Constraint Games: Framework and Local Search Solver

Thi-Van-Anh Nguyen, Arnaud Lallouet, Lucas Bordeaux

Constraint Games: Framework and Local Search Solver

Thi-Van-Anh Nguyen, Arnaud Lallouet
GREYC, Université de Caen Basse-Normandie
Caen, France
{*thi-van-anh.nguyen,arnaud.lallouet*}*@unicaen.fr*

Lucas Bordeaux
Microsoft Research
Cambridge, United Kingdom
lucasb@microsoft.com

***Abstract*—Game theory is a highly successful paradigm for strategic decision making between multiple agents having conflicting objectives. Since a few years, games have been studied in a computational perspective, raising new issues like complexity of equilibria or succinctness of representation. Indeed, the main representation for general games is still a n-dimensional matrix of exponential size called *normal form*. In this paper, we introduce the framework of *Constraint Games* to model strategic interaction between players. A Constraint Game is composed of a set of variables shared by all the players. Among these variables, each player owns a set of *decision variables* she can control and a Constraint Optimization Problem defining her preferences. Since the preferences of a player depend on the decisions taken by the other players, each player may try to improve her position by choosing an assignment that optimizes her preferences. Pure Nash equilibria are situations in which no player may improve her preferences unilaterally. Constraint Games are thus a generic tool to model general games and can be exponentially more succinct than their normal form. We show the practical utility of the framework by modelling a few realistic problems and we propose an algorithm based on tabu search to compute pure Nash equilibria in Constraint games that outperforms the algorithms based on normal form. In addition, Constraint Games raise some interesting research issues that deserve further attention.**

***Keywords*-Constraint Programming; Game Theory; Nash Equilibria;**

I. Introduction

The mathematical field of *game theory* [18] has been set up to address problems of strategic decision making when modeling interacting agents with conflicting objectives. Game theory has an incredible success in description of economic processes, but is also used in various other domains such as biology, political sciences or philosophy. One of the most fundamental problems in computational game theory is undoubtedly the computation of a Nash equilibrium [20], which models a situation where no agent has an incentive to change his decision unilaterally. Other issues are numerous and include simultaneous or sequential interaction mode, complete or incomplete knowledge, determinism, coalitions, repetition, etc.

A game is composed of a set of players, each of them having a set of possible actions they can perform. A game in normal form is represented as a payoff matrix stating for each player the reward she will get for any combination of actions of all players. NP-completeness [13] follows from this representation or from a representation for which the payoffs can be computed in polynomial time. Indeed, one important problem of game theory is the compactness of payoff representation because the matrix grows exponentially with the number of players. Surprisingly, although compactness can be achieved by switching to a combinatorial setting for payoffs, there are few attempts to define compact yet generic languages for the expression of games.

In this paper, we present *Constraint Games* which use Constraint Satisfaction Problems (CSP) as basic tool for expressing players preferences. In a constraint game, each player controls a set of finite domain variables and their Cartesian product defines a possible action space for the player. In addition, each player owns a CSP on all players variables which defines satisfaction. Given the partial state defined by the other players moves, a player can choose his variable assignment in order to satisfy his own CSP. A global solution to such a problem is given by the notion of *pure Nash equilibrium*, in which no player can improve unilaterally his own satisfaction.

We propose four natural variants of this concept that differ from whether optimization is allowed or not and from the definition of the search space. In *Constraint Satisfaction Games* (CSG), the payoff of a player is simply defined as the satisfaction of a CSP. In *Constraint Optimization Games* (COG), the objective of a player is to optimize some value according to some constraints. We also introduce two variants called CSG-HC and COG-HC (HC stands for *hard constraints*) allowing to model problems in which all players must respect some hard rules of the games besides theirs preferences expressed by their goals.

Works on game theory are too numerous to be mentioned. But Constraint Games inherit from different lines of work. The closest framework is certainly the one of *boolean games* [4], [14]. In boolean games, each player owns a SAT problem which defines his satisfaction. However, unlike constraint games, there is no mean to specify inside the language a non-boolean utility. This is why it is required to provide an external way to define preferences. In [5], CP-nets [9] were used to define players' preferences. Another difference is that we provide hard constraints that limit the equilibrium condition to the satisfiable part of the search space. Games with Hard Constraints (originally called Shared Constraints)

[23] are not related to constraint programming but to general constrained optimization. No specific algorithm has been proposed to solve boolean games.

Compact representation of utility is a challenge in computational game theory. Several proposals have contributed to define more tractable representations, like *Graphical games* [17], in which a player's utility only depends on a subset of the other players. Then it is possible to reduce the size of the utility tables by only storing useful utilities just like conditional probabilities in bayesian networks. There is no problem to import this formalism to Constraint Games, as it already defined for boolean games [3], and we leave this for future work. *Action-graph games* [16] exploit properties of certain games like context independence and anonymity to achieve a polynomial representation and efficient solving but they are not as declarative and natural as constraint games. *Congestion games* [24], Routing games [25] are other kinds of specific games that enjoy a compact representation, as well as sparce, symmetric, anonymous, local-effect or multimatrix games [21]. However they are not general games.

First, it may be noticed that Constraint Game encodings are never larger than the normal form. But just like boolean games, Constraint Games can be exponentially more succint than the payoff matrix. An example is when an agent has constraints encoded by a CSP $x_i = y_i$ for $2n$ variables (x_i) and (y_i) on a domain with d elements. The representation takes n constraints while there are d^n solutions, which means that a payoff matrix, even in a sparse representation that only keeps positive entries, would be exponential. In the worst case, however, Constraint Games, just like CSP can blow-up to a size similar to their set of models, as shown by a simple counting argument (in any formal language, using N bits we can encode 2^N models while there are 2^{2^N} Boolean functions of N inputs). As for CSP, our feeling is that this worst case does not happen often in practice. Moreover, we believe that the modeling facilities offered by constraint languages, especially with global constraints, allows to encode many useful problems in an elegant way.

From a modelling perspective, solution concepts are heavily discussed in the game theory community, as pure Nash equilibria do not provide a satisfactory notion of solution all the time. The main directions are mixed equilibria [20] and taking a subset of equilibria with additional properties like Pareto-optimality or subgame equilibria [21].

There are few attempts to use Constraint Programming in Game Theory. In [13], the authors presented a CSP encoding of the reaction operator in graphical games. In [7], it has been proposed to compute a mixed equilibrium using continuous constraints. Some other formalism solve one combinatorial problem by multiple agents, either with a predefined assignment of variables to agents like in DCOP [12] or by letting the agents select dynamically their variable like in SAT-Games [28] and Adversarial CSP [10]. Other types of equilibria such as Stackelberg equilibria have been investigated within the QCSP framework [2], [6].

Elimination of dominated strategies can be seen as a form of propagation for games [1]. Several types of domination have been devised, among them the best known are *strong domination, weak domination* and *never best response*. However, this detection is very costly (actually Σ_2^p-complete for boolean games, see [4]).

In this paper, we introduce an algorithm based on tabu search to compute equilibria in Constraint Games. We also give an illustration of complex real-world problems that can be modelled using this new framework in a concise way and show on benchmarks that the constraint approach goes beyond current state of the art in general strategic game solving. Local search has been used to compute Nash equilibria but only for two-player games and for mixed equilibria [11].

The plan of the paper is as follows: we present Constraint Games in Section II, we give examples in Section III, we present algorithms in Section IV and experiments in Section V.

II. Constraint Games

Constraints and CSP: Let V be a set of variables and $D = (D_X)_{X \in V}$ be the family of their (finite) domains. For $W \subseteq V$, we denote by D^W the set of tuples on W, namely $\Pi_{X \in W} D_X$. Projection of a tuple (or a set of tuples) on a variable (or a set of variables) is denoted by $|$. For example, for $t \in D^V$, $t|_W = (t_X)_{X \in W}$ and for $E \subseteq D^V$, $E|_W = \{t|_W \mid t \in E\}$. For $W, U \subseteq V$, the join of $A \subseteq D^W$ and $B \subseteq D^U$ is $A \bowtie B = \{t \in D^{W \cup U} \mid t|_W \in A \wedge t|_U \in B\}$. When $W \cap U = \emptyset$, we denote the join of tuples $t \in D^W$ and $u \in D^U$ by (t, u). A *constraint* $c = (W, T)$ is a couple composed of a subset $W = \mathit{var}(c) \subseteq V$ of variables and a relation $T = \mathit{sol}(c) \subseteq D^W$ (called *solutions*). A *Constraint Satisfaction Problem* (or CSP) is a set of constraints. We denote by $\mathit{var}(C) = \bigcup_{c \in C} \mathit{var}(c)$ its set of variables and by $\mathit{sol}(C) = \bowtie_{c \in C} \mathit{sol}(c)$ its set of solutions.

Constraint Satisfaction Games: Let $\mathcal{P}$ be a set of players and V a set of variables. Each player i is given a set of *controlled* variables $V_i \subseteq V$. The sets $(V_i)_{i \in \mathcal{P}}$ are disjoint. Thus each variable is controlled by at most one player. A variable which is not controlled by any player is called an *existential* variable and belongs to V_E.

Definition 1 (Constraint Satisfaction Game): A *Constraint Satisfaction Game* (or CSG) is a 4-tuple $(\mathcal{P}, V, D, G)$ where $\mathcal{P}$ is a finite set of players, V is a set of variables composed of a family of disjoint sets (V_i) for each player $i \in \mathcal{P}$ and a set V_E of *existential* variables disjoint of all the players variables, $D = (D_X)_{X \in V}$ is the family of their domains and $G = (G_i)_{i \in \mathcal{P}}$ is a family of CSP on V.
The CSP G_i is called the *goal* of the player i. The intuition behind CSG is that, while a player i can only control her own subset of variables V_i, her satisfaction will depend also on variables controlled by all the other players. A controlled

$z=0$		y		
		0	1	2
x	0	(0,0,1)	(0,1,0)	(0,1,0)
	1	(1,0,0)	(0,1,0)	**(1,1,0)**
	2	(1,0,0)	(1,0,0)	(0,1,0)
$z=1$		y		
		0	1	2
x	0	(0,0,0)	(0,0,1)	(0,1,0)
	1	(0,0,1)	(0,0,0)	**(0,1,0)**
	2	(1,0,0)	(1,0,0)	**(0,1,0)**
$z=2$		y		
		0	1	2
x	0	(0,0,0)	(0,0,0)	**(0,0,1)**
	1	(0,0,0)	**(0,0,1)**	*(0,0,0)*
	2	**(0,0,1)**	*(0,0,0)*	*(0,0,0)*

Figure 1. Boolean payoff matrix of Example 1. Nash equilibria are depicted in bold and italics (italics stands for equilibria in which no player is satisfied).

variable is called a *decision variable*. The intuition behind existential variables is that they are existentially quantified (but most of the time they will be functionally defined from decision variables).

Example 1: We consider the following CSG: the set of players is $\mathcal{P} = \{X, Y, Z\}$. Each player owns one variable: $V_X = \{x\}, V_Y = \{y\}$ and $V_Z = \{z\}$ with $D_x = D_y = D_z = \{0, 1, 2\}$. The goals are $G_X = \{x \neq y, x > z\}$, $G_Y = \{x \leq y, y > z\}$ and $G_Z = \{x + y = z\}$.
A *strategy* for player i is an assignment of the variables V_i controlled by player i. A *strategy profile* $s = (s_i)_{i \in \mathcal{P}}$ is the given of a strategy for each player.

Definition 2 (Winning strategy): A strategy profile s is winning for i if it satisfies the goal of i: $s \in sol(G_i)$.
A CSG can be interpreted as a classical game with a boolean payoff function which takes value 1 when the player's CSP is satisfied and 0 when not. The boolean payoff 3-dimensional matrix of Example 1 in normal form is depicted in Figure 1.

We denote by s_{-i} the projection of s on $V - V_i$. Given a strategy profile s, a player i has a *beneficial deviation* if $s \notin sol(G_i)$ and $\exists s'_i \in D^{V_i}$ such that $(s'_i, s_{-i}) \in sol(G_i)$. Beneficial deviation represents the fact that a player will try to maximize her satisfaction by changing the assignment of the variables she can control if she is unsatisfied by the current assignment. Then we define the notion of solution of a CSG by consistent Nash equilibrium (see Section VI for a discussion on *solution concepts*):

Definition 3 (Nash Equilibrium): A strategy profile s is a *Nash equilibrium* of the CSG $\mathcal{C}$ if and only if no player has a beneficial deviation

Example 2 (Example 1 continued): Each solution of G_X is a winning strategy for player X. For example, 100 (which stands for $x = 1, y = 0, z = 0$) is a winning strategy for player X. However, 100 is not a solution of the CSG because Player Y may deviate from 0 to 1 to get the winning strategy 110 solution of G_Y. Player Z is able to do the same with 101. The strategy profile 120 is a Nash equilibrium because it is solution for X and Y, and Player Z is unable to deviate because neither 120, 121 or 122 are solution of G_Z.

Proposition 1 (derived of [4]): CSG are Σ_2^p-complete.

Hard Constraints: The players goals could be considered as soft constraints or preferences. It may happen however some games have rules that forbid some strategy profile as they model impossible situations. It is natural to reject such profile by setting *hard constraints* shared by all players. Hard constraints have been introduced in game theory under the name of *shared constraints* [23] and were not related to constraint programming but to general optimization under constraints. Hard constraints can be easily expressed in the framework of Constraint Games by adding an additional CSP on the whole set of variables in order to constrain the set of possible strategy profiles:

Definition 4 (CSG with Hard Constraints): A *CSG with Hard Constraints* (or CSG-HC) is a 5-uplet $(\mathcal{P}, V, D, C, G)$ where $(\mathcal{P}, V, D, G)$ is a CSG and C is a CSP on V.
The intended meaning of the hard constraints is that beneficial deviation is only allowed in the satisfiable subspace defined by the additional CSP. It is useful to distinguish a strategy profile which does not satisfy any player's goal from a strategy profile which does not satisfy the hard constraints. The former can be a PNE if no player has a beneficial deviation while the latter cannot. Therefore hard constraints provide an increase of modelling expressibility (without however changing the general complexity of CSG).

Constraint Optimization Games: By adding an optimization condition it is possible to represent classical games. A *Constraint Optimization Game* (or COG) is an extension of CSG in which each player tries to optimize his goal. This is achieved by adding for each player i an optimization condition $\min(x)$ or $\max(x)$ where $x \in V_i$ is a variable controlled by i.

Definition 5 (Constraint Optimization Game): A *Constraint Optimization Game* (or COG) is a 5-tuple $(\mathcal{P}, V, D, G, Opt)$ where $(\mathcal{P}, V, D, G)$ is a CSG and $Opt = (Opt_i)_{i \in \mathcal{P}}$ is a family of optimization conditions for each player of the form $\min(x)$ or $\max(x)$ where $x \in V$.
A winning strategy for player i is still a strategy profile which satisfies G_i. However, the notion of beneficial deviation needs to be slightly adapted. Given a strategy profile s, a player i having as optimization condition $\min(x)$ (resp. $\max(x)$) has a *beneficial deviation* if $\exists s'_i \in D^{V_i}$ such that $s' = (s'_i, s_{-i}) \in sol(G_i)$ and $s'|_x < s|_x$ (resp $s'|_x > s|_x$). Given this, the notion of solution is the same as for CSG. In addition, COG can be extended with hard constraints the same way CSG are, yielding COG-HC.

III. Motivating Examples

Here we give examples of problems which acknowledge the usefulness and versatility of Constraint Games.

Example 3 (Prisoner's dilemma): The classical *prisoner's dilemma* [22] introduces two prisoners put in jail without being able to talk to each other. The police plans to sentence both of them one year, but proposes to each of them to testify against his partner in exchange of liberty for him and three years for his partner. But if both testify, then both are sentenced to two years in jail. In this game, each player has possibility to play 0 (defect) or 1 (cooperate) with respect to the other player. The prisoner's dilemma can be represented by the values given in Figure 2 by the following COG:

- $\mathcal{P} = \{A, B\}$
- $V_A = \{x\}$, $V_B = \{y\}$, $V_E = \{z_A, z_B\}$
- $D(x) = D(y) = \{0, 1\}$
- $G_A = \{z_A = -x + 2y + 1\}, G_B = \{z_B = 2x - y + 1\}$
- $Opt_A = min(z_A), Opt_B = min(z_B)$

		y	
		0	1
x	0	(1,1)	(3,0)
	1	(0,3)	(2,2)

Figure 2. Prisoner's dilemma: COG representation on the left and bimatrix normal form on the right.

Example 4 (Cloud Resource Allocation Game): Resource allocation is a central issue in cloud computing where clients use and pay computing resources on demand. In order to manage conflicting interests between clients, [15] has proposed the framework of *CRAG* (Cloud Resource Allocation Game) in which resource assignments are defined by game equilibria. According to the authors, this allocation shows a performance increase from 15 to 88% with respect to standard round-robin scheduling commonly used by cloud vendors.

A cloud computing provider owns a set $\mathcal{M} = \{M_1, \ldots, M_m\}$ of m machines, each machine M_j having a capacity c_j representing the amount of resource available (for example CPU-hour, memory). The cost of using machine j is given by $l_j(x) = x \times u_j$ where x is the number of resources requested and u_j some unit cost. A set of n clients $\mathcal{P} = \{1, 2, .., n\}$ wants to use simultaneously the cloud in order to perform tasks. Client $i \in \mathcal{P}$ has m_i tasks $\{T_{i1}, ..., T_{im_i}\}$ to perform, with respective requested capacity of $\{d_{i1}, ..., d_{im_i}\}$. Each client $i \in \mathcal{P}$ chooses selfishly an allocation r_{ik} for the task T_{ik} ($k \in 1..m_i$) and wishes to minimize her cost $cost_i = \sum_{k=1..m_i} l_{r_{ik}}(d_{ik})$. We assume that the provider's resources amount is sufficient to accommodate the resources requested by all of the clients: $\sum_{i\in[1..n]} \sum_{k\in[1..m_i]} d_{ik} \leq \sum_{j\in[1,..m]} c_j$. This problem can be modelled by the following COG-HC:

- $\mathcal{P} = \{1, .., n\}$
- $\forall i \in \mathcal{P}, V_i = \{r_{i1}, ..., r_{im_i}\}$
- $\forall i \in \mathcal{P}, \forall k \in [1, ..., m_i], D(r_{ik}) = \{1, \ldots m\}$
- C is composed of the following constraints:
  - channelling constraints for boolean variables stating that machine j is requested by task t_{ik}: $(r_{ik} = j) \leftrightarrow (choice_{ijk} = 1)$
  - capacity constraints: $\forall j \in [1, .., m]$, $\sum_{i\in[1..n]} \sum_{k\in[1..m_i]} choice_{ijk} \times d_{ik} \leq c_j$
- $\forall i \in \mathcal{P}, G_i$ is composed of the following constraint:

$$cost_i = \sum_{j=1..m} \sum_{k=1..m_i} choice_{ijk} \times l_j(d_{ik})$$

- $\forall i \in \mathcal{P}, Opt_i = min(cost_i)$

Example 5 (Network Game): This example is inspired by [8] and taken from telecommunication industry. A network provider owns m links to transfer data. Each link j is specified by 3 parameters: capacity c_j, speed per data unit s_j and price per data unit p_j. A group of n clients would like to transfer data across theses links (client i from a source x_i to a target y_i, each source and target are fully connected to each link of the vendor and each path has to cross a tolled arc). In order to reach a link j of the network, a client i has to pay a fixed fee α_{ij} and a fixed delay β_{ij}.

Hence, with any link j customer i chooses, she has to pay an addition cost α_{ij} per data unit and it also takes an additional time β_{ij} per data unit to transit on the tolled arc. Each customer could always choose another provider with cost ψ_i, so if the provider's price offered is competitive, she therefore wishes to minimize her transferred data time. This problem can be modelled as a COG-HC as follows:

- $\mathcal{P} = \{1, .., n\}$
- $\forall i \in \mathcal{P}, V_i = \{r_i\}$
- $\forall i \in \mathcal{P}, D(r_i) = \{1, \ldots m\}$
- C is composed of the following constraints:
  - channelling constraints for boolean variables stating that link j is requested by data d_i: $(r_i = j) \leftrightarrow (choice_{ij} = 1)$
  - capacity constraints:

$$\forall j \in [1, .., m], \sum_{i\in[1..n]} choice_{ij} \times d_i \leq c_j$$

- $\forall i \in \mathcal{P}, G_i$ is composed of the following constraints:
  - $cost_i = \sum_{j=1..m} choice_{ij} \times d_i \times (p_j + \alpha_{ij})$
  - $cost_i \leq \psi_i$
  - $time_i = \sum_{j=1..m} choice_{ij} \times d_i \times (s_j + \beta_{ij})$
- $\forall i \in \mathcal{P}, Opt_i = min(time_i)$

IV. Solving Constraint Games

Solving games is a remarkably difficult task. We are not aware of any efficient complete algorithm for games expressed in normal form [27]. Hence, we present here a local search algorithm based on tabu search called *CG-tabu*. It allows to find the first pure Nash equilibrium in game instances whose size is way beyond the size accessible to algorithms based on normal form.

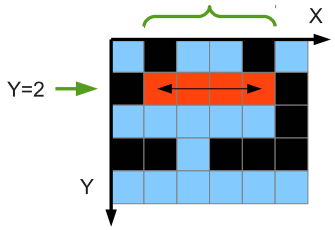

Figure 3. Possible moves for player X when player Y chooses value 2 are given in braces. In black are depicted forbidden states that do not satisfy hard constraints.

Algorithm IV.1 function br-csg(*CSG* C, player i, strategy profile x) **returns** strategy profile

```
initialize TmpSolver with G_i and hard constraints (if exists);
for all y ∈ V_{-i} do
    TmpSolver.add_constraint(y = x|_y);
end for
set search tree strategy to begin with V_i = x|_{V_i};
return TmpSolver.getSolution();
```

In *CG-tabu*, the notion of move is given by the deviation a player may perform to get her optimal strategy profile (see Figure 3). The tabu list is used to forbid a player to be chosen too early after he has moved. In contrast with classical search space where a local search algorithm wants to escape local optima, it may happen that the trajectory gets stucked in cycles. Thus a tabu list of size S allows to avoid direct cycles of length S and in practice allows also larger cycles to be escaped. Algorithm IV.1 (*br-csg*: best response for CSG) and IV.2 (*br-cog*: best response for COG) detail how to find the best neighbor of a player.

Algorithm IV.2 function br-cog(*COG* C, player i, strategy profile x) **returns** strategy profile

```
initialize TmpSolver with G_i and hard constraints (if exists);
for all y ∈ V_{-i} do
    TmpSolver.add_constraint(y = x|_y);
end for
set search tree strategy to begin with V_i = x|_{V_i};
add objective Opt_i
return TmpSolver.getOptimalSolution();
```

Algorithm IV.1 aims at finding a strategy profile satisfying the goal of player i. If the current strategy profile already satisfies the goal of player i, there is no reason to move so the algorithm simply returns the state itself. We do that by

Algorithm IV.3 function CG-tabu(*C*G* C) **returns** strategy profile

```
1: Nash = false; Tabu_list = ∅; iter = 0;
2: current = choose a strategy profile at random;
3: while (not Nash) and (iter < max_tries) do
4:    Nash = true;
5:    Restart = false;
6:    for all i ∈ (P − Tabu_list) do
7:       strategy = br-c*g(C,i,current);
8:       if (strategy ≠ current and strategy ≠ null) then
9:          Nash = false;
10:         Add (i,tabu_length) to Tabu_list;
11:         current = strategy;
12:         break;
13:      end if
14:   end for
15:   if (Nash) then
16:      for all i ∈ Tabu_list do
17:         strategy = br-c*g(C,i,current);
18:         if (strategy ≠ current and strategy ≠ null) then
19:            Restart = true;
20:            break;
21:         end if
22:      end for
23:   end if
24:   if (Restart) then
25:      Nash = false;
26:      current = choose a strategy profile at random;
27:   end if
28:   iter++;
29:   for all (i,length) ∈ Tabu_list do
30:      if (length = 0) then
31:         remove (i,length) from Tabu_list
32:      else
33:         decrement length
34:      end if
35:   end for
36:   if (Nash) then
37:      initialize TmpSolver with hard constraints;
38:      for all y ∈ V do
39:         TmpSolver.add_constraint(y = x|_y);
40:      end for
41:      if (TmpSolver.getSolution() = null) then
42:         Nash = false;
43:      end if
44:   end if
45: end while
46: if (Nash) then
47:    return current
48: else
49:    return null
50: end if
```

#Machines	10			15			20		
#Clients	$N/4$	$N/3$	$N/2$	$N/4$	$N/3$	$N/2$	$N/4$	$N/3$	$N/2$
20	1.70	1.57	1.25	3.62	3.29	2.47	6.45	6.39	5.11
40	19.61	16.82	11.18	31.33	27.81	19.61	51.06	45.67	33.77
60	79.02	64.55	42.51	144.93	118.32	82.61	236.52	191.37	138.64
80	69.83	226.12	131.37	442.17	376.01	243.29	625.98	542.16	359.31
100	714.74	587.07	340.18	1046.62	875.17	553.34	1861.98	1497.26	809.80
120	804.62	641.21	381.93	2912.82	2463.06	1507.98	4687.44	3302.11	1622.21
140	3702.29	3207.02	1937.31	8495.97	7332.04	2913.71	8929.80	7007.92	3456.73
160	5982.00	4760.32	3146.78	12946.80	11359.80	5228.31	8349.11	6217.61	3830.52
180	3745.62	2867.99	1592.94	5148.36	4106.83	2466.34	11505.52	8891.57	5120.90
200	5865.95	4584.21	2575.30	9963.22	7655.15	4097.97	19008.18	14212.29	7926.21

Table I
AVERAGE TIME OF THREE CG-TABU VARIANTS ON CRAG

#Links	10			15			20		
#Clients	$N/4$	$N/3$	$N/2$	$N/4$	$N/3$	$N/2$	$N/4$	$N/3$	$N/2$
30	10.81	8.79	6.07	17.68	15.40	12.26	29.06	24.91	20.20
60	85.15	73.79	54.20	128.60	110.83	82.57	187.97	161.19	117.29
90	257.11	216.76	157.34	477.41	389.46	274.79	713.32	603.30	448.34
120	738.30	618.95	474.72	1712.33	1449.23	1034.00	1995.64	1650.89	1129.20
150	2087.465	1723.35	1347.01	2967.13	2495.77	1722.22	4730.62	3684.05	2558.09
180	4120.63	3450.34	2542.30	5539.07	4557.53	2966.74	12195.39	9937.90	6886.45
210	6736.89	5753.13	3677.70	10887.19	9146.39	6628.14	6594.18	5424.25	3575.65

Table II
AVERAGE TIME OF THREE CG-TABU VARIANTS ON NG

forcing the enumeration strategy to start with the current state. In case there is no solution to the problem, the *null* state is returned and no move is performed. Finding the best response of a player in COG is depicted in Algorithm IV.2. Due to the additional optimality check, the objective of the player is added (line 6) in order to find the optimal solution. Otherwise, the technique is the same.

Algorithm IV.3 specifies how CG-Tabu processes. The notation *c*g* is a generic replacement for CSG and COG. The main loop (lines 3-45) iterates until an equilibrium is found or a maximum number of moves have been done. In lines 6-14, non-tabu players are checked against deviation. If *br-c*g* returns the same state (line 7), the player is already satisfied and we move to the next player. If it returns the *null* state, then all assignments of player i are unsatisfiable and it may be a case for an equilibrium. If no player can deviate, then tabu players are also checked (lines 15-23) to ensure that a Nash equilibrium is found. If only a tabu player can deviate, a restart is performed (lines 24-27). Lines 29-35 are devoted to the update of the tabu list. Then if the current state is an equilibrium candidate, a last check is performed on hard constraints to check whether they are satisfied or not (lines 36-44). If not, then the state cannot be an equilibrium.

Proposition 2: CG-tabu is correct.

> *Proof:* A reported equilibrium is correct because it has been successively checked against deviation for all tabu and non-tabu players and it satisfies the hard constraints. ■

V. EXPERIMENTS

We have implemented a solver called *CG-Solve* on top of the constraint library Choco [26]. This solver allows to express Constraint Games and solves them using *CG-Tabu*. An important point is that our solver accepts all constraints provided by Choco, and reuses the constraint propagators already present in the library.

We provide here the experiment results on the Cloud Resource Allocation Game (CRAG) and Network Game (NG). All experiments have been run on a server with four 12-core AMD Opteron processors 6174 at 2,2 GHz and 256 GB of RAM memory. For CRAG, we have run experiments with main parameters the number of clients and the number of machines. For NG, the parameters are the number of clients and the number of links. In both games, a set of instances have been randomly generated to set the other parameters (for example, for CRAG, it is capacity and demand). The games considered are very large, up to 20^{400} strategy profiles (200 clients, 20 machines, 2 tasks) for CRAG and 20^{210} (210 clients, 20 links) for NG.

All time reported are mean values of 50 runs, i.e. 5 instances, 10 times per instance with different initial points randomly chosen. Table I and II show the runtime (measured in seconds) obtained by three CG-tabu variants on the two problems. The variants differ by the length of Tabu list: $N/4$, $N/3$, et $N/2$ where N means the number of players. They are launched on the same instances. As we can see from the tables, in all of the game instances, the longer

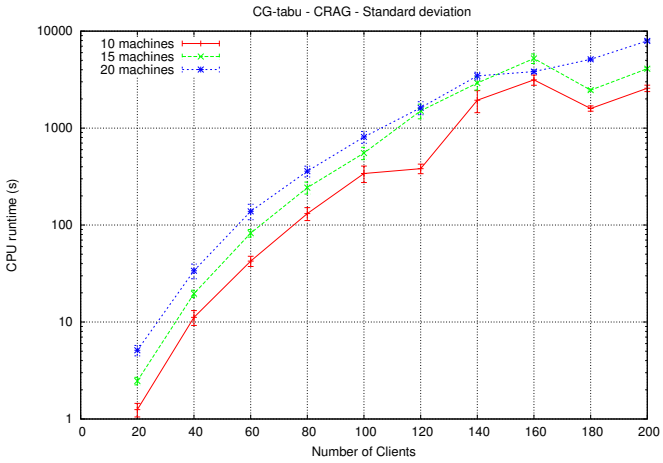

Figure 4. Study of standard deviation of CG-tabu for CRAG for different instances with same parameters

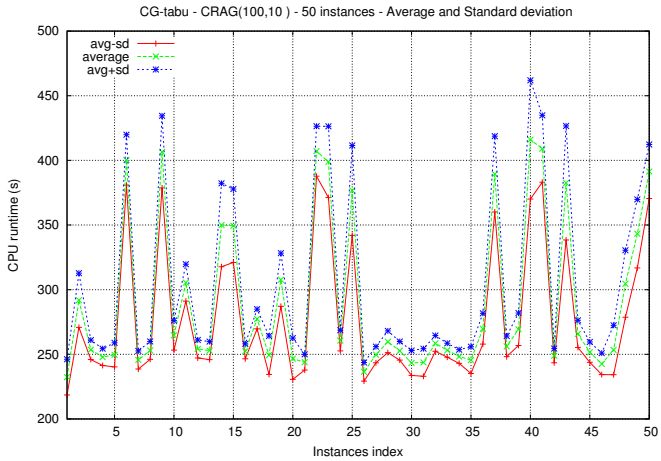

Figure 5. Study of standard deviation of CG-tabu on CRAG for different runs of the same instance

the Tabu tenure is, the faster the runtime is. The reason is that the Tabu list allows larger cycles to be escaped. So, the algorithm converges more quickly. An interesting point of the overall performance is that results presented are competitive with state-of-the art ad-hoc approaches for game solving. For example, in [15], CRAG games are solved up to 200 players like we do with CG-tabu. In comparison, the only general solver publicly available is *Gambit* [19] and is based on exhaustive enumeration. The largest CRAG instance we were able to solve with this solver is for 6 clients and 4 machines.

We also have studied standard deviation, and the results are shown in Figures 4 and 5. In Figure 4, we have depicted the average and standard deviation for a set of CRAG instances. Each point corresponds to 10 runs on 5 different instances. The deviation is around 4%. In Figure 5, we have performed a more detailed test on only one set of parameters (100 clients and 10 machines) on 50 different instances and depicted the standard deviation for each instance along 10 runs of the algorithm.

An apparently interesting idea is to import a kind of "min conflict" heuristics. We have performed tests in which the non-tabu player chosen by the algorithm is not the first one but the player who has the best improvement of his goal. Unfortunately, the time needed to evaluate all player moves is too important and the overall performance is at least one order of magnitude slower then the naive choice.

VI. Conclusion and Perspectives

In this paper, we propose *Constraint Games*, the first framework allows to model and solve in a natural way strategic games by using Constraint Programming. Constraint Games come in two flavors: Constraint Satisfaction Games (CSG) and Constraint Optimization Games (COG), with or without hard constraints. The notion of solution of a Constraint Game is the one of Nash equilibrium, a

situation in which no agent has an incentive to deviate from the current situation. We present real-world problems that can be modelled by Constraint Games, we propose a local search solver based on tabu search to quickly find the first Nash equilibrium and we demonstrate that it has competitive performances with ad-hoc approaches on a set of benchmarks.

ACKNOWLEDGEMENT

This work is supported by Microsoft Research grant MRL-2011-046

REFERENCES

[1] Apt, K.R.: Uniform proofs of order independence for various strategy elimination procedures. CoRR cs.GT/0403024 (2004)

[2] Benedetti, M., Lallouet, A., Vautard, J.: Quantified constraint optimization. In: Stuckey, P.J. (ed.) CP. Lecture Notes in Computer Science, vol. 5202, pp. 463–477. Springer (2008)

[3] Bonzon, E., Lagasquie-Schiex, M.C., Lang, J.: Dependencies between players in boolean games. Int. J. Approx. Reasoning 50(6), 899–914 (2009)

[4] Bonzon, E., Lagasquie-Schiex, M.C., Lang, J., Zanuttini, B.: Boolean games revisited. In: Brewka, G., Coradeschi, S., Perini, A., Traverso, P. (eds.) ECAI. Frontiers in Artificial Intelligence and Applications, vol. 141, pp. 265–269. IOS Press (2006)

[5] Bonzon, E., Lagasquie-Schiex, M.C., Lang, J., Zanuttini, B.: Compact preference representation and boolean games. Autonomous Agents and Multi-Agent Systems 18(1), 1–35 (2009)

[6] Bordeaux, L., Monfroy, E.: Beyond np: Arc-consistency for quantified constraints. In: Hentenryck, P.V. (ed.) CP. Lecture Notes in Computer Science, vol. 2470, pp. 371–386. Springer (2002)

[7] Bordeaux, L., Pajot, B.: Computing equilibria using interval constraints. In: Faltings, B., Petcu, A., Fages, F., Rossi, F. (eds.) CSCLP. Lecture Notes in Computer Science, vol. 3419, pp. 157–171. Springer (2004)

[8] Bouhtou, M., Erbs, G., Minoux, M.: Joint optimization of pricing and resource allocation in competitive telecommunications networks. Networks 50(1), 37–49 (2007)

[9] Boutilier, C., Brafman, R.I., Domshlak, C., Hoos, H.H., Poole, D.: Cp-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. J. Artif. Intell. Res. (JAIR) 21, 135–191 (2004)

[10] Brown, K.N., Little, J., Creed, P.J., Freuder, E.C.: Adversarial constraint satisfaction by game-tree search. In: de Mántaras, R.L., Saitta, L. (eds.) ECAI. pp. 151–155. IOS Press (2004)

[11] Ceppi, S., Gatti, N., Patrini, G., Rocco, M.: Local search methods for finding a nash equilibrium in two-player games. In: Huang, J.X., Ghorbani, A.A., Hacid, M.S., Yamaguchi, T. (eds.) IAT. pp. 335–342. IEEE Computer Society Press (2010)

[12] Faltings, B.: Distributed Constraint Programming, chap. 20, pp. 699–729. Handbook of Constraint Programming, Elsevier (2006)

[13] Gottlob, G., Greco, G., Scarcello, F.: Pure nash equilibria: Hard and easy games. J. Artif. Intell. Res. (JAIR) 24, 357–406 (2005)

[14] Harrenstein, P., van der Hoek, W., Meyer, J.J.C., Witteveen, C.: Boolean Games. In: van Benthem, J. (ed.) TARK. Morgan Kaufmann (2001)

[15] Jalaparti, V., Nguyen, G., Gupta, I., Caesar, M.: Cloud resource allocation games. Technical report, University of Illinois at Urbana-Champaign (2010), http://hdl.handle.net/2142/17427

[16] Jiang, A.X., Leyton-Brown, K., Bhat, N.A.R.: Action-graph games. Games and Economic Behavior 71(1), 141–173 (2011)

[17] Kearns, M.J., Littman, M.L., Singh, S.P.: Graphical models for game theory. In: Breese, J.S., Koller, D. (eds.) UAI. pp. 253–260. Morgan Kaufmann (2001)

[18] Leyton-Brown, K., Shoham, Y.: Essentials of Game Theory: A Concise Multidisciplinary Introduction. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers (2008)

[19] McKelvey, R.D., McLennan, A.M., Turocy, T.L.: Gambit: Software tools for game theory (2010), http://www.gambit-project.org

[20] Nash, J.: Non-cooperative games. Annals of Mathematics 54(2), 286–295 (1951)

[21] Papadimitriou, C.H.: The complexity of Finding Nash Equilibria, chap. 2, pp. 29–51. Algorithmic game theory, Cambridge University Press (2007)

[22] Poundstone, W.: Labyrinths of Reason: Paradox, Puzzles, and the Fragility of Knowledge. Anchor Doubleday Publishing Company (1988)

[23] Rosen, J.B.: Existence and uniqueness of equilibrium points for concave n-person games. Econometrica 33(3), 520–534 (July 1965)

[24] Rosenthal, R.W.: A class of games possessing pure-strategy nash equilibria. International Journal of Game Theory 2(1), 65–67 (1973)

[25] Roughgarden, T.: Routing Games, chap. 18, pp. 461–486. Algorithmic game theory, Cambridge University Press (2007)

[26] The Choco Team: Choco : An Open Source Java Constraint Programming Library. Ecole des Mines de Nantes (2008-2013), http://www.emn.fr/z-info/choco-solver/

[27] Turocy, T.L.: Personal communication (2013)

[28] Zhao, L., Müller, M.: Game-sat: A preliminary report. In: SAT (2004)