Recent Developments in Algorithm Design

Lecture 2: Generalizations and Variants of Set Cover

(Hellerstein)

NYU | TANDON SCHOOL OF ENGINEERING

Recall Linear Program Duality

Linear Program (for minimization problem) in canonical form

- minimize $\sum_{j=1}^{n} c_j x_j$

  such that $\sum_{j=1}^{n} a_{i,j} x_j \geq b_i$ for $i, \ldots, m$

  $x_j \geq 0$ for $j = 1, \ldots, n$

Linear Program (for minimization problem) in canonical form — vector notation version

- minimize $c^T x$

  such that $Ax \leq b$

  $x \geq 0$

Linear Program in canonical form and its dual — vector notation version

- minimize $c^T x$

  such that $Ax \geq b$

  $x \geq 0$

- maximize $b^T y$

  such that $A^T y \leq c$

  $y \geq 0$

Linear Program (for minimization problem) in canonical form, and its dual

- minimize $\sum_{j=1}^{n} c_j x_j$

  such that $\sum_{j=1}^{n} a_{i,j} x_j \geq b_i$ for $i, \ldots, m$

  $x_j \geq 0$ for $j = 1, \ldots, n$

- maximize $\sum_{i=1}^{m} b_i y_i$

  such that $\sum_{i=1}^{m} a_{i,j} y_i \leq c_j$ for $j = 1, \ldots, n$

  $y_i \geq 0$ for $i = 1, \ldots, m$

Terminology

- A solution to an LP is “feasible” if it satisfies all the constraints
- Given a feasible solution to an LP, we say that a constraint of the LP is made “tight” by this solution if it causes the left hand side of the constraint to equal the right hand side

Weak duality

- If $\hat{x}$ is a feasible solution to the primal LP (i.e., it satisfies all the constraints of the primal LP), and $\hat{y}$ is a feasible solution to the dual LP (i.e., it satisfies all the constraints of the dual LP), then

$$c^T \hat{x} \geq b^T \hat{y}$$

Strong duality

- If x^* is an optimal solution to the primal LP and y^* is an optimal solution to the dual LP, then

$$c^T x^* = b^T y^*$$

Primal-Dual Algorithm for Set Cover

Recall Set Cover Problem

- Input: Ground set (universe) $\mathscr{U} = \{e_1, \ldots, e_n\}$ and family of subsets $\mathscr{F} = \{S_1, \ldots, S_m\}$ where each $S_i \subseteq \mathscr{U}$, such that $\bigcup_{i=1}^{m} S_i = \mathscr{U}$

- Task: Find a minimum size subset of $\mathscr{F}$ that covers all the elements of $\mathscr{U}$

LP for Set Cover

- minimize $\sum_{S \in \mathcal{F}} x_S$

  such that $\sum_{S : e \in S} x_S \geq 1 \qquad \forall e \in \mathcal{U}$

  $x_S \geq 0 \qquad \forall S \in \mathcal{F}$

LP for weighted Set Cover

- Each $S \in \mathcal{F}$ has an associated weight $w_S \geq 0$.
  Want minimum weight set cover.

- minimize $\displaystyle\sum_{S \in \mathcal{F}} w_S x_S$

  such that $\displaystyle\sum_{S : e \in S} x_S \geq 1 \qquad \forall e \in \mathcal{U}$

  $x_S \geq 0 \qquad \forall S \in \mathcal{F}$

Primal LP and Dual LP for Set Cover

- Primal LP: minimize $\sum_{S \in \mathcal{F}} w_S x_S$

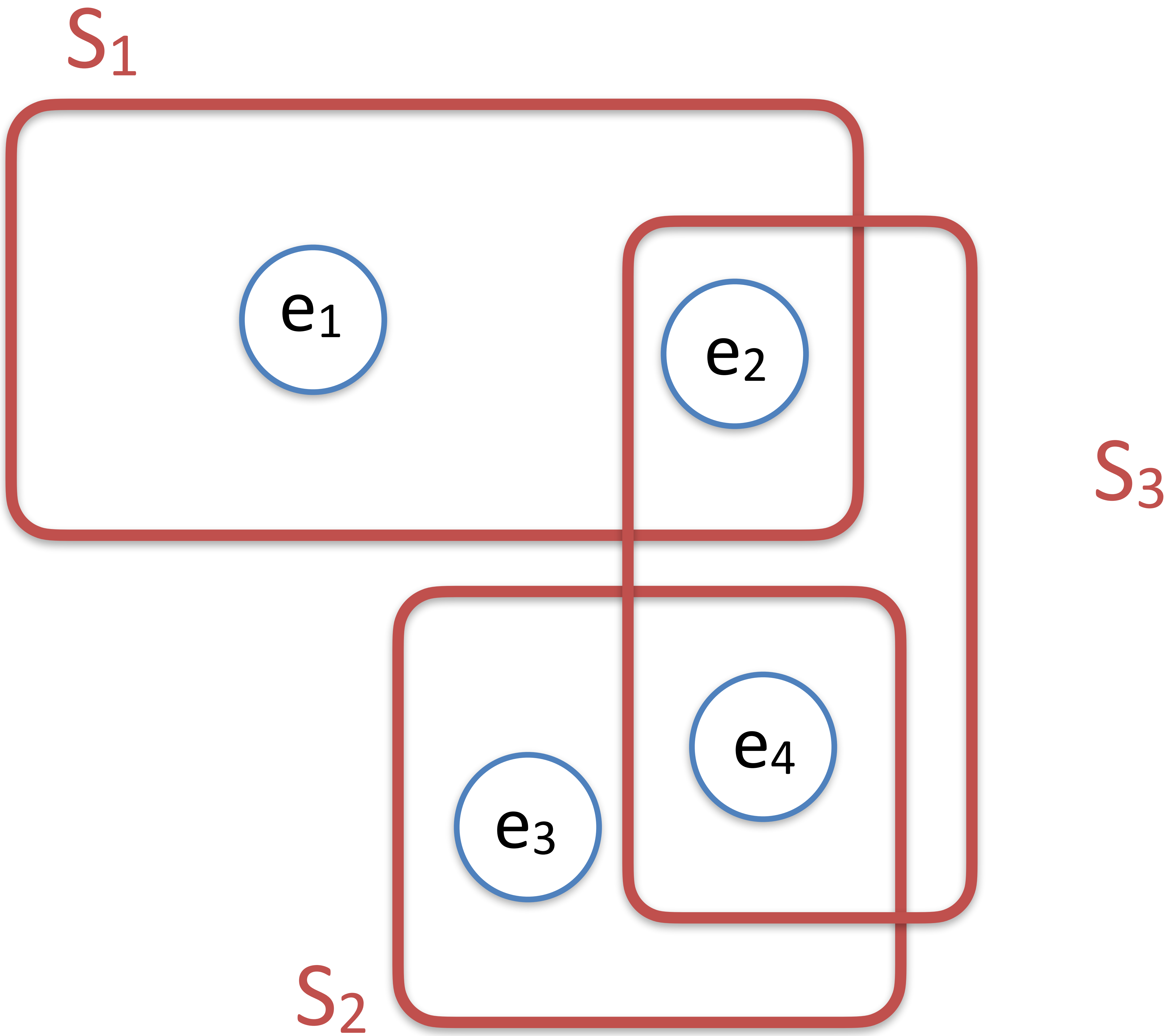
  such that $\sum_{S: e \in S} x_S \geq 1 \qquad \forall e \in \mathcal{U}$

  $x_S \geq 0 \qquad \forall S \in \mathcal{F}$

- Dual LP: maximize $\sum_{e \in \mathcal{U}} y_e$

  such that $\sum_{e: e \in S} y_e \leq w_S \qquad \forall S \in \mathcal{F}$

  $y_e \geq 0 \qquad \forall e \in \mathcal{U}$



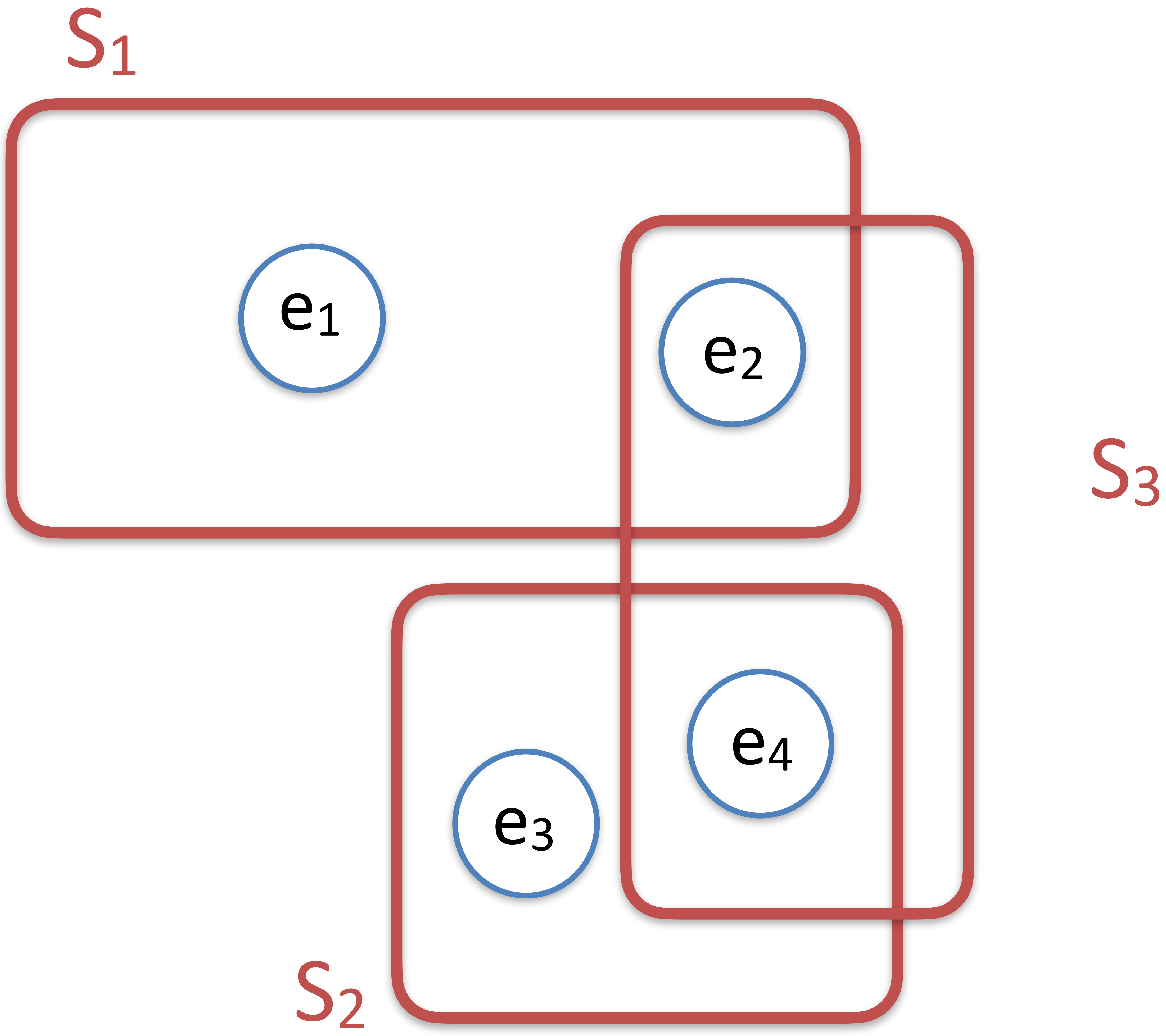
$$w(S_1) = 5, \;\; w(S_2) = 7, \;\; w(S_3) = 6$$

Primal:

min $5x_{S_1} + 7x_{S_2} + 6x_{S_3}$

Dual:

max $y_{e_1} + y_{e_2} + y_{e_3} + y_{e_4}$



$$w(S_1) = 5, \quad w(S_2) = 7, \quad w(S_3) = 6$$

Primal:

$$\textbf{min } 5x_{S_1} + 7x_{S_2} + 6x_{S_3}$$

s.t.

$$x_{S_1} \geq 1$$
$$x_{S_1} + x_{S_3} \geq 1$$
$$x_{S_2} \geq 1$$
$$x_{S_2} + x_{S_3} \geq 1$$
$$x_{S_1}, x_{S_3}, x_{S_3} \geq 0$$

Dual:

$$\textbf{max } y_{e_1} + y_{e_2} + y_{e_3} + y_{e_4}$$

s.t.

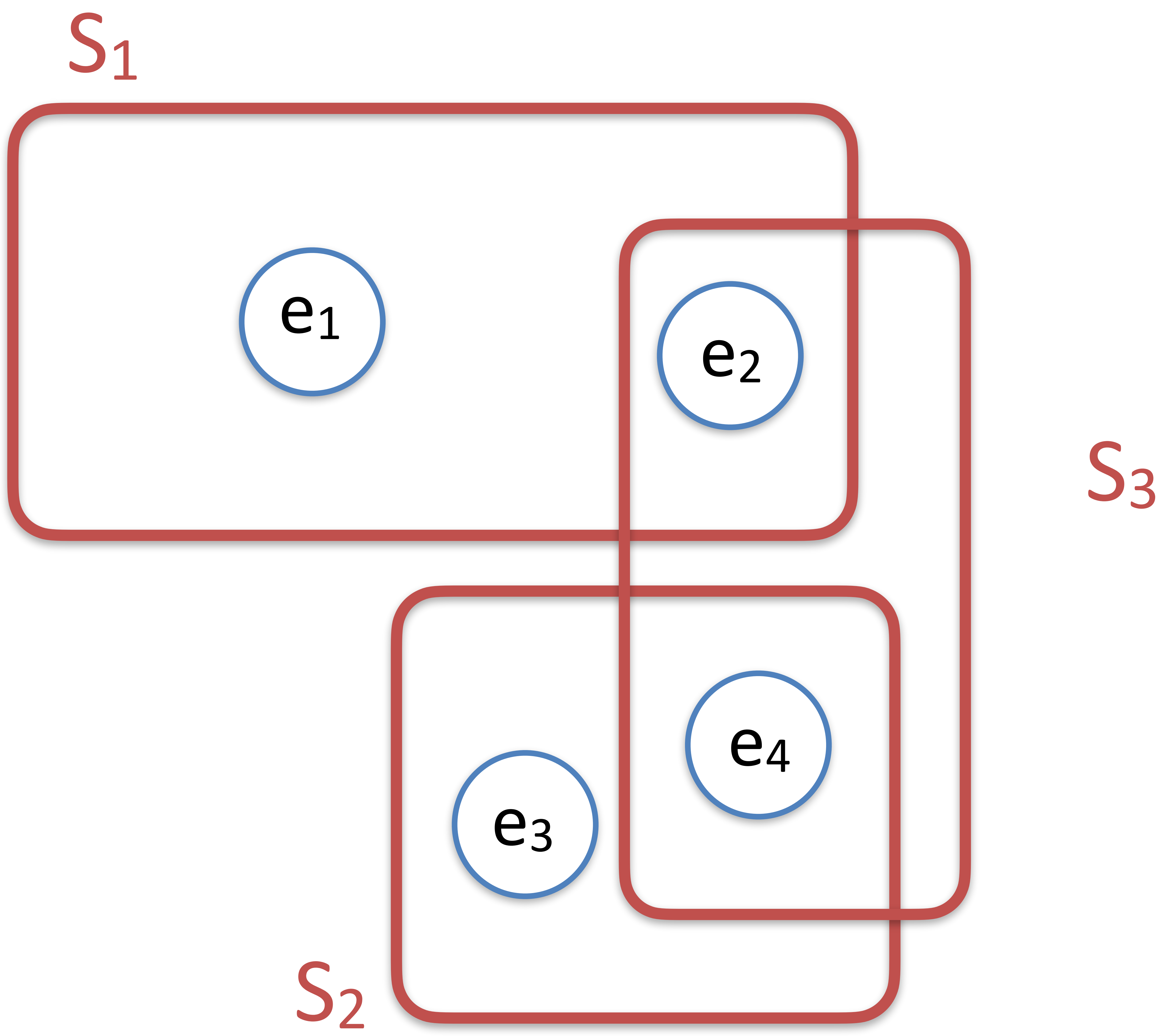
$$y_{e_1} + y_{e_2} \leq 5$$
$$y_{e_3} + y_{e_4} \leq 7$$
$$y_{e_2} + y_{e_4} \leq 6$$
$$y_{e_1}, y_{e_2}, y_{e_3}, y_{e_4} \geq 0$$

Primal-Dual algorithm for set cover

- Another approximation algorithm for (weighted) Set Cover, produces a cover of total weight $f * OPT$, where f is max number of sets containing an element $e \in \mathscr{U}$.
- Simple, combinatorial algorithm, doesn't require using an LP solver to produce an optimal solution to the linear progam
- But LP and its dual used in motivation behind algorithm and in its analysis

Primal Dual algorithm for set cover

- Begin with feasible solution $\hat{y} = 0$ for the Dual LP. (i.e., for all e, $\hat{y}_e$=0 is value assigned to y_e)
- $F' = \emptyset$. \\ We will add subsets $S \in \mathcal{F}$ to F' until subsets in F' cover all the elements in $\mathcal{U}$.
- while there is an element $e \in \mathcal{U}$ such that e not covered by F'
  - Increase $\hat{y}_e$ until some constraint in the dual that contains y_e becomes tight
    \\ if a constraint containing $\hat{y}_e$ already tight, "increase" $\hat{y}_e$ by 0
  - Let S be subset associated with this constraint
  - $F' = F' \cup \{S\}$
- return F'



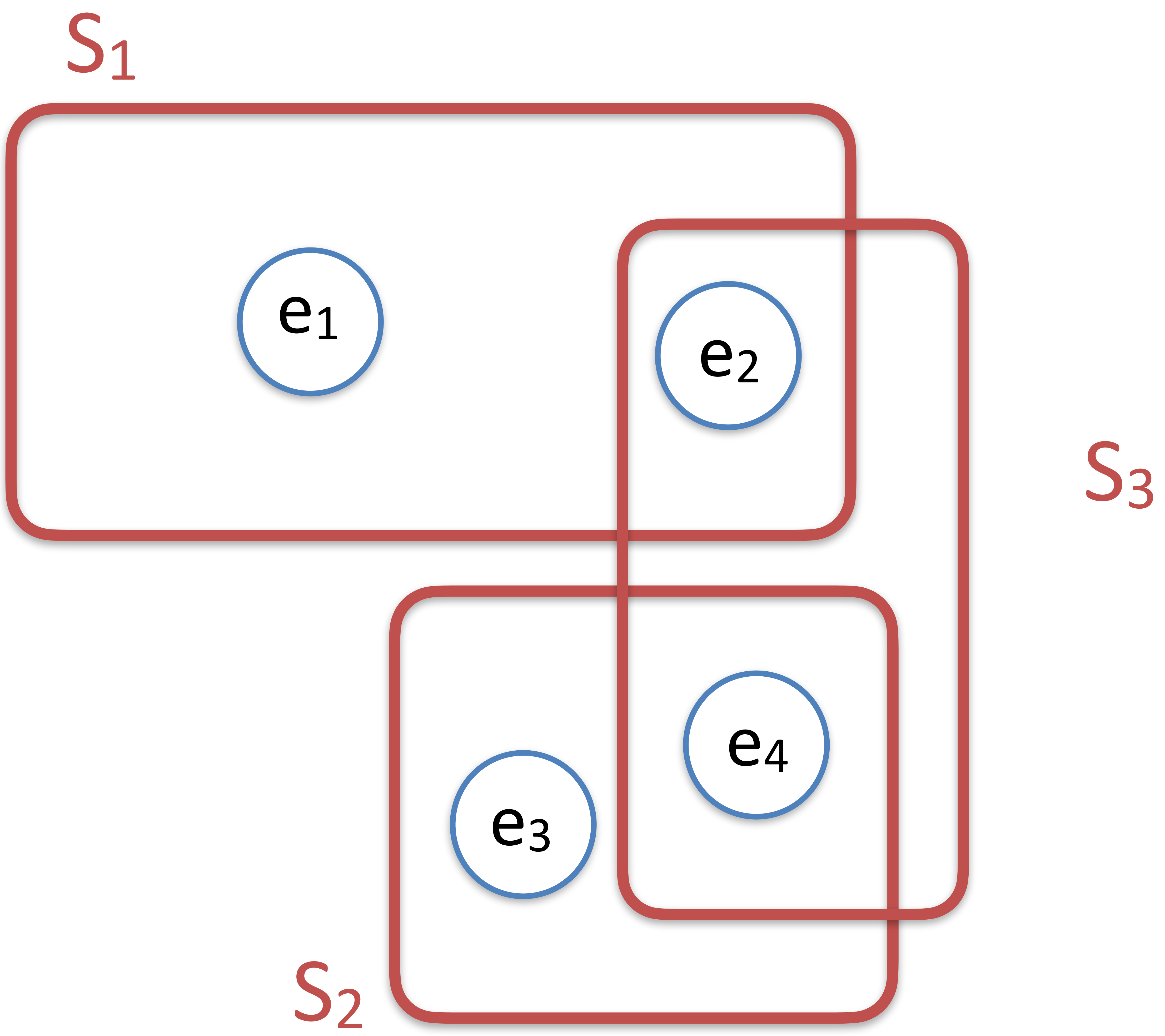
$$w(S_1) = 5, \quad w(S_2) = 7, \quad w(S_3) = 6$$

Dual:

$$\textbf{max } y_{e_1} + y_{e_2} + y_{e_3} + y_{e_4}$$

s.t.

$$y_{e_1} + y_{e_2} \leq 5$$
$$y_{e_3} + y_{e_4} \leq 7$$
$$y_{e_2} + y_{e_4} \leq 6$$
$$y_{e_1}, y_{e_2}, y_{e_3}, y_{e_4} \geq 0$$



$$w(S_1) = 5, \;\; w(S_2) = 7, \;\; w(S_3) = 6$$

$$\hat{y}_{e_1} = \hat{y}_{e_2} = \hat{y}_{e_3} = \hat{y}_{e_4} = 0$$

Increase $\hat{y}_{e_2}$

$$\hat{y}_{e_2} = 5$$

Add S_1 **to cover**

e_3 **and** e_4 **still uncovered**

Increase y_{e_4}

$$\hat{y}_{e_4} = 2$$

Add S_2 **to cover**

Dual:

max $y_{e_1} + y_{e_2} + y_{e_3} + y_{e_4}$

s.t.

$$y_{e_1} + y_{e_2} \leq 5$$
$$y_{e_3} + y_{e_4} \leq 7$$
$$y_{e_2} + y_{e_4} \leq 6$$
$$y_{e_1}, y_{e_2}, y_{e_3}, y_{e_4} \geq 0$$

Analysis of Primal-Dual algorithm for Set Cover

- Thm: Primal-Dual algorithm constructs a set cover of size at most $f \times OPT$, where OPT is the value of the optimal solution (minimum weight set cover), and f is the max number of sets $S \in \mathscr{F}$ covering any element $e \in \mathscr{U}$
- Pf:

  Can show that algorithm successfully produces cover F'

- Consider cover F' constructed by the algorithm, and final value of $\hat{y}$. Total weight of constructed cover F' is $\sum_{S \in F'} w_S$

  For each $S \in F'$, the corresponding dual constraint is tight, $\sum_{e \in S} \hat{y}_e = w_S$, so the total weight of F' is

$$\sum_{S \in F'} w_S = \sum_{S \in F'} \sum_{e \in S} \hat{y}_e$$

$$\leq f \sum_{e \in \mathscr{U}} \hat{y}_e$$ since each $e \in \mathscr{U}$ belongs to at most f sets $S \in \mathscr{F}$

$$\leq f \times LP$$ where LP is the optimal value of the LP, by duality and fact that $\hat{y}$ is feasible solution to Dual LP

$$\leq f \times OPT$$

Greedy Algorithm for Weighted Set Cover

Generalization of (primal) Greedy algorithm for weighted set cover

- Previous greedy algorithm didn't handle weights on the subsets, just needed to minimize size of cover

- Natural generalization of greedy algorithm, using greedy rule:

    - Choose subset S maximizing

$$\frac{\text{number uncovered elements in S}}{w_S}$$

    maximizing “bang-for-the-buck”

- Same approximation factor as for unweighted greedy algorithm, total weight of constructed cover is at most $\ln(|\mathscr{U}|+1)OPT$

Summary of Algorithms for Weighted Set Cover

Summary of presented Weighted Set-Cover Algorithms

	Approximation factor
Primal Dual	f
Greedy	ln U + 1
Solve LP and round deterministically	f
Solve LP, randomized rounding with alteration	ln U + 1 (expected)

Submodular (Set) Cover

Submodular (Set) Cover

- Let V be a finite set. Call the elements of V "items"

- Let $u : 2^V \to \mathbb{R}^{\geq 0}$

  Function u assigns non-negative real value to each subset of items

  We will call u a "utility function"

- Monotonicity:

  Say u is monotone if for all A, B where $A \subseteq B \subseteq V$, $u(A) \leq u(B)$

  Adding additional items to a set can never decrease utility

Submodularity

- Definition:
  A function $u : 2^V \to \mathscr{R}^{\geq 0}$ is said to be *submodular* if for all A, B such that $A \subseteq B \subseteq V$, and $i \in V \backslash B$,
  $$u(A \cup \{i\}) - u(A) \geq u(B \cup \{i\}) - u(B)$$

- Submodularity is sometimes called the *diminishing returns property.* Why?

Equivalent Definition of Submodularity

- Definition:
  A function $u : 2^V \to \mathscr{R}^{\geq 0}$ is said to be *submodular* if for all A, B such that $A, B \subseteq V$, and $i \in V \backslash B$,
  $$u(A \cup B) + u(A \cap B) \leq u(A) + u(B)$$

Coverage functions

- Recall that an instance of the set cover problem is a set system consisting of universe $\mathscr{U}$ and a family $\mathscr{F}$ of subsets of $\mathscr{U}$
- Consider the utility function $u : 2^V \to \mathscr{R}^{\geq 0}$ where $V = \mathscr{F}$ and for all $F \subseteq \mathscr{F}$
  $u(F) = |\bigcup_{S \in F} S| =$ # elements of $\mathscr{U}$ covered by the sets S in F
- This type of utility function, based on a set system, is called a "coverage function"
- The "items" of the coverage function correspond to the subsets S in $\mathscr{F}$ (each item is a subset of elements of universe $\mathscr{U}$)
- If u is a coverage function, it is
  - monotone
  - submodular
  - has the property that $u(\emptyset) = 0$

Other examples of submodular functions

- Rank function of a vector space
  - V = set of all vectors in a vector space, and $u(V') =$ rank of space spanned by vectors in V'
- Modular (additive) functions
  - Each item $i \in V$ has an associated real weight w_i and $u(V') = \sum_{i \in V} w_i$

    If weights are non-negative, then u is monotone.
- "Budget additive" functions
  - Each item $i \in V$ has an associated weight $w_i \geq 0$ and for some $B \geq 0$, $u(V') = \min\{B, \sum_{i \in V} w_i\}$
- Non-monotone example: Graph cut functions
  - $V =$ set of all vertices in a graph G, $u(V')$ = number of edges (u, v) of G with $u \in V'$, $v \in V \backslash V'$

Generalization of set cover: Submodular Cover

- Problem: Given a finite set V, a utility function $u : 2^V \to \mathbb{Z}^{\geq 0}$ such that u is monotone, submodular, and satisfies $u(\emptyset) = 0$, find a minimum size subset $V' \subseteq V$ such that $u(V') = u(V)$
- Notes:
  - Problem statement doesn't mention how function u is given as input. We assume that it is given by an oracle that, given as input a subset $V' \subseteq V$, will return $u(V')$ in constant time.
  - Call $V' \subseteq V$ such that $u(V') = u(V)$ a "cover" of u
  - Set Cover is the special case of this problem where u is a coverage function (where the answers to the oracle queries are easily computed from the given set system)
  - Can extend to real-valued utility functions but results get a little messier

Generalization of Max (Set) Coverage Problem: Submodular Max Coverage Problem

- Problem: Given a finite set V, a utility function $u : 2^V \to \mathbb{R}^{\geq 0}$ such that u is monotone, submodular, and satisfies $u(\emptyset) = 0$, and an integer $k \geq 0$ such that $k \leq |V|$, find a subset $V' \subseteq V$ of size k that maximizes $u(V')$

- Greedy algorithm and its analysis almost same as they were for Max (Set) Coverage.

Greedy algorithm for Submodular Max Coverage problem

- Input: Finite set V of items, oracle for utility function $u : 2^V \to \mathscr{R}^{\geq 0}$, such that u is monotone, submodular, and $u(\emptyset) = 0$

- $V'_0 = \emptyset$ \\\\ V'_t will contain the first t items chosen

- for $t = 1$ to k

  - using oracle for u, find an item $i \in V \backslash V'_{t-1}$ that maximizes $u(V'_{t-1} \cup \{i\}) - u(V'_{t-1})$
    call this item $i*$ \\\\ $i*$ is the greedy choice

  - $V'_t = V'_{t-1} \cup \{i*\}$

- return V'_k

Analysis of Greedy Algorithm for Submodular Max Coverage problem

- Let $V*$ be an optimal solution, a set of k items that maximizes $u(V*)$. Let $OPT = u(V*)$
- At start of t-th greedy step, think of $OPT - u(V_{t-1})$ as the remaining distance to OPT
- In t-th greedy step, greedily choose item to add to V'_{t-1}
- If you added all the elements in $V*$ to V'_{t-1}, you'd reach utility value OPT, so increase in utility would be $OPT - u(V'_{t-1})$.
    - By submodularity and monotonicity of u, can show there must be an element of $V*$ that when added to V'_{t-1}would increase utility by at least $\frac{1}{|V*|} = \frac{1}{k}$ of the remaining distance to OPT $\Rightarrow$ after adding t-th greedy element, remaining distance to goal, $OPT - u(V_t) \leq (1 - \frac{1}{|OPT|}) * (OPT - u(V_{t-1}))$
- Inductively: $OPT - u(V'_k) \leq (1 - \frac{1}{k})^k OPT \ \leq \frac{1}{e} OPT \ \Rightarrow u(V'_k) \geq (1 - \frac{1}{e}) OPT$
- Therefore, Greedy outputs a solution that achieves utility at least $(1 - \frac{1}{e}) OPT$

Greedy algorithm for Submodular Cover

- Input: Finite set V of items, oracle for utility function $u : 2^V \rightarrow \mathscr{R}^{\geq 0}$, such that u is monotone, submodular, and $u(\emptyset) = 0$
- t=0
- $V'_t = \emptyset$ \\ V'_t contains items already put into V' at end of step t of greedy algorithm
- while $u(V'_t) \neq u(V)$
  - $t = t + 1$
  - using oracle for u, find an item $i \in V \backslash V'_{t-1}$ that maximizes $u(V'_{t-1} \cup \{i\}) - u(V'_{t-1})$
    call this item $i*$ \\ $i*$ is the greedy choice
  - $V'_t = V'_{t-1} \cup \{i*\}$
- return V'_t

Analysis of Greedy Algorithm for Submodular Cover

- Let OPT be the minimum size of a cover of u, i.e., min size of a subset $V*$ such that $u(V*) = u(V)$
- At start of t-th greedy step, think of $u(V) - u(V_{t-1})$ as the remaining distance to the goal utility value, $u(V)$
- As in analysis of max-coverage can show that inductively,
$$u(V) - u(V_t') \leq (1 - \frac{1}{OPT})^t u(V) \ < e^{-\frac{t}{OPT}} u(V)$$
- Setting $t = \lceil OPT \ln u(V) \rceil$, we get $u(V) - u(V_t') < 1$, which implies that $u(V_t') = u(V)$, since u is an integer valued function
- Therefore, Greedy algorithm runs for at most $\lceil OPT \ln u(V) \rceil \leq OPT(\ln u(V) + 1)$ steps, so its output solution has size at most $OPT(\ln u(V) + 1)$

Sequencing problems

Sequencing problems

- Problems where we want to find the optimal order in which to do something
- Examples:
  - Traveling salesman problem
    - Find the “best” permutation of vertices, given graph with weighted edges
    - NP-hard problem
    - polytime 1.5-approximation algorithm (assuming weights obey triangle inequality)

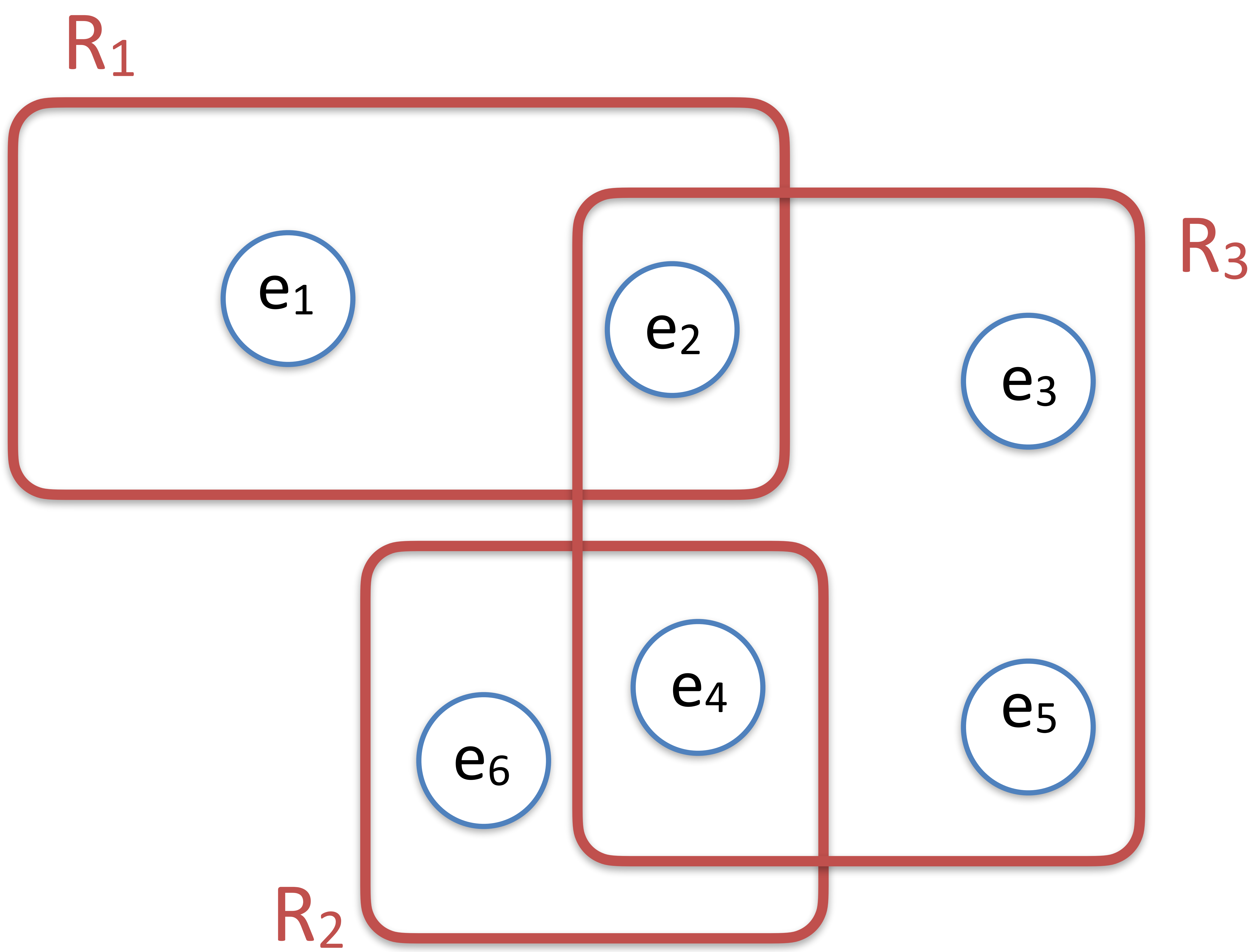
- Scheduling problems, e.g.. Min-sum completion time problem:
    - Set of n “jobs” to be scheduled on a single “processor”
    - Processor can only process one job at a time
    - Given the length ℓ_j of job j for j=1,…,n
    - Find permutation of jobs that minimizes the sum of completion times of the jobs (same as minimizing the average completion time)
    - Optimal solution is to order jobs in increasing length
    - If jobs have weights w_j , and want to minimize weighted sum of completion times
        - exercise: determine optimal solution for this case

Min-Sum Set Cover

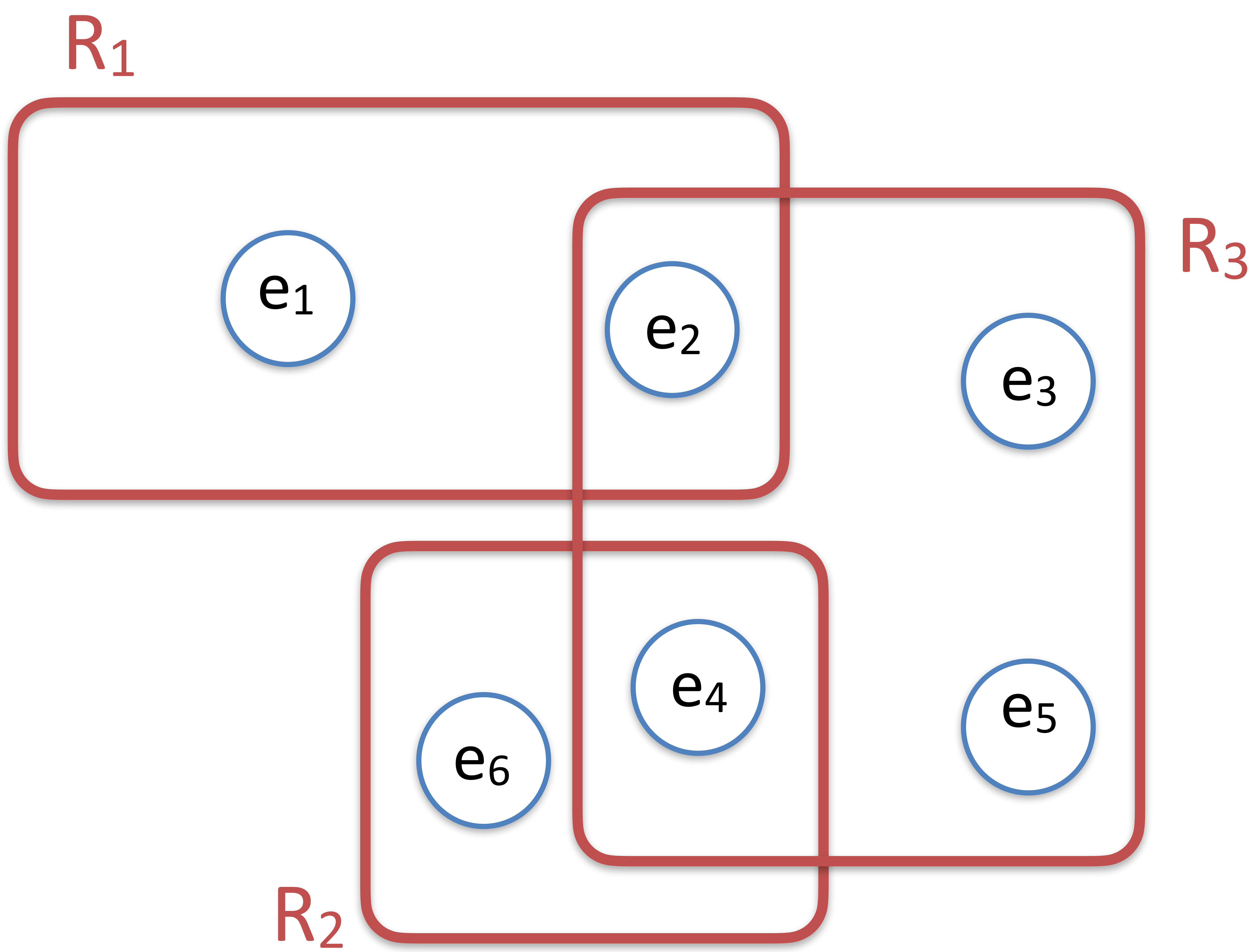
Min-Sum Set Cover [Lovasz et al. 02]

- Input: Ground set (universe) $\mathscr{U} = \{e_1, \ldots, e_m\}$ and family of subsets $\mathscr{F} = \{R_1, \ldots, R_n\}$ where each $R_i \subseteq \mathscr{U}$, such that $\bigcup_{i=1}^{n} R_i = \mathscr{U}$

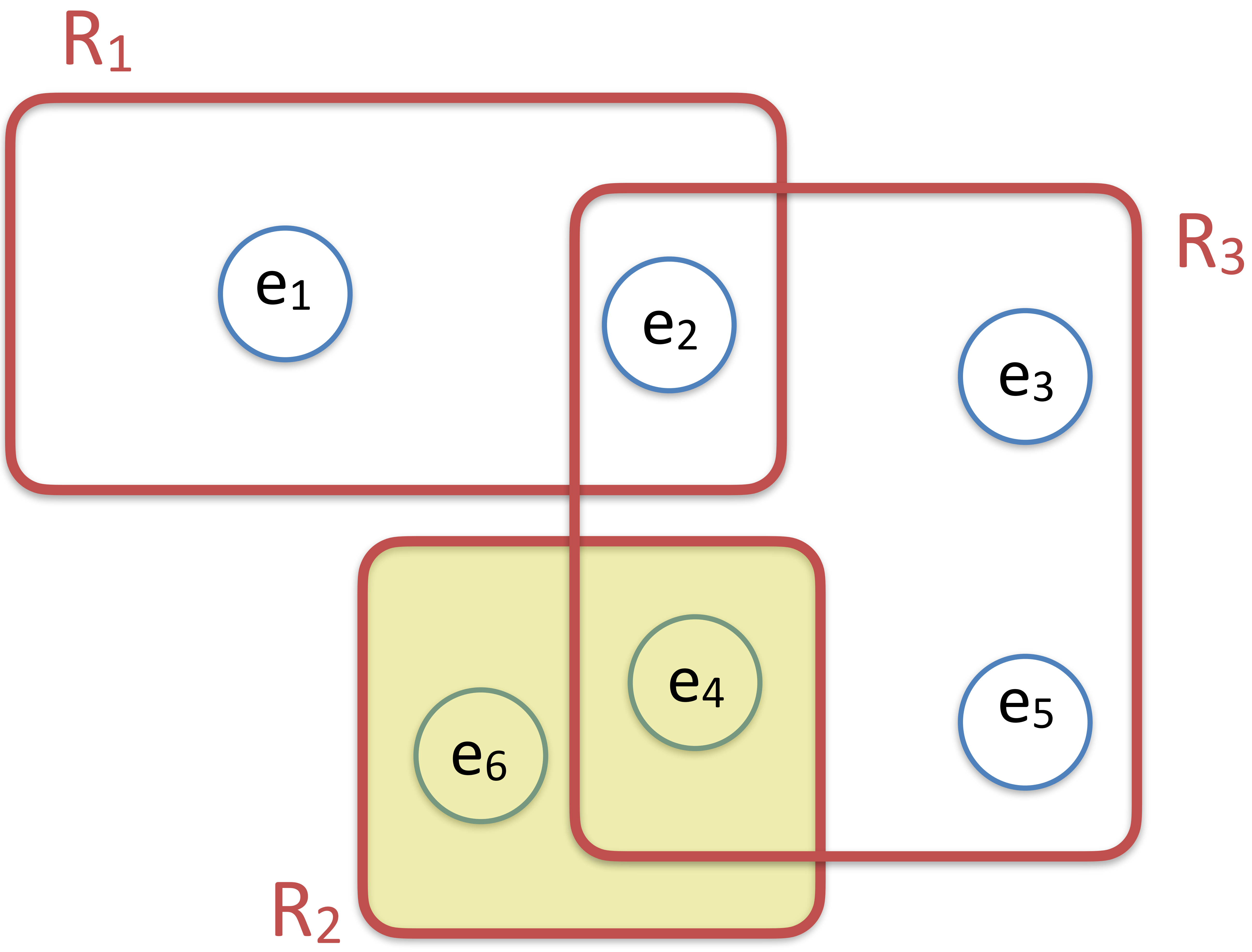
- Task: Find the permutation of the subsets that minimizes the sum of the *covering times* of the ground elements
  - If an element is covered by the jth element in the permutation, we say that it is covered at time j



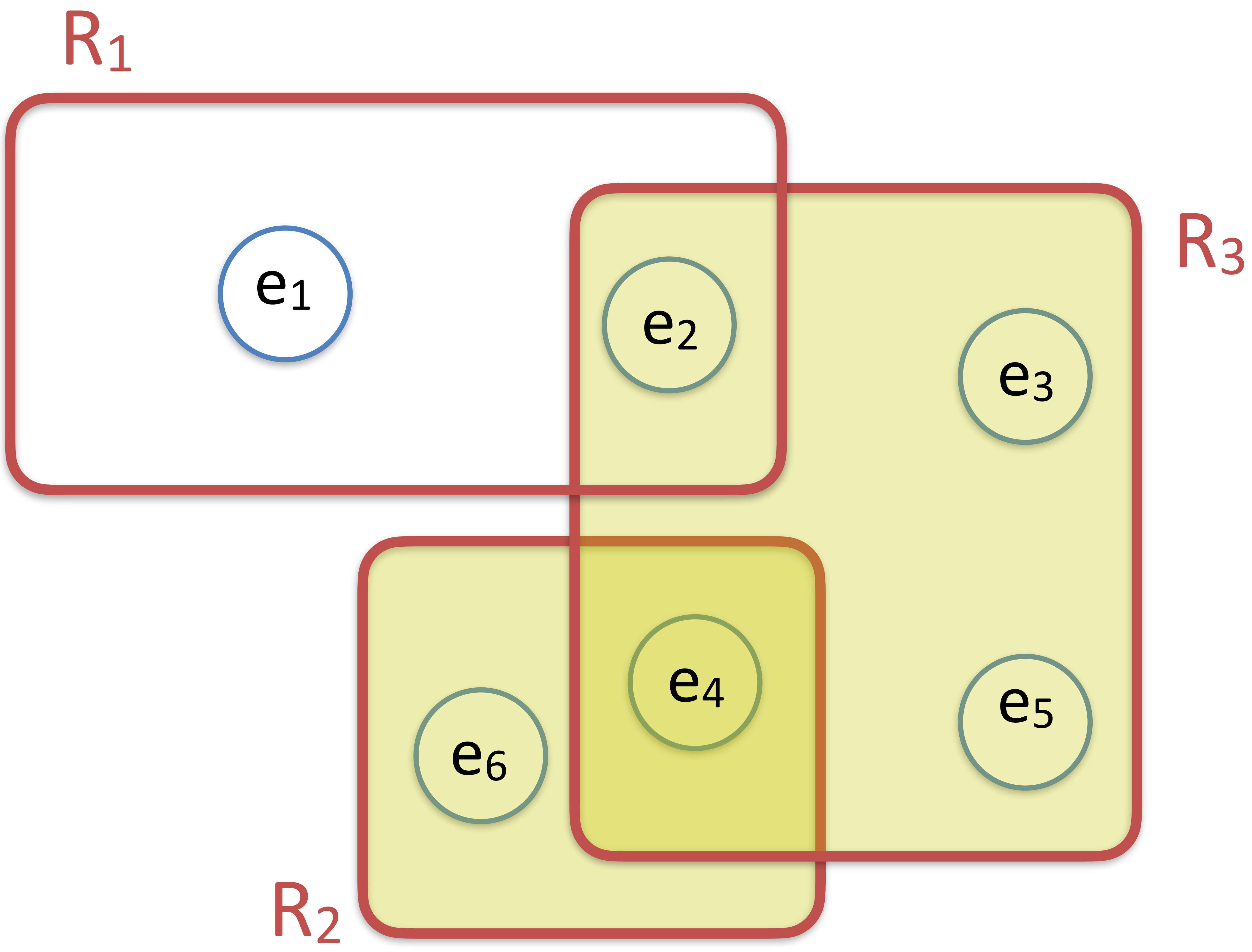
R_1
R_3
R_2
e_1
e_2
e_3
e_4
e_5
e_6



R_2, R_3, R_1

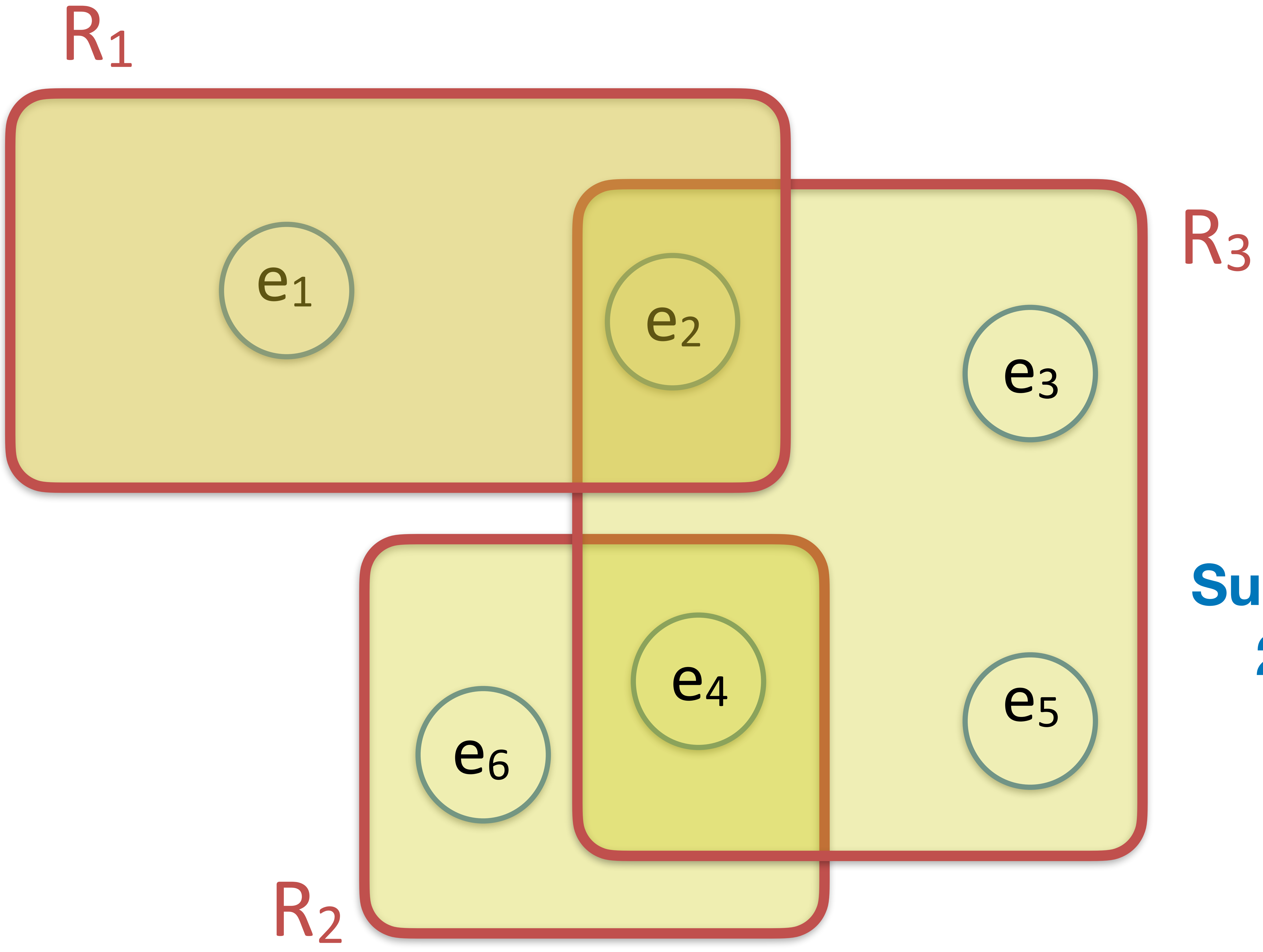


R_2, R_3, R_1 covers e_4, e_6 with R_2 (at time 1)



R_2, R_3, R_1 covers e_4, e_6 with R_2 (at time 1)

then e_2, e_3, e_5 with R_3 (at time 2)



Sum of cover times is
2*1+3*2+1*3 = 11

R_2, R_3, R_1 covers e_4, e_6 with R_2 (at time 1)

then e_2, e_3, e_5 with R_3 (at time 2)

then e_1 with R_1 (at time 3)

Greedy Algorithm

- Greedy Rule: Choose subset that covers the maximum number of uncovered elements
- Same greedy algorithm we used for “classical” set cover problem.
- What approximation factor does it achieve for Min-Sum Set Cover problem?

Approximation factor for Greedy Algorithm applied to Min-Sum Set Cover?

- Consider “bad” instance for greedy algorithm applied to Classical Set Cover. Greedy algorithm doesn’t do so badly!

Greedy Algorithm

- Thm [Lovasz et al. 02] :
  Greedy Algorithm is a 4-approximation algorithm for Min-Sum Set Cover

Proof based on histograms
(next lecture)