28] J. Maturana, F. Saubion, A compass to guide genetic algorithms, in: G. Rudolph, T. Jansen, S.M. Lucas, C. Poloni, N. Beume (Eds.), PPSN, Lecture Notes in Computer Science, vol. 5199, Springer, 2008, pp. 256–265.
[29] J.P. Métivier, P. Boizumault, S. Loudni, Solving nurse rostering problems using soft global constraints, in: Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming (CP), Lecture Notes in Computer Science, vol. 5732, Springer, 2009, pp. 73–87.
[30] E. Monfroy, C. Castro, B. Crawford, R. Soto, F. Paredes, C. Figueroa, A reactive and hybrid constraint solver, J. Exp. Theor. Artif. Intell 25 (2013) 1–22.
[31] P. Refalo, Impact-based search strategies for constraint programming, in: Proceedings of Principles and Practice of Constraint Programming (CP), Lecture Notes in Computer Science, vol. 3258, Springer, 2004, pp. 557–571.
[32] J.C. Régin, A filtering algorithm for constraints of difference in CSPs, in: Proceedings of the 12th National Conference on Artificial Intelligence (AAAI), 1994, pp. 362–367.
[33] L. Rodrigues, L. Magatão, Enhancing supply chain decisions using constraint programming: a case study, in: Proceedings of the 6th Mexican International Conference on Artificial Intelligence (MICAI), Lecture Notes in Computer Science, Springer, 2007, pp. 1110–1121.
[34] D. Sabin, E.C. Freuder, Configuration as composite constraint satisfaction, in: Proceedings of Artificial Intelligence and Manufacturing, Research Planning Workshop, AAAI Press, 1996, pp. 153–161.
[35] P. Schaus, J.C. Régin, R.V. Schaeren, W. Dullaert, B. Raa, Cardinality reasoning for bin-packing constraint: application to a tank allocation problem, in: Proceedings of 18th International Conference on Principles and Practice of Constraint Programming (CP), Lecture Notes in Computer Science, vol. 7514, Springer, 2012, pp. 815–822.

[36] T. Serra, G. Nishioka, F. Marcellino, The offshore resources scheduling problem: detailing a constraint programming approach, in: Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP), Lecture Notes in Computer Science, vol. 7514, Springer, 2012, pp. 823–839.
[37] R. Soto, B. Crawford, E. Monfroy, V. Bustos, Using autonomous search for generating good enumeration strategy blends in constraint programming, in: Proceedings of the 12th International Conference on Computational Science and Its Applications (ICCSA), Lecture Notes in Computer Science, vol. 7335, Springer, 2012, pp. 607–617.
[38] R. Soto, H. Kjellerstrand, O. Durán, B. Crawford, E. Monfroy, F. Paredes, Cell formation in group technology using constraint programming and Boolean satisfiability, Expert Syst. Appl. 39 (2012) 11423–11427.
[39] R. Stansbury, A. Agah, A robot decision making framework using constraint programming, Artif. Intell. Rev. 38 (2012) 67–83.
[40] Y. Xu, D. Stern, H. Samulowitz, Learning adaptation to solve constraint satisfaction problems, in: Proceedings of the 3rd International Conference on Learning and Intelligent Optimization (LION), 2009, pp. 507–523.