Constraint-Based Design in Participatory Housing Planning

Dirk Donath[1], Luis Felipe González Böhme[2]
[1]Bauhaus-Universität Weimar, Germany and [2]Federico Santa María University of Technology, Chile.
[1] caad@archit.uni-weimar.de, [2] luisfelipe.gonzalez@usm.cl

The research presented in this paper deals with the yet unexplored development of a constraint-based design strategy to support participatory housing planning processes in Latin America. The article discusses the implementation criteria of a constraint satisfaction approach to solving the building bulk design problem. This elementary problem to the architecture practice, is concerned with the synthesis of the boundary geometry from the volume, shape and allocation of the building and any part thereof located inside a given zoning lot. A legal solution to a building bulk design problem is a building cubature that complies with all the applicable bulk regulations. The case study applies to the common class of single-family house units produced in Chile and the regulatory framework implemented there. Two different computer implementation criteria are being tested in an ongoing series of trials. The first, and most extensively developed, makes use of Maxon's XPresso® visual scripting environment to set up a semi-automated controllable design environment that allows to create parametric feature-based 3D models of building bulk solutions. The second approach is currently being tested by using Ilog's OPL Studio® constraint programming environment to achieve fully automated search and 2D graphic visualization of the complete set of solutions to separate subdomains of the bulk problem.

Keywords*: Constraint-based design; constraint satisfaction problems; building bulk design; participatory planning; low-income housing.*

Introduction

Participatory planning and decision-making are the strategic means for contributing to the achievement of the poverty reduction targets set for 2015 by the United Nations (UN-Habitat, 2002). Self-management housing, in its many variants, has proved cost-effectiveness in reducing the shortage and increasing the quality of low-income housing in many developing countries. The call for dwellers collaborative involvement in Chile – Latin America's oldest housing policy (Hidalgo, 1999) – goes back to the 1950s (Green,

2004) and becomes stronger today supported by the nationwide Housing Solidary Fund (www.fsv.cl: April 2006). Chilean participatory model addresses poor households and runs exclusively on a competitive project-based funding scheme. Housing subsidy applicants are expected to take the initiative in self-organizing and co-developing concrete projects with an authorized NGO. Thus, dwellers and planners compete conjointly against other applicants for the financial backing of their own developed project. The current scheme reveals itself as the only way for the poor households to access homeownership in the future, and probably not only in Chile. However, the definition of civic participation is pretty vague yet, even for many state agents (Surawski, 2005). The Unesco (2005) has detected that in order to achieve truly participatory and inclusive decision-making processes at all levels, current criteria, standards, requirements, language and procedures must be simplified. In this highly competitive scenario, efficiency regarding time and cost of planning directly affects dwellers' quality of life, especially when lacking in adequate shelter during the project development time. The early incorporation of regulatory requirements into the design project certainly allows to stand a better chance in the competition for the financing funds. Additionally, lawful design solutions avoid delays in the permit process.

The building bulk design problem

In the architecture practice, "bulk" is the term used to describe the three-dimensional size and shape distribution of buildings or other structures, and their topological relationships to each other and to open areas and lot lines. The general purpose of all bulk regulations in its combined effect, is to improve urban habitability standards by guaranteeing basic quality criteria, like e.g. proper solar access, ventilation and sufficient open space. The elementary set of bulk regulations includes:

(i) The lot coverage coefficient, is a real value ranging over [0..1] which, when multiplied by the lot area, produces the maximum amount of ground floor area allowed to build on the lot.

(ii) The constructability coefficient, is a real value ranging over [0..n] which, when multiplied by the lot area, produces the maximum amount of total floor area allowed to build on the lot.

(iii) The building height, determines the maximum vertical distance allowed between the natural ground level and the highest point of the building.

(iv) The story height, determines the minimum vertical distance allowed between the finished floor surface and the finished ceiling surface immediately above.

(v) The setback, determines the minimum horizontal distance between a lot line and the nearest point of a building or any projection thereof, excluding projecting roof structures, eaves, beams, windowboxes or canopies.

(vi) The zero-lot line length, is a real value ranging over [0..1] which, when multiplied by the length of a side lot line abutting an adjoining zoning lot, produces the maximum horizontal distance allowed to build directly on that side lot line, without encroaching upon transverse setback lines.

(vii) The zero-lot line height determines the maximum vertical distance allowed between the natural ground level or the mean level between adjoining lots and the highest point of the zero-lot line building side.

The complete set of Chilean bulk regulations (Minvu, 2004) includes a lot more provisions and conditions, but these exposed above, include the most influential control variables affecting the bulk design of any conventional single-family house unit produced in that country today. Bulk regulations do not work isolated, they all are related to each other with distinct dependence degree, turning the solution finding into an expert task. The very structure of the building bulk design problem reveals a complex system of non-redundant combination of all applicable bulk control variables.

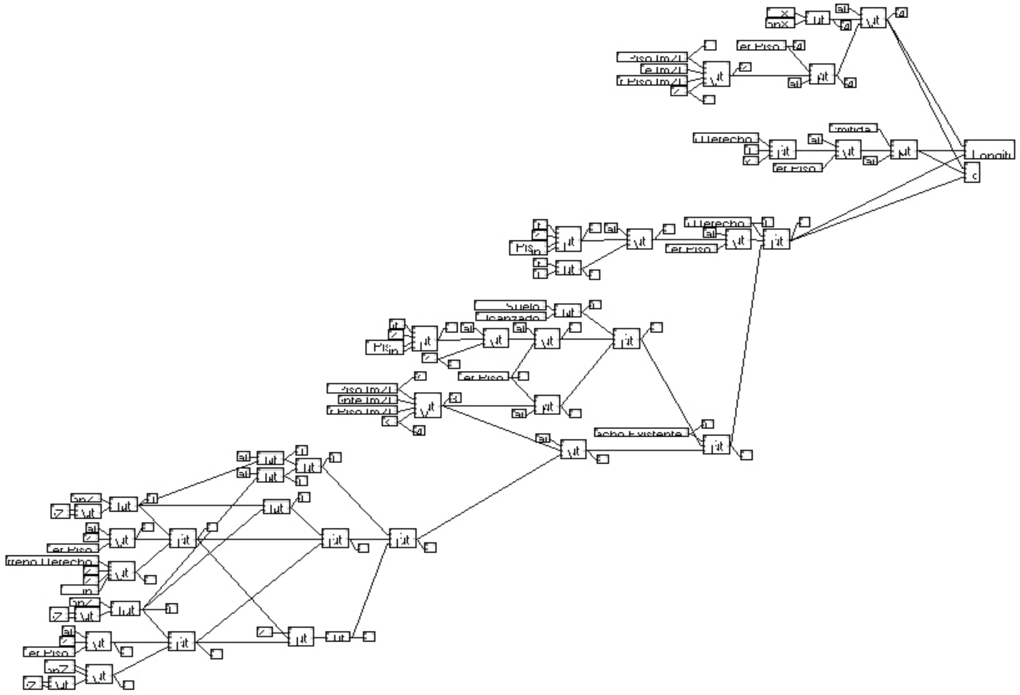


Figure 1
Search space for size of a zero-lot line building part

Conventional criteria to solve the building bulk design problem

There are basically two solution-finding methods the architects are used to apply: (a) a top-down method we may call "reductive" and (b) a bottom-up method we may call "constructive". The reductive method consists of sketching a preliminary cubature for the putative building and then to "prune" it by replacing the original configuration values with some allowed by the applicable bulk regulations. A popular and useful heuristics is to choose first extreme values, wherewith the search space becomes considerably smaller. The constructive method instead, begins with directly choosing allowed values for each configuration variable of different parts of the putative building. These parts coincide with those into which the bulk regulation itself semantically decomposes the building cubature on a zoning lot, like e.g. a zero-lot line part of the building or a second floor detached part thereof. Anyhow, both methods include an evaluation cycle during which some tradeoffs must be made between the assigned values that specify the shape distribution among the building parts. Whatever the method may be, the conventional procedure to produce legal solutions for common building bulk design problems consists of the following general steps:

1. Create an instance of solution to the building cubature (or part of it) by assigning a value to each configuration variable that specifies that instance in the three-dimensional space unambiguously.
2. Loop until every assigned value are labeled LEGAL: (a) Verify whether the assigned value violates any bulk regulation.

    i. If the assigned value does not violate any bulk regulation, label it as LEGAL.
    ii. Otherwise, assign a new value and return to step (a).
3. Evaluate the legal solution to the building cubature according to additional criteria (e.g. cost, aesthetics etc):
    i. If the legal solution does satisfy most of those criteria, quit.
    ii. Otherwise, return to step 1.

Eventually, step 3 may precede step 2. In any case, the exhaustive verification loop (here, at step 2) may consume a lot of time. In participatory housing planning, the availability of time, human and technical resources to try additional solution alternatives is rare.

The constraint-based design approach in architectural planning

Constraint satisfaction reflects exactly what the architectural design process really is about. Design constraints represent many different requirements a design solution must meet, regarding e.g. customer needs and preferences; planning and zoning regulations; aesthetic, economic and structural specifications etc. The idea of describing the design as the exploration of alternative sets of constraints and of the regions of alternative solutions they bound (Gross, 1986), opens new, more efficient ways to implement automated reasoning in architectural design. The philosophy of the constraint-based design approach is based on the paradigm of constraint satisfaction problems (CSP).

A constraint satisfaction problem (or CSP) is defined by a set of variables, X_1, X_2, …, X_n, and a set of constraints, C_1, C_2, …, C_m. Each variable X_i has a nonempty domain D_i of possible values. Each constraint C_i involves some subset of the variables and specifies the allowable combinations of values for that subset. A state of the problem is defined by an assignment of values to some or all of the variables, {X_i = v_i, X_j = v_j, …}. An assignment that does not violate any constraints is called a consistent or legal assignment. A complete assignment is one in which every variable is mentioned, and a solution to a CSP is a complete assignment that satisfies all the constraints (Russell & Norvig, 2002 p. 137).

Architectural planning is basically a two-step problem solving process consisting of programming and design. The task of architectural programming is to specify a set of constraints or "conditions" describing the design problem in as accurate a way as possible. The aim of architectural programming is to provide a specifically enough statement of the given design problem (Peña, 2001). As mentioned above, the architectural design task is to satisfy all constraints describing the problem structure.

In this sense, architectural design may also be considered as a two-step process consisting of propagating the available constraints throughout the implicit network of dependencies between them, and to search for the or a solution by introducing new constraints to the problem. Dependencies between constraints are usual, and the reason for that is because many constraints involve more than one variable of the problem and many variables participate in more than one constraint (Rich & Knight, 1991). Constraint propagation, the first task of designing, is to conjugate the known constraints to try to find a solution to the problem by only exploiting a well enough specified description (if there is one) of the problem structure. As constraints are propagated, additional constraints may be inferred from those that were originally given as part of the problem statement. That leads to a reduction of the domain of possible values for many variables. Constraint propagation terminates for one of two reasons: (a) a contradiction may be detected, e.g. because the problem has been over-constrained, or (b) it becomes impossible to make further changes on the basis of the current knowledge, whether any solution has been found or not (Rich & Knight, 1991). If no solution has been found thereafter, search may begin.

Search demands to make a guess about a value for a particular variable which may lead either to a contradiction (inconsistency) or to the generation of additional constraints. Notice that if the design does not provide any convincing solution or at least a legal one, programming begins again and so on. In such terms, architectural planning may consist of the following general steps:

1. Architectural programming.
   (a) Identify the minimum number of variables required to describe the design problem and to define a complete solution to that problem.
   (b) Specify a number of constraints on subsequences of these variables.
2. Architectural design.
   (a) Propagate the available constraints until the domain of every known variable has been reduced to its minimum range:
   i. If the combined effect of all constraints defines a solution, quit and proceed to evaluate that solution.
   ii. If the combined effect of all constraints defines an inconsistency, report failure and return to step 1.
   iii. Otherwise, proceed to step 2(b).
   (b) Search until a solution is found or all possible solutions have been discarded:
   i. Select a variable whose value is not yet determined, strengthen as much as possible the set of constraints that apply to that variable and return to step 2(a).

Modeling a problem by means of constraints is natural to the architect, whenever it comes to describing as much the complete problem as partially the or a solution of it. The more accurately the problem structure is described, the easier it will be to solve it. In these terms, the act of designing may be simplified to the straightforward assignment of values to a well-defined sequence of problem variables. The alone task of assigning values is not further considered an expert task and therefore it can be shared and discussed by laymen and experts concurrently. That is how constraint-based design may actually support truly participatory planning at all levels.

Computer implementation strategy

We are trying different implementation strategies, considering all aspects informed above. The basics of any strategy required to achieve the mechanical solving of a problem demand to define its structure accurately enough to reduce the problem to a selection problem (Minsky 1956 cited in Alexander 1967). The most important advantage of a CSP is that its standard representation already reveals the structure of the problem itself.

Since the search space containing all the conceivable design solution alternatives is too big to be explicitly represented, the solution space can be implicitly represented by means of a constraint

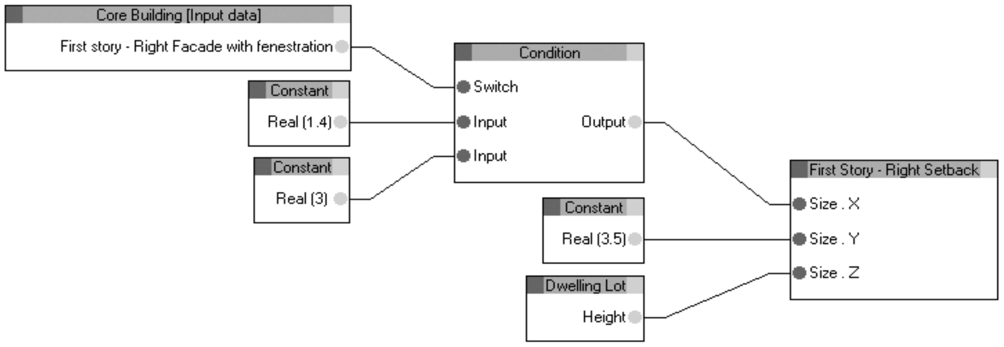


Figure 2
Right side setback constraint

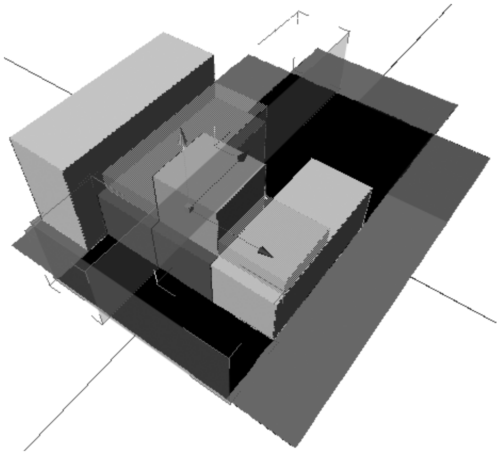


Figure 3
Sample of bulk design units

system. Kramer (1992a) defines a constraint system as the collection of geometric entities and the constraints that describe how these entities interact with each other. Solving a geometric CSP is to find the positions, orientations, and dimensions of these geometric entities, so as to satisfy all constraints simultaneously. We adopted Kramer's point of view but renaming the geometric entities as "units".

The first prototype

For the implementation trial on Maxon's XPresso® visual scripting environment, we defined six classes of units required to specify the general building bulk design problem. We represent each unit as a rigid body (Kramer, 1992b), such that the sequence of six configuration variables *P_x_i*, *P_y_i*, *P_z_i*, *S_x_i*, *S_y_i*, *S_z_i* is considered as the minimum number of parameters required to specify a unit *i* in the three-dimensional space unambiguously. The subsequence *P_x_i*, *P_y_i*, *P_z_i* represents the three translational DOFs (degrees of freedom) and the subsequence *S_x_i*, *S_y_i*, *S_z_i* the three dimensional DOFs for any unit *i* involved in the design problem. The following mathematical model to represent the building regulations exposed above is still in refinement due to the different technical requirements of each implementation platform. Due to the size of the mathematical model required to represent the building regulations exposed above, we may only show an excerpt thereof.
Considering the following definitions (ineq. 1 to 3):

The floor area of a unit i is
A_Floor_i = S_x_i * S_y_i (1)

The sequence of ground floor units is
U_GF = {U_1, ..., U_(a-1)} (2)

The sequence of upper floor units is
U_UF = {U_a, ..., U_b} (3)

we may state the following examples of bulk constraints (ineq. 4 and 5):

The lot coverage constraint is
Sum(i = 1; a-1) A_Floor_i <= A_Floor_j*lambda (4)

where the subindex *j* represents the lot unit and *lambda* is the lot coverage coefficient.

The constructability constraint is
Sum(i = 1; b) A_Floor_i <= A_Floor_j*kappa (5)

where the subindex *j* represents the lot unit and *kappa* is the constructability coefficient.

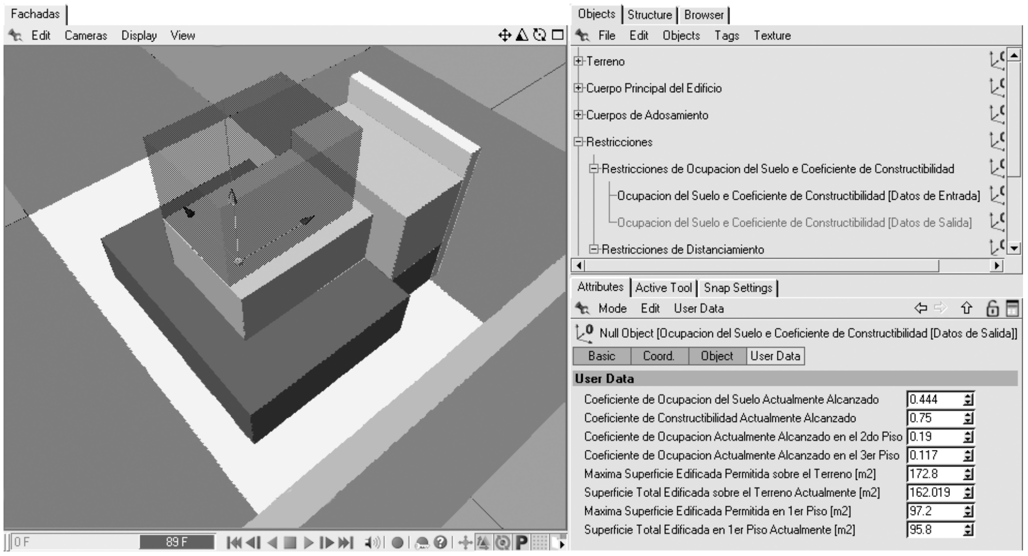
The building height constraint is
S_z_j <= h_Bmax_j (6)

where h_Bmax is a constant specifying the maximum allowable building height.

The implementation on Maxon's Cinema 4D® supports the constructive method to produce one parametric feature-based 3D model of building bulk at a time.

The result of modeling the constraints using Maxon's XPresso® visual scripting language was an implicit representation of the constrained search space itself. The resulting constraint digraphs display the complexity of the problem structure.

Figure 4
Prototype running on Cinema 4D


Fachadas
Edit Cameras Display View
Objects Structure Browser
File Edit Objects Tags Texture
Attributes Active Tool Snap Settings
Mode Edit User Data
Basic Coord. Object User Data
0 F
89 F

Terreno
Cuerpo Principal del Edificio
Cuerpos de Adosamiento
Restricciones
Restricciones de Ocupacion del Suelo e Coeficiente de Constructibilidad
Ocupacion del Suelo e Coeficiente de Constructibilidad [Datos de Entrada]
Ocupacion del Suelo e Coeficiente de Constructibilidad [Datos de Salida]
Restricciones de Distanciamiento

Null Object [Ocupacion del Suelo e Coeficiente de Constructibilidad [Datos de Salida]]

User Data

Coeficiente de Ocupacion del Suelo Actualmente Alcanzado	0.444
Coeficiente de Constructibilidad Actualmente Alcanzado	0.75
Coeficiente de Ocupacion Actualmente Alcanzado en el 2do Piso	0.19
Coeficiente de Ocupacion Actualmente Alcanzado en el 3er Piso	0.117
Maxima Superficie Edificada Permitida sobre el Terreno [m2]	172.8
Superficie Total Edificada sobre el Terreno Actualmente [m2]	162.019
Maxima Superficie Edificada Permitida en 1er Piso [m2]	97.2
Superficie Total Edificada en 1er Piso Actualmente [m2]	95.8

The second prototype
The series of trial with Ilog's optimization modeling system OPL Studio® can only provide two-dimensional graphics, but its built-in solver is able to find and display the complete sequence of solutions for a given problem automatically. However, up to now, we are only able to solve separate subdomains of the building bulk design problem, due mainly to the lack of three-dimensional visualization. A sample of how constraints may be stated using OPL Studio follows:

The setback constraint group

```
forall(i in 1..Nglr) {
  Px[i] >= DistL;   (1)
  Py[i] >= DistF;   (2)
  Px[i]+Sx[i] <= Sxlot-DistR;(3)
  Py[i]+Sy[i] <= Sylot-DistB;(4)
};
```

where the variable *Nglr* controls the total number of rooms on the ground level; the variables *DistL*, *DistF*, *DistR* and *DistB* represent the left, front, right and rear setback distances respectively, and *Sxlot* represents the lot width and *Sylot* the lot depth.

Let us suppose we want to find all possible ways of covering 30% of a 10x10m square lot, having three ground level rooms of variable size ranging over 3m to 10m. Furthermore, there are setback requirements of 1m to the front, 2m to the left, 3m to the rear side, and 2m to the right side of the lot. We modeled 37 constraints defined on 14 variables and the Ilog's solver found 72 solutions for that specific problem in 2.19 seconds by using depth first search.

Conclusions

One of the greatest advantages of OPL Studio (van Hentenryck, 1999) is the fact that it is a declarative language. The declarative nature of constraints has many benefits: the order in which constraints are imposed has no importance and additional constraints, that capture new properties of the solutions, can be added without worrying about the interaction with existing constraints and the search procedure. The contrary case is the prototype implemented on

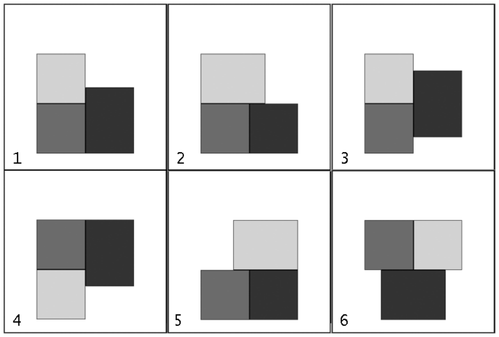


Figure 5
Sample of six solutions (of 72) found by Ilog's OPL Studio 3.7

Maxon's Cinema 4D©, where the constraint system must be modeled as a directed graph.

There is more than one representation of a problem as a constraint satisfaction problem. Constraint programming provides a fully automated procedure to solve a CSP, but it is not the only one. More important than the particular tool to solve a problem seems to be its formal representation, i.e. the result of understanding the problem in depth and be able to explain it to others or to translate it into computer language. The level of effectiveness in using one or the other prototype exposed here strongly depends on the problem structure itself. The usual error in CAAD research is to attempt full automation of the architectural design process or part of it. The continuous or discrete nature of the solution space specified by the problem itself is absolutely decisive when outlining R&D goals in the field of architectural design.

References

Green, M.: 2004, El programa de vivienda progresiva en Chile 1990-2002. Inter-American Development Bank Publication, Social Development Division.

Gross, M. (ed.): 1986, Design as exploring constraints, Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge.

Hidalgo, R.: 1999, La vivienda social en Chile: La acción del estado en un siglo de planes y programas, Scripta Nova [Internet], vol. 45(1). Available from: <http://www.ub.es/geocrit/sn-45-1.htm> [Accessed 30 January, 2007].

Kramer, G.: 1992 a, A geometric constraint engine, Artificial Intelligence, Special volume on constraint-based reasoning, 58(1-3), pp. 327-360.

Kramer, G.: 1992 b, Solving Geometric Constraint Systems: A Case Study in Kinematics, The MIT Press, Cambridge, Massachusetts.

Minsky, M.: 1956, Heuristic aspects of the artificial intelligence problem, Group Reports 34-55, MIT Lincoln Laboratory. Cited in: Alexander, C. (ed.): 1967, Notes on the Synthesis of Form, Harvard University Press, Cambridge, Massachusetts.

MINVU (ed.): 2004, Ordenanza General de Urbanismo y Construcciones (General Ordinance of Urbanism and Constructions), Ministerio de Vivienda y Urbanismo de Chile (Ministry of Housing and Urbanism of Chile), Santiago de Chile.

Peña, W. (ed.) and Parshall, S.: 2001, Problem Seeking: An Architectural Programming Primer, 4th ed. AIA Press, New York.

Rich, E. and Knight, K. (ed.): 1991, Artificial Intelligence, McGraw-Hill Education, New York and London.

Russell, S. and Norvig, P. (eds.): 2002, Artificial Intelligence: A Modern Approach, 2nd ed. Prentice Hall, New Jersey.

Surawski, A. and Cubillos, J.: 2005, Origen e implementación del programa fondo solidario de vivienda, Programa Ciudadanía, Participación y Políticas Públicas, INAP, Universidad de Chile.

UNESCO (ed.): 2005, International Public Debates: Urban Policies and The Right to the City, Division of Social sciences Research and Policy, UNESCO, Paris.

UN-HABITAT (ed.): 2002, The Global Campaign on Urban Governance, United Nations Human Settlements Programme, UN-HABITAT, Nairobi.

Van Hentenryck, P.: 1999, The OPL Optimization Programming Language. The MIT Press, Cambridge, Massachusetts.