# A comprehensive survey of multi-agent reinforcement learning

By: L. Busoniu, R. Babuska, and B. De Schutter

Bin Tang
Computer Science Department
California State University Dominguez Hills

Slides by Christopher Gonzalez

# Outline

- Introduction
- Single-agent RL
- Multi-agent RL
- Our own effort
  - Prize-Collecting Traveling Salesman Problem
    - Virtual Network Function Placement in Cloud/Edge
  - Data Preservation in Basestation-less Sensor Networks

# Introduction to MARL

- A multi-agent system can be defined as a group of autonomous, interacting entities sharing a common environment, which they perceive with sensors and upon which they act with actuators.
- Applications in: robotic teams, distributed control, resource management, collaborative decision support systems, data mining, etc
- Although the agents in a multi-agent system can be programmed with behaviors designed in advance, it is often necessary that they learn new behaviors online, such that the performance of the agent or of the whole multi-agent system gradually improves
- In an environment that changes over time, a hardwired behavior may become unappropriate.
- Many different goals, but two focal points: stability of the agents' learning dynamics, and adaptation to the changing behavior of the other agents.
- The MARL algorithms aim at one of these two goals or at a combination of both, in a fully cooperative, fully competitive, or more general setting

# Challenges for RL in multi-agent Systems

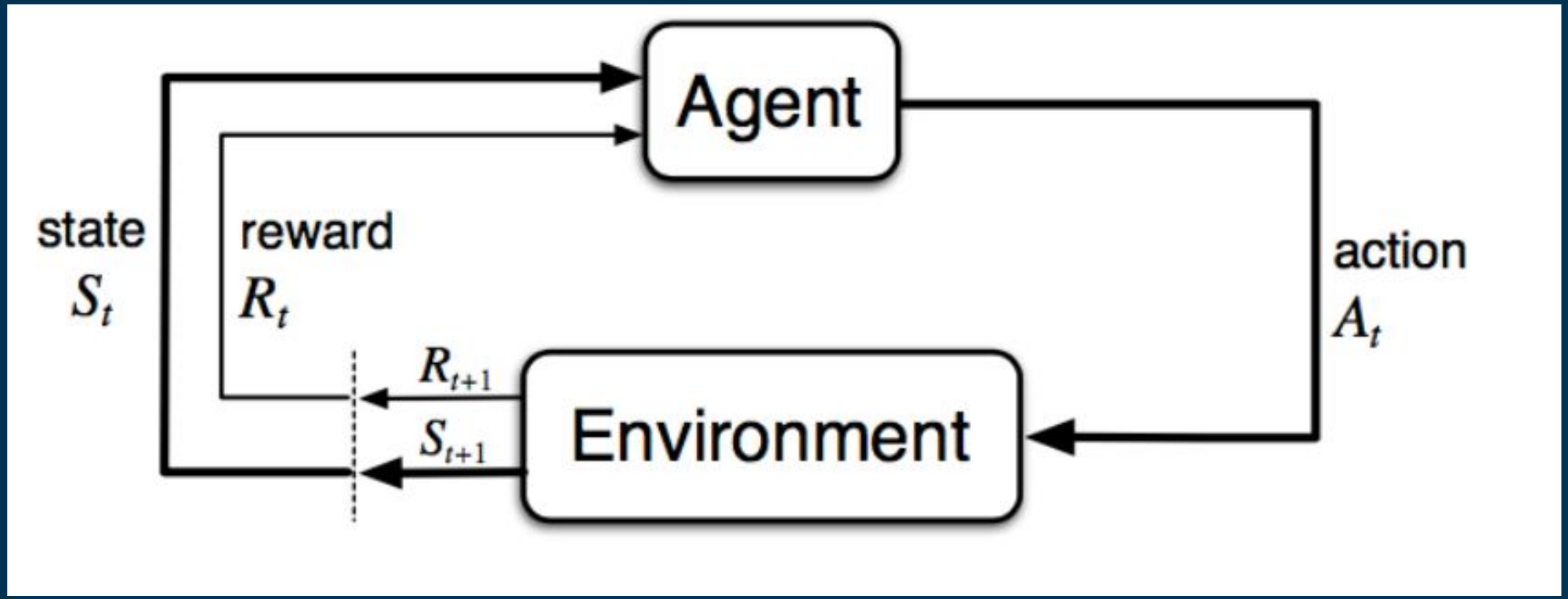
- A reinforcement learning (RL) agent learns by trial-and error interaction with its dynamic environment.
- At each time step, the agent perceives the complete state of the environment and takes an action, which causes the environment to transit into a new state.
- The agent receives a scalar reward signal that evaluates the quality of this transition.
- The difficulty of defining a good learning goal for the multiple RL agents.
- Furthermore, most of the times each learning agent must keep track of the other learning (and therefore, nonstationary) agents.
- Only then will it be able to coordinate its behavior with theirs, such that a coherent joint behavior results.
- How autonomous multiple agents learn to solve dynamic tasks online, using learning techniques with roots in dynamic programming and temporal-difference RL.

# SINGLE-AGENT REINFORCEMENT LEARNING

# Single-agent Reinforcement Learning

- The environment of the agent is described by a Markov decision process, a tuple {X, U, f, ρ} where
  - X is the finite set of environment states,
  - U is the finite set of agent actions,
  - $f : X \times U \times X \to [0, 1]$ is the state transition probability function, and
  - $\rho : X \times U \times X \to R$ is the reward function.
- The behavior of the agent is described by its policy h, which specifies how the agent chooses its actions given the state.
- The policy may be either stochastic: $h : X \times U \to [0, 1]$ or deterministic: $\overline{h} : X \to U$
- A policy is called stationary if it does not change over time

# Markov Decision Process (MDP)



# Single-agent Reinforcement Learning

- The state signal $x_k \in X$ describes the environment at each discrete time step k. The agent can alter the state at each time step by taking action $u_k \in U$.
- As a result of action $u_k$, the environment changes state from x to some $x_{k+1} \in X$ according to the state transition probabilities given by
  - f: the probability of ending up in $x_{k+1}$ given that $u_k$ is executed in $x_k$ is:

$$f(x_k, u_k, x_{k+1})$$

- The agent receives a scalar reward $r_{k+1} \in \mathbb{R}$, according to the reward function ρ:

$$r_{k+1} = \rho(x_k, u_k, x_{k+1})$$

- This reward evaluates the immediate effect of action $u_k$, i.e., the transition from $x_k$ to $x_{k+1}$.

# Single-agent Reinforcement Learning

- The agent's goal is to maximize, at each time step k, the expected discounted return:

$$R_k = E\left\{\sum_{j=0}^{\infty} \gamma^j r_{k+j+1}\right\}$$

  - $R_k$: the accumulated reward in the long run
  - $\gamma \in [0, 1)$ is the discount factor
  - The expectation is taken over the probabilistic state transitions

# Action-Value Function (Q function)

- The action-value function (Q-function), $Q^h : X \times U \to \mathbb{R}$, is the expected return of a state-action pair given the policy $h$: $Q^h(x,u) = E\left\{\sum_{j=0}^{\infty} \gamma^j r_{k+j+1} \mid x_k = x, u_k = u, h\right\}$

- The optimal Q-function is defined as $$Q^*(x,u) = \sum_{x' \in X} f(x,u,x')\left[\rho(x,u,x') + \gamma \max_{u'} Q^*(x',u')\right] \quad (1)$$
  - This equation states that the optimal value of taking u in x is the expected immediate reward plus the expected (discounted) optimal value attainable from the next state (the expectation is explicitly written as a sum since X is finite).
- The greedy policy is deterministic and picks for every state the action with the highest Q-value: $\overline{h}(x) = \arg\max_u Q(x,u)$

# Q-learning RL Algorithm

- Single-agent RL algorithms:
  - Model-based methods based on dynamic programming
  - Model-free methods based on online estimation of value functions
  - Model-learning methods that estimate a model, and then learn using model-based techniques.
- Most MARL algorithms are derived from a model-free algorithm called **Q-learning**
- Q-learning turns (1) into an iterative approximation procedure:

$$Q_{k+1}(x_k, u_k) = Q_k(x_k, u_k) + \alpha_k \left[ r_{k+1} + \gamma \max_{u'} Q_k(x_{k+1}, u') - Q_k(x_k, u_k) \right] \quad (4)$$

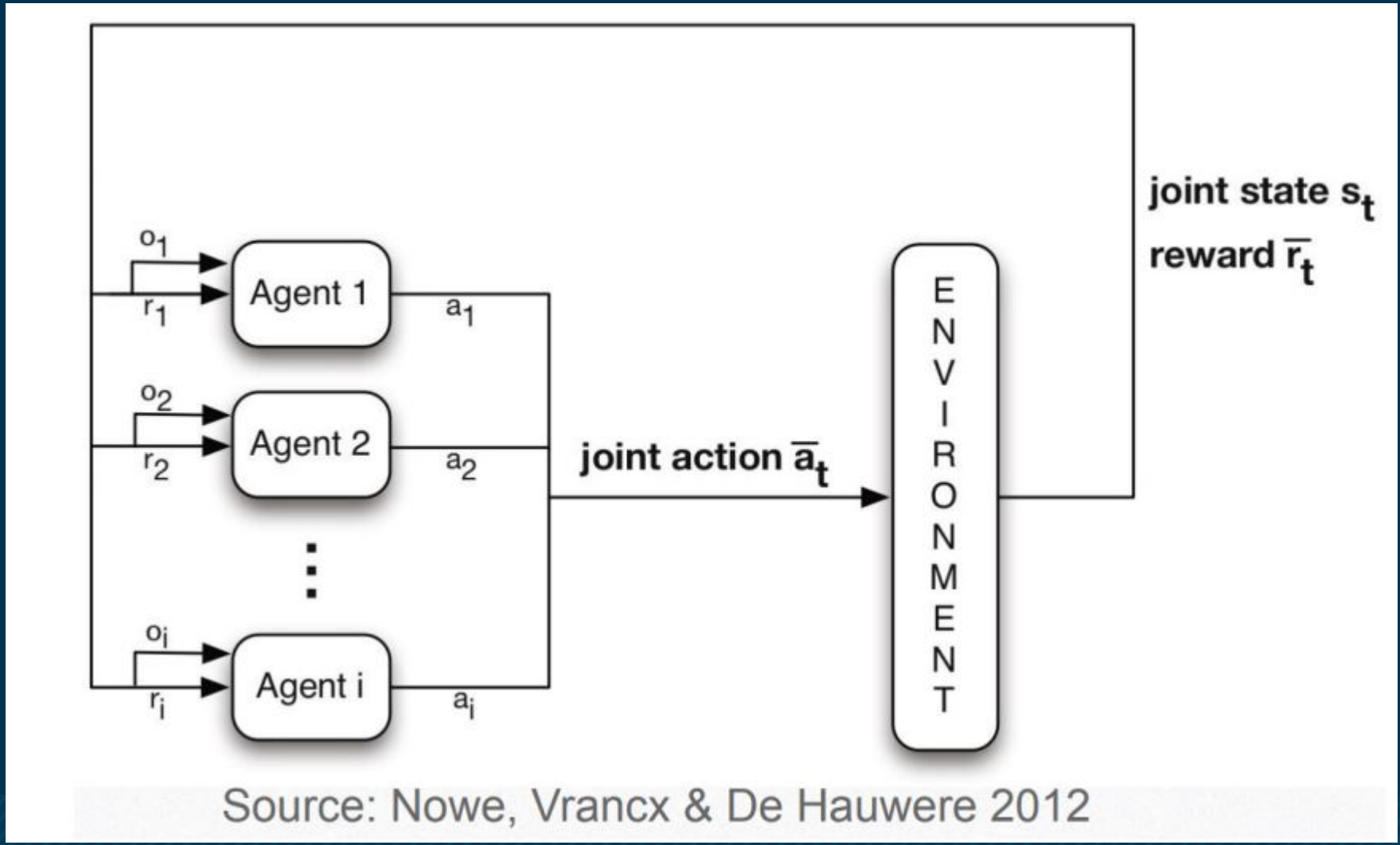
  - Model-free
  - Current estimate of $Q*$ is updated using estimated samples of the r.h.s of (1)
  - These samples are computed using actual experience with the task, in the form of rewards $r_{k+1}$ and pairs of subsequent states $x_k$, $x_{k+1}$:
  - The expression inside the square brackets is the temporal difference, i.e., the difference between estimates of $Q(x_k, u_k)$ at two successive time steps, k + 1 and k

# MULT-AGENT REINFORCEMENT LEARNING (MARL)

# Definition of MARL - Stochastic Game

- Stochastic game (SG): generalization of the MDP to the multi-agent case, tuple {X, U1, . . . , Un, f, ρ1, . . . , ρn}
  - n is the number of agents
  - X is the discrete set of environment states
  - Joint action set: $U = U1 \times \cdots \times Un$
  - State transition probability function: $f : X \times U \times X \rightarrow [0, 1]$ is the, and
  - Reward functions of each agent: $\rho i : X \times U \times X \rightarrow R, i = 1, \ldots, n$
- In the multi-agent case, the state transitions are the result of the joint action of all the agents
- Consequently, the rewards $r_{i,k+1}$ and the returns $R_{i,k}$ also depend on the joint action
- The policies $hi : X \times Ui \rightarrow [0, 1]$ form together the joint policy h.
- The Q-function of each agent depends on the joint action and is conditioned on the joint policy

# B. The multi-agent case




Source: Nowe, Vrancx & De Hauwere 2012

# The multi-agent case

- If $\rho 1 = \cdot \cdot \cdot = \rho n$, all the agents have the same goal (to maximize the same expected return), and the SG is fully cooperative.
- If $n = 2$ and $\rho 1 = -\rho 2$, the two agents have opposite goals, and the SG is fully competitive.
- Mixed games are stochastic games that are neither fully cooperative nor fully competitive.

# Game theory

- Game theory – the study of multiple interacting agents trying to maximize their rewards and especially the theory of learning in games, make an essential contribution to MARL.
- The authors focus on algorithms for dynamic multiagent tasks, whereas most game-theoretic results deal with static (stateless) one-shot or repeated tasks.

# Static, repeated, and stage games

- Many MARL algorithms are designed for static (stateless) games, or work in a stage-wise fashion, looking at the static games that arise in each state of the stochastic game.
- Some game-theoretic definitions and concepts regarding static games are therefore necessary to understand these algorithms
- A static (stateless) game is a stochastic game with $X = \emptyset$. (empty state set)
- Since there is no state signal, the rewards depend only on the joint actions ρi : $U \rightarrow R$
- When there are only two agents, the game is often called a bimatrix game, because the reward function of each of the two agents can be represented as a $|U1| \times |U2|$ matrix with the rows corresponding to the actions of agent 1, and the columns to the actions of agent 2, where $|\cdot|$ denotes set cardinality.
- Fully competitive static games are also called zero-sum games, because the sum of the agents' reward matrices is a zero matrix.
- Mixed static games are also called general-sum games, because there is no constraint on the sum of the agents' rewards.

# Static, repeated, and stage games

- When played repeatedly by the same agents, the static game is called a repeated game.
- The main difference from a one-shot game is that the agents can use some of the game iterations to gather information about the other agents or the reward functions, and make more informed decisions thereafter.
- A stage game is the static game that arises when the state of an SG is fixed to some value.
- The reward functions of the stage game are the expected returns of the SG when starting from that particular state.
- Since in general the agents visit the same state of an SG multiple times, the stage game is a repeated game.
- In a static or repeated game, the policy loses the state argument and transforms into a strategy $\sigma i : Ui \rightarrow [0, 1]$.
- An agent's strategy for the stage game arising in some state of the SG is its policy conditioned on that state value.
- MARL algorithms relying on the stage-wise approach learn strategies separately for every stage game.
- The agent's overall policy is then the aggregate of these strategies.

# Static, repeated, and stage games

- Stochastic strategies (and consequently, stochastic policies) are of a more immediate importance in MARL than in single-agent RL, because in certain cases, like for the Nash equilibrium, the solutions can only be expressed in terms of stochastic strategies
- An important solution concept for static games, is the Nash equilibrium.

- First, define the best response of agent i to a vector of opponent strategies as the strategy σ ∗ i that achieves the maximum expected reward given these opponent strategies:

$$E\{r_i|\sigma_1,\ldots,\sigma_i,\ldots,\sigma_n\} \leq E\{r_i|\sigma_1,\ldots,\sigma_i^*,\ldots,\sigma_n\} \quad \forall\sigma_i \quad (6)$$

# Static, repeated, and stage games- Nash Equilibrium

- A Nash equilibrium is a joint strategy [σ∗1, . . . , σ∗n ] T such that each individual strategy σ∗i is a best-response to the others
- The Nash equilibrium describes a status quo, where no agent can benefit by changing its strategy as long as all other agents keep their strategies constant.
- Any static game has at least one (possibly stochastic) Nash equilibrium; some static games have multiple Nash equilibria.
- Nash equilibria are used by many MARL algorithms reviewed in the sequel, either as learning goal, or both as learning goal and directly in the update rules.

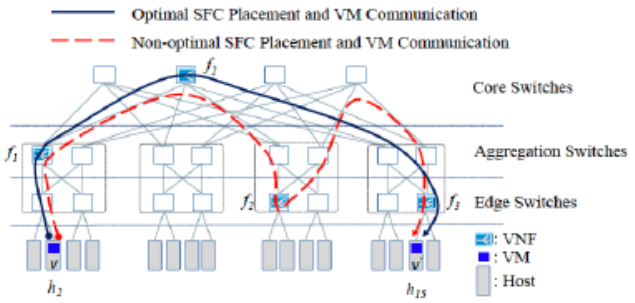
# Our Preliminary Results

# Service Function Chain Placement Problem in Cloud Data Centers

## SFC Placement in Fat-tree Data Centers

- One pair of communicating VMs $(v, v')$, $v$ is at host $s(v)$ and $v'$ at $s(v')$
- SFC with $n$ VNFs: $f_1, f_2, \ldots,$ and $f_n$
- *Place the $n$ VNFs to minimize communication cost:*

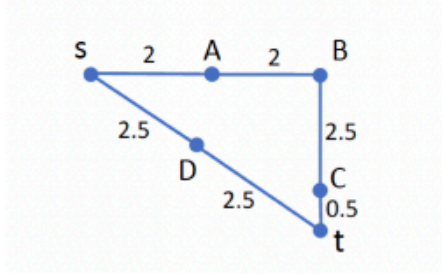
$$C_c(p) = \sum_{j=1}^{n-1} c(p(j), p(j+1)) + \left(c(s(v_i), p(1)) + c(p(n), s(v_i'))\right).$$




7

## SFC Placement is Equivalent to $k$-Stroll Problem

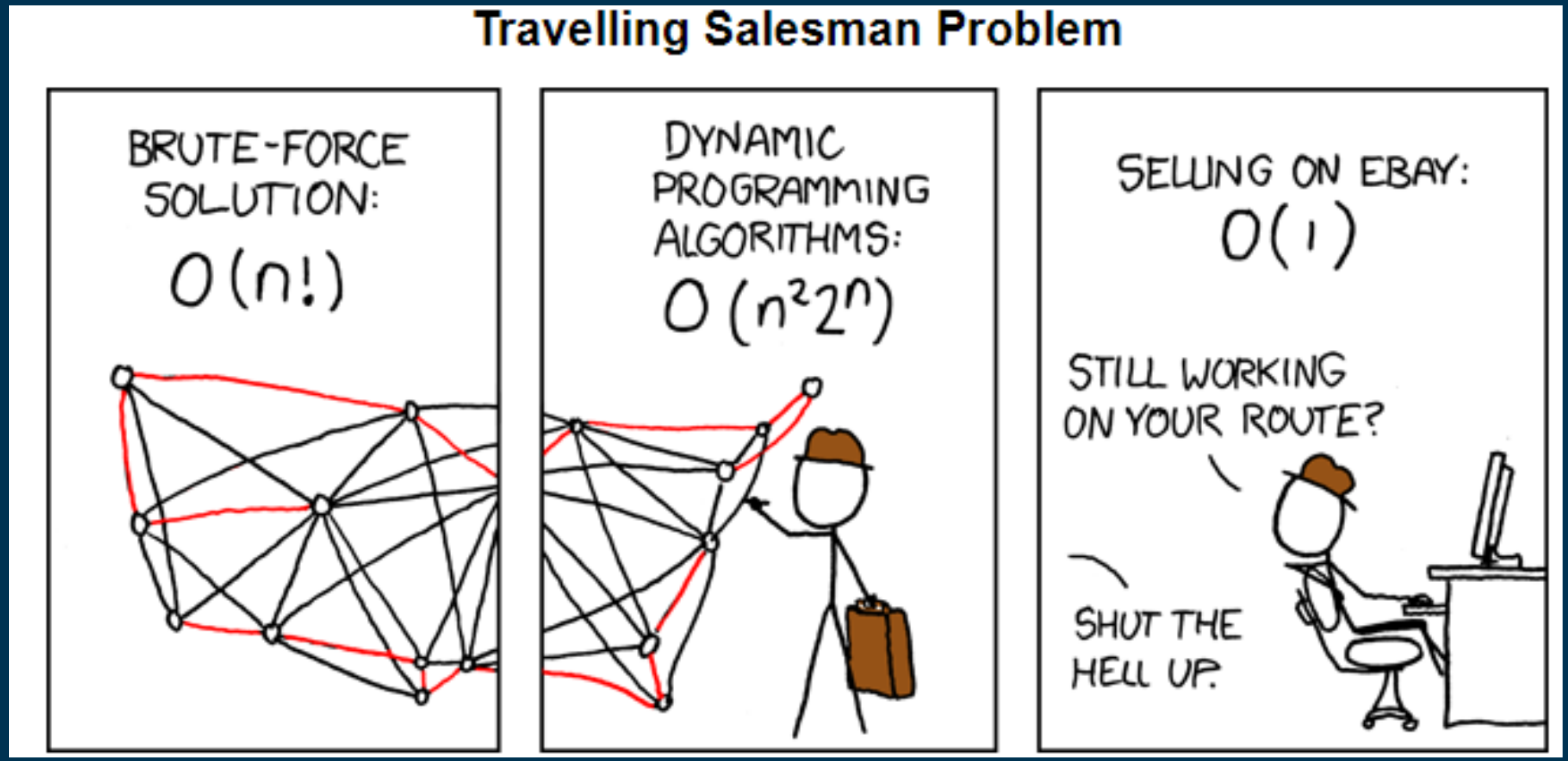
- ***k*-Stroll Problem**
  - Given
    - a weighted undirected graph
    - source $s$ and destination $t$,
    - an integer $k$
  - Goal: find an $s$-$t$ path or walk (i.e., a stroll) of minimum length that visits at least $k$ distinct nodes excluding $s$ and $t$.
- $k$-stroll is NP-hard



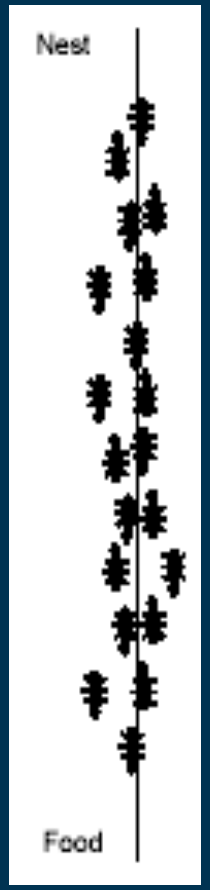

Optimal 2-stroll between $s$ and $t$: $s, D, t, C, t$ with cost 6

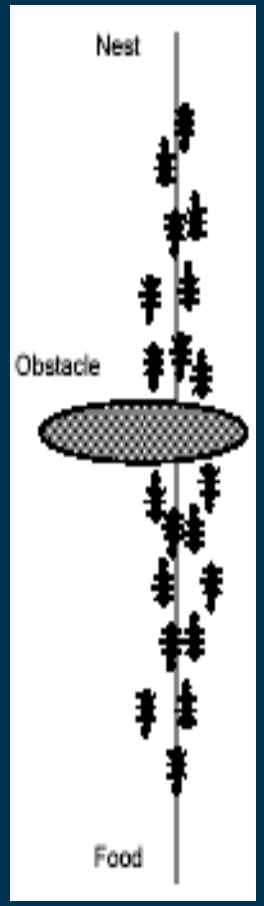
6

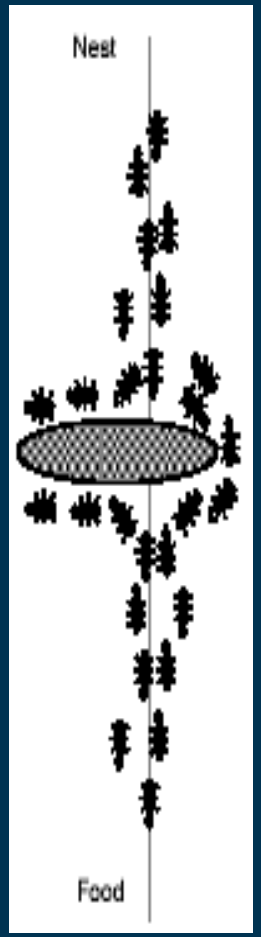
# Prize-Collecting Traveling Salesman Problem

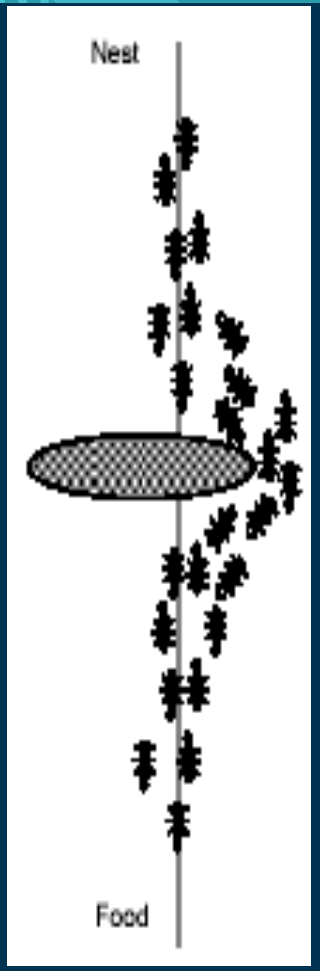


# Click to edit Master title style



# Data Preservation in Basestation-less Sensor Networks

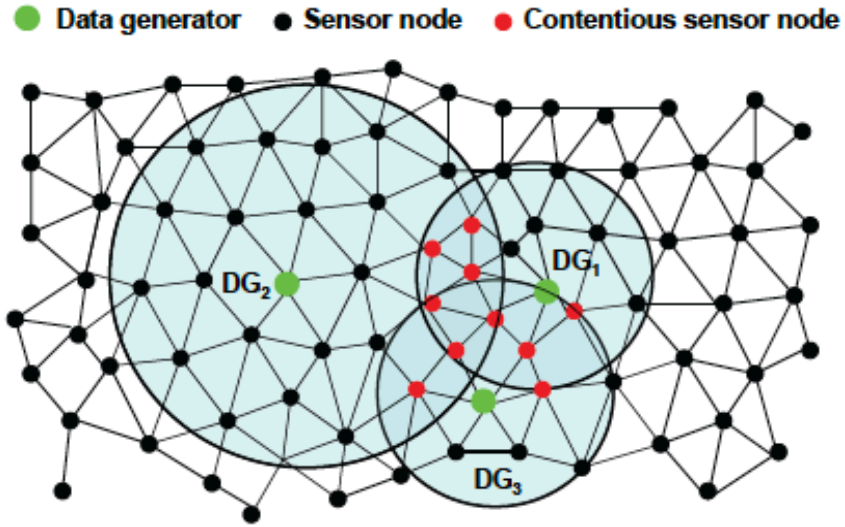



Fig. 1. Data redistribution problem with three data generators $DG_1$, $DG_2$, and $DG_3$. Each shaded circle represents each data generator's offloading area (the set of sensor nodes that are likely to store the offloaded data from data generators). Redistribution contention arises when data generators have overlapping offloading areas – the sensor nodes in those areas are referred to as *contentious sensor nodes*.

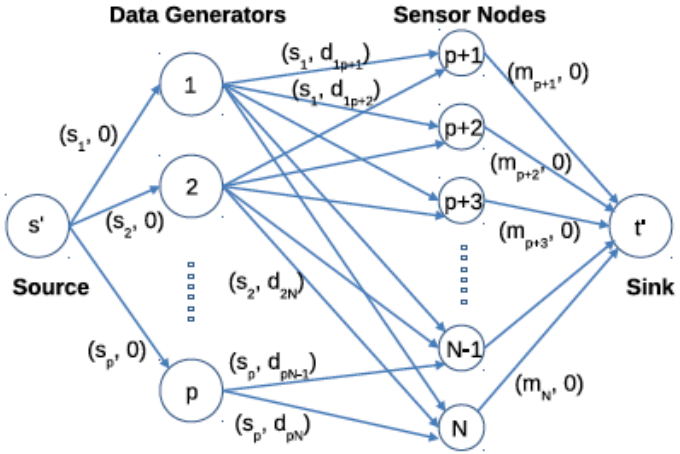



Fig. 3. Data redistribution problem with unit data sizes is equivalent to minimum cost flow problem.

# Thanks!

btang@csudh.edu