Advanced Topics in AI

- Linear Programming -

Gerald Steinbauer

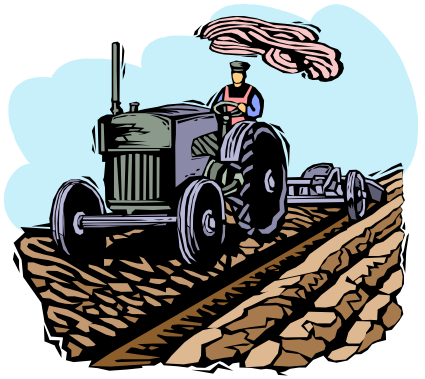
Institute for Software Technology

Outline

- Linear Programming (Morning – Seminar Room)
  - Formalization and Properties
  - Modeling
  - Solving (Simplex)
- Mixed Integer Programming (Morning – Seminar Room)
  - Formalization & Properties
  - Modeling
  - Solving (Branch and Bound)
- Applications (Afternoon – Robot Lab)
  - Task Assignments
  - Sudoku
- Practice (Afternoon – Robot Lab)

Linear Programming

Example



- a farmer has 75 acre of land
- he can grow wheat or barley
- wheat needs fertilizer for $ 120 for an acre, barley for $ 210
- the farmer can invest $ 15000
- he can store 4000 bushels of crops
- one acre produces 110 bushels of wheat or 30 bushels of barley
- the profit for a bushel of wheat is $ 1.3, $ 2.0 for barley
- what should the farmer do to maximize his profit

Motivation

- many problems are optimization problems – maximize an objective function w.r.t. constraints
- or can reformulated as such
- linear programming (LP) is a (quite) efficient way to solve them
- in worst-case exponential – in (practical) average polynomial
- first examples during WWII
- applications
  - resource or task allocation
  - production planning
  - network flows

Canonical Form

- a linear programming problem in canonical form comprises

1. variables $x_j \geq 0, j \in \{1, \ldots, n\}$
2. variables satisfy linear constraints of the form $\sum_{j=1}^{n} a_{ij} x_j \leq b_{i,}, i \in \{1, \ldots, m\}$
3. goal is to minimize a linear (objective) function of the variables: $c_1 + c_2 + \cdots + c_n$

- easily written in matrix form:

$$\begin{aligned} \min \quad & \boldsymbol{c}^T \boldsymbol{x} \\ s.t. \quad & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \\ & x_j \geq 0 \end{aligned}$$

- we call it feasible canonical form if all $b_i \geq 0$

Interpretation of the Canonical Form

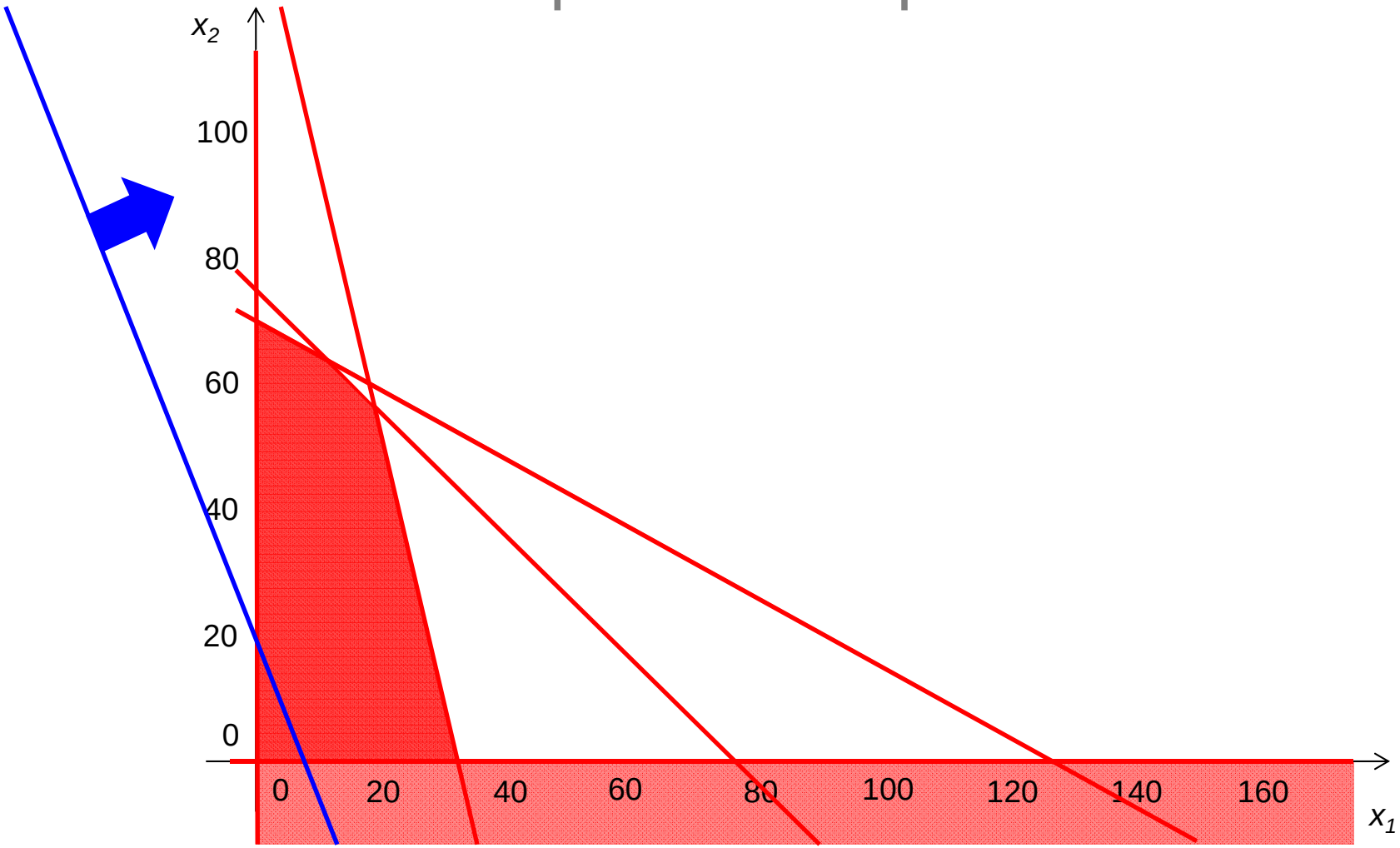
- $\boldsymbol{x}$ is a vector of variables of length n – represents non-negative quantities
- $\boldsymbol{A}$ is a matrix of size $m \times n$ – represents cost coefficients
- $\boldsymbol{b}$ is a vector of length m – represents available resources
- $\boldsymbol{c}$ is a vector of length n – represents profit/quality coefficients

Initial Properties

- ***x*** is called a feasible solution to the LP problem if it satisfies all constraints
- there might be no (infeasible) or a finite or infinite number of solutions
- ***x*** is called an optimal solution to the LP problem if it is feasible and minimizes the objective function
- a LP problem is called unbound if there are infinite many feasible solutions but no optimal solution
- there might be infinite many feasible solutions that all are optimal solutions

Farmer Example

Graphical Interpretation



Standard (Slack) From

- for some reasons, e.g. easier solving, a different representation can be used

$$\begin{aligned} &max\ \boldsymbol{c'}^T\boldsymbol{x'} \\ &\text{s.t.}\quad \boldsymbol{A'x'} = \boldsymbol{b} \\ &\qquad x'_j \geq 0 \end{aligned}$$

- additionally to the components of the canonical form we may introduce slack variables s_i for $i \in \{1,\ldots,p\}$
- $\boldsymbol{A'}=[\boldsymbol{A}|\boldsymbol{Q}]$ with $\boldsymbol{Q}$ a matrix of size $m \times p$, $\boldsymbol{x'}^T=[\boldsymbol{x}^T,\boldsymbol{s}^T]$, $\boldsymbol{c'}^T=[\boldsymbol{c}^T,\boldsymbol{0}^T]$ with $\boldsymbol{0}$ a null vector of length p
- instead of minimizing the objective we look now for the maximum

Standard (Slack) From

- there are rules to transform different LP representations to the standard form

1. if the problem is to minimize z, transform it to maximize $-z$
2. if a constraint has the form $a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \leq b_i$ add a non-negative slack variable s_i, $a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n + s_i = b_i$
3. if a constraint has the form $a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \geq b_i$ subtract a non-negative slack variable s_i, $a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n - s_i = b_i$
4. if some variable x_j is unrestricted in sign, replace it by $x_j' - x_j''$ where $x_j' \geq 0$ and $x_j'' \geq 0$

Example Standard Form

Duality

- LP problems can be transformed into a dual LP problem
- the original LP problem is called primal problem
- sometimes dual problems are easier to solve
- primal:

$$\begin{aligned} &max\ \boldsymbol{c}^T\boldsymbol{x} \\ &\text{s.t.}\quad \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \\ &\qquad x_j \geq 0, j \in \{1, \dots, N\} \end{aligned}$$

- dual:

$$\begin{aligned} &min\ \boldsymbol{b}^T\boldsymbol{y} \\ &\text{s.t.}\quad \boldsymbol{A}^T\boldsymbol{y} \geq \boldsymbol{c} \\ &\qquad y_i \geq 0, i \in \{1, \dots, M\} \end{aligned}$$

Duality Properties

- symmetry: the dual of a dual of a LP problem is the primal problem
- weak duality: if $\boldsymbol{x}$ is feasible for the primal and $\boldsymbol{y}$ feasible for the dual, then $\boldsymbol{c}^T\boldsymbol{x} \leq \boldsymbol{b}^T\boldsymbol{y}$, the dual provides an upper bound for the objective function of the primal
- strong duality: if $\boldsymbol{x}$ is optimal for the primal than the dual has an optimal solution $\boldsymbol{y}$ and $\boldsymbol{c}^T\boldsymbol{x} = \boldsymbol{b}^T\boldsymbol{y}$
- unbound: if the dual is infeasible the primal is unbounded
- infeasible: if the dual is unbounded the primal is infeasible

Basic Solution

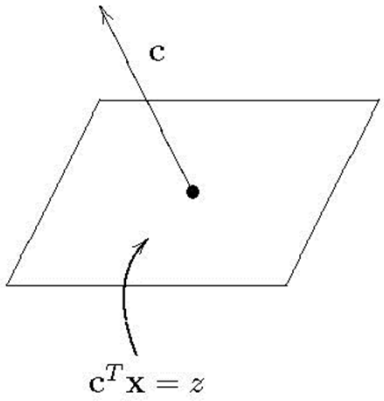
- consider $\boldsymbol{Ax} = \boldsymbol{b}$ with $rank(\boldsymbol{A}) = m < n$
- let $\boldsymbol{B}$ a matrix formed by m columns of n columns of $\boldsymbol{A}$
- if $\boldsymbol{B}$ is non-singular and all variables x_j associated with selected columns are set zero the resulting solution called basic solution
- variables x_j associated with selected columns are called basic variables ($\boldsymbol{x_B}$), all others non-basic variables ($\boldsymbol{x_\beta}$)

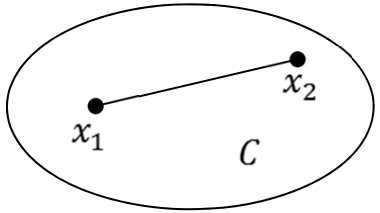
$$\boldsymbol{x_\beta} = \boldsymbol{0}, \boldsymbol{x_B} = \boldsymbol{B}^{-1}\boldsymbol{b}, x = \begin{bmatrix} \boldsymbol{B}^{-1}\boldsymbol{b}, \\ \boldsymbol{0} \end{bmatrix}$$

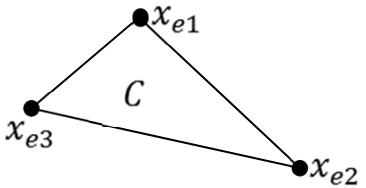
- a basic solution is called degenerate if one or more

Some more Basics

- a hyperplane in $\mathbb{R}^n$ is the set $\{x|c^T x = z$ with c a fixed non-zero vector in R^n and $z \in \mathbb{R}$; vector c forms the normal to the hyperplane
- a set C is called convex if for all x_1 and x_2 in C and $\lambda \in [0,1]$ we have $\lambda x_1 + (1-\lambda)x_2 \in C$
- a point x_e is an extreme point of a convex set C if there are no two distinct points x_1 and x_2 in C such that $x_e = \lambda x_1 + (1-\lambda)x_2$ for some $\lambda \in (0,1)$







Feasible Region

- the feasible region contains all feasible solutions of a linear programming problem

a. all hyperplanes are convex

b. all closed half-spaces $\{x|c^T x \leq, \geq z\}$ are convex

c. all open half-spaces $\{x|c^T x <, > z\}$ are convex

d. any intersection of convex sets is still convex

e. the set of feasible solutions of a linear programming problem is a convex set

- constraints $(\boldsymbol{a_i x} =, \leq, \geq b_i)$ according to a,b,c are convex
- the intersection forms the set of feasible solutions which is convex according to d

Supporting Hyperplane

- a point $\boldsymbol{x}$ is a boundary point of a set $\boldsymbol{C}$ if every neighborhood of $\boldsymbol{x}$ contains a point in $\boldsymbol{C}$ and a point not in $\boldsymbol{C}$
- let $\boldsymbol{y}$ be a boundary point of convex set $\boldsymbol{C}$, then $X = \{\boldsymbol{x}|\boldsymbol{a}^T\boldsymbol{x} = z\}$ is called a supporting hyperplane of $\boldsymbol{C}$ at $\boldsymbol{y}$ if $\boldsymbol{y} \in X$ and either $\boldsymbol{C} \subseteq \{\boldsymbol{x}|\boldsymbol{a}^T\boldsymbol{x} \geq z\}$ **or** $\boldsymbol{C} \subseteq \{\boldsymbol{x}|\boldsymbol{a}^T\boldsymbol{x} \leq z\}$
- if $\boldsymbol{C}$ is a closed convex set there is at least one supporting hyperplane at $\boldsymbol{y}$
- a set $\boldsymbol{C}$ is said to be bounded from below if $inf\{x_j|x_j \ \ldots j\text{-th component } of\ \boldsymbol{x} \text{ in } \boldsymbol{C}\} > -\infty$ for all j

Basis of Simplex Algorithm

- $\mathbb{R}_+^n \equiv \{\boldsymbol{x} \in \mathbb{R}^n | \boldsymbol{x}^T = [x_1, \dots, x_n], x_1 \geq 0, j = 1, \dots, n\}$ is a set bounded from below
- let $\boldsymbol{C}$ be a closed convex set which is bounded from below, then every supporting hyperplane of $\boldsymbol{C}$ contains an extreme point
- the feasible region of a linear programming problem is convex, closed and bounded from below
- a feasible solution to the linear programming problem is called feasible basic solution if it is a basic solution
- if a feasible basic solution is non-degenerate we call it non-degenerate feasible basic solution

Basis of Simplex Algorithm cont.

- the objective function is a hyperplane of the form $X = \{\boldsymbol{x} | \boldsymbol{c}^T \boldsymbol{x} = z\}$
- the feasible basic solutions are extreme points of the corresponding feasible region
- the optimal solution of a linear programming problem occurs on the extreme points of the feasible region
- the optimal hyperplane is a supporting hyperplane of the feasible region at the optimal solution $\boldsymbol{x}_0$

Simplex Algorithm

- we like to have an efficient algorithm to solve linear programming problems optimally
- so far we know
  - optimal solutions sit on extreme points of the feasible region
  - basic feasible solutions are extreme points of the feasible region
  - the optimal hyperplane at the optimal solution is a supporting hyperplane
- Idea
  1. obtain an initial basic feasible solution
  2. check for optimality $\rightarrow$ exit
  3. generate a new basic feasible solution
  4. goto 2
- there are $\binom{n}{m}$ basic solutions

Start with a basic feasible solution

- we assume a linear programming problem in standard form:

$$\max \; z = \boldsymbol{c}^T \boldsymbol{x}$$
$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$
$$x_i \geq 0$$

- we further assume $rank(\boldsymbol{A}) = m, b_i \geq 0$

1. select m columns $\boldsymbol{a}_j$ from $\boldsymbol{A}$ and form the matrix $\boldsymbol{B} = [\boldsymbol{a}_j^1, \dots, \boldsymbol{a}_j^m]$, lets assume $\boldsymbol{B}$ to be non-singular
2. solve the basic variables $\boldsymbol{x}_B = \boldsymbol{B}^{-1}\boldsymbol{b}$ and set the non-basic variables $\boldsymbol{x}_\beta$ to zero
3. check if basic solution $x^T = [\boldsymbol{x}_B^T, \boldsymbol{x}_\beta^T]$ is feasible and $\boldsymbol{x}_B$ is non-degenerate

Improving a Basic Feasible Solution

- we like to quickly find another basic feasible solution with a larger objective value
- we can represent all columns $\boldsymbol{a}_j$ of $\boldsymbol{A}$ using the basis $\boldsymbol{B}$ so that $\boldsymbol{a}_j = \boldsymbol{B}\boldsymbol{y}_j$ with $\boldsymbol{y}_j^T = [\boldsymbol{y}_{1j}, \dots, \boldsymbol{y}_{mj}]$
- we introduce so called reduced cost coefficients w.r.t. $\boldsymbol{B}$ $c_j - z_j$ with $z_j = \boldsymbol{c}_B^T y_j$ and $\boldsymbol{c}_B^T$ represents the cost coefficients w.r.t. $\boldsymbol{B}$
- using this we can replace the *r*-th column of $\boldsymbol{B}$ with the *j*-th column of $\boldsymbol{A}$
- we change only one column for the new basis

Feasibility and Optimality Conditions

- now we can directly update the feasible basic solution:

$$\hat{\boldsymbol{x}}_B = \begin{cases} x_{B_i} - x_{B_r} \dfrac{y_{ij}}{y_{rj}}, i = 1, \dots, j-1, j+1, m \\ \dfrac{x_{B_r}}{y_{rj}}, i = j \end{cases}$$

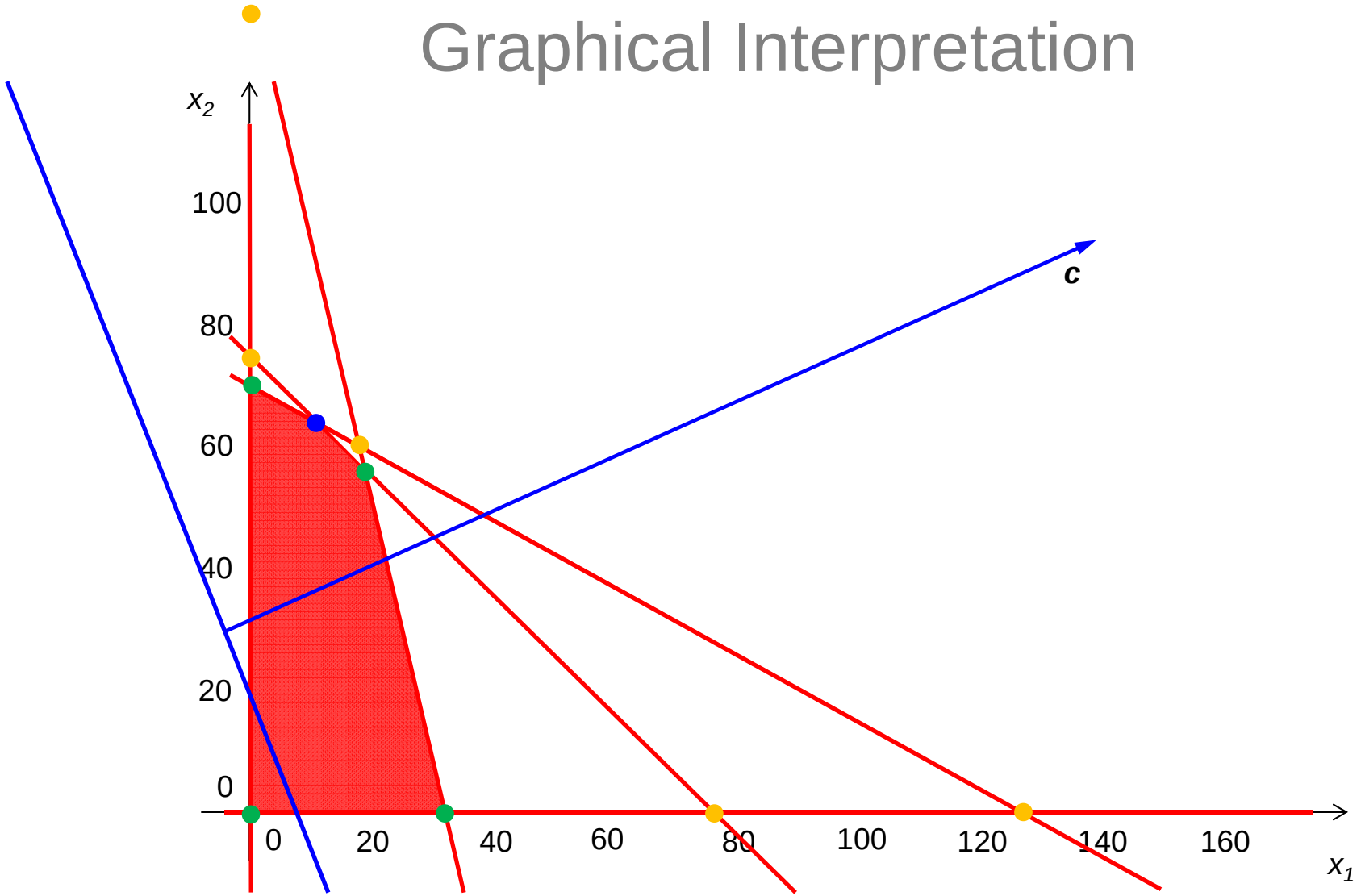
- feasibility: we chose *r* so that

$$\frac{x_{B_r}}{y_{rj}} = \min_{i=1,2,\dots,m} \left\{ \frac{x_{B_r}}{y_{ij}} : y_{ij} > 0 \right\}$$

- optimality: a feasible basic solution $\boldsymbol{x}_B$ is optimal if $z_j - c_j \geq 0$ for every column $\boldsymbol{a}_j$ in $\boldsymbol{A}$

Farmer Example

Graphical Interpretation



Simplex in Tableau Form

- starting from a feasible canonical form: $\max\ z = \boldsymbol{c}^T \boldsymbol{x}$ $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$ $x_i, b_i \geq 0$
- we add slack variables $\boldsymbol{x}_s = x_{n+1}, \ldots, x_{n+m}$ with $\boldsymbol{A}\boldsymbol{x} + \boldsymbol{I}\boldsymbol{x}_s = \boldsymbol{b}$
- then then we can easily run the simplex method using a tableau starting with an initial tableau T_1
- each iteration of the simplex method generates a new T_{i+1}

Tableau Layout

	x_B	n+m+1 columns						
		x_1	$\cdots$	x_n	x_{n+1}	$\cdots$	x_{m+n}	b
m+ 1 rows	x_{B_1}	y_{11}	$\cdots$	y_{1n}	y_{1n+1}	$\cdots$	y_{1m+n}	x_{B1}
	x_{B_2}	y_{21}		y_{2n}	y_{2n+1}		y_{2m+n}	x_{B2}
	$\vdots$		$\ddots$			$\ddots$		$\vdots$
	$x_{B_{m-1}}$	y_{m-11}		y_{m-1n}	y_{m-1n+1}		y_{m-1m+n}	x_{Bm-1}
	x_{B_m}	y_{m1}	$\cdots$	y_{mn}	y_{mn+1}	$\cdots$	y_{mm+n}	x_{Bm}
	x_0	y_{01}	$\cdots$	y_{0n}	y_{0n+1}	$\cdots$	y_{0m+n}	z

Initial Tableau Layout

	x_B	n+m+1 columns						
		x_1	$\cdots$	x_n	x_{n+1}	$\cdots$	x_{m+n}	b
m+ 1 rows	x_{n+1}	a_{11}	$\cdots$	a_{1n}	1	$\cdots$	0	b_1
	x_{n+2}	a_{21}		a_{2n}	0		0	b_2
	$\vdots$		$\ddots$		0	$\ddots$	0	$\vdots$
	x_{n+m-1}	a_{m-11}		a_{m-1n}	0		0	b_{m-1}
	x_{n+m}	a_{11}	$\cdots$	a_{mn}	0	$\cdots$	1	b_m
	x_0	$-c_1$	$\cdots$	$-c_n$	0	$\cdots$	0	z

Managing the Tableau

- checking optimality: if all coefficient in x_0 are non-negative an optimal solution is reached
- selecting optimal column (x_j to enter basis $\boldsymbol{x}_B$): select column j with most negative coefficient in x_0
- selecting feasibility row (x_r to leave basis $\boldsymbol{x}_B$): select row r so that $\frac{x_{Br}}{y_{rj}} = \min_{i=1,..,m} \left\{ \frac{b_i}{y_{ij}} | y_{ij} > 0 \right\}$ where y_{ij} is the pivot element

Managing the Tableau cont.

- update the basis $\boldsymbol{x}_B$: replace *r*-th entry in $\boldsymbol{x}_B$ with x_j
- update pivot row: replace y_{rk} by $\frac{y_{rk}}{y_{rj}}$ for $k = 1, \dots, m + n$
- update pivot column: replace y_{rj} by 1 and y_{rk} by 0 for $k = 1, \dots, m, k \neq r$
- update other elements: replace y_{ik} by $y_{ik} - \frac{y_{ij} y_{rk}}{y_{rj}}$ and x_{Bi} by $x_{Bi} - \frac{y_{ij} x_{Br}}{y_{rj}}$
- update row x_0: replace y_{0i} by $\boldsymbol{c}_B^T \boldsymbol{y}_i - c_i$ and z by $\boldsymbol{c}_B^T \boldsymbol{x}_B$

Farmer Example

Initialization from Standard Form

- we start now from (no initial identity basis):

$$\max \; z = \boldsymbol{c}^T \boldsymbol{x}$$
$$\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$$
$$x_i, b_i \geq 0$$

- we chose m linear independent columns from $\boldsymbol{A}$ for $\boldsymbol{B}$ an can represent $\boldsymbol{A}$ by $[\boldsymbol{N}\boldsymbol{B}]$
- we get the augmented system:

$$\boldsymbol{N}\boldsymbol{x}_N + \boldsymbol{B}\boldsymbol{x}_B = \boldsymbol{b}$$
$$x_0 - \boldsymbol{c}_N^T \boldsymbol{x}_N - \boldsymbol{c}_B^T \boldsymbol{x}_B = 0$$

- transform to general representation:

$$\boldsymbol{B}^{-1}\boldsymbol{N}\boldsymbol{x}_N + \boldsymbol{x}_B = \boldsymbol{B}^{-1}\boldsymbol{b}$$
$$x_0 - (\boldsymbol{c}_N^T - \boldsymbol{z}_B^T)\boldsymbol{x}_N = \boldsymbol{c}_B^T \boldsymbol{B}^{-1}\boldsymbol{b}$$

Table from Standard Form

	$\boldsymbol{x_N}$	$\boldsymbol{x_B}$	$\boldsymbol{b}$
$\boldsymbol{x_B}$	$\boldsymbol{B^{-1}N}$	$\boldsymbol{I}$	$\boldsymbol{B^{-1}b}$
x_0	$-(\boldsymbol{c_N^T} - \boldsymbol{z_B^T})\boldsymbol{x_N}$	$\mathbf{0}$	$\boldsymbol{c_B^T B^{-1} b}$

Some Pathological Cases

- no feasible solution: if we reach an tableau with all coefficients in $\boldsymbol{x}_0$ non-negative (optimal solution) but the basic solution is infeasible
- unbounded feasible region: if all $y_{ij} > 0$ for a non-basic variable x_j; the solution is unbounded if $z_j - c_j < 0$ too
- infinite number of optimal solutions: if the reduced costs $z_j - c_j$ of a non-basic variable x_j is zero
- degeneracy and cycling: if a basic feasible solution is degenerate on or more constraints are redundant – the method may cycle between basic solutions

(Mixed) Integer Linear Programming

Motivation

- in some contexts it makes sense to take integral quantities of certain goods or resources
  - human resource planning
  - locations
- in some contexts it makes sense to take binary decisions
  - production planning
  - assignment problem
  - time tables
- prominent examples
  - Integer Rucksack probem
  - minimum spanning tree problem

(Mixed) Integer Linear Program

- a mixed integer linear programming problem (MILP) is of the form

$$\begin{aligned} \min\ & \boldsymbol{c}^T \boldsymbol{x} \\ & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & x_i \geq 0 \\ & x_i \in \mathbb{Z}, \text{for some } i \end{aligned}$$

- if all variables x_i need to be integral it is called (pure) integer linear program problem (ILP)
- some variables x_i might be binary (only 0 or 1)
- if all variables x_i need to be binary it is called 0-1 linear program problem

Complexity

- including integer variables increase enormously the modeling power
- but it increases the complexity drastically
- linear programming can be solved in polynomial time
- integer programming is NP-complete
  - no known polynomial-time algorithm
  - even small problems may be hard to solve

Decision Variables

- binary variables are a powerful tool
- they can be used as decision variables
  - a task is assigned to person
  - a task consumes particular resources

Example – Knapsack Problem

- we use binary variables to model yes/no decisions
- we are given a set of items with associated values and weights
- we wish to select a subset of maximum value such that the total weight is less than a constant *K*
- we associate a 0-1 variable with each item indicating whether it is selected or not

$$\max \sum_{j=1}^{m} c_j x_j$$

$$\sum_{j=1}^{m} w_j x_j \leq K$$

$$x_i \geq 0$$

$$x_i \; binary$$

LP Relaxation of MILP

- given a MILP:

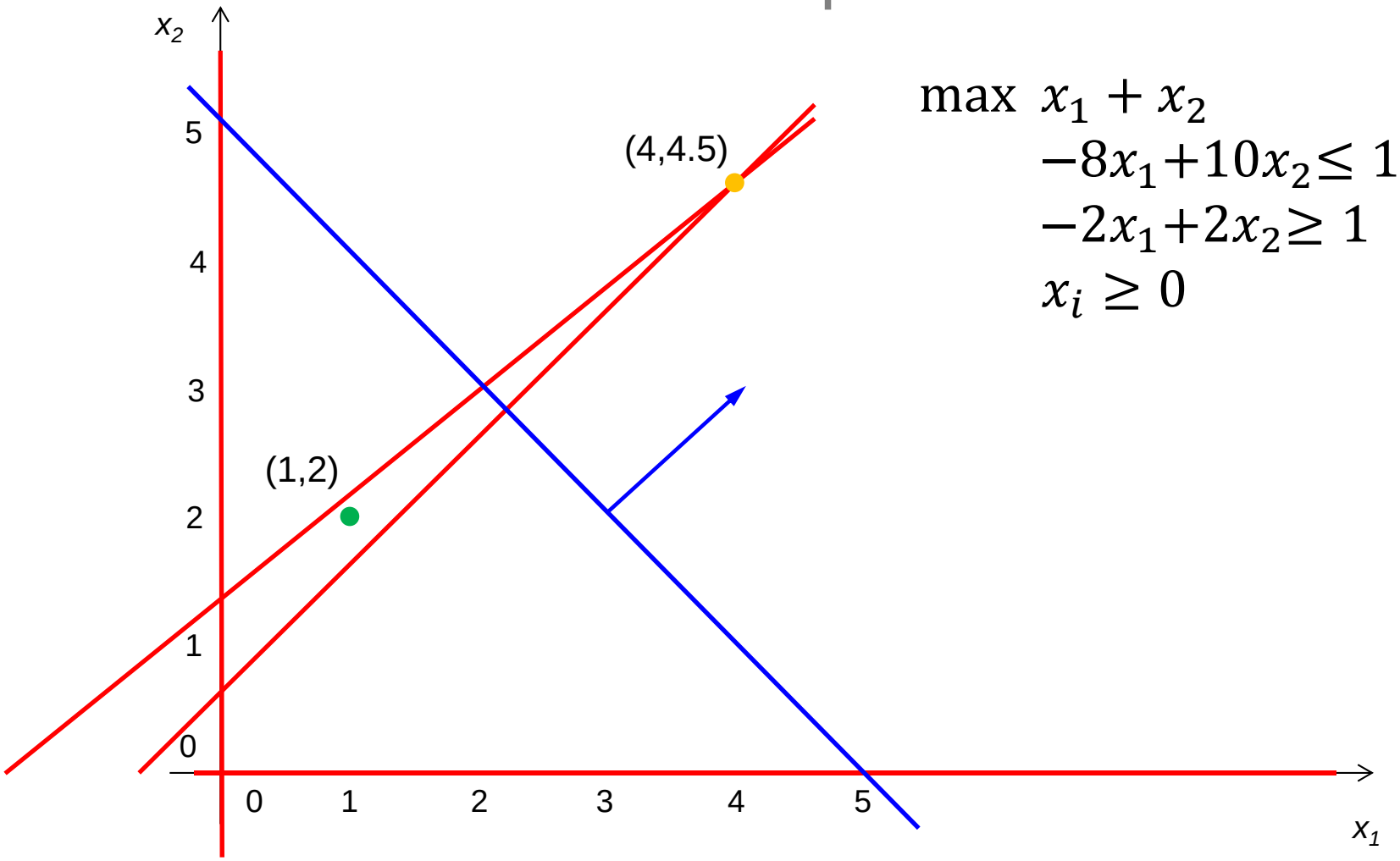
$$\begin{aligned} \min\ & \boldsymbol{c}^T \boldsymbol{x} \\ & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & x_i \geq 0 \\ & x_i \in \mathbb{Z}, \text{for some } i \end{aligned}$$

- its linear relaxation consists of the LP obtained by dropping the integrality constrains:

$$\begin{aligned} \min\ & \boldsymbol{c}^T \boldsymbol{x} \\ & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & x_i \geq 0 \end{aligned}$$

- can we simply round to obtain the optimal solution

Example



$$\max\ x_1 + x_2$$
$$-8x_1 + 10x_2 \leq 13$$
$$-2x_1 + 2x_2 \geq 1$$
$$x_i \geq 0$$

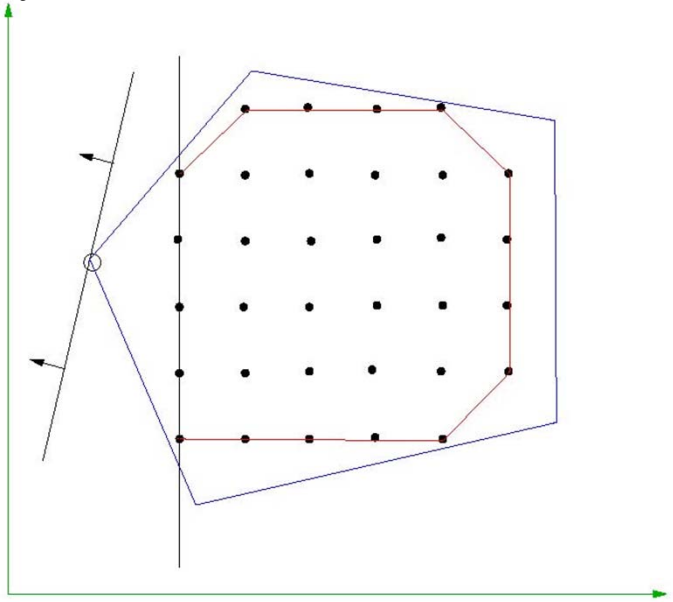
Rounding is not an Option

- in general an optimal solution to a LP relaxation des not tell us anything about the optimal solution of the MILP
- rounding to a feasible integral solution might be difficult
- the optimal solution of LP relaxation can be arbitrarily far away from the optimal solution of the MILP
- but it provides a lower bound on the optimal solution to the MILP

The Geometry of Integer Programming

$$\min \ \boldsymbol{c}^T \boldsymbol{x}$$
$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$
$$x_i \geq 0$$
$$x_i \in \mathbb{Z}$$

- let's consider again an integer linear program
- the feasible region is the integer points inside a polyhedron



- it is easy to see why solving the LP relaxation does not necessarily yield a good solution

Branch and Bound

- branch and bound is the most commonly-used algorithm for solving MILP
- it is a divide and conquer approach
- suppose F is the feasible region for some MILP and we wish to solve $\min_{x \in F} c^T x$
- consider a partition of F into subsets $F_1, \cdots, F_k$. Then

$$\min_{x \in F} c^T x = \min_{i=1,\ldots,k} \left\{ \min_{x \in F_i} c^T x \right\}$$

- we can optimize over each subset separately.
- idea: if we can't solve the original problem directly, we might be able to solve the smaller subproblems recursively

LP-based Branch and Bound

- in LP-based branch and bound, we first solve the LP relaxation of the original problem. The result is one of the following:

1. the LP is infeasible ⇒ MILP is infeasible
2. we obtain a feasible solution for the MILP ⇒ optimal solution
3. we obtain an optimal solution to the LP that is not feasible for the MILP ⇒ lower bound

- in the first two cases, we are finished
- in the third case, we must branch and recursively solve the resulting subproblems

subproblems.

Branching in LP-based Branch and Bound

- the most common way to branch is as follows:
- select a variable i whose value $\hat{x}_i$ is fractional in the LP solution
- create two subproblems
  - in one subproblem, impose the constraint $x_i \leq \lfloor \hat{x}_i \rfloor$
  - in the other subproblem, impose the constraint $x_i \geq \lceil \hat{x}_i \rceil$
- such a method of branching is called a branching rule

Continuing the Algorithm After Branching

- after branching, we solve each of the subproblems recursively
- now we have an additional factor to consider if the optimal solution value to the LP relaxation is greater than the current upper bound, we need not consider the subproblem further
- this is the key to the efficiency of the algorithm

Continuing the Algorithm After Branching

- Terminology
- the subproblems form a search tree
- each subproblems is linked to its parent and eventually to its children
- eliminating a problem from further consideration is called pruning
- the act of bounding and then branching is called processing
- a subproblem that has not yet been considered is called a candidate for processing
- the set of candidates for processing is called the candidate list

LP-based Branch and Bound Algorithm

1. to start, derive an upper bound U using a heuristic method
2. put the original problem on the candidate list
3. select a problem S from the candidate list and solve the LP relaxation to obtain the bound $b(S)$
   - if the LP is infeasible ⇒ node can be pruned
   - otherwise, if $b(S) \geq U$ ⇒ node can be pruned
   - otherwise, $b(S) < U$ and the solution is feasible for the MILP ⇒ set $U \leftarrow b(S)$
   - otherwise, branch and add the new subproblem to the candidate list.
4. if the candidate list in nonempty, go to Step 2, otherwise, the algorithms completed.

Branch and Bound

$S \coloneqq \{P_0\}$ /* set of pending problems */

$Z \coloneqq +\infty$ /* best cost found so far */

while $S \neq \emptyset$ do

  remove P from S; solve $LP(P)$

  if $LP(P)$ is feasible then

    let β be basic solution obtained after solving $LP(P)$

    if β satisfies integrality constraints then

      if β is optimal for $LP(P)$ then

        if $cost(\beta) < Z$ then store β; update Z

      else return UNBOUNDED

    else

      if β is optimal for $LP(P)$ and P can be pruned then continue

      let x_j be integer variable such that $\beta_j \notin \mathbb{Z}$

      $S \coloneqq S \cup \{P \wedge x_j \leq \lfloor \beta_j \rfloor, P \wedge x_j \geq \lceil \beta_j \rceil\}$

return Z

Choices in Branch and Bound

- selecting the next candidate to process:
  - "Best-first" always chooses the candidate with the lowest lower bound
  - this rule minimizes the size of the tree
  - there may be practical reasons to deviate from this rule
  - breadth-first
  - depth-first
- choosing a branching rule
  - branching wisely is extremely important
  - a "poor" branching can slow the algorithm significantly
  - variable ordering
  - closest halfway two integers

MILP Example

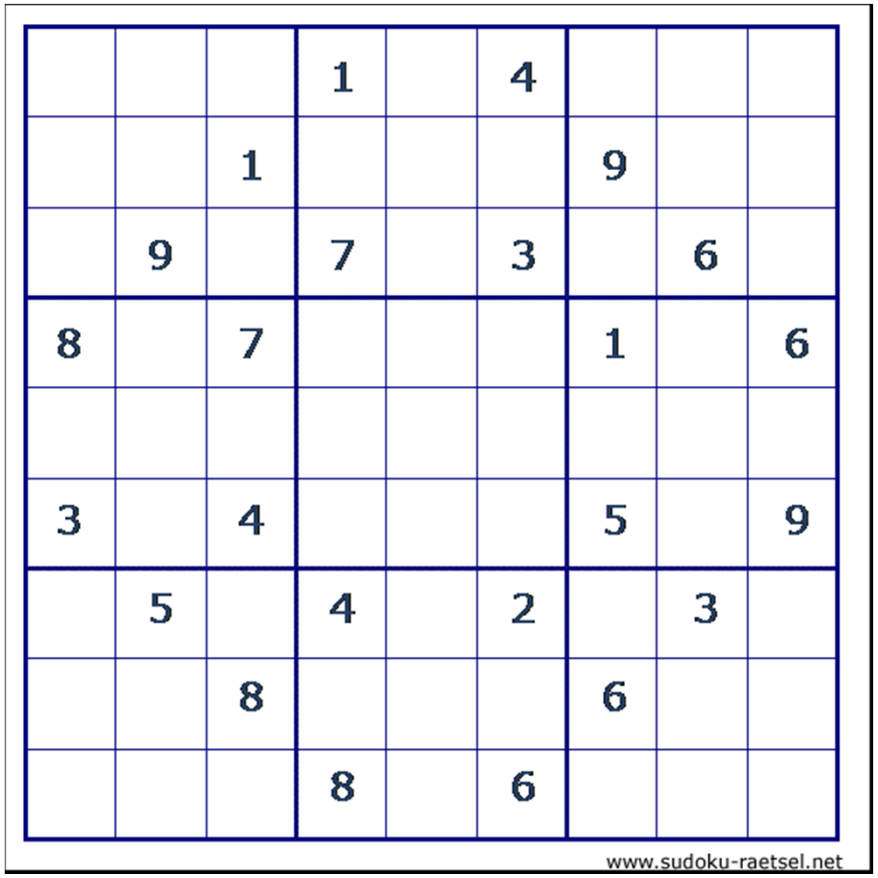
Modeling Issues – Binary Variables

- we point out already that binary variables are powerful tools
- m out of n selection

$$\sum_{j=1}^{n} x_i = m$$

Applications

Sudoku



Properties

- we have an order m
- we have $n=m^2$ distinctive numbers, 1,…,n
- each number must occure only once in each row
- each number must occure only once in each column
- each number must occure only once in each sub block of size mxm

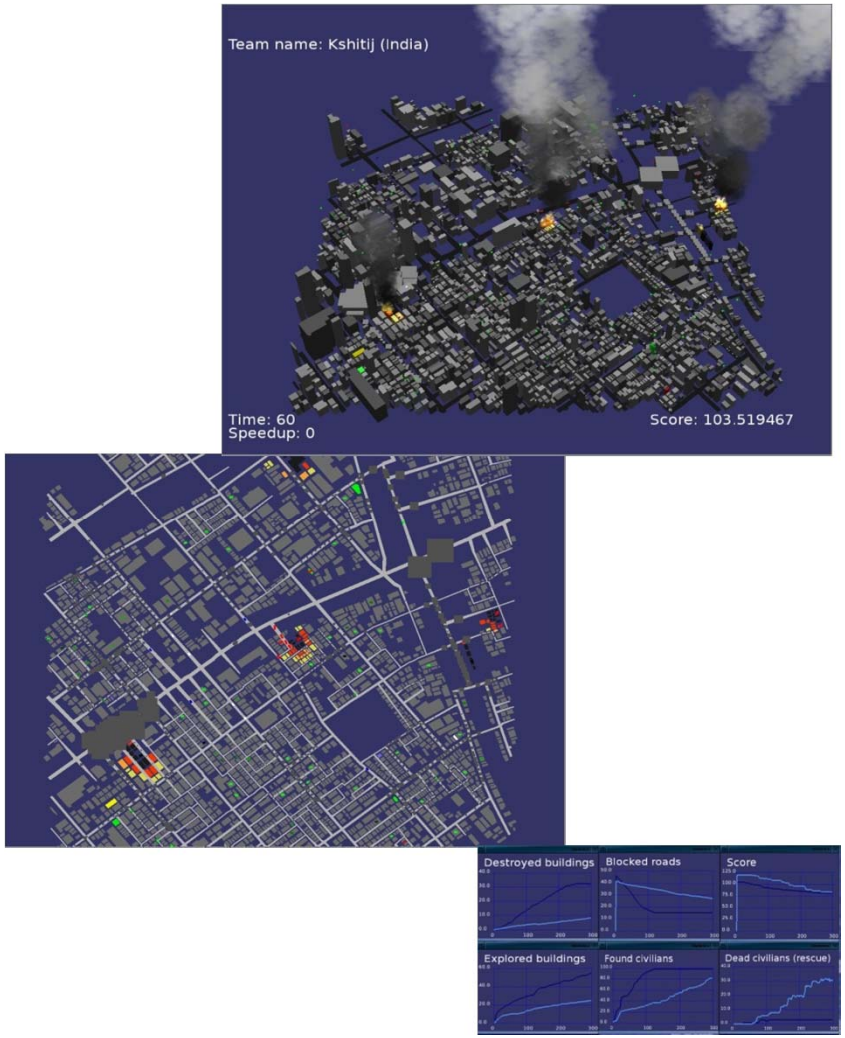
Brute Force

- Sudoku of order 3 has about $6{,}7 \cdot 10^{21}$ final grids
- representation: numbers on the grids
- generate all possible grids and check if they are a solution
- iterate trough all assignments of numbers to cells

Rescue Simulation League

Agent Competition

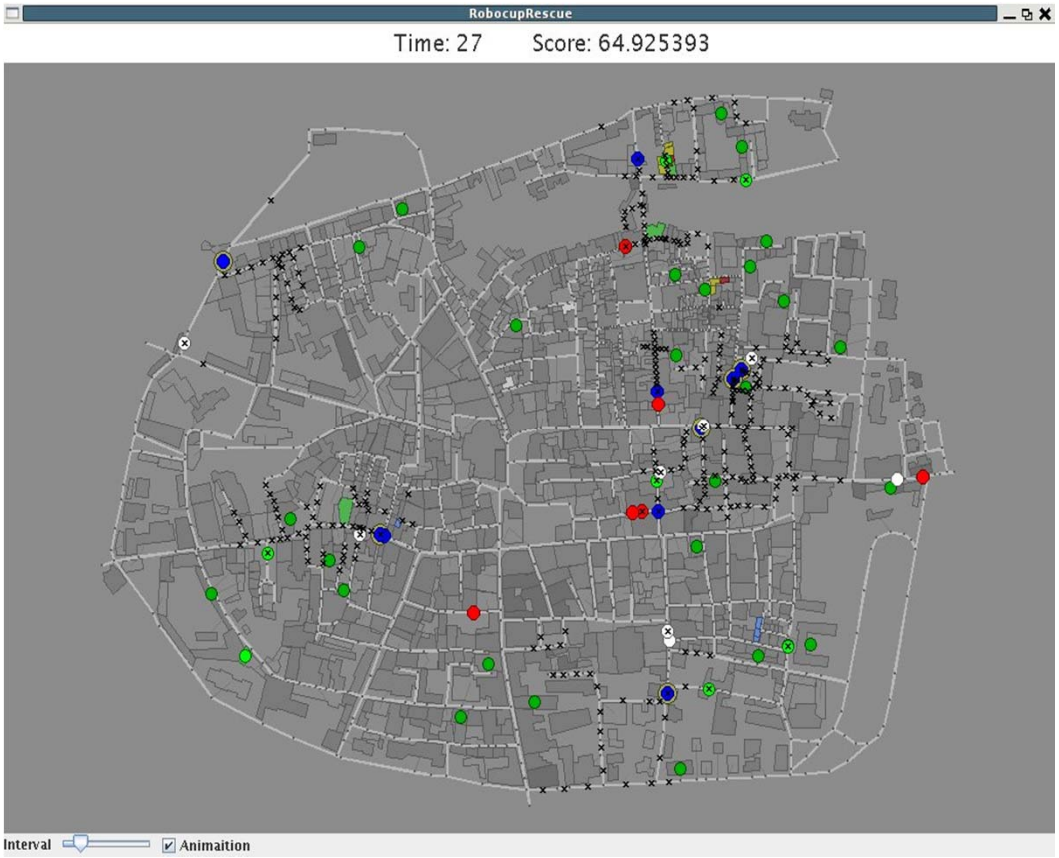
- large scale disaster simulation
- simulators for earthquake, fire, civilians, and traffic
- the task is to develop software agents with different roles, that
  - make roads passable (police)
  - extinguish the fires (fire brigades)
  - rescue all civilians (ambulances)
- difference to Soccer Simulation:
  - a challenging Multi-Agent Problem since Agents **must** cooperate
  - simulator components are developed within the “Infrastructure Competition”
- the more civilians alive and fires put out, the higher the score

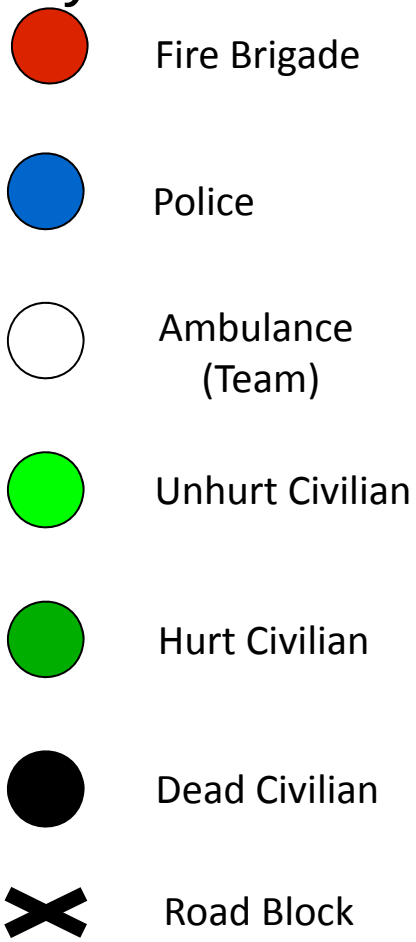


RoboCup Rescue Scenario

exemplar application: earthquake in a given city

Foligno – dense building layout





Platoon Agents

- Fire Brigades
  - extinguish fires
  - have limited water tank capacity
  - can collaborate to extinguish more quickly
- Ambulances
  - dig out civilians who are buried under rubble (takes a number of cycles)
  - carry victims to a refuge
  - can collaborate to dig out more quickly
- Police Force
  - can remove obstructions on roads
  - cannot collaborate to remove obstructions faster

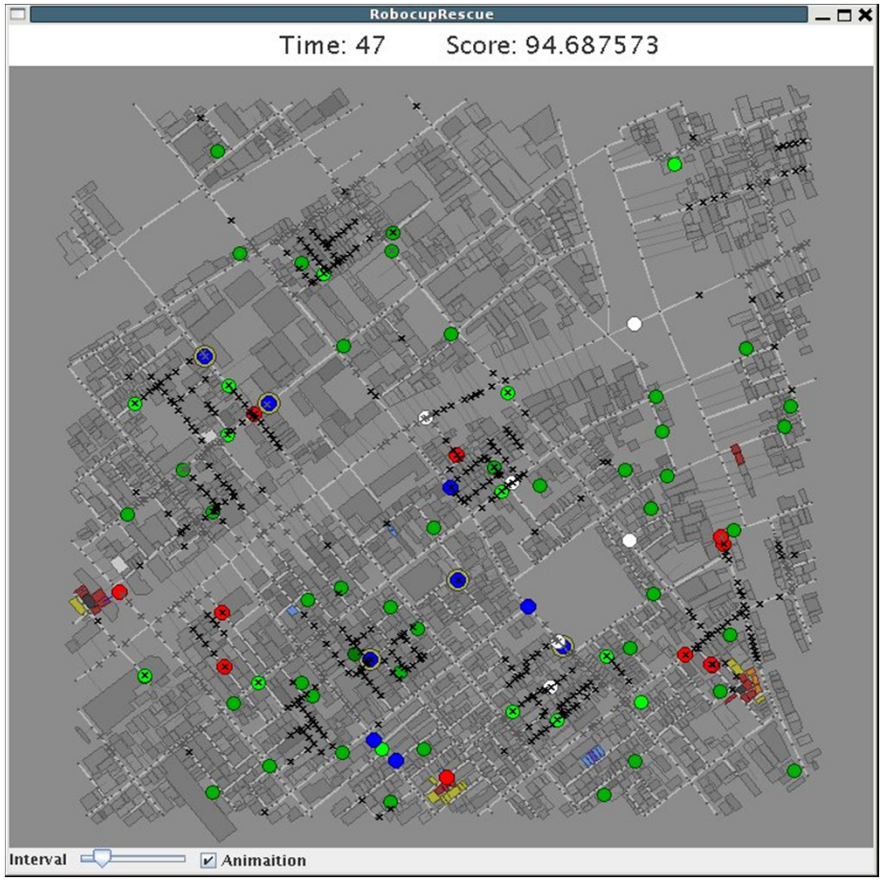
Centre Agents

- Ambulance Centre
  - talks to ambulances and to other centres
- Fire Centre
  - talks to fire brigades and other centres
- Police Centre
  - talks to police force agents and other centres

- located in a building which may or may not collapse or get burnt
- usually act as a synchronisation point/computation centre

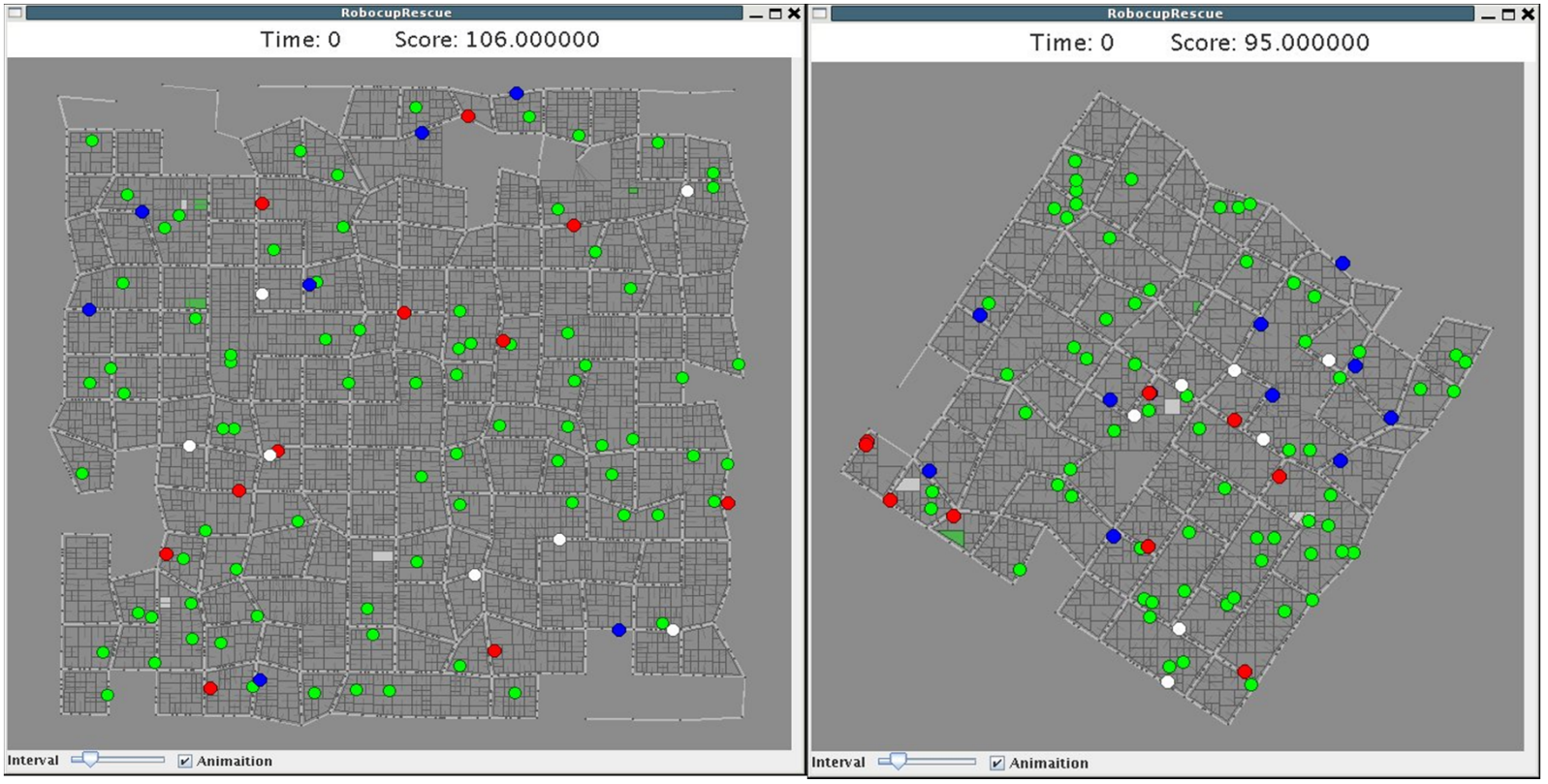
Kobe (1/4)

long distances, loose building layout, sparse bits



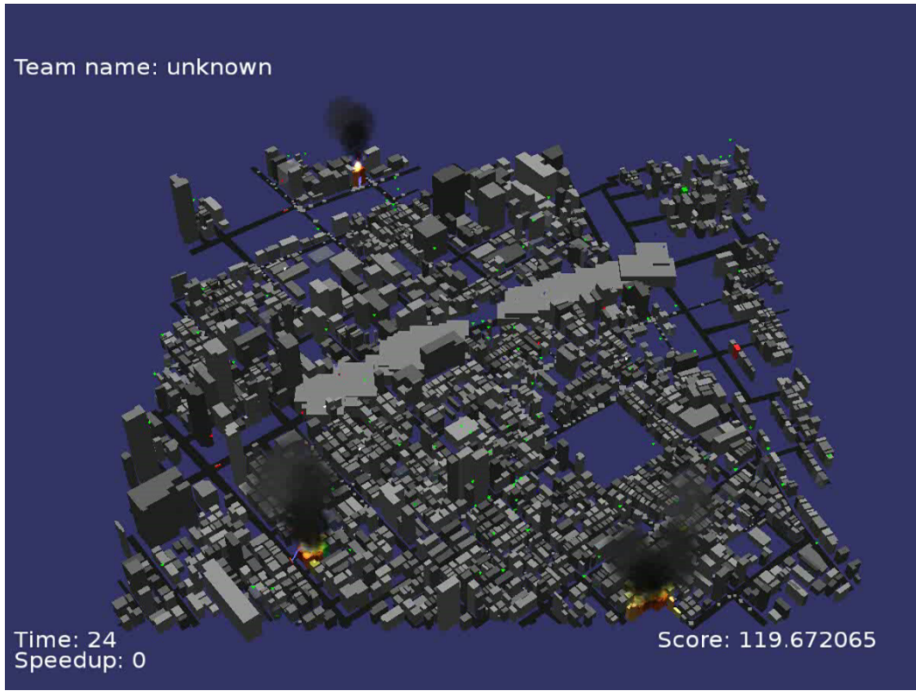
Random Maps

with different building/road densities



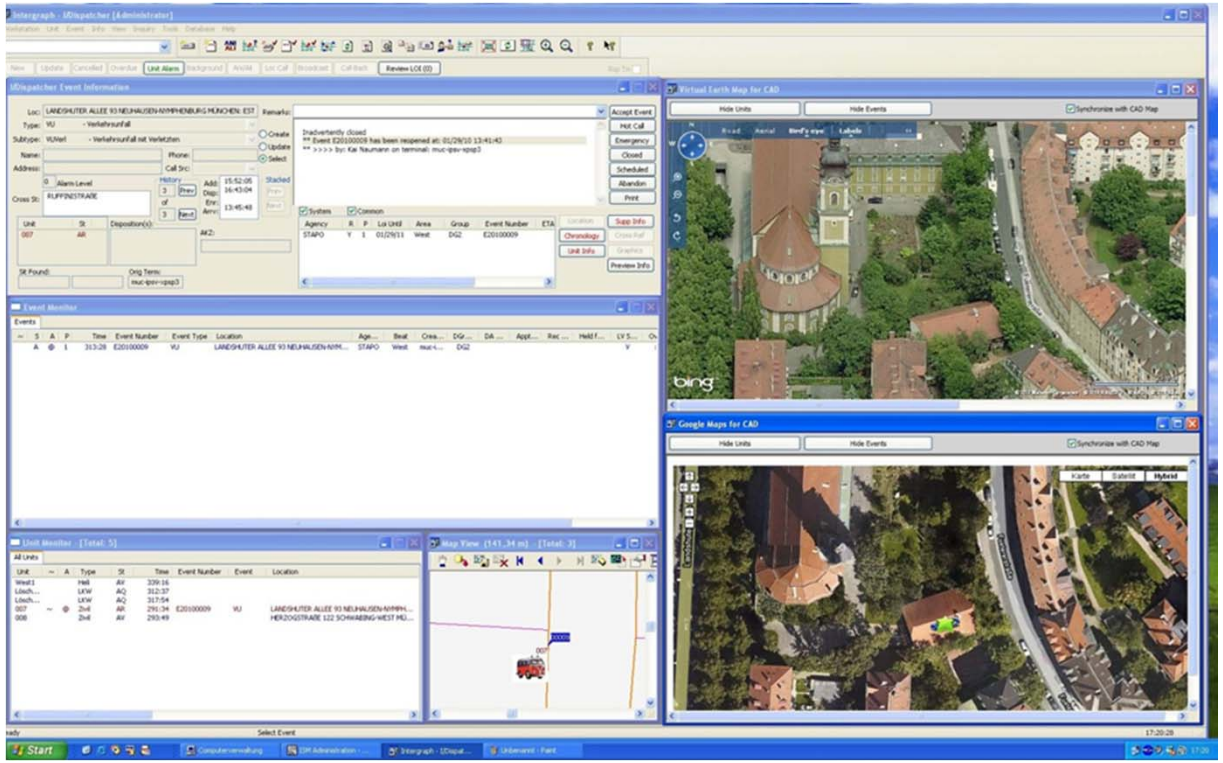
Exploration

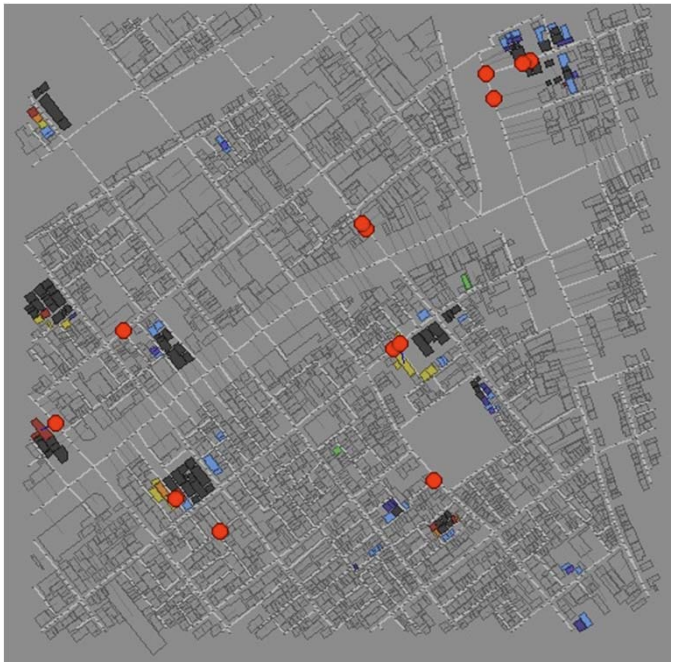
Attacking Fires



Motivation

C2/C4I Systems





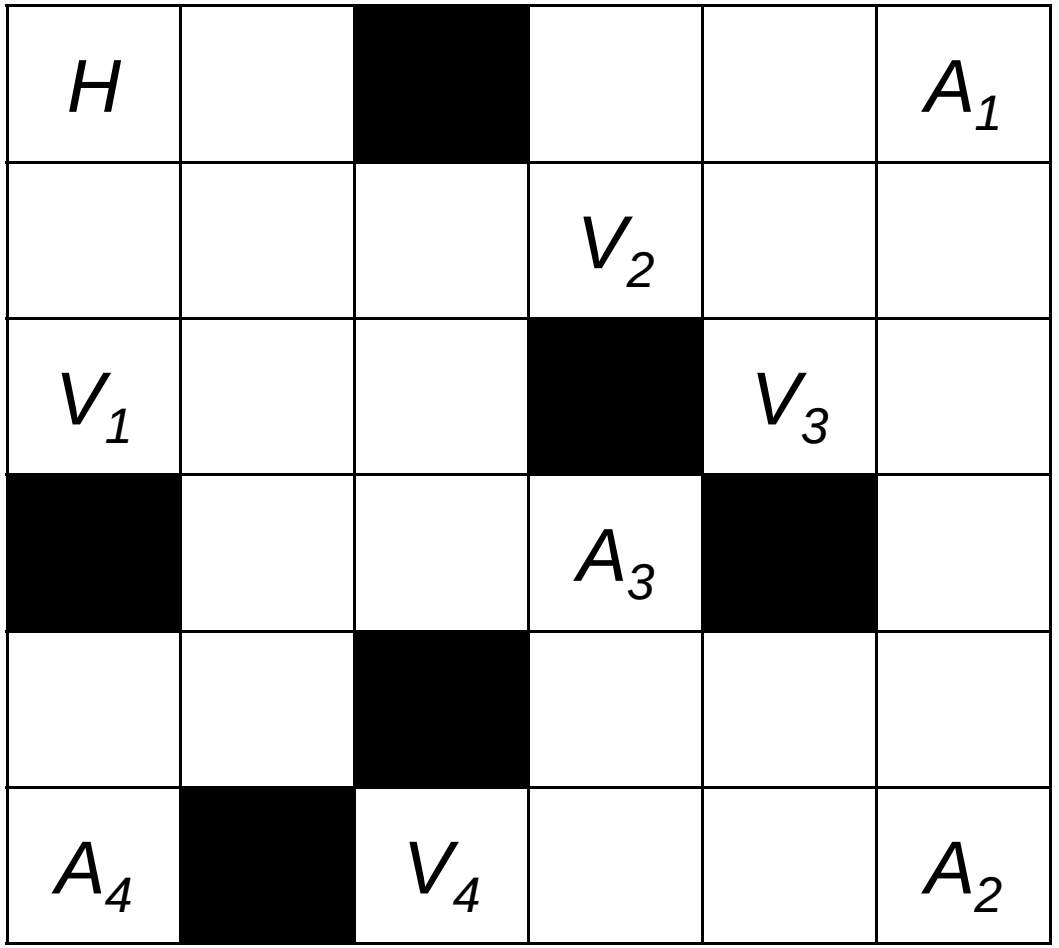
RoboCup Rescue Agent Simulator

Ambulance Assignment

- an assignment task for ambulances is given
- assign ambulances to victims and maximize the number of survived victims
- given:
  - a 6x6 grid world, some of the cells might be blocked
  - a hospital *H* the victims have to be brought to
  - a set of ambulances $<A_1,A_2,A_3,A_4>$, ambulances can move to its neighboring cell in one time step – but not diagonal, a ambulance can fetch a victim if it in the same cell
  - a set of victims $<V_1,V_2,V_3,V_4>$, a victim will die after t_i time steps if it not reaches the hospital
- goal: find an optimal assignment of ambulances to victims

Ambulance Assignment

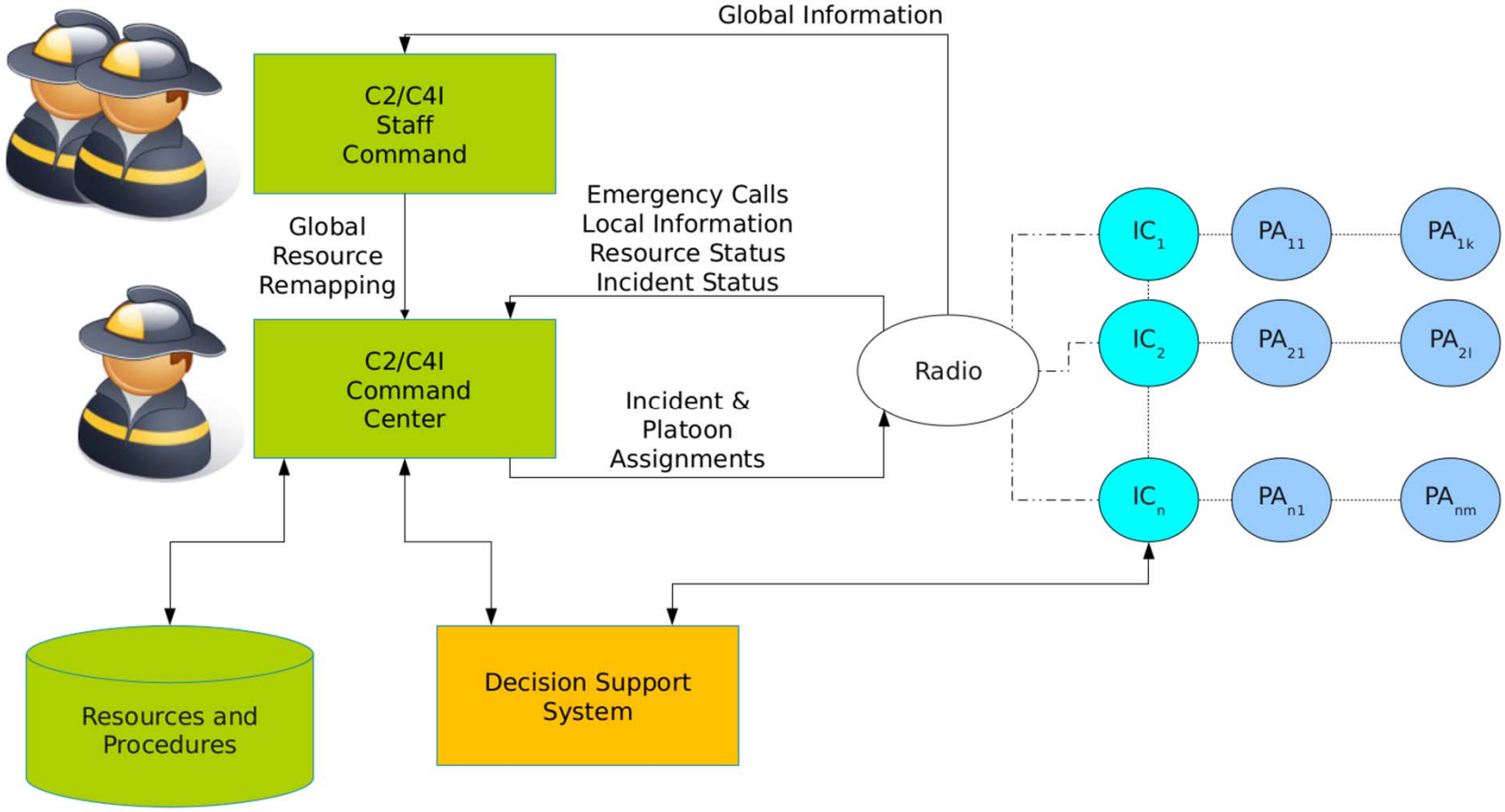
t_1=10
t_2=8
t_3=12
t_4=14



Ambulance Assignment

- formalize the example as optimal assignment problem
- specify a proper objective function
- formulate and solve the optimization using linear programming

Mission Overview

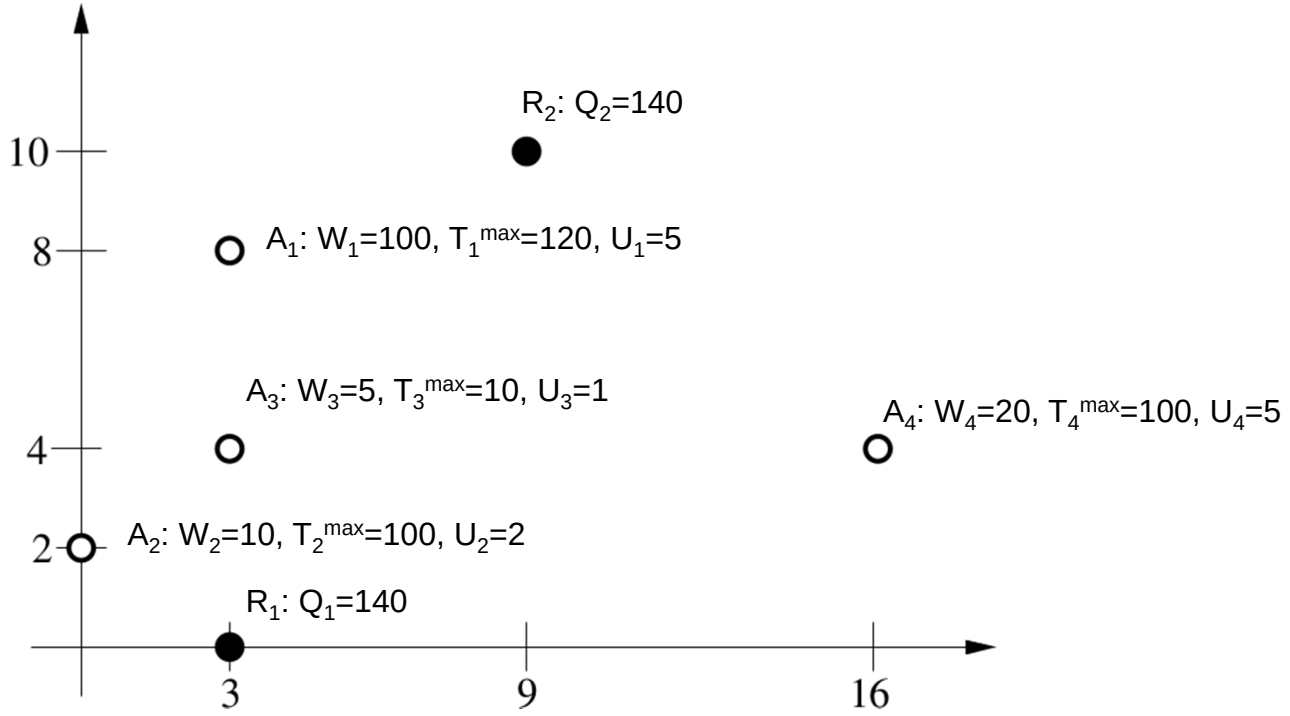


(General) Problem Formulation

- assignment of heterogeneous agents for disaster mitigation
- can serves as interface for systems and algorithms

- Agents: $\mathbf{R}=\{R_1,R_2,\ldots,R_N\}$
- Activities: $\mathbf{A}=\{A_1,A_2,\ldots,A_M\}$
- Agent Capability Sets: C_r with r=1…N
- Activity Requirement Sets: Q_i with i=1…M
- Activity Locations: L_i with i=1…M
- Activity Reward: L_i with i=1…M
- Activity Distances: d_{ij} with i,j=1…M
- Activity Deadlines: T_i^{max} with i=1…M

Simple Example



- A_1 assigned to R_2 with A_1_s=6 and A_1_d=100
- A_2 assigned to R_1 with A_2_s=50 and A_2_d=10
- A_3 assigned to R_2 with A_3_s=110 and A_3_d=5
- A_4 assigned to R_1 with A_4_s=14 and A_4_d=20

Thank You!