Expert Systems with Applications 40 (2013) 5817–5821

Contents lists available at SciVerse ScienceDirect

Expert Systems with Applications

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

A hybrid AC3-tabu search algorithm for solving Sudoku puzzles

Ricardo Soto [a,b,*], Broderick Crawford [a,c], Cristian Galleguillos [a], Eric Monfroy [d], Fernando Paredes [e]

[a] Pontificia Universidad Católica de Valparaíso, Av. Brasil 2950, Valparaíso, Chile
[b] Universidad Autónoma de Chile, Av. Pedro de Valdivia 641, Santiago, Chile
[c] Universidad Finis Terrae, Av. Pedro de Valdivia 1509, Santiago, Chile
[d] CNRS, LINA, University of Nantes, 2 rue de la Houssinière, Nantes, France
[e] Escuela de Ingeniería Industrial, Universidad Diego Portales, Manuel Rodríguez Sur 415, Santiago, Chile

ARTICLE INFO

Keywords:
Metaheuristics
Tabu search
Constraint satisfaction
Sudoku

ABSTRACT

The Sudoku problem consists in filling a $n^2 \times n^2$ grid so that each column, row and each one of the $n \times n$ sub-grids contain different digits from 1 to n^2. This is a non-trivial problem, known to be NP-complete. The literature reports different incomplete search methods devoted to tackle this problem, genetic computing being the one exhibiting the best results. In this paper, we propose a new hybrid AC3-tabu search algorithm for Sudoku problems. We merge a classic tabu search procedure with an arc-consistency 3 (AC3) algorithm in order to effectively reduce the combinatorial space. The role of AC3 here is do not only acting as a single pre-processing phase, but as a fully integrated procedure that applies at every iteration of the tabu search. This integration leads to a more effective domain filtering and therefore to a faster resolution process. We illustrate experimental evaluations where our approach outperforms the best results reported by using incomplete search methods.

1. Introduction

The Sudoku puzzle consists in a board commonly of size 9×9, subdivided into sub-grids of size 3×3 including pre-filled cells with digits that cannot be changed or moved (see Fig. 1). The puzzle is solved when the board is filled so that each row, column and sub-grid contain different digits from 1 to 9. The general Sudoku problem of $n^2 \times n^2$ board size including $n \times n$ sub-grids is known to be NP-complete (Yato, Seta, & Ito, 2003). Then exact methods may solve the problem in exponential time. Sudokus are often classified in terms of difficulty. The relevance and positioning of the problem may vary its difficulty, however the number of pre-filled cells has little or no incidence. Mantere and Koljonen (2007) classify Sudokus into tree categories: easy, medium, and hard.

Various approaches have been proposed during the last years to solve Sudoku puzzles. Most works range from complete search methods such as constraint programming (Moon & Gunther, 2006; Rossi, van Beek, & Walsh, 2006; Simonis, 2005) and Boolean satisfiability (Lynce & Ouaknine, 2006) to incomplete search methods such as genetic programming (Asif, 2009; Mantere & Koljonen, 2007) and metaheuristics in general (Moraglio, Togelius, & Lucas, 2006; Lewis, 2007; Moraglio & Togelius, 2007; Mantere & Koljonen, 2008a). Other less traditional techniques in this context such as rewriting rules (Santos-García & Palomino, 2007), Sinkhorn balancing (Moon, Gunther, & Kupin, 2009) and entropy minimization (Gunther & Moon, 2012) have also been proposed to tackle this problem.

In this paper we focus on incomplete search methods. We propose a new hybrid algorithm for Sudoku puzzles by combining a classic tabu search with an arc-consistency 3 (AC3) algorithm that acts as a domain reducer. The idea is to apply the AC3 procedure as a pre-processing phase but also at each iteration of the tabu search in order to actively filter the domains. This integration clearly reduce the number of tabu search iterations speeding up the solving process. We illustrate experimental results where our approach outperforms better than the incomplete methods reported in the literature.

This paper is structured as follows. The related work is presented in Section 2 followed by the preliminaries. The new hybrid AC3-tabu search algorithm for Sudokus is described in Section 4. Experiments are illustrated in Section 5. Finally, we conclude and give some directions for future work.

2. Related work

The literature presents several techniques for solving, rating and generating Sudoku problems. Sudoku problems can definitely be solved by using brute-force algorithms, backtracking-like procedures or complete search methods in general (Crawford, Aranda, Castro, & Monfroy, 2008; Lynce & Ouaknine, 2006; Moon & Gunther, 2006; Simonis, 2005). In this paper, we focus on incomplete search methods to solve Sudoku puzzles, in particular hard instances of such a problem. In this context, different approaches

* Corresponding author at: Pontificia Universidad Católica de Valparaíso, Chile. Tel.: +56 32 2273659.
E-mail address: ricardo.soto@ucv.cl (R. Soto).

0957-4174/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.eswa.2013.05.019

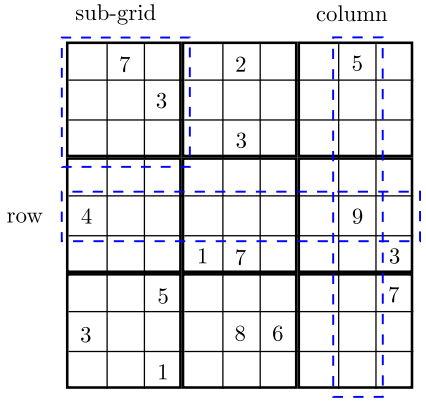

Fig. 1. Sudoku puzzle instance.

has been reported. For instance, Asif (2009) proposes an ant colony optimization algorithm for solving Sudoku problems. The author employs as heuristic information the number of digits correctly placed on the board. However, the best value reached is 76, 81 being the global optimum. In Moraglio and Togelius (2007) a geometric particle swarm optimization (GPSO) algorithm is proposed. Their goal was rather to validate the use of GPSO for non-trivial combinatorial spaces than the performance of results. Indeed they achieve a 72% of success; for 36 out of 50 tries the global optimum was reached.

Hill-climbers have also been tested to solve Sudoku puzzles (Moraglio et al., 2006), they are able to succeed for easy Sudokus failing for medium and hard ones. A genetic algorithm (GA) for Sudoku puzzles is illustrated in Moraglio et al. (2006) as well. The behaviour of geometric crossovers is studied, in particular Hamming space crossovers and swap space crossovers. Both approaches performs better than hill-climbers and mutations alone. But, they are not able to solve medium Sudokus; and only by using the swap space crossover, 15 out of 30 hard Sudokus are solved. In Mantere and Koljonen (2007) another GA approach is proposed. Here, different categories of Sudoku are solved: easy, medium, and hard. The algorithm is an extension of one devoted to solve magic square problems. Good results are exhibited for solving easy and medium Sudokus, being able to solve 2 out of 30 hard Sudokus. A similar approach using cultural algorithms is proposed in Mantere and Koljonen (2008a), but is in general superseded by the GA previously reported.

Lewis (2007) presents a simulated annealing algorithm for Sudokus. The idea is to model the puzzle as an optimization problem where the goal is to minimize the number of incorrectly placed digits on the board. However, the approach is mostly centered on creating solvable problem instances than solving hard Sudoku puzzles.

3. Preliminaries

3.1. Tabu search

Tabu search (TS), introduced by Glover, is a metaheuristic especially devoted to solve combinatorial optimization problems. It has successfully been used for tackling different kind of real-life problems as well as well-known problems from academic literature such as the travelling salesman problem, the knapsack problem, the quadratic assignment problem, or the timetabling problem.

The core idea of TS relies in employing a local search procedure that allows to iteratively move from one potential solution to another promising one until some stop criterion has been reached. This procedure is complemented with a memory structure called tabu list, which is perhaps a main feature that distinguishes TS from many incomplete methods. The goal of this memory structure is twofold: (1) to help the TS to escape from poor-scoring areas, (2) and to avoid returning to recent visited states.

Algorithm 1 depicts the classic procedure of tabu search for minimization. As input, it receives the size of the tabu list and as output it returns the best solution reached. Then, an initial solution is created, which is commonly chosen at random. At line 3, a while loop manages the iterations of the process until a given stop condition is met. Some examples of stop condition are a number of iterations limit or a threshold on the solution cost. At line 7, the neighboring solutions are added to the candidate list only if they do not contain elements on the tabu list. Then, a potential best candidate is selected, which commonly corresponds to the best quality solution according to the cost. At line 11, the cost of the selected candidate is evaluated. If it is better than the one of S_{best}, its features are added to the tabu list and the candidate becomes the new S_{best}. Finally, some elements are allowed to expire from the tabu list, generally in the same order they were added.

Algorithm 1 – Tabu search

```
Input: TabuList_size
Output: S_best
1  S_best ← ConstructInitialSolution ()
2  TabuList ← 0
3  While ¬ StopCondition do
4    CandidateList ← 0
5    For (S_candidate ∈ S_best_neighboorhood ) do
6      If ¬ ContainsAnyFeatures(S_candidate,TabuList)
7        CandidateList← S_candidate + CandidateList
8      End If
9    End For
10   S_candidate ← LocateBestCandidate (CandidateList)
11   If cost(S_candidate) ⩽cost(S_best)
12       TabuList ← FeatureDifferences(S_candidate, S_best)
13       S_best ← S_candidate
14       While TabuList > TabuList_size do
15         ExpireFeature(TabuList)
16       End While
17     End If
18   End While
19   Return S_best
```

3.2. Arc consistency

As illustrated by Simonis (2005), Sudokus can be represented as constraint networks and as a consequence techniques from constraint satisfaction can be applied over them. Arc-consistency is one of the most used filtering techniques in constraint satisfaction for reducing the combinatorial space of problems. Arc-consistency is formally defined as a local consistency within the constraint programming field (Rossi et al., 2006). A local consistency defines properties that the constraint problem must satisfy after constraint propagation. Constraint propagation is simply the process when the given local consistency is enforced to the problem. In the following, some necessary definitions are stated (Bessière, 2006).

Definition 1 (*Constraint*). A constraint c is a relation defined on a sequence of variables $X(c) = (x_{i_1}, \ldots, x_{i_{|X(c)|}})$, called the scheme of c. c is the subset of $\mathbb{Z}^{|X(c)|}$ that contains the combinations of values (or tuples) $\tau \in \mathbb{Z}^{|X(c)|}$ that satisfy c. $|X(c)|$ is called the arity of c. A constraint c with scheme $X(c) = (x_1, \ldots, x_k)$ is also noted as $c(x_1, \ldots, x_k)$.