Subgraph Isomorphism and Constraint Satisfaction

Gabriel Valiente
Technical University of Catalonia
Department of Software
valiente@lsi.upc.es

Rīga, March 28–April 13, 2001

- J. Larrosa and G. Valiente. Graph pattern matching using constraint satisfaction. In *Proc. Joint APPLIGRAPH and GETGRATS Workshop on Graph Transformation Systems*, pages 189–196, Berlin, 2000.

Contents

- Subgraph Isomorphism
- Constraint Satisfaction
- Subgraph Isomorphism and Constraint Satisfaction
- A New Algorithm for Subgraph Isomorphism
- A Benchmark for Subgraph Isomorphism
- Experimental Results
- Discussion

Subgraph Isomorphism

Finding an isomorphic image of a given graph in another graph

- Application of a graph transformation rule to a graph succeeds by first finding a subgraph isomorphism or a subgraph homomorphism of the left-hand side of the rule into the graph to be transformed

Subgraph Isomorphism

Increasing interest in the field of graph transformation in finding better subgraph isomorphism algorithms

4. H. Bunke, T. Glauser, T.-H. Tran. An efficient implementation of graph grammars based on the RETE matching algorithm.

5. A. Zündorf. Graph pattern matching in PROGRES.

6. M. Rudolf. Utilizing constraint satisfaction techniques for efficient graph pattern matching.

Subgraph Isomorphism

Subgraph homomorphism is not needed for graph transformation

- A. Habel, J. Müller, D. Plump. Double-pushout approach with injective matching.

Subgraph Isomorphism

- Exact methods
  - Backtracking
  - Discrete relaxation
  - Constraint satisfaction
- Optimization methods
  - Continuous
    - Nonlinear optimization
    - Linear optimization
  - Discrete

Constraint Satisfaction

- Constraint satisfaction problem
  - Ordered set of n variables $X = (1, 2, \ldots, n)$
  - Finite domain D_i of possible values for each variable i
  - Set C of constraints among variables

    A constraint $R_{j_1,\ldots,j_r}$ on the ordered set of variables $(j_1, \ldots, j_r)$ is a subset of $D_{j_1} \times \cdots \times D_{j_r}$ which only contains the allowed combinations of values for variables $j_1, \ldots, j_r$

Constraint Satisfaction

- Solution for a constraint satisfaction problem
  - An assignment of values to variables is complete if it includes every variable in X
  - A (possibly incomplete) assignment is consistent if it satisfies every constraint it is involved with
  - A solution for a constraint satisfaction problem is a consistent and complete assignment (that is, it satisfies every constraint)

Constraint Satisfaction

- Typical tasks of interest in constraint satisfaction
  - Find one solution
  - Find all solutions
  - Find the best solution (under some preference criterion)

Constraint Satisfaction

Example. *Place n queens on an $n \times n$ chessboard in such a way that no queen can take any other*

- *There are n variables, representing the n rows of the chessboard*
- *Each variable has domain $\{1, \ldots, n\}$, representing the n columns of the chessboard*
- *No two queens can be on the same row, the same column, or the same diagonal of the chessboard*

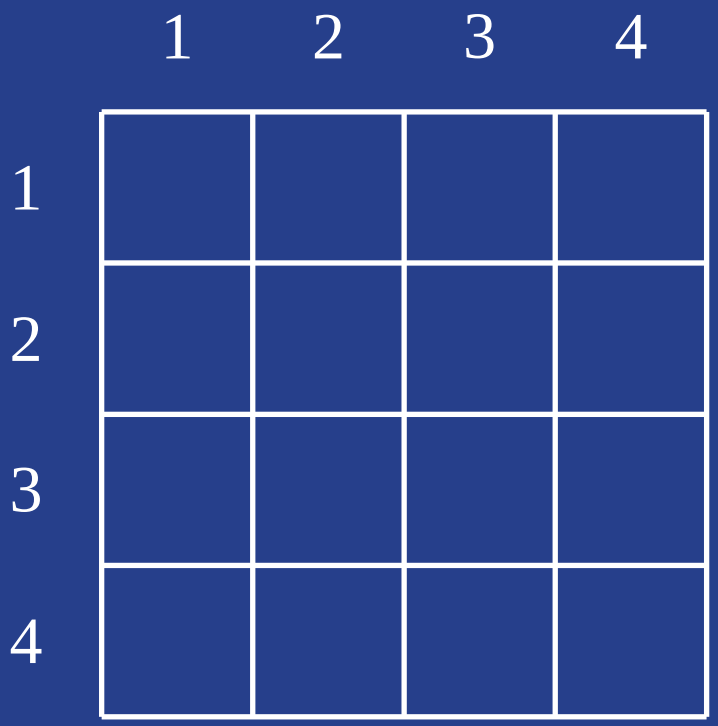
Constraint Satisfaction

Example. *Place n queens on an $n \times n$ chessboard in such a way that no queen can take any other*

- $X = (1, \ldots, n)$
- $D_i = \{1, \ldots, n\}$ *for* $1 \leqslant i \leqslant n$
- $R_{ij} = \{(a, b) \mid a \neq b \wedge |i - j| \neq |a - b|\}$ *for* $1 \leqslant i, j \leqslant n$

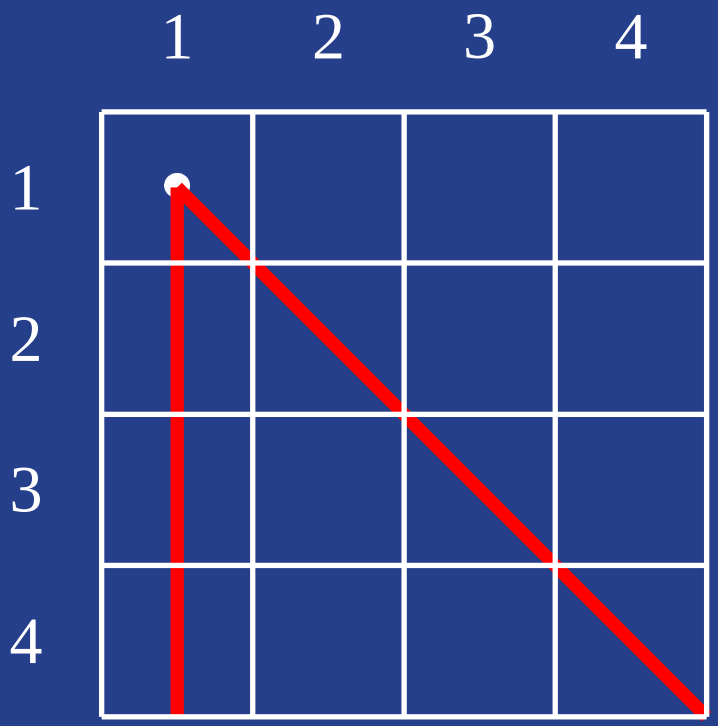
Constraint Satisfaction

Example. *Place n queens on an $n \times n$ chessboard in such a way that no queen can take any other*



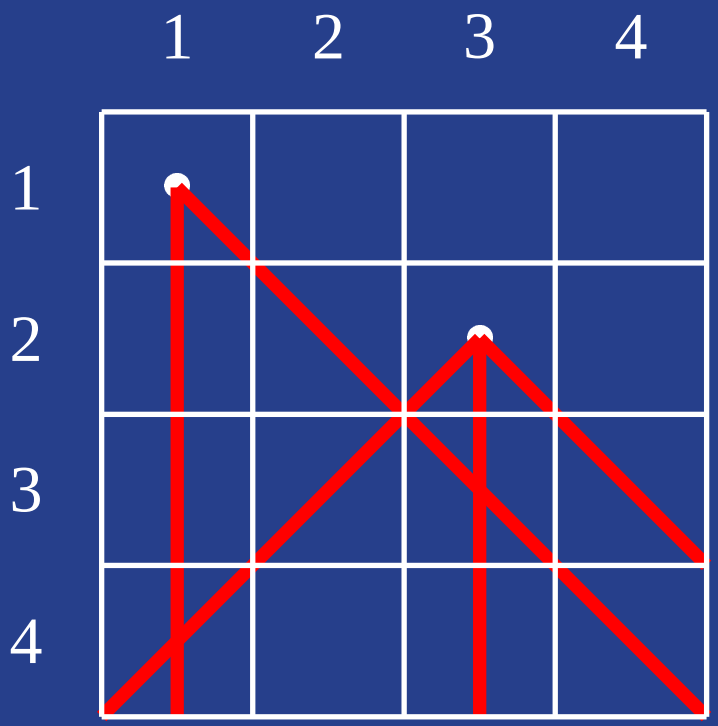
Constraint Satisfaction

Example. *Place n queens on an $n \times n$ chessboard in such a way that no queen can take any other*



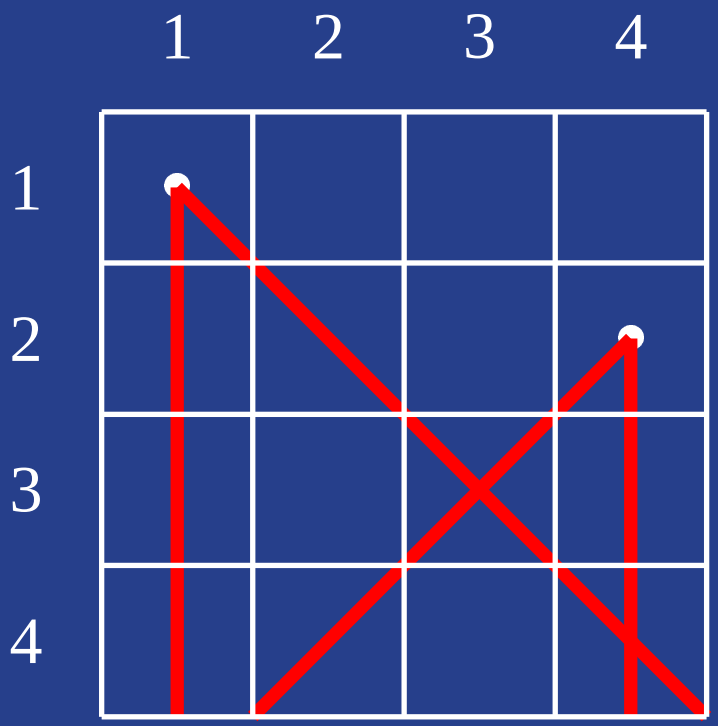
Constraint Satisfaction

Example. *Place n queens on an $n \times n$ chessboard in such a way that no queen can take any other*



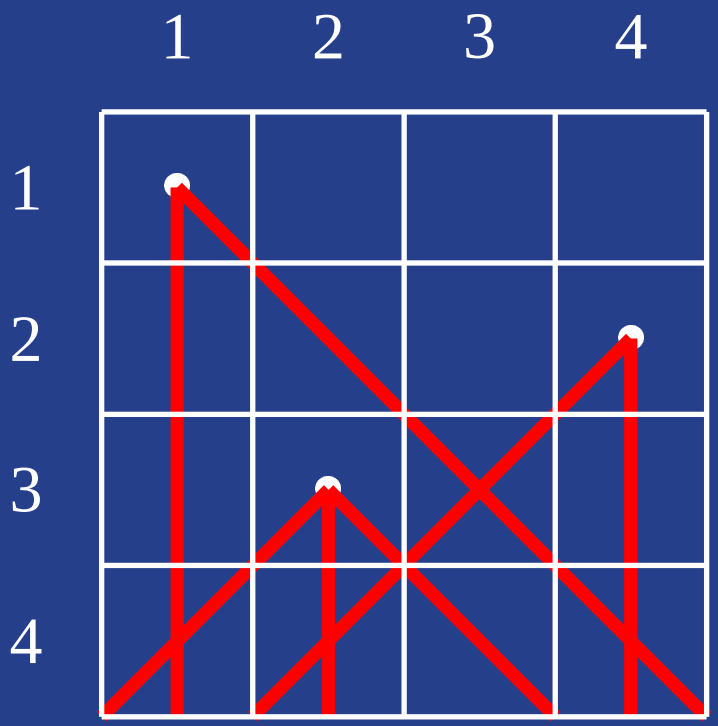
Constraint Satisfaction

Example. *Place n queens on an $n \times n$ chessboard in such a way that no queen can take any other*



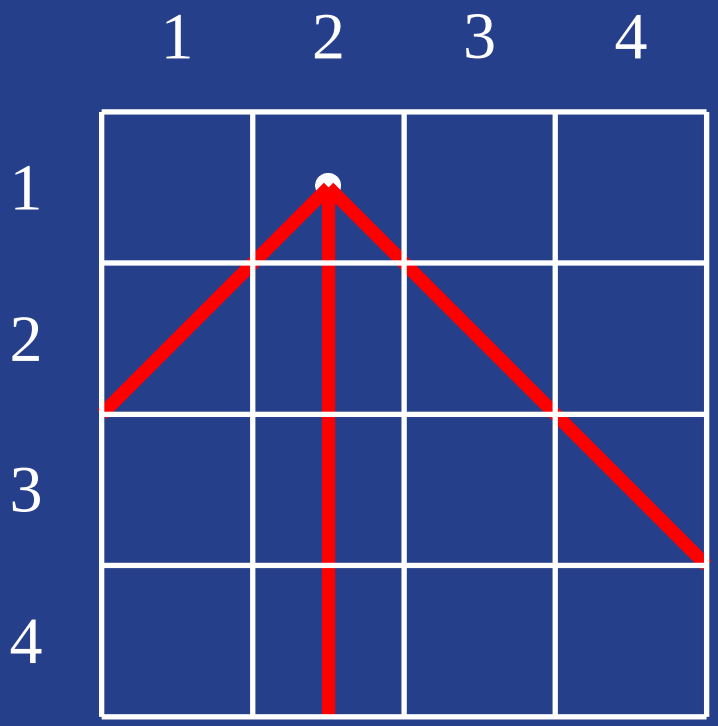
Constraint Satisfaction

Example. *Place n queens on an $n \times n$ chessboard in such a way that no queen can take any other*



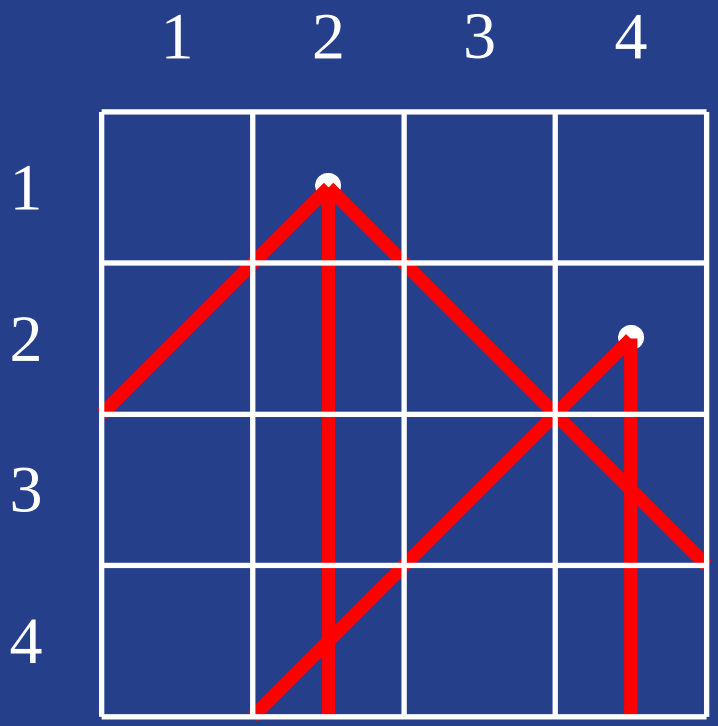
Constraint Satisfaction

Example. *Place n queens on an $n \times n$ chessboard in such a way that no queen can take any other*



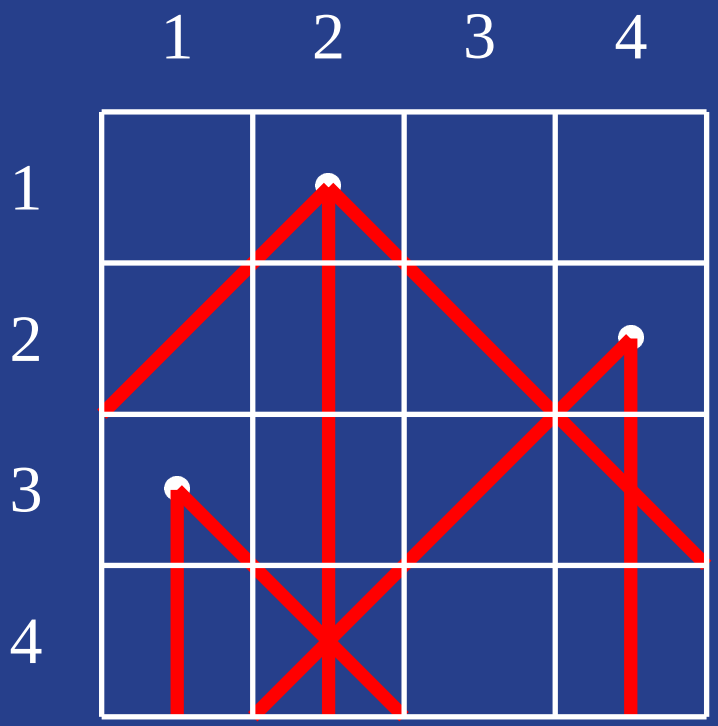
Constraint Satisfaction

Example. *Place n queens on an $n \times n$ chessboard in such a way that no queen can take any other*



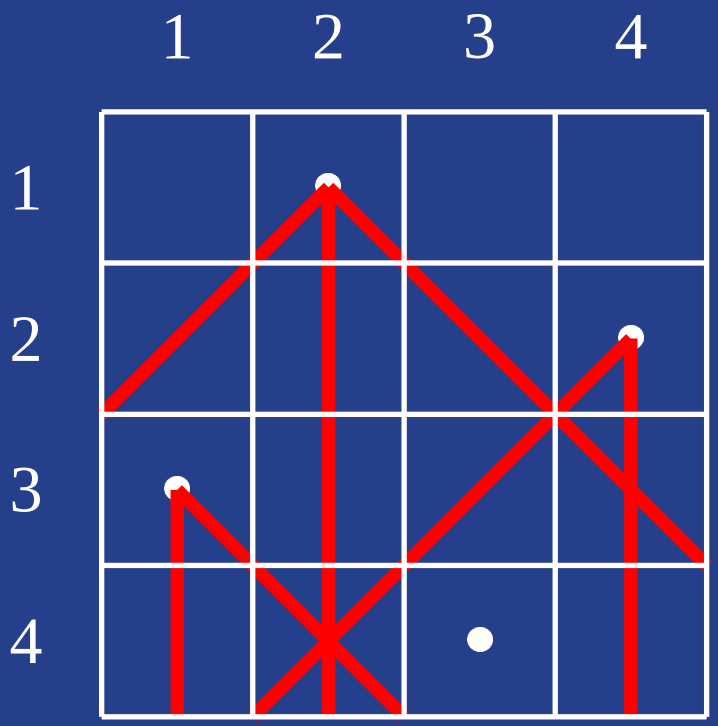
Constraint Satisfaction

Example. *Place n queens on an $n \times n$ chessboard in such a way that no queen can take any other*



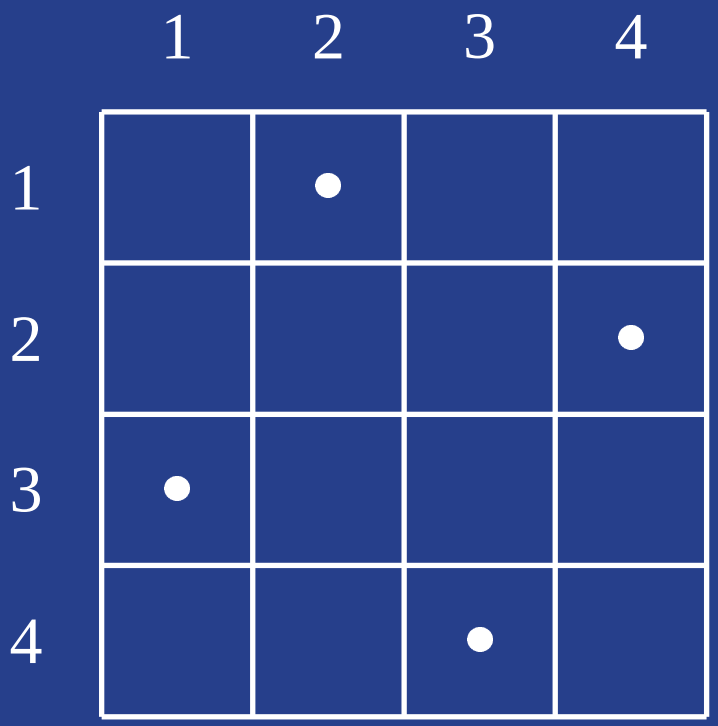
Constraint Satisfaction

Example. *Place n queens on an $n \times n$ chessboard in such a way that no queen can take any other*



Constraint Satisfaction

Example. *Place n queens on an $n \times n$ chessboard in such a way that no queen can take any other*



Constraint Satisfaction

- Solving a constraint satisfaction problem with finite domains is NP-complete

Constraint Satisfaction

- Algorithms for constraint satisfaction problems
  - Start with an empty (trivially consistent) assignment and attempt to extend it by adding one variable at a time, while keeping the assignment consistent
  - When no extension is possible, backtrack and change a previous decision
  - Stop when a complete assignment has been computed, or when the whole search space has been unsuccessfully traversed

Constraint Satisfaction

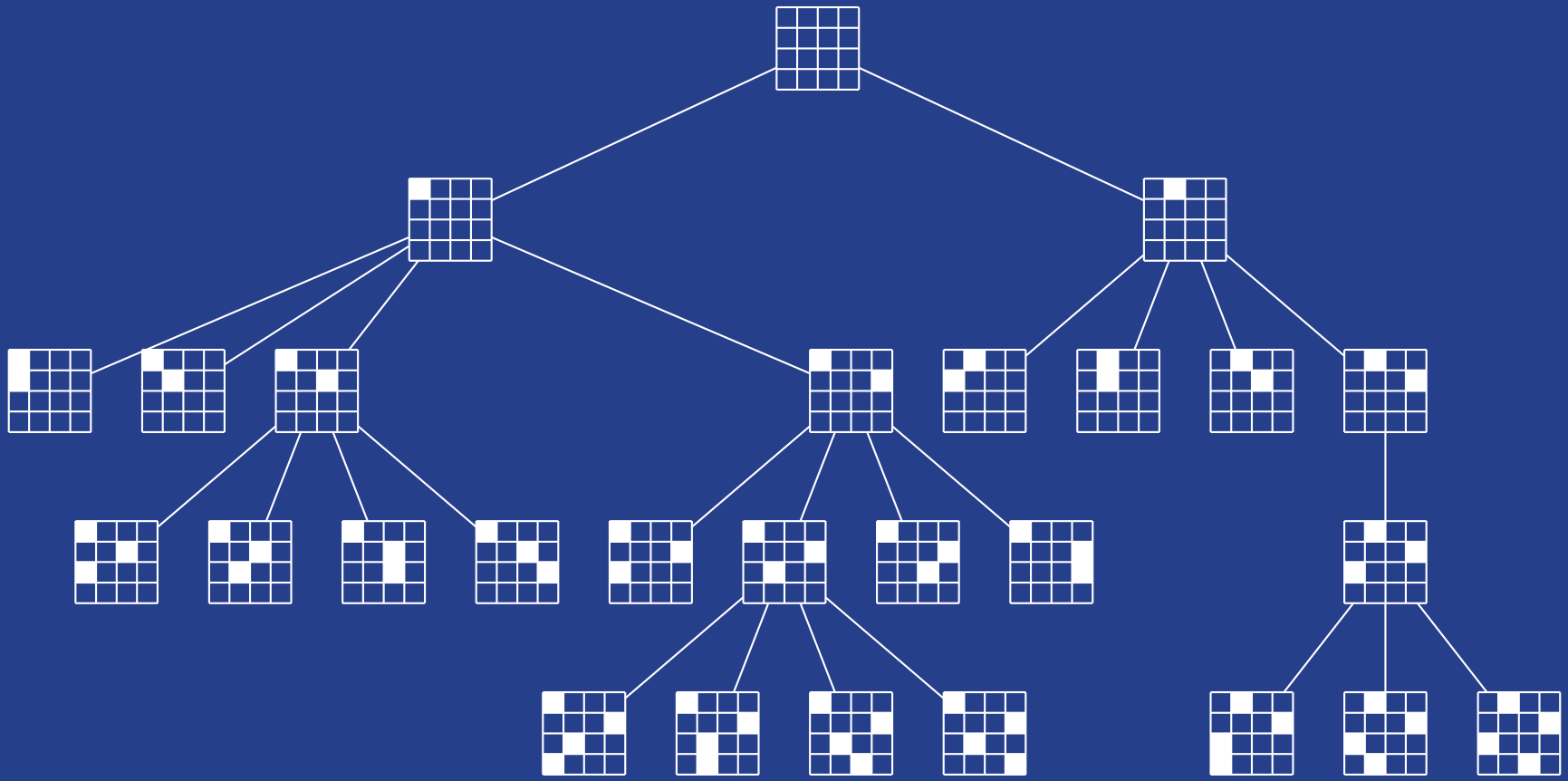
- Algorithms for constraint satisfaction problems
  - During search,
    - assigned variables are called past variables
    - unassigned variables are called future variables
    - the variable under consideration is called the current variable

Constraint Satisfaction

- Algorithms for constraint satisfaction problems
  - Systematic search
    - Generate-and-test
    - Backtracking
  - Consistency enforcement
    - Node consistency
    - Arc consistency
    - Path consistency
  - Constraint propagation
    - Look-ahead
      - Forward checking
      - Full look ahead
    - Look-back
      - Backmarking
      - Backjumping

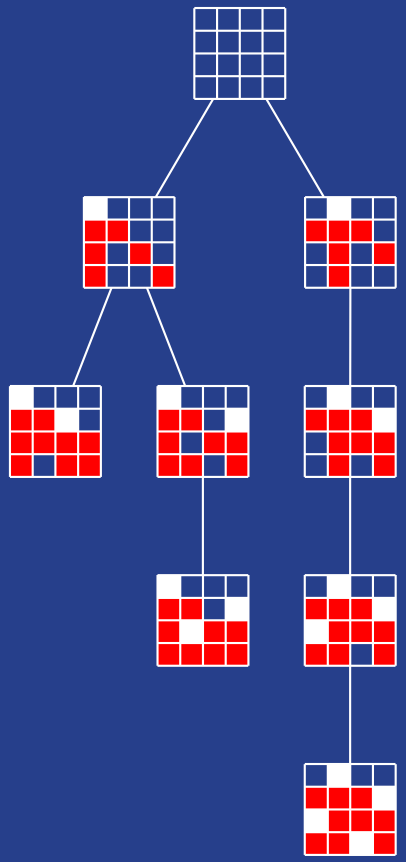
Constraint Satisfaction

- Search space for 4-queens using backtracking



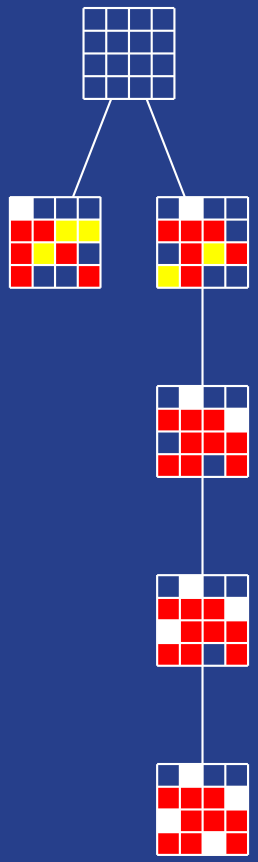
Constraint Satisfaction

- Search space for on 4-queens using forward checking



Constraint Satisfaction

- Search space for 4-queens using full look-ahead

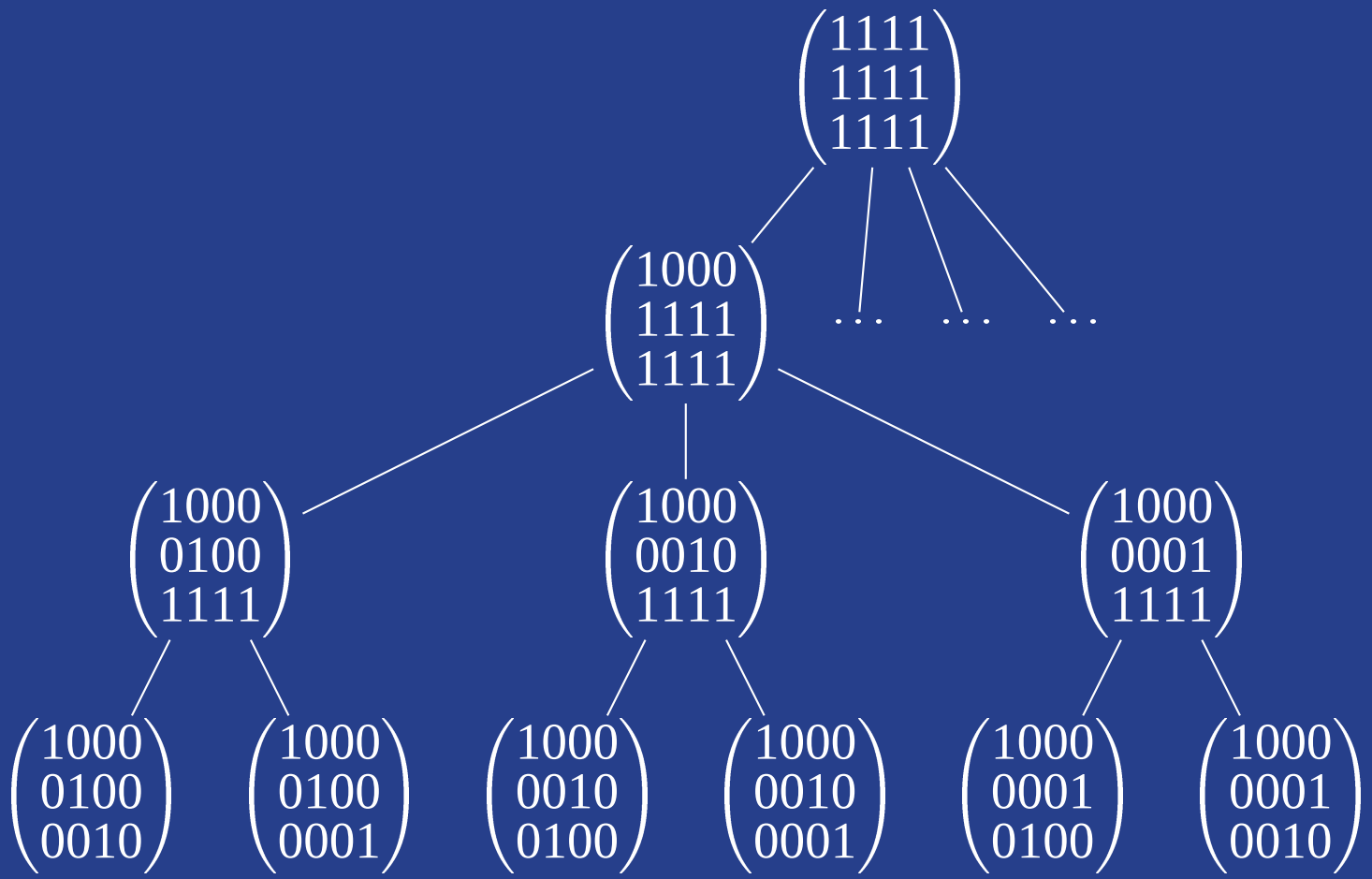


Subgraph Isomorphism and Constraint Satisfaction

- Look-ahead algorithms
  - Best choice for non-trivial problems
  - Each time the current variable is assigned, these algorithms remove values from future variable domains that cannot participate in a solution extending the current assignment
  - Default in most constraint programming tools

Subgraph Isomorphism and Constraint Satisfaction

- Search space for subgraph isomorphism using backtracking



Subgraph Isomorphism and Constraint Satisfaction

- Really full look-ahead
  - J. R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM*, 23(1):31–42, 1976.

Subgraph Isomorphism and Constraint Satisfaction

- Forward checking
  - J. J. McGregor. Relational consistency algorithms and their application in finding subgraph and graph isomorphisms. *Information Sciences*, 19:229–250, 1979.

Subgraph Isomorphism and Constraint Satisfaction

- Back jumping (cannot improve on forward checking)
  - M. Rudolf. Utilizing constraint satisfaction techniques for efficient graph pattern matching. In *Theory and Application of Graph Transformations*, volume 1764 of *Lecture Notes in Computer Science*, pages 215–227. Springer-Verlag, 2000.

Subgraph Isomorphism and Constraint Satisfaction

Problem formulation

- A variable i is associated with each vertex $v_i \in V_1$
- All variables take values on domain V_2
- Subgraph isomorphism of graph $G_1 = (V_1, E_1)$ into graph $G_2 = (V_2, E_2)$ as finding a complete assignment satisfying a structure constraint

Subgraph Isomorphism and Constraint Satisfaction

Definition. [Structure constraint] *Vertices (variables) of G_1 can be mapped to vertices (values) of G_2 if*

$$R_{i,j} = \{(v_a, v_b) \in V_2 \times V_2 \mid \\ v_a \neq v_b \wedge (i,j) \in E_1 \Rightarrow (v_a, v_b) \in E_2\}$$

is satisfied for all $i, j = 1, \ldots, n$ with $i \neq j$.

- Really full look-ahead. Ullmann (1976)
- Forward checking. McGregor (1979)

Subgraph Isomorphism and Constraint Satisfaction

Definition. [Degree constraint] *A vertex (variable) of G_1 can be mapped to a vertex (value) of G_2 if*

$$R_i = \{v_a \in V_2 \mid \deg(G_1, i) \leqslant \deg(G_2, v_a)\}$$

is satisfied for all $i = 1, \ldots, n$.

- Really full look-ahead. Ullmann (1976)

A New Algorithm for Subgraph Isomorphism

Definition. [Neighborhood constraint] *A vertex (variable) $i \in V_1$ can only be mapped to vertex (value) $v_a \in V_2$ if*

$$R_{i,p,\ldots,q} = \left\{ \begin{array}{c} (v_{a_1}, \ldots, v_{a_r}) \in V_2 \times \cdots \times V_2 \mid \\ v_{a_k} \neq v_{a_l} \quad \forall k,l = 1,\ldots,r \quad (k \neq l) \\ \wedge (v_{a_1}, v_{a_k}) \in E_2 \quad \forall k = 2,\ldots,r \end{array} \right\}$$

is satisfied for all $i = 1,\ldots,n$, where $\{p,\ldots,q\}$ is the neighborhood of vertex i in graph G_1.

A New Algorithm for Subgraph Isomorphism

Vertices $i, j \in V_1$ are mappable to vertices $v_a, v_b \in V_2$ if

Structure constraint $v_a \neq v_b$ and $(i, j) \in E_1 \Rightarrow (v_a, v_b) \in E_2$.

Vertex $i \in V_1$ is mappable to vertex $v_a \in V_2$ if

Degree constraint $\deg(i) \leqslant \deg(v_a)$.

Neighborhood constraint $\Gamma(i)$ is mappable to $\Gamma(v_a)$.

A New Algorithm for Subgraph Isomorphism

- Structure constraint

A New Algorithm for Subgraph Isomorphism

- Degree constraint

A New Algorithm for Subgraph Isomorphism

- Neighborhood constraint

A New Algorithm for Subgraph Isomorphism

Theorem. *The new algorithm for subgraph isomorphism never visits more nodes than really full look-ahead and than forward checking using degree constraints and structure constraints.*

A Benchmark for Subgraph Isomorphism

- Experimental comparison of algorithms for subgraph isomorphism
  - Based on results reported by J. R. Ullmann (1976)
  - Using a particular model of random graph
  - Comparison is based on only 4 problem instances
  - Original description is not sufficient for reproducing the experiments
- No benchmark has been developed yet for subgraph isomorphism

A Benchmark for Subgraph Isomorphism

- Based on a large library of well-defined graphs provided in the Stanford GraphBase for the empirical comparison of combinatorial algorithms
- Comprising 6441 problem instances for undirected graphs and 1770 problem instances for directed graphs
- Available from

  `http://www.lsi.upc.es/~valiente/`

  for use with the Stanford GraphBase and with LEDA

A Benchmark for Subgraph Isomorphism

	undirected		directed	
	yes	no	yes	no
simple	102	11	54	5
acyclic	1	112	34	25
connected	71	42	47	12
biconnected	53	60	39	20
triconnected	43	70	26	33
bipartite	32	81	14	45
planar	19	94	7	52

Experimental Results

- Focused on a subset of 1769 problem instances
- Undirected graphs ranging from 10 to 377 vertices and from 15 to 4950 edges
- Comparison of constraint satisfaction algorithms
  - Number of nodes visited
  - Number of consistency checks performed
  - Number of problem instances solved within a fixed time limit

Experimental Results

- Comparison of the new algorithm with forward checking (number of problem instances)

	Forward Checking	Neighborhood Constraint
No solution found	851	1175
Solution found	183	234
Not solved	735	360

Discussion

- New algorithm for subgraph isomorphism
  - Constraint satisfaction formulation of the problem
  - Exploitation of neighborhood constraints for domain filtering
  - Never visits more nodes than previous methods
  - Best at finding the first solution to a subgraph isomorphism problem
  - Other algorithms are best at finding the next solution to a subgraph isomorphism problem

Discussion

- Benchmark for subgraph isomorphism
  - No standard benchmark for subgraph isomorphism algorithms
  - Use of the Stanford GraphBase as a source of instances for the empirical comparison of subgraph isomorphism algorithms
  - Large number of problem instances readily available for use with the Stanford GraphBase and with LEDA
  - Large number of problem instances remain open