ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume **17**, Number 2, 2014, 204–216

Sequentially Dependent Meta-Constraint Satisfaction Problem: An Application to Video Games

Ricardo SOTO[1,2,3], Broderick CRAWFORD[1,4,5], Eric MONFROY[6], Fernando PAREDES[7]

[1] Pontificia Universidad Católica de Valparaíso, Chile
[2] Universidad Autónoma de Chile, Chile
[3]Universidad Central de Chile, Chile
[4]Universidad San Sebastián, Chile
[5]Universidad Finis Terrae, Chile
[6]CNRS, LINA, Université de Nantes, France
[7]Escuela de Ingeniería Industrial, Universidad Diego Portales, Chile

Abstract. A Constraint Satisfaction Problem (CSP) consists in a sequence of variables holding a domain of possible values and relations among these variables called constraints. A meta-CSP can be seen as a metaproblem whose decomposition leads to a set of CSPs. The meta-variables correspond to sub-problems of the original problem, and a meta-constraint is a relation among those meta-variables. Meta-CSPs find many applications in industry, usually in processes that involve time and actions such as the control of a robot, a manufacturing process, or the scheduling of any common activity. In this paper, we introduce the notion of Sequentially Dependent Meta-CSP (SD Meta-CSP), which extends the meta-CSP in order to support applications where a dependency between sub-problems is mandatory. In this case, the meta-CSP is decomposed into a set of sub-problems $\{P_i, P_{i+1}, \ldots, P_n\}$, but the instance of the sub-problem P_{i+1} sequentially depends on the solution of the sub-problem P_i. In this work we provide a formal definition for the SD Meta-CSPs, a framework to handle it, and we illustrate its applicability to video games. In particular, we model and implement agents as SD Meta-CSPs able to autonomously play two classic games: Ms. Pac-Man and Super Mario Bros.

Key-words: Video-games, artificial intelligence, constraint satisfaction problem.

1. Introduction

A Constraint Satisfaction Problem (CSP) consists in a sequence of variables holding a domain of possible values and relations among these variables which are called constraints. A solution to a CSP is a complete value-variable assignment that satisfies the set of constraints. In his seminal work [11], Freuder introduces the notion of meta-CSP, which can be regarded as a meta-problem composed of sub-problems.

A meta-CSP owns meta-variables and meta-constraints. The meta-variables correspond to sub-problems of the original problem, and a meta-constraint is a relation among those meta-variables. For instance, consider two meta-variables corresponding to sub-problems A and B. A meta-constraint between those meta-variables enforces all the constraints in the original problem that involve one variable from A and one from B. Finally, the set of meta-values for a meta-variable is the set of solutions to the sub-problem. A meta-CSP finds different applications in real-life, certainly one of the most important is related to systems that reasons about time and actions, such as a manufacturing process, the movement of a robot, or the scheduling of any activity [41, 30]. In this context, time and actions are modeled as a Temporal CSP, which is seen as a meta-CSP composed of a set of simple temporal problems [4].

In this paper, we extend the notion of meta-CSP in order to support a new kind of application where a dependency among sub-problems is mandatory. As an example, let us consider the well-known Ms. Pac-Man video game, which consists in driving an avatar through a maze, by eating pills and avoiding to be attacked by ghosts (see Fig. 1).

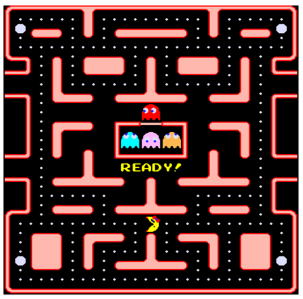

Fig. 1. Ms. Pac-Man screenshot.

In this game, each move of Ms. Pac-Man can be modeled as a CSP, where the variables represent its possible movements, and the constraints restrict the parts of the space that must avoided in order to succeed. Then, the meta-CSP put into group those CSPs models where an agent defines and controls the set of moves needed to pass a given level. However, the components of the CSP for computing move $i + 1$ strictly depends on the position of Ms. Pac-Man in move i, which has been computed by the preceding CSP. In this case, the meta-CSP contains a set of sub-problems $\{P_i, P_{i+1}, \ldots, P_n\}$, where the instance of the sub-problem P_{i+1} sequentially depends on the solution of the sub-problem P_i, we refer to this model as Sequentially Dependent Meta-CSP (SD Meta-CSP).

This work provides a formal definition for SD Meta-CSPs which is a framework to handle it, and we illustrate its applicability to video games. In particular, we use the SD Meta-CSP framework to design and implement agents able to autonomously play two well-known video games: Ms. Pac-Man and Mario Bros.The first one is a widely used game for testing artificial intelligence (AI) and computational intelligence (CI) algorithms [31, 3, 37, 1, 29].This video game appears to be relatively simple; however it demands complex and intelligent strategies to become a superb player. The second one is also a broadly used video game for testing AI/CI methods [36, 34, 35, 16, 28], especially for reinforcement learning and game AI techniques. In addition, both video games arise as excellent candidates for validating the applicability of the proposed approach.

The remainder of this paper is structured as follows: in Section 2 we introduce the related work, followed by the preliminaries in Section 3. The SD Meta-CSP is presented in Section 4 followed by Section 5 which shows the application of this new framework to video games. Finally, we conclude and give some directions for future work.

2. Related Work

A CSP is a formal representation of a constraint-based problem that has been subjected to a large number of extensions. A preliminary example is the work proposed by [24], who introduced the notion of dynamic CSP, also known as conditional CSP. This framework distinguishes from the standard CSP essentially by the use of variables that may change their status to active or non-active. The status of variables is controlled by the so-called activity constraints. [30] call composite CSP to a combination of meta-CSPs, hierarchical domain CSP [22, 17], and dynamic CSP. In this framework a variable can represent a sub-problem, and a domain can be composed of a set of variables (instead of values as in the classic CSP), which can be hierarchically organized. Finally, the activation of variables is controlled by activity constraints as in dynamic CSPs. Recently, a framework that mixes composite, conditional, and temporal CSPs has been proposed by [26]. There is also another extension proposed by the same authors which introduces the notion of preferences to dynamic CSPs [27].

The max-CSP is a useful CSP extension for handling over-constrained problems. In this context, the idea is to find an assignment that minimizes the number of violated constraints [13, 19, 18, 38]. There are various frameworks which derived from max-CSP have been suggested. For instance in weighted CSPs [20], a weight is associated to each constraint to express violation priorities, where constraints with lower weights are preferable to violate. Then, the idea is to minimize the total sum of weights from violated constraints. A very similar approach named possibilistic CSPs is proposed in [32]. Here, a preference level between 0 and 1 is attached to constraints, 1 being the maximum level of importance. In this framework, an idempotent operator `max` is used to compute the solution cost instead of a strict monotonic operator such as '+'. In fuzzy CSPs [10], also a preference level between 0 and 1 is considered, however it is associated to each tuple of each constraint. The semiring CSP [2] and the valued

CSP [33, 5], are more general frameworks that encapsulate the notion of most max-CSP extensions, *i.e.*, a value is associated to constraints or tuples and the set of valuations is introduced within an objective function.

Lexicographic ordered CSP is another CSP extension [14, 12] allowing the use preferred variables. This approach has also been adapted to support conditional preferences under the name of conditional lexicographic CSP [39]. In [23], the dynamic flexible CSP is proposed,its goal is to consider soft constraints (constraints that can be violated) in a dynamic environment. In mixed CSP [15], the idea is to support both mixed (CSP involving numeric and discrete variables) and conditional CSPs in a single framework. Finally, in stochastic CSPs [40], domains are linked to a probability distribution.

As illustrated, the research work has conducted to several adaptations of the classic CSP. However, to the best of our knowledge, no published work addresses meta-CSP where dependency among sub-problems is mandatory.

3. CSP Background

A CSP is mainly composed of a sequence of variables holding a domain of possible values and a set of constraints over those variables. Formally, a CSP $\mathcal{P}$ is defined by a triplet $\mathcal{P} = \langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$ where:

- $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ is the set of variables.
- $\mathcal{D} = \{d_{v_i} \mid v_i \in \mathcal{V}\}$ is the set of domains and $d_{v_i} = \{a_{i_1}, a_{i_2}, \ldots, a_{i_t}\}$ represents the set of values that variable v_i can take.
- $\mathcal{C} = \{C_R \mid R \subseteq V, R \neq \emptyset\}$ is the set of constraints. C_R is a constraint over variables in R and it is defined as a subset of the Cartesian product of domains of variables in R.

A solution to a CSP is an assignment $\{v_1 \rightarrow a_1, \ldots, v_n \rightarrow a_n \mid a_i \in d_{v_i}, i \in 1..n\}$ that satisfies the whole set of constraints. Moreover, the set of solutions of a CSP $\mathcal{P}$ is usually denoted by $sol(\mathcal{P})$.

Constraint Satisfaction is in general NP-Complete and the most used solving approach is to combine a backtracking algorithm with constraint propagation. In addition , a Constraint propagation aims at pruning values that do not lead to any solution during search, which is possible by enforcing a given local consistency to the problem such as the well-known arc-consistency [21].

Algorithm 1 - SolveCSP
Input: a CSP $\mathcal{P} = \langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$
Output: Either a solution, or a notification that $\mathcal{P}$ is inconsistent
1 $\mathcal{D}'_i \leftarrow \mathcal{D}_i \ \forall i \in \{1, n\}$
2 $i \leftarrow 1$
3 **while** $1 \leq i \leq n$
4 $propagate(i, \mathcal{D}', failure)$

```
5       if failure then
6         i ← i − 1
7         reset(i, D′)
8       else
9         evaluate(v_i)
10        i ← i + 1
11      end if
12    end while
13    if i = 0 then
14      return inconsistent
15    else
16      return {a_1, ..., a_n} ∈ sol(P)
17    end if
```

Algorithm 1 illustrates a general procedure for solving CSPs, where a backtracking procedure has been merged with constraint propagation [7]. The algorithm receives as input the CSP and returns a solution for it or informs that the CSP is inconsistent. At the beginning, the procedure is initialized by performing a copy of all domains from $\mathcal{D}$ that allows propagation and reset actions. Then, a while loop encloses a set of actions to be performed until the i counter reaches n, n being the number of variables. The first action is responsible for constraint propagation in order to eliminate from $\mathcal{D}'$ the inconsistent values. Then, if a failure occurs, the procedure backtracks and reset each $\mathcal{D}'_k,\ k > i$, to its value before v_i was last instantiated. Otherwise, the variable v_i is evaluated and the algorithm moves forward in the search space. Finally, the procedure returns the solution of the CSP if found, or informs that the CSP is inconsistent.

4. SD Meta-CSP

A SD Meta-CSP can be defined analogously than traditional CSPs, by using a triplet $\mathcal{P} = \langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$, where variables represent an entire sub-problem $\mathcal{P}' = \langle \mathcal{V}', \mathcal{D}', \mathcal{C}' \rangle$, and the problem $\mathcal{P}'_i$ sequentially depends on $\mathcal{P}'_{i-1}$. Hence, formally, the SD Meta-CSP is composed of:

- $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ is the set of metavariables, where v_i represents the sub-problem $\mathcal{P}'_i$.
- $\mathcal{D} = \{d_{v_i} \mid v_i \in V\}$ is the set of domains and $d_{v_i} = \{a_{i_1}, a_{i_2}, \ldots, a_{i_t}\}$ represents the set of meta-values that meta-variable v_i can take. Note that d_{v_i} represents the set of solutions of $\mathcal{P}'_i$. Then, if a_i is the solution of $\mathcal{P}'_i$ we can denote the dependency among sub-problems as a function $f : a_i \rightarrow v_{i+1}$, $\forall i \in \{1, \ldots, n-1\}$, that takes as input the solution of $\mathcal{P}'_i$ to generate as output $\mathcal{P}'_{i+1}$
- $\mathcal{C} = \{C_R \mid R \subseteq V, R \neq \emptyset\}$ is the set of constraints.

A solution to a SD Meta-CSP is an assignment $\{x_1 \rightarrow a_1, \ldots, v_n \rightarrow a_n \mid a_i \in d_{v_i}, i \in 1..n\}$ that satisfies the whole set of meta-constraints.

Algorithm 2 describes the proposed procedure for solving SD Meta-CSP. The idea is to decompose the SD Meta-CSP and solving each sub-problem in a sequential scheme (line 1). Then, the solution $\{a_1, \ldots, a_n\} \in sol(\mathcal{P}')$ of sub-problem $\mathcal{P}'$ is an input of the next sub-problem. We call this input $inst_{P'}$ and represents the initial configuration of the sub-problem. This can be exemplified by recalling the Ms. Pac-Man game introduced in Section 1. Here, $inst_{\mathcal{P}'_i}$ holds the current position of Ms. Pac-Man within the maze and the position of ghosts to be used as input of sub-problem $\mathcal{P}'_i$ in order to compute its solution. Then, the solution of $\mathcal{P}'_i$ is assigned to $inst_{\mathcal{P}'_{i+1}}$ which will be in turn the input of $\mathcal{P}'_{i+1}$ (line 2). The process is done repeatedly until no sub-problem remains. Let us note that if a sub-problem has no solution, the SD Meta-CSP also becomes inconsistent.

Algorithm 2 - SolveSDMeta-CSP
Input:a CSP $\mathcal{P} = \langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$
Output: Either a solution, or a notification that $\mathcal{P}$ is inconsistent
1 **foreach** $\mathcal{P}' \in \mathcal{P}$
2 $inst_{P'} \leftarrow$ solveSubCSP($inst_{P'}, \mathcal{P}'$)
3 **if** P' *is inconsistent* **then**
4 **return** *inconsistent*
5 **end if**
4 **end foreach**

Algorithm 3 - SolveSubCSP
Input: $inst_{\mathcal{P}'}$, a CSP $\mathcal{P}' = \langle \mathcal{V}', \mathcal{D}', \mathcal{C}' \rangle$
Output: Either a solution, or a notification that $\mathcal{P}'$ is inconsistent
1 $\mathcal{D}''_i \leftarrow \mathcal{D}'_i \ \forall i \in \{1, n\}$
2 $i \leftarrow 1$
3 **while** $i \leq 1 \leq n$
4 $propagate(i, \mathcal{D}''_i, failure, inst_{\mathcal{P}'})$
5 **if** $failure$ **then**
6 $i \leftarrow i - 1$
7 $reset(i, \mathcal{D}''_i)$
8 **else**
9 $evaluate(v_i, inst_{\mathcal{P}'})$
10 $i \leftarrow i + 1$
11 **end if**
12 **end while**
13 **if** $i = 0$ **then**
14 **return** *inconsistent*
15 **else**
16 **return** $\{a_1, \ldots, a_n\} \in sol(\mathcal{P}')$
17 **end if**

Then, a slight modification has been done to Algorithm 1 to be used by SolveSD-Meta-CSP (see Algorithm 3), the $inst_{\mathcal{P}'}$ input must be used in the propagation (line 4)

and evaluation (line 9). Indeed $inst_{\mathcal{P}'}$ may fix some variables and/or to affect some constraints.

5. Modeling Video Games as SD Meta-CSPs

The design of agents for simulating the human behavior in video games has become an interesting research area receiving much attention during the last five years [31, 3, 37, 1, 29, 36, 34, 35, 16, 28] . Indeed, agents are known to be useful from two main standpoints. For instance, from a gaming perspective, agents can be used to guide players when they get stuck in a particular level of the game, to implement demo play features, to automatically test new level and features, or to create new content for the game. From an AI perspective, the design of video game agents is known to be a challenging task that may demand complex AI algorithm to generate agents able to success in video games. In this section, we illustrate how two video game agents can be modeled as SD Meta-CSPs and implemented using constraint programming solvers.

5.1. Modeling Ms. PacMan

Ms. Pac-Man is an arcade game originally produced by Namco, whose goal is to drive Ms. Pac-Man through a maze by eating pills and avoiding to be attacked by ghosts. The player is taken to the next level once all pills of the maze are eaten. Ghosts can be eaten by a limited period of time when Ms. Pac-Man eat a power pill. From a design agent standpoint, Ms. Pac-Man is considered much more complex that the classic Pac-Man, since the behavior of ghost is non-deterministic, being not possible to learn optimal routes.

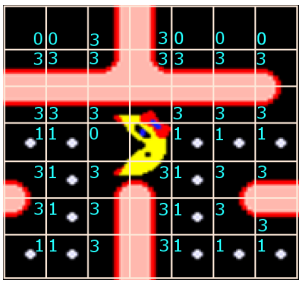

Fig. 2. An extract of the Ms. Pac-Man observation matrix.

The Ms. Pac-Man API provides the information of the maze through a two dimensional array called observation matrix. This matrix describes the elements around Ms. Pac-Man where each array cell represents a block of the game (see Fig. 2). In this matrix, 0 represents a passable block with no pill, 1 a passable block with pill, 2 a passable block with power pill, 3 an impassable block (wall), and 4 represents a block with a ghost. The position of Ms. Pac-Man is provided via x and y coordinates.In

addition, it is always possible to know the distance from Ms. Pac-Man to its enemies and to impassable blocks in order to allow the agent to take the correct decision. Then, by employing the observation matrix as the $inst_{\mathcal{P}'}$ parameter, the agent for Ms. Pac-Man modeled as a SD Meta-SP is defined in the following.

- The SD Meta-CSP $\mathcal{P} = \langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$ represents the set of moves allowing Ms. Pac-Man to successfully pass a given level.
- $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ is the set of meta-variables, where v_i represents the sub-problem $\mathcal{P}'_i$, which models the move i of Ms. Pac-Man.
- $\mathcal{D} = \{d_{v_i} \mid v_i \in V\}$ is the set of domains and $d_{v_i} = \{a_{i_1}, a_{i_2}, \ldots, a_{i_t}\}$ represents the set of solutions for move i of Ms. Pac-Man. Then, the problem $\mathcal{P}'_{i+1}$ representing the move $i+1$ is generated via the function $f : a_i \rightarrow v_{i+1}$, $\forall i \in \{1, \ldots, n-1\}$ where a_i is the solution of $\mathcal{P}'_i$.
- $\mathcal{C} = \{C_R \mid R \subseteq V, R \neq \emptyset\}$ is the set of constraints.

Hence, a sub-problem $\mathcal{P}' = \langle \mathcal{V}', \mathcal{D}', \mathcal{C}' \rangle$ representing a single move of Ms. Pac-Man is defined as follows.

- $\mathcal{V}' = \{left, right, up, down\}$, are Boolean variables that define the next move of Ms. Pac-Man. For instance is $left$ adopts 1 as value, Ms. Pac-Man moves to the left. The behavior for variables $right, up$, and $down$ is analogous.
- $\mathcal{D}' = \{0, 1\}\ \forall\ v_i \in V$
- $\mathcal{C}'$ is the set of constraints, where each C_R describes a possible move of Ms. Pac-Man, which can be generically defined as follows.

$$(v = 1) \Longleftrightarrow \diamond_{w=1}^{n} (block_{x,y} = val)_w$$

where v is a variable from $\mathcal{V}'$, $\diamond \in \{\wedge, \vee\}$, n is the number of constraints that must be satisfied within the conjunction/disjunction for allowing the move of Ms. Pac-Man, *block* is a given block placed on the position (x, y) of the space, and *val* represents a possible value for this block {0: passable block with no pill, 1: a passable block with pill, 2: a passable block with power pill, 3: an impassable block (wall), and 4: a block with a ghost}. For instance, assuming that Ms. Pac-Man is placed in the position (x, y) of the space, the constraint $(right = 1) \Longleftrightarrow (block_{x+1,y} = 0 \ \vee \ block_{x+1,y} = 1 \ \vee \ block_{x+1,y} = 2)$ ensures that Ms. Pac-Man moves right iff there is a passable block by walking in his next right block.

5.2. Modeling the Mario AI Benchmark

The Mario AI Benchmark is a modified and open source version of the famous video game Super Mario Bros from Nintendo (see Fig. 3). It has been adapted from

the original game to allow the integration of AI algorithms mainly for agent design. The game consist in moving Mario, which is an avatar with humanoid form, through a two-dimensional environment avoiding obstacles such as enemies and holes. Mario can walk, run and jump to the left and to the right. There are also three states for Mario: small, big, and fire. The big and the fire states allow Mario to kill enemies by jumping over them and by shooting fireballs, respectively.

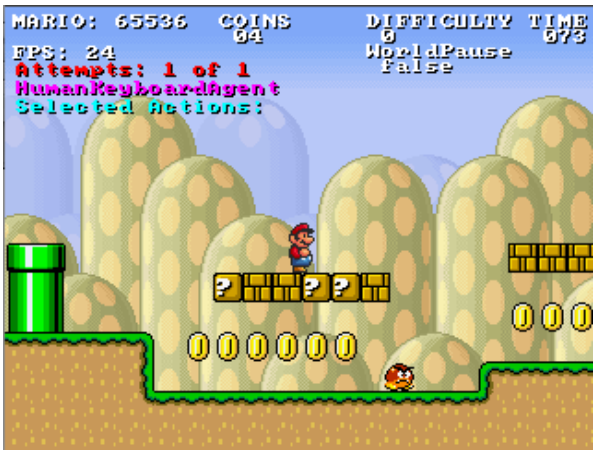

Fig. 3. Mario AI benchmark screenshot.

The Mario AI benchmark API also provides information about the environment of Mario. As in Ms. Pac-Man, the observation matrix describes the elements around Mario analogously (0 represents an impassable block, 1 a passable block, 2 a coin, and 3 an enemy). The position of Mario via x and y coordinates is also used. Then, the corresponding SD Meta-CSP is equivalent to the one of Ms. Pac-Man, and the sub-problem $\mathcal{P}' = \langle \mathcal{V}', \mathcal{D}', \mathcal{C}' \rangle$ representing a single move of Mario is defined as follows.

- $\mathcal{V}' = \{left, right, jump, sAct\}$, are Boolean variables that define the next move of Mario. $sAct$ represents an special action, for instance if $sAct = 1$ Mario run.
- $\mathcal{D}' = \{0, 1\}\ \forall\ v_i \in V$
- $\mathcal{C}'$ is the set of constraints, where each C_R can be defined by the same generic formula $(v = 1) \iff \diamond_{w=1}^{n}(block_{x,y} = val)_w$. For instance, the constraint $(right = 1) \iff (block_{x+1,y} = 1)$ ensures that Mario moves right iff there is a passable block by walking in his next right block.

5.3. Implementation

We have modeled but also implemented agents for Ms. Pac-Man and Mario AI Benchmark in order to validate the proposed SD Meta-CSP. We used the Choco Solver[1] for solving each sub-problem sequentially. Figure 4 illustrates the game-solver interaction scheme.

[1] http://www.emn.fr/z-info/choco-solver/

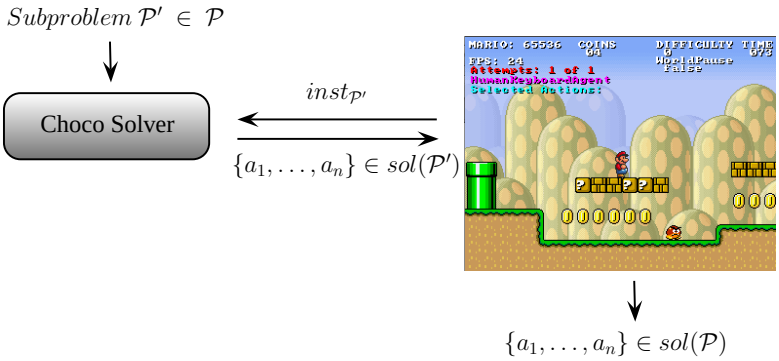

Fig. 4. Interaction scheme.

The game delivers the observation matrix to the solver which is able to compute a solution for a sub-problem of the SD Meta-CSP. Once the solution is reached, it is sent to the game that triggers the movement of the avatar. This interaction is performed as many times as needed to complete the stage. The solver is able to reach solutions in about 40 milliseconds (on a 3.1GHz Intel Core i3-2100 CPU with 4 GB RAM running Windows 7 Professional), which is mandatory for having a suitable interaction with the game. For both games, we have performed 40 tries; and although our goal here is rather to validate the use of SD Meta-CSP rather than performance of agents, we consider that results are reasonable. In the case of Ms. Pac-Man, 98% of tries pass the first level, 53% pass the second level, and 10% reached the 4th level. For Mario, different seeds lead to different stages, which 40 out of 40 were successfully solved by the agent.

6. Conclusion

In this paper, we have introduced the notion of SD Meta-CSPs, which is an extension of a meta-CSP where a sequential dependency among its sub-problems exists. The introduction of this new extension has been supported by the corresponding formal definition, a framework to handle it, and the illustration of its real-world applicability through the modeling and implementation of game agents as SD Meta-CSP for two well-known human-controlled avatar video games: Ms. Pac-Man and Mario Bros.

The role of the SD Meta-CSP here is to automatically control the avatar of the video game in order to success in the game. This is possible by decomposing the SD Meta-CSP in a set of CSPs, each one representing a move of the avatar in order to reach a goal. The CSP is modeled by considering as variables the possible movements of the avatar, and the constraints restrict the parts of the space that may be risky for the avatar or the situations that hinder to reach the goal of the game. The resolution of a SD Meta-CSP is handled through an interaction framework where the game systematically communicates with the solver that provides solution for each CSP resulting from the SD Meta-CSP decomposition. Our framework is able to successfully pass levels for both games.

The introduction of this new framework may conduct to different directions for future work. One clear way to follow is about solving new kind of problems as SD Meta-CSP. In particular, we visualize the modeling of transportation problems as SD Meta-CSPs such as the different variations of the dial-a-ride problem [9, 8], where each request of the transport network can be seen as a sub-problem of the SD Meta-CSP, which in turn represent the entire network. Another research direction is about performance, for instance to tune the model representing both the problems and sub-problems in order to achieve faster resolution processes and/or better scores in the game. Pursuing a similar goal, the tuning of the solver that handles the sub-problems may also improve the solving process. In this context, it would be interesting to employ autonomous search [25, 6] in the resolution, which has demonstrated to be useful for solving several CSPs.

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

Conflict of Interest. The authors declare that there is no conflict of interests regarding the publication of this article.

References

[1] ALHEJALI A. M., LUCAS S. M., *Using a training camp with genetic programming to evolve Ms Pac-Man agents*, Proceedings of the 2011 IEEE Conference on Computational Intelligence and Games (CIG), IEEE, pp. 118–125, 2011.

[2] BISTARELLI S., MONTANARI U., ROSSI F., SCHIEX T., VERFAILLIE G., FARGIER H., *Semiring-based CSPS and valued CSPS: Frameworks, properties, and comparison*, Constraints, **4** (3), pp. 199–240, 1999.

[3] BURROW P., LUCAS S. M., *Evolution versus temporal difference learning for learning to play Ms. Pac-Man*, Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games, CIG. IEEE, pp. 53–60, 2009.

[4] CHOUEIRY B., XU L., *An efficient consistency algorithm for the temporal constraint satisfaction problem*, AI Commun., **17** (4), pp. 213–221, 2004.

[5] COOPER M. C., *High-order consistency in valued constraint satisfaction*, Constraints, **10** (3), pp. 283–305, 2005.

[6] CRAWFORD B., SOTO R., MONFROY E., PALMA W., CASTRO C., PAREDES F., *Parameter tuning of a choice-function based hyperheuristic using particle swarm optimization*, Expert Syst. Appl., **40** (5), pp. 1690–1695, 2013.

[7] DECHTER R., *Constraint Processing*, Morgan Kauffman, 2003.

[8] DESAULNIERS G., DESROSIERS J., ERDMANN A., SOLOMON M. M., SOUMIS F., *VRP with Pickup and Delivery. Discrete Mathematics and Applications*, SIAM Publications, Ch. 9, pp. 225–242, 2002.

[9] DESROSIERS J., DUMAS Y., SOLOMON M. M., SOUMIS F., *Chapter 2 time constrained routing and scheduling*, in Ball M., Magnanti T., Monma C., Nemhauser G.

(Eds.), *Network Routing*, Vol. **8** of *Handbooks in Operations Research and Management Science*, Elsevier, pp. 35–139, 1995.

[10] DUBOIS D., FARGIER H., PRADE H., *The calculus of fuzzy restrictions as a basis for exible constraint satisfaction*, Proceedings of Second IEEE International Conference on Fuzzy Systems, (FUZZ) IEEE, pp. 1131–1136, 1993.

[11] FREUDER E., *Constraint solving techniques*, in Mayoh E. T. B., Penjaen J. (Eds.), *Constraint Programming of series F: Computer and Sytems Sciences*, NATO ASI Series, pp. 51–74, 1992.

[12] FREUDER E. C., HEFFERNAN R., WALLACE R. J., WILSON N., *Lexicographically-ordered constraint satisfaction problems*, Constraints, **15** (1), pp. 1–28, 2010.

[13] FREUDER E. C., WALLACE R. J., *Partial constraint satisfaction*, Artif. Intell., **58** (1–3), pp. 21–70, 1992.

[14] FREUDER E. C., WALLACE R. J., HEFFERNAN R., *Ordinal constraint satisfaction*, Proceedings of the 5th International Workshop on Soft Constraints, 2003.

[15] GELLE E., FALTINGS B., *Solving Mixed and Conditional Constraint Satisfaction Problems*, Constraints, **8** (2), pp. 107–141, 2003.

[16] KARAKOVSKIY S., TOGELIUS J., *The Mario Ai Benchmark and Competitions*, IEEE Trans. Comput. Intellig. and AI in Games, **4** (1), pp. 55–67, 2012.

[17] KÖKÉNY T., *A new ARC consistency algorithm for CSPS with hierarchical domains*, Proceedings of the 6th International Conference on Tools with Artificial Intelligence (ICTAI), pp. 439–445, 1994.

[18] LARROSA J., MESEGUER P., *Partition-based lower bound for MAX-CSP*, Proceedings of the International Conference on Principles and Practice of Constraint Programming, Vol. **1713** of Lecture Notes in Computer Science, Springer, pp. 303–315, 1999.

[19] LARROSA J., MESEGUER P., SCHIEX T., *Maintaining reversible DAC for MAX-CSP*, Artif. Intell., **107** (1), pp. 149–163, 1999.

[20] LARROSA J., SCHIEX T., *Solving weighted CSP by maintaining ARC consistency,*. Artif. Intell., **159** (1–2), pp. 1–26, 2004.

[21] MACKWORTH A., *Consistency in Networks of Relations*, Artificial Intelligence, **8** (1), pp. 99–118, 1977.

[22] MACKWORTH A., MULDER J., HAVENS W., *Hierarchical ARC Consistency: Exploiting Structured Domains in Constraint Satisfaction Problems*, Computational Intelligence, **1** (1), pp. 118–126, 1985.

[23] MIGUEL I., SHEN Q., *Dynamic exible constraint satisfaction*, Appl. Intell., **13** (3), pp. 231–245, 2000.

[24] MITTAL S., FALKENHAINER B., *Dynamic Constraint Satisfaction Problems*, Proceedings of the 8th National Conference on Artificial Intelligence (AAAI), The MIT Press, pp. 25–32, 1990.

[25] 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), pp. 1–22, 2013.

[26] MOUHOUB M., SUKPAN A., *Conditional and composite temporal CSPS*, Appl. Intell., **36** (1), pp. 90–107, 2012.

[27] MOUHOUB M., SUKPAN A., *Managing dynamic CSPS with preferences*, Appl. Intell., **37** (3), pp. 446–462, 2012.

[28] PEDERSEN C., TOGELIUS J., YANNAKAKIS G. N., *Modeling player experience in Super Mario Bros*, Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games (CIG) IEEE, pp. 132–139, 2009.

[29] ROBLES D., LUCAS S. M., *A simple tree search method for playing Ms. Pac-Man*, Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games (CIG) IEEE, pp. 249–255, 2009.

[30] SABIN D., FREUDER E. C., *Configuration as composite constraint satisfaction*, Proceedings of Artificial Intelligence and Manufacturing. Research Planning Workshop, AAAI Press, pp. 153–161, 1996.

[31] SAMOTHRAKIS S., ROBLES D., LUCAS S. M., *Fast Approximate Max-N Monte Carlo Tree Search for Ms Pac-Man*, IEEE Trans. Comput. Intellig. and AI in Games, **3** (2), pp. 142–154, 2011.

[32] SCHIEX T., *Possibilistic constraint satisfaction problems or "how to handle soft constraints?"*, Proceedings of the Eighth Annual Conference on Uncertainty in Artificial Intelligence (UAI) Morgan Kaufmann, pp. 268–275, 1992.

[33] SCHIEX T., FARGIER H., VERFAILLIE G., *Valued constraint satisfaction problems: Hard and easy problems*, Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI) Morgan Kaufmann, pp. 631–639, 1995.

[34] SHAKER N., NICOLAU M., YANNAKAKIS G. N., TOGELIUS J., O'NEILL M., *Evolving Levels for Super Mario Bros Using Grammatical Evolution*, Proceedings of the 2012 IEEE Conference on Computational Intelligence and Games (CIG) IEEE, pp. 304–311, 2012.

[35] SHAKER N., TOGELIUS J., YANNAKAKIS G. N., WEBER B. G., SHIMIZU T., HASHIYAMA T., SORENSON N., PASQUIER P., MAWHORTER P. A., TAKAHASHI G., SMITH G., BAUMGARTEN R., *The 2010 Mario AI Championship: Level Generation Track*, IEEE Trans. Comput. Intellig. and AI in Games, **3** (4), pp. 332–347, 2011.

[36] SHAKER N., YANNAKAKIS G. N., TOGELIUS J., NICOLAU M., O'NEILL M., *Evolving Personalized Content for Super Mario Bros Using Grammatical Evolution*, Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE), The AAAI Press, 2012.

[37] SOMBAT W., ROHLFSHAGEN P., LUCAS S. M., *Evaluating The Enjoyability of The Ghosts in Ms Pac-Man*, Proceedings of the 2012 IEEE Conference on Computational Intelligence and Games (CIG), pp. 379–387, 2012.

[38] VERFAILLIE G., LEMAITRE M., SCHIEX T., *Russian Doll Search for Solving Constraint Optimization Problems*, AAAI/IAAI, Vol. **1**, pp. 181–187, 1996.

[39] WALLACE R. J., WILSON N., *Conditional Lexicographic Orders in Constraint Satisfaction Problems*, Annals OR, **171** (1), pp. 3–25, 2009.

[40] XU K., LI W., *Exact Phase Transitions in Random Constraint Satisfaction Problems*, Journal of Artificial Intelligence Research, **12**, pp. 93–103, 2000.

[41] XU L., CHOUEIRY B. Y., *Improving Backtrack Search for Solving The TCSP*, Proceedings of Principles and Practice of Constraint Programming (CP), Vol. **2833** of Lecture Notes in Computer Science, Springer, pp. 754–768, 2003.