An approach based on BDI architecture for space occupation

J. Boussaa and M. Sadgal

Computer Science Department,
Faculty of Sciences Semlalia,
University Cadi Ayyad,
Bd Prince My Abdellah BP 2390, Marrakech,
sadgal@ucam.ac.ma

Abstract—Space occupation covers a class of problems as well asserting of constraint satisfaction and optimization approaches. The used methods tend to privilege one of two aspects (optimization or satisfaction) and do not lead to a complete solution. In spite of these methods success in certain problems under constraints, it is interesting to consider new ways of design, in particular to use methods resulting from Artificial Intelligence (A.I) techniques. In order to evaluate possible solutions for this problem, we propose, in this study, to use an approach based on the paradigm of Intelligent Agents. This result in design and implementation of a new problem solver for space occupation under constraints.

Keywords—Space occupation, Constraint satisfaction, Artificial Intelligence, BDI Agent.

I. INTRODUCTION

The problem of space occupation consists of the occupation of objects in a preset space while respecting a certain number of imposed constraints (declared) and others (not declared) which appear at the time of the occupation itself. Of course such an occupation must be optimal.

The mathematical models which tried to deal with this problem often bring back to emphasize the optimization aspect (reduction of occupied space for example) but not directly the satisfaction aspect of constraints. In these models, the enormous difficulty encountered in the constraint expression is circumvented by the search for an evaluation function.

Our problem can be comparable, with a Constraint Satisfaction Problem (CSP), treated on a scale of constraint “severities”. Thus, the preferences will be examined after having satisfied the “severe” constraints. All techniques for problem solving can be integrated: graph exploration, heuristics, improvement by Hill Climbing, simulated annealing and genetic algorithms [17].

The objective of this article is to present a new approach based on cognitive BDI agents (Belief-Desire-Intention) architecture to express easily and naturally the constraints and to solve the problem using the A.I techniques: reasoning, planning, inferences....

The following section presents the state of the art in this field. Section 3 exposes the theoretical and practical aspects of BDI agents used in our approach. We proceed to clarify the new approach in part 4. Section 5 will be devoted to illustrate our system by an example. Finally we conclude on an optimist note about work to come.

II. RELATED WORKS

A. Space occupation problem

A.1. Formulation and objectives

Space occupation recovers mainly many fields of design such as: pieces fitting of furniture, architecture, cutting, placement/routing problems in: electronics, vehicles, boats, factories,...

This space occupation problem is defined as follows: Being given a space of limited places that we will indicate by E, and a set of objects subjected to constraints. The problem is to find sites (places), orientations and dimensions of the objects in this space to satisfy the constraints by realizing objectives as well as possible. Classically, it was thought that the problem is reduced in the majority of the cases to the minimization (or maximization) of an evaluation function which one must choose well.

A.2. Resolution: Various approaches

To try to solve this problem, several approaches were used. We can quote them here in three main categories:

- The constructive approach is a top-down category approach: Fletcher [5] and al. used an algorithm containing “generate and test”. Baykan and Fox [4] carry out a research directed by the constraints.
- The iterative approach tries to improve a placement starting from an existing placement by moving an object or by permuting two objects: Shaookar and Mazunder [12].
- The hybrid approach is a coupling, more or less extremely, of the two preceding approaches. Watanabe and al. [14] conceived an expert system for a room of computers fitting. Donikian and Hégron [3] developed a descriptive method dedicated to the design of drafts of architectural environments.
B. Techniques of constraint satisfaction

B.1. Traditional techniques

A CSP (Constraint Satisfaction Problem) is a problem modeled in the form of a constraint set on variables, each one of these variables taking its values in a field. In a more formal way, one defines a CSP by a triplet (X, D, C) such as:

- \( X = \{X_1, X_2, \ldots, X_n\} \) is a set of variables (unknown factors) of the problem.
- \( D \) is the function which associates with each variable \( X_i \) its field \( D(X_i) \), i.e. the set of the values which \( X_i \) can take.
- \( C = \{C_1, C_2, \ldots, C_k\} \) is a set of constraints. Each constraint \( C_j \) is a relation between certain variables of \( X \), restricting the values which these variables can take simultaneously.

This representation permits to find one or more solutions (variables assignment) satisfying constraints, without optimality.

B.2 Resolution of a SCSP (Space CSP)

The basic algorithm use the chronological backtrack [6]: it is an in-depth research initially with backtrack chronological and test of the constraints when their variables are instantiated. Many research was undertaken to improve this algorithm, like the anticipation and the introduction of the consistency graph [8, 10]. The first research orientation consists in propagating the constraints as soon as possible for quickly detecting inconsistencies. It is one of the keys to the success of constraint programming.

A second axis relates to backtrack itself. The basic mechanism is chronological: in the failure case, one reconsiders the last choice, without worrying to know if this choice has any responsibility in the current failure. However:

- There are systems which do not admit a solution.
- There are problems which one cannot represent without losing of information (unreliable representation)
- There are constraints which one cannot represent formally (mathematically)
- The traditional methods (CSP) does not deal with the optimality problem.

The problem becomes thus NP-Complete, from where need for distributing the treatment between several intelligent entities: Distributed Artificial Intelligence.

III. BDI Architecture and 3-APL Platform

We used the 3APL platform (“triple-a-p-l”) [1] to implement our approach for practical reasons on the level of development and prototyping.

A. Intelligent agents.

Generally, an agent receives perceptions of its environment by the means of its sensors and acts as return on its environment by the means of its effectors (Fig. 1). It can be more or less accessible, i.e. in an accessible environment the agent has access to complete information, precise and up to date of the state of the environment.

![Fig. 1 general Sight of an agent](image)

B. BDI Agent

Many characterizations of BDI model (Belief-Desired-Intention) were proposed. One finds of them formulations logical, such as those produced by Rao and Georgeff [11] as much as formal specifications [9] or procedural [16]. It is this type of procedural vision, nearer to the implementation, which will be useful for us within the framework of our data-processing validation. A BDI architecture is then a good candidate to model the behavior of a rational intelligent agent [15].

C. 3APL Platform

3APL (An Abstract Agent Programming Language “triple-a-p-l”) [1] is a programming language oriented agent that was developed by Koen Hindriks [7] within the framework of his thesis at the Utrecht university and which was the subject of continuous developments since then [4]. The language 3APL permits to program cognitive agents in term of beliefs, actions, plans, goals and rules of reasoning (practical reasoning rules). The language 3APL offers facilities to the implementation of these mental attitudes and the processes of deliberation that handle them.

C.1. 3APL agent components:

Each agent has its own interpreter, using a PROLOG engine [2], which determines the behavior of the agent. The agent architecture is illustrated by figure (Fig. 2).

Beliefs: The beliefs describe the situation in which the agent is. The base of beliefs contains information that the agent has on the world.

Goals: Each agent acts on its environment using goals which it must try to satisfy. The beliefs and the goals constitute the mental state updated during its execution.

Elementary actions: In order to satisfy its goals, an agent uses plans. A plan is composed of an elementary actions set (BASIC actions).

The base of actions is static: The mental actions are used to update the agent belief base. They are made up of a name, preconditions and postconditions.

Plans: A plan is a composition of elementary actions being able to combine operators of sequence, loop or conditional choice with the manner of an imperative language.

Reasoning rules: In order to reason on the goals and the plans, 3APL offers two types of rules, the goal planning rules.
of goals and the plan revision rules. These rules are also static [1].

![Diagram of 3-APL agent architecture](image)

**Fig. 2 the 3-APL agent architecture [9].**

### C.2. A sample example of 3APL agent

```prolog
PROGRAM "agent_placement" % name of the program
LOAD "process_base.pl" % load of a prolog file which will be placed in the section

CAPABILITIES {
  % definition of the elementary basic actions.
  {Pos (X, Y)} Goto (X1, Y1) {NOT pos (X, Y), pos (X1, Y1)},
  { TRUE} est_place () {se_deplacer ()}
}

BELIEFBASE {
  % knowledge base
  Mon_module (1),
  pos (0, 0)
}

GOALBASE {
  se_deplacer ()
}

PG-RULES {
  % rules appealing to plans to carry out the purposes
  se_deplacer () < - mon_module(n) AND pos (X, Y) {chercher_place (n, X, Y),
      Goto (X1, Y1); IF NOT pos (X, Y)
      THEN est_place ()
  }
}
```

### IV. SUGGESTED APPROACH

We sought to develop a system allowing a strong expression of constraints and to implement a resolution based on the deliberation and the reasoning. That is made possible by using the predicate logic and the cognitive agent properties like BDI.

In these approaches, one is interested in the resolution of the problem by satisfaction of constraints initially. They thus should be represented by an exploitable internal structure of the solver system which must reflect "accurately" their natural expression.

To illustrate the data processing quoted previously we will present a placement system (a solver). It is about prototype RABDI (Solver with BDI Architecture) produced with an aim of developing a solver in various fields on the one hand and of improving the data structure as well as the resolution mode on the other hand.

RABDI is a prototype of placement in the plan. This introduce to use the agent paradigm for solving the placement problems in many Fields. It offers a solution space from which we can extract interesting ones by heuristics.

### Placement data:
Get E a space of places in the plan, O a set of geometrical objects called modules to be placed in E, C is a set of constraints, and B is a set of objectives.

### Problem:
- The sought goal is to lead to an occupation of the modules in E by satisfying the C set and by respecting the B set as well as possible. Although the geometrical model (object shape) used in RABDI is reduced to the rectangle and that the constraints use only the space aspect (geometrical, topological.) between objects, the system covers a broad class of applications.

### A. Description of prototype “RABDI”

#### A.1. The architecture of the system.

The general architecture of RABDI and its environment is presented by the figure (Fig. 3) below:

![Prototype RABDI](image)

**Fig. 3 prototype RABDI (seen on the user part)**

#### A.2. Knowledge description (BELIEFS).

The platform 3APL is supporting completely language PROLOG [2]. That allows a direct use of PROLOG control mechanism combined with agent BDI organization.

The syntactic form of an assertion in PROLOG:

\[ \text{nom_predicat (A}_1, A_2, \ldots, A_p) \]

that expresses a relation between objects represented by \( A_1, A_2, \ldots, A_p \) (variables and/or values).


The data are expressed directly using the predicates:

- **Space E:**
  He is regarded as a rectangle (Fig. 4) of origin \((X_0=0, Y_0=0)\) and of end \((XE, YE)\) defined by the user. He is represented by the predicate: \( \text{carte}(XE, YE) \). \( XE \) and \( YE \) are the \( E \) end coordinates. Space \( E \) is subdivided in a grid on which the positions of the modules are given.
The point coordinates: \((X, Y) \in E; \quad X \geq X_0, Y \geq Y_0; \quad X \leq XE \text{ and } Y \leq YE.\)

**Fig. 4 placement Space and Modules.**

- **Modules:**
  They are rectangular forms. Information concerning these modules (geometry, position and orientation) is represented by the following predicates:
  - `geom (N, Lo, La):` module dimensions, \(N\) is the module name, \(Lo\) is the length, is to \(La\) the width.
  - `pos (N, X, Y):` position of the module \(N\), \(X\) and \(Y\) are coordinates in \(E\).
  - `orients (N, O, State):` \(O \in \{\text{south, east, west, north}\}\), it is the orientation compared to the axis \(X\). \(State \in \{\text{mobile, fixe}\}\), it authorizes or prohibits the module \(N\) to be oriented.

- **Relations:**
  We express constraints by:
  - Three unary relations:
    - Fixed position: \(i.e\) `pos (N1, 30, 40)`.\n    - Fixed orientation: \(i.e\) `orients (N2, north, fixe)`.
    - Closed area: \(i.e\) `zone_pro (N1, 20, 20, 40, 50)` area closed for module \(N1\).
  - Two binary relations:
    - Distance between two faces.
    - Adjacency of two faces.

This is expressed by using only one predicate `en_regard (N1, N2, F1, F2, D)` which means:

- The \(F1\) face of the \(N1\) module must be compared to the \(F2\) face of the module \(N2\) at a distance \(D\) according to its sign:
  - \(D \geq 0 \Rightarrow \text{by higher values (at least } D)\)
  - \(D < 0 \Rightarrow \text{by lower values (at most } D)\)

According to the anonymity of certain variables (noted “_” in PROLOG) one can have several interpretations of this predicate.

**Examples of constraints:**
- `en_regard (_, _, _, _, +5)` expresses the constraint: all the modules must be spaced by a minimal distance of 5 units.
- `en_regard (N1, N2, F1, F2, _)` expresses: The face \(F1\) must be at an unspecified distance from \(F2\) (but must remain in faces) for the modules \(N1\) and \(N2\).

- \(en_regard (N1, N2, _, _, -5)\) expresses: \(N1\) and \(N2\) are distant of to the more 5 units.

**A.2.2. Rule Expression.**

The resolution procedures are expressed using the Horn clauses under PROLOG. One is pressed directly on the PROLOG control mechanism.

A procedure is thus a collection of predicates of the type: \(p (\ldots) \Leftarrow p_1(\ldots), p_2(\ldots), \ldots, p_n (\ldots)\) that means in the 1st order logic: \(p_1(\ldots) \land p_2(\ldots) \ldots \land p_n (\ldots) \Rightarrow p(\ldots)\).

To prove \(p\) PROLOG tends to prove \(p_1, p_2, ..., p_n\) by using the same procedure (resolution) to prove the \(p_i\) predicates.

**B. Resolution (Plans).**

We constructed treatments (Fig. 5) exploiting the data in the preceding formalism to obtain a correct placement in space \(E\). Many Plans are developed.

**B.1. Plan: Constraint rewriting.**

The relations “\(en_regard\)” are rewritten with replacement of the anonymous variables of their the first two arguments by all the possible values. This rewriting makes it possible to extract a subset from binary constraints for each module.

**B.2. Plan: Modules scheduling**

Certain modules can have priorities in the placement. These priorities are based on criteria fixed by the user. For example: “START TO PLACE THE MODULES OF BIG SIZE.” is a criterion which results in a module sorting by a decreasing order of their size. Generally a weight is affected for each module and calculated starting from the criteria specified by the user. Then scheduling is done according to the order increasing or decreasing of this weight. The modules are put in an ordered list (first order is static).

**B.3. Plan: Satisfaction Zones (\(E_N\))**

Initially a subset space \(E_N\) is sought for a module \(N\) to be placed. In \(E_N\), the constraints weighing on the module \(N\) and those already placed are checked and satisfied. Then, a place in \(E_N\) for the module \(N\) is determined.

\(E_N\) is determined in the following way:

For each module \(N_i\) already placed and each \(C_{ik}\) constraint expressed (between \(N_i\) and \(N\)) by:

\(en_regard (N_i, N, F_i, F_k, D_{ik})\) \(i\) possible different faces \(F_i\) of \(N_i\) and different faces \(F_k\) of \(N\). The plan determines a \(E_{N_{ik}}\) zone where it is satisfied. The strategy pursued by this plan is the **obligatory** satisfaction of all the constraints weighing on \(N\). This satisfaction is possible in the zone intersection of all the \(E_{N_{ik}}\) zones.

The subspace \(E_N\) is thus given by the relation:

\[ E_N = \bigcap_{i,k} E_{N_{ik}} \]

**B.4. Illustration:**

\(N_3\) to be placed,
\(N_4, N_5, N_6, N_7\) are already placed.

**Constraints:**
\( C_{311} = \text{en}_\text{regard} (N_1, N_5, \text{east, west, } +5) \),
\( C_{311} = \text{en}_\text{regard} (N_3, N_5, \text{west, east, } +3) \),
\( C_{411} = \text{en}_\text{regard} (N_4, N_5, \text{south, north, } +3) \).

We introduced the predicate:
\( \text{interpl}(NR, Xz0, Yz0, Xzf, Yzf) \) to represent \( E_N \). \( \text{interpl} \) is quite simply the envelope of \( E_N \) (rectangle) which can contain obstacles. Those are areas closed by \text{en}_\text{regard} relations during the placement or given by the user.

**B.5. Positioning Plan.**

The placement itself is carried out by a heuristic function which consists in finding a place in the space represented by \( \text{interpl} \). The function seeks a position while avoiding the obstacles (forced if overlapping not permitted). In the success case, the position of the module is returned: a \( \text{posit} (N, X, Y) \) will be added; when there is failure, it immediately calls in backtracking mode a substituted plan to reconsider the earlier positions.

This plan develops an ‘or’ graph of States according to the strategy depth initially with the manner of prolog. The default backtrack is systematic (chronological), but it is ameliorated by anticipation (determination of \( E_N \)s of all the modules not yet placed) and by using a dynamic order : choice to place the module that have the smallest \( E_N \). That is more convenient in a system multi-agents of BDIs where each agent dealing with module can announce the widening or the contracting of its zone of possible places \( \text{interpl} \) when a module is being placed.

V. AN APPLICATION EXAMPLE

We present in this part an application example allowing to know what \text{RABDI} can treat and to appreciate the flexibility of constraint expression introduced by the approach of the A.I. That will lead us to present some performances and to realize of some limiting that we detail in the conclusion part.

**Problem:**

“We have a square zone of a surface 200x200 m\(^2\). We want to place 8 objects (numbered from 1 to 8) of different sizes. A ditch of width 5 m is envisaged with a height of 50m from the origin to the zone extremity (in width). We must, at least, leave a 5m space between objects (including the ditch). Object 3 is surrounded of the objects 4 and 7 which must be in front of its northern face respectively at a minimal distance from 20m (northern face of 4) and a maximum distance from 10m (face Eastern face of 7).”

**Question:** How much will be able one to reduce occupied space when all objects are placed?

**Fact Base expression:**

That results in the following fact base (the ditch is represented by module 9 with fixed position):
- \( \text{carte} (200,200) \). (represents a size rectangle of the space 200x200),
- \( \text{geom} (1,30,30) \). (geom (N, Lo, La): represents dimensions of module N ),
- \( \text{geom} (2,10,40) \).
- \( \text{geom} (3,40,60) \).
- \( \text{geom} (4,50,60) \).
- \( \text{geom} (5,20,40) \).
- \( \text{geom} (6,10,80) \).
- \( \text{geom} (7,10,10) \).
- \( \text{geom} (8,40,80) \).
- \( \text{geom} (9,200,5) \).
- \( \text{posit}(9,0,50) \). (fixed position of the module No 9 (ditch) at (0,50))
- \( \text{en}_\text{regard} (_, _, _, _, +5) \). (to leave a distance of 5 m between the modules),
- \( \text{en}_\text{regard} (3,4, N, N, +20) \). (the faces Northern of 3 and 4 must move away more than 20 m),
- \( \text{en}_\text{regard} (6,5, E, S, +50) \). (faces East of 6 and South of 5 must move away more than 50),
- en_regard (7,3, E, N, -10). (Not exceed 10 m between 7 face East and 3 North face),
- pas_conseille (2,2): (grid of positions to step of 2m),
- ordrep (“forced”): (order based on the number of constraints weighing on a module),
- place_sens (‘x’): (displacement according to the x axis initially).

This PROLOG file is loaded by our BDI agent at start in its base BELIEFBASE.

Some options can be modified by the user:
- pas_conseillé: specify the minimal step used by the system in the grid.
- orderp: order recommended for the placement of the modules.
- place_sens (‘x’): the direction of displacement used by the system for displacements on the grid of positions according to the axes (X or there).

The plugin: WidowPlugin programmed by our care with the assistance the API java offered by 3APL (Fig. 7): graphic environment in which the agent places space E and the objects. agent BDI: one can distinguish the various components from the agent and mainly: section CAPABILTY, BELIEFBASE, GOALBASE, PLANBASE, PG_RULES and PR_RULES.

Note: The elementary plans and under-plans are loaded with agent BDI.

Results: The execution of this agent produces the following results (Fig. 6):

It is noticed that at the end of the execution the BELIEFBASE contains the found positions (pos (N, X, Y)) for the modules, the PLANBASE and GOALBASE are empty (no more to carry out)

The figure (Fig. 7) illustrates the result in the graphic environment

![Fig. 7 graphic environment: the placement result](image)

The space is the large framework (square 200x200) in blue, the fixed module (ditch) is rectangle 9 in blue, the other modules are green. The features in the middle of the rectangles symbolize their Southern face. Thus modules 3, 6 and 7 were directed following the constraints.

The solution suggests that one can reduce the space size to the limit of the module place number 3, 40m and reduce the width to the limit of module 5, of 15m. Because the large ditch, the only bottom part will be reduced. Then the reduction is (15*200 + 40*145)/200*200 = 22%.

The current placement occupies 78% of entire surface. To optimize the occupied space, a simple heuristic (in Positioning Plan) is used: ‘place a module in a position that minimize the distance from the space (chart) origin’.

VI. CONCLUSION

The proposed RABDI system permit the express the constraints in a great simplicity and in a naturally way. The choice of criterion of scheduling and the choice of final solution constitute an aspect of the flexibility offered by the system. The system is open, it is able to integrate other forms of constraints expressed in predicates. Moreover, it is possible to compare suggested solutions by integrating objectives in the position choice.

In spite of this performance, real limits still exist. The representation of the geometrical objects in rectangles does not permit to express all the geometrical and topological constraints necessary for certain problems. The placement plan is limited to one algorithm.

To surmount these difficulties, we plan to introduce distribution for resolution: that makes it possible to solve the problem by various architectures of agents BDI (more than an algorithm in a competition way).

REFERENCES

Systems: Languages, frameworks, techniques, and tools (ProMAS03), July 2003.


[9] Inverno, Mr., Luck, Mr., Georgeff, Mr., Kinny, D., and Wooldridge, Mr. (2004). The dmars architecture: With speciation off the distributed multi-agent reasoning system. Newspaper off Autonomous Agents and Systems Multi-Agent, 1-2 (9):5.53.


J. Boussa is a PhD student in Computational Intelligence and constraint satisfaction problems at Cadi Ayyad University. She received her B.A. and DESS degree in computer science from Cadi Ayyad University. She is an assistant for computer science training in Industrial world.

M. Sadgal, received the Ph.D. degree in Computer Science from the University of Lyon in 1989. He received the Ph.D degree in computer vision in 2005. He is currently a Professor (since 2002) at Cadi Ayyad University (Marrakesh, Morocco). From 1985 to 1987 he was associated Searcher at Lyon I (France). He was Project Leader Engineer for NET, CAD (Computer Aided Design) and CAM (Computer Aided Manufacturing) from 1987 to 1994 at CONSEPT Society (France). His research interests include Computer Vision, Artificial Intelligence and Multi-agent Systems.