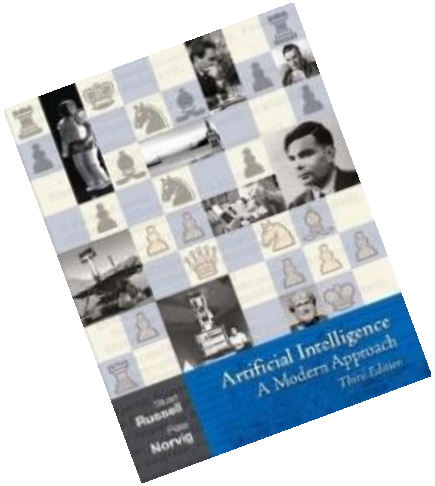


Introduction to Artificial Intelligence

כללי

מפגש שלישי

- פרק 1 - מבוא
- פרק 2 - סוכנים אינטיליגנטיים
- פרק 3 - פתרון בעיות באמצעות חיפוש
- פרק 4 - אלגוריתמים לחיפוש מקומי ובעיות אופטימיזציה (רק סעיף 4.1)
- פרק 5 - חיפוש בתנאי יריבות (סעיפים 5.1-5.4)
- פרק 6 - בעיות סיפוק אילוצים (סעיפים 6.1-6.4)
- פרק 7 - סוכנים לוגיים
- פרק 8 - לוגיקה מסדר ראשון (סעיפים 8.1-8.3 בלי 8.3.3)
- פרק 9 - היסק בלוגיקה מסדר ראשון (בלי 9.3.3, בסעיף 9.4 רק את 9.4.1)
- פרק 10- תכנון (10.1-10.3, 10.4.1, 10.4.4)
- פרק 13 - אי-וודאות
- פרק 14 - הנמקה הסתברותית (14.1-14.2)
- פרק 17 - קבלת החלטות מורכבות (בלי 17.4)
- פרק 18 - למידה מדוגמאות (18.1-18.5)

בעיות שיש להם תנאים לסיפוק

CSP - Constraint Satisfaction Problems

- הגדרות ודוגמאות
- תכונות של בעיות CS
- היוריסטיקות
- אלגוריתם איטרטיבי עבור בעיות CS

הבעיה

- בכל אלגוריתמי החיפוש שלמדנו אינם מניחים דבר על מבנה המצבים,
- למעשה מבחינת אלגוריתם החיפוש כל מצב הוא אובייקט אטומי
- רק הפונקציות הייחודיות לתחום - פונקציית העוקב, פונקציית העלות ופונקציית המחיר יודעות על מבנה המצב
- לכן אי אפשר לפתח יוריסטיקות כלליות שאינן תלויות בתחום בעיות, קשה להתמודד בצורה כללית עם מרחב המצבים, אנחנו מחפשים שיטות כלליות יותר.....

Exercise : Sudoku

- But, before we get into all that, let's do a puzzle…
- http://www.websudoku.com/
- Each Sudoku has a unique solution that can be reached logically without guessing. Enter digits from 1 to 9 into the blank spaces. Every row must contain one of each digit. So must every column, as must every 3x3 square.

Puzzle

				9		3		
	2		6		1	5	9	
3			5		8		2	4
	9					4	8	
6			2		9			1
	5	2					3	
9	7		1		5			3
	3	8	7		4		1	
		5		8				

Problem Formulation

- What is the state space?
- What is the initial state?
- What is the successor function?
- What is the goal test?
- What is the path cost?

Points

- Large search space
- commutativity
- what about using bfs or dfs?
  - fixed depth

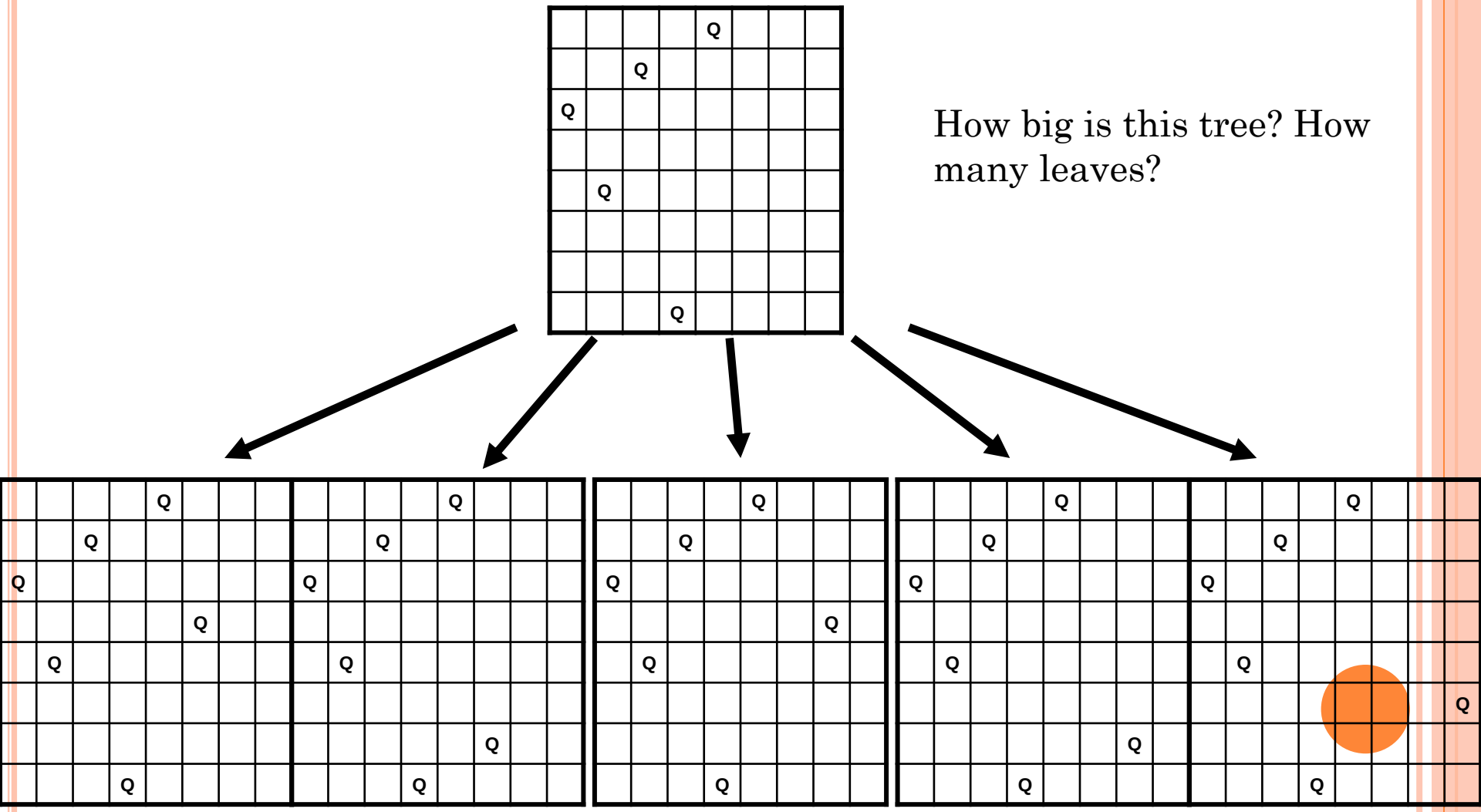
Queens puzzle

- Place eight queens on a chessboard so that no two attack each other

				Q			
		Q					
Q							
						Q	
	Q						
							Q
					Q		
			Q				

Search formulation of the queens puzzle

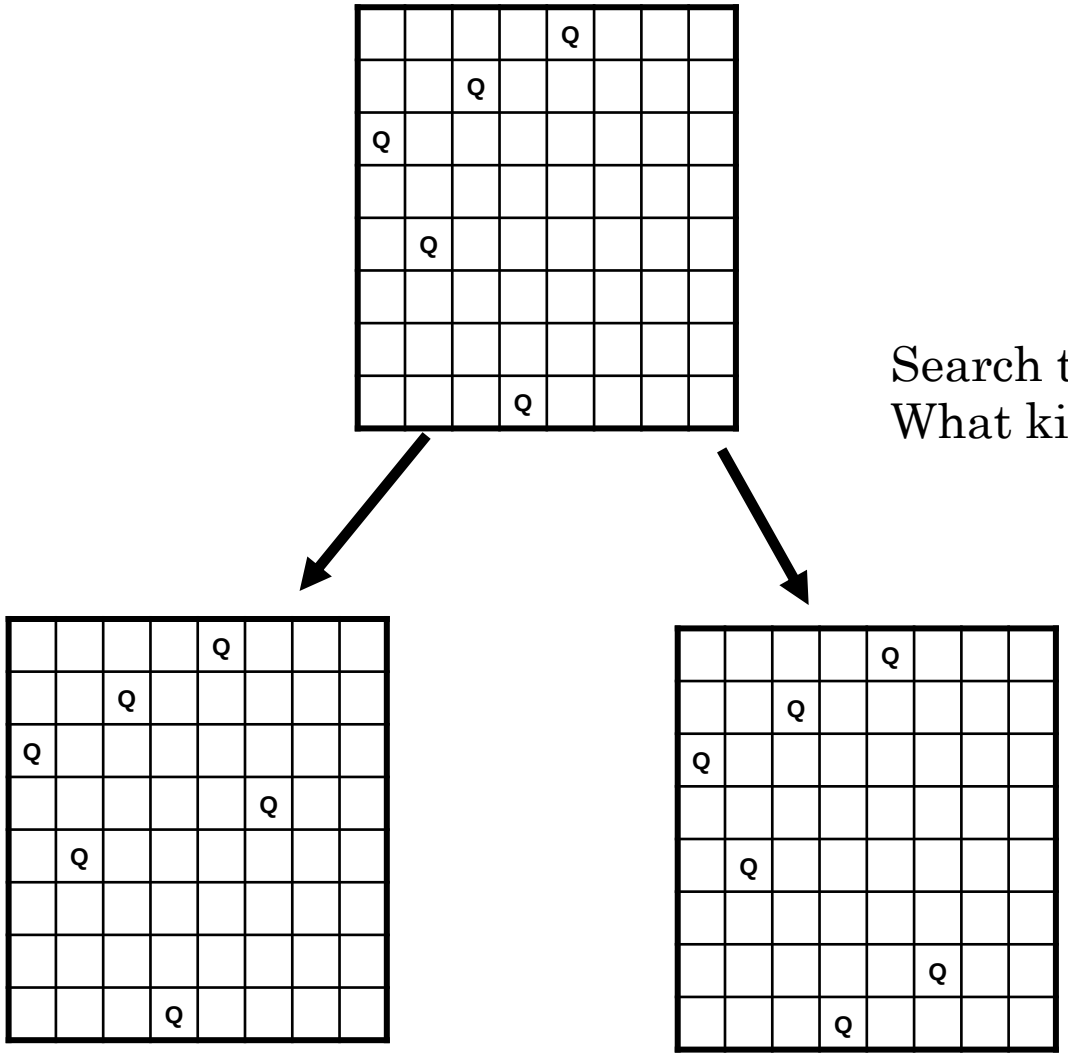
- **Successors**: all valid ways of placing additional queen on the board; **goal**: eight queens placed



How big is this tree? How many leaves?

Search formulation of the queens puzzle

- **Successors**: all valid ways of placing a queen in the next column; **goal**: eight queens placed



Search tree size?
What kind of search is best?

CSP - Constraint Satisfaction Problems

- בעיות הניתנות לניסוח באופן הבא:

  מצא פתרון (יתכן אופטימלי) שמספק מספר אילוצים.

- דוגמאות:

  - בנה מערכת שעות עם כמה שפחות סתירות.
  - שבץ טייסים לטיסות כך שלא יעברו מכסת שעות טיסה, הפסקה סבירה בין הטיסות, ועוד.
  - מצא השמת ערכי אמת לסיפוק נוסחה לוגית.
  - בעית 8 המלכות.
  - בעית תרגיל מוצפן.

CS ניסוח פורמלי של בעיית

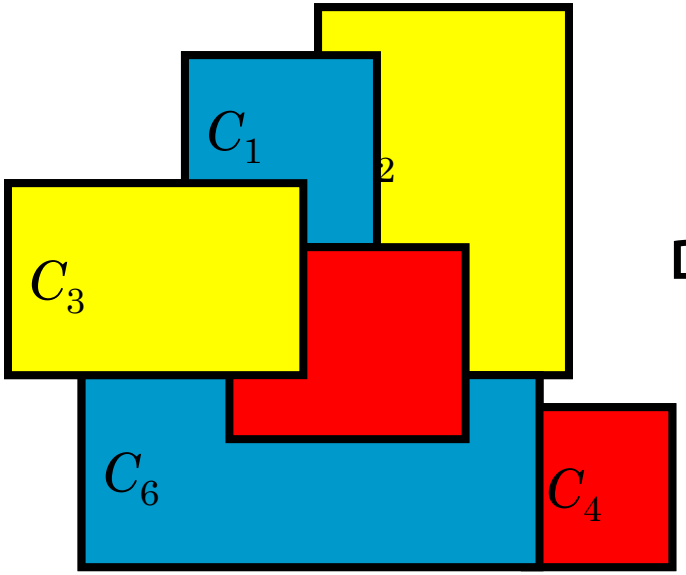
- יצוג פקטורי : מצב מיוצג ע"י אוסף של ערכים למשתנים ניתן לנצל את המבנה כדי להגדיר יוריסטיקות כלליות יותר ולא מחוייבות.
- מצבים:
  - קבוצה של משתנים: $x_1, x_2, \ldots, x_n$
  - תחומי המשתנים: $D_1, D_2 \ldots D_n$
  - וקטור מצב : $(x_1, c_2, x_3, c_4, \ldots, c_n)$ - וקטור של משתנים וערכים מוצבים.
- מבחן מטרה: אילוצים על המשתנים.
  דוגמא: $x_i \neq x_j$
- מצב התחלתי: וקטור של משתנים ללא ערכים.
- מצב מטרה: וקטור של ערכים בלבד שמספקים את כל האילוצים.

ניסוח פורמלי של בעיית CS

- בעיות חיפוש כלליות
  - מצב הוא מין "קופסא שחורה"
  - יכול להיות כל מבנה נתונים שתומך בפונקציה המנפקת את הצאצאים, פונקציה היוריסטית, ומבחן המטרה
  -
- CSP
  - מצב מוגדר על ידי משתני Xi עם טווח ערכים מהתחום D_i
  - מצב מטרה הוא מערכת של אילוצים המפרטים שילובים מותרים של ערכים עבור תת קבוצות של משתנים
  - דוגמה פשוטה לשפת ייצוג פורמאלית
  - מאפשר אלגוריתמים שימושיים למטרות כלליות, עם יותר כוח מאשר אלגוריתמי חיפוש הסטנדרטיים

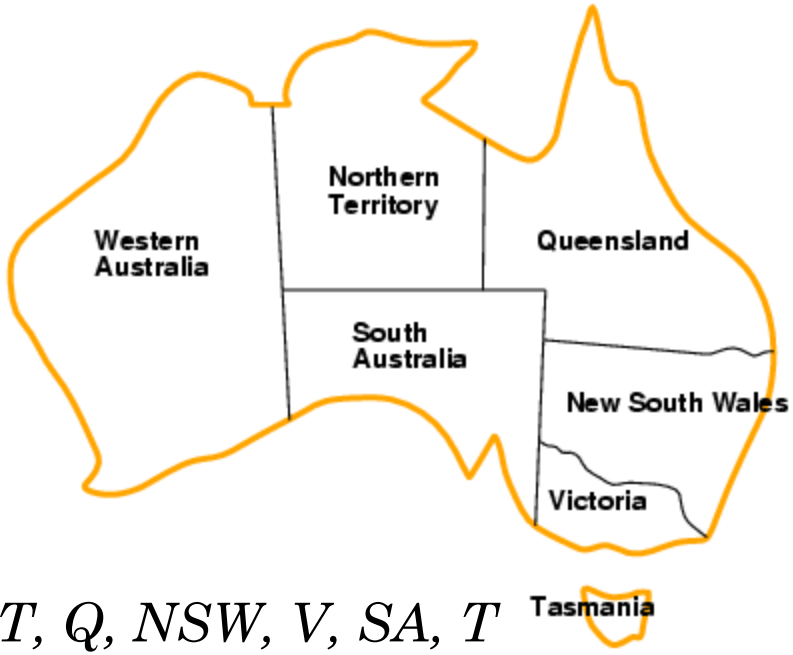
דוגמאות - CS ניסוח פורמלי של בעיית

- בעיית צביעת מפה
- יש לצבוע את המשטחים כך שכל שני משטחים נושקים יהיו בצבעים שונים



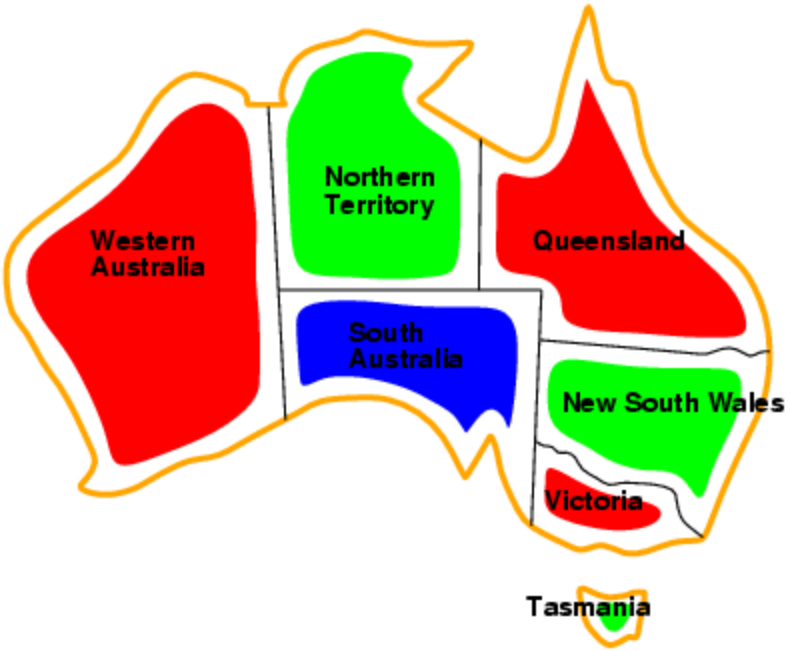
- משתנים: C_1 , C_2 ,C_3 ,C_4 , C_5 ,C_6
- תחומים: לכולם {אדום, כחול, צהוב}
- אילוצים : $C_1 \neq C_3$,$C_1 \neq C_2$ $C_1 \neq C_5$, . . .

Example: Map-Coloring



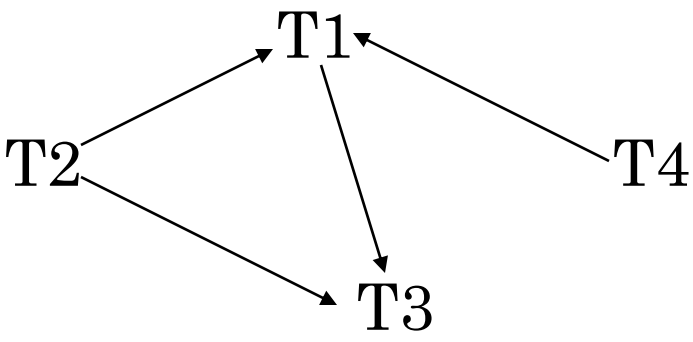
- Variables *WA, NT, Q, NSW, V, SA, T*
- Domains D_i = {red,green,blue}
- Constraints: adjacent regions must have different colors
-
- e.g., WA ≠ NT, or (WA,NT) in {(red,green),(red,blue),(green,red),(green,blue),(blue,red),(blue,green)}
-

Example: Map-Coloring



- Solutions are complete and consistent assignments, e.g., WA = red, NT = green,Q = red,NSW = green,V = red,SA = blue,T = green
-

Example: Task Scheduling



T1 must be done during T3
T2 must be achieved before T1 starts
T2 must overlap with T3
T4 must start after T1 is complete

- Are the constraints compatible?
- Find the temporal relation between every two tasks

דוגמאות - CS ניסוח פורמלי של בעיית

- בעיית תרגיל מוצפן

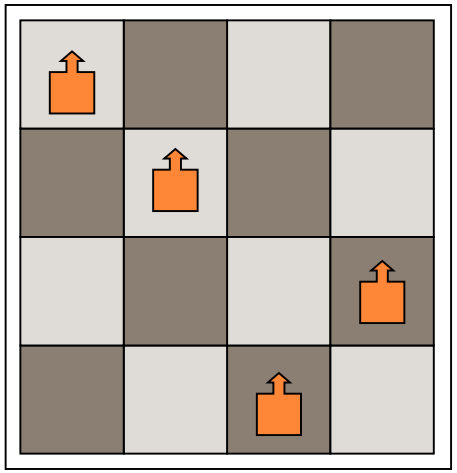
```
   S E N D
+
   M O R E
 -----------
M O N E Y
```

- משתנים: D, E, M, N, O, R, S, Y
- תחומים: לכולם $\{0,1,2,3,4,5,6,7,8,9\}$
- אילוצים: $M \neq 0,\ \ S \neq 0$

$$D \neq R\ ,\ D \neq N\ ,\ D \neq M\ ,\ D \neq E\ \ldots$$

$Y = D+E$ או $Y = D+E-10\ \ ,\ \ldots$

דוגמאות - CS ניסוח פורמלי של בעיית



- בעיית 4 המלכות
- נניח שבמצב ההתחלתי כל מלכה ממוקמת בעמודה נפרדת .
- יש למצוא באיזו שורה למקם כל מלכה.

- משתנים : Q_1, Q_2, Q_3, Q_4
- תחומים: לכולם $\{1,2,3,4\}$
- אילוצים : $Q_i \neq Q_j$ (לא באותה שורה)

$|Q_i - Q_j| \neq |i - j|$ (לא באותו אלכסון)

Example: Sudoku CSP

- variables:
- domains:
- constraints:
- Goal: Assign a value to every variable such that all constraints are satisfied

Example: Sudoku CSP

- variables: $X_{11}, \ldots, X_{99}$
- domains: $\{1,\ldots,9\}$
- constraints:
  - row constraint: $X_{11} \neq X_{12}, \ldots, X_{11} \neq X_{19}$
  - col constraint: $X_{11} \neq X_{12}, \ldots, X_{11} \neq X_{19}$
  - block constraint: $X_{11} \neq X_{12}, \ldots, X_{11} \neq X_{33}$
- Goal: Assign a value to every variable such that all constraints are satisfied

סוגי המשתנים בבעיות CS

- CSPs עם משתנים בדידים
  - תחומים סופיים
  - תחומים אינסופיים
- CSPs עם משתנים רציפים

CSPs עם משתנים בדידים - תחומים סופיים

- בסוגים הפשוטים ביותר של בעיות CS, למשתנים יש תחומים בדידים וסופיים.
- דוגמאות:
  - צביעת מפה.
  - בעיית תזמון (scheduling) עם אילוצים.
  - בעיית 8 המלכות (המשתנים הם המיקומים בכל אחת מהעמודות. התחומים הם {1,2,3,4,5,6,7,8}).
  -

CSPs עם משתנים בדידים - תחומים אינסופיים

- תחום בדיד יכול להיות גם אינסופי כגון: קבוצת המספרים השלמים, קבוצת המחרוזות וכד'. (אם לא נקבע זמן יעד (deadline) עבור בעיית תזמון תהליכים, יהיה מספר אינסופי של זמני התחלה לכל משתנה).
- לא ניתן לתאר את האילוצים על-ידי מניית כל הצירופים האפשריים של ערכים. במקום זאת, יש להשתמש בשפת אילוצים, למשל: $T_1+d_1 \leq T_2$ מבלי למנות את קבוצת הזוגות האפשריים עבור (T_1, T_2).
- לעיתים אפשר להפוך את התחומים לסופיים על-ידי חסימת הערכים של כל המשתנים. למשל, בבעיית תזמון אפשר לחסום את הערכים על-ידי הסכום של האורך של כל התהליכים.

עם משתנים רציפיCSPs

- בעיות CS עם תחומים רציפים שכיחות בעולם האמיתי.
- למשל: scheduling of experiments on the Hubble Space Telescope
- דורש תזמונים מאד מדוייקים של זמני ההתחלה/סיום של התצפיות. המשתנים הינם בעלי ערכים רציפים הצריכים לעמוד במגוון אילוצי קדימות וכח אסטרונומיים.
-
- CSPs עם תחומים רציפים נלמדות בתחום של חקר ביצועים.
- הידועות ביותר הן אלה של תכנון ליניארי שבהן האילוצים הם אי-שוויונים ליניאריים. לבעיות תכנון ליניארי יש אלגוריתמים פולינומיאליים.

סוגי אילוצים

- אילוץ אונרי: מערב משתנה אחד
  - למשל, בבעיית צביעת המפה: $SA \neq green$.
- אילוץ בינארי: מערב שני משתנים
- למשל, בבעיית צביעת המפה: למשל: $SA \neq NSW$.
- אילוץ **n-ary**: הוא אילוץ שבו מעורבים מספר כלשהו של משתנים ונקרא גם אילוץ גלובלי (לא כל משתני הבעיה צריכים להיות מעורבים באילוץ גלובלי).
  - למשל, בבעיות סודוקו – ערכי המשתנים בשורה או בעמודה צריכים להיות שונים זה מזה.

אבל .....

- נשים לב כי אפשר להעביר מערכת אילוצים כלליים למערכת שקולה של אילוצים בינאריים ע"י שימוש במשתני עזר
- ניתן להפטר מאילוצים אונריים ע"י צמצום התחום של המשתנים
- לכן נדבר בדרך כלל על **מערכת אילוצים בינאריים**
- **בעיית CS בינרית היא בעיית CS עם אילוצים בינריים בלבד והיא ניתנת לייצוג כגרף אילוצים כגון זה שבאיור 6.1(b) בעמוד 204 בספר הלימוד.**
- **הצמתים בגרף הם המשתנים והקשתות הן האילוצים.**
-

אילוצי קדימות

- עד כה תארנו אילוצים מוחלטים שאסור להפר אותם.
- הרבה בעיות CS מהעולם האמיתי, כוללות אילוצים המציינים אלו פתרונות הינם עדיפים.
- למשל, אדום טוב יותר מירוק, או בבעיית שיבוץ מרצים לכיתות (בעיית בניית מערכת שעות לפי העדפות המרצים) באוניברסיטה:
- דוגמה לאילוץ אבסולוטי: מרצה אינו יכול ללמד בשתי כיתות בו-זמנית.
- דוגמה לאילוץ קדימות: פרופ' א' מעדיף ללמד בבוקר ואילו פרופ' ב' מעדיף ללמד אחה"צ. מערכת שעות שבה פרופ' א' מלמד אחה"צ מהווה פתרון אפשרי אך לא אופטימלי.
- אפשר לייצג אילוצים כאלה על-ידי קביעת מחיר לכל השמה של משתנה.
- למשל, מחיר השמת משבצת זמן אחה"צ עבור פרופ' א' הינה 2 נקודות לעומת נקודה אחת במשבצת בוקר.
- פתרון בעיות כאלה ייעשה על-ידי שימוש בשיטות חיפוש לאופטימיזציה (חיפוש מסלול או חיפוש מקומי).

השמה

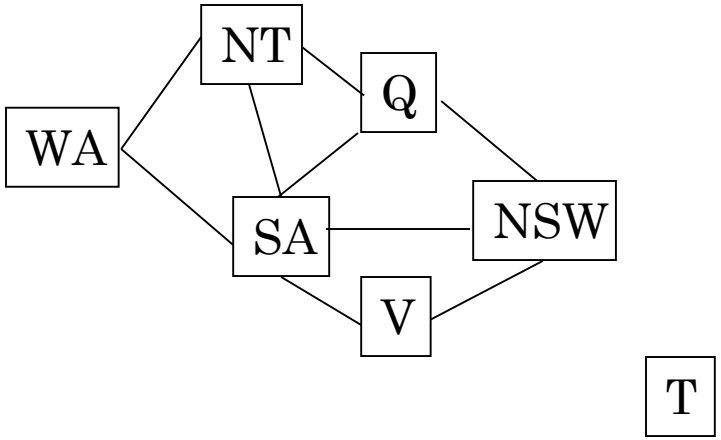
- הפיתרון שלנו יורכב מהשמות
- השמה: אוסף של זוגות של משתנה וערך
- השמה עקבית: השמה שמקיימת את כל האילוצים
- השמה מלאה: השמה בה לכל משתנה יש ערך
- השמה חלקית: השמה שאינה מלאה

Varieties of constraints

- Unary constraints involve a single variable.
  - e.g. $SA \neq green$
- Binary constraints involve pairs of variables.
  - e.g. $SA \neq WA$
- Higher-order constraints involve 3 or more variables.
  - e.g. cryptharithmetic column constraints.
- Preference (soft constraints) e.g. *red* is better than *green* often representable by a cost for each variable assignment → constrained optimization problems.

Constraint Graph

Binary constraints



What is the constraint graph for sudoku?

Two variables are adjacent or neighbors if they are connected by an edge or an arc

Remark

- Finite CSP include 3SAT as a special case (see class on logic)
- 3SAT is known to be NP-complete
- So, in the worst-case, we cannot expect to solve a finite CSP in less than exponential time

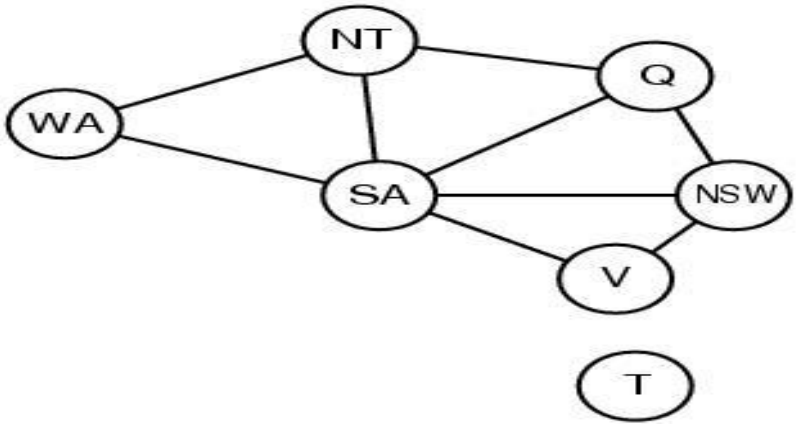
Commutativity of CSP

The order in which values are assigned to variables is irrelevant to the final assignment, hence:

1. Generate successors of a node by considering assignments for only one variable

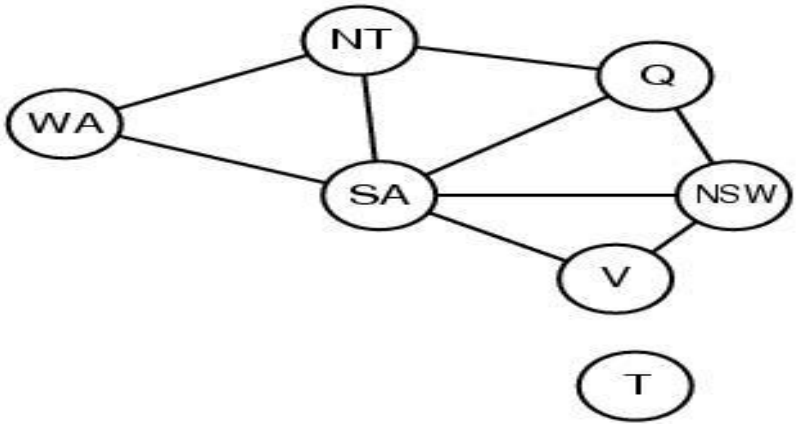
2. Do not store the path to node

Problem structure



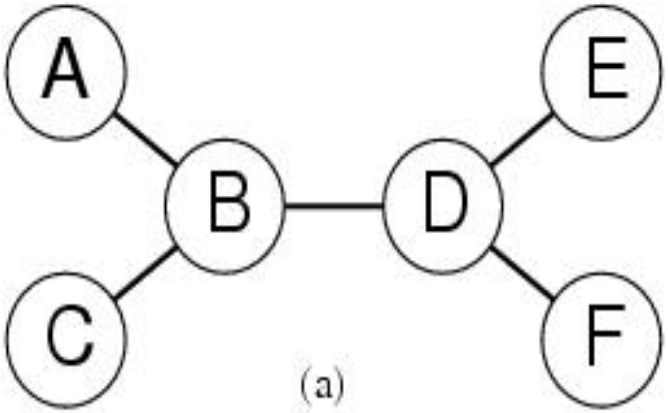
- *How can the problem structure help to find a solution quickly?*
- Subproblem identification is important:
  - Coloring Tasmania and mainland are independent subproblems
  - Identifiable as connected components of constrained graph.
- Improves performance

Problem structure



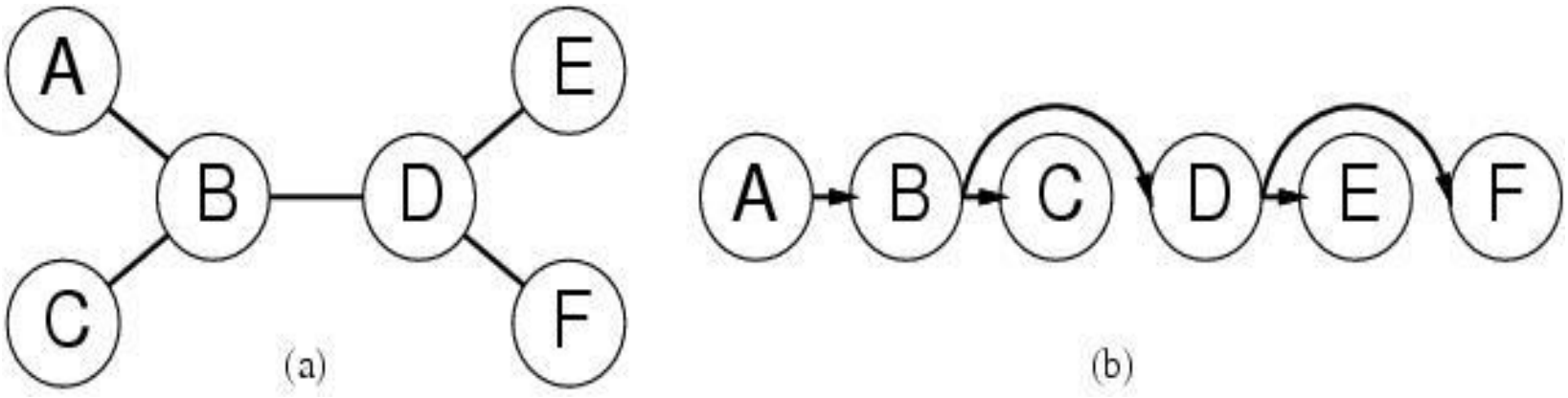
- Suppose each problem has c variables out of a total of n.
- Worst case solution cost is $O(n/c\ d^c)$, i.e. linear in n
  - Instead of $O(d^n)$, exponential in n
- E.g. $n= 80,\ c= 20,\ d=2$
  - 2^{80} = 4 billion years at 1 million nodes/sec.
  - $4 * 2^{20}$= .4 second at 1 million nodes/sec

Tree-structured CSPs



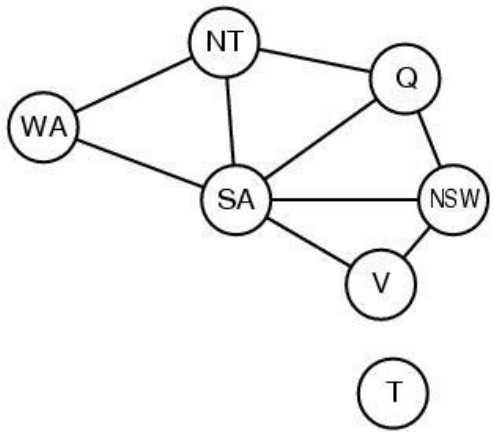
- Theorem: if the constraint graph has no loops then CSP can be solved in $O(nd^2)$ time
- Compare difference with general CSP, where worst case is $O(d^n)$

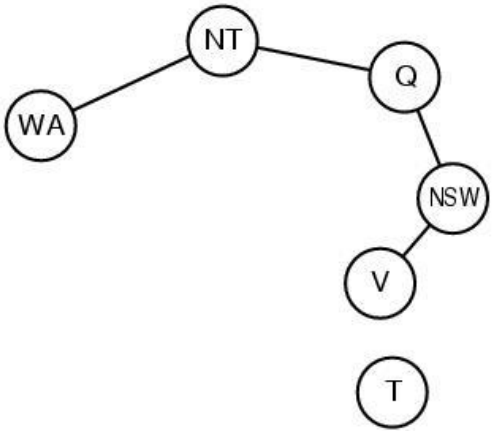
Tree-structured CSPs



- In most cases subproblems of a CSP are connected as a tree
- Any tree-structured CSP can be solved in time linear in the number of variables.
  - Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering. (label var from X_1 to X_n)
  - For j from n down to 2, apply REMOVE-INCONSISTENT-VALUES(Parent(X_j),X_j)
  - For j from 1 to n assign X_j consistently with Parent(X_j)

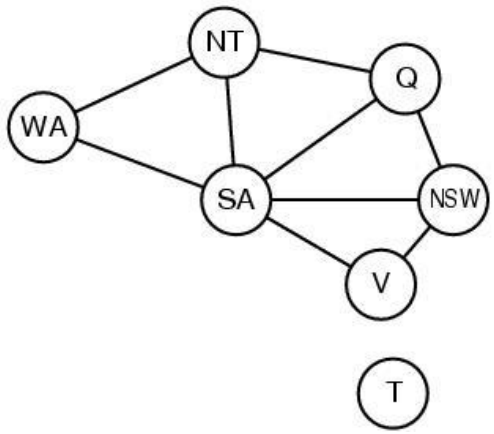
Nearly tree-structured CSPs

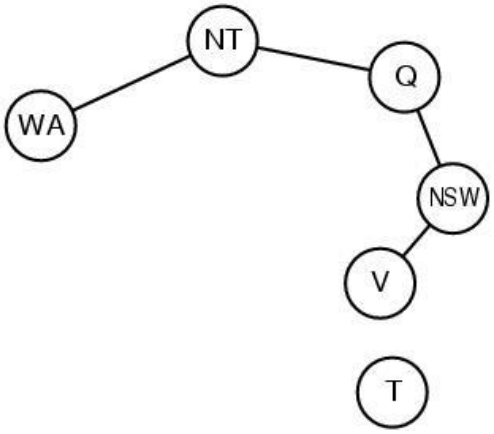




- *Can more general constraint graphs be reduced to trees?*
- Two approaches:
  - Remove certain nodes
  - Collapse certain nodes

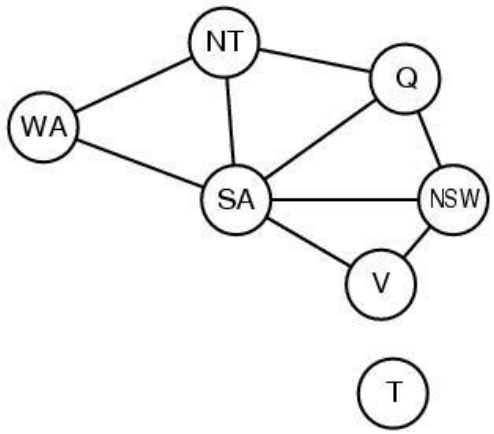
Nearly tree-structured CSPs

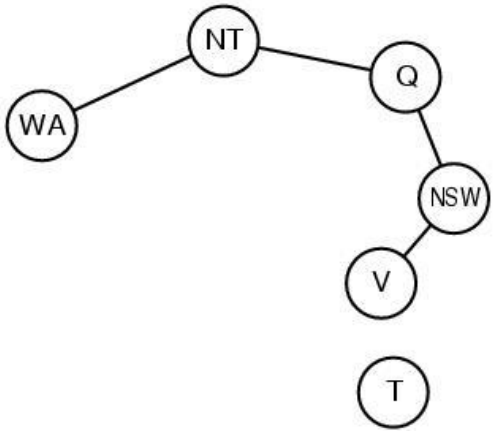




- Idea: assign values to some variables so that the remaining variables form a tree.
- Assume that we assign *{SA=x}* ← *cycle cutset*
  - And remove any values from the other variables that are inconsistent.
  - The selected value for SA could be the wrong one so we have to try all of them

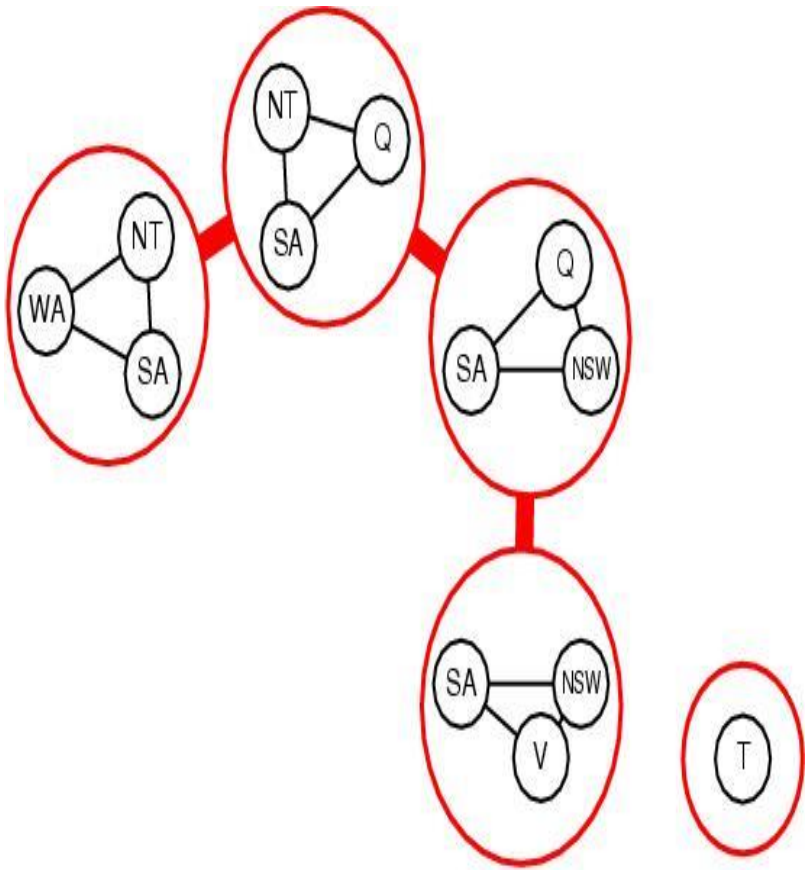
Nearly tree-structured CSPs





- This approach is worthwhile if cycle cutset is small.
- Finding the smallest cycle cutset is NP-hard
  - Approximation algorithms exist
- This approach is called *cutset conditioning*.

Nearly tree-structured CSPs



- Tree decomposition of the constraint graph in a set of connected subproblems.
- Each subproblem is solved independently
- Resulting solutions are combined.
- Necessary requirements:
  - Every variable appears in ar least one of the subproblems.
  - If two variables are connected in the original problem, they must appear together in at least one subproblem.
  - If a variable appears in two subproblems, it must appear in each node on the path.

חיפוש עבור בעיית CS

- אסטרטגית החיפוש:

1. בחר משתנה, בחר ערך מתוך התחום שלו, בדוק אם כל האילוצים מסופקים.
2. אם האילוצים מסופקים המשך באופן רקורסיבי.
3. אחרת, חזור אחורה לבחירה האחרונה ובצע בחירה אחרת .

- גודל עץ החיפוש: b^d

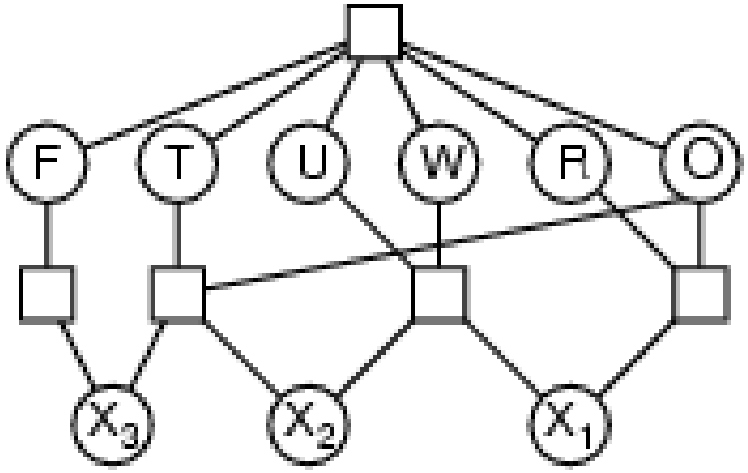
מספר המשתנים - $d=n$

בכל שלב בוחרים רק משתנה אחד - $b=max|D_i|$

- **ניתן לפתור-n מלכות עבור n≈15**

Example: Cryptarithmetic

$$\begin{array}{r} T\ W\ O \\ +\ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$



- Variabl
  $R\ O\ X_1\ X_2\ X_3$
- Domains: $\{0,1,2,3,4,5,6,7,8,9\}$
- Constraints: *Alldiff (F,T,U,W,R,O)*
-
  - $O + O = R + 10 \cdot X_1$
  -
  - $X_1 + W + W = U + 10 \cdot X_2$
  -
  - $X_2 + T + T = O + 10 \cdot X_3$
  - $X_3 = F,\ T \neq 0,\ F \neq 0$
  -

דוגמאות - CS ניסוח פורמלי של בעיית

- בעיית תרגיל מוצפן

```
   S E N D
+
   M O R E
 -----------
M O N E Y
```

- משתנים: D, E, M, N, O, R, S, Y
- תחומים: לכולם $\{0,1,2,3,4,5,6,7,8,9\}$
- אילוצים: $M \neq 0,\ \ S \neq 0$

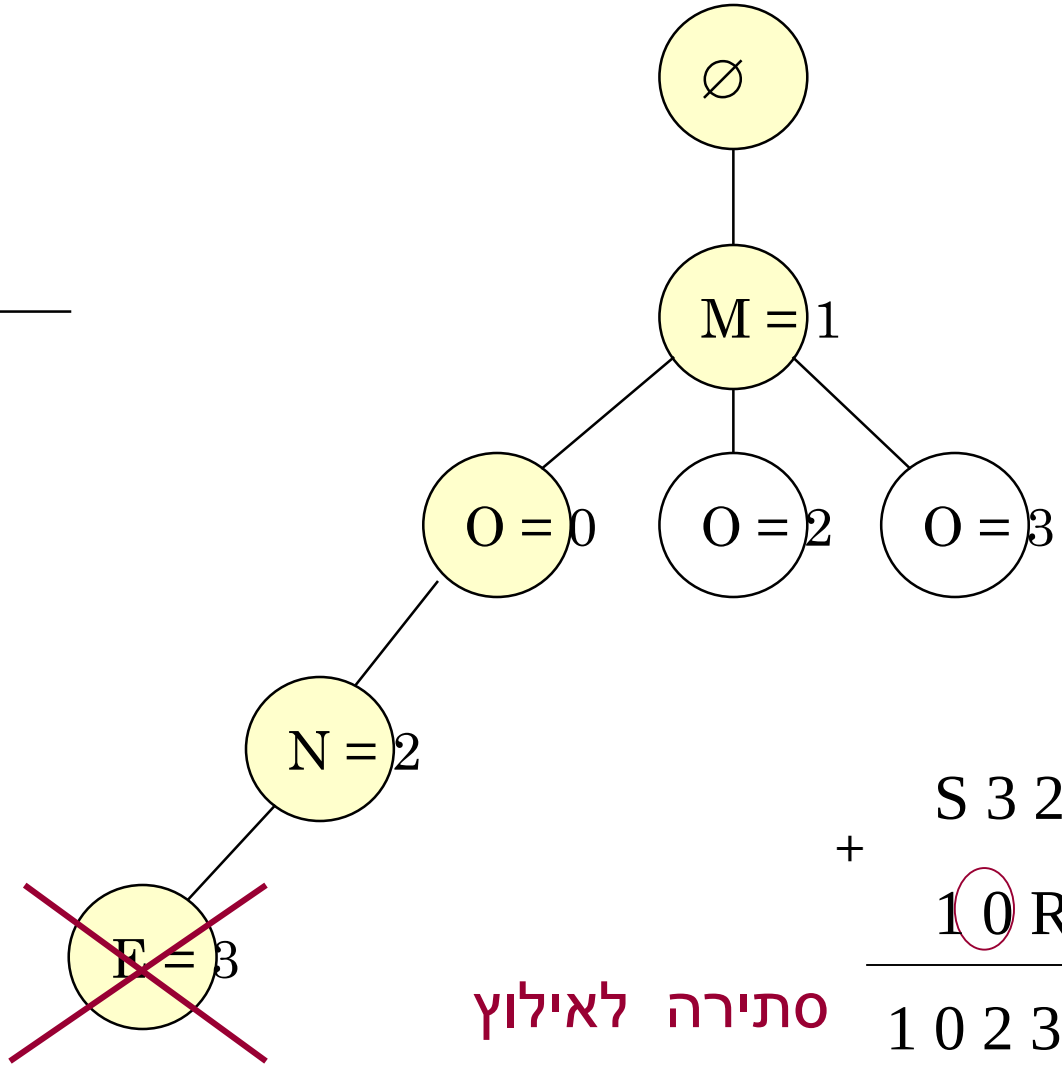
$$D \neq R\ ,\ D \neq N\ ,\ D \neq M\ ,\ D \neq E\ \ldots$$

$Y = D+E$ או $Y = D+E-10\ \ ,\ \ldots$

CS חיפוש עבור בעיית

C_1 C_2 C_3 C_4

```
   S E N D
+
  M O R E
---------
M O N E Y
```




```
  S 3 2 D
+
  1 0 R 3
---------
1 0 2 3 Y
```

סתירה לאילוץ

CP - Constraint Propagation

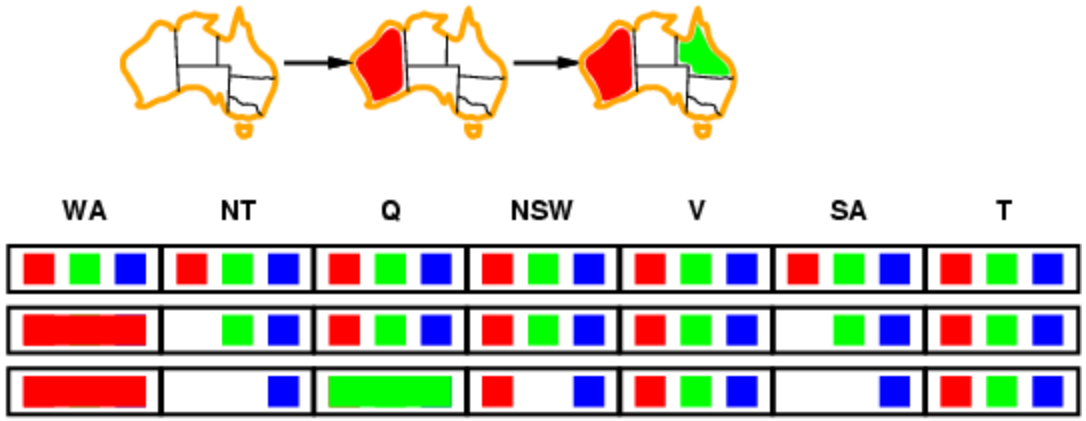
- בחיפוש רגיל במרחב מצבים, אלגוריתם יכול לעשות רק דבר אחד: לחפש.
- בבעיות CS יש בחירה: אלגוריתם יכול
  - לחפש (= לבחור השמה חדשה למשתנה מתוך כמה אפשרויות)
  - לבצע היסק מסוג מסוים הנקרא הפצת אילוצים (Constraint Propagation (CP.

CP - Constraint Propagation

- הפצת אילוצים הוא המונח הכללי של הפצת ההשפעות של האילוצים לצמצום התחומים של המשתנים.
- משתמשים באילוצים כדי לצמצם את מספר הערכים החוקיים עבור משתנה, דבר היכול להפחית את מספר הערכים החוקיים למשתנה אחר, וכן הלאה.
- ניתן לשלב ביצוע CP עם החיפוש, או לבצעה כשלב מקדים לפני שהחיפוש מתחיל.
- הרעיון המרכזי הוא עקביות מקומית (local consistency)אם מתייחסים לכל משתנה כאל צומת בגרף ולכל אילוץ בינרי כאל קשת, אזי התהליך שמחייב עקביות מקומית בכל חלק של הגרף גורם לכך שערכים שאינם עקביים יימחקו .

Constraint propagation

- NT ו-SA הם איזורים סמוכים זה לזה ולכן לא יכולים שניהם להיות באותו הצבע (כחול)!
- כאשר מוחקים ערך מתחום של משתנה X, יש לבדוק את כל המשתנים הקשורים ל-X.
- בדיקה קדימה מקפידה שהמשתנה הנוכחי יהיה עקבי עם כל המשתנים "הסמוכים" לו. אך FC אינה מסתכלת מספיק קדימה ואינה מקפידה שכל המשתנים האחרים ("הסמוכים" למשתנה הנוכחי) יהיו עקביים עם "הסמוכים" להם.
-



Node consistency

- עקביות צומת – משתנה יחיד המתאים לצומת ברשת CSP הוא צומת עקבי אם כל הערכים בתחום המשתנה מספקים את אילוצי המשתנה (האונריים).

Arc consistency

- הרעיון של עקביות קשת מספק שיטה מהירה להפצת אילוצים, שהיא חזקה יותר מבדיקה קדימה באופן משמעותי.
- הצורה הפשוטה ביותר של הפצה דואגת לכך שכל קשת בגרף האילוצים של בעיית ה-CS תהיה עקבית.
-
- $X \rightarrow Y$ היא **קשת עקבית (AC)** אם"ם לכל ערך x בתחום של המשתנה X יש ערך y בתחום של המשתנה Y המקיים את האילוץ הבינרי על הקשת $X \rightarrow Y$ (y עקבי עם x).
- אם הבדיקה נכשלת, יש לצמצם את התחום של X כלומר, אותם ערכי x שאין להם ערך עקבי יכולים להימחק מהתחום של X.

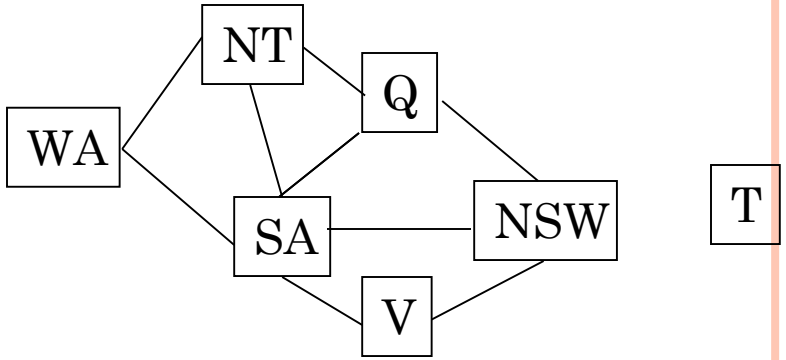
Arc consistency algorithm AC-3

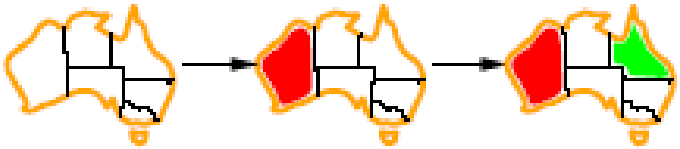
- באיור 6.3 מובא **האלגוריתם AC-3**, המחזיר פתרון שבו כל הקשתות הן קשתות עקביות, אם בעית סיפוק האילוצים פתירה. אחרת, מחזיר כשלון.
- האלגוריתם משתמש בתור המאחסן את הקשתות. (למעשה אפשר להשתמש בקבוצה כי סדר האיברים אינו חשוב).
- בתחילה מכיל התור את כל הקשתות שבבעית סיפוק האילוצים. האלגוריתם מוציא מהתור באופן שרירותי קשת (X_i, X_j) ודואג לכך שהמשתנה X_i יהיה עקבי-קשת ביחס ל- X_j.
- אם כתוצאה מכך D_i אינו משתנה אזי האלגוריתם מתקדם לקשת הבאה.
- ואילו אם התחום D_i השתנה (כלומר הצטמצם בעקבות מציאת חוסר עקביות), יתכן ויש לצמצם גם את התחום D_k ולכן מוסיפים לתור את כל הקשתות (X_k, X_i) כך ש- X_k שכן של X_i.
- אם D_i הצטמצם כך שנהיה ריק, אזי לבעיה כולה אין פתרון עקבי, והאלגוריתם מחזיר כשלון. אחרת, ממשיכים לבדוק, ומנסים למחוק מתחומי המשתנים ערכים עד שלא נשארות קשתות בתור. הבעיה שמטפלים בה כעת, עם התחומים המצומצמים, שקולה לבעיה המקורית – לשתיהן פתרון זהה – אך בדרך כלל החיפוש יעיל יותר בבעיה שבה הקשתות עקביות משום שלמשתניה תחומים קטנים יותר.

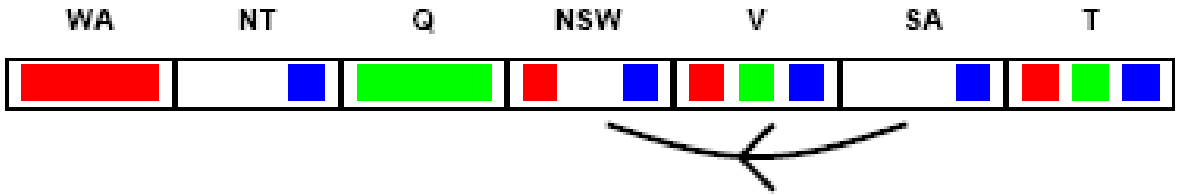
Arc consistency algorithm

```
function AC-3(csp) return the CSP, possibly with reduced domains
    inputs: csp, a binary csp with variables {X1, X2, …, Xn}
    local variables: queue, a queue of arcs initially the arcs in csp

    while queue is not empty do
            (Xi, Xj) ← REMOVE-FIRST(queue)
            if REMOVE-INCONSISTENT-VALUES(Xi, Xj)  then
                    for each Xk in NEIGHBORS[Xi ] - Xj do
                    add (Xk, Xi) to queue

function REMOVE-INCONSISTENT-VALUES(Xi, Xj) return true iff we remove a value
    removed ←  false
    for each x in DOMAIN[Xi] do
            if no value y in DOMAIN[Xi] does (x, y) satisfy constraints between Xi and Xj
            then delete x from DOMAIN[Xi];
            removed ←  true
    return removed
```

Arc consistency





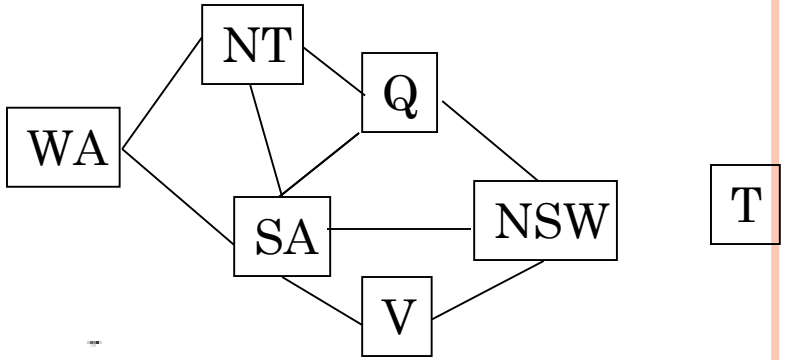


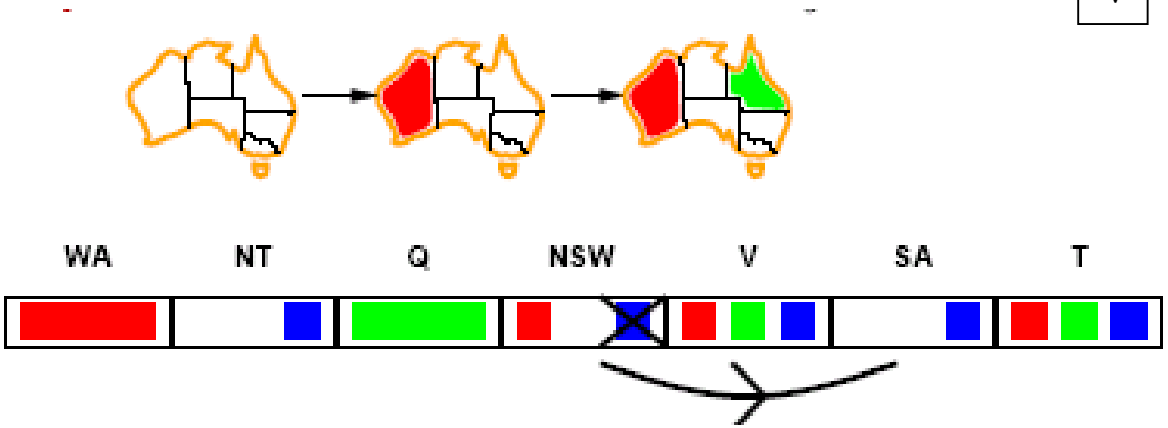
- $X \rightarrow Y$ is consistent iff

  for *every* value x of X there is some allowed y
- $SA \rightarrow NSW$ is consistent iff

  SA=*blue* and NSW=*red*

Arc consistency





- $X \rightarrow Y$ is consistent iff

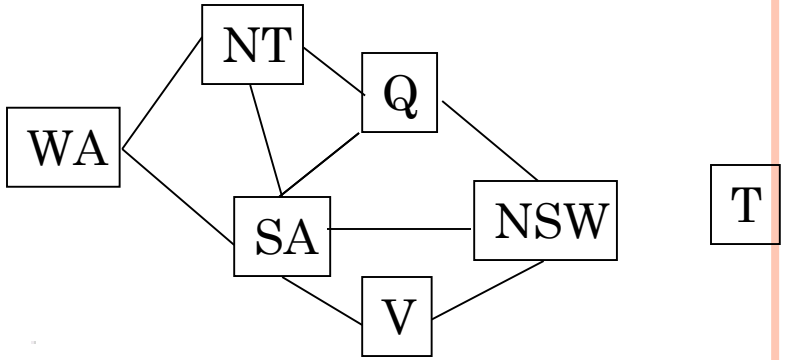
  for *every* value x of X there is some allowed y
- $NSW \rightarrow SA$ is consistent iff

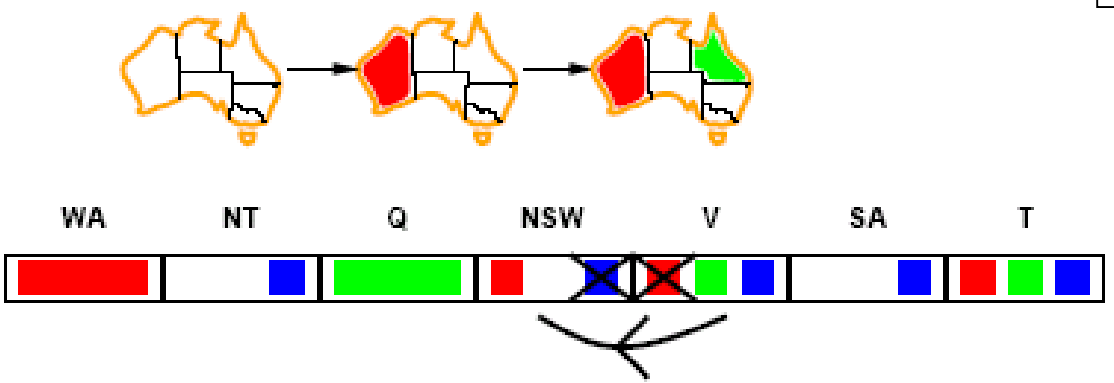
  NSW=red and *SA=blue*

  NSW=blue and SA=???

Arc can be made consistent by removing *blue* from *NSW*

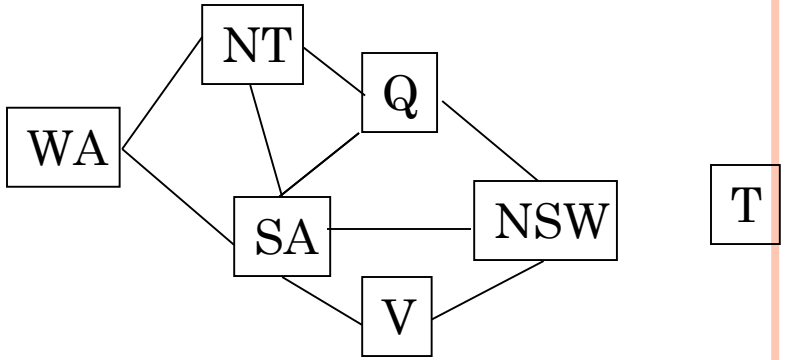
Arc consistency

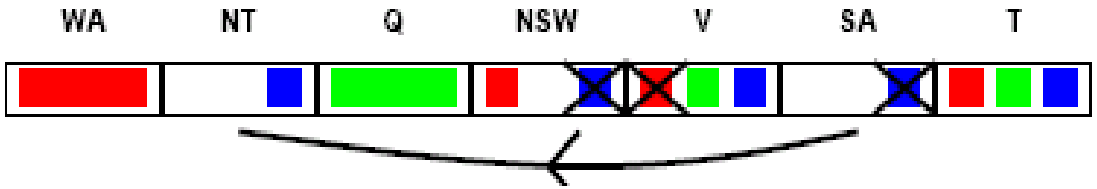




- Arc can be made consistent by removing *blue* from *NSW*
- RECHECK neighbors of *NSW*!!
  - Remove red from *V*

Arc consistency





- Arc can be made consistent by removing *blue* from *NSW*
- RECHECK neighbors of *NSW*!!
  - Remove red from *V*
- Arc consistency detects failure earlier than FC
- Can be run as a preprocessor or after each assignment.
  - Repeated until no inconsistency remains

Arc consistency algorithm AC-3

```
function AC-3( csp) returns the CSP, possibly with reduced domains
   inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
   local variables: queue, a queue of arcs, initially all the arcs in csp

   while queue is not empty do
      (Xi, Xj) ← REMOVE-FIRST(queue)
      if RM-INCONSISTENT-VALUES(Xi, Xj) then
         for each Xk in NEIGHBORS[Xi] do
            add (Xk, Xi) to queue
```

```
function RM-INCONSISTENT-VALUES( Xi, Xj) returns true iff remove a value
   removed ← false
   for each x in DOMAIN[Xi] do
      if no value y in DOMAIN[Xj] allows (x,y) to satisfy constraint(Xi, Xj)
         then delete x from DOMAIN[Xi];  removed ← true
   return removed
```

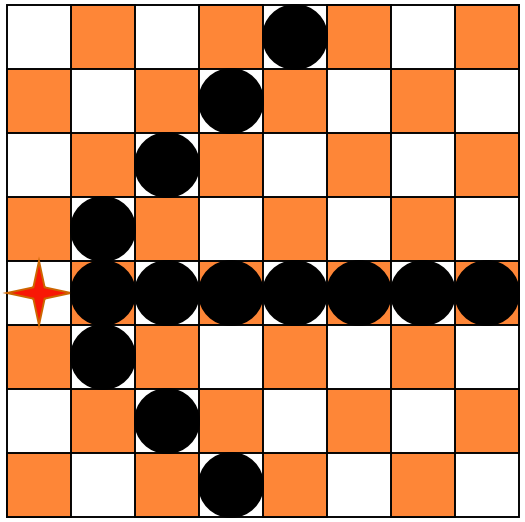
- Time complexity: $O(n^2d^3)$
-

K-CONSISTENCY

- Arc consistency does not detect all inconsistencies:
  - Partial assignment *{WA=red, NSW=red}* is inconsistent.
- Stronger forms of propagation can be defined using the notion of k-consistency.
- A CSP is k-consistent if for any set of k-1 variables and for any consistent assignment to those variables, a consistent value can always be assigned to any kth variable.
  - E.g. 1-consistency or node-consistency
  - E.g. 2-consistency or arc-consistency
  - E.g. 3-consistency or path-consistency

Forward checking is only on simple form of constraint propagation

When a pair (X←v) is added to assignment A do:
  For each variable Y not in A do:
    For every constraint C relating Y to variables in A do:
      Remove all values from Y's domain that do not satisfy C



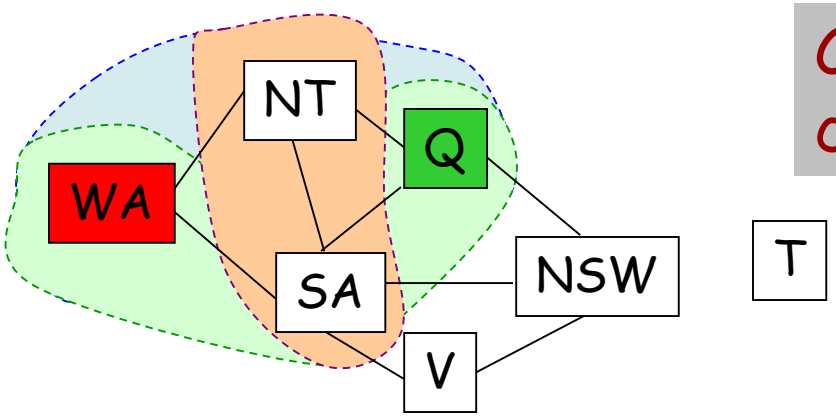
- n = number of variables
- d = size of initial domains
- s = maximum number of constraints involving a given variable ($s \leq n-1$)
- Forward checking takes O(nsd) time

Forward Checking in Map Coloring

Empty set: the current assignment
{(WA ← R), (Q ← G), (V ← B)}
does not lead to a solution

WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB
R	~~G~~B	G	R~~G~~B	RGB	~~G~~B	RGB
R	B	G	R~~B~~	B	~~B~~	RGB

Forward Checking in Map Coloring

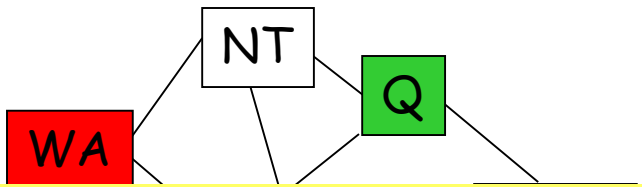


Contradiction that forward checking did not detect

WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB
R	~~G~~B	G	R~~G~~B	RGB	~~G~~B	RGB
R	B	G	R~~B~~	B	~~B~~	RGB

Forward Checking in Map Coloring

Contradiction that forward checking did not detect



Detecting this contradiction requires a more powerful constraint propagation technique

WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB
R	~~G~~B	G	R~~G~~B	RGB	~~G~~B	RGB
R	B	G	R~~B~~	B	~~B~~	RGB

Constraint Propagation for Binary Constraints

REMOVE-VALUES(X,Y) removes every value of Y that is incompatible with the values of X

REMOVE-VALUES(X,Y)

1. removed ← false
2. For every value v in the domain of Y do
   - If there is no value u in the domain of X such that the constraint on (X,Y) is satisfied then
     a. Remove v from Y's domain
     b. removed ← true
3. Return removed

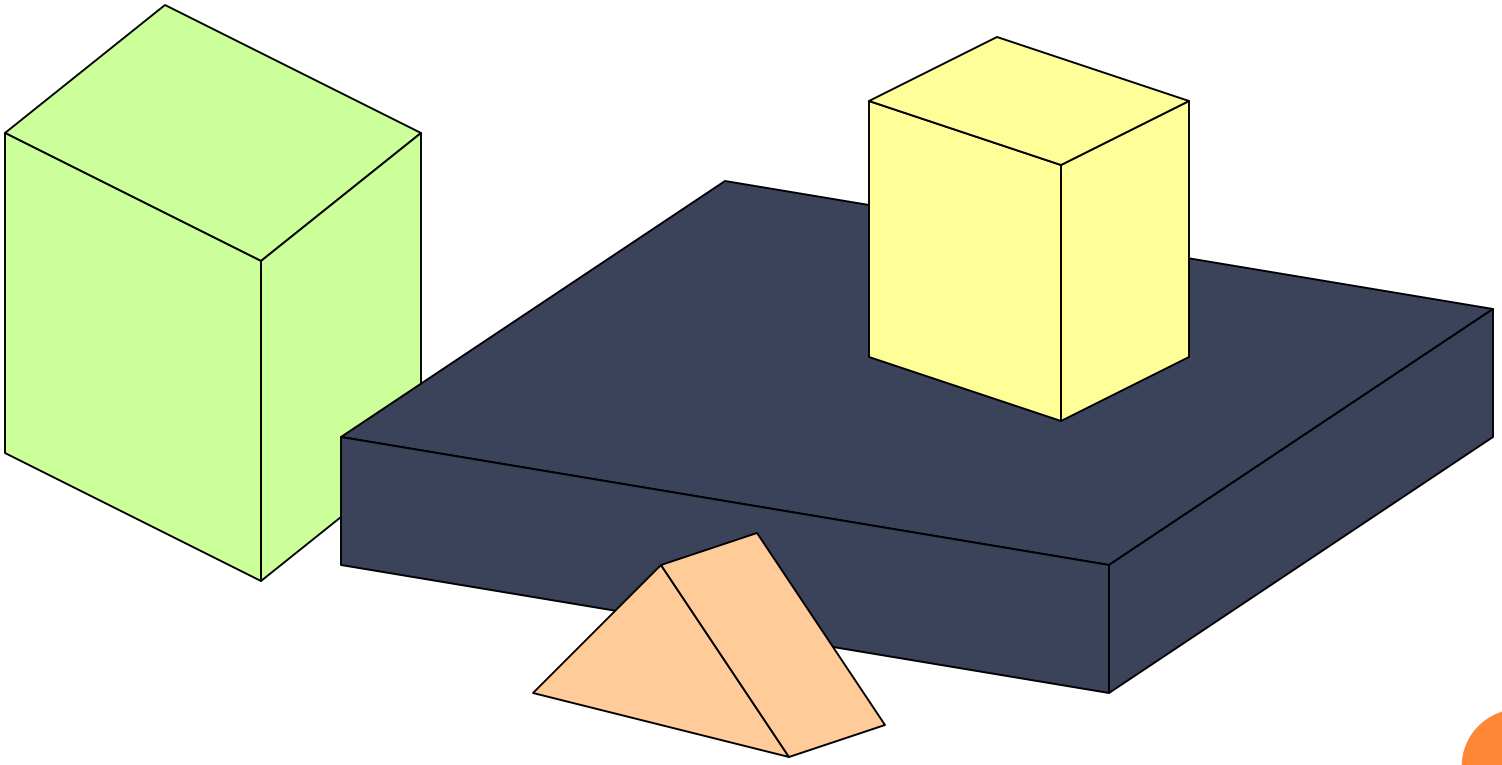
Constraint Propagation for Binary Constraints

AC3

1. Initialize queue Q with all variables (not yet instantiated)
2. While Q ≠ ∅ do
   a. X ← Remove(Q)
   b. For every (not yet instantiated) variable Y related to X by a (binary) constraint do
      - If REMOVE-VALUES(X,Y) then
        i. If Y's domain = ∅ then exit
        ii. Insert(Y,Q)

Edge Labeling

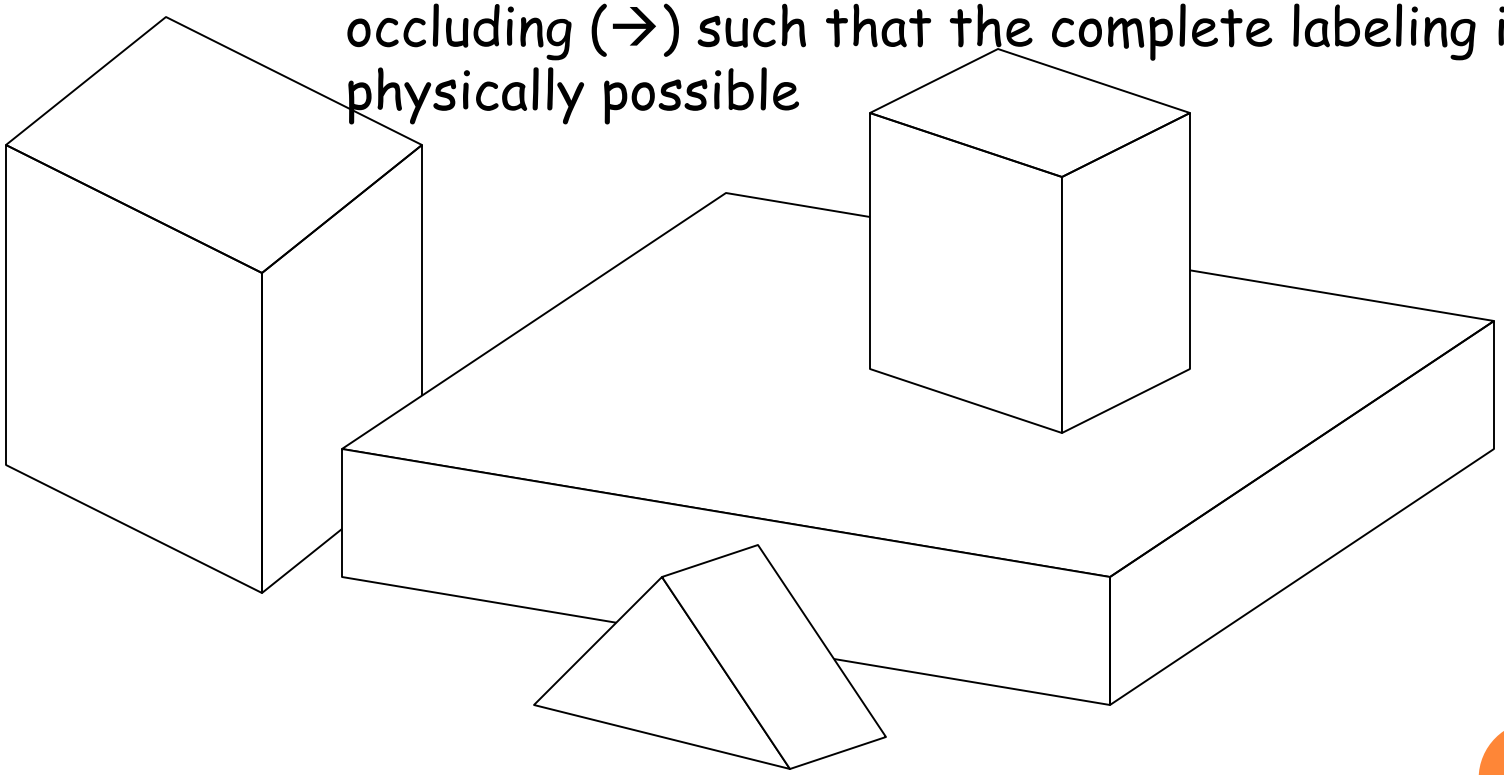
We consider an image of a scene composed of polyhedral objects such that each vertex is the endpoint of exactly three edges

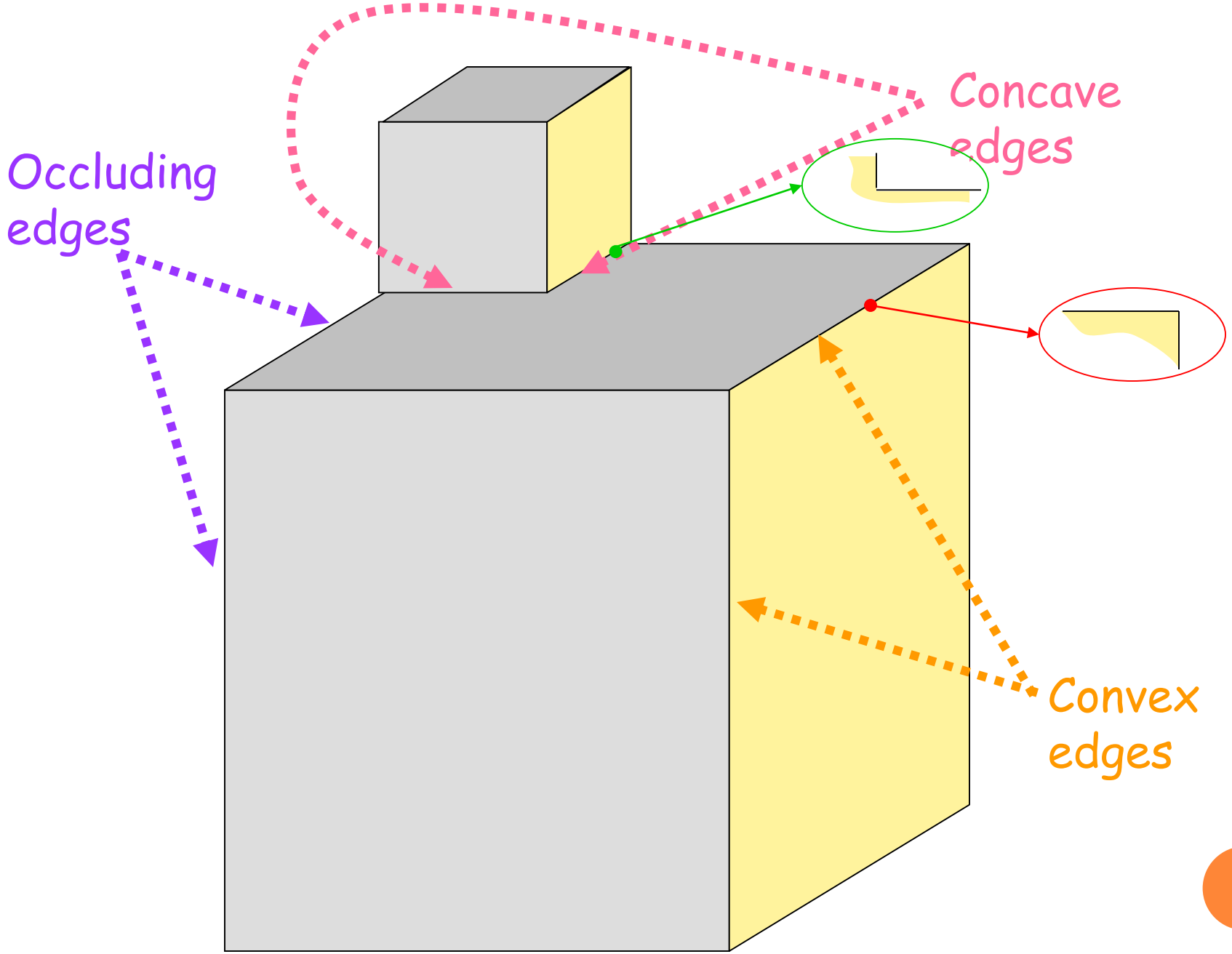


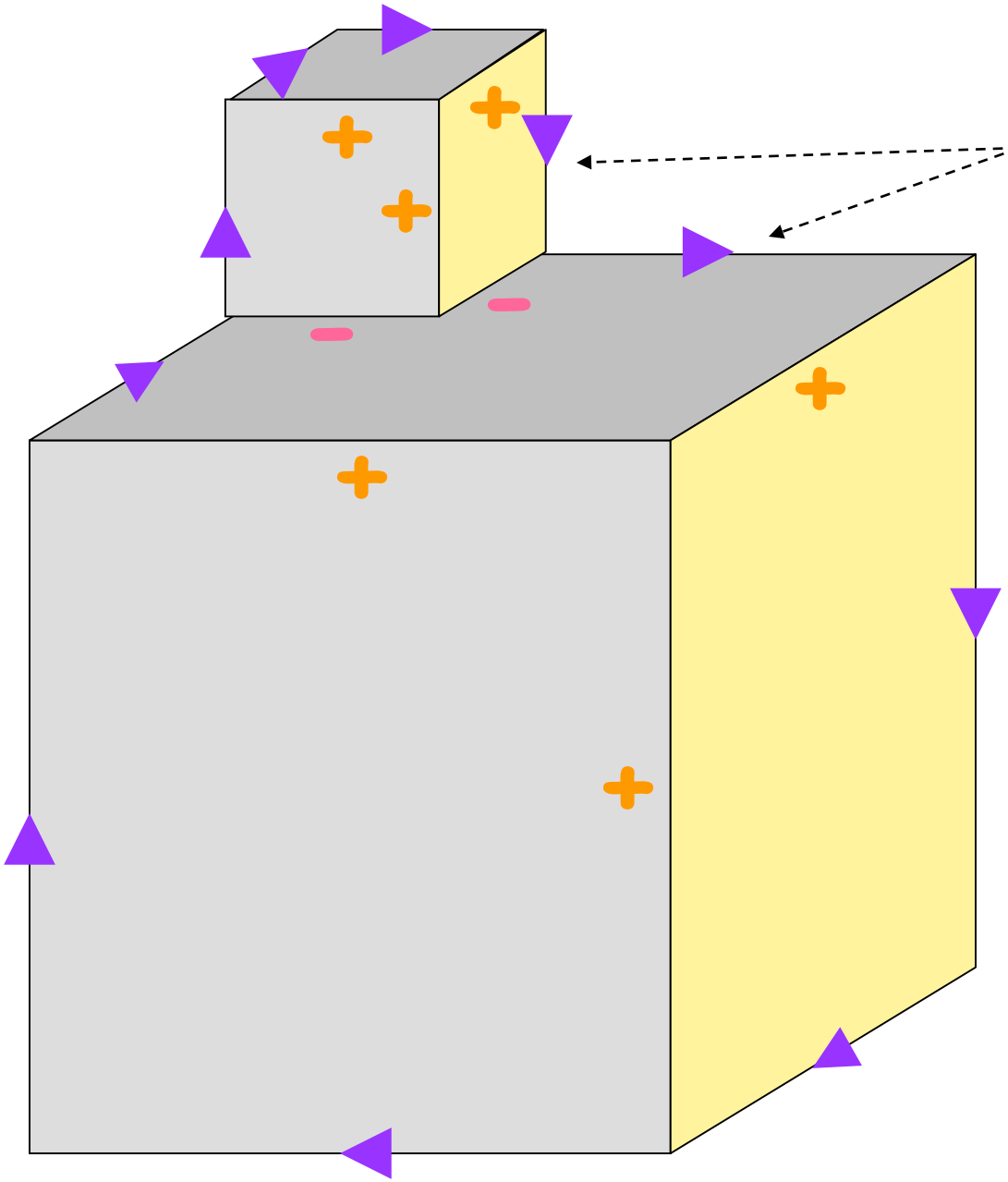
R&N: Chap. 24, pages 881-884

Edge Labeling

An "edge extractor" has accurately extracted all the visible edges in the image. The problem is to label each edge as convex (+), concave (-), or occluding (→) such that the complete labeling is physically possible

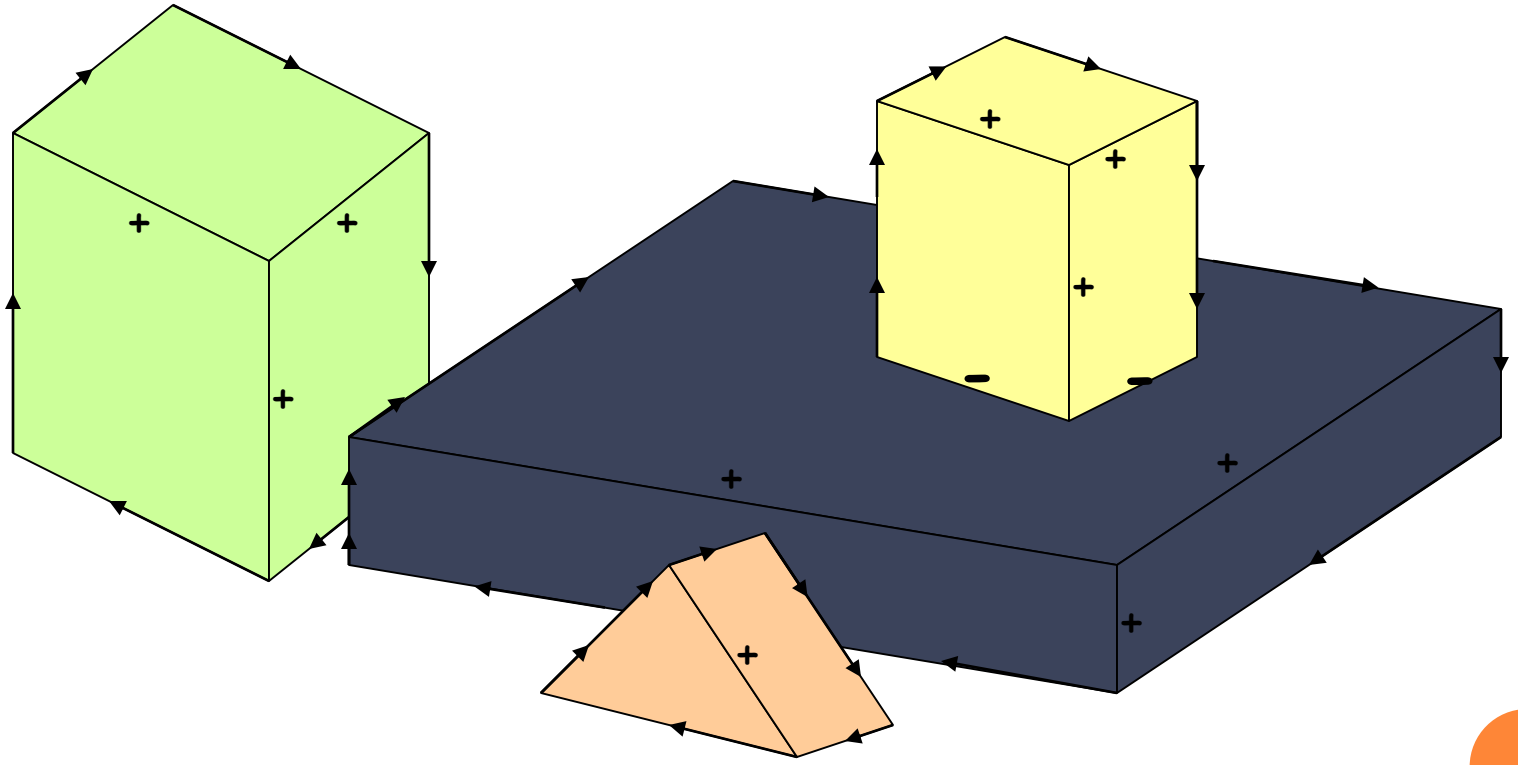



Occluding edges
Concave edges
Convex edges

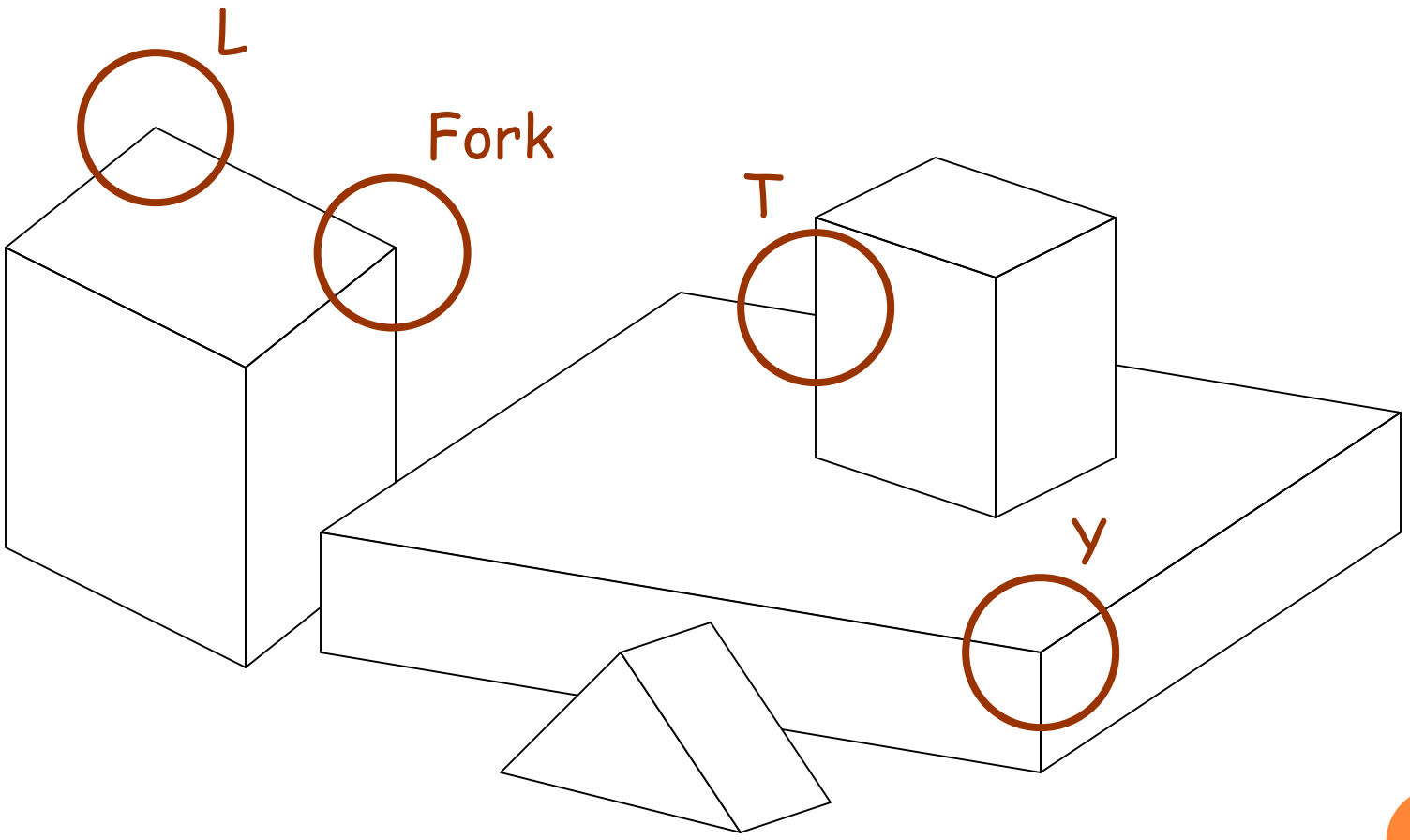


The arrow is oriented such that the object is on the right of the occluding edge

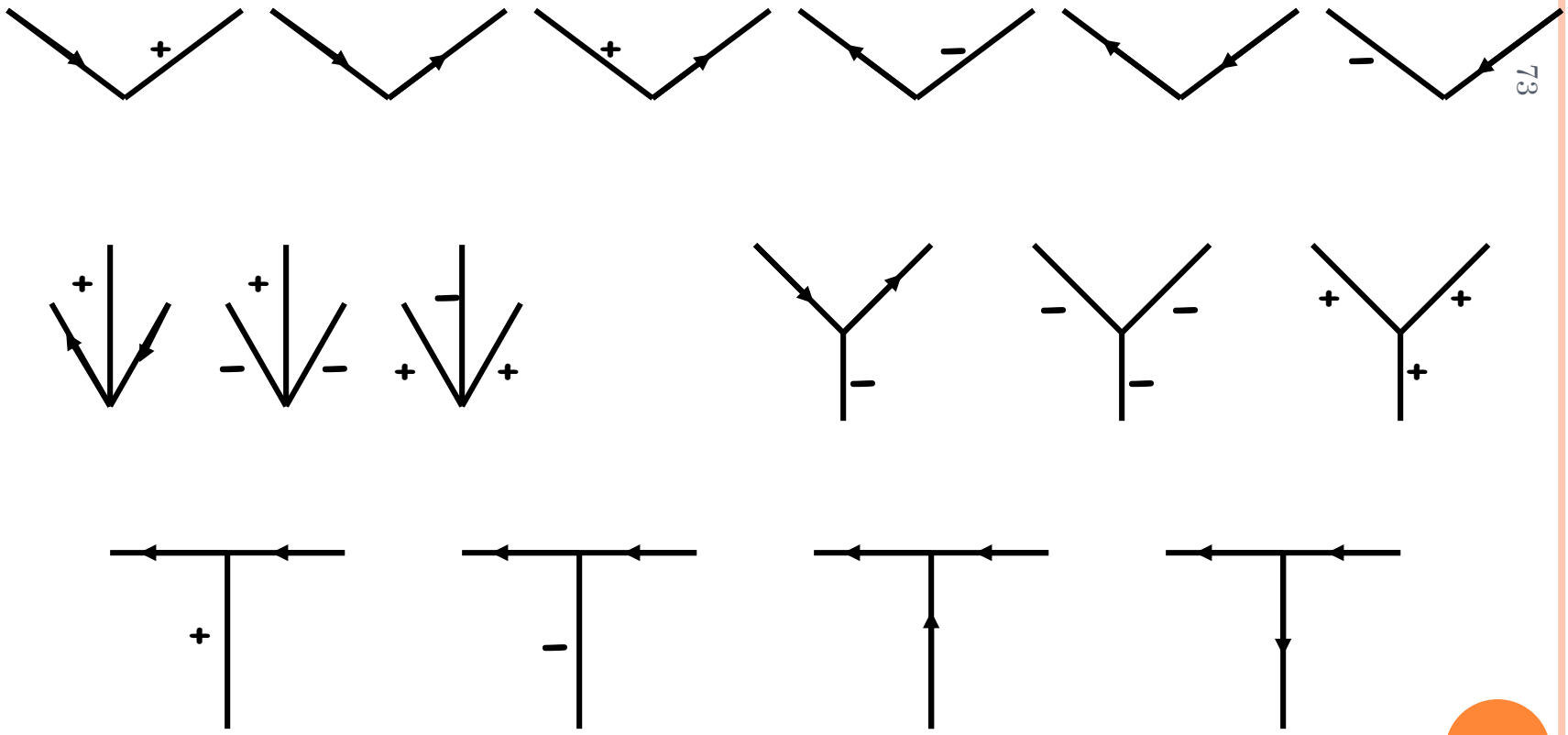
One Possible Edge Labeling



Junction Types



Junction Label Sets



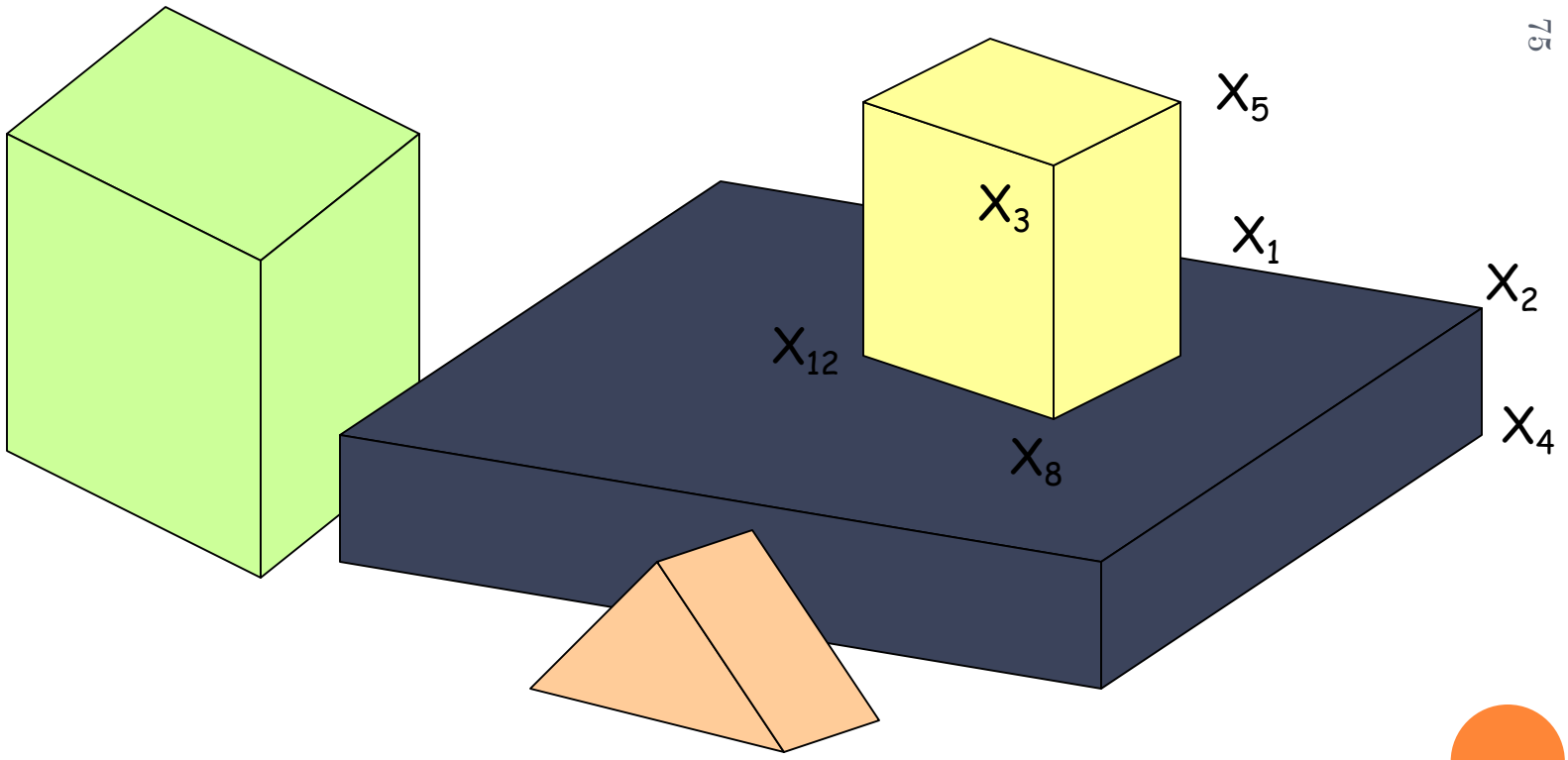
(Waltz, 1975; Mackworth, 1977)

Edge Labeling as a CSP

- A variable is associated with each junction
- The domain of a variable is the label set associated with the junction type
- Constraints: The values assigned to two adjacent junctions must give the same label to the joining edge

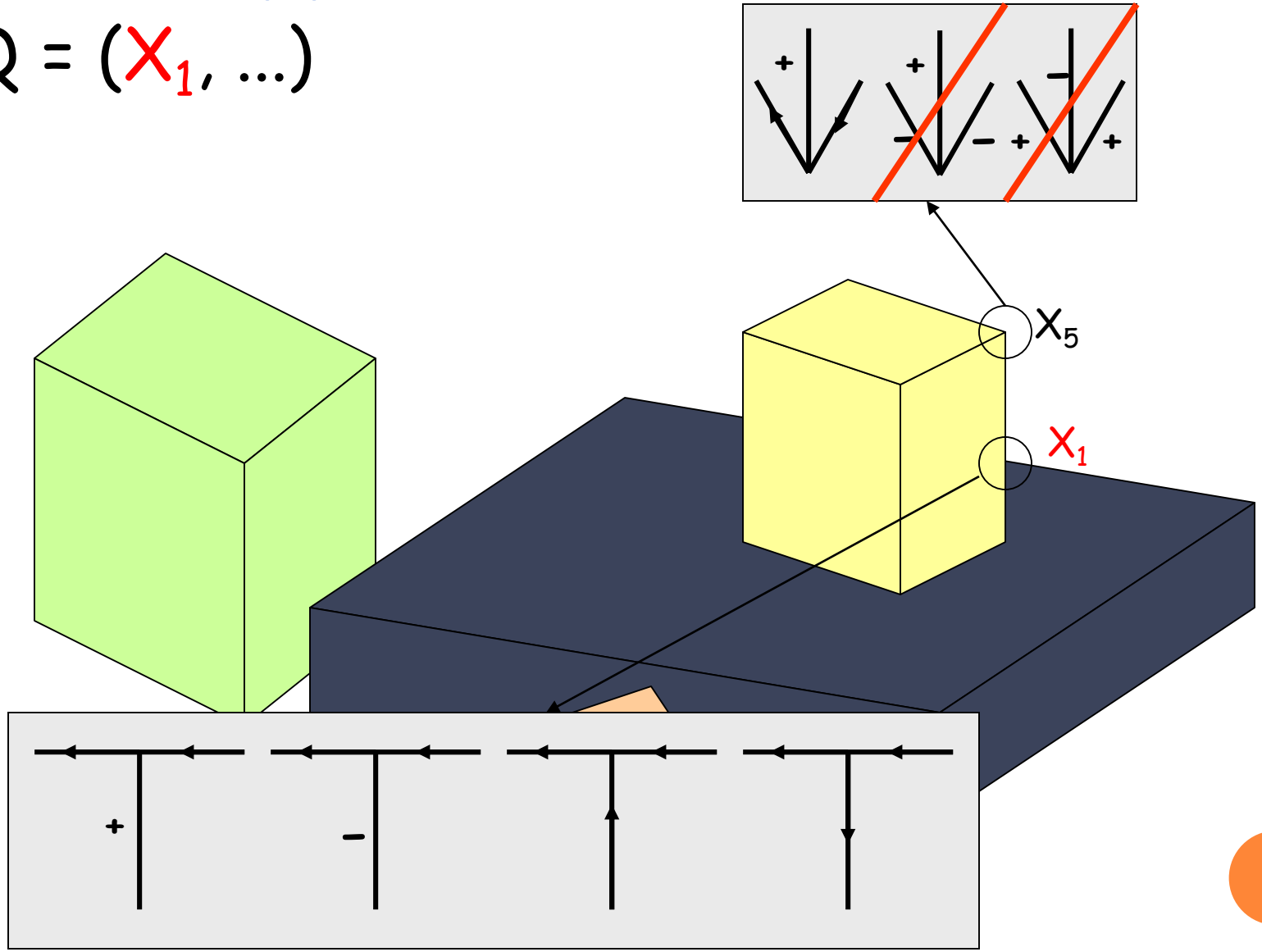
AC3 Applied to Edge Labeling

$Q = (X_1, X_2, X_3, \ldots)$

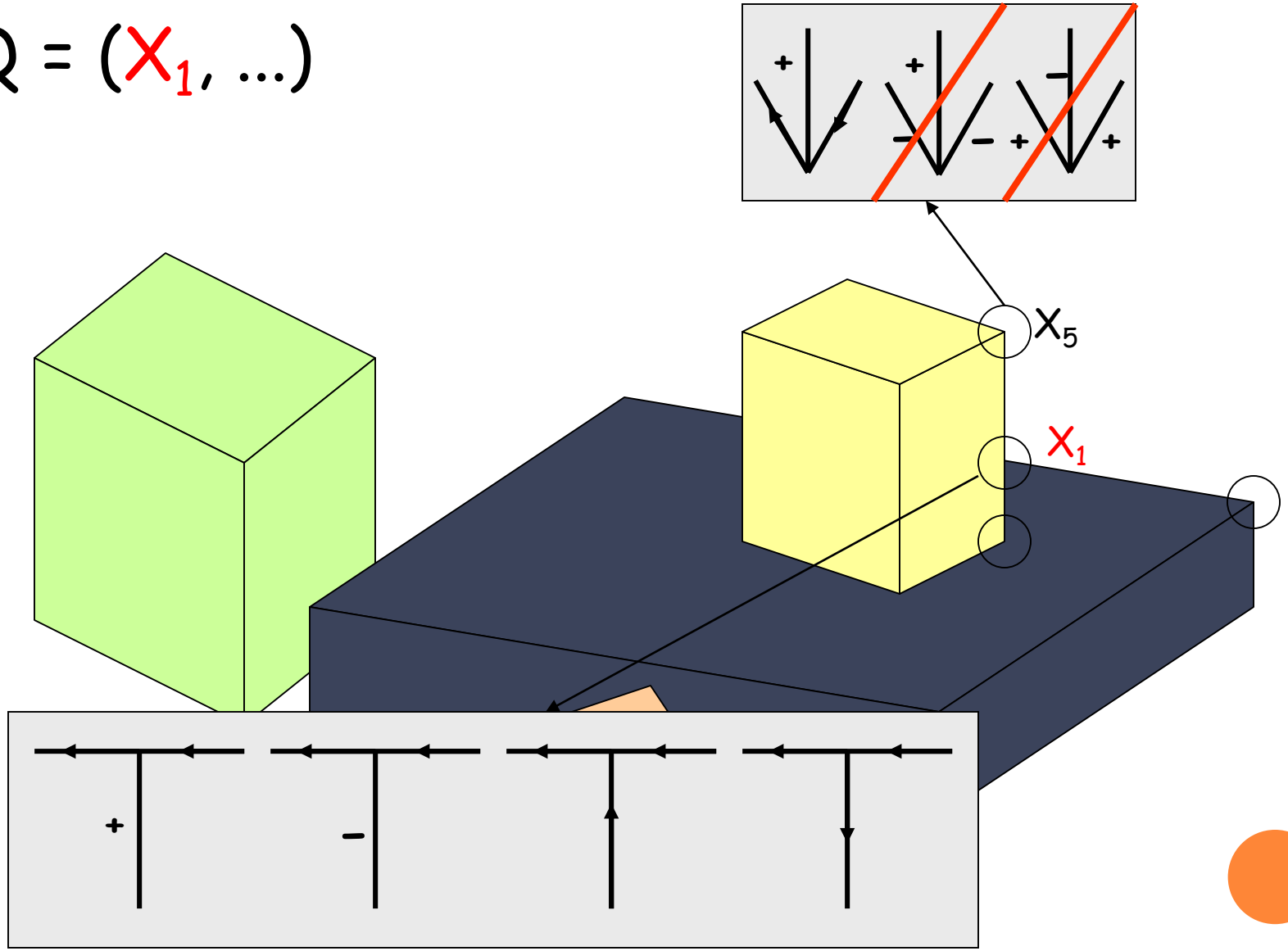


AC3 Applied to Edge Labeling

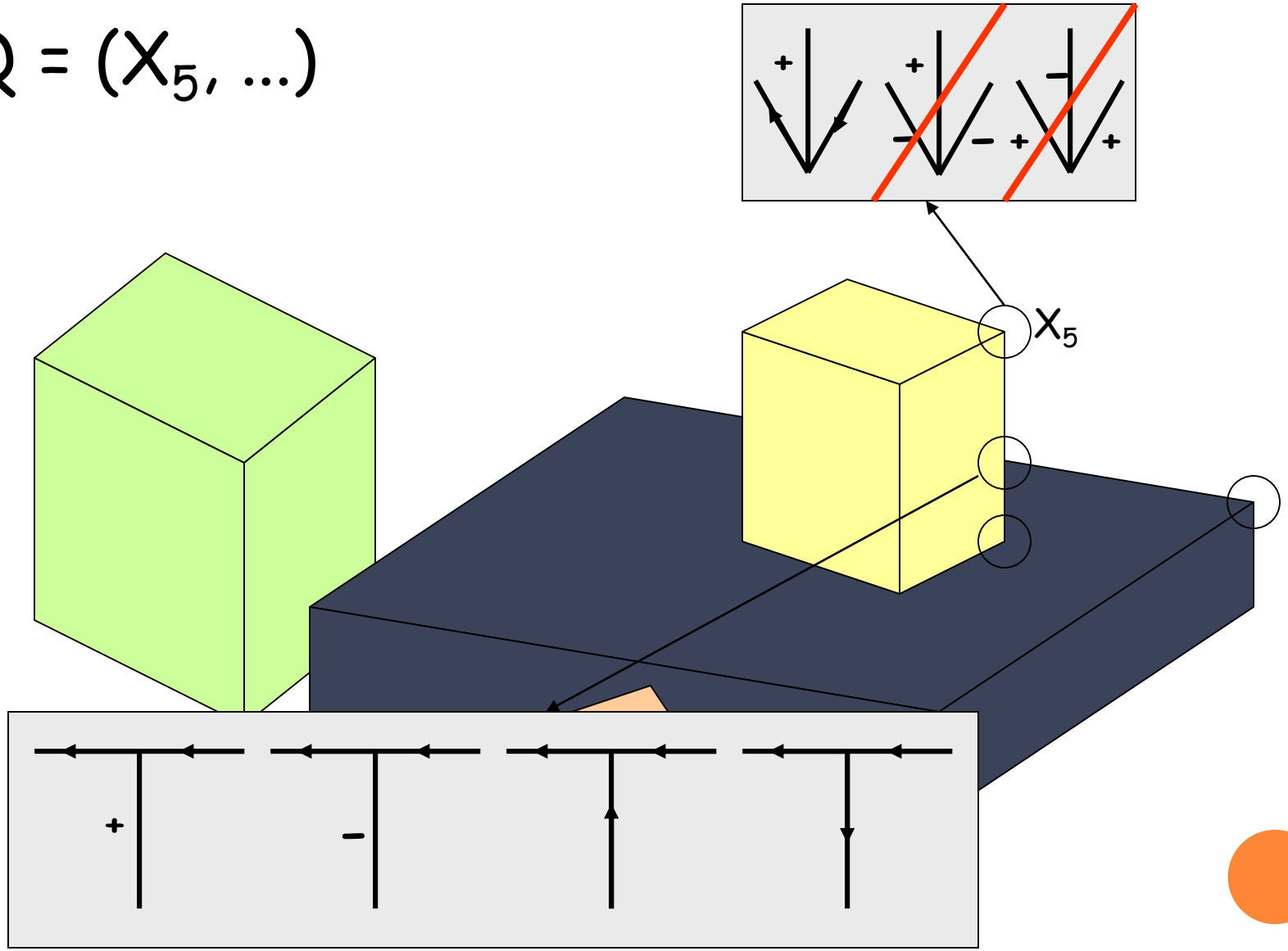
$Q = (X_1, \ldots)$



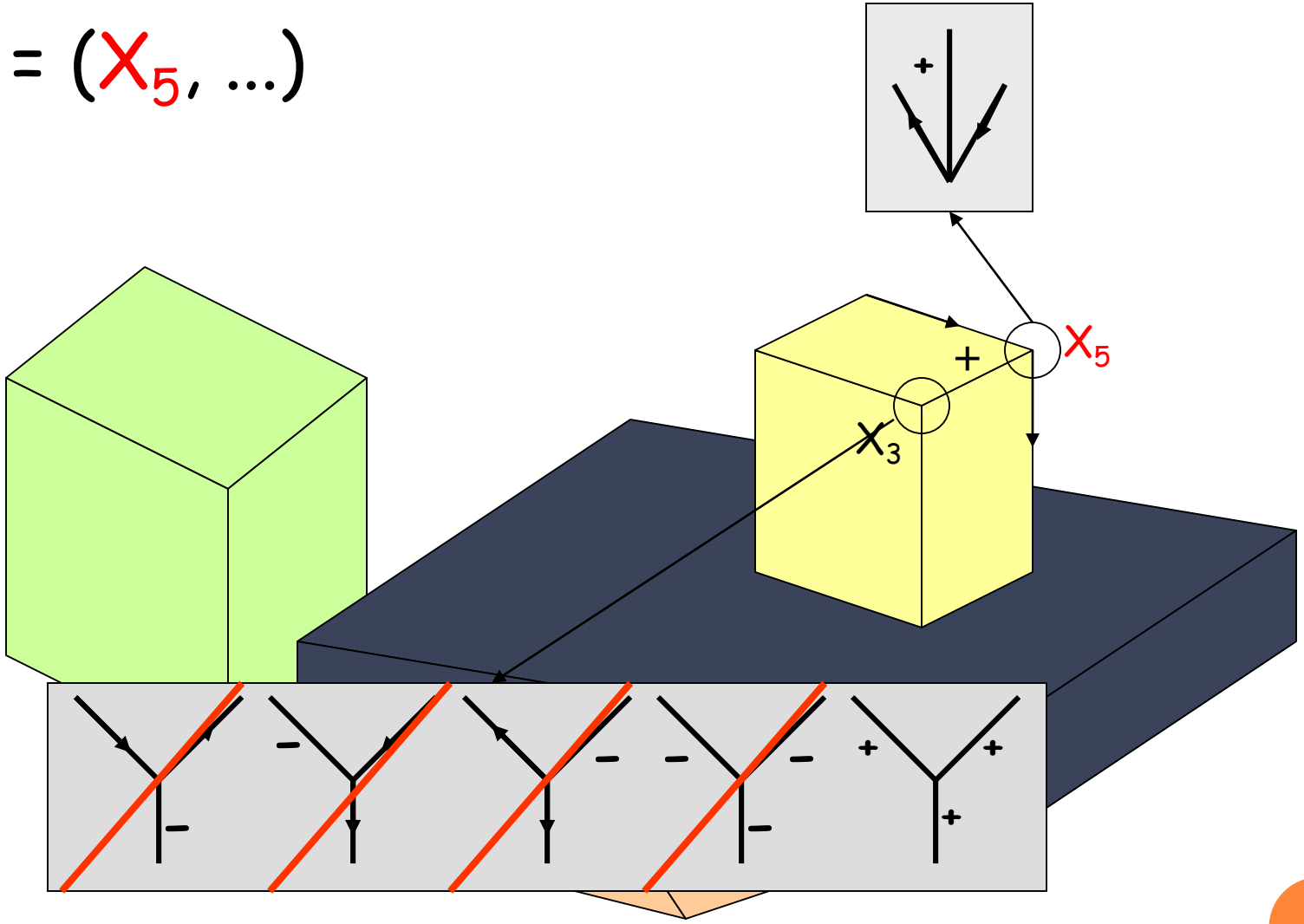
$Q = (X_1, \ldots)$



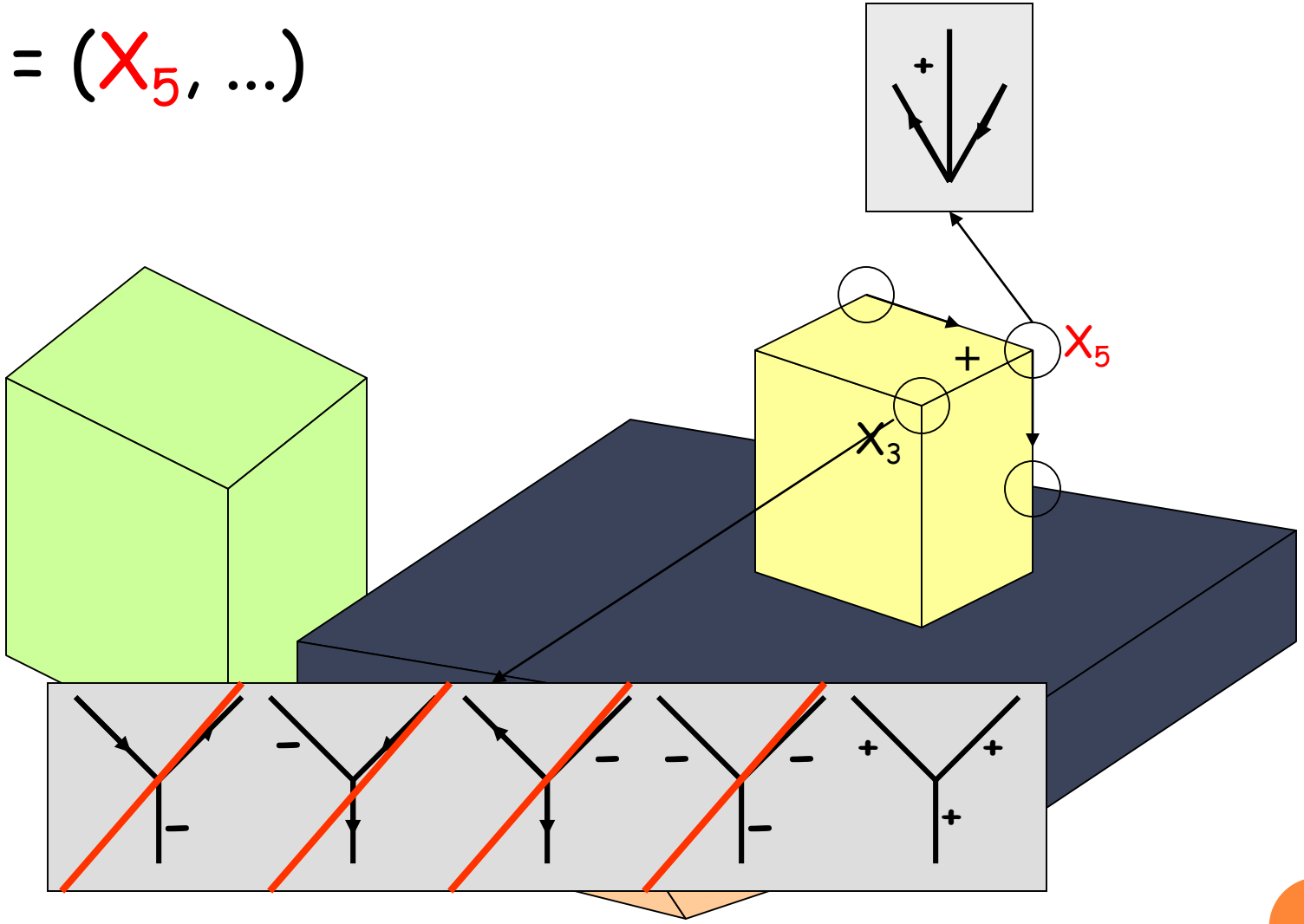
$Q = (X_5, \ldots)$



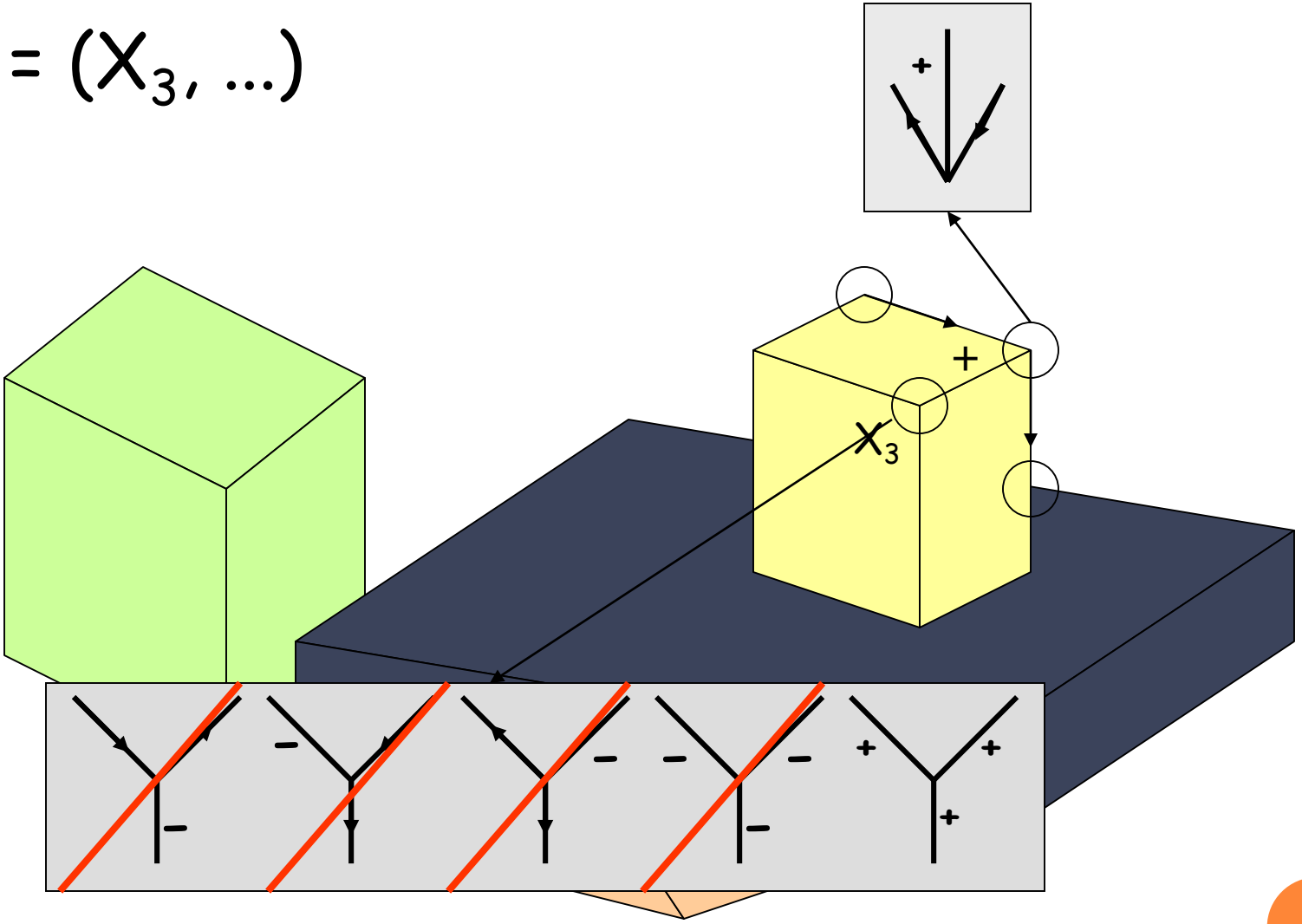
$Q = (X_5, \ldots)$



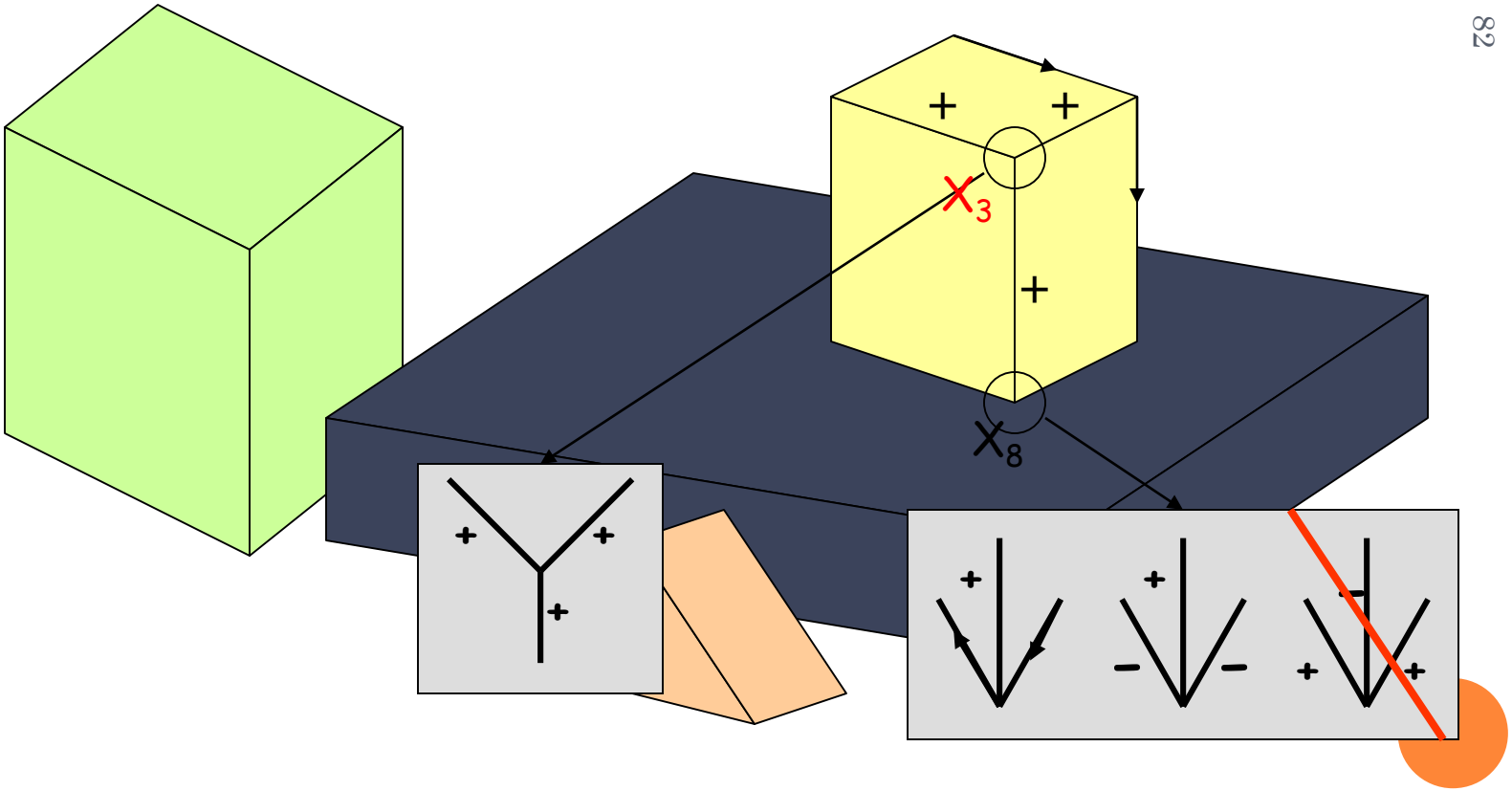
$Q = (X_5, \ldots)$



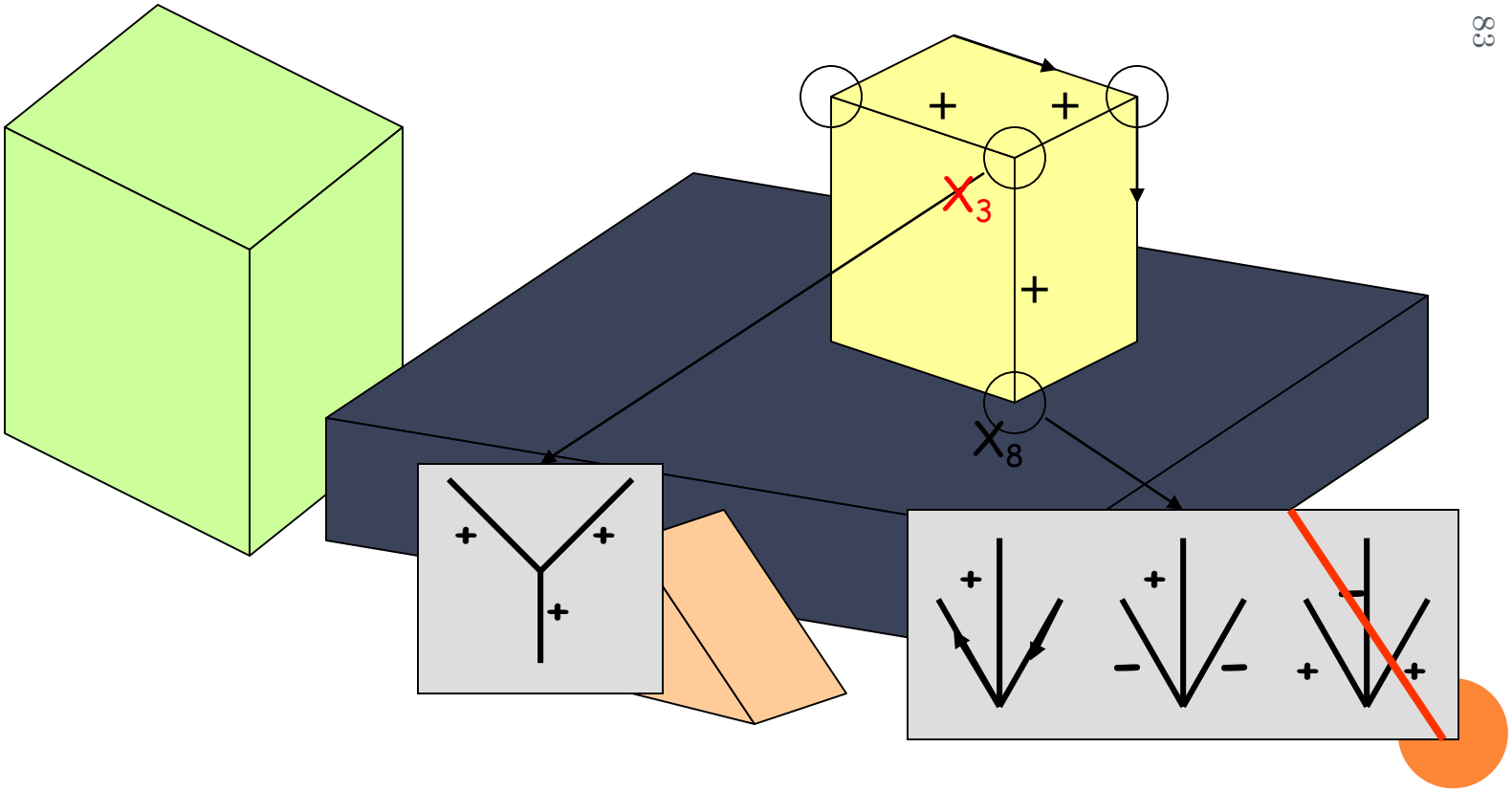
$Q = (X_3, \ldots)$



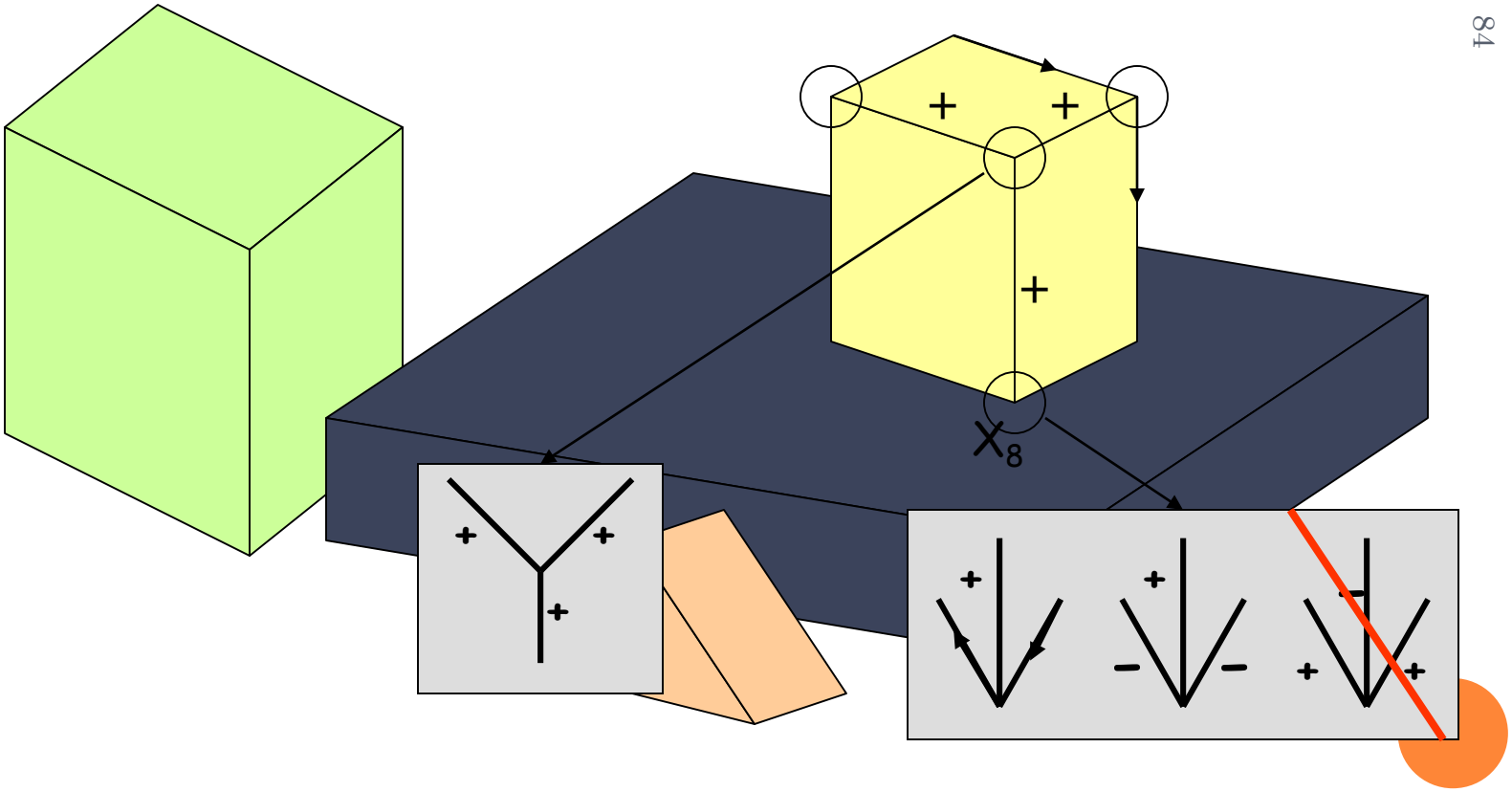
$Q = (X_3, \ldots)$



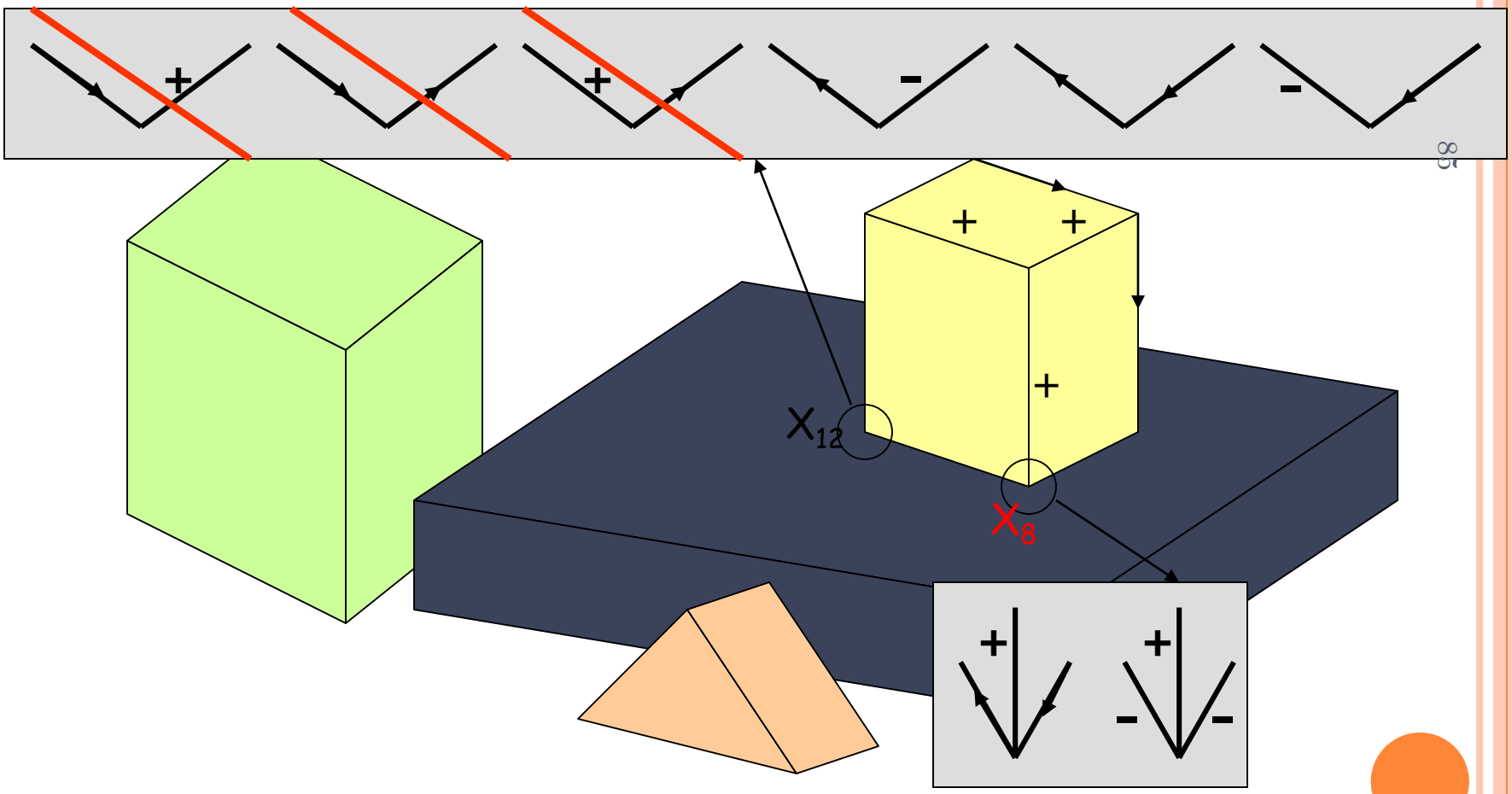
$Q = (X_3, \ldots)$



$Q = (X_8, \ldots)$



$Q = (X_8, \ldots)$



Complexity Analysis of AC3

- n = number of variables
- d = size of initial domains
- s = maximum number of constraints involving a given variable ($s \leq n-1$)
- Each variable is inserted in Q up to d times
- REMOVE-VALUES takes $O(d^2)$ time
- AC3 takes $O(n \times d \times s \times d^2) = O(n \times s \times d^3)$ time
- Usually more expensive than forward checking

AC3

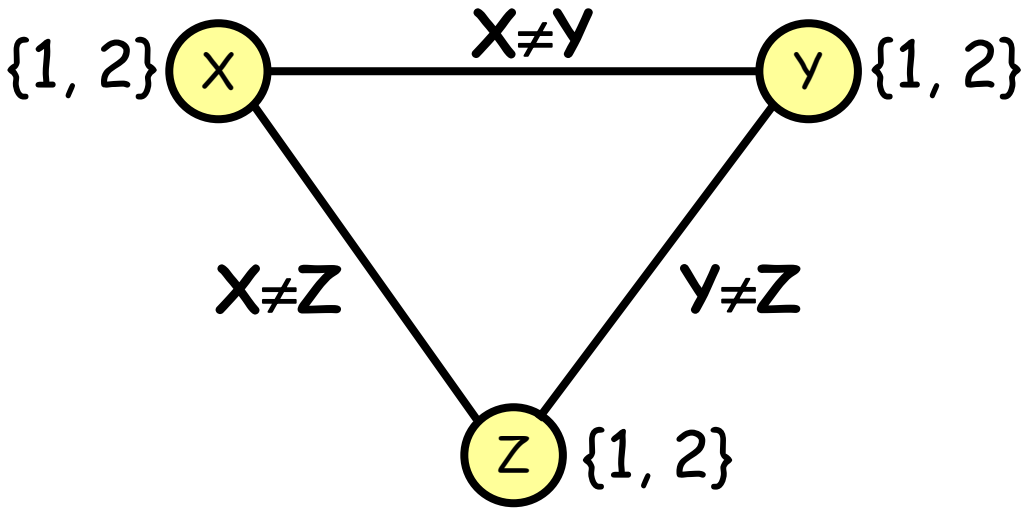
1. Initialize queue Q with all variables (not yet instantiated)
2. While Q ≠ ∅ do
   a. X ← Remove(Q)
   b. For every (not yet instantiated) variable Y related to X by a (binary) constraint do
      - If REMOVE-VALUES(X,Y) then
        i. If Y's domain = ∅ then exit
        ii. Insert(Y,Q)

REMOVE-VALUES(X,Y)

1. removed ← false
2. For every value v in the domain of Y do
   - If there is no value u in the domain of X such that the constraint on (x,y) is satisfied then
     a. Remove v from Y's domain
     b. removed ← true
3. Return removed

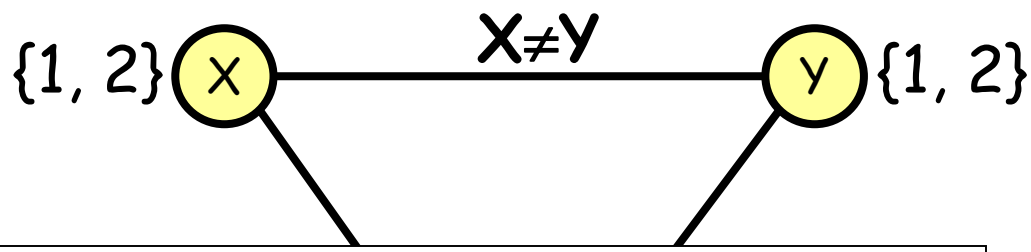
Is AC3 all that we need?

- No !!
- AC3 can't detect all contradictions among binary constraints



Is AC3 all that we need?

- No !!
- AC3 can't detect all contradictions among binary constraints

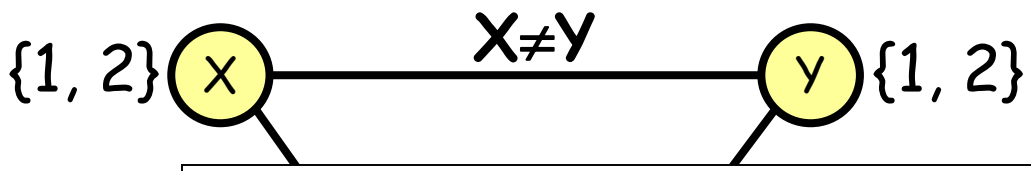


REMOVE-VALUES(X,Y)

1. removed ← false
2. For every value v in the domain of Y do
   - If there is no value u in the domain of X such that the constraint on (X,Y) is satisfied then
     a. Remove v from Y's domain
     b. removed ← true
3. Return removed

Is AC3 all that we need?

- No !!
- AC3 can't detect all contradictions among binary constraints

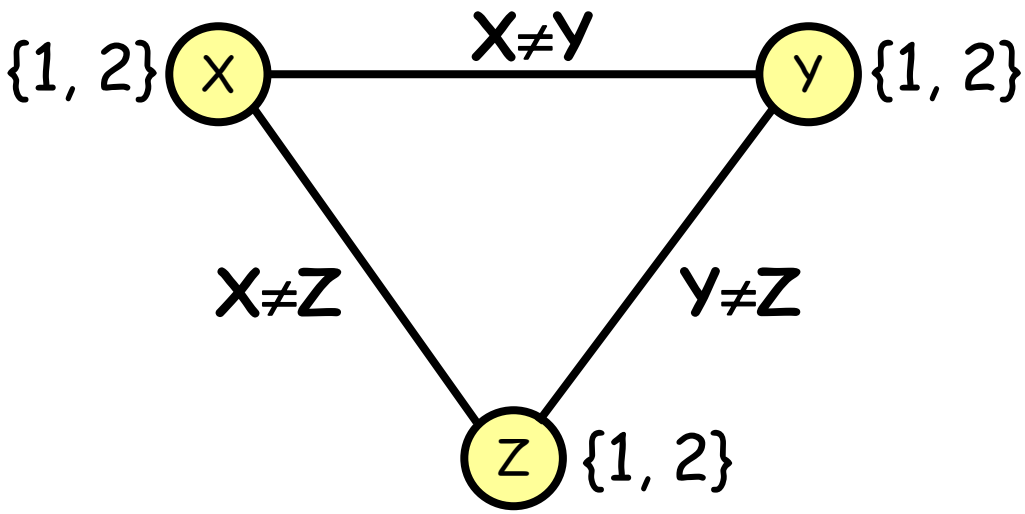


REMOVE-VALUES(X,Y
1. removed ← false
2. For every value v
   - If there is no
     that the const
     a. Remove v
     b. removed
3. Return removed

REMOVE-VALUES(X,Y,Z)
1. removed ← false
2. For every value w in the domain of Z do
   - If there is no pair (u,v) of values in the domains of X and Y verifying the constraint on (X,Y) such that the constraints on (X,Z) and (Y,Z) are satisfied then
     a. Remove w from Z's domain
     b. removed ← true
3. Return removed

Is AC3 all that we need?

- No !!
- AC3 can't detect all contradictions among binary constraints



- Not all constraints are binary

Tradeoff

Generalizing the constraint propagation algorithm increases its time complexity

→ Tradeoff between time spent in backtracking search and time spent in constraint propagation

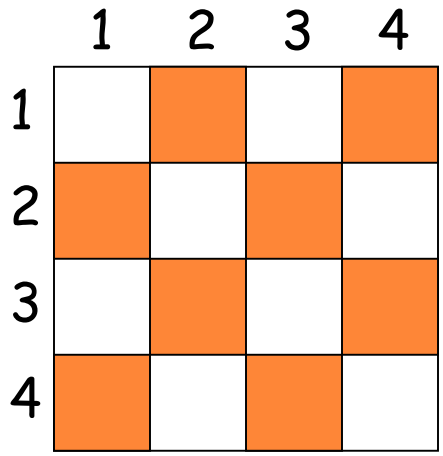
A good tradeoff when all or most constraints are binary is often to combine backtracking with forward checking and/or AC3 (with REMOVE-VALUES for two variables)

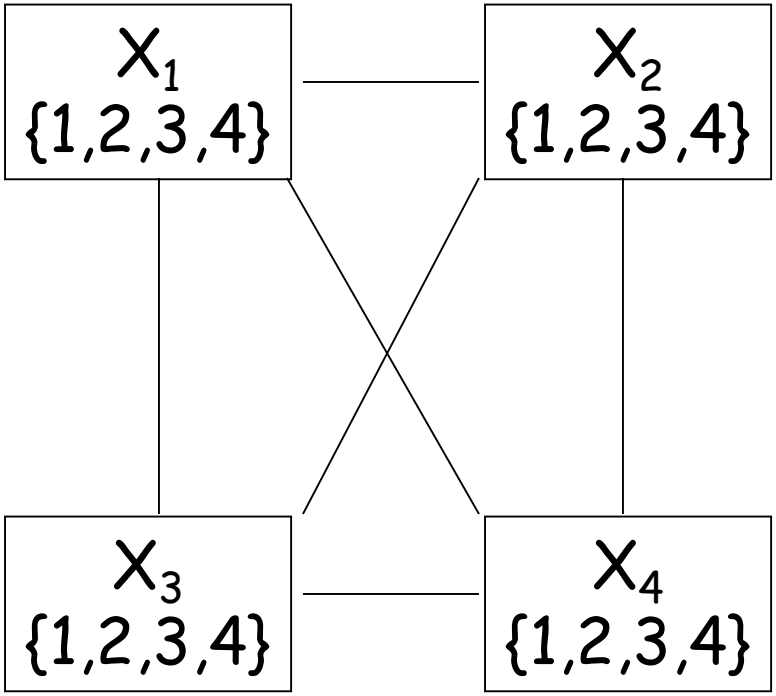
Modified Backtracking Algorithm with AC3

CSP-BACKTRACKING(A, var-domains)

1. If assignment A is complete then return A
2. Run AC3 and update var-domains accordingly
3. If a variable has an empty domain then return failure
4. X ← select a variable not in A
5. D ← select an ordering on the domain of X
6. For each value v in D do
   a. Add (X←v) to A
   b. var-domains ← forward checking(var-domains, X, v, A)
   c. If no variable has an empty domain then
      (i) result ← CSP-BACKTRACKING(A, var-domains)
      (ii) If result ≠ failure then return result
   d. Remove (X←v) from A
7. Return failure

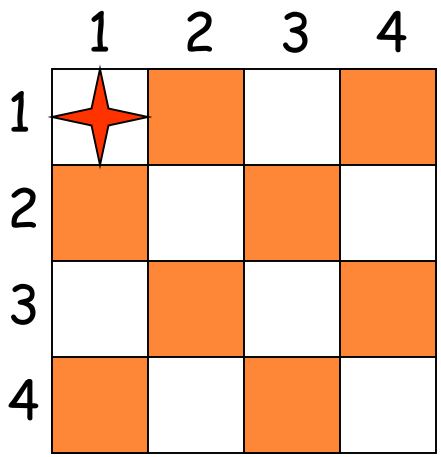
A Complete Example: 4-Queens Problem

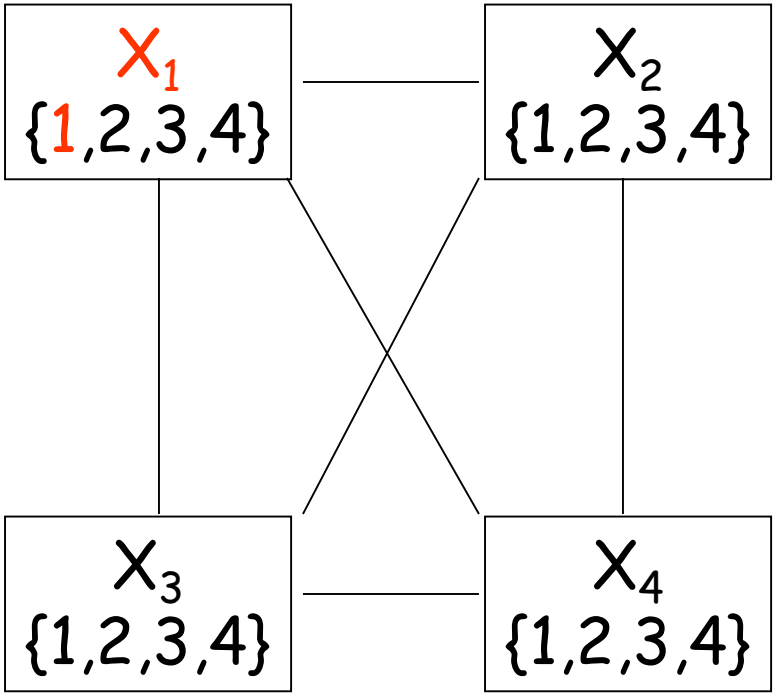




1) The modified backtracking algorithm starts by calling AC3, which removes no value

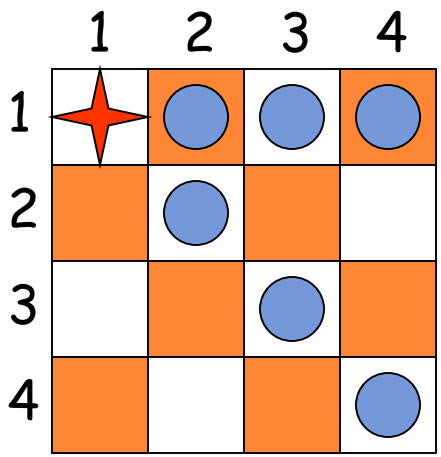
4-Queens Problem

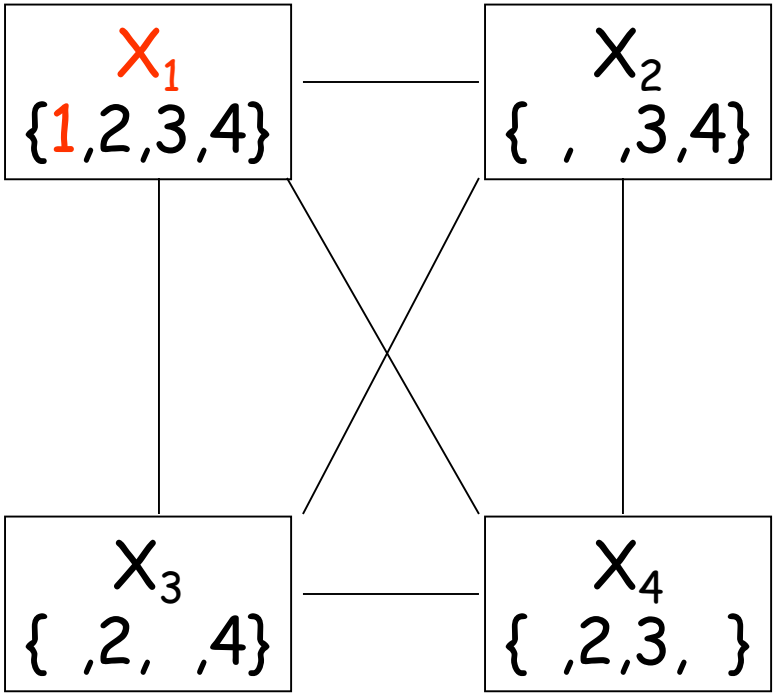




2) The backtracking algorithm then selects a variable and a value for this variable. No heuristic helps in this selection. X_1 and the value 1 are arbitrarily selected

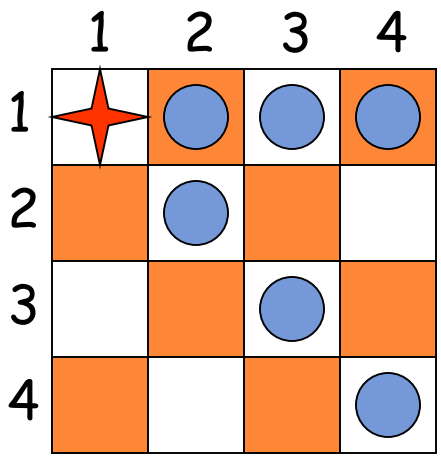
4-Queens Problem

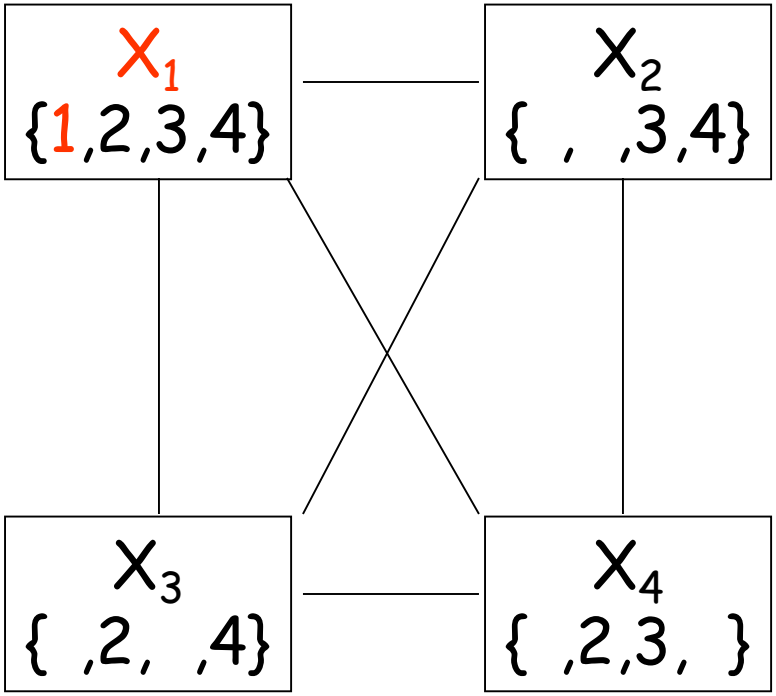




3) The algorithm performs forward checking, which eliminates 2 values in each other variable's domain

4-Queens Problem



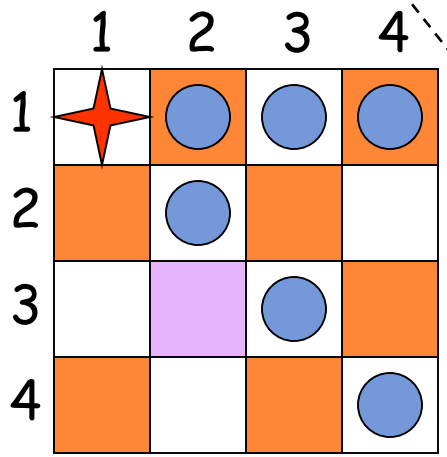


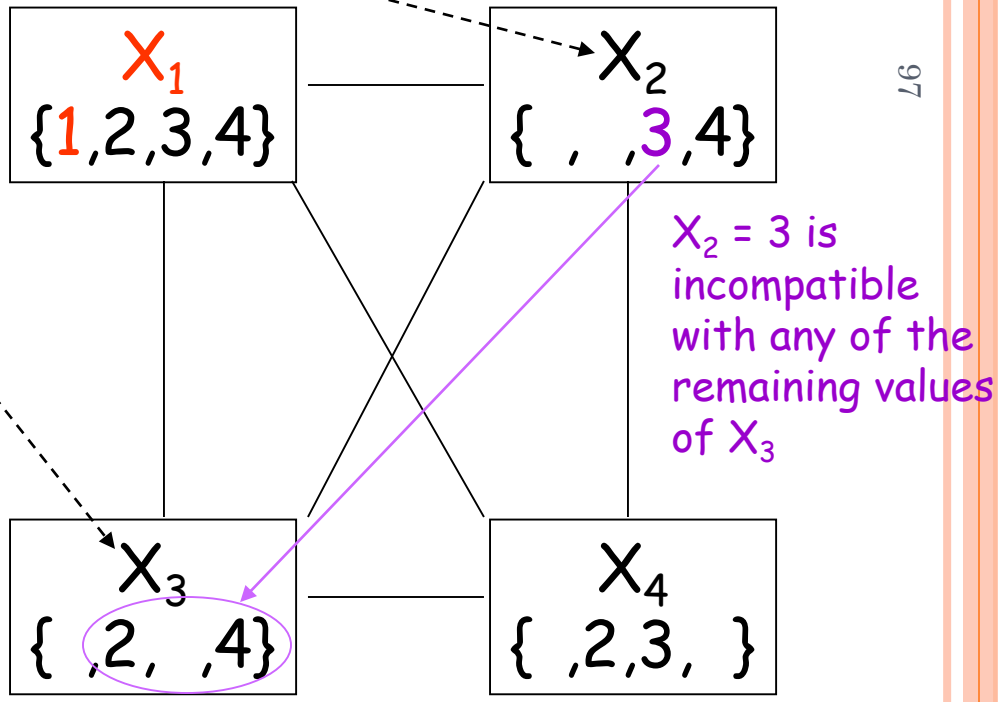
4) The algorithm calls AC3

4-Queens Problem

REMOVE-VALUES(X,Y)

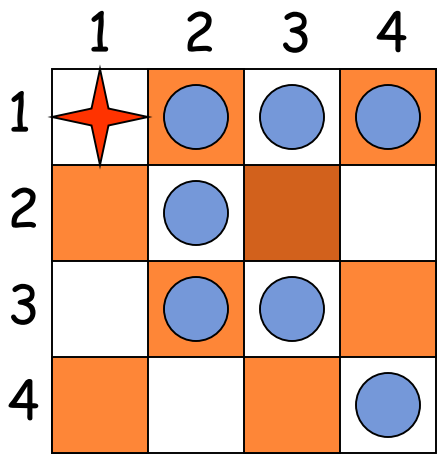
1. removed ← false
2. For every value v in the domain of Y do
   - If there is no value u in the domain of X such that the constraint on (x,y) is satisfied then
     a. Remove v from Y's domain
     b. removed ← true
3. Return removed

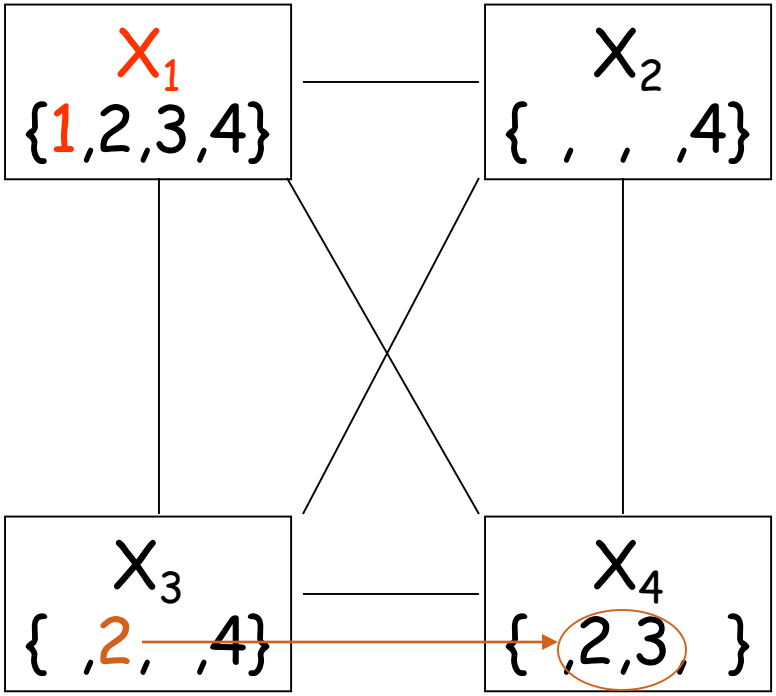




4) The algorithm calls AC3, which eliminates 3 from the domain of X_2

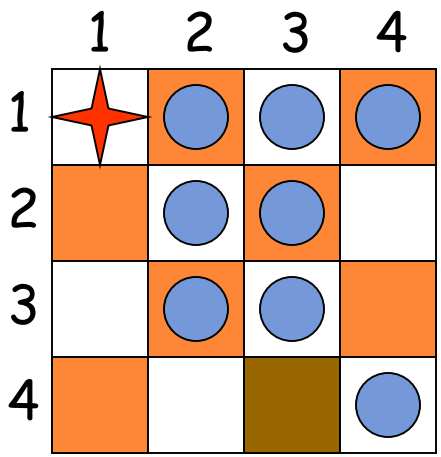
4-Queens Problem

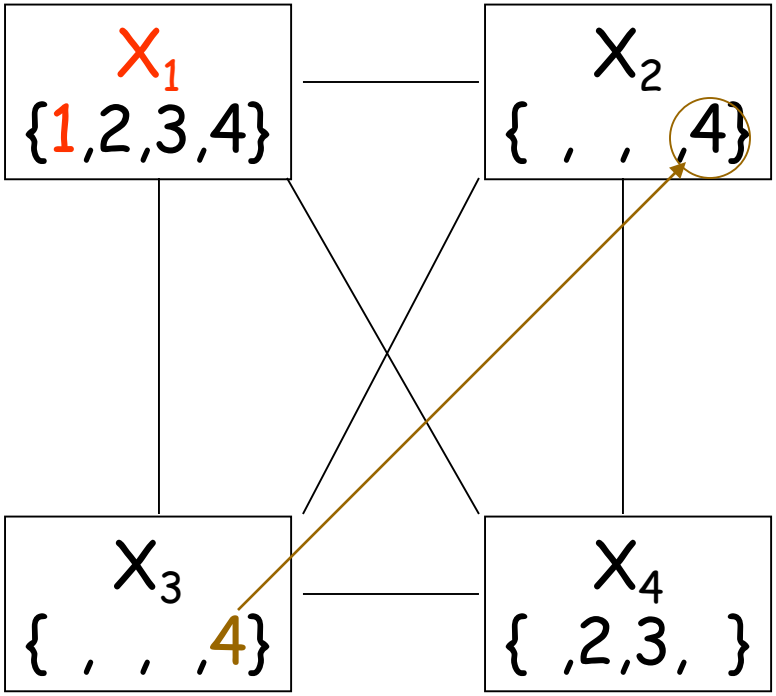




4) The algorithm calls AC3, which eliminates 3 from the domain of X_2, and 2 from the domain of X_3

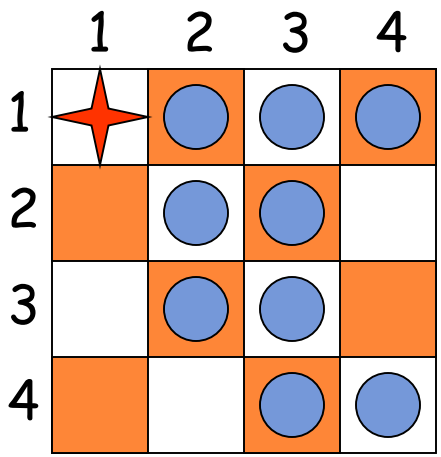
4-Queens Problem

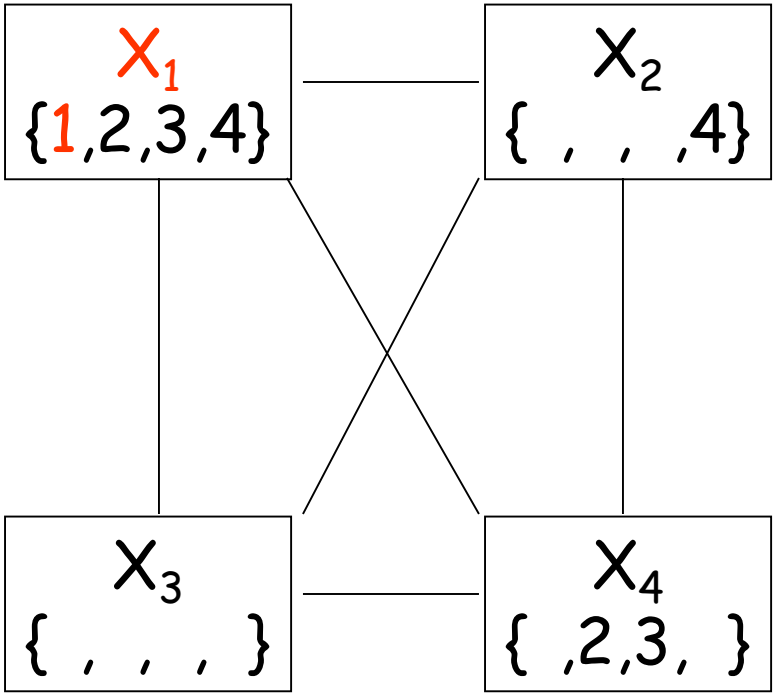




4) The algorithm calls AC3, which eliminates 3 from the domain of X_2, and 2 from the domain of X_3, and 4 from the domain of X_3

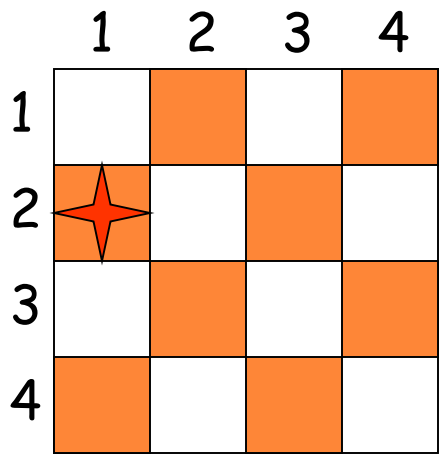
4-Queens Problem

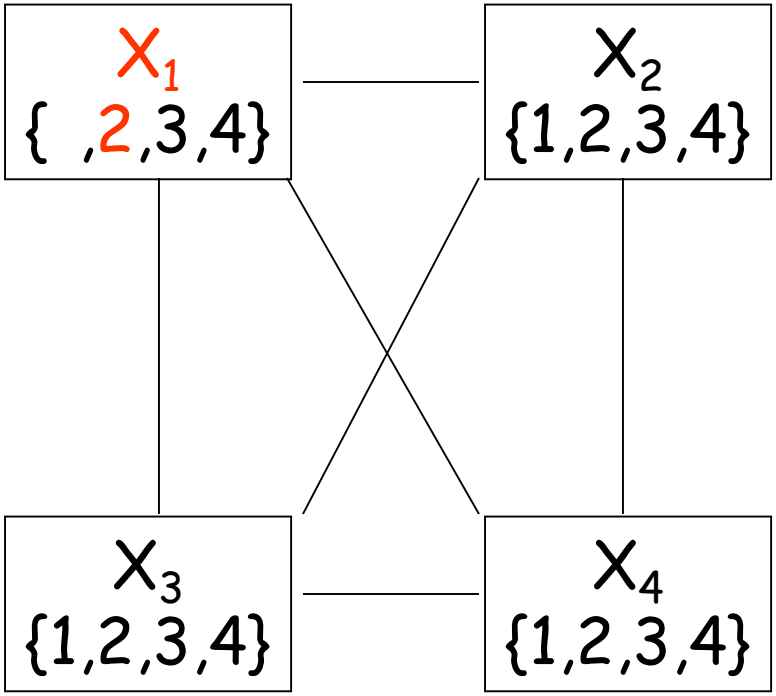




5) The domain of X_3 is **empty** → backtracking

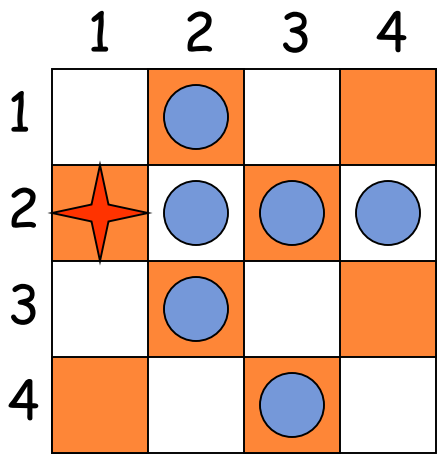
4-Queens Problem

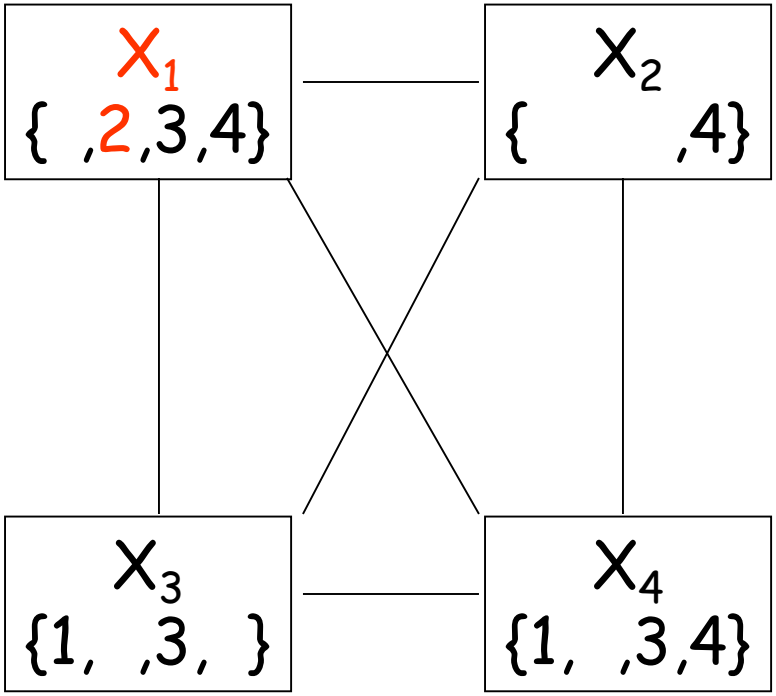




6) The algorithm removes 1 from X_1's domain and assign 2 to X_1

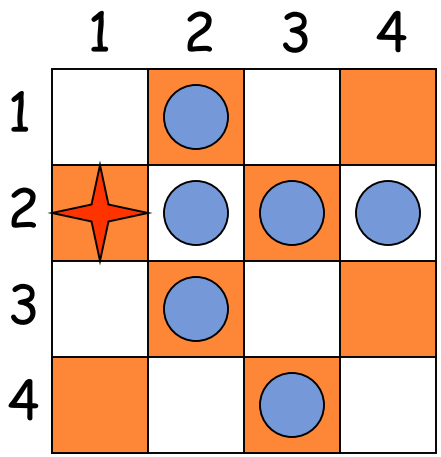
4-Queens Problem

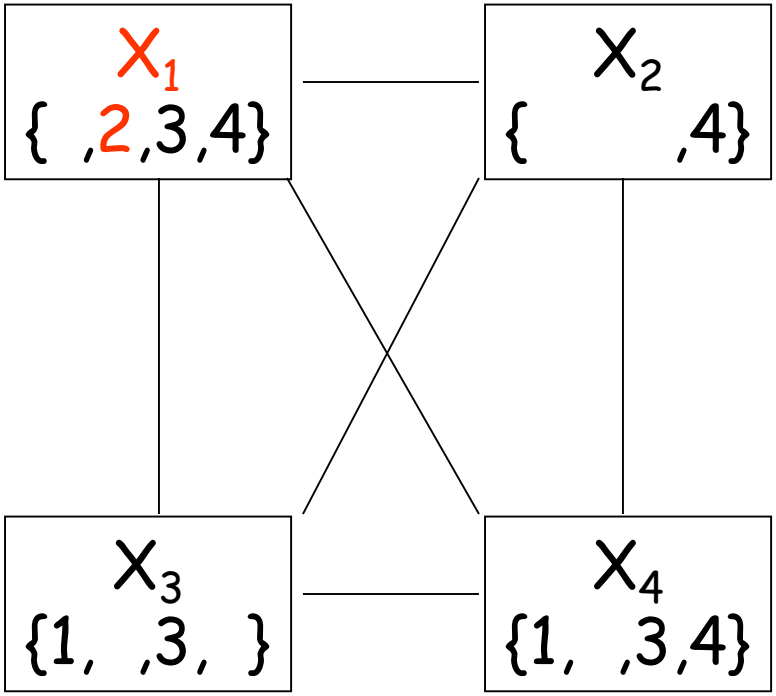




7) The algorithm performs forward checking

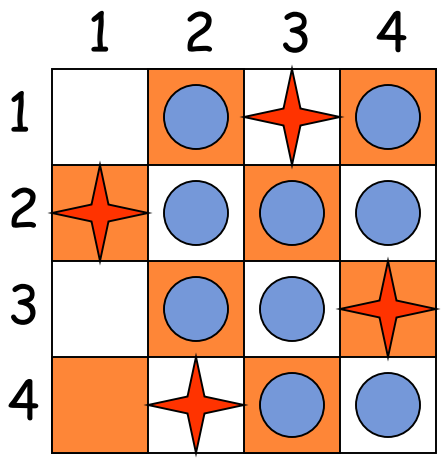
4-Queens Problem

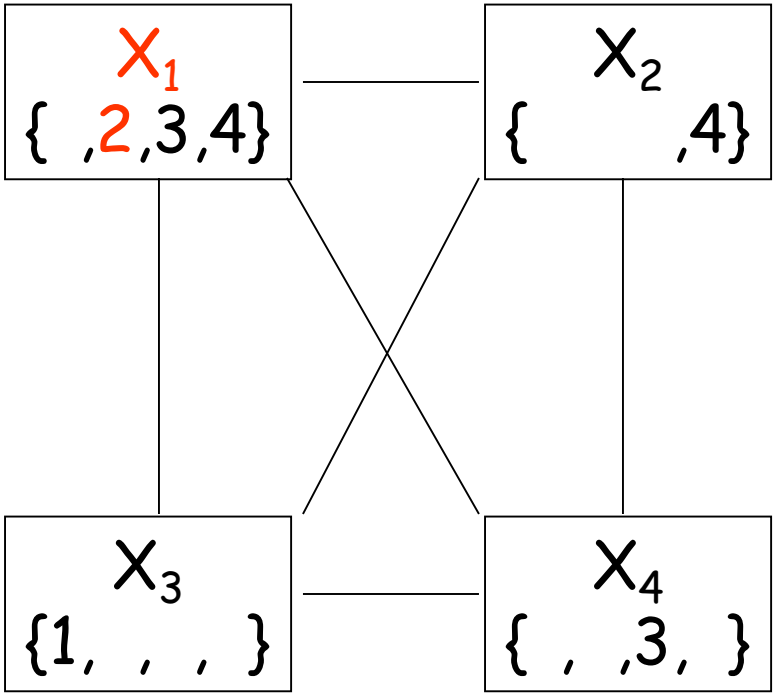




8) The algorithm calls AC3

4-Queens Problem

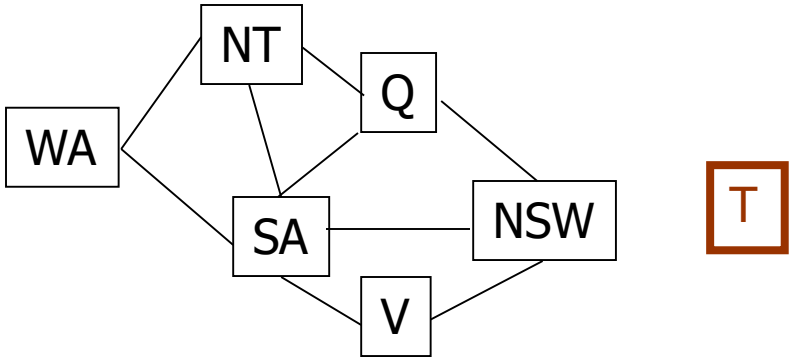




8) The algorithm calls AC3, which reduces the domains of X_3 and X_4 to a single value

Exploiting the Structure of CSP

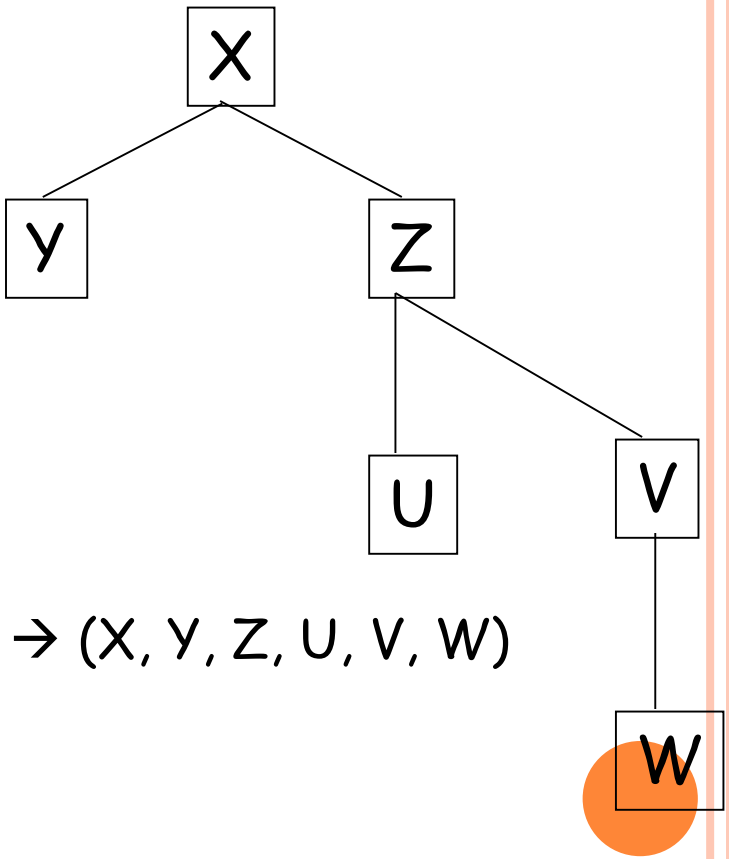
If the constraint graph contains several components, then solve one independent CSP per component



Exploiting the Structure of CSP

If the constraint graph is a tree, then :

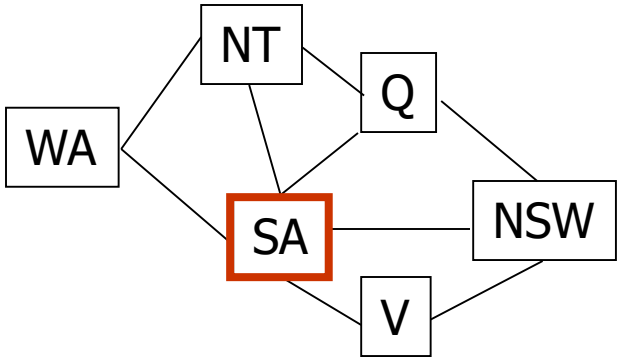
1. Order the variables from the root to the leaves
   → $(X_1, X_2, \ldots, X_n)$
2. For j = n, n-1, …, 2 call REMOVE-VALUES(X_j, X_i) where X_i is the parent of X_j
3. Assign any valid value to X_1
4. For j = 2, …, n do
   Assign any value to X_j consistent with the value assigned to its parent X_i



→ (X, Y, Z, U, V, W)

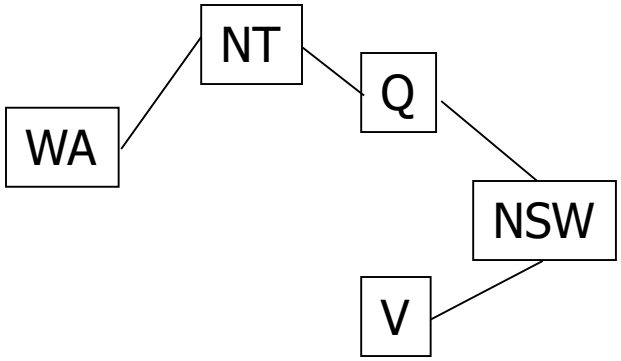
Exploiting the Structure of CSP

Whenever a variable is assigned a value by the backtracking algorithm, propagate this value and remove the variable from the constraint graph



Exploiting the Structure of CSP

Whenever a variable is assigned a value by the backtracking algorithm, propagate this value and remove the variable from the constraint graph



If the graph becomes a tree, then proceed as shown in previous slide

K-CONSISTENCY

- A graph is strongly k-consistent if
  - It is k-consistent and
  - Is also (k-1) consistent, (k-2) consistent, … all the way down to 1-consistent.

- This is ideal since a solution can be found in time $O(nd)$ instead of $O(n^2d^3)$
- YET *no free lunch*: any algorithm for establishing n-consistency must take time exponential in n, in the worst case.

Backtracking Algorithm

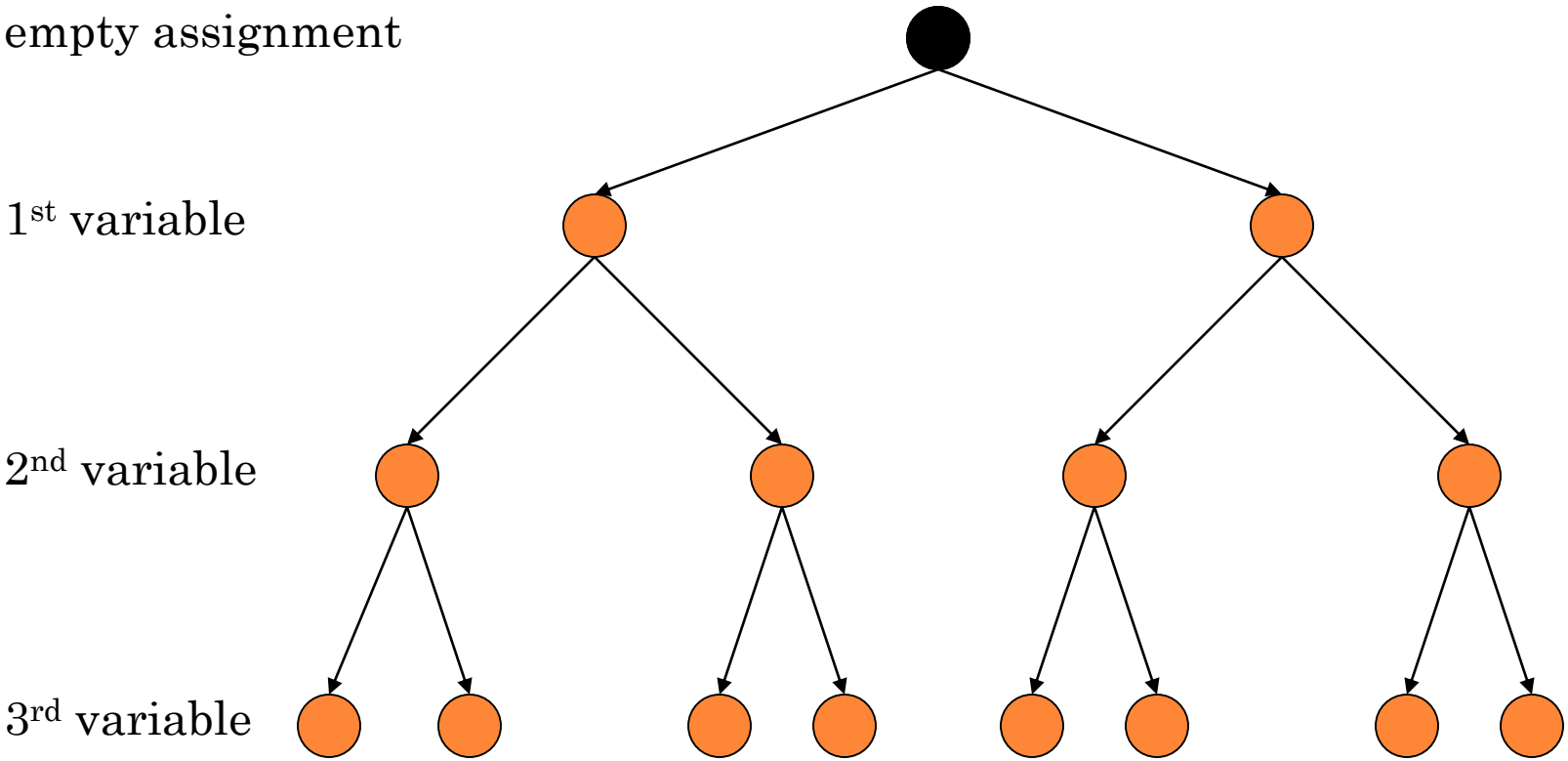
partial assignment of variables

CSP-BACKTRACKING({})

CSP-BACKTRACKING(a)

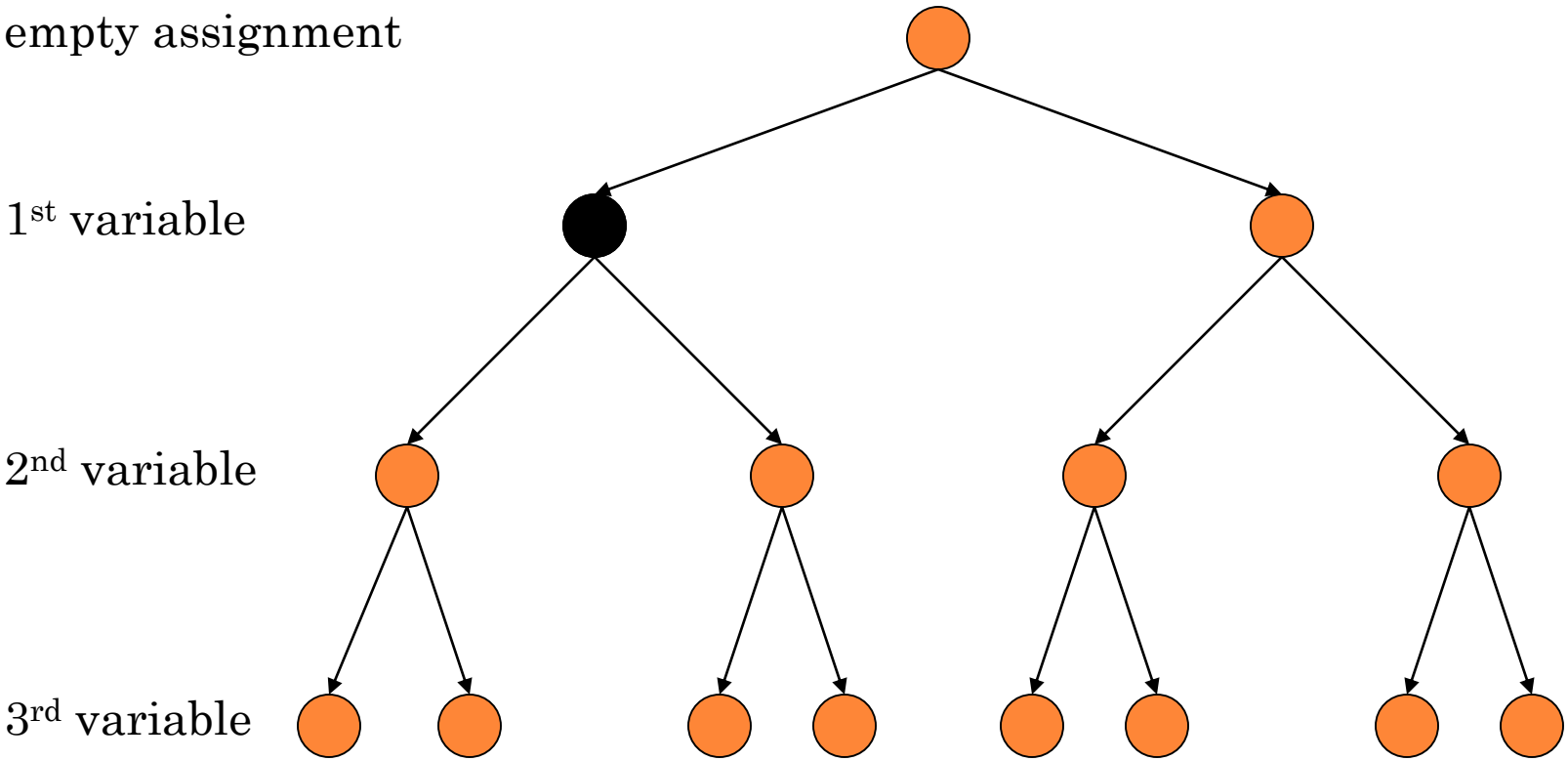
- If a is complete then return a
- X ← select unassigned variable
- D ← select an ordering for the domain of X
- For each value v in D do
  - If v is consistent with a then
    - Add (X= v) to a
    - result ← CSP-BACKTRACKING(a)
    - If result ≠ *failure* then return result
- Return *failure*

→ Backtracking Search



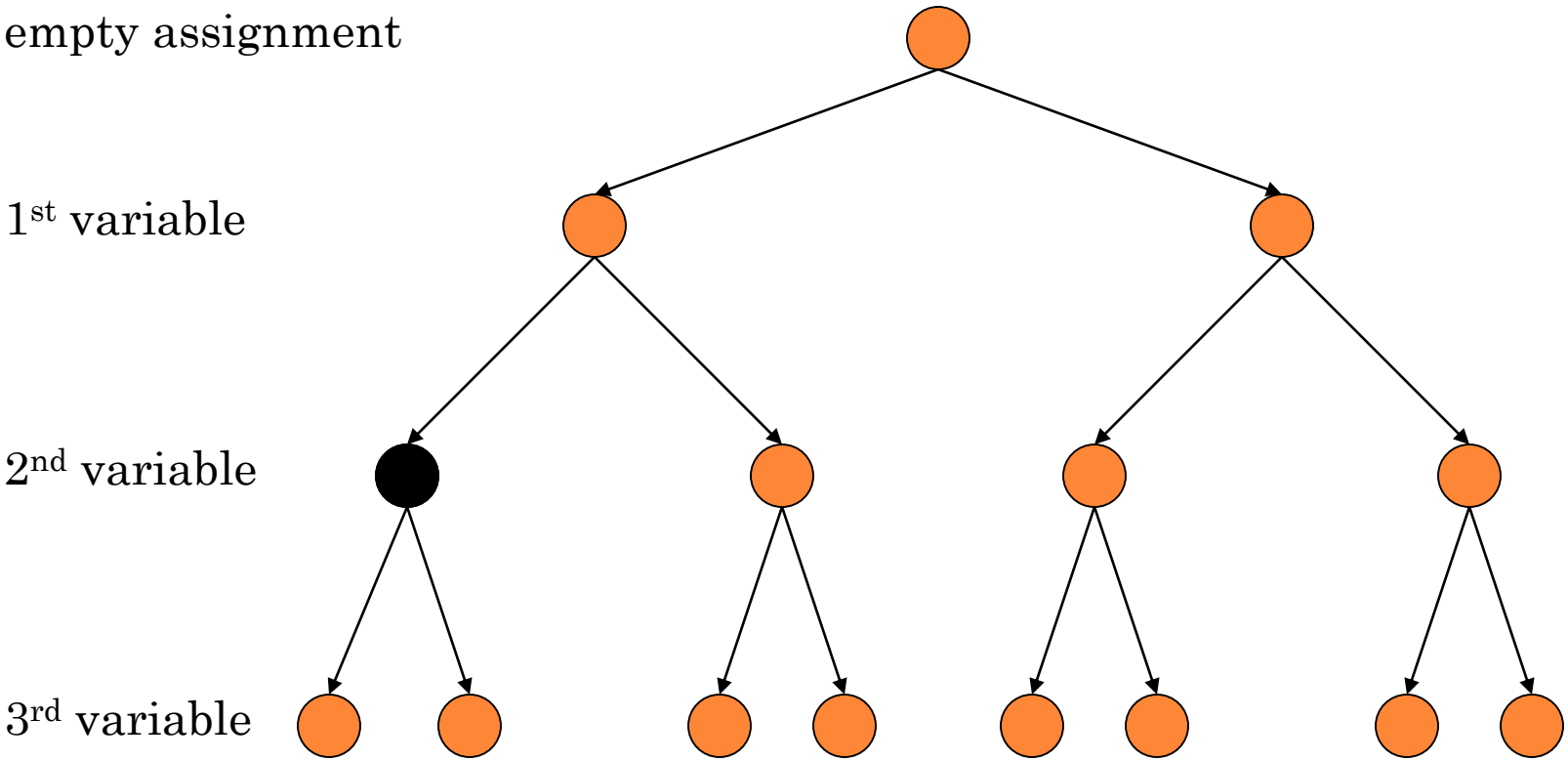
Assignment = {}

→ Backtracking Search



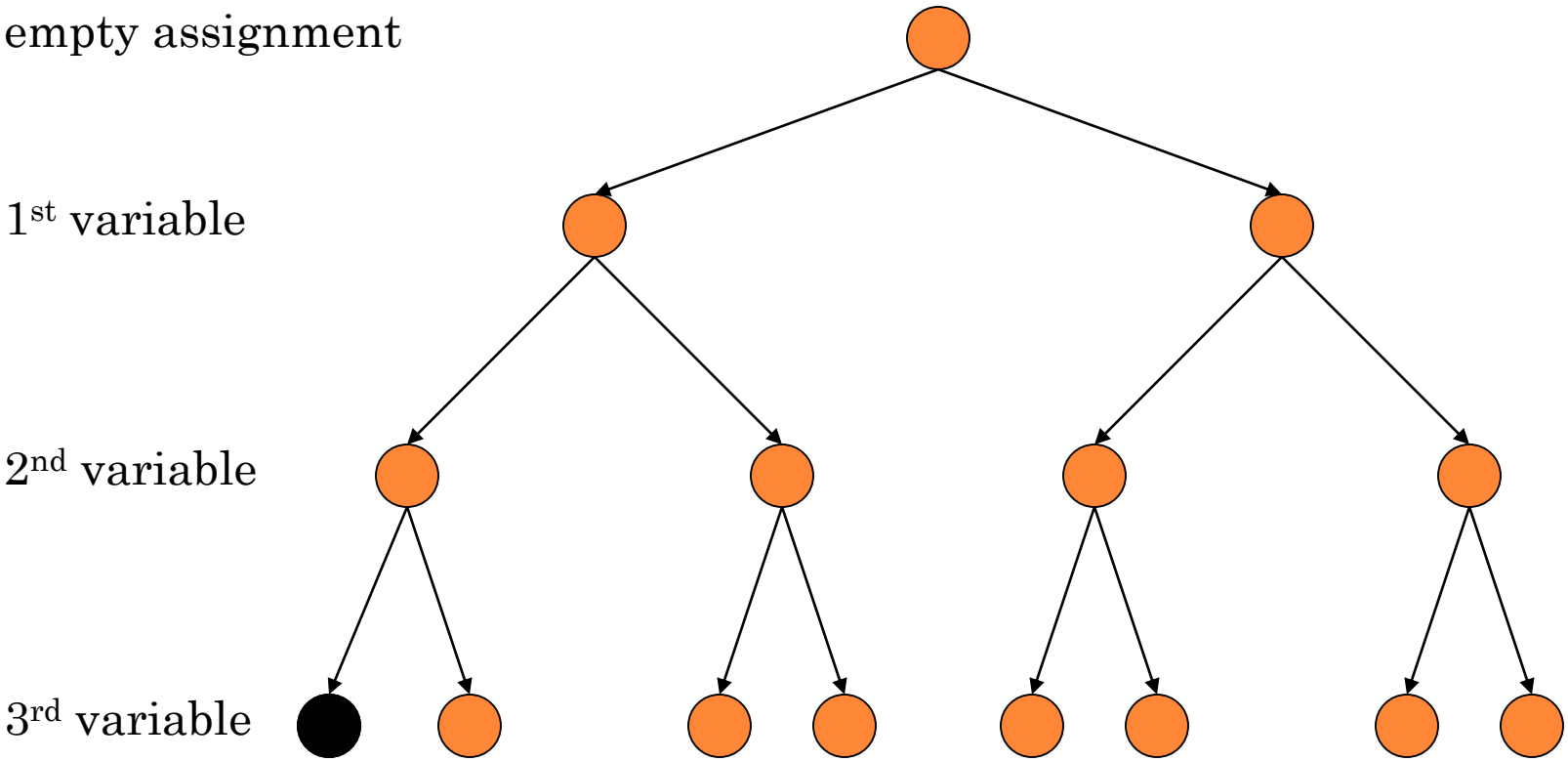
Assignment = {(var1=v11)}

→ Backtracking Search



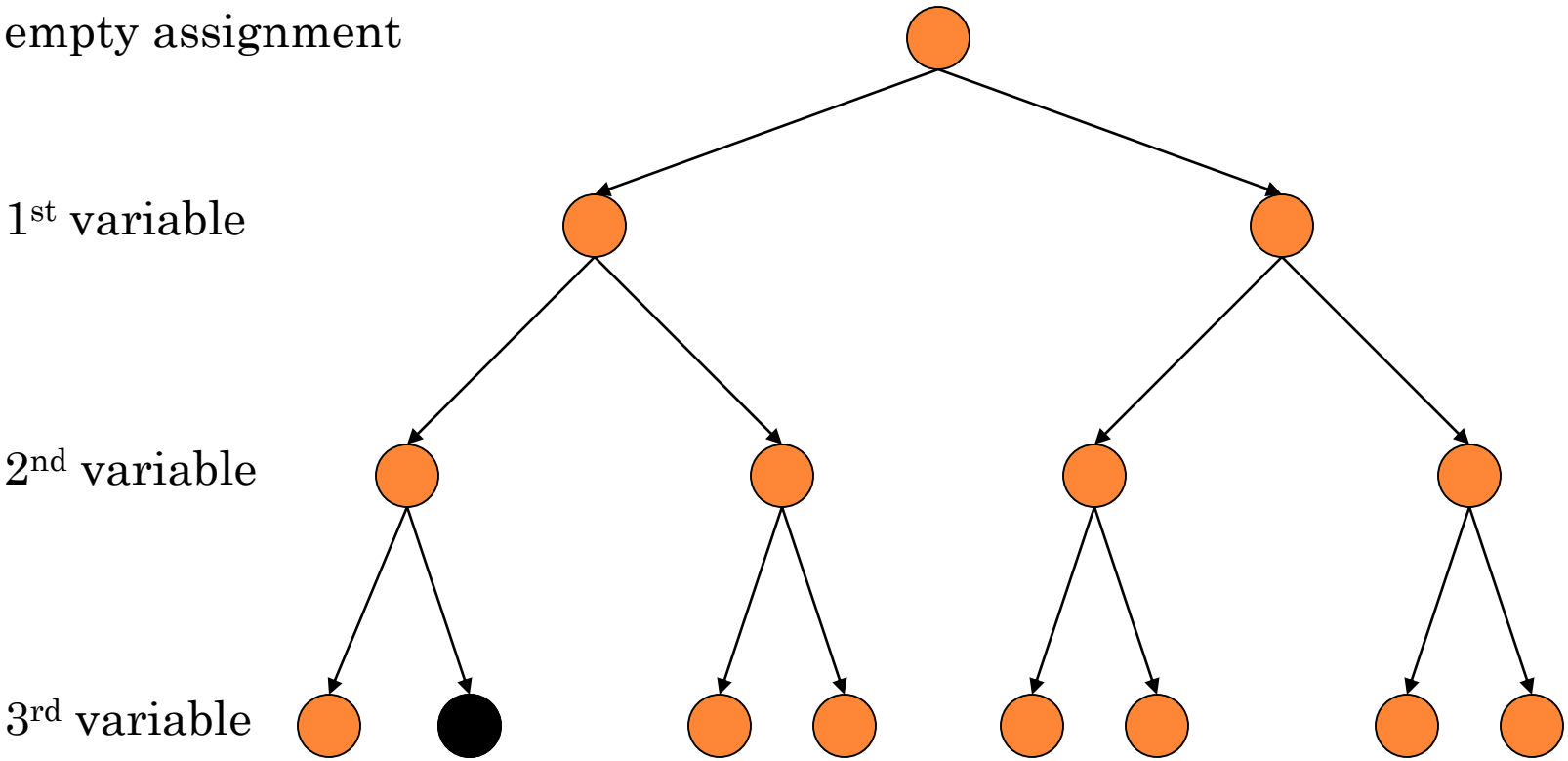
Assignment = {(var1=v11),(var2=v21)}

→ Backtracking Search



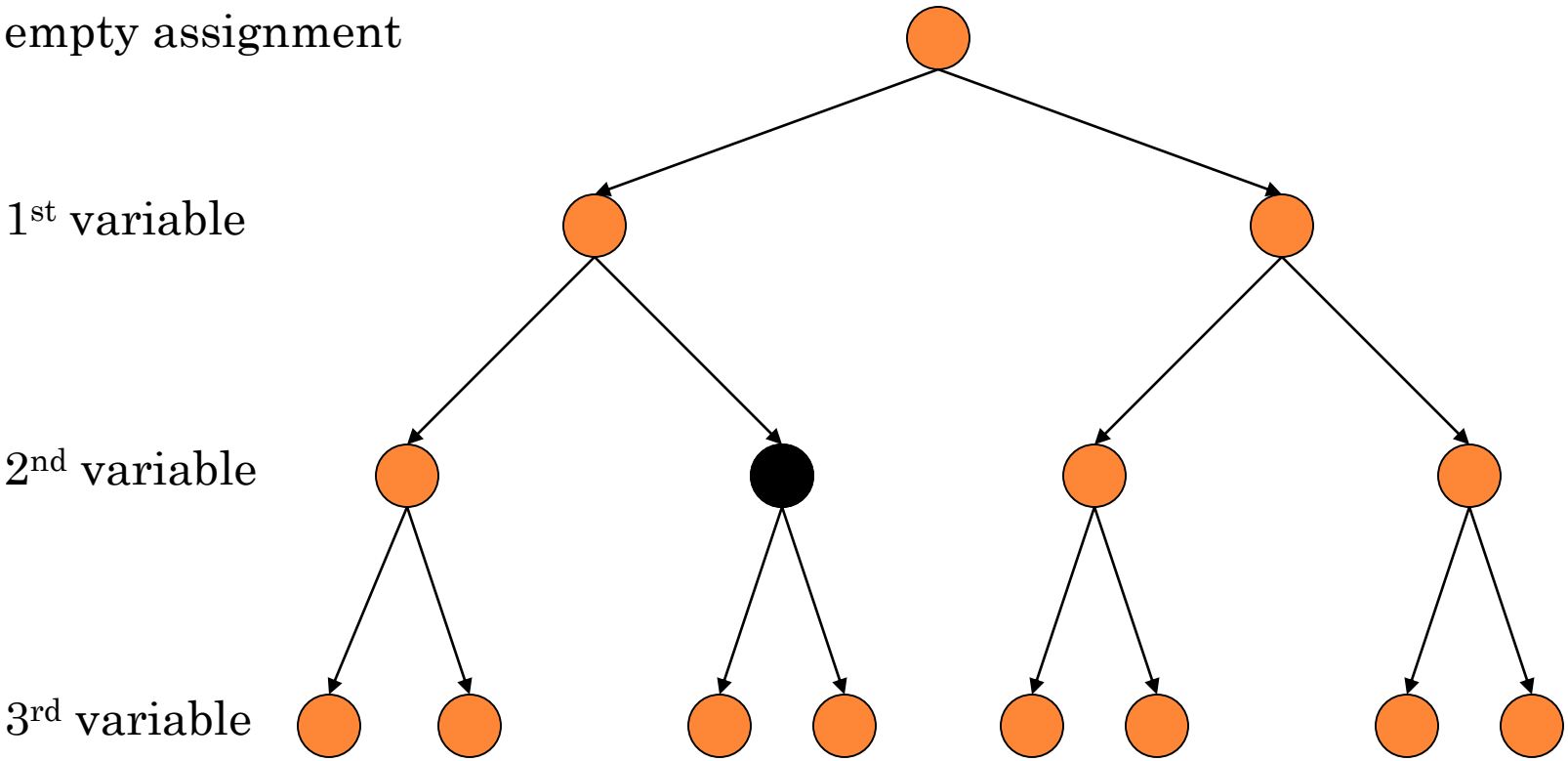
Assignment = {(var1=v11),(var2=v21),(var3=v31)}

→ Backtracking Search



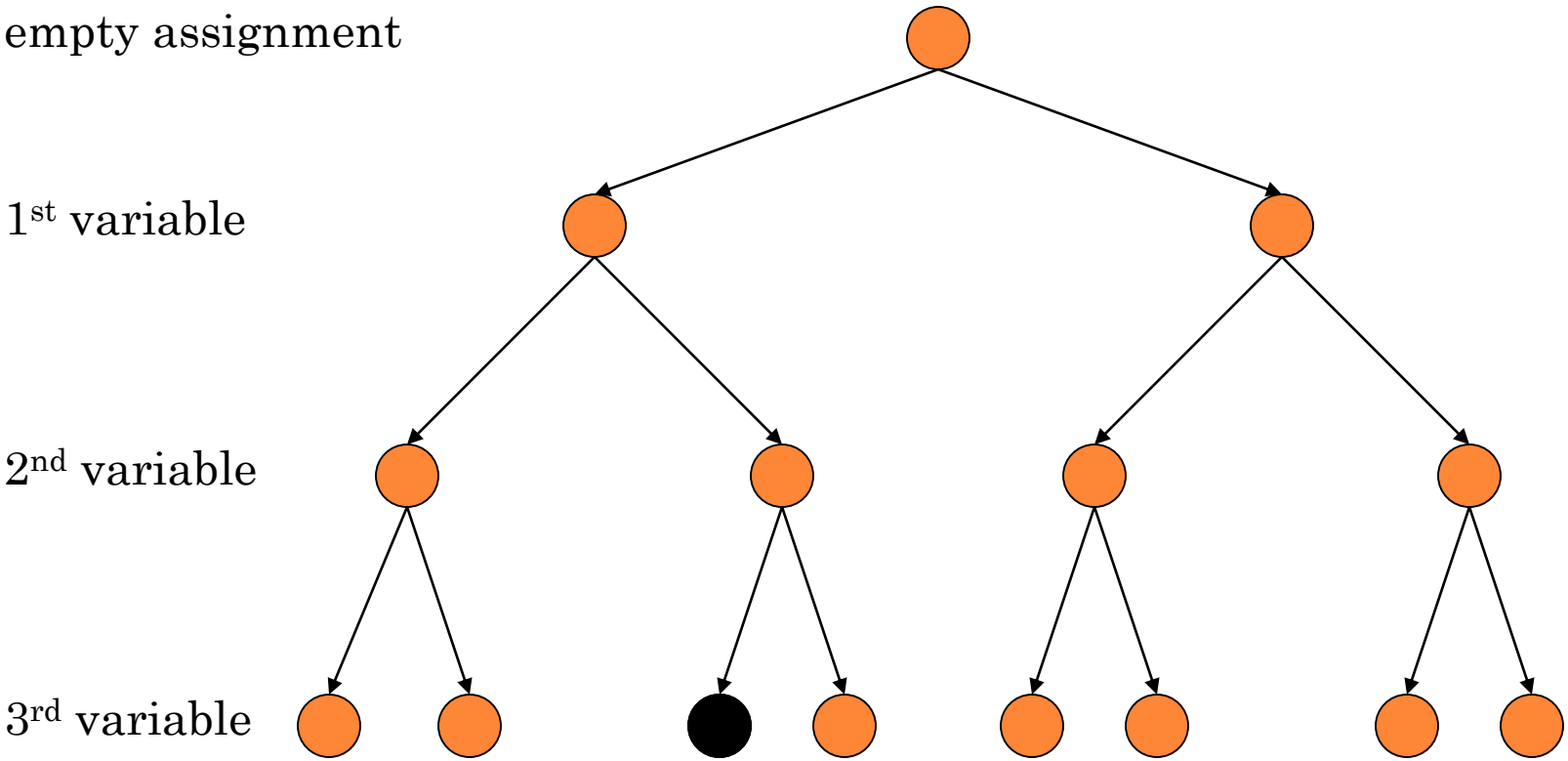
Assignment = {(var1=v11),(var2=v21),(var3=v32)}

→ Backtracking Search



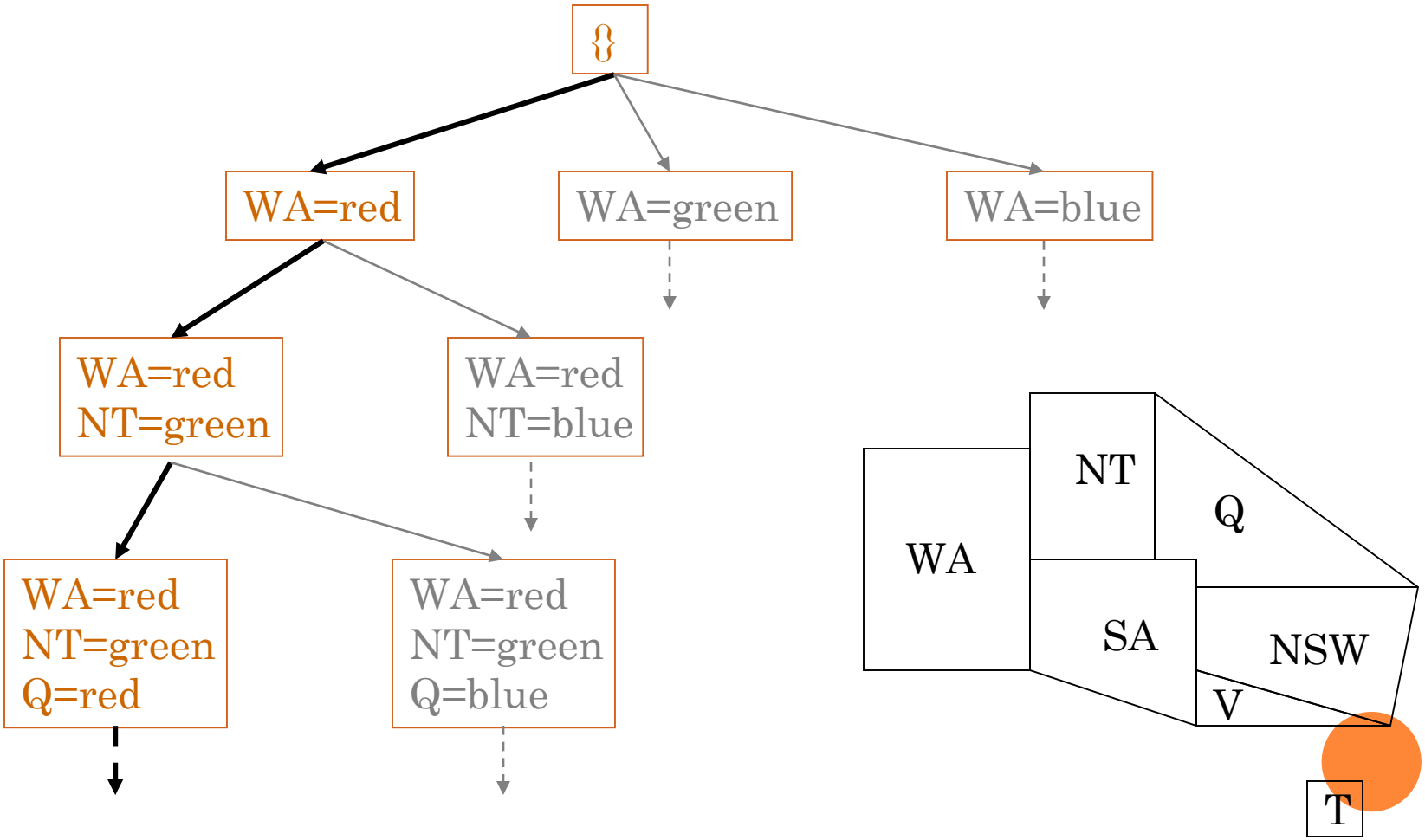
Assignment = {(var1=v11),(var2=v22)}

→ Backtracking Search



Assignment = {(var1=v11),(var2=v22),(var3=v31)}

Map Coloring



Questions

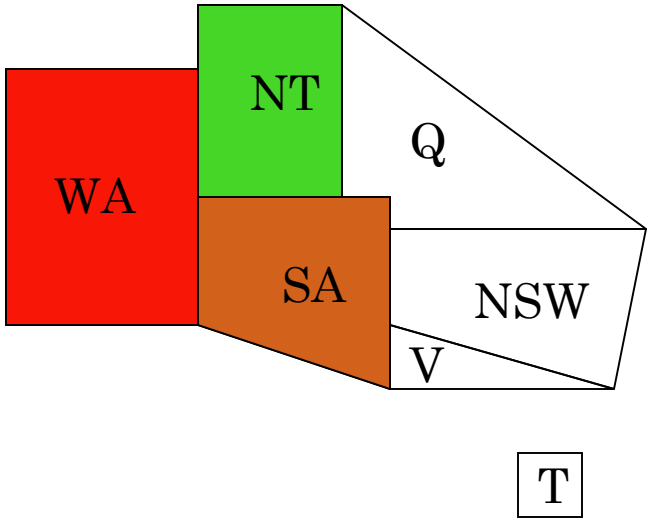
1. Which variable X should be assigned a value next?
2. In which order should its domain D be sorted?

Questions

1. Which variable X should be assigned a value next?
2. In which order should its domain D be sorted?
3. What are the implications of a partial assignment for yet unassigned variables? (→ Constraint Propagation)

Choice of Variable

- Map coloring

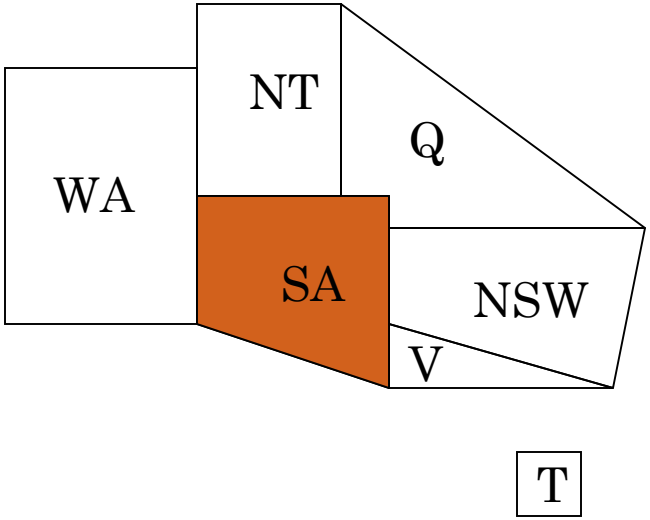


Choice of Variable

#1: Minimum Remaining Values (aka Most-constrained-variable heuristic or Fail First):

Select a variable with the fewest remaining values

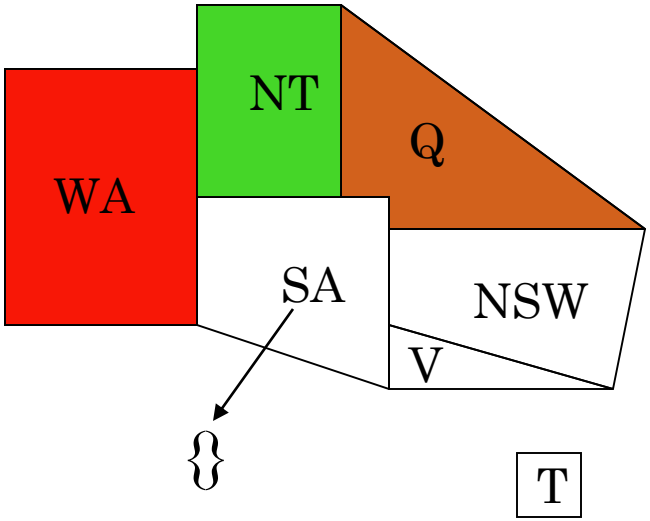
Choice of Variable



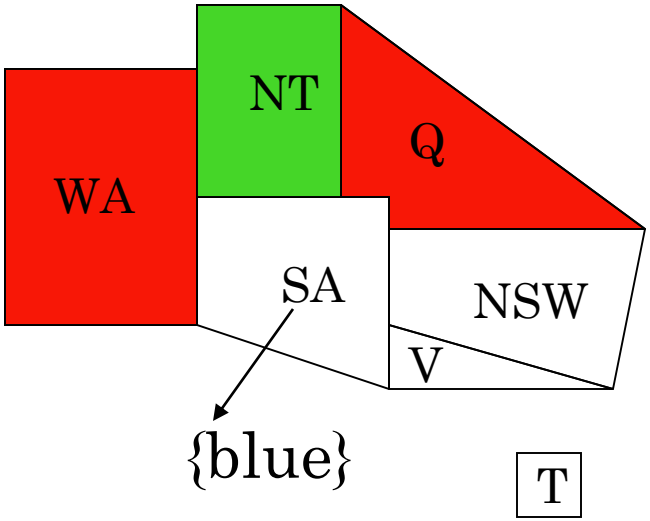
#2: Degree Heuristic:

Select the variable that is involved in the largest number of constraints on other unassigned variables

Choice of Value



Choice of Value



#3: Least-constraining-value heuristic:

Prefer the value that leaves the largest subset of legal values for other unassigned variables

Sudoku

- Variable Selection:
  - Minimum Remaining Values?
  - Degree
- Value Selection:
  - Least constraining value?
- Others?

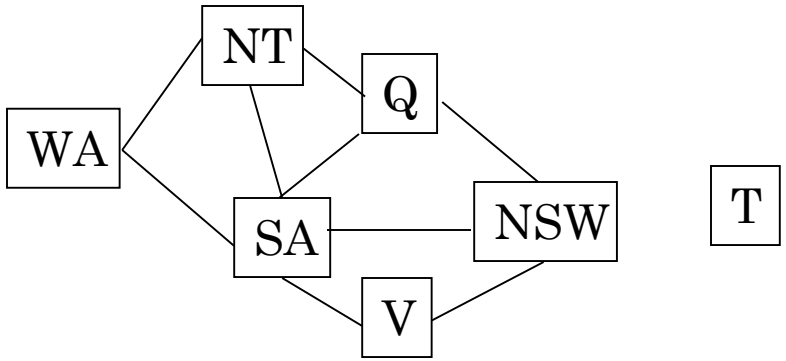
Constraint Propagation …

… is the process of determining how the possible values of one variable affect the possible values of other variables

Forward Checking

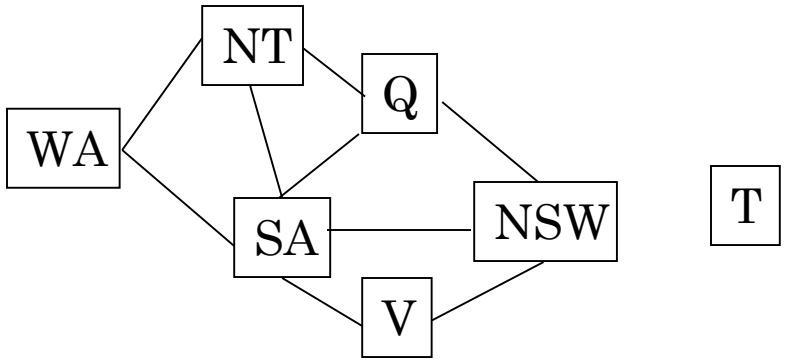
After a variable X is assigned a value v, look at each unassigned variable Y that is connected to X by a constraint and deletes from Y's domain any value that is inconsistent with v

Map Coloring: FC



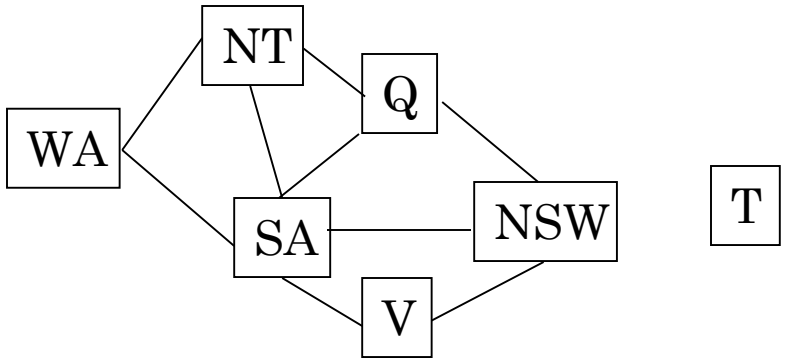
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB

Map Coloring: FC



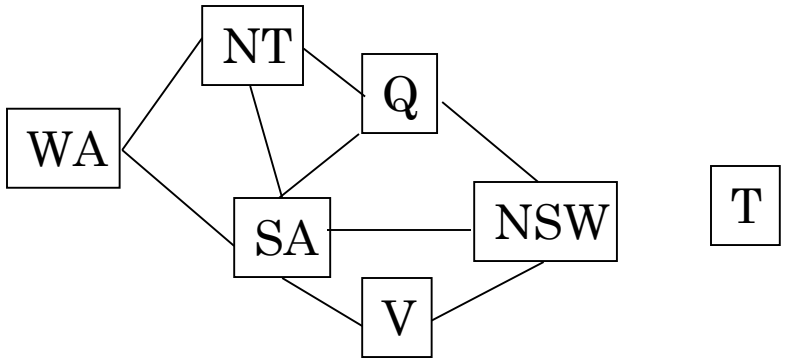
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
1: R	RGB	RGB	RGB	RGB	RGB	RGB

Map Coloring: FC



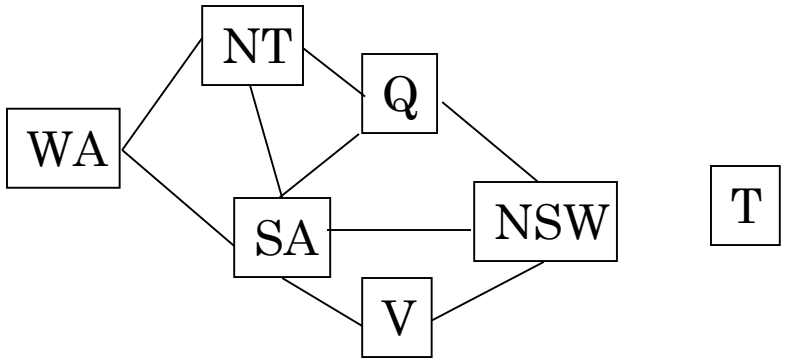
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
1: R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB

Map Coloring: FC



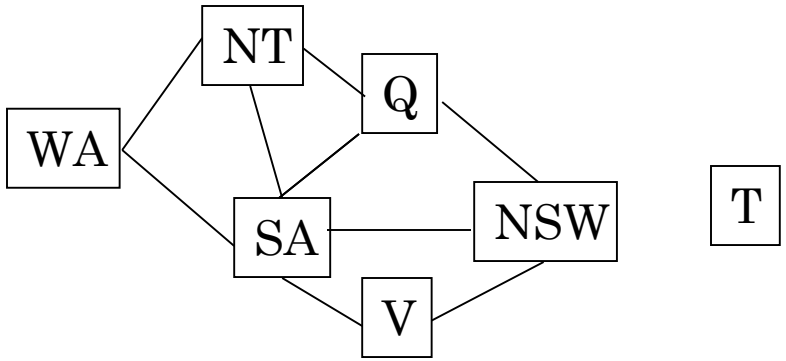
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB
R	~~R~~GB	2: G	RGB	RGB	~~R~~GB	RGB

Map Coloring:FC



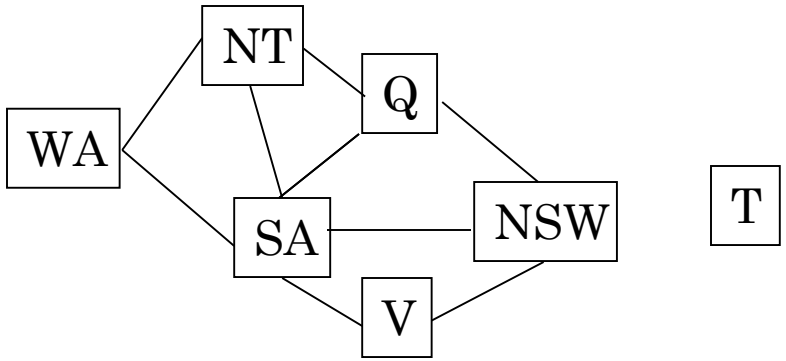
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB
R	~~RG~~B	2: G	~~R~~GB	RGB	~~RG~~B	RGB

Map Coloring: FC



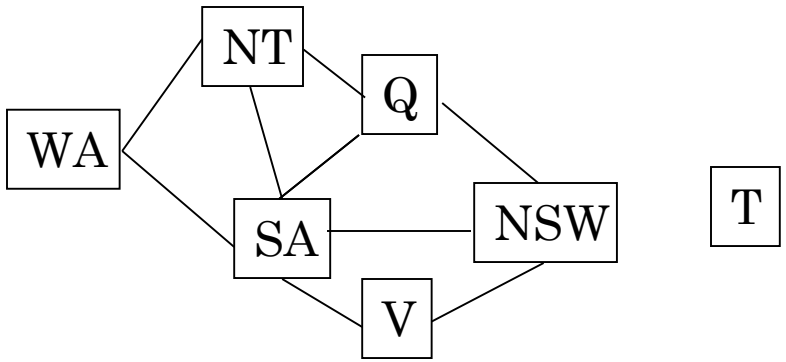
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB
R	~~RG~~B	G	~~R~~GB	RGB	~~RG~~B	RGB
R	~~RG~~B	G	~~R~~GB	3:B	~~RG~~B	RGB

Map Coloring: FC



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB
R	~~RGB~~	G	~~R~~GB	RGB	~~RGB~~	RGB
R	~~RGB~~	G	~~RGB~~	3:B	~~RGB~~	RGB

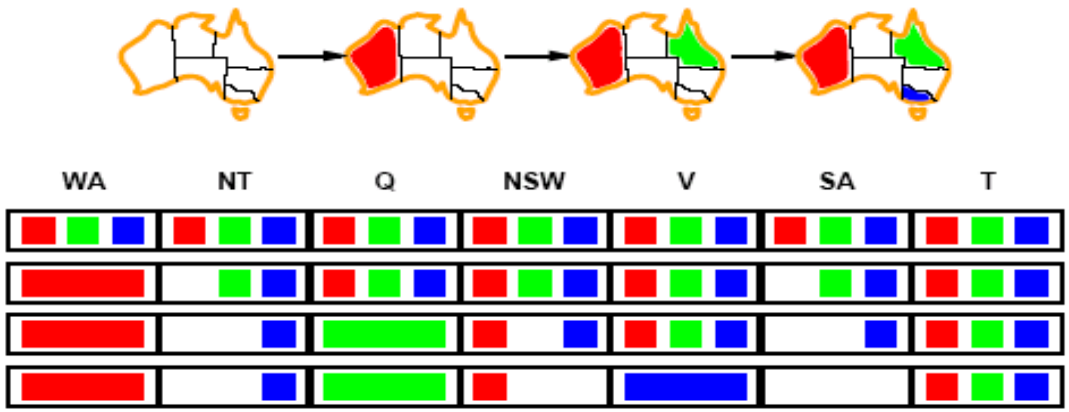
Other inconsistencies?



Impossible assignments that forward checking do not detect

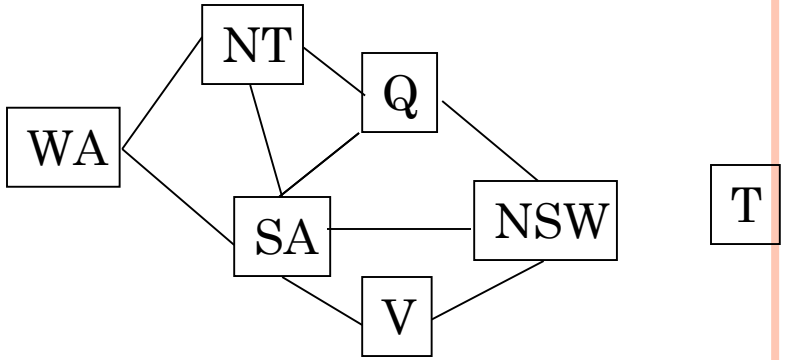
WA	NT	Q				
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB
R	~~RG~~B	G	~~R~~GB	RGB	~~RG~~B	RGB
R	~~RG~~B	G	~~RG~~B	3:B	~~RG~~B	RGB

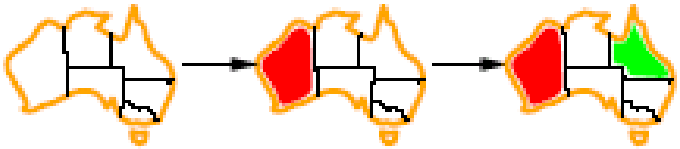
Constraint propagation

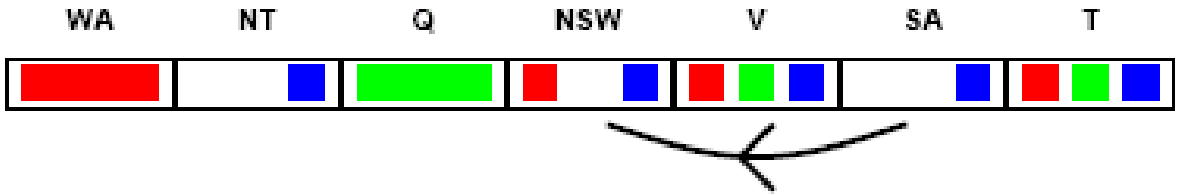


- Solving CSPs with combination of heuristics plus forward checking is more efficient than either approach alone.
- FC checking propagates information from assigned to unassigned variables but does not provide detection for all failures.
  - More advanced constraint propagation methods repeatedly enforces constraints locally

Arc consistency





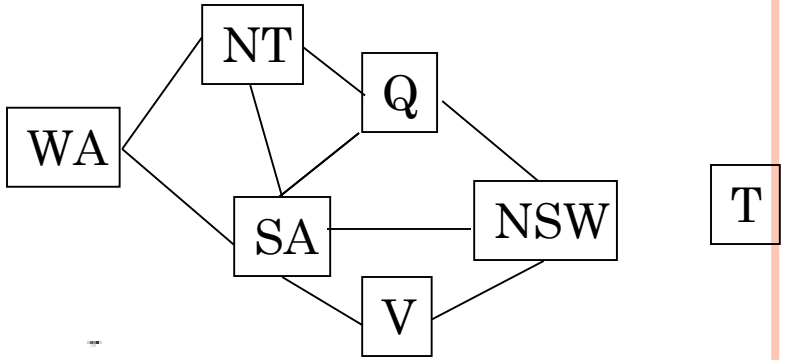


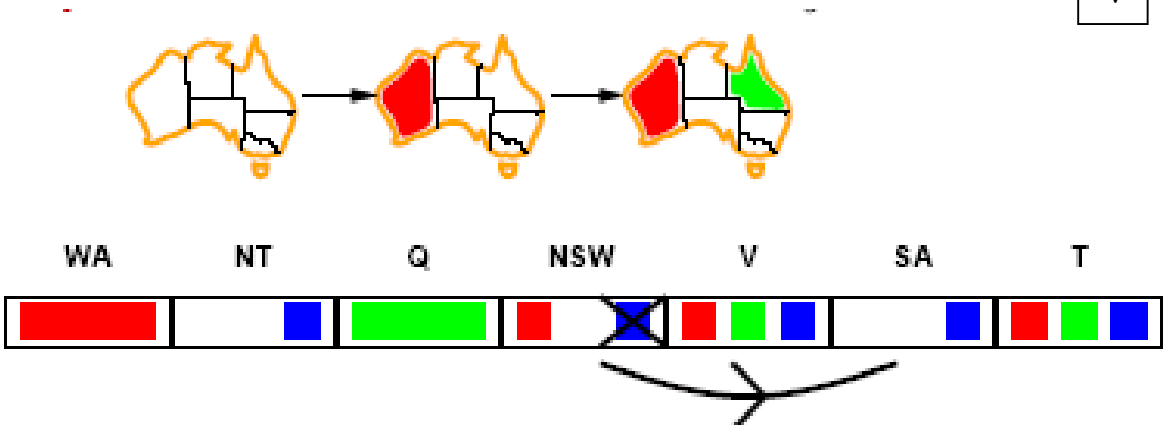
- $X \rightarrow Y$ is consistent iff

  for *every* value x of X there is some allowed y
- $SA \rightarrow NSW$ is consistent iff

  SA=*blue* and NSW=*red*

Arc consistency





- $X \rightarrow Y$ is consistent iff

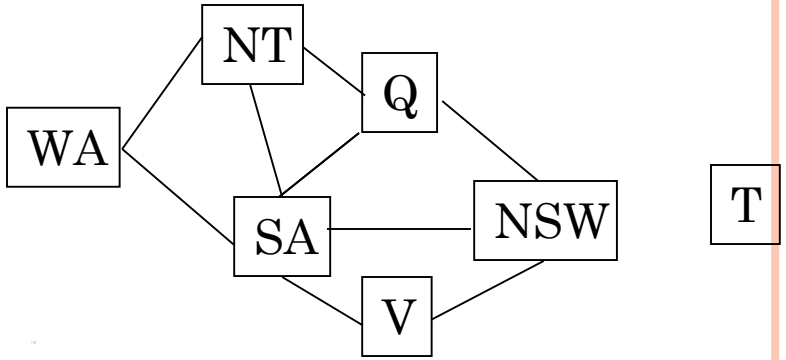
  for *every* value x of X there is some allowed y
- $NSW \rightarrow SA$ is consistent iff

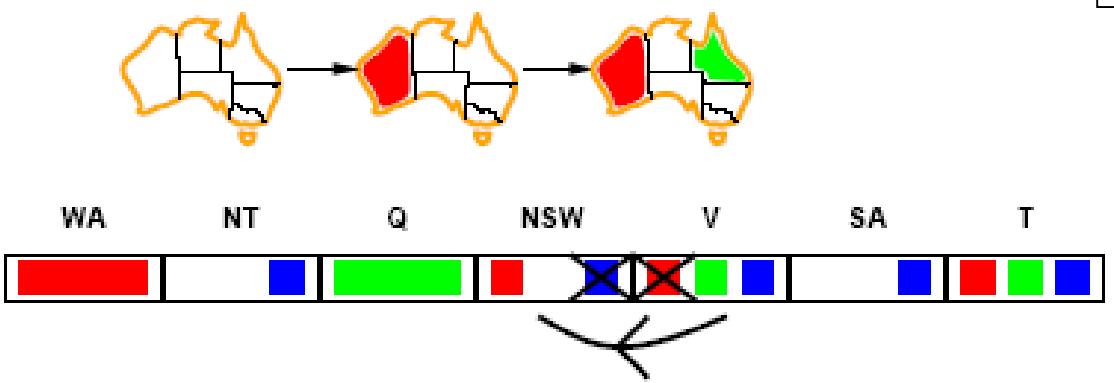
  NSW=red and *SA=blue*

  NSW=blue and SA=???

Arc can be made consistent by removing *blue* from *NSW*

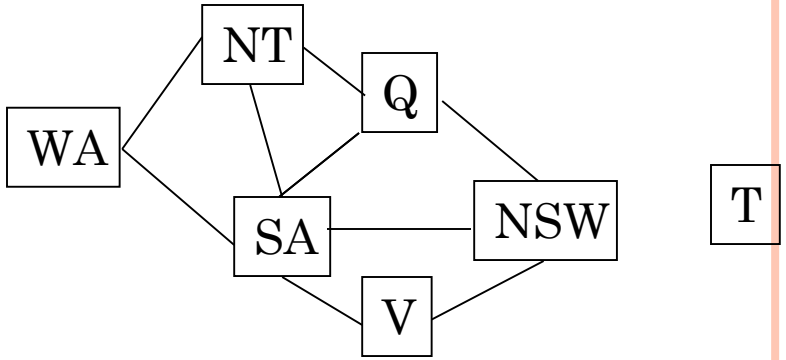
Arc consistency

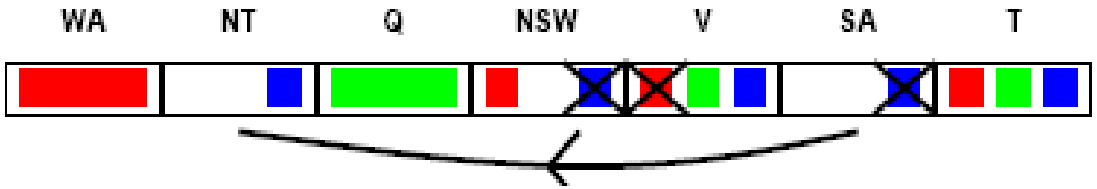




- Arc can be made consistent by removing *blue* from *NSW*
- RECHECK neighbors of *NSW*!!
  - Remove red from *V*

Arc consistency





- Arc can be made consistent by removing *blue* from *NSW*
- RECHECK neighbors of *NSW*!!
  - Remove red from *V*
- Arc consistency detects failure earlier than FC
- Can be run as a preprocessor or after each assignment.
  - Repeated until no inconsistency remains

Arc consistency algorithm

```
function AC-3(csp) return the CSP, possibly with reduced domains
    inputs: csp, a binary csp with variables {X1, X2, …, Xn}
    local variables: queue, a queue of arcs initially the arcs in csp

    while queue is not empty do
            (Xi, Xj) ← REMOVE-FIRST(queue)
            if REMOVE-INCONSISTENT-VALUES(Xi, Xj)  then
                    for each Xk in NEIGHBORS[Xi ] - Xj do
                    add (Xk, Xi) to queue

function REMOVE-INCONSISTENT-VALUES(Xi, Xj) return true iff we remove a value
    removed ←  false
    for each x in DOMAIN[Xi] do
            if no value y in DOMAIN[Xi] does (x, y) satisfy constraints between Xi and Xj
            then delete x from DOMAIN[Xi];
            removed ←  true
    return removed
```

K-CONSISTENCY

- Arc consistency does not detect all inconsistencies:
  - Partial assignment *{WA=red, NSW=red}* is inconsistent.
- Stronger forms of propagation can be defined using the notion of k-consistency.
- A CSP is k-consistent if for any set of k-1 variables and for any consistent assignment to those variables, a consistent value can always be assigned to any kth variable.
  - E.g. 1-consistency or node-consistency
  - E.g. 2-consistency or arc-consistency
  - E.g. 3-consistency or path-consistency

K-CONSISTENCY

- A graph is strongly k-consistent if
  - It is k-consistent and
  - Is also (k-1) consistent, (k-2) consistent, … all the way down to 1-consistent.

- This is ideal since a solution can be found in time $O(nd)$ instead of $O(n^2d^3)$
- YET *no free lunch*: any algorithm for establishing n-consistency must take time exponential in n, in the worst case.

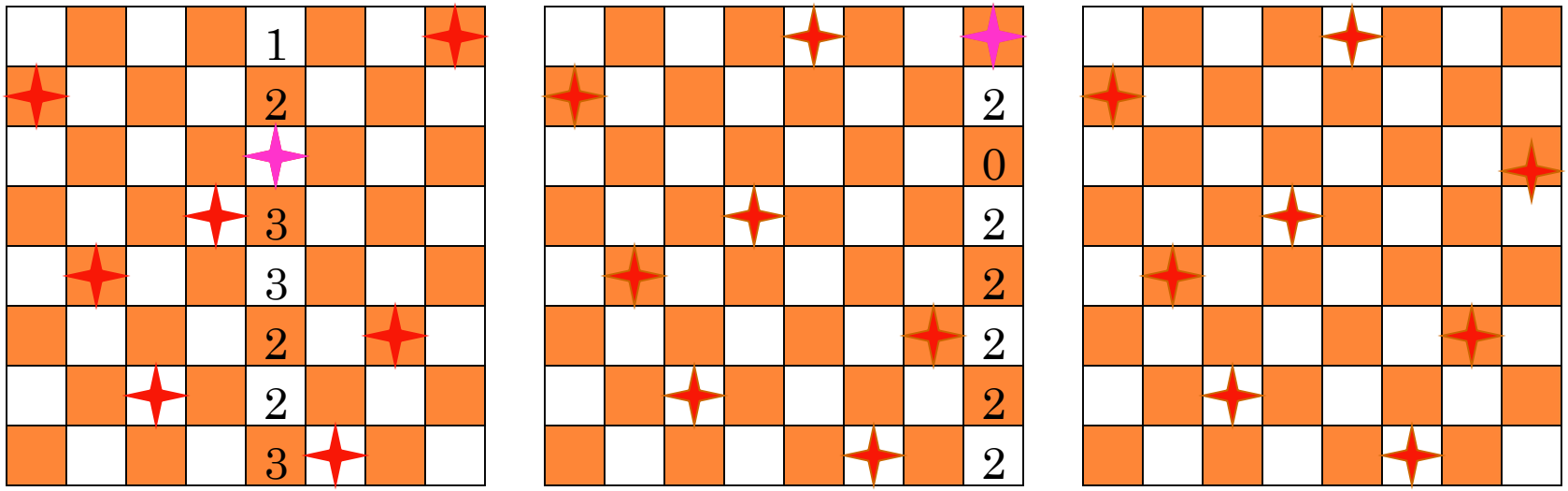
Further improvements

- Checking special constraints
  - Checking Alldif(…) constraint
    - *E.g. {WA=red, NSW=red}*
  - Checking Atmost(…) constraint
    - *Bounds propagation for larger value domains*
- Intelligent backtracking
  - Standard form is chronological backtracking i.e. try different value for preceding variable.
  - More intelligent, backtrack to conflict set.
    - Set of variables that caused the failure or set of previously assigned variables that are connected to X by constraints.
    - Backjumping moves back to most recent element of the conflict set.
    - Forward checking can be used to determine conflict set.

Local search for CSP

- Use complete-state representation
- For CSPs
  - allow states with unsatisfied constraints
  - operators **reassign** variable values
- Variable selection: randomly select any conflicted variable
- Value selection: *min-conflicts heuristic*
  - Select new value that results in a minimum number of conflicts with the other variables

Local Search for CSP



Pick initial complete assignment (at random)
Repeat

- Pick a conflicted variable var (at random)
- Set the new value of var to minimize the number of conflicts
- If the new assignment is not conflicting then return it

(min-conflicts heuristics)

Local search for CSP

function MIN-CONFLICTS(*csp, max_steps*) **return** solution or failure

inputs: *csp*, a constraint satisfaction problem

max_steps, the number of steps allowed before giving up

current ← an initial complete assignment for *csp*

for *i* = 1 to *max_steps* **do**

if *current* is a solution for *csp* then return *current*

var ← a randomly chosen, conflicted variable from VARIABLES[*csp*]

value ← the value *v* for *var* that minimize CONFLICTS(*var,v,current,csp*)

set *var* = *value* in *current*

return *failure*

Remark

- Local search with min-conflict heuristic works extremely well for million-queen problems
- The reason: Solutions are densely distributed in the $O(n^n)$ space, which means that on the average a solution is a few steps away from a randomly picked assignment

Backtracking search

- Variable assignments are commutative}, i.e.,

[WA = red then NT = green] same as [NT = green then WA = red]

- Only need to consider assignments to a single variable at each node
  → b = d and there are d^n leaves

- Depth-first search for CSPs with single-variable assignments is called backtracking search
-

- Backtracking search is the basic uninformed algorithm for CSPs
-

- Can solve n-queens for $n \approx 25$
-

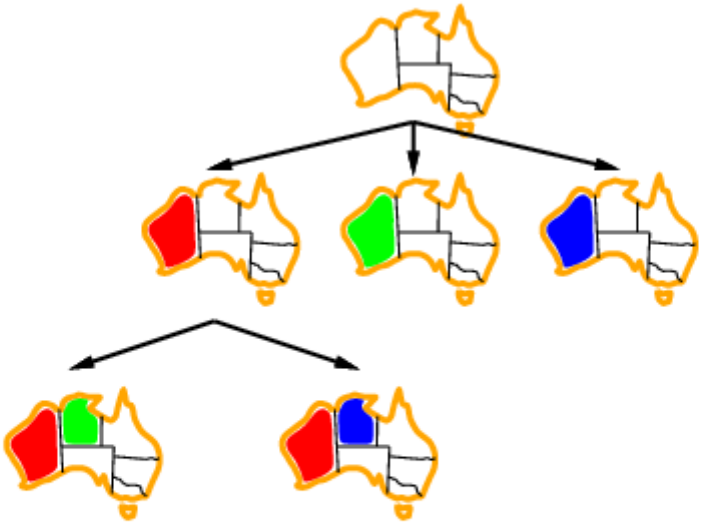
Backtracking search

```
function BACKTRACKING-SEARCH( csp) returns a solution, or failure
   return RECURSIVE-BACKTRACKING({}, csp)

function RECURSIVE-BACKTRACKING( assignment,csp) returns a solution, or
failure
   if assignment is complete then return assignment
   var ← SELECT-UNASSIGNED-VARIABLE( Variables[csp], assignment, csp)
   for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment according to Constraints[csp] then
         add { var = value } to assignment
         result ← RECURSIVE-BACKTRACKING(assignment, csp)
         if result ≠ failue then return result
         remove { var = value } from assignment
   return failure
```

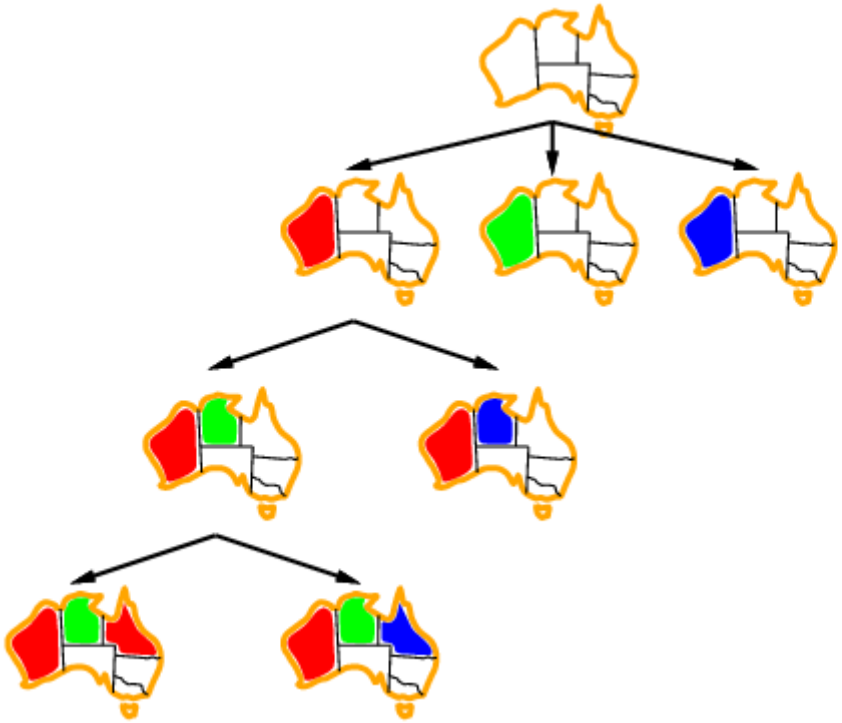
Backtracking example

Backtracking example

Backtracking example



Backtracking example



Improving backtracking efficiency

- General-purpose methods can give huge gains in speed:
-
  - Which variable should be assigned next?
  -
  - In what order should its values be tried?
  -
  - Can we detect inevitable failure early?
  -

בעיית CS - סיכום ביניים

- מקרה פרטי של בעיית חיפוש.
- התחומים יכולים להיות בדידים או רציפים.
- אין חשיבות לסדר האילוצים.
- מבחן המטרה בא לידי ביטוי ע"י האילוצים.
- אם קבוצת משתנים בלתי תלויה (אין אילוצים ביניהם) ואם לתתי קבוצות של משתנים יש קבוצות אילוצים נפרדות ניתן לפרק את הבעיה לתתי בעיות בהתאם, ולפתור כל אחת בנפרד.

המשך - CS תכונות של בעיית

- במהלך החיפוש, אם אילוץ סופק או אם נוצרת סתירה בתנאי החיפוש ממשיך בכל מקרה, ויגלה את הבעיה רק בהמשך.
- סדר בחירת הערכים למשתנים משפיע על זמן החיפוש.
- מצריך שיטה לבדיקת סיפוק האילוצים .

Improving backtracking efficiency

- General-purpose methods can give huge gains in speed:
-
  - Which variable should be assigned next?
  -
  - In what order should its values be tried?
  -
  - Can we detect inevitable failure early?
  -

היוריסטיקה 1 עבור בעיית CS

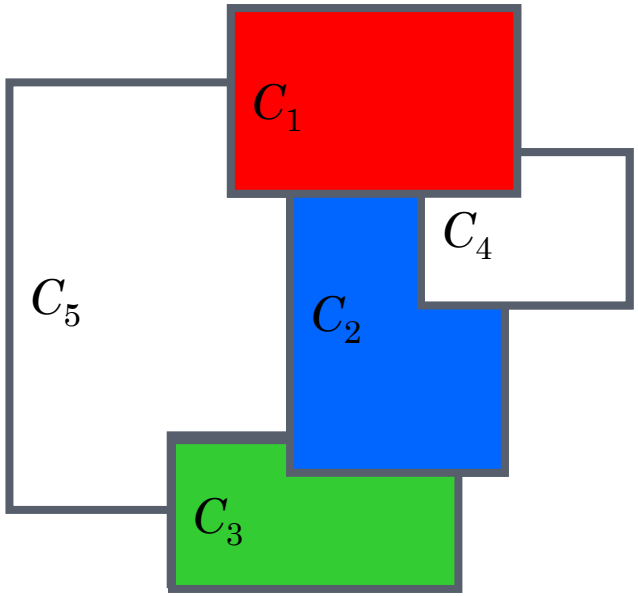
- בדיקה קדימה: כאשר מתבצעת השמת ערך למשתנה, בטל את הערכים שלא ייתכנו עבור משתנים אחרים. כאשר תחום מתרוקן - אין פתרון עם השמה זו.

לרוב יותר יעיל מ- backtracking פשוט.

היוריסטיקה עבור בעיית - CS דוגמא

- צביעת מפה - בדיקה קדימה

	אדום	כחול	ירוק
C_1	√		
C_2	×	√	
C_3		×	√
C_4	×	×	
C_5	×	×	×



- **ניתן לפתור-n מלכות עבור $n \approx 30$**

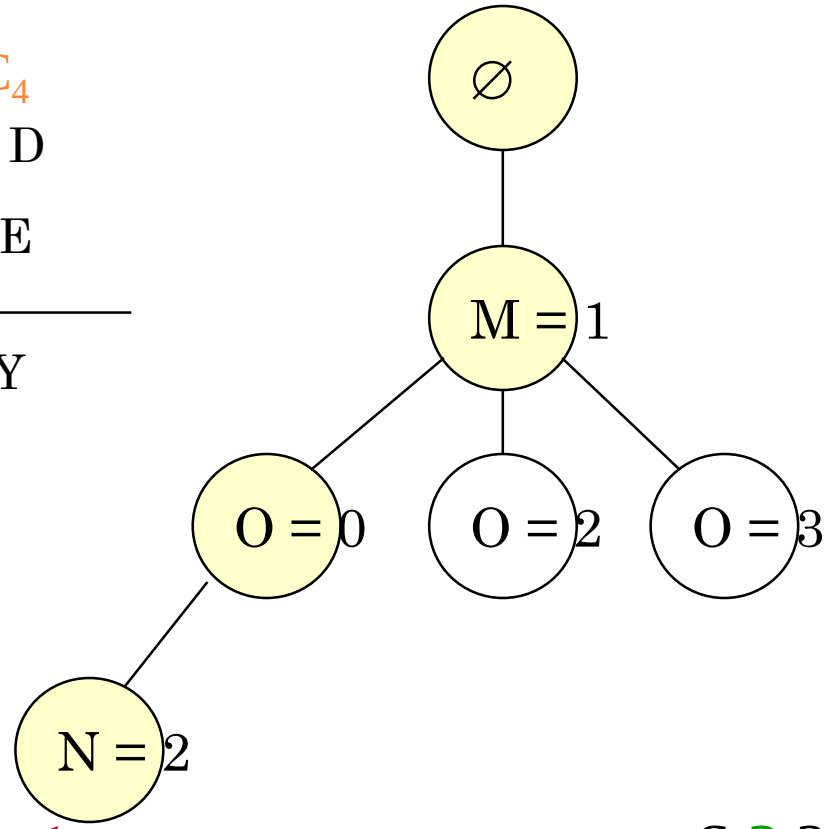
היוריסטיקה 2 עבור בעיית CS

- השמה עקבית: לכל משתנה יש ערך בתחום שלו שאם נציב אותו המצב לא יגרום לסתירה באילוצים. (אם אין כזה לא ממשיכים)

היוריסטיקה עבור בעיית - CS דוגמא

C_1 C_2 C_3 C_4

```
   S E N D
+  M O R E
---------
M O N E Y
```



לא קיימת
השמה עיקבית

- השמה עיקבית

$C_1 = 1$
$C_2 + S + 1 = O$

$C_2 + S + 1 = 0$
$C_3 + E + 0 = N$
$C4 + N + R = E$

```
   S ? 2 D
+  1 0 R ?
---------
1 0 2 ? Y
```

היוריסטיקה 3 עבור בעיית CS

- מי המשתנה הבא להשמה?

המשתנה המאולץ: בחר במשתנה שהתחום שלו קטן ושמופיע בהכי הרבה תנאים.

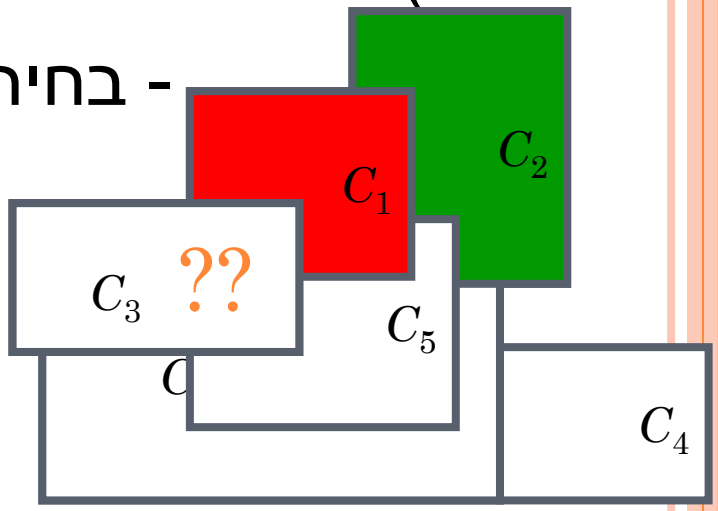
- איזו השמה לבחור?

הערך המאלץ המינימלי: בחר עבור המשתנה את הערך שיגרום לכמה שפחות אילוצים.

היוריסטיקה עבור בעיית - CS דוגמא

- צביעת מפה: - בחירת המשתנה הבא להשמה (מאולץ ביותר)

  - בחירת ערך למשתנה (מאלץ מינימלי)



- בהינתן אדום=C_1, ירוק =C_2. מי המשתנה הבא ??

C_5

- בהינתן אדום=C_1, ירוק = C_2. בחר C_3 = ??

ירוק = C_3

ניתן לפתור-n מלכות עבור n≈1000

אלגוריתמים איטרטיבים לבעיות CS

- כדי ליישם את האלגוריתמים Hill climbing ו Simulated annealing -עבור בעיות CS נבצע את השינויים:
  בכל מצב יש השמה <u>לכל</u> המשתנים (גם במצב ההתחלתי).
- מרשים מצבים עם תנאים לא מסופקים (כלומר, השמות הגורמות לסתירות בתנאים).
- הפעולות: שינוי ערך למשתנה.
- בחר באופן אקראי איזשהו משתנה הגורם לסתירה.

CS היוריסטיקה עבור בעיית

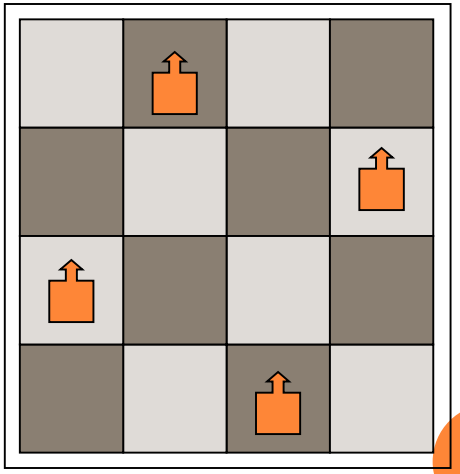
- מינימום סתירות באילוצים: בחר בערך שיגרום להכי מעט סתירות באילוצים.

  Hill climbing עם $h(n)$ =מספר הסתירות

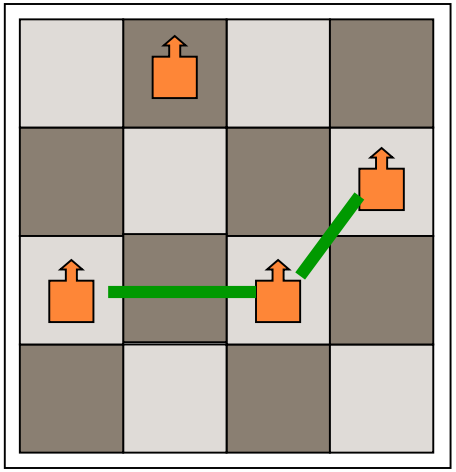
- ניתן לפתור-n מלכות עבור $n \approx 10000000$ בזמן כמעט קבוע עם הסתברות גבוהה.

היוריסטיקה - מינימום סתירות - דוגמא

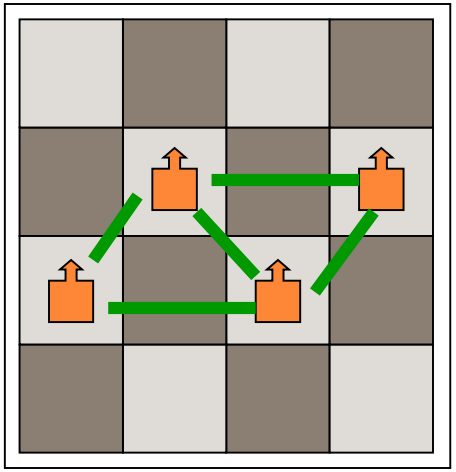
- בעיית 4 המלכות
- פונקצית הערכה: $h(n) =$ מספר המלכות המאוימות
  - מצבים: 4 מלכות ב-4 עמודות
  - פעולות: הזזת מלכה בעמודה
  - מבחן מטרה: אין מלכות מאוימות



מצב מטרה

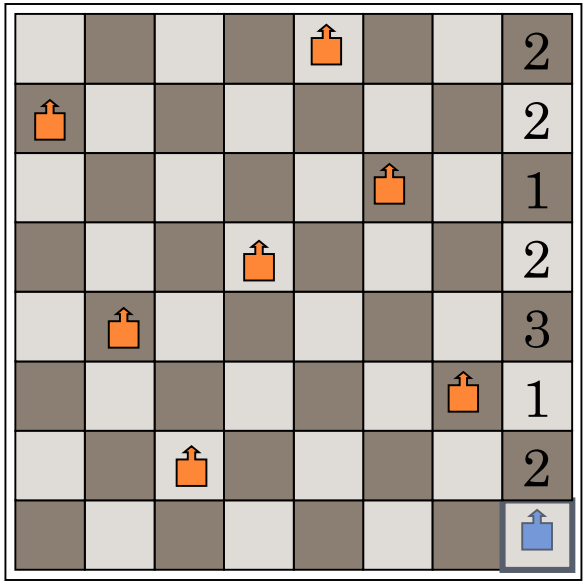


h(n)=2

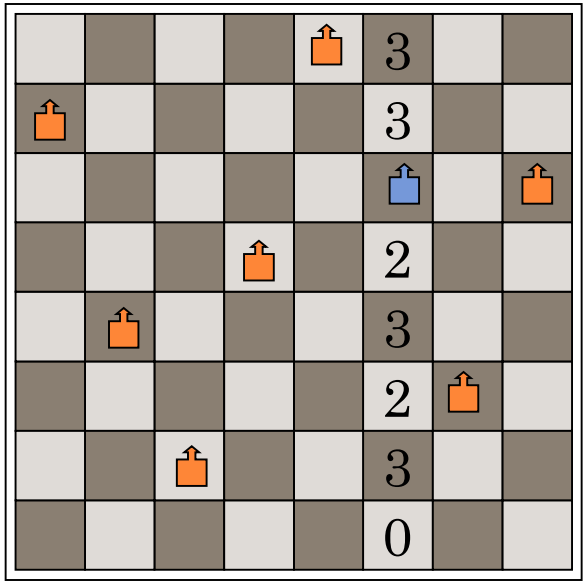


h(n)=5

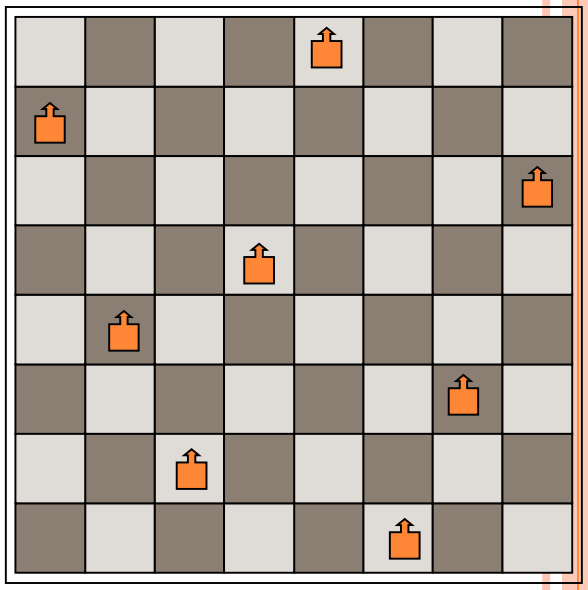
דוגמא: בעית 8 המלכות



1

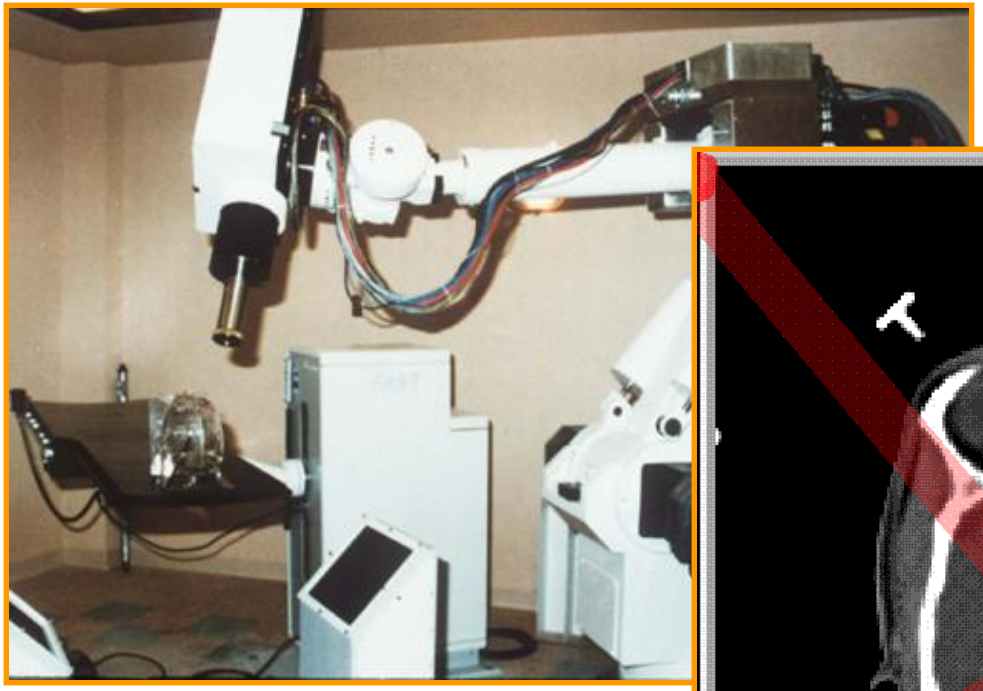


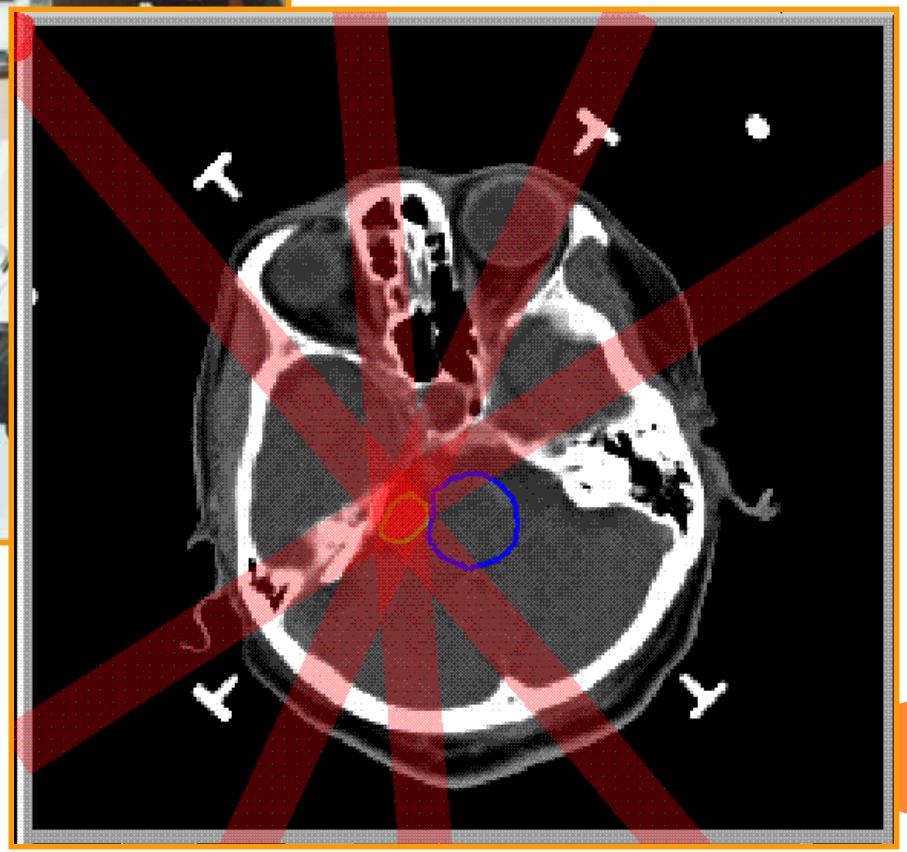
2



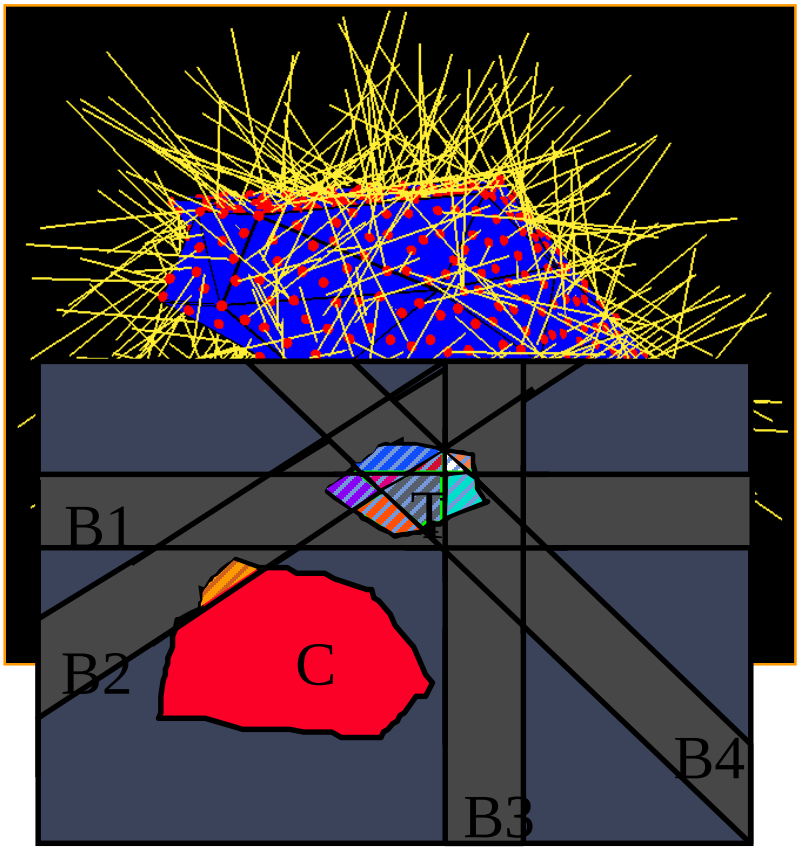
3

Stereotaxic Brain Surgery





Stereotaxic Brain Surgery



- 2000 < Tumor < 2200
  2000 < B2 + B4 < 2200
  2000 < B4 < 2200
  2000 < B3 + B4 < 2200
  2000 < B3 < 2200
  2000 < B1 + B3 + B4 < 2200
  2000 < B1 + B4 < 2200
  2000 < B1 + B2 + B4 < 2200
  2000 < B1 < 2200
  2000 < B1 + B2 < 2200

- 0 < Critical < 500
  0 < B2 < 500

→ Constraint Programming

“Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it.”

Eugene C. Freuder, Constraints, April 1997

When to Use CSP Techniques?

- When the problem can be expressed by a set of variables with constraints on their values
- When constraints are relatively simple (e.g., binary)
- When constraints propagate well (AC3 eliminates many values)
- Local Search: when the solutions are "densely" distributed in the space of possible assignments

Applications

- CSP techniques allow solving very complex problems
- Numerous applications, e.g.:
  - Crew assignments to flights
  - Management of transportation fleet
  - Flight/rail schedules
  - Task scheduling in port operations
  - Design
  - Brain surgery
- See **`www.ilog.com`**

Summary

- CSPs are a special kind of problem: states defined by values of a fixed set of variables, goal test defined by constraints on variable values
- Backtracking=depth-first search with one variable assigned per node
- Variable ordering and value selection heuristics help significantly
- Forward checking prevents assignments that lead to failure.
- Constraint propagation does additional work to constrain values and detect inconsistencies.
- The CSP representation allows analysis of problem structure.
- Tree structured CSPs can be solved in linear time.
- Local Search, e.g., min-conflicts, is often effective in practice.

CPS 270: Artificial Intelligence

http://www.cs.duke.edu/courses/fall08/cps270/

More search: When the path to the solution doesn't matter

Instructor: Vincent Conitzer

Search where the path doesn't matter

- So far, looked at problems where the path was the solution
  - Traveling on a graph
  - Eights puzzle
- However, in many problems, we just want to find a goal state
  - Doesn't matter how we get there

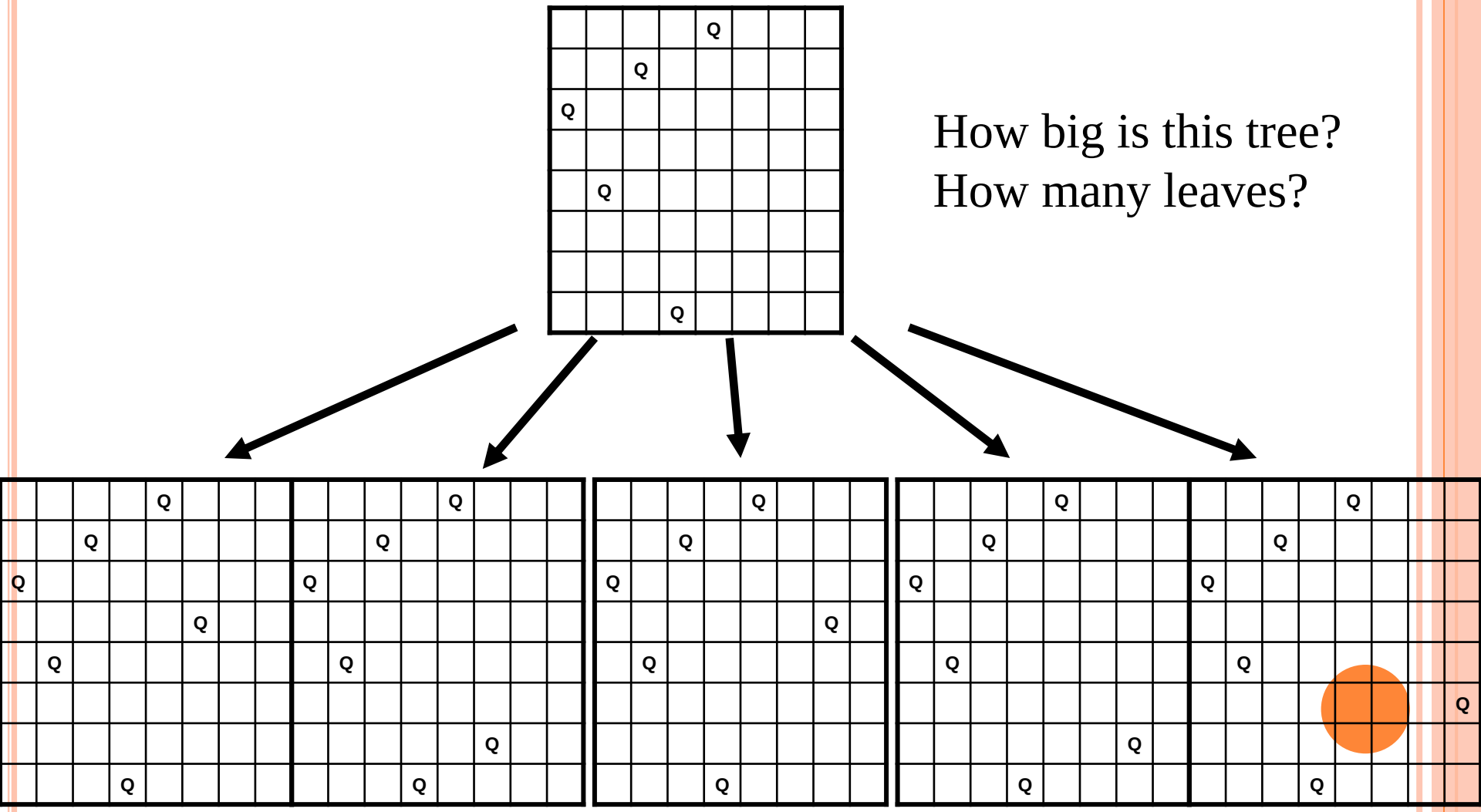
Queens puzzle

- Place eight queens on a chessboard so that no two attack each other

				Q			
		Q					
Q							
						Q	
	Q						
							Q
					Q		
			Q				

Search formulation of the queens puzzle

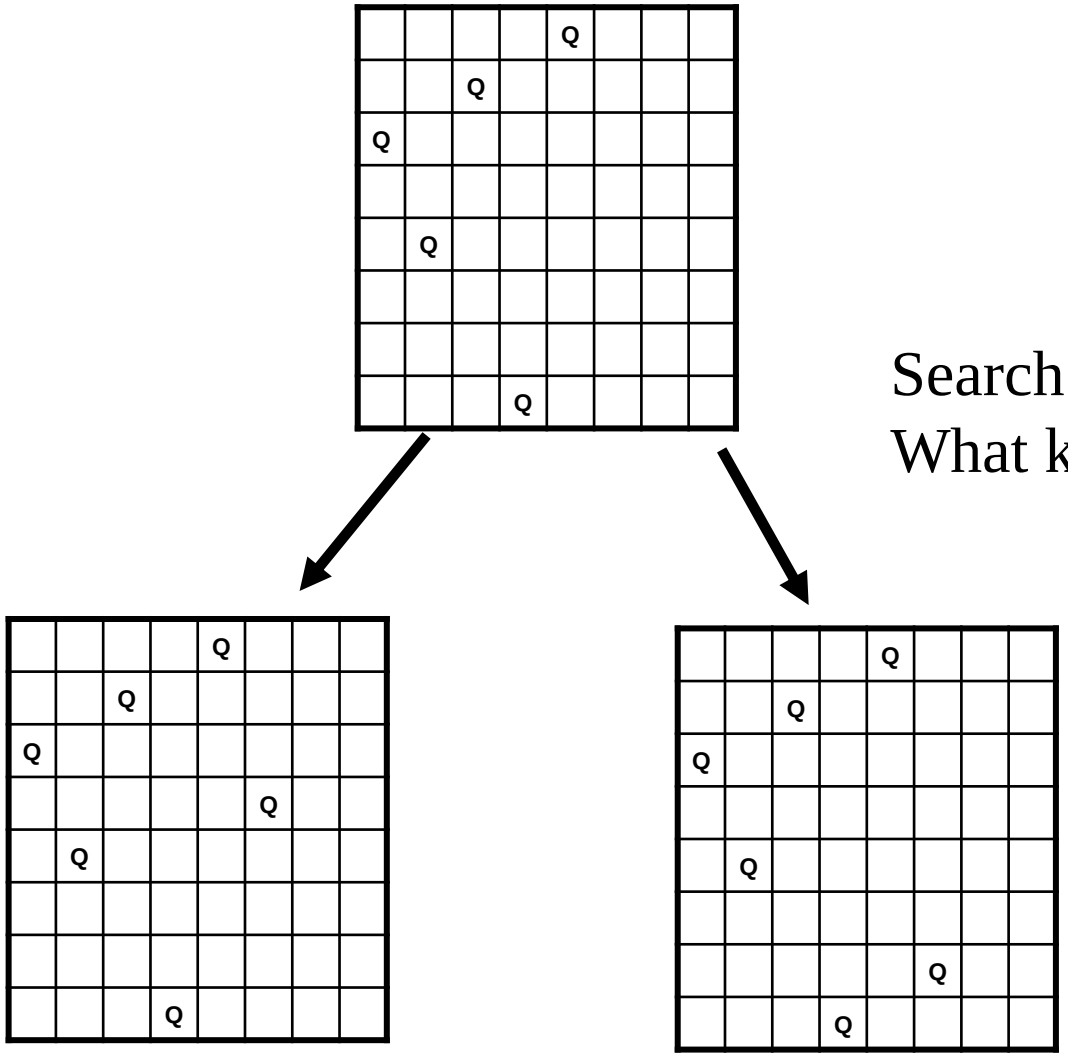
- **Successors**: all valid ways of placing additional queen on the board; **goal**: eight queens placed



How big is this tree?
How many leaves?

Search formulation of the queens puzzle

- **Successors**: all valid ways of placing a queen in the next column; **goal**: eight queens placed



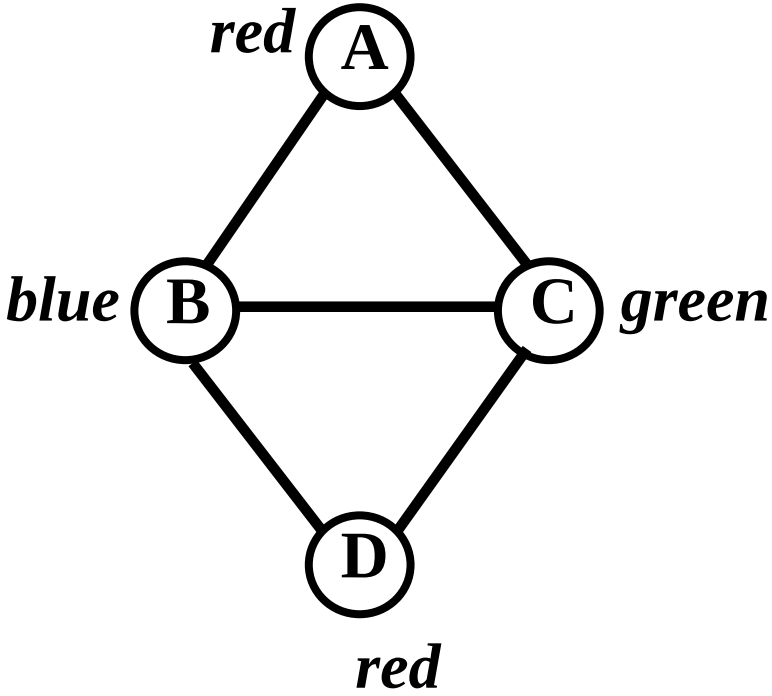
Search tree size?
What kind of search is bes

Constraint Satisfaction Problems (CSPs)

- Defined by:
  - A set of variables $x_1, x_2, \ldots, x_n$
  - A domain D_i for each variable x_i
  - Constraints $c_1, c_2, \ldots, c_m$
- A constraint is specified by
  - A subset (often, two) of the variables
  - All the allowable joint assignments to those variables
- Goal: find a complete, consistent assignment
- Queens problem: (other examples in next slides)
  - x_i in {1, …, 8} indicates in which row in the ith column to place a queen
  - For example, constraint on x_1 and x_2: {(1,3), (1,4), (1,5), (1,6), (1,7), (1,8), (2,4), (2,5), …, (3,1), (3,5), … …}

Graph coloring

- Fixed number of colors; no two adjacent nodes can share a color



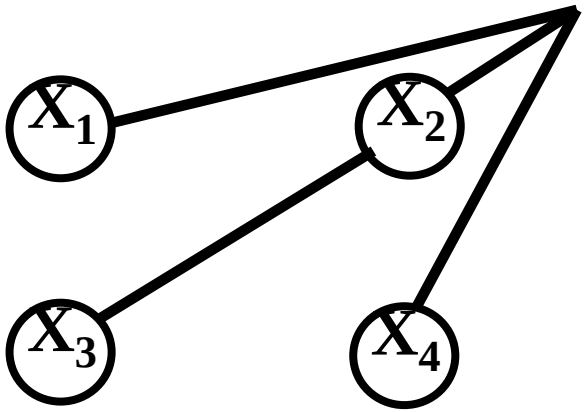
Satisfiability

- Formula in conjunctive normal form:

  (x_1 OR x_2 OR NOT(x_4)) AND (NOT(x_2) OR NOT(x_3)) AND …

  - Label each variable x_j as true or false so that the formula becomes true

Constraint hypergraph: each hyperedge represents a constraint



Cryptarithmetic puzzles

T W O

T W O +

~~F O U R~~

E.g., setting F = 1, O = 4, R = 8, T = 7, W = 3, U = 6 gives 734+734=1468

Cryptarithmetic puzzles…

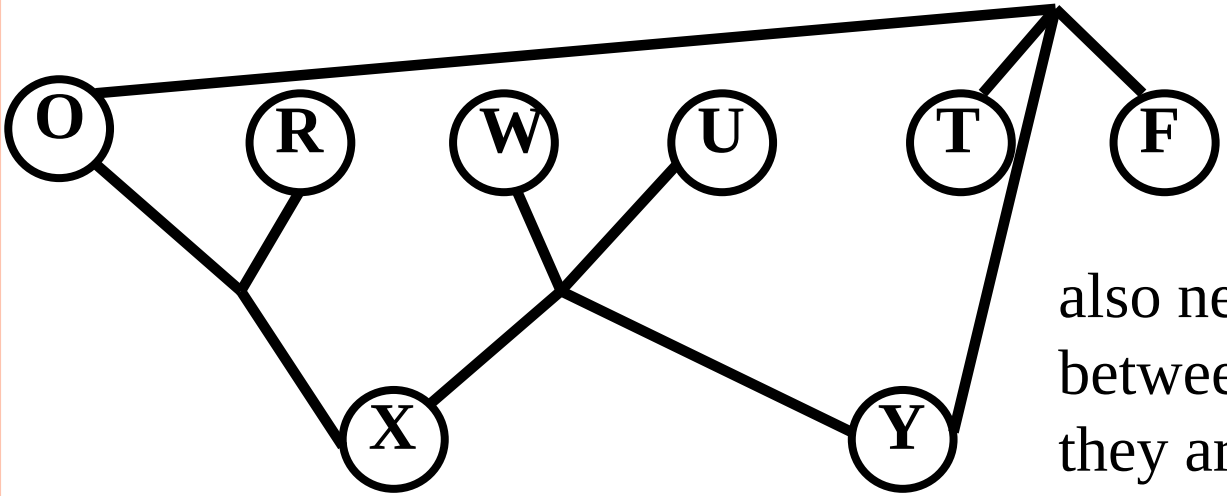
T W O
T W O +
~~F O U R~~

Trick: introduce auxiliary variables X, Y

O + O = 10X + R

W + W + X = 10Y + U

T + T + Y = 10F + O



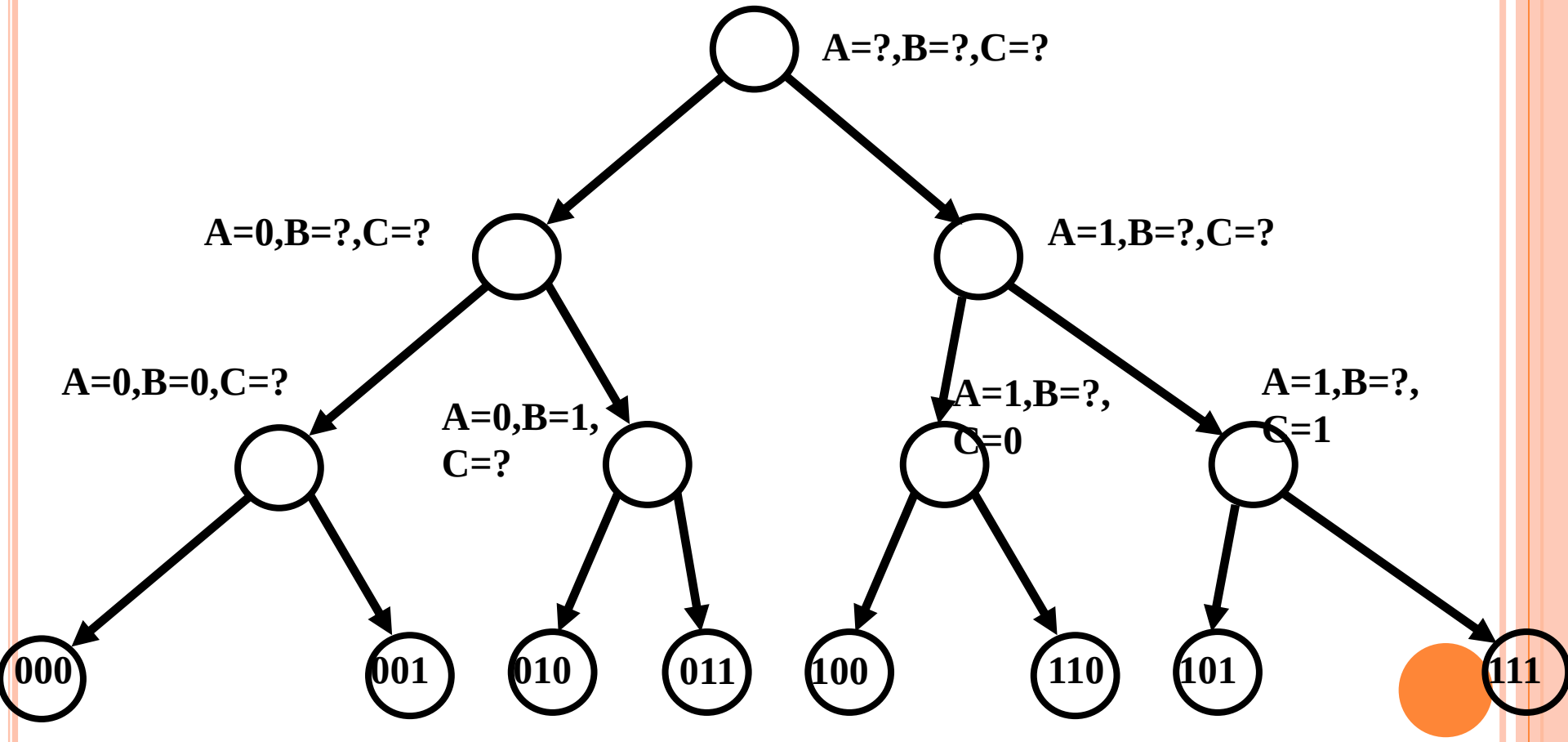
also need pairwise constraints between original variables if they are supposed to be different

Generic approaches to solving CSPs

- State: some variables assigned, others not assigned
- Naïve successors definition: any way of assigning a value to an unassigned variable results in a successor
  - Can check for consistency when expanding
  - How many leaves do we get in the worst case?
- CSPs satisfy commutativity: order in which actions applied does not matter
- Better idea: only consider assignments for a single variable at a time
  - How many leaves?

Choice of variable to branch on is still flexible!

- Do not always need to choose same variable at same level
- Each of variables A, B, C takes values in {0,1}



- Can you prove that this never increases the size of the tree?

A GENERIC RECURSIVE SEARCH ALGORITHM

- **Search(*assignment*, *constraints*)**
- If *assignment* is complete, return it
- Choose an unassigned variable *x*
- For every value *v* in *x*'s domain, if setting *x* to *v* in *assignment* does not violate *constraints:*
  - Set *x* to *v* in *assignment*
  - *result* := Search(*assignment*, *constraints*)
  - If *result* != *failure* return *result*
  - Unassign *x* in *assignment*
- Return *failure*

Keeping track of remaining possible values

- For every variable, keep track of which values are still possible

				Q	X	X	X
		Q			X	X	X
Q					X	X	X
							X
	Q				X	X	X
					X	X	
							X
			Q		X	X	X

				Q	X	X	
		Q			X	X	
Q					X	X	
					X		
	Q				X	X	
					X	X	Q
						X	
			Q		X	X	

				Q			
		Q					
Q							
						Q	
	Q						
							Q
					Q		
			Q				

only one possibility for last column; might as well fill in

now only one left for other two columns

done! (no real branching needed!)

- General heuristic: branch on variable with fewest values remaining

Arc consistency

- Take two variables connected by a constraint
- Is it true that for **every** remaining value *d* of the first variable, there exists **some** value *d'* of the other variable so that the constraint is satisfied?
  - If so, we say the arc from the first to the second variable is consistent
  - If not, can remove the value *d*
- General concept: constraint propagation

Q				X			X
				X			
		Q		X			X
				X			
				X	Q		X
				X			

Is the arc from the fifth to the eighth column consistent? What about the arc from the eighth to

Maintaining arc consistency

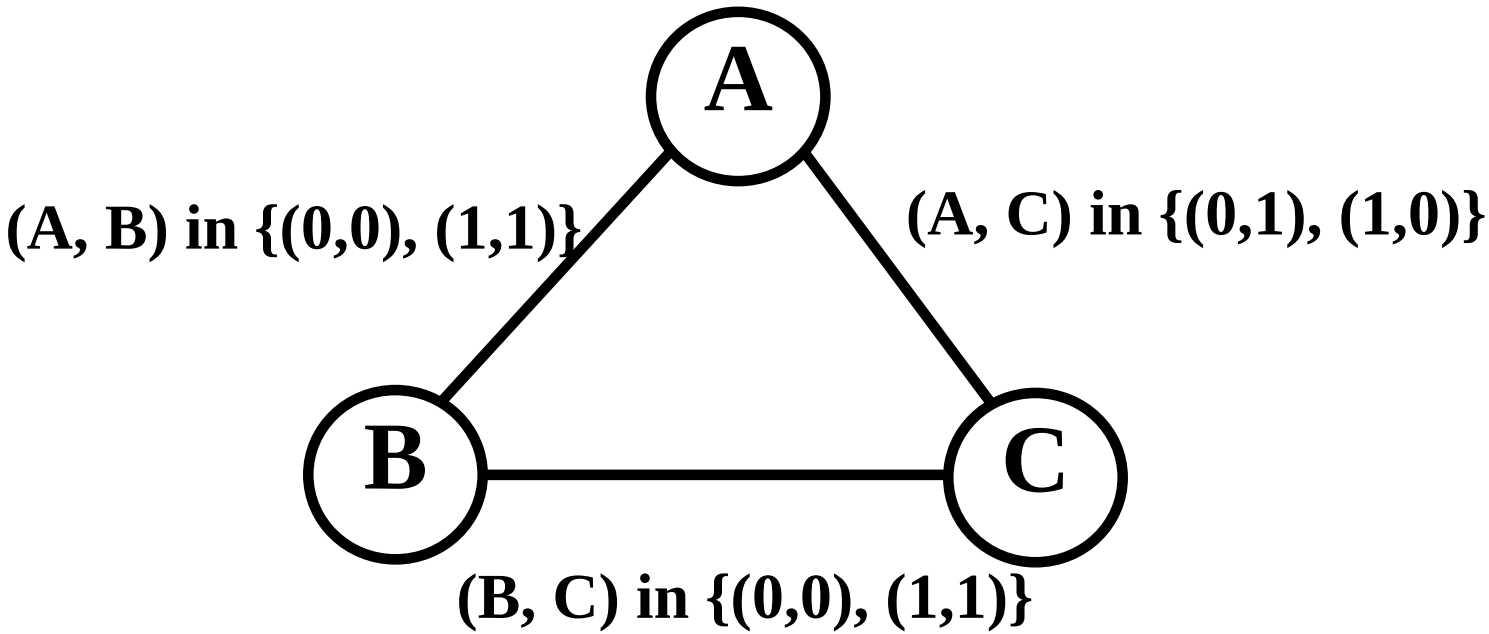
- Maintain a queue Q of all ordered pairs of variables with a constraint (arcs) that need to be checked
- Take a pair (x, y) from the queue
- For every value v in x's domain, check if there is some value w in y's domain so that $x=v$, $y=w$ is consistent
  - If not, remove v from x's domain
- If anything was removed from x's domain, add **every** arc (z, x) to Q
- Continue until Q is empty

- Runtime?
- n variables, d values per domain
- $O(n^2)$ arcs;
- each arc is added to the queue at most d times;
- consistency of an arc can be checked with d^2 lookups in the constraint's table;
- so $O(n^2d^3)$ lookups
- Can we do better?

Maintaining arc consistency (2)

- For every arc (x, y), for every value v for x, maintain the number $n((x, y), v)$ of remaining values for y that are consistent with $x=v$
- Every time that some $n((x, y), v) = 0$,
  - remove v from x's domain;
  - for every arc (z, x), for every value w for z, if $(x=v, z=w)$ is consistent with the constraint, reduce $n((z, x), w)$ by 1
- Runtime:
  - for every arc (z, x) (n^2 of them), a value is removed from x's domain at most d times;
  - each time we have to check for at most d of z's values whether it is consistent with the removed value for x;
  - so $O(n^2d^2)$ lookups

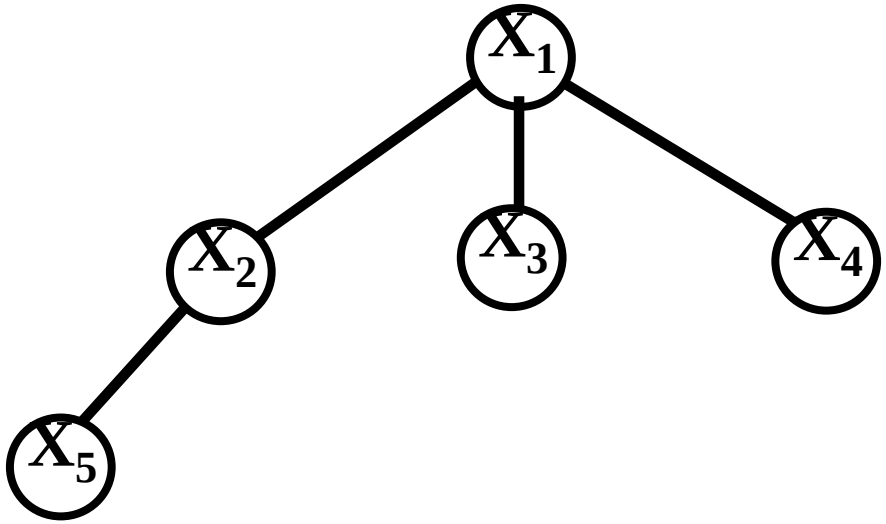
An example where arc consistency fails



- A = B, B = C, C ≠ A – obviously inconsistent
  - ~ Moebius band
- However, arc consistency cannot eliminate anything

Tree-structured constraint graphs

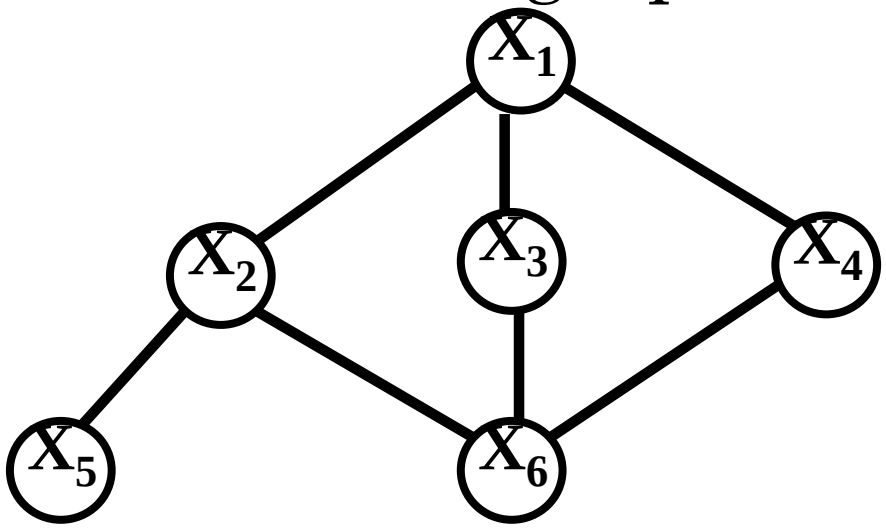
- Suppose we only have pairwise constraints and the graph is a tree (or forest = multiple disjoint trees)



- Dynamic program for solving this (linear in #variables):
  - Starting from the leaves and going up, for each node *x*, compute all the values for *x* such that the subtree rooted at *x* can be solved
    - Equivalently: apply arc consistency from each parent to its children, starting from the bottom
  - If no domain becomes empty, once we reach the top, easy to fill in solution

Generalizations of the tree-based approach

- What if our constraint graph is "almost" a tree?



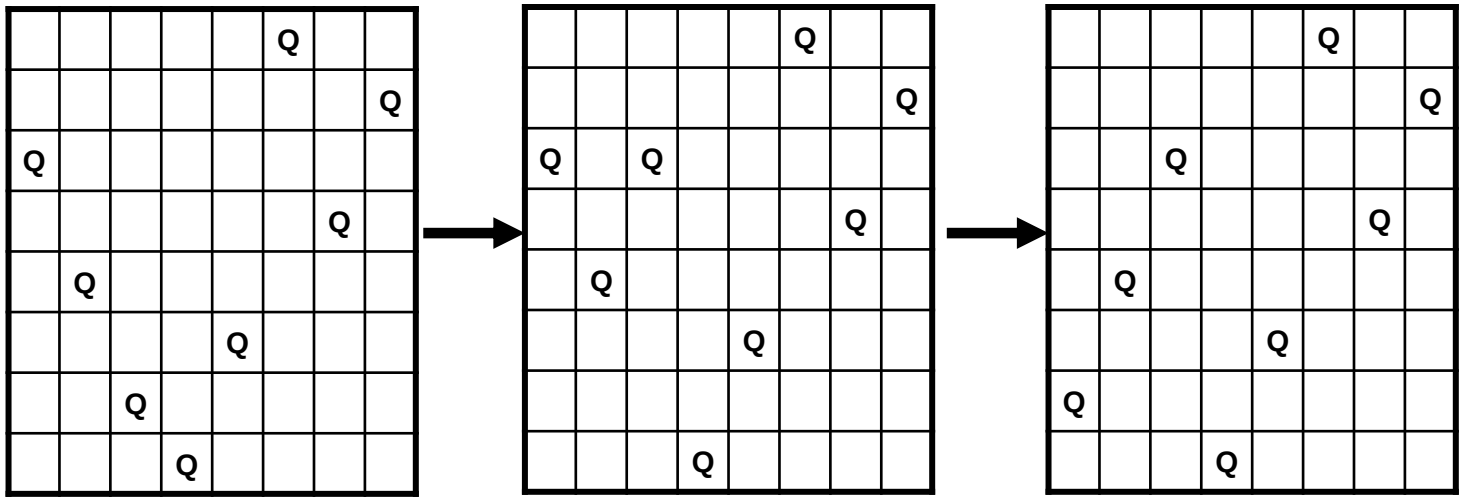
- A cycle cutset is a set of variables whose removal results in a tree (or forest)
  - E.g. $\{X_1\}$, $\{X_6\}$, $\{X_2, X_3\}$, $\{X_2, X_4\}$, $\{X_3, X_4\}$
- Simple algorithm: for every internally consistent assignment to the cutset, solve the remaining tree as before (runtime?)
- Graphs of bounded treewidth can also be solved in polynomial time (won't define these here)

A different approach: optimization

- Let's say every way of placing 8 queens on a board, one per column, is feasible
- Now we introduce an objective: minimize the number of pairs of queens that attack each other
  - More generally, minimize the number of violated constraints
- Pure optimization

Local search: hill climbing

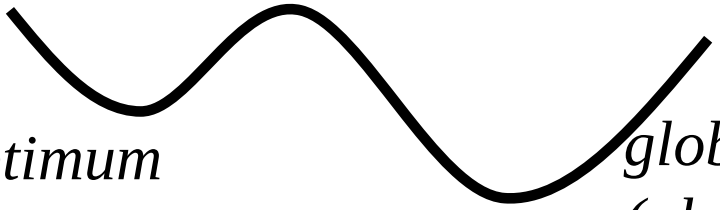
- Start with a complete state
- Move to successor with best (or at least better) objective value
  - Successor: move one queen within its column



4 attacking pairs *3 attacking pairs* *2 attacking pairs*

no more improvements

- Local search can get stuck in a local optimum



local optimum

global optimum (also a local

Avoiding getting stuck with local search

- Random restarts: if your hill-climbing search fails (or returns a result that may not be optimal), restart at a random point in the search space
  - Not always easy to generate a random state
  - Will **eventually** succeed (why?)
- Simulated annealing:
  - Generate a random successor (possibly worse than current state)
  - Move to that successor with some probability that is sharply decreasing in the badness of the state
  - Also, over time, as the "temperature decreases," probability of bad moves goes down

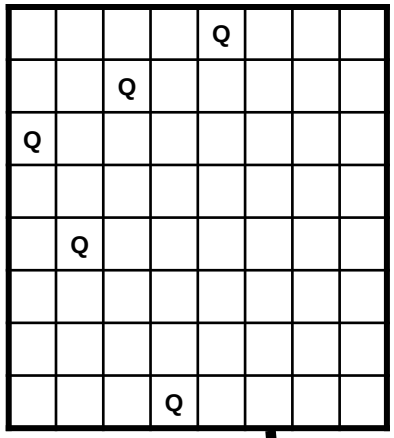
Constraint optimization

- Like a CSP, but with an objective
  - E.g., minimize number of violated constraints
  - Another example: no two queens can be in the same row or column (hard constraint), minimize number of pairs of queens attacking each other diagonally (objective)
- Can use all our techniques from before: heuristics, A*, IDA*, …
- Also popular: depth-first branch-and-bound
  - Like depth-first search, except do not stop when first feasible solution found; keep track of best solution so far
  - Given admissible heuristic, do not need to explore nodes that are worse than best solution found so far

Minimize #violated diagonal constraints

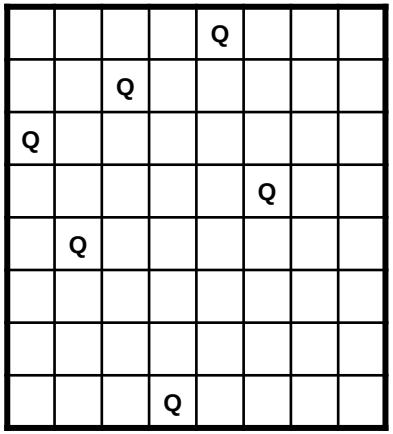
- **Cost of a node**: #violated diagonal constraints so far
- No heuristic

(matter of definition; could just as well say that violated constraints so far is the heuristic and interior nodes have no cost)

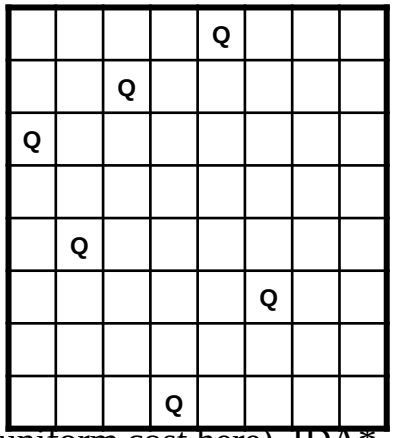


cost = 0

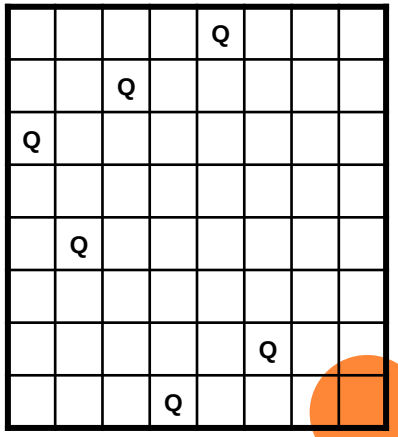
cost = 0



cost = 1



cost = 0



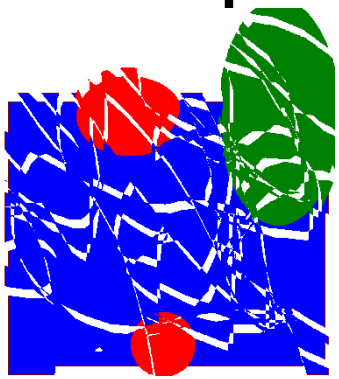
Depth first branch and bound will find a suboptimal solution here first (no way to tell at this point this is worse than right node)

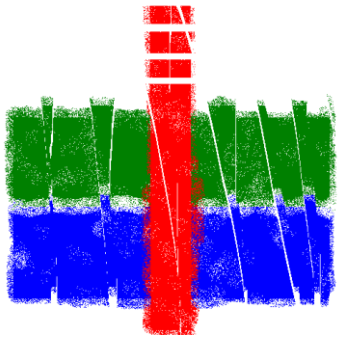
A* (=uniform cost here), IDA* (=iterative lengthening here) will **never** explore this node

Optimal solution is down here (cost 0)

Linear programs: example

- We make reproductions of two paintings





$$\begin{aligned} &\textit{maximize } 3x + 2y \\ &\textit{subject to} \\ &4x + 2y \leq 16 \\ &x + 2y \leq 8 \\ &x + y \leq 5 \\ &x \geq 0 \\ &y \geq 0 \end{aligned}$$

- Painting 1 sells for $30, painting 2 sells for $20
- Painting 1 requires 4 units of blue, 1 green, 1 red
- Painting 2 requires 2 blue, 2 green, 1 red
- We have 16 units blue, 8 green, 5 red

Solving the linear program graphically

maximize $3x + 2y$

subject to

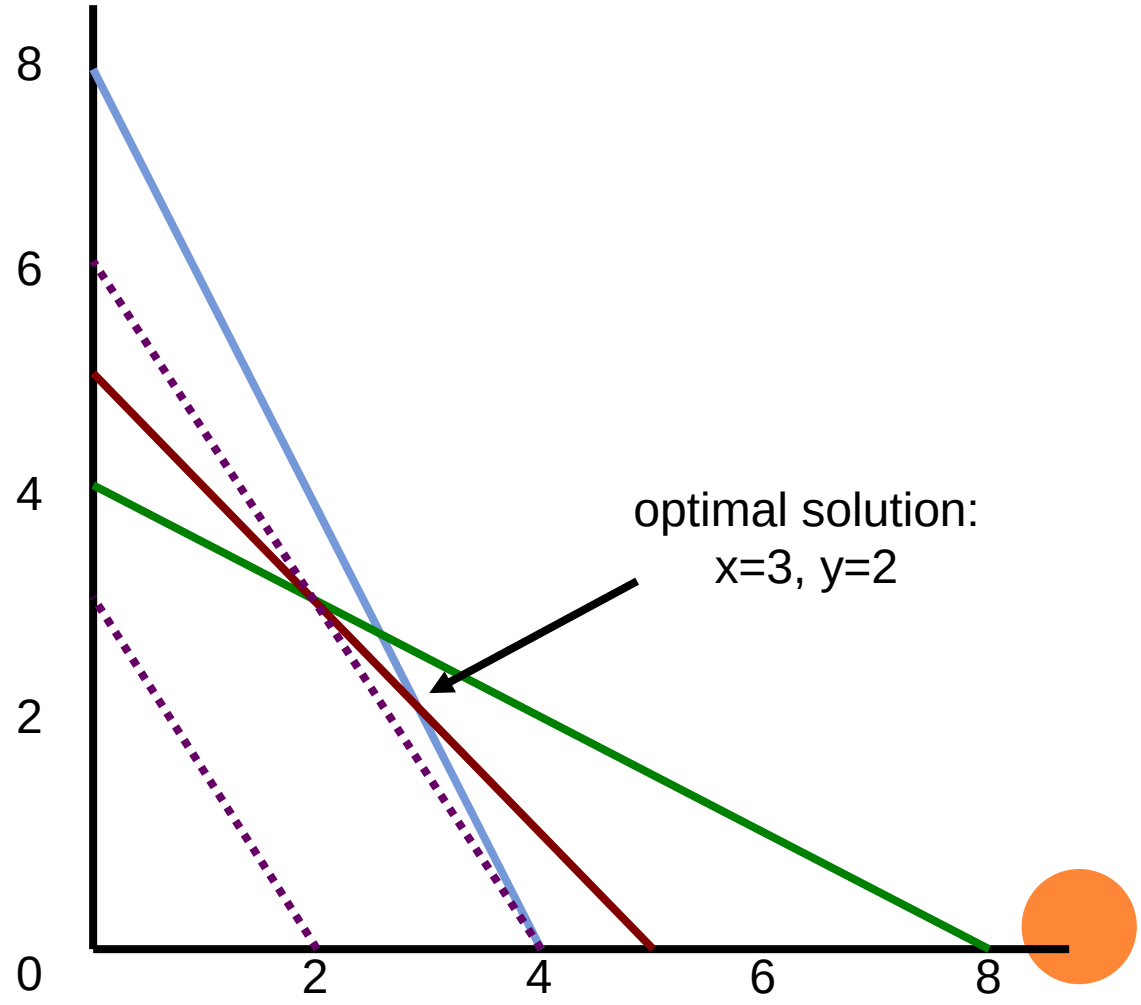
$4x + 2y \leq 16$

$x + 2y \leq 8$

$x + y \leq 5$

$x \geq 0$

$y \geq 0$



Modified LP

maximize $3x + 2y$

subject to

$4x + 2y \leq 15$

$x + 2y \leq 8$

$x + y \leq 5$

$x \geq 0$

$y \geq 0$

Optimal solution: x = 2.5, y = 2.5

Solution value = 7.5 + 5 = 12.5

Half paintings?

Integer (linear) program

maximize $3x + 2y$

subject to

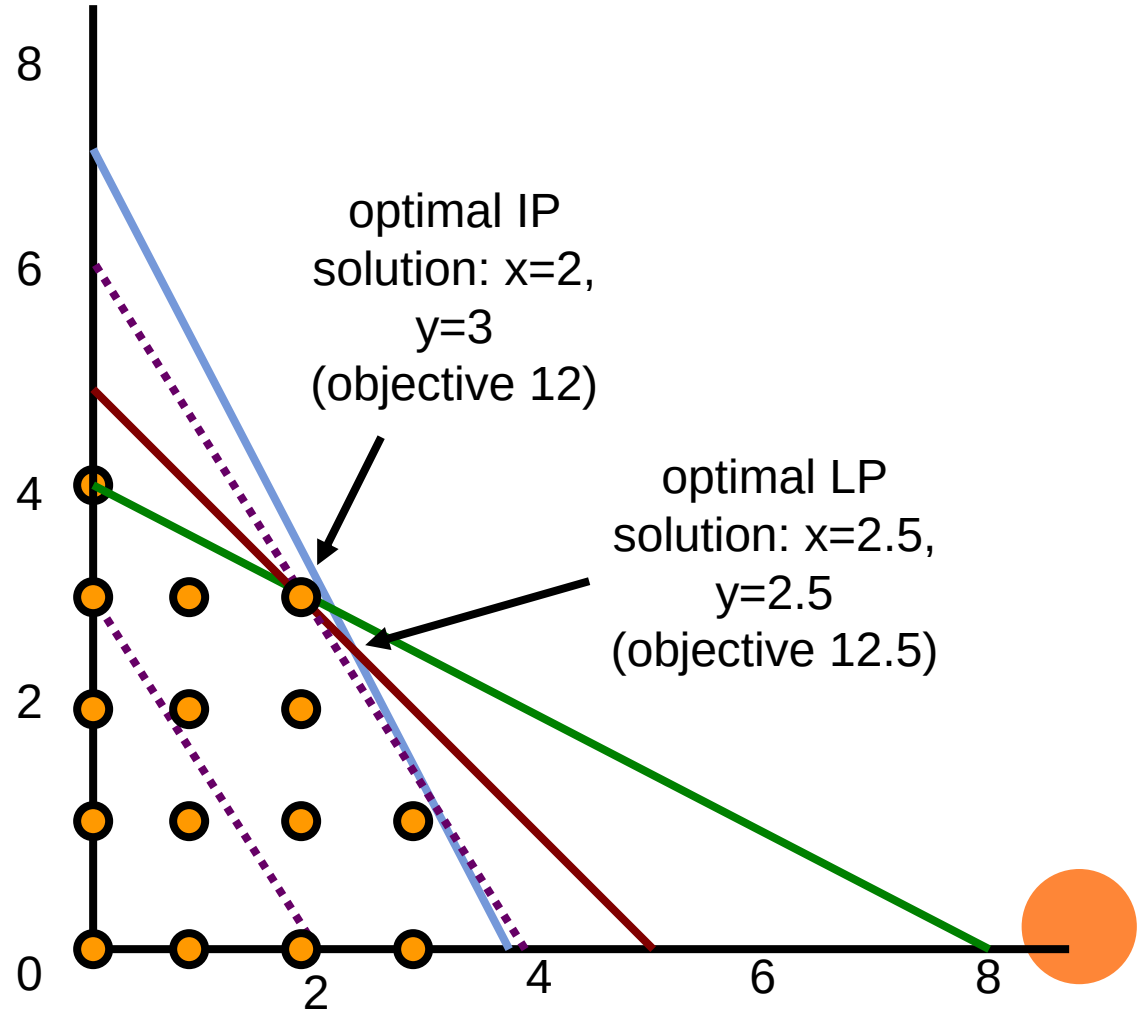
$4x + 2y \leq 15$

$x + 2y \leq 8$

$x + y \leq 5$

$x \geq 0$, integer

$y \geq 0$, integer



Mixed integer (linear) program

maximize $3x + 2y$

subject to

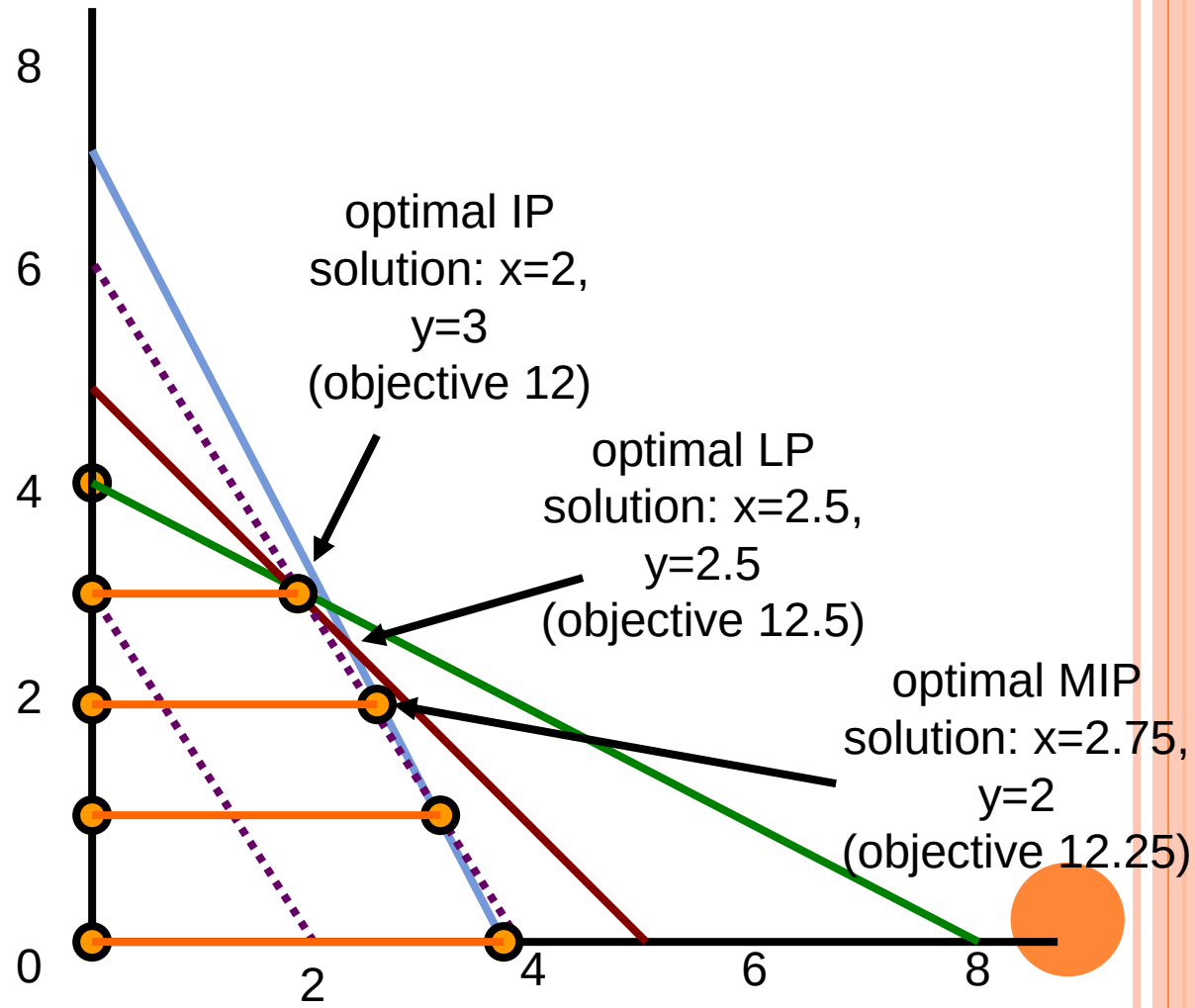
$4x + 2y \leq 15$

$x + 2y \leq 8$

$x + y \leq 5$

$x \geq 0$

$y \geq 0$, integer



Solving linear/integer programs

- Linear programs can be solved efficiently
  - Simplex, ellipsoid, interior point methods…
- (Mixed) integer programs are NP-hard to solve
  - Quite easy to model many standard NP-complete problems as integer programs (try it!)
  - Search type algorithms such as branch and bound
- Standard packages for solving these
  - GNU Linear Programming Kit, CPLEX, …
- LP relaxation of (M)IP: remove integrality constraints
  - Gives upper bound on MIP (~admissible heuristic)

Satisfiability as an integer program

$(x_1$ OR x_2 OR NOT$(x_4))$ AND $($NOT(x_2) OR NOT$(x_3))$ AND

…

becomes

for all x_j, $0 \leq x_j \leq 1$, x_j integer (shorthand: x_j in $\{0,1\}$)

$x_1 + x_2 + (1-x_4) \geq 1$

$(1-x_2) + (1-x_3) \geq 1$

…

Solving integer programs is at least as hard as satisfiability, hence NP-hard (we have reduced SAT to IP)

Try modeling other NP-hard problems as (M)IP!

Solving the integer program with DFS branch and bound

trick: for integer x and k, either $x \le k$ or $x \ge k+1$

maximize $3x + 2y$
subject to
$4x + 2y \le 15$
$x + 2y \le 8$
$x + y \le 5$

LP solution: x=2.5, y=2.5, obj = 12.5

maximize $3x + 2y$
subject to
$4x + 2y \le 15$
$x + 2y \le 8$
$x + y \le 5$
$x \ge 3$

LP solution: x=3, y=1.5, obj = 12

maximize $3x + 2y$
subject to
$4x + 2y \le 15$
$x + 2y \le 8$
$x + y \le 5$
$x \le 2$

LP solution: x=2, y=3, obj = 12

maximize $3x + 2y$
subject to
$4x + 2y \le 15$
$x + 2y \le 8$
$x + y \le 5$
$x \ge 3$
$y \ge 2$

LP solution: infeasible

maximize $3x + 2y$
subject to
$4x + 2y \le 15$
$x + 2y \le 8$
$x + y \le 5$
$x \ge 3$
$y \le 1$

LP solution: x=3.25, y=1, obj = 11.75

maximize $3x + 2y$
subject to
$4x + 2y \le 15$
$x + 2y \le 8$
$x + y \le 5$
$x \ge 3$
$y \le 1$
$x \ge 4$

LP solution: infeasible

maximize $3x + 2y$
subject to
$4x + 2y \le 15$
$x + 2y \le 8$
$x + y \le 5$
$x \ge 3$
$y \le 1$
$x \le 3$

LP solution: x=3, y=1, obj = 11

if LP solution is integral, we are done

Again with a more fortunate choice

maximize $3x + 2y$
subject to
$4x + 2y \leq 15$
$x + 2y \leq 8$
$x + y \leq 5$

LP solution: x=2.5, y=2.5, obj = 12.5

maximize $3x + 2y$
subject to
$4x + 2y \leq 15$
$x + 2y \leq 8$
$x + y \leq 5$
$x \geq 3$

LP solution: x=3, y=1.5, obj = 12

maximize $3x + 2y$
subject to
$4x + 2y \leq 15$
$x + 2y \leq 8$
$x + y \leq 5$
$x \leq 2$

LP solution: x=2, y=3, obj = 12

done!

Russell and Norvig:
Chapter 5

Constraint Satisfaction Problems

CMSC 421 – Fall 2006

Outline

- Constraint Satisfaction Problems (CSPs)
- Backtracking for CSP
- Local search for CSPs
- Problem structure and decomposition

Exercise #2: Sudoku

- But, before we get into all that, let's do a puzzle…
- http://www.websudoku.com/
- Each Sudoku has a unique solution that can be reached logically without guessing. Enter digits from 1 to 9 into the blank spaces. Every row must contain one of each digit. So must every column, as must every 3x3 square.

Puzzle

				9		3		
	2		6		1	5	9	
3			5		8		2	4
	9					4	8	
6			2		9			1
	5	2					3	
9	7		1		5			3
	3	8	7		4		1	
		5		8				

Problem Formulation

- What is the state space?
- What is the initial state?
- What is the successor function?
- What is the goal test?
- What is the path cost?

Points

- Large search space
- commutativity
- what about using bfs or dfs?
  - fixed depth

CSP as a Search Problem

- Initial state: empty assignment
- Successor function: a value is assigned to any unassigned variable, which does not conflict with the currently assigned variables
- Goal test: the assignment is complete
- Path cost: irrelevant

- Branching factor b at the top level is nd.
- $b=(n-l)d$ at depth l, hence $n!d^n$ leaves (only d^n complete assignments).

We can do better…….

What else is needed?

- Not just a successor function and goal test
- But also a means to propagate the constraints imposed by one move on the others and an early failure test
- → Explicit representation of constraints and constraint manipulation algorithms

Constraint Satisfaction Problem

- Set of variables $\{X_1, X_2, \ldots, X_n\}$
- Each variable X_i has a domain D_i of possible values
  - Usually D_i is discrete and finite
- Set of constraints $\{C_1, C_2, \ldots, C_p\}$
  - Each constraint C_k involves a subset of variables and specifies the allowable combinations of values of these variables
- Goal: Assign a value to every variable such that all constraints are satisfied

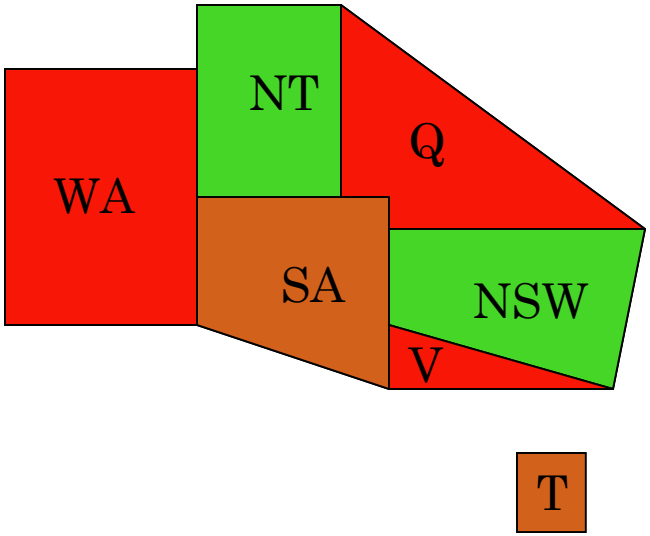
Example: Sudoku CSP

- variables:
- domains:
- constraints:
- Goal: Assign a value to every variable such that all constraints are satisfied

Example: Sudoku CSP

- variables: $X_{11}, \ldots, X_{99}$
- domains: $\{1,\ldots,9\}$
- constraints:
  - row constraint: $X_{11} \neq X_{12}, \ldots, X_{11} \neq X_{19}$
  - col constraint: $X_{11} \neq X_{12}, \ldots, X_{11} \neq X_{19}$
  - block constraint: $X_{11} \neq X_{12}, \ldots, X_{11} \neq X_{33}$
- Goal: Assign a value to every variable such that all constraints are satisfied

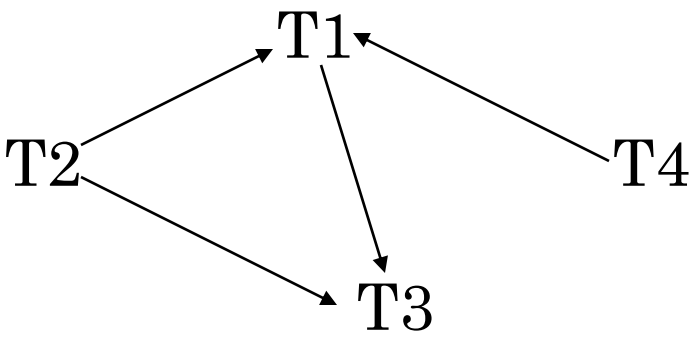
Example: Map Coloring



- 7 variables {WA,NT,SA,Q,NSW,V,T}
- Each variable has the same domain {red, green, blue}
- No two adjacent variables have the same value:

WA≠NT, WA≠SA, NT≠SA, NT≠Q, SA≠Q, SA≠NSW, SA≠V,Q≠NSW, NSW≠V

Example: Task Scheduling



T1 must be done during T3
T2 must be achieved before T1 starts
T2 must overlap with T3
T4 must start after T1 is complete

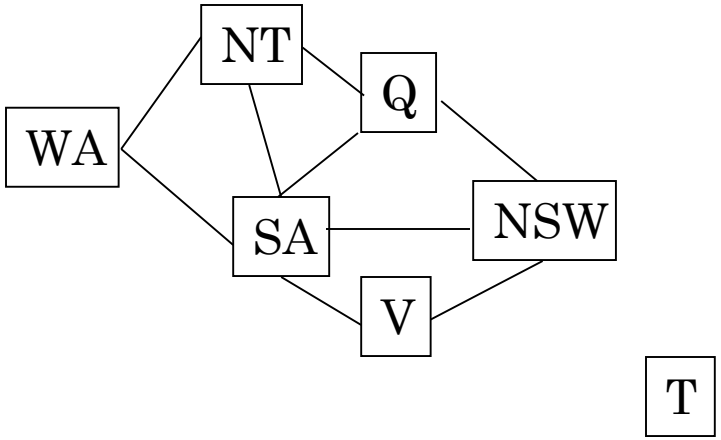
- Are the constraints compatible?
- Find the temporal relation between every two tasks

Varieties of constraints

- Unary constraints involve a single variable.
  - e.g. $SA \neq green$
- Binary constraints involve pairs of variables.
  - e.g. $SA \neq WA$
- Higher-order constraints involve 3 or more variables.
  - e.g. cryptharithmetic column constraints.
- Preference (soft constraints) e.g. *red* is better than *green* often representable by a cost for each variable assignment → constrained optimization problems.

Constraint Graph

Binary constraints



What is the constraint graph for sudoku?

Two variables are adjacent or neighbors if they are connected by an edge or an arc

Remark

- Finite CSP include 3SAT as a special case (see class on logic)
- 3SAT is known to be NP-complete
- So, in the worst-case, we cannot expect to solve a finite CSP in less than exponential time

Commutativity of CSP

The order in which values are assigned to variables is irrelevant to the final assignment, hence:

1. Generate successors of a node by considering assignments for only one variable

2. Do not store the path to node

Backtracking Algorithm

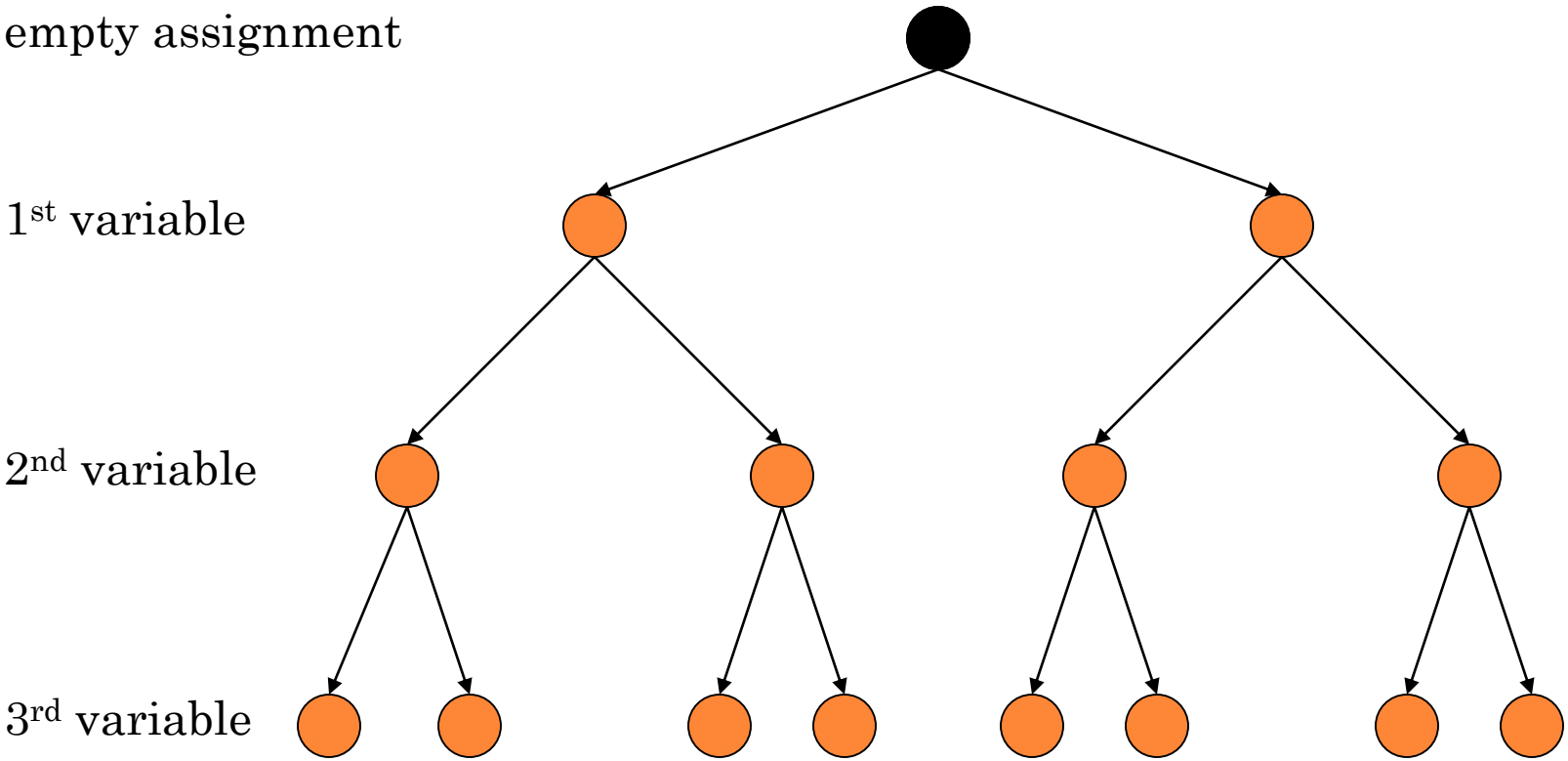
partial assignment of variables

CSP-BACKTRACKING({})

CSP-BACKTRACKING(a)

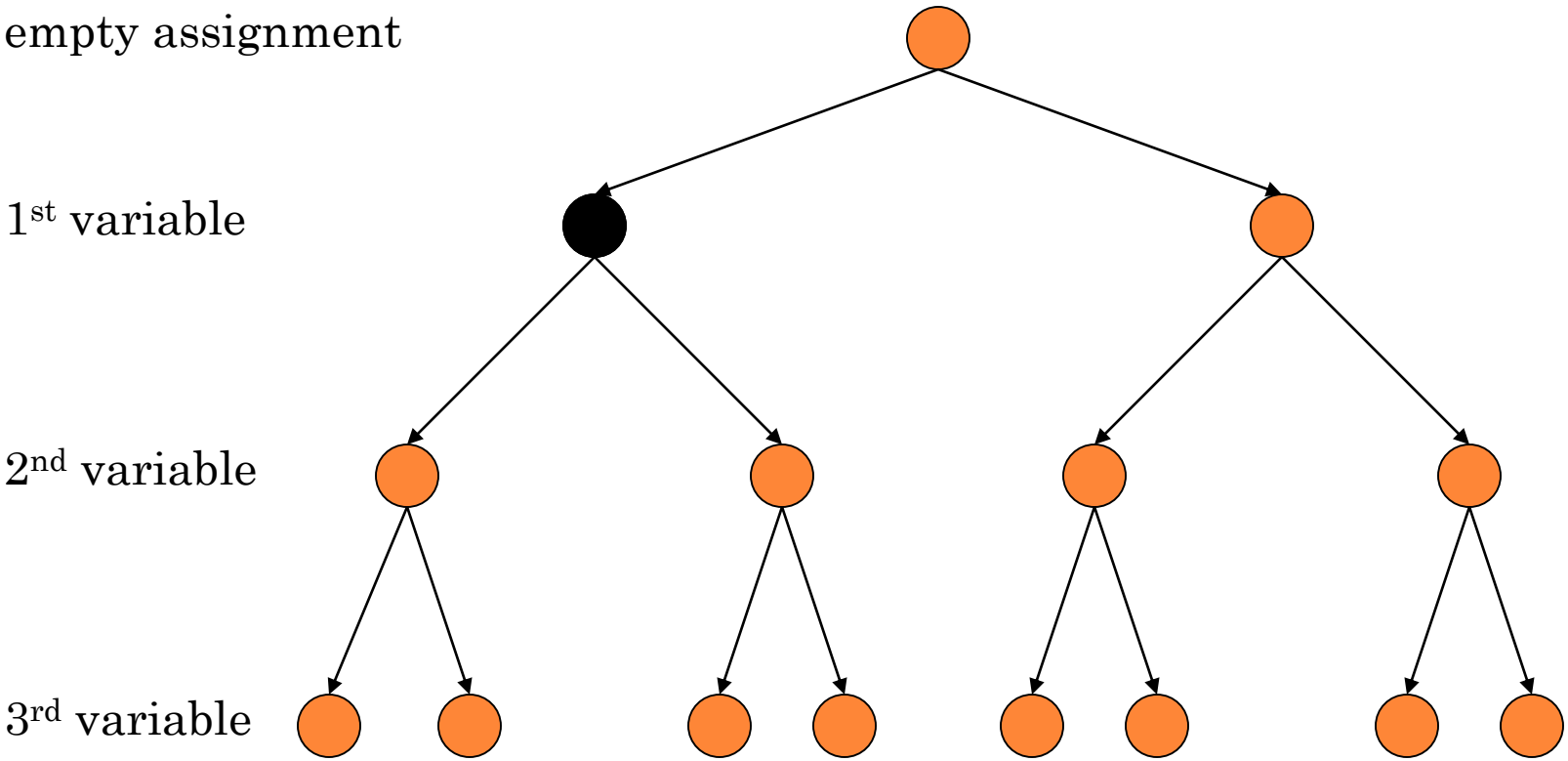
- If a is complete then return a
- X ← select unassigned variable
- D ← select an ordering for the domain of X
- For each value v in D do
  - If v is consistent with a then
    - Add (X= v) to a
    - result ← CSP-BACKTRACKING(a)
    - If result ≠ *failure* then return result
- Return *failure*

→ Backtracking Search



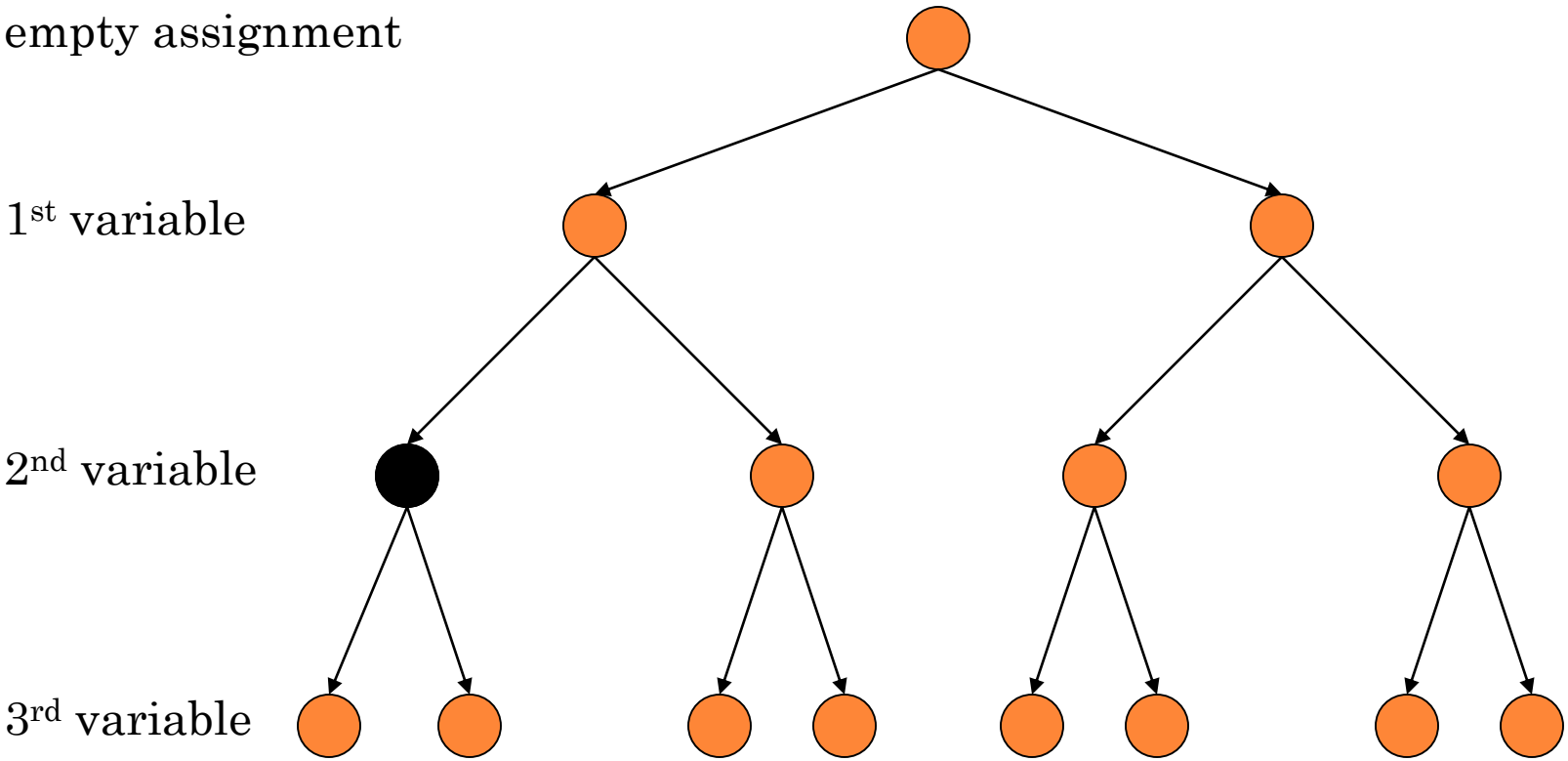
Assignment = {}

→ Backtracking Search



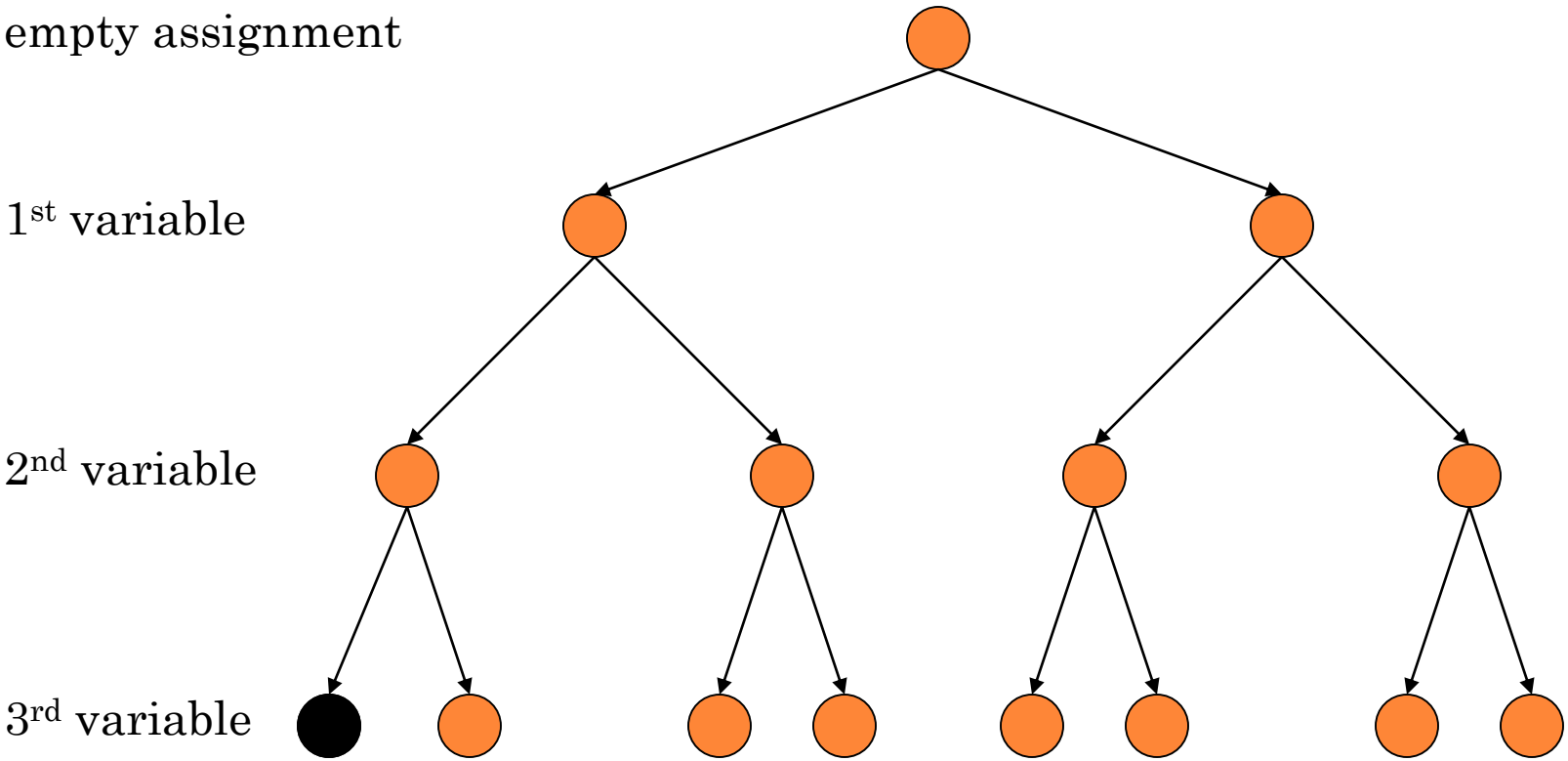
Assignment = {(var1=v11)}

→ Backtracking Search



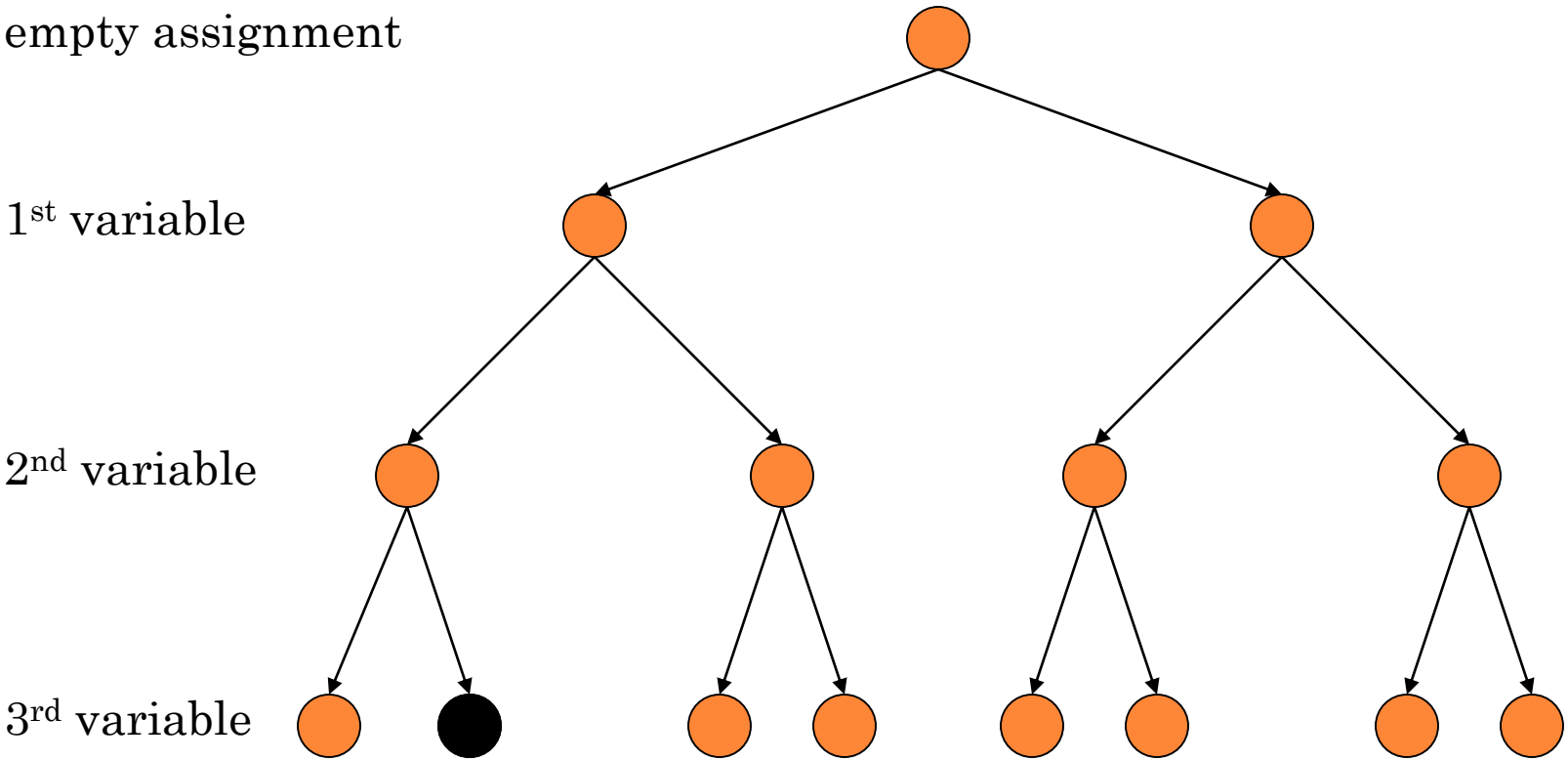
Assignment = {(var1=v11),(var2=v21)}

→ Backtracking Search



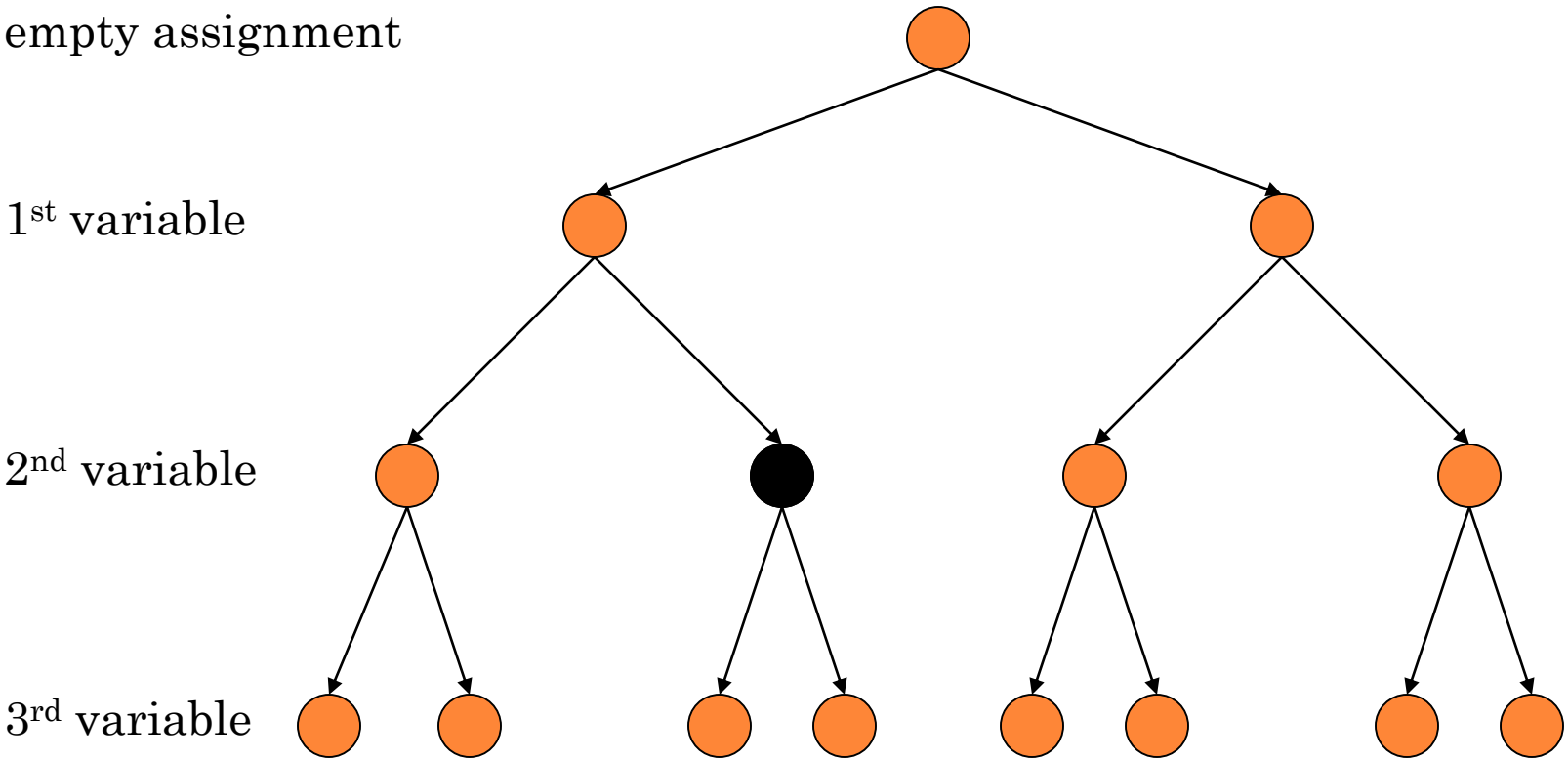
Assignment = {(var1=v11),(var2=v21),(var3=v31)}

→ Backtracking Search



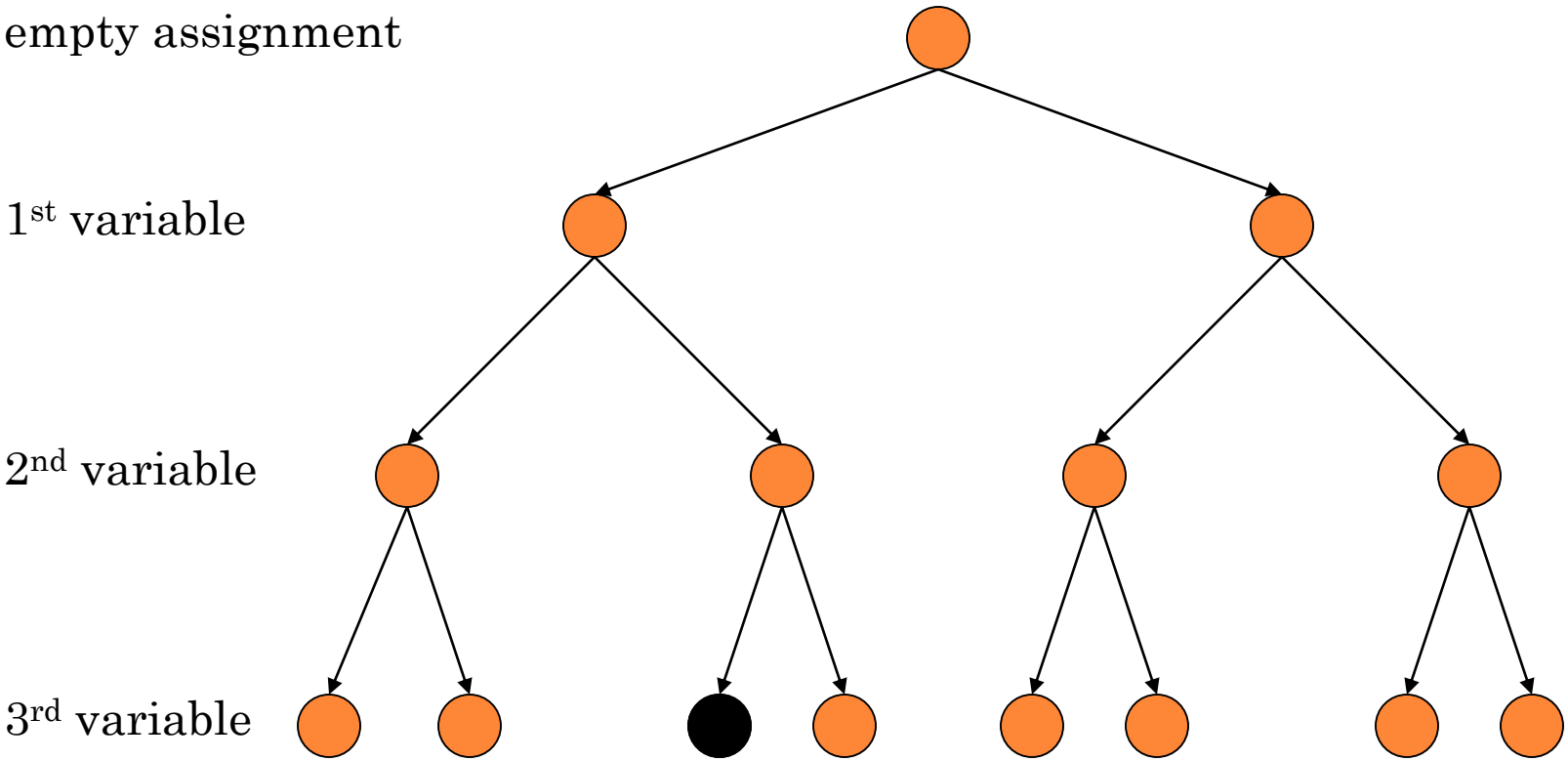
Assignment = {(var1=v11),(var2=v21),(var3=v32)}

→ Backtracking Search



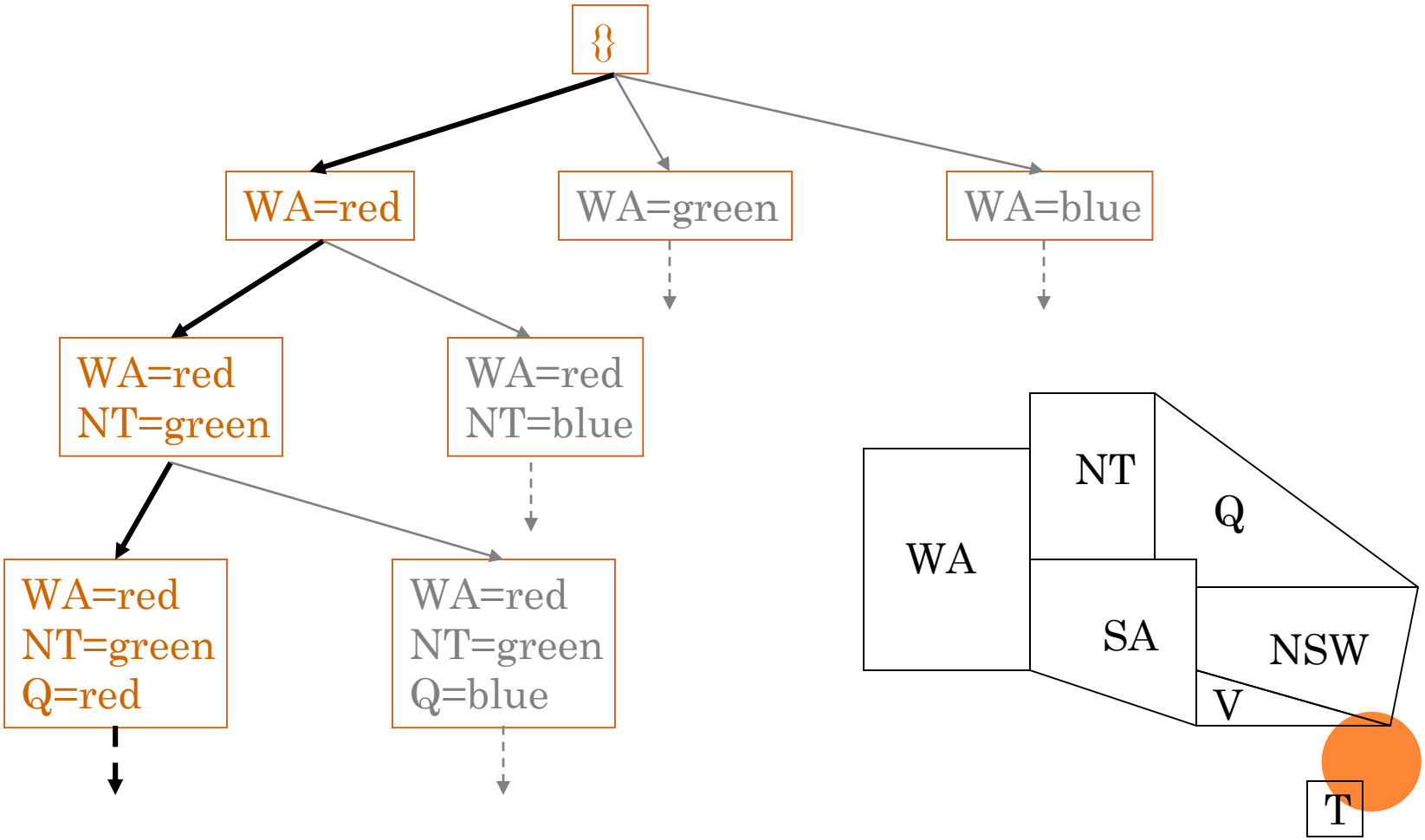
Assignment = {(var1=v11),(var2=v22)}

→ Backtracking Search



Assignment = {(var1=v11),(var2=v22),(var3=v31)}

Map Coloring



Questions

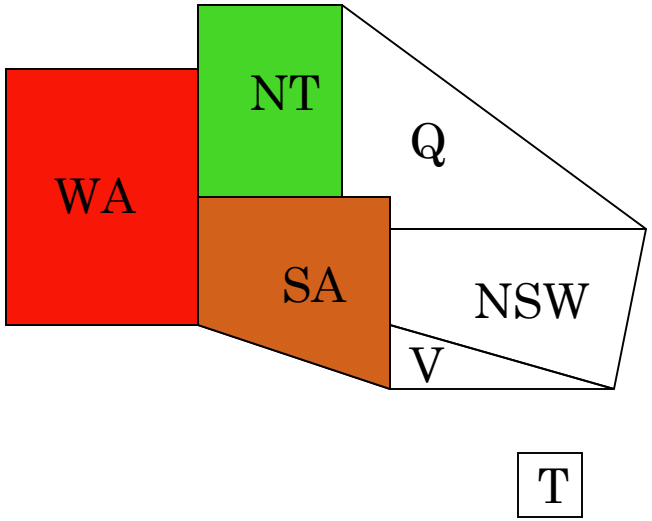
1. Which variable X should be assigned a value next?
2. In which order should its domain D be sorted?

Questions

1. Which variable X should be assigned a value next?
2. In which order should its domain D be sorted?
3. What are the implications of a partial assignment for yet unassigned variables? (→ Constraint Propagation)

Choice of Variable

- Map coloring

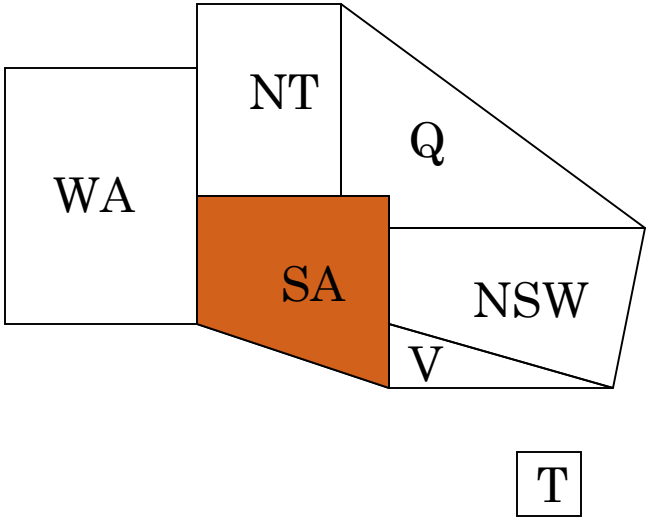


Choice of Variable

#1: Minimum Remaining Values (aka Most-constrained-variable heuristic or Fail First):

Select a variable with the fewest remaining values

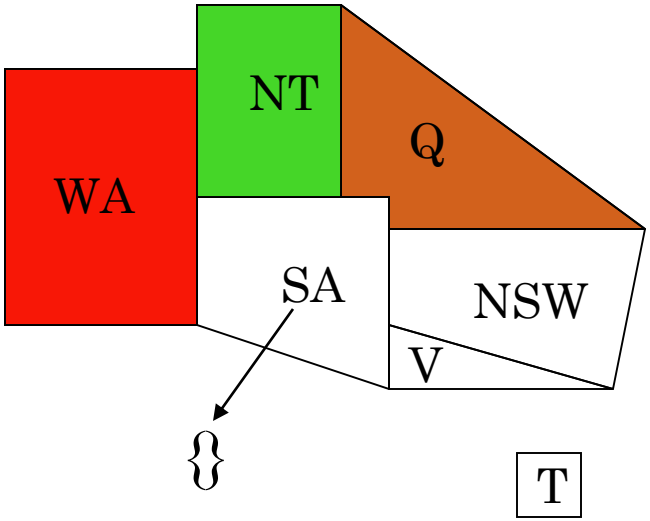
Choice of Variable



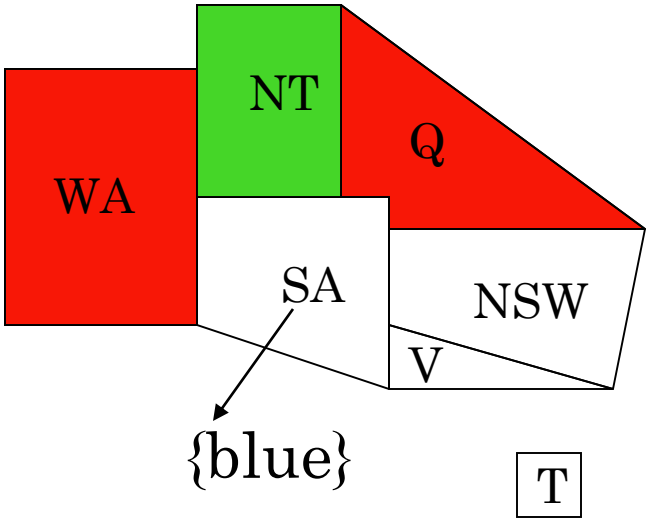
#2: Degree Heuristic:

Select the variable that is involved in the largest number of constraints on other unassigned variables

Choice of Value



Choice of Value



#3: Least-constraining-value heuristic:

Prefer the value that leaves the largest subset of legal values for other unassigned variables

Sudoku

- Variable Selection:
  - Minimum Remaining Values?
  - Degree
- Value Selection:
  - Least constraining value?
- Others?

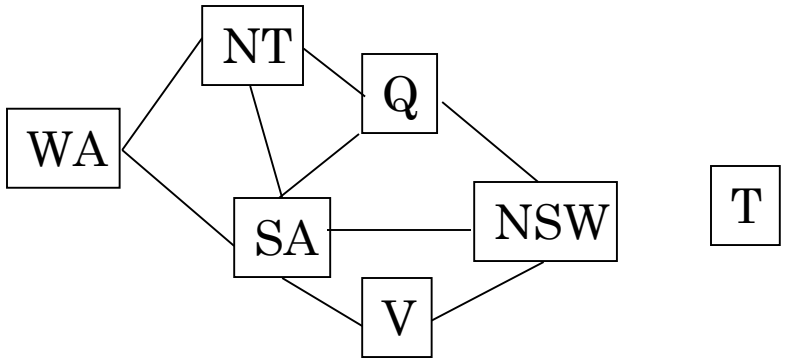
Constraint Propagation …

… is the process of determining how the possible values of one variable affect the possible values of other variables

Forward Checking

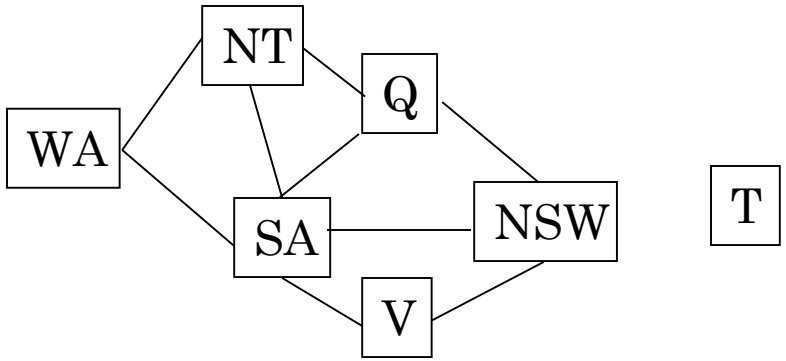
After a variable X is assigned a value v, look at each unassigned variable Y that is connected to X by a constraint and deletes from Y's domain any value that is inconsistent with v

Map Coloring: FC



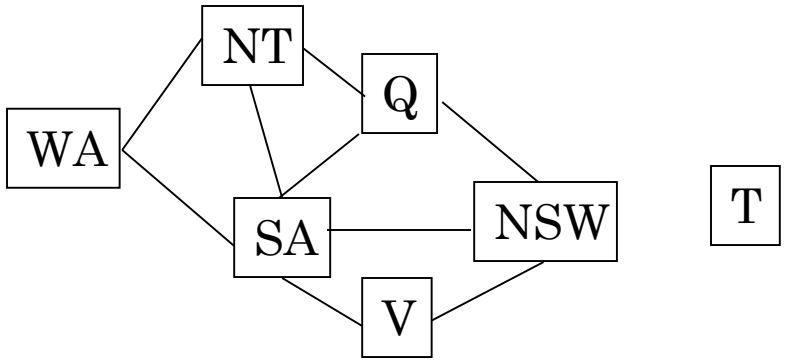
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB

Map Coloring: FC



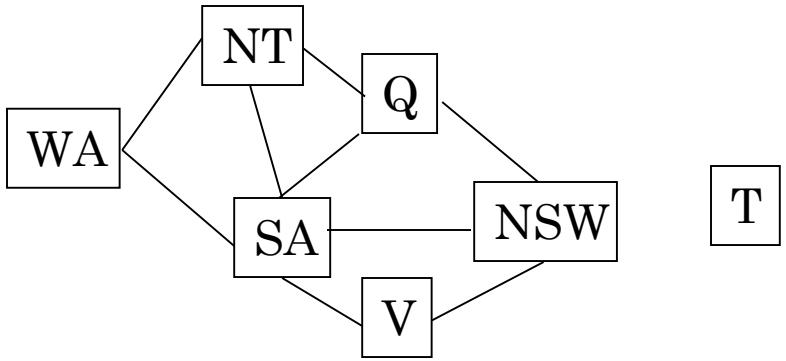
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
1: R	RGB	RGB	RGB	RGB	RGB	RGB

Map Coloring: FC



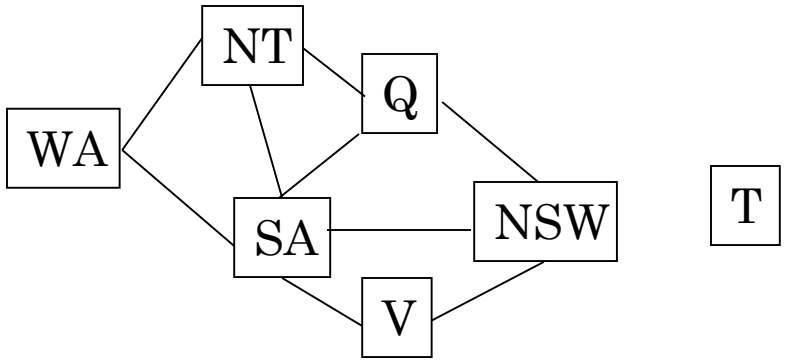
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
1: R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB

Map Coloring: FC



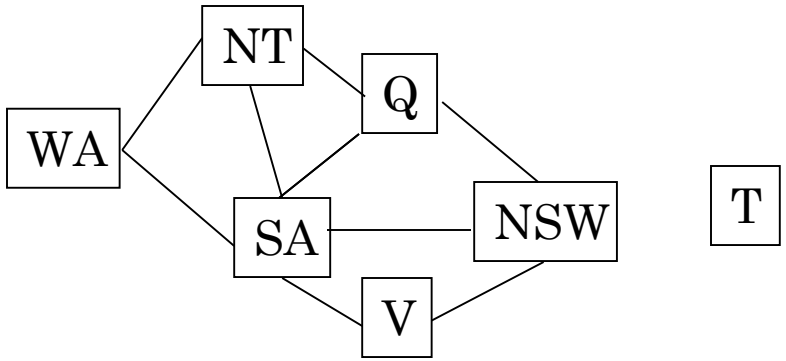
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB
R	~~R~~GB	2: G	RGB	RGB	~~R~~GB	RGB

Map Coloring:FC



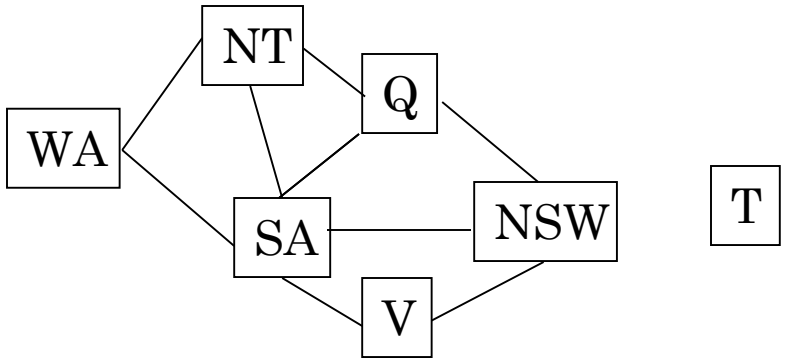
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB
R	~~RG~~B	2: G	~~R~~GB	RGB	~~RG~~B	RGB

Map Coloring: FC



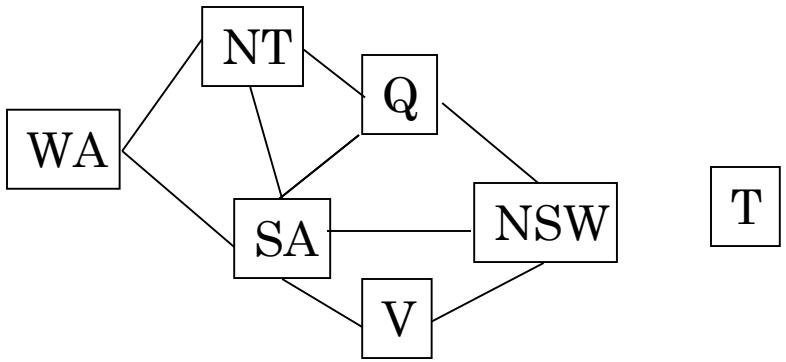
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB
R	~~RG~~B	G	~~R~~GB	RGB	~~RG~~B	RGB
R	~~RG~~B	G	~~R~~GB	3:B	~~RG~~B	RGB

Map Coloring: FC



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB
R	~~RGB~~	G	~~R~~GB	RGB	~~RGB~~	RGB
R	~~RGB~~	G	~~RGB~~	3:B	~~RGB~~	RGB

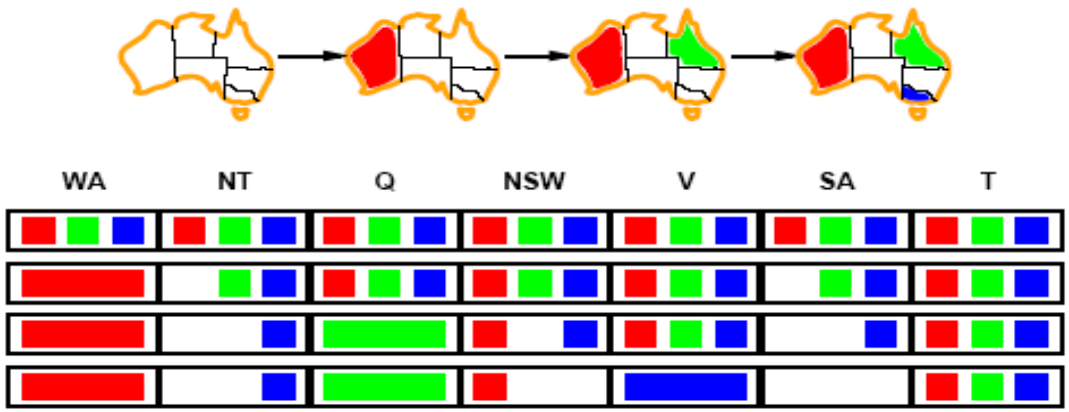
Other inconsistencies?



Impossible assignments that forward checking do not detect

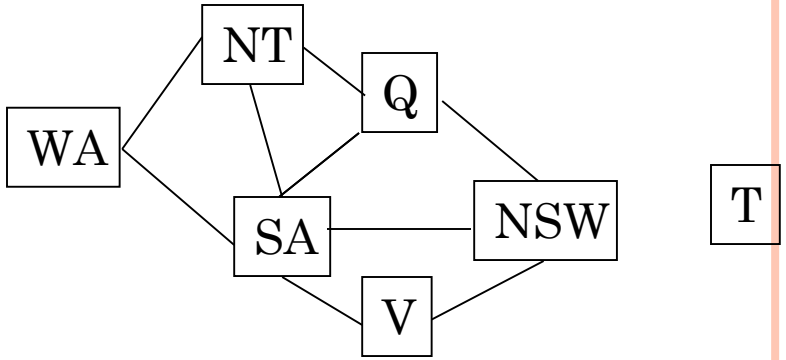
WA	NT	Q				
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB
R	~~RG~~B	G	~~R~~GB	RGB	~~RG~~B	RGB
R	~~RG~~B	G	~~RG~~B	3:B	~~RG~~B	RGB

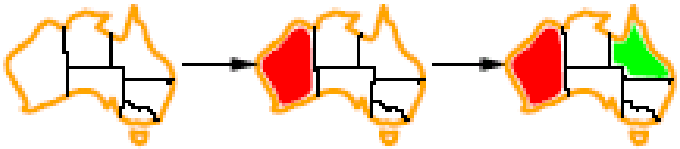
Constraint propagation

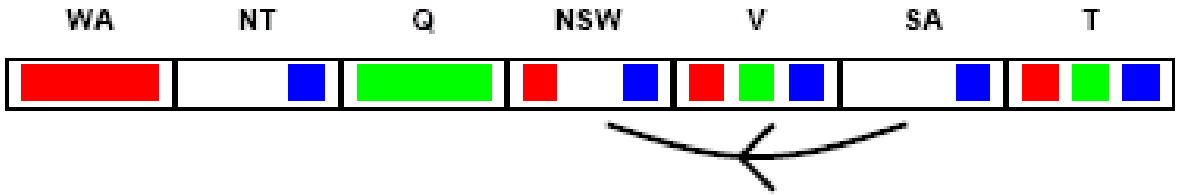


- Solving CSPs with combination of heuristics plus forward checking is more efficient than either approach alone.
- FC checking propagates information from assigned to unassigned variables but does not provide detection for all failures.
  - More advanced constraint propagation methods repeatedly enforces constraints locally

Arc consistency





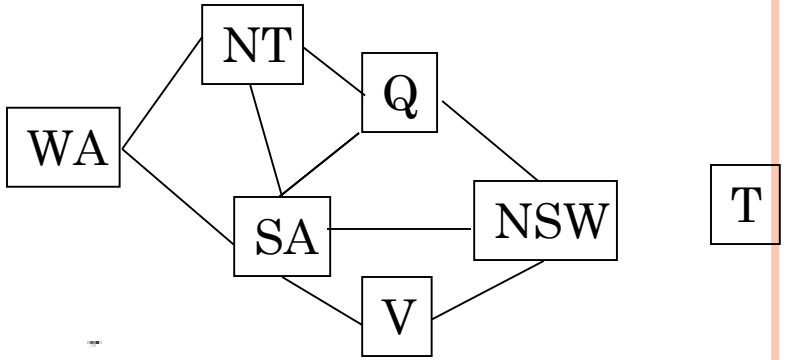


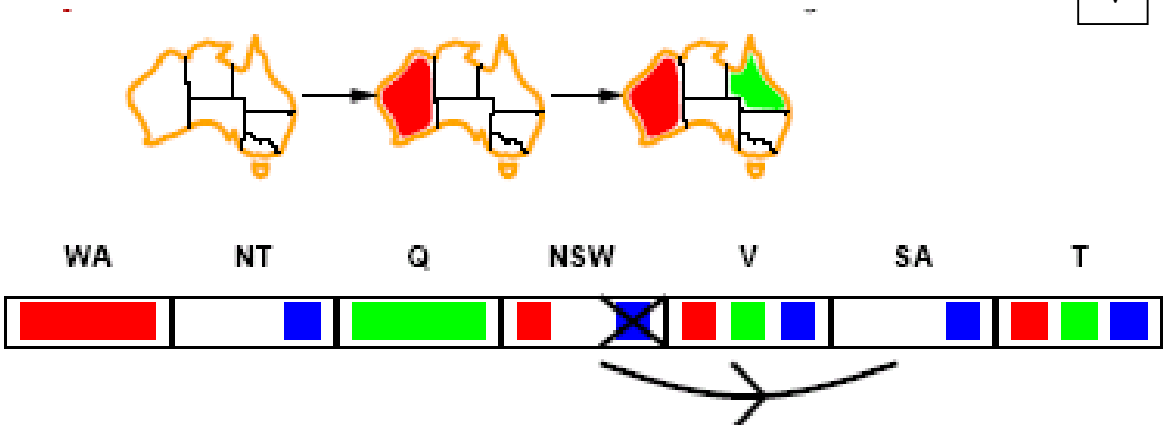
- $X \rightarrow Y$ is consistent iff

  for *every* value x of X there is some allowed y
- $SA \rightarrow NSW$ is consistent iff

  SA=*blue* and NSW=*red*

Arc consistency





- $X \rightarrow Y$ is consistent iff

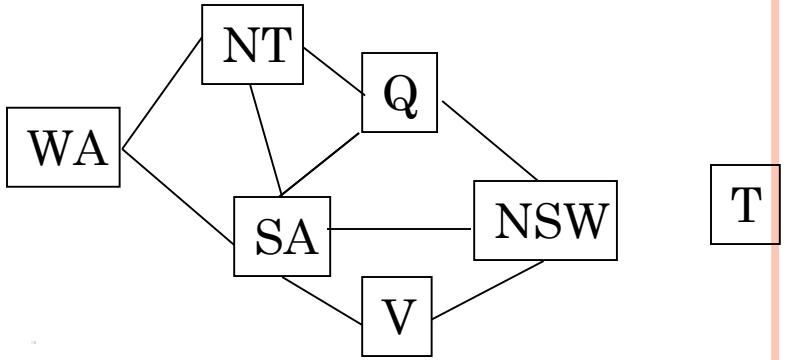
  for *every* value x of X there is some allowed y
- $NSW \rightarrow SA$ is consistent iff

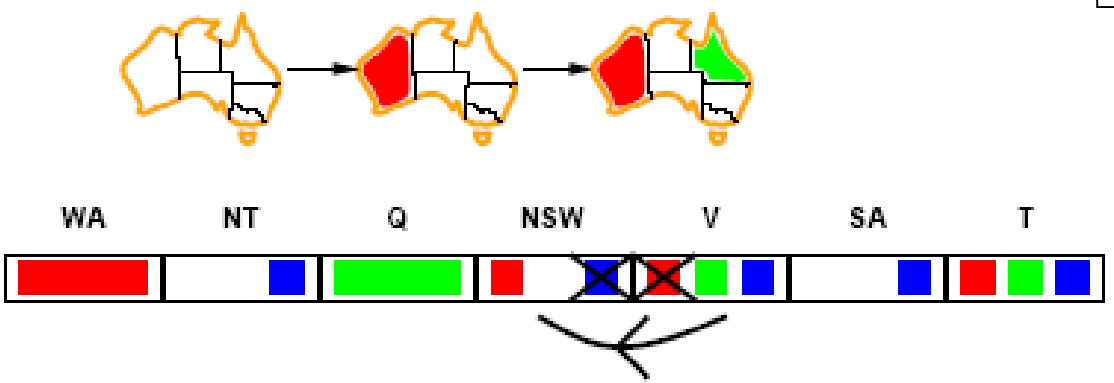
  NSW=red and *SA=blue*

  NSW=blue and SA=???

Arc can be made consistent by removing *blue* from *NSW*

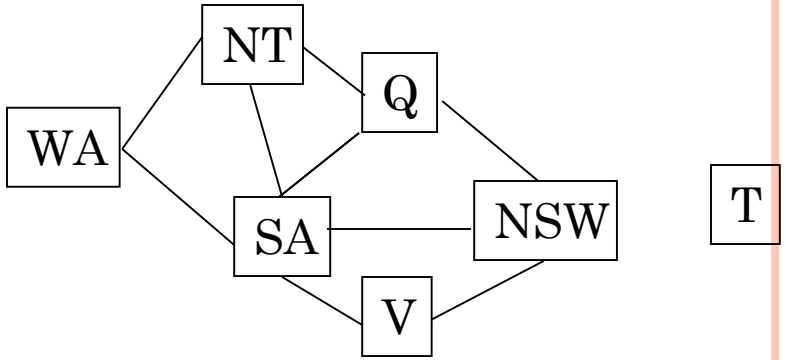
Arc consistency

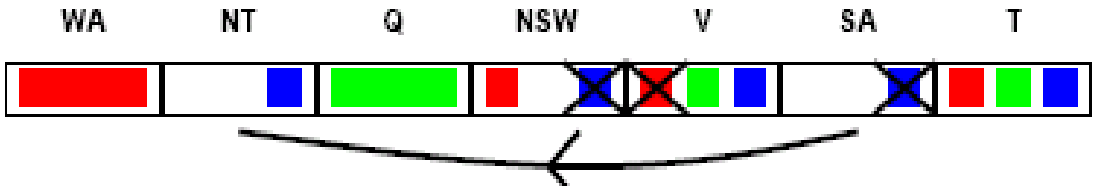




- Arc can be made consistent by removing *blue* from *NSW*
- RECHECK neighbors of *NSW*!!
  - Remove red from *V*

Arc consistency





- Arc can be made consistent by removing *blue* from *NSW*
- RECHECK neighbors of *NSW*!!
  - Remove red from *V*
- Arc consistency detects failure earlier than FC
- Can be run as a preprocessor or after each assignment.
  - Repeated until no inconsistency remains

Arc consistency algorithm

```
function AC-3(csp) return the CSP, possibly with reduced domains
    inputs: csp, a binary csp with variables {X1, X2, …, Xn}
    local variables: queue, a queue of arcs initially the arcs in csp

    while queue is not empty do
            (Xi, Xj) ← REMOVE-FIRST(queue)
            if REMOVE-INCONSISTENT-VALUES(Xi, Xj)  then
                    for each Xk in NEIGHBORS[Xi ] - Xj do
                    add (Xk, Xi) to queue

function REMOVE-INCONSISTENT-VALUES(Xi, Xj) return true iff we remove a value
    removed ←  false
    for each x in DOMAIN[Xi] do
            if no value y in DOMAIN[Xi] does (x, y) satisfy constraints between Xi and Xj
            then delete x from DOMAIN[Xi];
            removed ←  true
    return removed
```

K-CONSISTENCY

- Arc consistency does not detect all inconsistencies:
  - Partial assignment *{WA=red, NSW=red}* is inconsistent.
- Stronger forms of propagation can be defined using the notion of k-consistency.
- A CSP is k-consistent if for any set of k-1 variables and for any consistent assignment to those variables, a consistent value can always be assigned to any kth variable.
  - E.g. 1-consistency or node-consistency
  - E.g. 2-consistency or arc-consistency
  - E.g. 3-consistency or path-consistency

K-CONSISTENCY

- A graph is strongly k-consistent if
  - It is k-consistent and
  - Is also (k-1) consistent, (k-2) consistent, … all the way down to 1-consistent.

- This is ideal since a solution can be found in time $O(nd)$ instead of $O(n^2d^3)$
- YET *no free lunch*: any algorithm for establishing n-consistency must take time exponential in n, in the worst case.

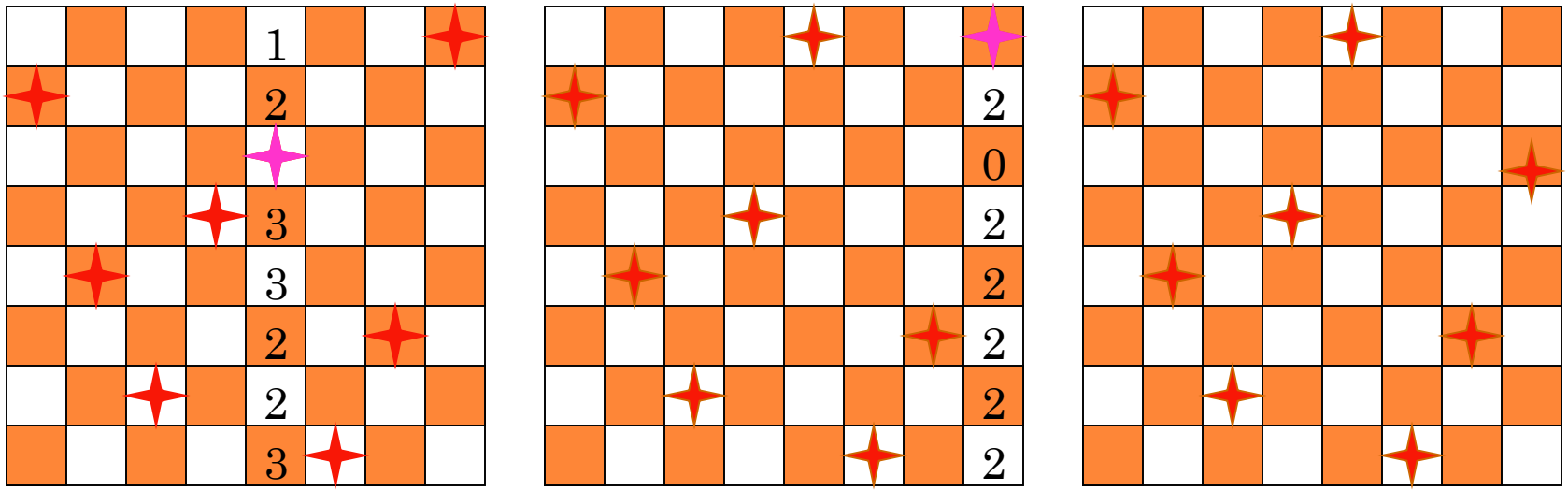
Further improvements

- Checking special constraints
  - Checking Alldif(…) constraint
    - *E.g. {WA=red, NSW=red}*
  - Checking Atmost(…) constraint
    - *Bounds propagation for larger value domains*
- Intelligent backtracking
  - Standard form is chronological backtracking i.e. try different value for preceding variable.
  - More intelligent, backtrack to conflict set.
    - Set of variables that caused the failure or set of previously assigned variables that are connected to X by constraints.
    - Backjumping moves back to most recent element of the conflict set.
    - Forward checking can be used to determine conflict set.

Local search for CSP

- Use complete-state representation
- For CSPs
  - allow states with unsatisfied constraints
  - operators **reassign** variable values
- Variable selection: randomly select any conflicted variable
- Value selection: *min-conflicts heuristic*
  - Select new value that results in a minimum number of conflicts with the other variables

Local Search for CSP



Pick initial complete assignment (at random)
Repeat

- Pick a conflicted variable var (at random)
- Set the new value of var to minimize the number of conflicts
- If the new assignment is not conflicting then return it

(min-conflicts heuristics)

Local search for CSP

function MIN-CONFLICTS(*csp, max_steps*) **return** solution or failure

inputs: *csp*, a constraint satisfaction problem

max_steps, the number of steps allowed before giving up

current ← an initial complete assignment for *csp*

for *i* = 1 to *max_steps* **do**

if *current* is a solution for *csp* then return *current*

var ← a randomly chosen, conflicted variable from VARIABLES[*csp*]

value ← the value *v* for *var* that minimize CONFLICTS(*var,v,current,csp*)

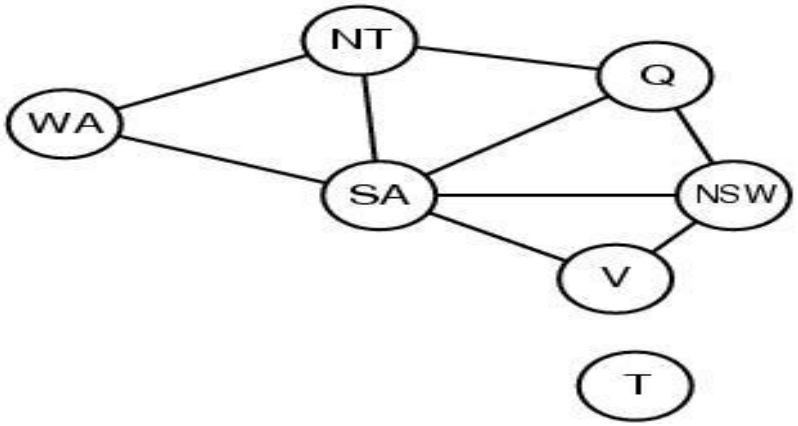
set *var* = *value* in *current*

return *failure*

Remark

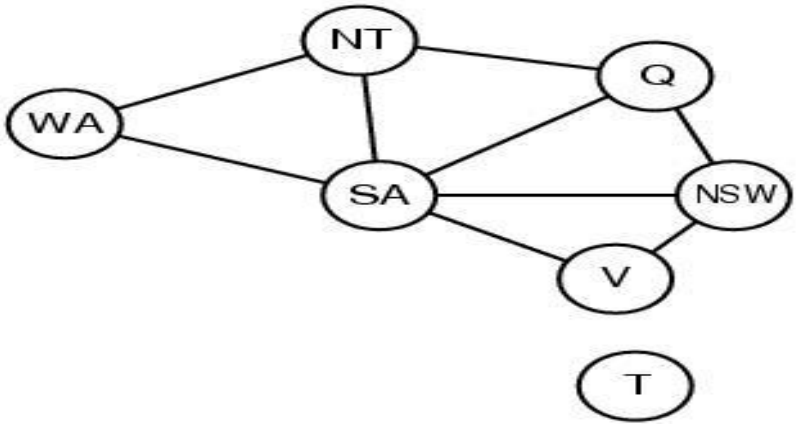
- Local search with min-conflict heuristic works extremely well for million-queen problems
- The reason: Solutions are densely distributed in the $O(n^n)$ space, which means that on the average a solution is a few steps away from a randomly picked assignment

Problem structure



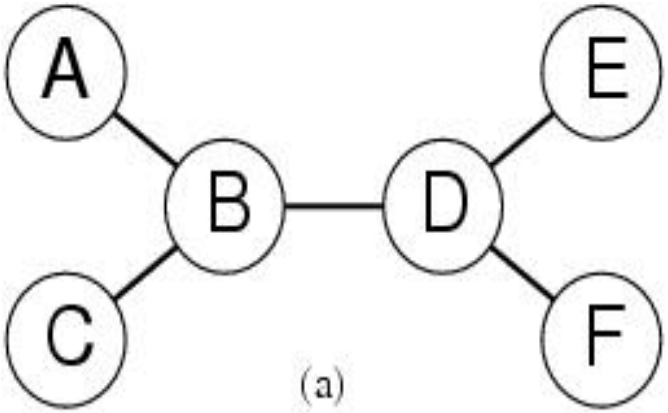
- *How can the problem structure help to find a solution quickly?*
- Subproblem identification is important:
  - Coloring Tasmania and mainland are independent subproblems
  - Identifiable as connected components of constrained graph.
- Improves performance

Problem structure



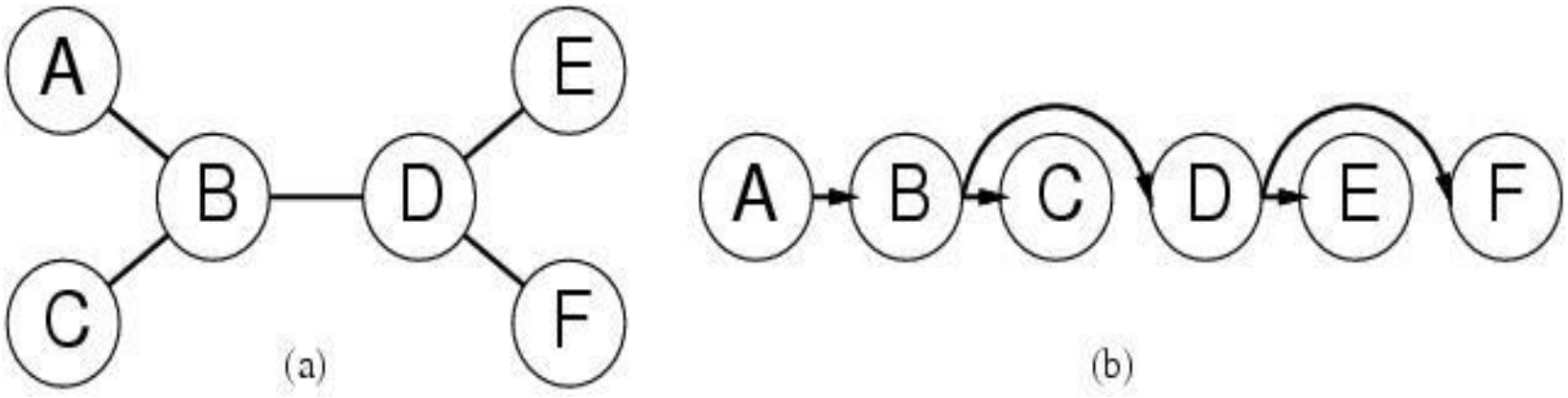
- Suppose each problem has c variables out of a total of n.
- Worst case solution cost is $O(n/c\ d^c)$, i.e. linear in n
  - Instead of $O(d^n)$, exponential in n
- E.g. $n= 80,\ c= 20,\ d=2$
  - 2^{80} = 4 billion years at 1 million nodes/sec.
  - $4 * 2^{20}$= .4 second at 1 million nodes/sec

Tree-structured CSPs



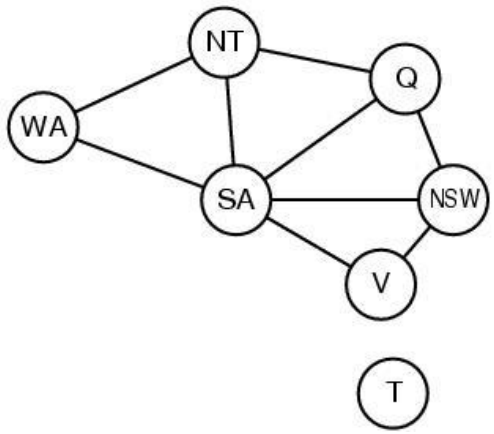
- Theorem: if the constraint graph has no loops then CSP can be solved in $O(nd^2)$ time
- Compare difference with general CSP, where worst case is $O(d^n)$

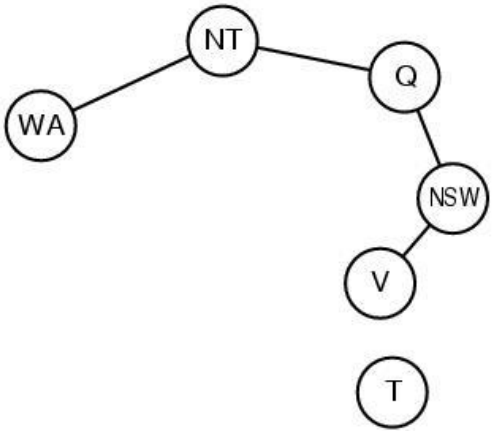
Tree-structured CSPs



- In most cases subproblems of a CSP are connected as a tree
- Any tree-structured CSP can be solved in time linear in the number of variables.
  - Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering. (label var from X_1 to X_n)
  - For j from n down to 2, apply REMOVE-INCONSISTENT-VALUES(Parent(X_j),X_j)
  - For j from 1 to n assign X_j consistently with Parent(X_j)

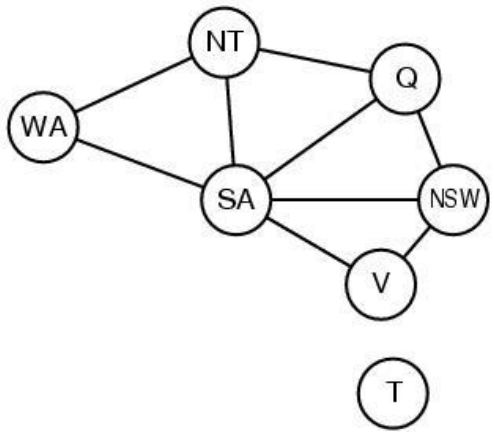
Nearly tree-structured CSPs

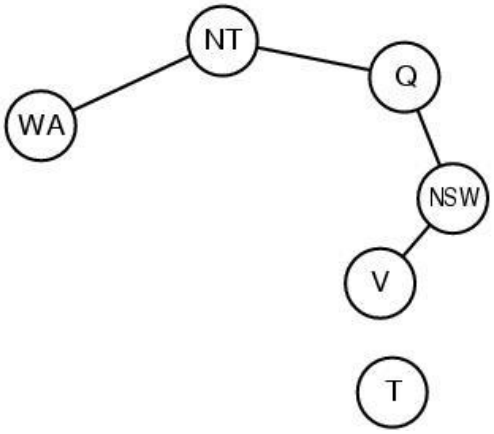




- *Can more general constraint graphs be reduced to trees?*
- Two approaches:
  - Remove certain nodes
  - Collapse certain nodes

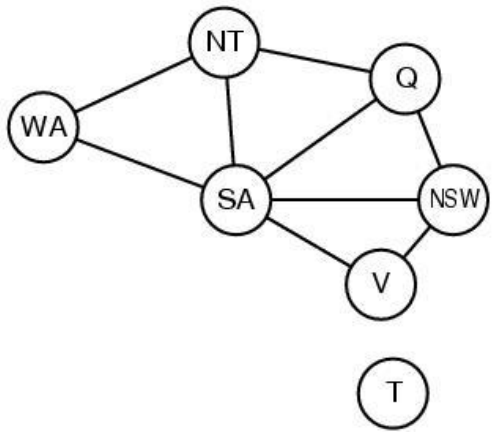
Nearly tree-structured CSPs

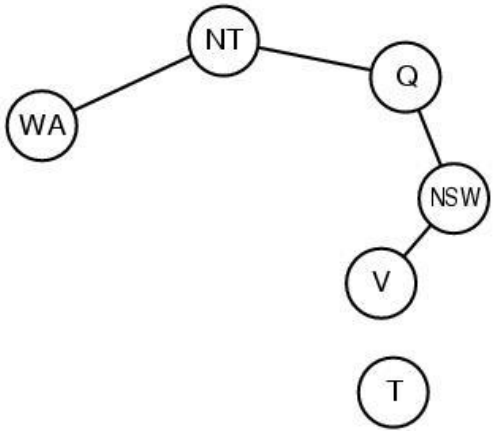




- Idea: assign values to some variables so that the remaining variables form a tree.
- Assume that we assign *{SA=x}* ← *cycle cutset*
  - And remove any values from the other variables that are inconsistent.
  - The selected value for SA could be the wrong one so we have to try all of them

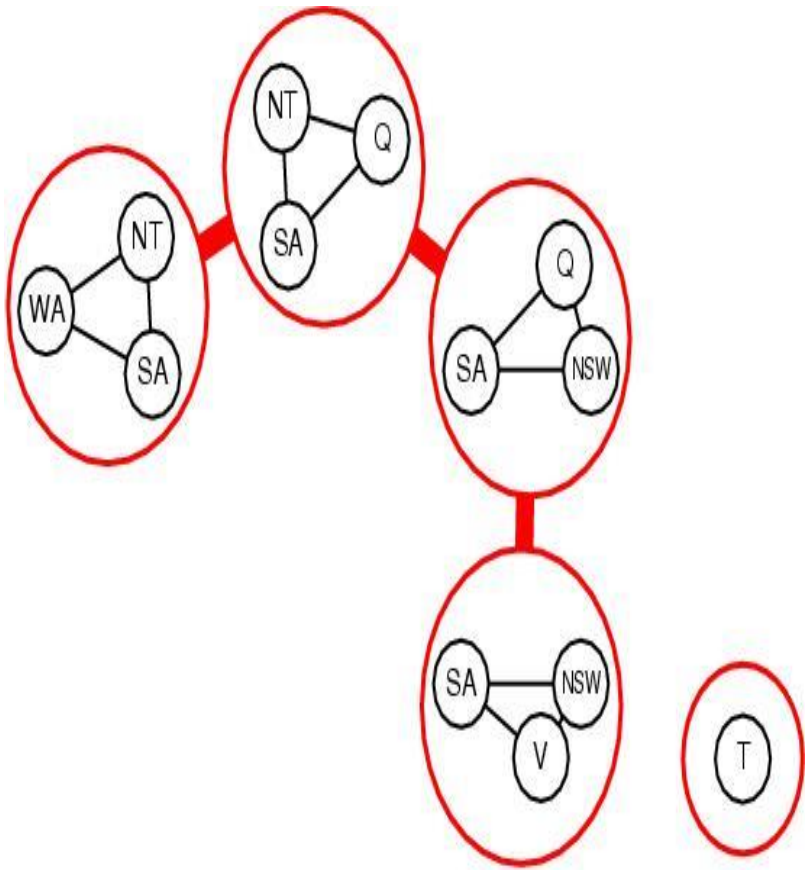
Nearly tree-structured CSPs





- This approach is worthwhile if cycle cutset is small.
- Finding the smallest cycle cutset is NP-hard
  - Approximation algorithms exist
- This approach is called *cutset conditioning*.

Nearly tree-structured CSPs

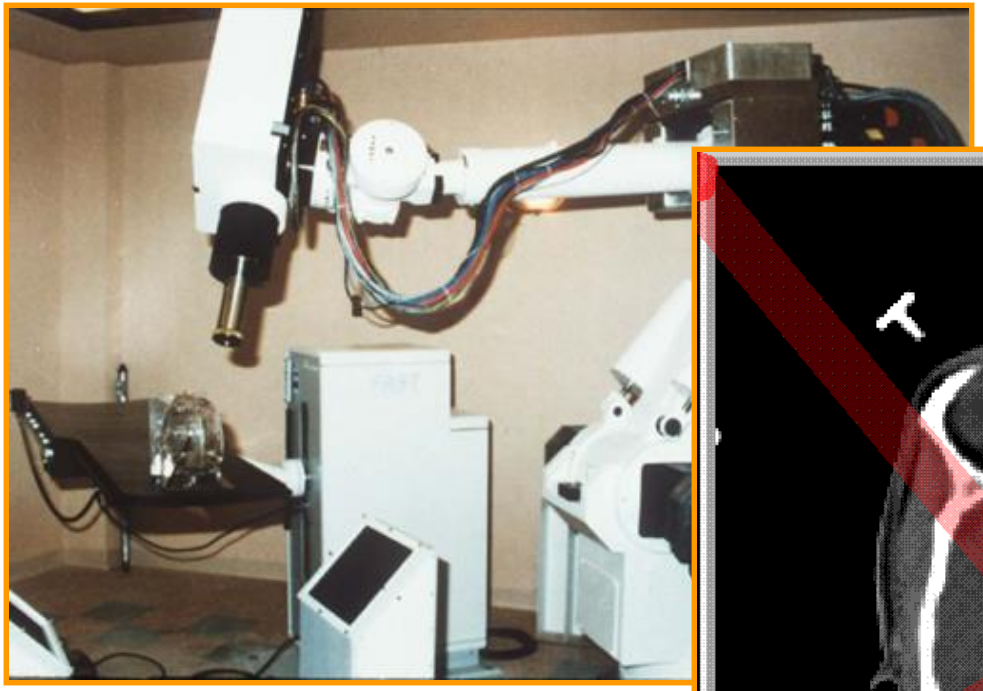


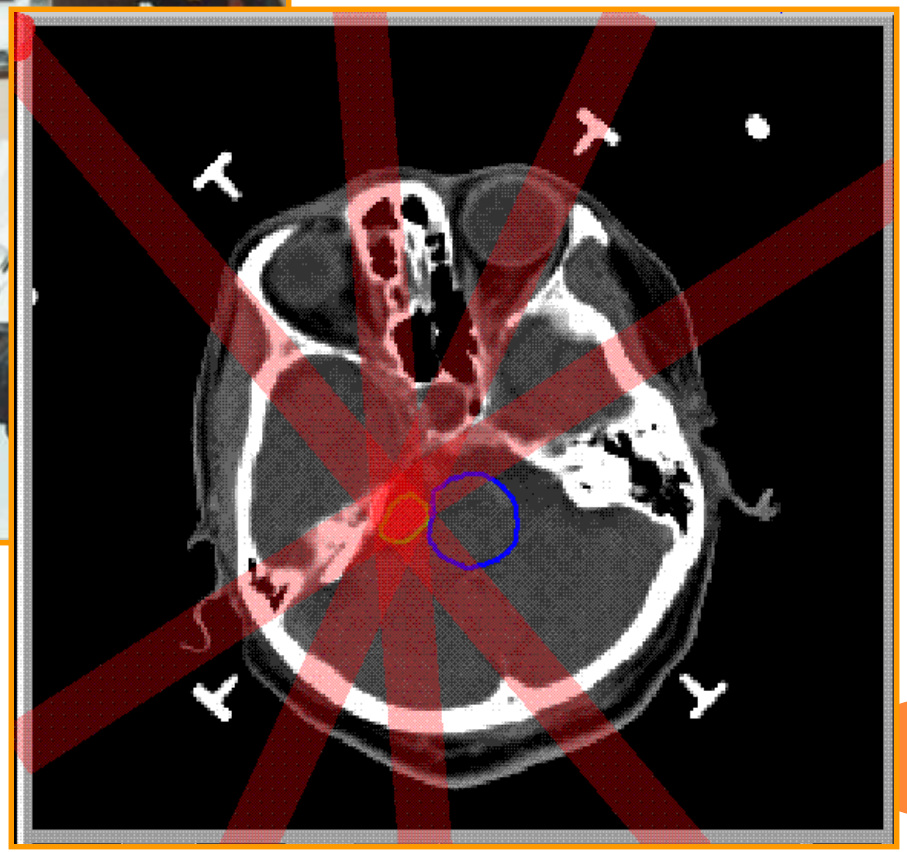
- Tree decomposition of the constraint graph in a set of connected subproblems.
- Each subproblem is solved independently
- Resulting solutions are combined.
- Necessary requirements:
  - Every variable appears in ar least one of the subproblems.
  - If two variables are connected in the original problem, they must appear together in at least one subproblem.
  - If a variable appears in two subproblems, it must appear in each node on the path.

Applications

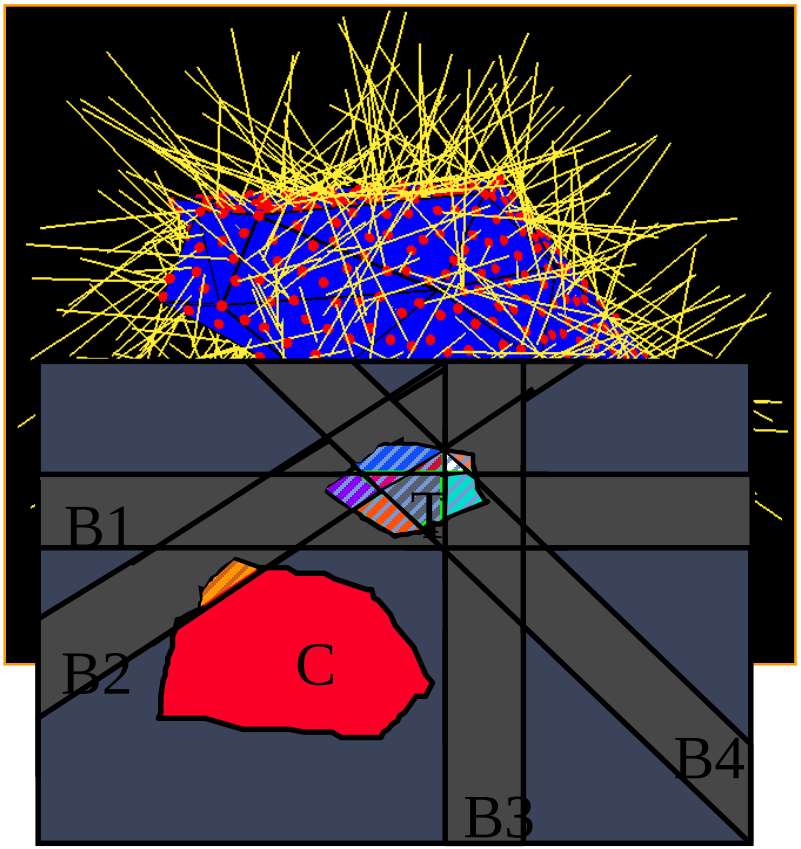
- CSP techniques allow solving very complex problems
- Numerous applications, e.g.:
  - Crew assignments to flights
  - Management of transportation fleet
  - Flight/rail schedules
  - Task scheduling in port operations
  - Design
  - Brain surgery
- See **`www.ilog.com`**

Stereotaxic Brain Surgery





Stereotaxic Brain Surgery



- 2000 < Tumor < 2200
  2000 < B2 + B4 < 2200
  2000 < B4 < 2200
  2000 < B3 + B4 < 2200
  2000 < B3 < 2200
  2000 < B1 + B3 + B4 < 2200
  2000 < B1 + B4 < 2200
  2000 < B1 + B2 + B4 < 2200
  2000 < B1 < 2200
  2000 < B1 + B2 < 2200

- 0 < Critical < 500
  0 < B2 < 500

→ Constraint Programming

“Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it.”

Eugene C. Freuder, Constraints, April 1997

When to Use CSP Techniques?

- When the problem can be expressed by a set of variables with constraints on their values
- When constraints are relatively simple (e.g., binary)
- When constraints propagate well (AC3 eliminates many values)
- Local Search: when the solutions are "densely" distributed in the space of possible assignments

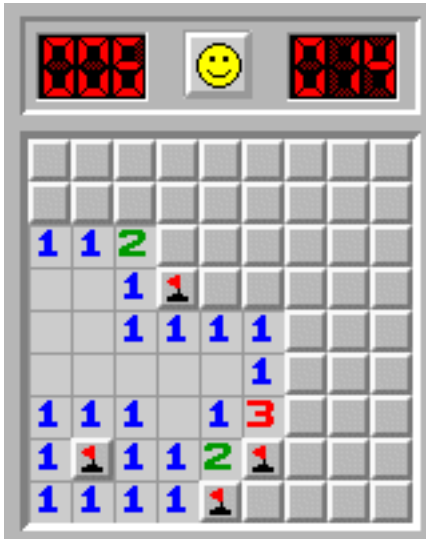
Summary

- CSPs are a special kind of problem: states defined by values of a fixed set of variables, goal test defined by constraints on variable values
- Backtracking=depth-first search with one variable assigned per node
- Variable ordering and value selection heuristics help significantly
- Forward checking prevents assignments that lead to failure.
- Constraint propagation does additional work to constrain values and detect inconsistencies.
- The CSP representation allows analysis of problem structure.
- Tree structured CSPs can be solved in linear time.
- Local Search, e.g., min-conflicts, is often effective in practice.

CSP: Examples

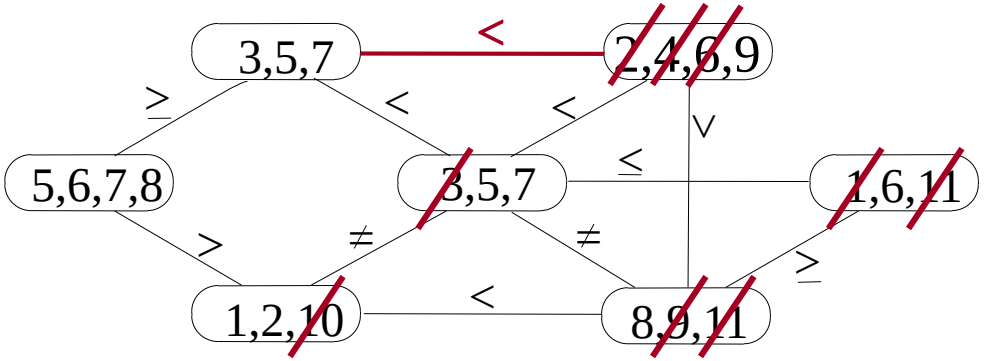
- Industrial applications: scheduling, resource allocation, product configuration, etc.
- AI: Logic inference, temporal reasoning, NLP, etc.
- Puzzles: Sudoku & Minesweeper

							1	3
			7				6	
			5		9			
			4			9		
1		6						
						2		
7	4						5	
	8					4		
				1				



Constraint propagation

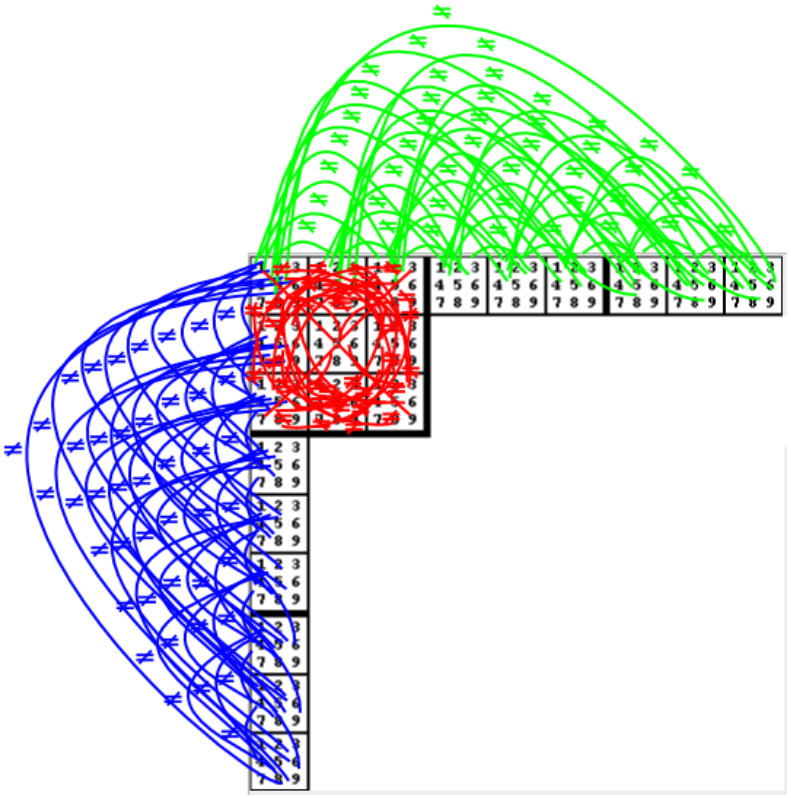
- Removes from the problem values (or combinations of values) that are inconsistent with the constraints

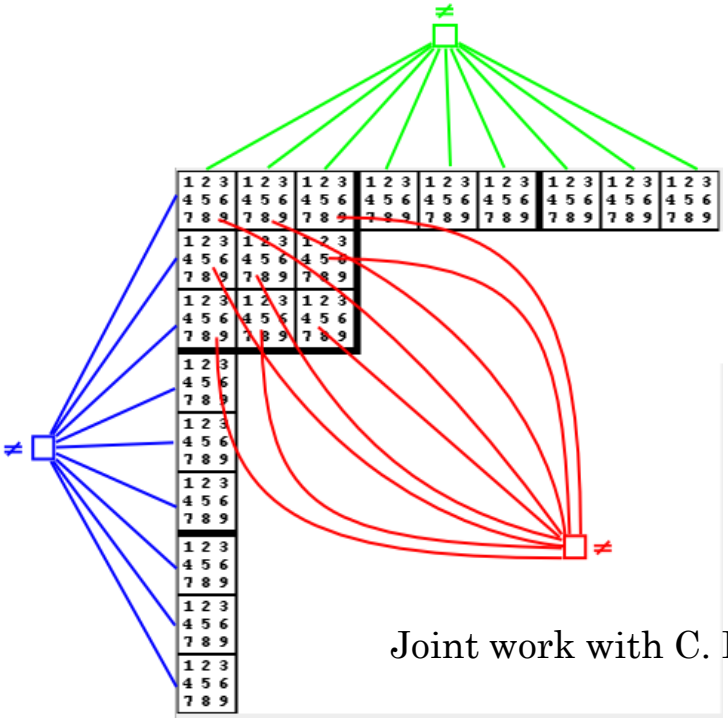


- Does not eliminate any solution

Sudoku as a CSP

- Each cell is a variable (decision) with the domain [1..9] (choices)
- Two models: Binary, 810 AllDiff binary constraints
  Non-binary, 27 AllDiff constraints of arity 9

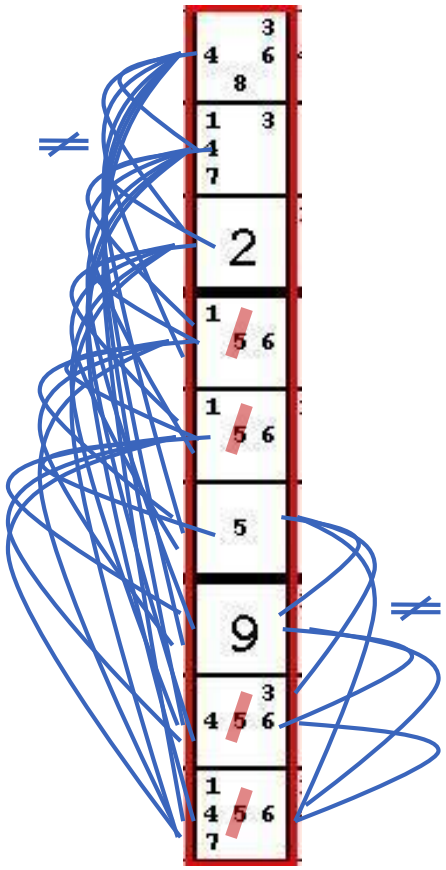


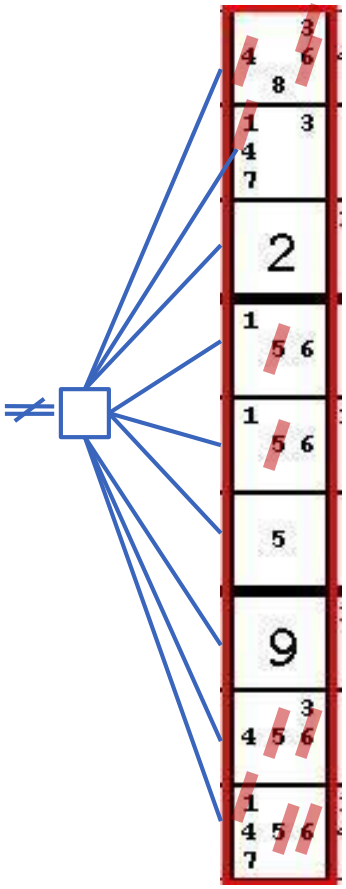


Joint work with C. Reeson

Propagation algorithms: demo

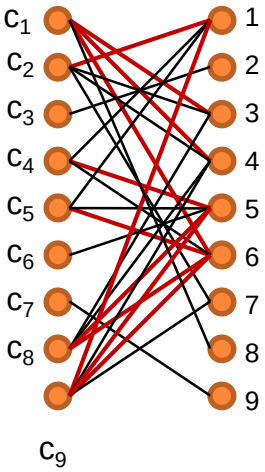
- Arc Consistency (AC)
- Generalized AC (GAC)





GAC on AllDiff [Régin, 94]

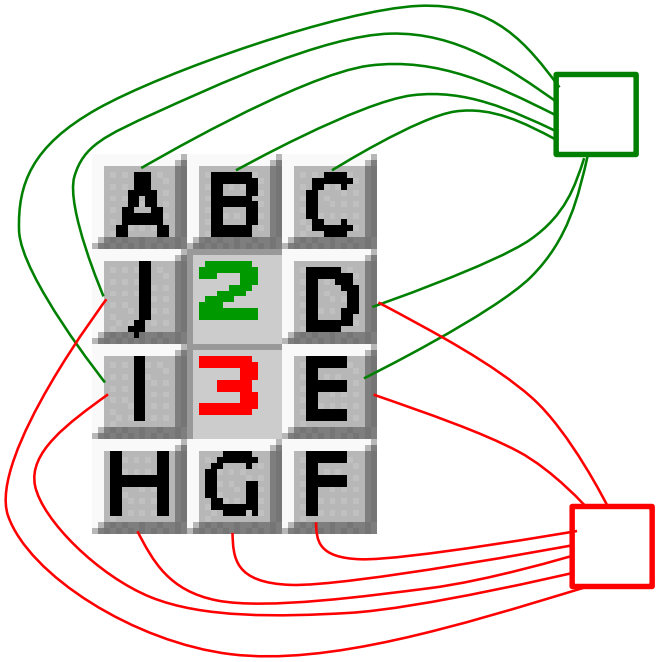
- Arcs that do not appear in any matching that saturates the variables correspond to variable-value pairs that cannot appear in any solution
- GAC on AllDiff is poly time



Minesweeper as a CSP

DEMO

- Variables are the cells
- Domains are {0,1} (i.e., safe or mined)
- One constraint for each cell with a number (arity 1...8)



Exactly two mines:

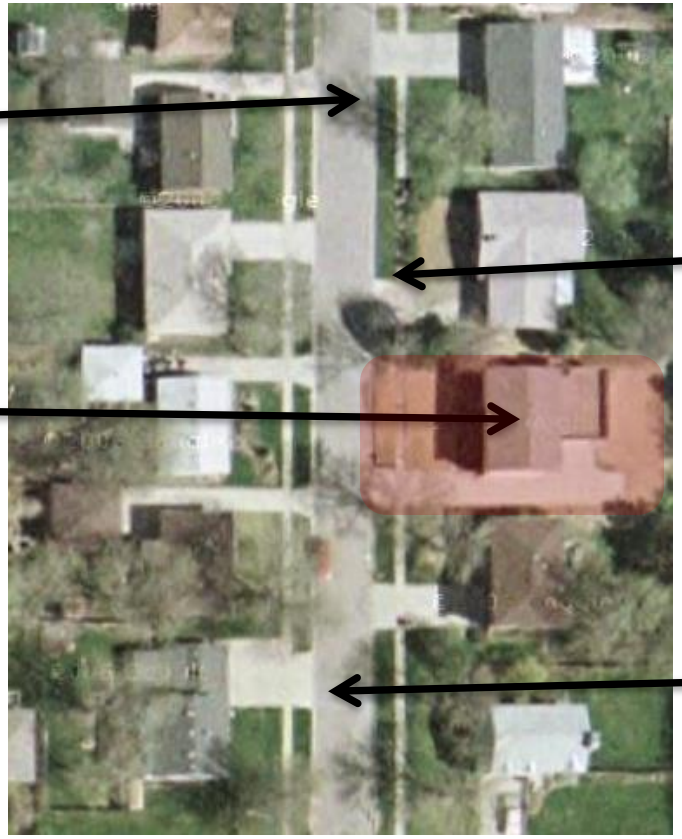
0000011
0000101
0000110, etc.

Exactly three mines:

0000111
0001101
0001110, etc.

Joint work with R. Woodward, K. Bayer & J. Snyder

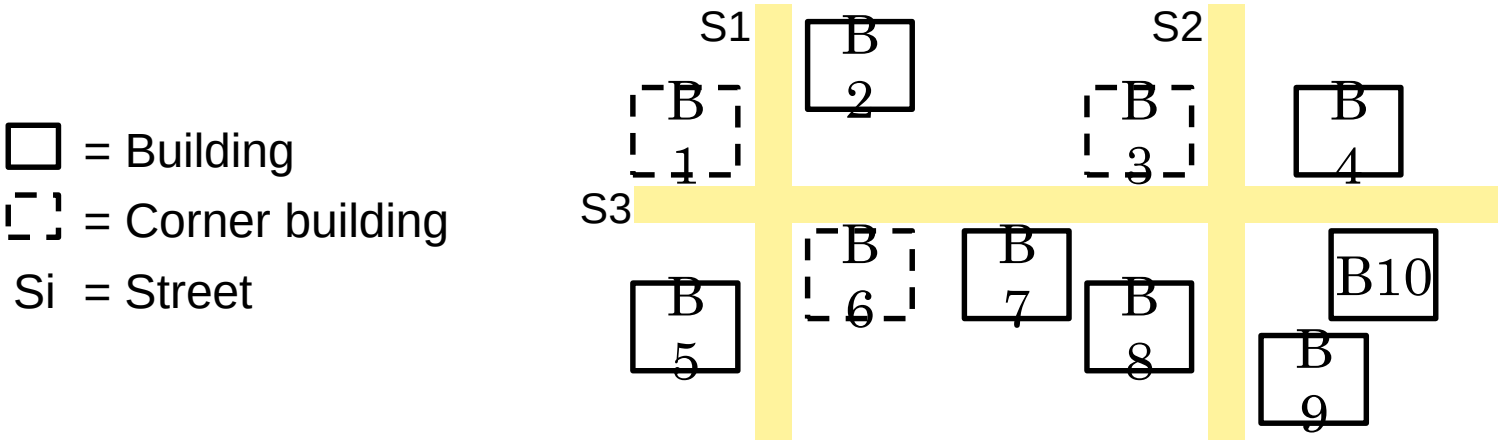
Geospatial reasoning



Joint work with K. Bayer, M. Michalowski & C.A. Knoblock (USC)

Building Identification (BID) problem

- Layout: streets and buildings

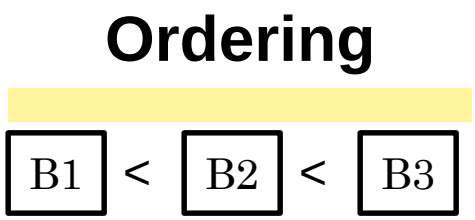


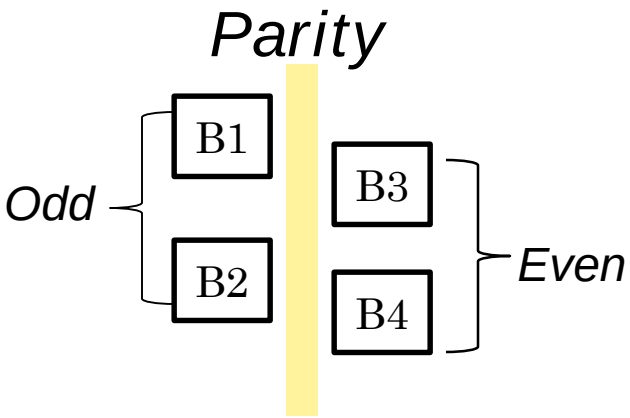
- Phone book
  - Complete/incomplete
  - Assumption: all addresses in phone book correspond to a building in the layout

S1#1, S1#4, S1#8,
S2#7, S2#8, S3#1,
S3#2, S3#3, S3#15,
…

Basic (address numbering) rules

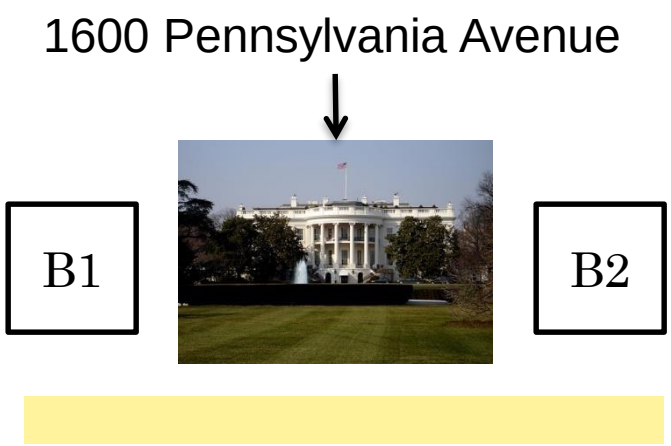
- No two buildings can have the same address
- Ordering
  - Numbers increase/decrease along a street
- Parity
  - Numbers on a given side of a street are odd/even



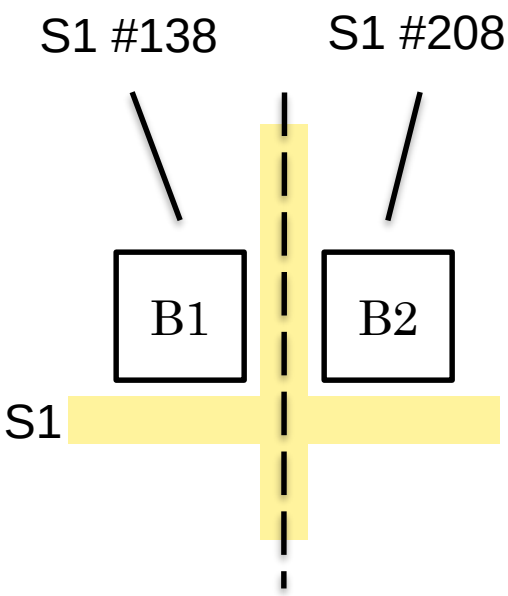


Additional information

Landmarks

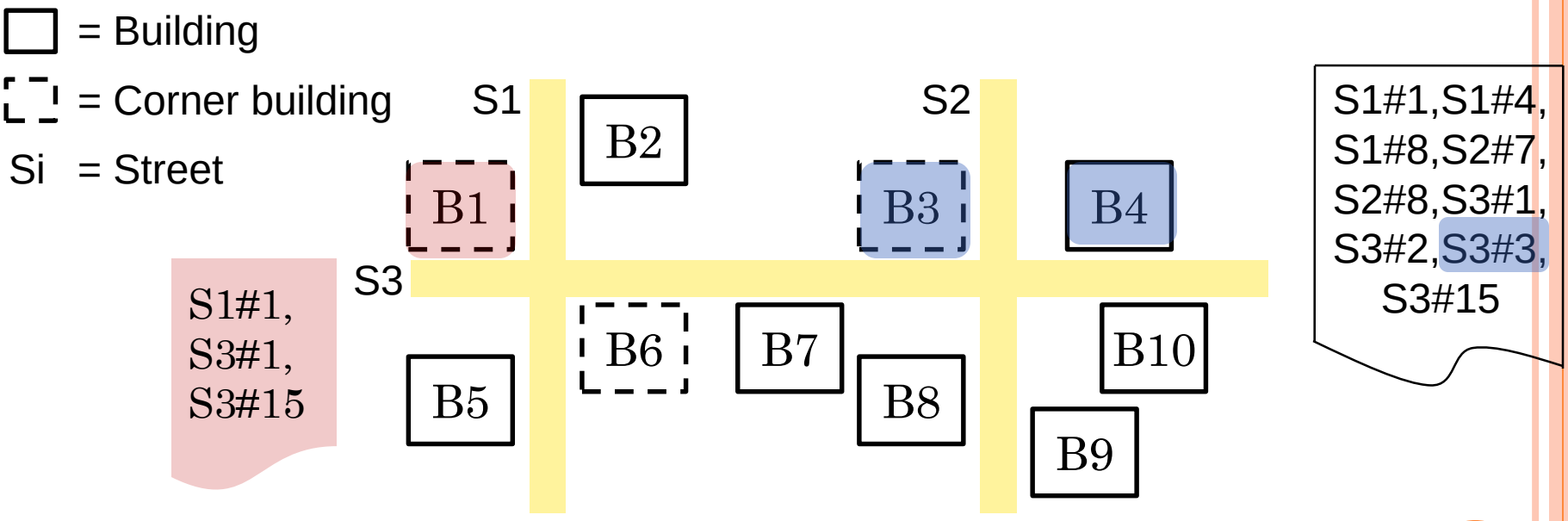


Gridlines



Query

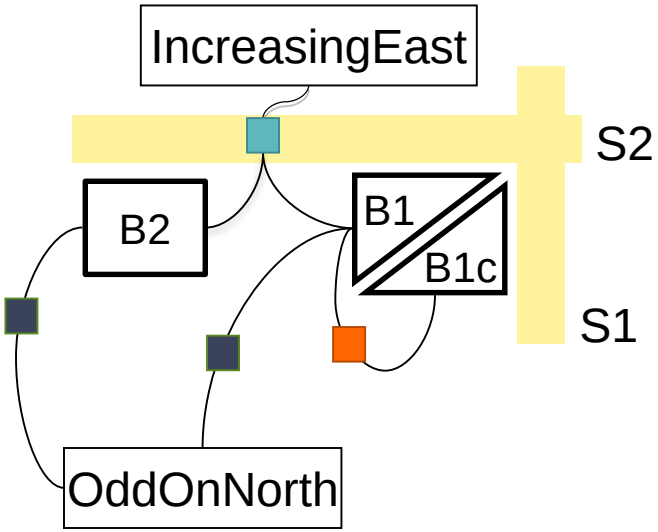
1. Given an address, what buildings could it be?
2. Given a building, what addresses could it have?

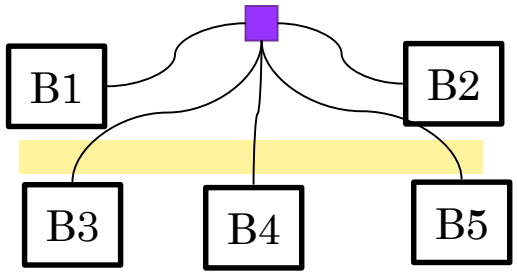


CSP MODEL

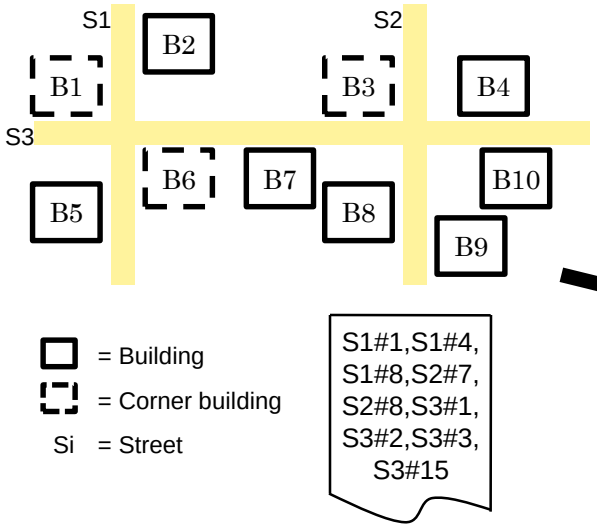
- **Parity constraints**
- **Ordering constraints**
- **Corner constraints**

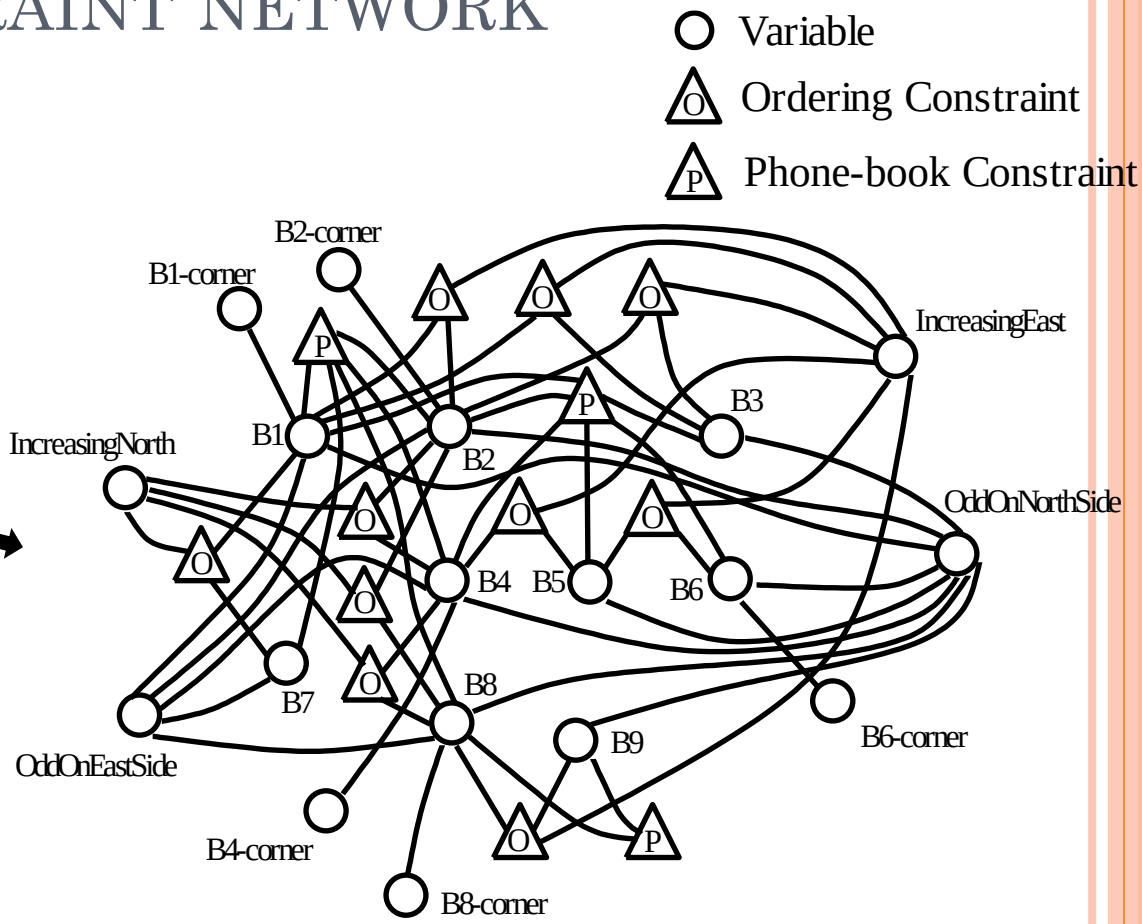
- **Phone-book constraints**
- Optional: grid constraints





Example constraint network





GTAAP: Task

- Hiring & managing GTAs as instructors + graders
  - Given
    - A set of courses
    - A set of graduate teaching assistants
    - A set of constraints that specify allowable assignments
  - Find a consistent & satisfactory assignment
    - *Consistent*: assignment breaks no (hard) constraints
    - *Satisfactory*: assignment maximizes
      1. number of courses covered
      2. happiness of the GTAs
- Often, number of hired GTAs is insufficient

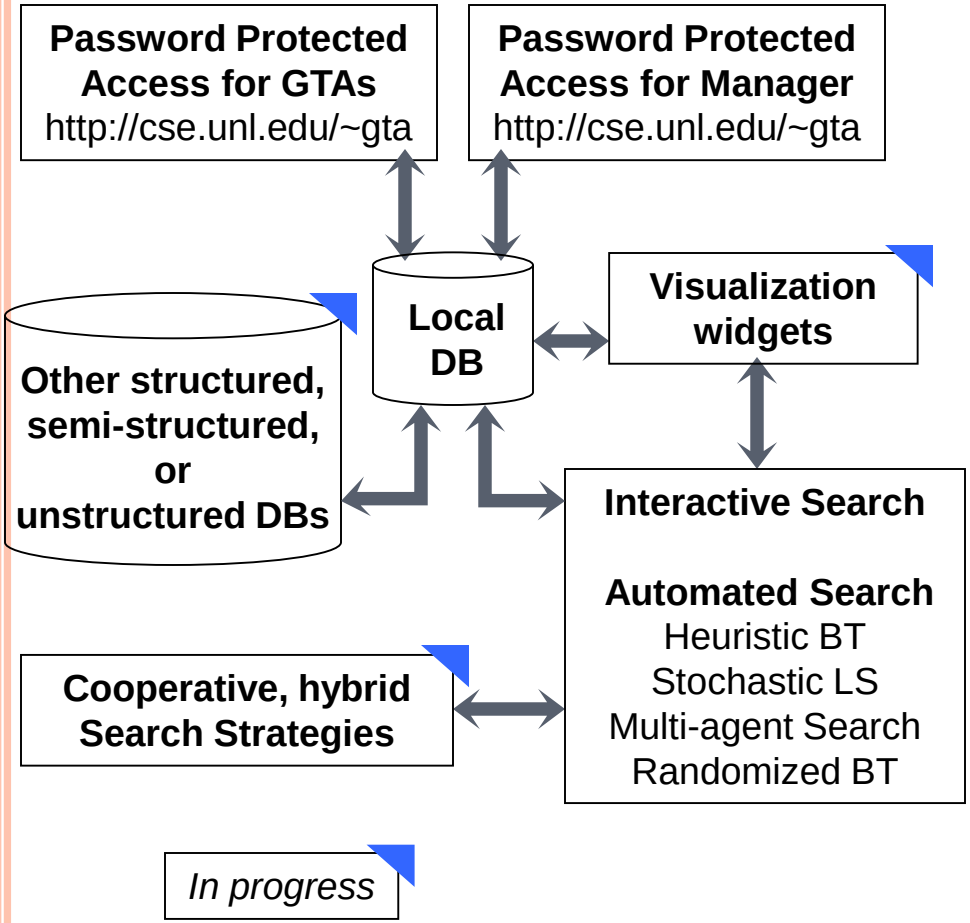
Motivation

- **Context**
  - "Most difficult duty of a department chair" [Reichenbach, 2000]
  - Assignments done manually, countless reviews, persistent inconsistencies
  - Unhappy instructors, unhappy GTAs, unhappy students
- **Observation**
  - Computers are good at maintaining consistency
  - Humans are good at balancing tradeoffs
- **Our solution**
  - An online, constraint-based system
  - With interactive & automated search mechanisms

Outline

- Task & Motivation
- System Architecture & Interfaces
- Scientific aspects
  - Problem Modeling
  - Problem Solving
  - Comparing & Characterizing Solvers
- Motivation revisited & Conclusions

System Architecture



1. Web-interface for applicants
2. Web-interface for manager
   - ☐ View / edit GTA records
   - ☐ Setup classes
   - ☐ Specify constraints
   - ☐ Enforce pre-assignments
3. A local relational database
   - ☐ Graphical selective queries
4. Drivers for
   - ☐ Interactive assignments
   - ☐ Automated search algorithms

GTA interface: Preference Specification

GTAP (DEMO)

http://csce.unl.edu/~gtademo/gta/gta_classes_frames.php?semester=2&ye

Print page | Main page | Academic record | Teaching preference | Login info | Course descriptions | Survey | Logout

Preference for Demo Student

Spring 2004

Undo changes | Submit

☑ I am applying to be a TA for this semester.

Class	Section	Description	Time	Days	Enrolled?	Preference	Justification for "0"
CSCE 101	001	Computer Science Fundamentals	1400 - 1515	T R	☐	1) Avoid if possible	
CSCE 101	001T	Computer Science Fundamentals	1400 - 1515	T R	☐	2) Able to handle	
CSCE 101L	001	Lab	1030 - 1220	R	☐	3) Qualified	
CSCE 101L	002	Lab	1330 - 1520	W	☐	4) Favorite	
CSCE 105	150	Problem Solving w/Computers	1330 - 1420	M W F	☐	5) Best choice	
CSCE 105	151	Lab	1530 - 1620	M	☑	0) Cannot handle	Enrolled
CSCE 105	152	Lab	1130 - 1220	W	☐	0) Cannot handle	time conflict
CSCE 105	153	Lab	1430 - 1520	R	☐	3) Qualified	
CSCE 150	150	Intro to Computer Programming	0800 - 0915	T R	☐	3) Qualified	
CSCE 150	151	Lab	1230 - 1320	M	☐	3) Qualified	
CSCE 150	152	Lab	1230 - 1320	T	☐	3) Qualified	
CSCE 150	153	Lab	1530 - 1620	W	☐	3) Qualified	

5) Best choice
You would love to teach this class

4) Favorite
You are very capable of teaching this class

3) Qualified
You don't mind this assignment.

2) Able to handle
You will survive ...

1) Avoid if possible
Your last choice before giving up your assistantship.

0) Cannot handle
You are either enrolled in the class or not able to teach it. Either way, you must produce a legitimate reason.

Done

Manager interface: TA Hiring & Load

GTAP manager interface (DEMO)

http://csce.unl.edu/~gtademo/manager/

Spring 2004 | GTAs | Classes | Interactive selection | Search | Sys Admin | Logout

About GTAs

All GTAs in system

Employment history

GTA Evaluations

GTAs in semester

Applied | Hired

Filter

Preference ranking

Applied | Hired

Edit record

View preferences

Delete GTA

Add GTA

Hired

GTAs in semester: Applied

Customize View | Print Page

Hire	Capacity	Name	Semesters supported as		Acad. record		Preference		Email
All			Spring 2004		All		All		All
Save	Save		GRA	GTA	Print	Updated	Print	Updated	Email
☑	1.00	Alescio, Garth	4	1	☐	04/03/30	☐	03/11/27	☐
☑	0.33	Arvay, Mellie	1	1	☐	04/04/30	☐	03/11/05	☐
☐	1.00	Baymon, Reagan	0	1	☐	03/11/10	☐	03/11/10	☐
☑	0.00	Betters, Lisandra	0	2	☐	04/04/11	☐	03/11/18	☐
☑	1.00	Beuchler, Sherron	0	2	☐	03/08/21	☐	03/11/16	☐
☑	0.66	Billett, Roger	6	3	☐	03/11/04	☐	03/11/21	☐
☑	0.66	Boshell, Candra	0	4	☐	03/11/04	☐	03/11/17	☐
☐	1.00	Breceda, Margrett	1	0	☐	04/05/13	☐	03/11/13	☐
☑	1.00	Chiaro, Cecille	0	3	☐	04/04/01	☐	03/11/10	☐
☐	1.00	Deel, Salina			☐	- ??? -	☐	04/02/27	☐
☐	1.00	Devora, James	0	3	☐	04/04/08	☐	03/11/06	☐
☑	0.66	Filteau, Florrie	0	1	☐	04/03/31	☐	03/11/10	☐
☑	1.00	Gargus, Azalee	0	2	☐	03/11/10	☐	03/11/11	☐
☑	0.50	Gillie, Antonina	4	2	☐	03/11/04	☐	03/11/04	☐
☑	1.00	Glascock, Mirta	0	3	☐	04/02/12	☐	03/11/04	☐
☐	1.00	Gluc, Edris	1	0	☐	03/11/05	☐	03/11/10	☐
☐	1.00	Gourdine, Janis	4	4	☐	04/04/01	☐	03/11/18	☐

Done

Outline

- Task & Motivation
- System Architecture & Interfaces
- Scientific aspects
  - Problem Modeling
  - Problem Solving
  - Comparing & Characterizing Solvers
- Motivation revisited & Conclusions

Constraint-based Model

- **Variables**
  - Grading, conducting lectures, labs & recitations
- **Values**
  - Hired GTAs (+ preference for each value in domain)
- **Constraints**
  - Unary: ITA certification, enrollment, time conflict, non-zero preferences, etc.
  - Binary (Mutex): overlapping courses
  - Non-binary: same-TA, capacity, confinement
- **Objective**
  - longest partial and consistent solution (primary criterion)
  - while maximizing GTAs' preferences (secondary criterion)

Outline

- Task & Motivation
- System Architecture & Interfaces
- Scientific aspects
  - Problem Modeling
  - Problem Solving
  - Comparing & Characterizing Solvers
- Motivation revisited & Conclusions

Problem Solving

- Interactive decision making
  - Seamlessly switching between perspectives
  - Propagates decisions (MAC)
- Automated search algorithms
  - Heuristic backtrack search (BT)
  - Stochastic local search (LS)
  - Multi-agent search (ERA)
  - Randomized backtrack search (RDGR)
  - Future: Auction-based, GA, MIP, LD-search, etc.
- On-going: Cooperative/hybrid strategies

Manager interface: Interactive Selection

GTAP manager interface (DEMO)

http://csce.unl.edu/~gtademo/manager/

Spring 2004 | GTAs | Classes | Interactive selection | Search | Sys Admin | Logout

Interactive Selections

Assign
- GTAs to Classes
- Classes to GTAs

Refetch Data &
- Keep Selections
- Clear Selections

Current Selections
- Assign
- Save
- Clear

Print Page

Legend
Pre-Assigned Courses

Assign: GTAs to Classses

Options Menu

☑ Auto Assign

Class	Section	Description	Time	Days	Load	Assigned GTAs
CSCE 101	001T	Computer Science Fundamentals	1400-1515	TR	1	Neuberger Berniece
CSCE 101	001	Computer Science Fundamentals	1400-1515	TR	0.66	5 -- Studniarz Amiee (0)
CSCE 101L	002	Lab	1330-1520	W	0.25	4 -- Leshure Rena (0.16)
CSCE 101L	001	Lab	1030-1220	R	0.25	3 -- Gargus Azalee (0.25)
CSCE 105	151	Lab	1530-1620	M	0.25	
CSCE 105	152	Lab	1130-1220	W	0.25	
CSCE 105	153	Lab	1430-1520	R	0.25	
CSCE 105	150	Problem Solving w/Computers	1330-1420	MWF	0.75	
CSCE 150	151	Lab	1230-1320	M	0.25	
CSCE 150	152	Lab	1230-1320	T	0.25	
CSCE 150	153	Lab	1530-1620	W	0.25	
CSCE 150	154	Lab	1230-1320	W	0.25	

----- Nobody -----

Available GTAs
- 4 -- Kraushaar Berna (1)
- 4 -- Reighard Genevie (0.25)
- 3 -- Chiaro Cecille (0.75)
- 3 -- Gargus Azalee (0.25)
- 3 -- Glascock Mirta (0.75)
- 3 -- Kilker Lorilee (1)
- 3 -- Miskiewicz Jerrie (0.25)
- 3 -- Prisk Hallie (1)
- 3 -- Thornsbury Beaulah (1)
- 2 -- Gillie Antonina (0.5)
- 2 -- Mull Lorrine (1)
- 2 -- Thorson Minna (0.5)

Busy GTAs
- 4 -- Leshure Rena (0.16)
- 1 -- Boshell Candra (0.16)

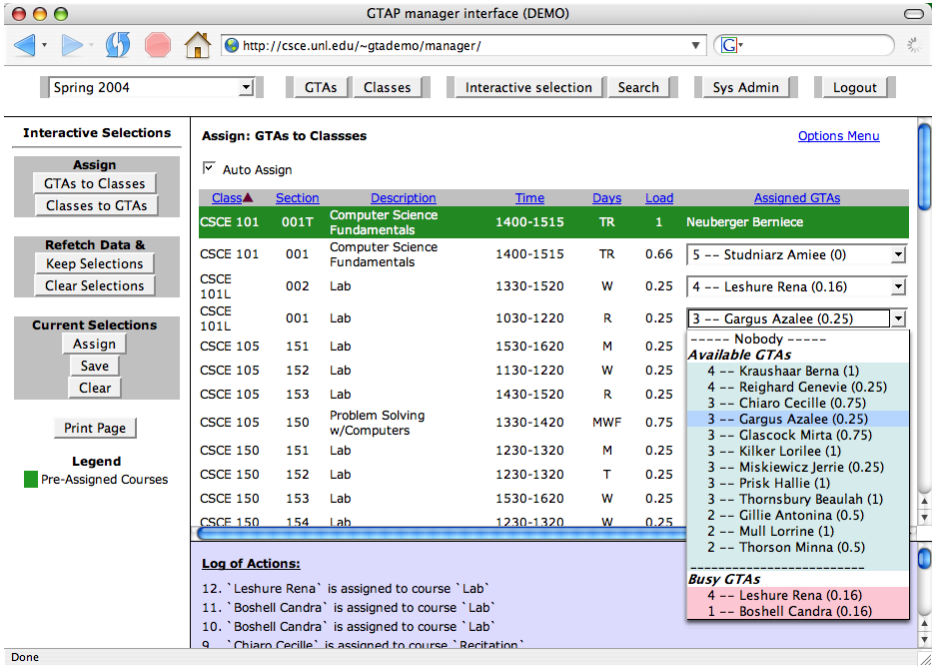
Log of Actions:

12. `Leshure Rena` is assigned to course `Lab`
11. `Boshell Candra` is assigned to course `Lab`
10. `Boshell Candra` is assigned to course `Lab`
9. `Chiaro Cecille` is assigned to course `Recitation`

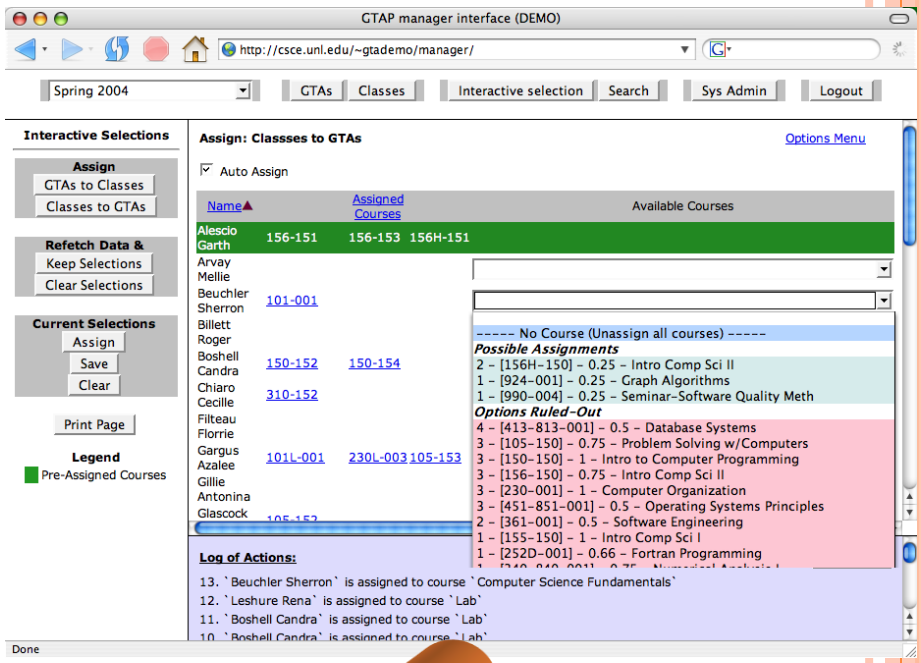
Done

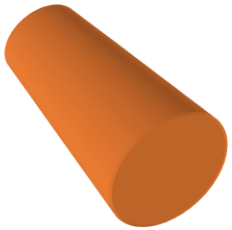
Dual perspective

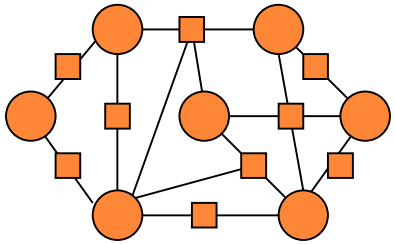
Task-centered view

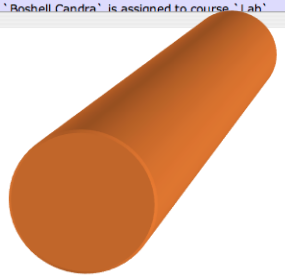


Resource-centered view



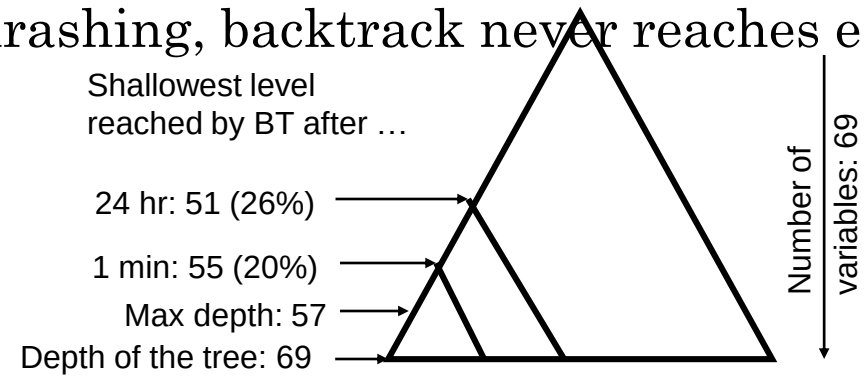






Heuristic BT Search

- Since we don't know, a priori, whether instance is solvable, tight, or over-constrained
  - Modified basic backtrack mechanism to deal with this situation
- We designed & tested various ordering heuristics:
  - Dynamic LD was consistently best
- Branching factor relatively huge (30)
  - Causes thrashing, backtrack never reaches early variables



Stochastic Local Search

- Hill-climbing with min-conflict heuristic
- Constraint propagation:
  - To handle non-binary constraints (e.g., high-arity capacity constraints)
- Greedy:
  - Consistent assignments are not undone
- Random walk to avoid local maxima
- Random restarts to recover from local maxima

Multi-Agent Search (ERA) [Liu et al. 02]

- "Extremely" decentralized local search
  - **A**gents (variables) seek to occupy best positions (values)
  - **E**nvironment records constraint violation in each position of an agent given positions of other agents
  - Agents move, egoistically, between positions according to reactive **R**ules
- Decisions are local
  - An agent can always kick other agents from a favorite position even when value of 'global objective function' is not improved
  - ✓ ERA appears immune to local optima
- Lack of centralized control
  - Agents continue to kick each other
  - × Deadlock appears in over-constrained problems

Randomized BT Search

- Random variable/value selection allows BT to visit a wider area of the search space [Gomes et al. 98]
- Restarts to overcome thrashing
- Walsh proposed RGR [Walsh 99]
- Our strategy, RDGR, improves RGR with dynamic choice of cutoff values for the restart strategy [Guddeti & Choueiry 04]

Optimizing solutions

- Primary criterion: solution length
  - BT, LS, ERA, RGR, RDGR
- Secondary criterion: preference values
  - BT, LS, RGR, RDGR
  - Criterion:
    - Average preference
    - Geometric mean
    - Maximum minimal preference

More Solvers…

- Interactive decision making
- Automated search algorithms
  - BT, LS, ERA, RGR, RDGR.
  - Future: Auction-based, GA, MIP, LD-search, etc.
- On-going: Cooperative / hybrid strategies

Outline

- Task & Motivation
- System Architecture & Interfaces
- Scientific aspects
  - Problem Modeling
  - Problem Solving
  - Comparing & Characterizing Solvers
- Motivation revisited & Conclusions

Conclusions

- Integrated interactive & automated problem-solving strategies
  - Reduced the burden of the manager
  - Lead to quick development of ‘stable’ solutions
- Our efforts
  - Helped the department
  - Trained students in CP techniques
  - Paved new avenues for research
    - Cooperative, hybrid search
    - Visualization of solution space

Other sample projects

1. Graduate TA Assignment Project (GTAAP)
   - Modeling, search, GUI
2. Temporal Reasoning
   - Constraint propagation, search, graph theory
3. Symmetry detection
   - Search, databases (computational)
4. Structural decompositions
   - Databases (theory), tractability results

The Research

- Modeling & Reformulation
- Propagation algorithms
- Search algorithms
- Decomposition algorithms
- Symmetry identification & breaking