This week: Section 9.2.1–9.2.3 of Dasgupta plus some extensions.

1. Vertex Cover
   - Algorithm A: maximal matching
   - Algorithm B: LP-rounding (not in Dasgupta)
   - Generalization to Set Cover (not in Dasgupta)
2. k-Clustering: Greedy Algorithm
3. The Traveling Salesman Problem (TSP)
   - Complexity of TSP
   - Double tree algorithm
   - Nearest addition algorithm (not in Dasgupta)
   - Christofides' algorithm (not in Dasgupta)

Approximation Algorithms

Definition 1. *An α-approximation algorithm for an optimization problem is a polynomial-time algorithm that, for each instance of the problem, produces a solution with a value that is within a factor α of the optimal value.*

For an instance I, we deonte by $\textsc{Opt}(I)$ the optimal value and by $\textsc{Alg}(I)$ the value returned by the algorithm.

To show that an algorithm is an α-approximation algorithm we need to show three things:

(1) The algorithm runs in polynomial time.

(2) The algorithm always produces a feasible solution.

(3) For any isntance I, the value is within a factor α of the optimal value:
$\textsc{Alg}(I) \leq \alpha\textsc{Opt}(I)$ (for a minimization problem, $\alpha \geq 1$)
$\textsc{Alg}(I) \geq \alpha\textsc{Opt}(I)$ (for a maximization problem, $\alpha \leq 1$)

1. Vertex cover

In this problem, we need to find for a given graph $G = (V, E)$ a subset of vertices such that each edge has an endpoint in the set. The goal is to minimize the number of vertices in the subset.

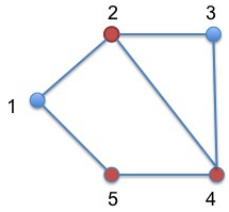


Figure 1: *The red vertices form a minimum vertex cover:* $S = \{2, 4, 5\}$.

Vertex Cover:

Instance:	Graph $G = (V, E)$.
Output:	$S \subseteq V$ such that each edge has at least one endpoint in S.
Goal:	Minimize $\lvert S \rvert$.

Algorithm A: Maximal matching

The Vertex Cover problem is $\mathcal{NP}$-hard. Thus, there is no polynomial time algorithm that solves the problem, unless $\mathcal{P} = \mathcal{NP}$. The next algorithm is a very simple 2-approximation.

Algorithm $\mathcal{A}$:
Find a maximal matching M and add all endpoints of the edges in M to S.

Theorem 1. *Algorithm $\mathcal{A}$ is a 2-approximation algorithm.*

Proof. We need to prove three things: [1] the running time is polynomial, [2] the solution is feasible, [3] the value of the solution is at most 2 times the optimal value.

[1] A maximal matching can be found by choosing the edges one by one until no edges can be added anymore. Clearly, this can be done in polynomial time.

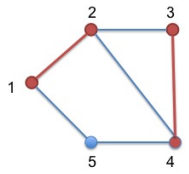


Figure 2: *The red edges form a maximal matching:* $M = \{(1,2),(3,4)\}$. *The solution given by the algorithm* $\mathcal{A}$ *is* $S = \{1,2,3,4\}$.

[2] Assume edge e is not covered. Then we can add e to M and get a bigger matching. This is not possible since we assumed that M is maximal. [3] Since the edges in M have no endpoints in common, we have $|S| = 2|M|$. Also, any solution must use at least one point from each of the M edges, but since these edges are independent (have no endpoints in common), we must have $\textsc{Opt} \geqslant |M|$. We conclude that $|S| = 2|M| \leqslant 2\textsc{Opt}$. □

Algorithm B: Linear Programming

The vertex cover problem can easily be formulated as an integer linear programming problem (ILP). Let $n = |V|$ be the number of vertices.

$$\begin{aligned} \text{(ILP)}\ \min \quad & Z = \sum_{j=1}^{n} x_j & \\ s.t. \quad & x_i + x_j \geqslant 1 & \text{for all } (i,j) \in E \\ & x_j \in \{0,1\} & \text{for all } j \in V. \end{aligned}$$

The vertex cover problem is $\mathcal{NP}$-hard which implies that the ILP above can not be solved in polynomial time, unless $\mathcal{P}$=$\mathcal{NP}$. However, the following LP-relaxation (in which $x_j \in \{0,1\}$ is replaced by $x_j \geq 0$) can be solved efficiently.

$$\begin{aligned} \text{(LP)}\ \min \quad & Z = \sum_{j=1}^{n} x_j & \\ s.t. \quad & x_i + x_j \geqslant 1 & \text{for all } (i,j) \in E \\ & x_j \geqslant 0 & \text{for all } j \in V. \end{aligned}$$

The idea of the algorithm is to solve (LP) and then *round* that solution in a feasible solution for (IP). This technique is called *LP-rounding*.

Algorithm $\mathcal{B}$:
Step 1: Solve the LP. $\rightarrow$ Optimal solution $x_1^*, x_2^*, \ldots, x_n^*$ with value Z_{LP}^*.
Step 2: Add j to solution S if $x_j^* \geqslant 1/2$.

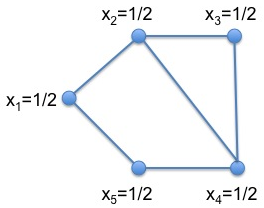


Figure 3: *In this example, an optimal LP-solution is $x_j^* = 1/2$ for all j. The value of the LP-solution is* 2.5. *Which is strictly smaller than the optimal ILP-solution which is 3. Algorithm $\mathcal{B}$ takes all five vertices: $S = \{1, 2, 3, 4, 5\}$ and has therefore value 5. The approximation ratio for this particular instance is therefore* 5/3. *In general, the ratio is never more than 2, as stated in Theorem 2.*

Theorem 2. *Algorithm $\mathcal{B}$ is a 2-approximation algorithm.*

Proof. We need to prove three things: [1] the running time is polynomial, [2] the solution is feasible, [3] the value of the solution is at most 2 times the optimal value.

[1]: True, since LP's can be solved in polynomial time. [2]: We need to show that every edge has an endpoint in S. Consider an arbitrary edge (i, j). We have $x_i^* + x_j^* \geqslant 1$. But then, either $x_i^* \geqslant 1/2$ or $x_j^* \geqslant 1/2$ (or both). That means that either i or j or both is added to S. [3] Denote the rounded solution by $\hat{x}$. That means: $\hat{x}_j = 1$ if $x_j^* \geqslant 1/2$ and $\hat{x}_j = 0$ otherwise. In either case, $\hat{x}_j \leqslant 2x_j^*$. The value of the solution found is

$$|S| = \sum_{j=1}^{n} \hat{x}_j \leqslant 2 \sum_{j=1}^{n} x_j^* = 2Z_{LP}^* \leqslant 2Z_{ILP}^* = 2\textsc{Opt}.$$

In the equation above, Z_{ILP}^* is the optimal value of the integer linear program ILP. (It is common to add a * to indicate the optimal solution and optimal value, as we do here.) □

Weighted case Now consider the *weighted* vertex cover problem. In this case, each vertex j has a given weight $w_j > 0$ and the goal is to minimize the total

weight of the cover. The analysis of algorithm $\mathcal{A}$ does not go through (Check this. Can you find an example where algorithm $\mathcal{A}$ gives a solution that is far from optimal?). However, the second algorithm, $\mathcal{B}$, does apply with only some minor changes: In the LP, there is only an extra term w_j and in the analysis there is only a small change in [3]

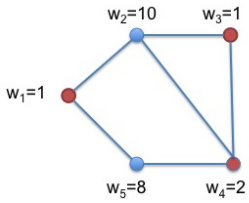


Figure 4: *Graph G with weights on the vertices is an instance of the weighted Vertex Cover problem. The optimal solution has total weight $1+1+2=4$.*

$$\begin{array}{lllll} \text{(LP)} & \min & Z = \sum_{j=1}^{n} w_j x_j & & \\ & s.t. & x_i + x_j \geqslant 1 & \text{for all } (i,j) \in E \\ & & x_j \geqslant 0 & \text{for all } j \in V. \end{array}$$

[3] The value of the solution found is

$$\sum_{j \in S} w_j = \sum_{j=1}^{n} w_j \hat{x}_j \leqslant 2 \sum_{j=1}^{n} w_j x_j^* = 2Z_{LP}^* \leqslant 2Z_{ILP}^* = 2\text{OPT}.$$

A generalization to Set Cover.

The Set Cover problem is a generalization of the Vertex Cover problem.

SET COVER:

Instance:	Set of items (elements) $E = \{e_1, \ldots, e_m\}$, subsets $S_1, \ldots, S_n \subseteq E$, and weights $w_1, \ldots, w_n \geqslant 0$.
Output:	$U \subseteq \{1, 2, \ldots, n\}$ such that each item is covered: $\bigcup_{j \in U} S_j = E$.
Goal:	Minimize the weight of the cover: $\sum_{j \in U} w_j$.

The vertex cover problem is the special case of the set cover problem in which each element e_i appears in exactly 2 sets. Now consider the Set Cover problem

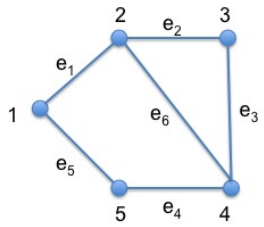


Figure 5: *Graph G is an instance of the Vertex Cover problem. Equivalently, we can write it as a Set Cover problem. For each vertex j there is a set S_j containing the adjacent edges: $S_1 = \{e_1, e_5\}$, $S_2 = \{e_1, e_2, e_6\}$, $S_3 = \{e_2, e_3\}$, $S_4 = \{e_3, e_4, e_6\}$, and $S_5 = \{e_4, e_5\}$.*

and assume that each item appears in at most f sets, for some constant f. The LP-rounding algorithm for vertex cover problem applies here in the same way.

$$\begin{aligned} \text{(ILP)} \quad \min \quad & Z = \sum_{j=1}^{n} w_j x_j \\ s.t. \quad & \sum_{j:e_i \in S_j} x_j \geqslant 1 \qquad \text{for all } i = 1, \ldots, m \\ & x_j \in \{0, 1\} \qquad \text{for all } j = 1, \ldots, n. \end{aligned}$$

The LP-relaxation is obtained by replacing $x_j \in \{0, 1\}$ by $x_j \geq 0$.

$$\begin{aligned} \text{(LP)} \quad \min \quad & Z = \sum_{j=1}^{n} w_j x_j \\ s.t. \quad & \sum_{j:e_i \in S_j} x_j \geqslant 1 \qquad \text{for all } i = 1, \ldots, m \\ & x_j \geqslant 0 \qquad \text{for all } j = 1, \ldots, n. \end{aligned}$$

Algorithm $\mathcal{B}$ (set cover):
Step 1: Solve the LP. $\rightarrow$ Optimal values $x_1^*, x_2^*, \ldots, x_n^*, Z_{LP}^*$
Step 2: Let U be all j for which $x_j^* \geqslant 1/f$.

Theorem 3. *Algorithm $\mathcal{B}$ is an f-approximation algorithm for Set Cover.*

Proof. [1] Clearly, the running time is polynomial since LP's can be solved in polynomial time and the second step can be done in linear time. (We only need to check each x_j^* once.)

[2] Any solution produced by this algorithm is feasible since each item appears in at most f sets. That means, there are at most f variables in the constraint for e_i and at least one of the variables must has value $\geqslant 1/f$.

[3] Denote the rounded solution by $\hat{x}$. That means, $\hat{x}_j = 1$ if $x_j^* \geqslant 1/f$ and $\hat{x}_j = 0$ otherwise. In either case, $\hat{x}_j \leqslant f x_j^*$. The value of the solution found is

$$\sum_{j \in U} w_j = \sum_{j=1}^{n} \hat{x}_j w_j \leqslant f \sum_{j=1}^{n} w_j x_j^* = f Z_{LP}^* \leqslant f Z_{ILP}^* = f \textsc{Opt}.$$

□

2. The k-cluster problem.

k-CLUSTER:

Instance:	Points $X = \{x_1, \ldots, x_n\}$ with underlying distance metric $d(,)$ and an integer k.
Output:	A partition of the points into k clusters $C_1, \ldots, C_k$.
Goal:	Minimize the maximum diameter of a clusters:

$$\text{Minimize:} \quad \max_j \left\{ \max_{x_a, x_b \in C_j} d(x_a, x_b). \right\}$$

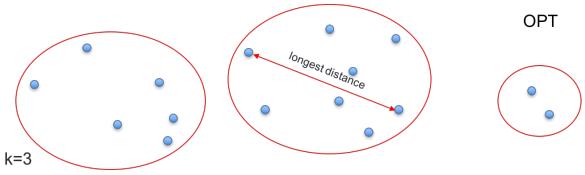


Figure 6: Example. The cost of the solution is the maximum distance between two points in a cluster.

Algorithm Greedy:

- Pick the first center μ_1 arbitrarily.
- For $i = 2$ to k:

  Let μ_i be the point in X that is farthest from $\{\mu_1, \ldots, \mu_{i-1}\}$.
- Create k clusters: C_i is the set of all $x \in X$ whose closest center is μ_i.

Theorem 4. *The Greedy algorithm is a 2-approximation algorithm.*

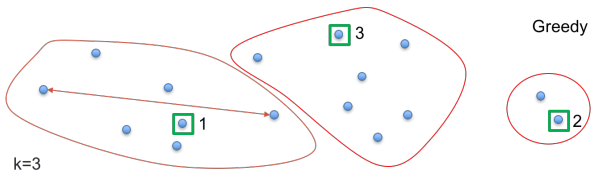
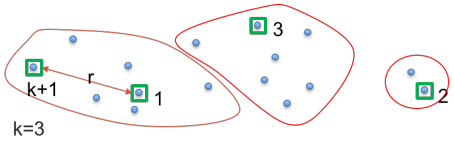


Figure 7: Example. The greedy solution if point 1 is chosen as the starting point.

Proof. Clearly, the algorithm runs in polynomial time. Also, it returns a feasible solution (a partition into k clusters). It remains to show that the value of the greedy solution is at most twice the optimal value.

Assume we make one more greedy step: we select center μ_{k+1}. Let r be distance of μ_{k+1} to its nearest centers among $\mu_1, \ldots, \mu_k$.



Observation 1: Since μ_{k+1} was selected as the point at maximum distance from $\{\mu_1, \ldots, \mu_k\}$, every point is at distance at most r from a center. Hence, for any two points x_a, x_b in the same cluster C_j we have (using the triangle inequality) $d(x_a, x_b) \leq d(x_a, \mu_j) + d(\mu_j, x_b) \leq r + r = 2r \quad \Rightarrow \quad \textsc{Alg} \leq 2r$.

Observation 2: The distance between any two of the centers $\mu_1, \ldots, \mu_{k+1}$ is at least r. Further, any solution must have a cluster that contains two of those. (Pigeon hole principle: There are k clusters and $k+1$ centers so at least one cluster has two centers.) So any solution has value at least $r \quad \Rightarrow \quad \textsc{Opt} \geq r$.

From the two observations: $\textsc{Alg} \leq 2r \leq 2\textsc{Opt}$. □

3. The traveling salesman problem.

TSP (SYMMETRIC):

Instance:	Complete graph with a cost c_{ij} for every pair i, j.
Output:	A cycle that goes through each point exactly once
Goal:	Minimize the length (sum of the edge costs) of the cycle.

In the *symmetric* TSP, the costs c_{ij} and c_{ji} are the same. In the *asymmetric* TSP, the cost c_{ij} may be different from c_{ji}. In that case, the cost depends on the direction in which the edge is traversed. For both versions, one usually considers the *metric* version, which means that the triangle inequality holds: $c_{ik} \leq c_{ij} + c_{jk}$ for every triple i, j, k.

Hardness of approximating the TSP

Theorem 5. *The Hamiltonian Cycle problem is reducible to the TSP problem.*

Proof. Given an instance $G = (V, E)$ of HC, form an instance of TSP by defining $c_{ij} = 1$ for all edges $(i, j) \in E$ and $c_{ij} = 2$ for all $(i, j) \notin E$. If there is no HC in G, then any TSP tour should use at least one of the edges of length 2. The other edges on the tour have length at least 1. The total length of the optimal TSP tour is at least $n + 1$. Hence,

$$\begin{array}{lcl} G \text{ has a HC} & \Rightarrow & \textsc{Opt}_{TSP} = n. \\ G \text{ has no HC} & \Rightarrow & \textsc{Opt}_{TSP} \geq n + 1. \end{array}$$

We conclude that G has a HC if and only if $\textsc{Opt}_{TSP} = n$. □

The Hamiltonian Cycle problem was one of the first problems shown to be NP-complete. It followed immediately (from the reduction above) that TSP is NP-complete (NP-hard) too. If we do not assume the triangle inequality, the same reduction shows a stronger hardness result.

Theorem 6. *For TSP without the triangle inequality assumption (the non-metric TSP), there does not exist an α-approximation algorithm for any number $\alpha \geqslant 1$, assuming $\mathcal{P} \neq \mathcal{NP}$.*

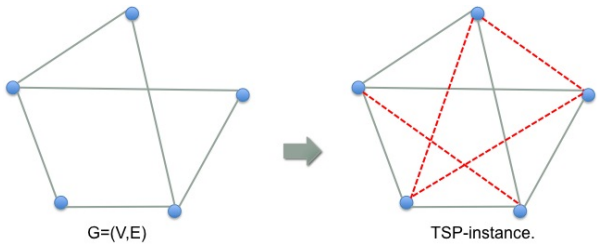


Figure 8: *An instance $G = (V, E)$ of the Hamiltonian Cycle problem and the corresponding TSP instance. Graph G has no Hamiltonian Cycle. Thus, any Hamiltonian Cycle in the graph to the right uses at least one of the dotted (red) edges.*

Proof. We follow the same proof as for Theorem 5 but instead of taking a cost 2 for the missing edges we take a much larger cost: $C = \alpha n$. (Any larger number will do too.) Assume there exists an α-approximation algorithm for some $\alpha \geq 1$. We show that such an algorithm can be used to solve the Hamiltonian Cycle (HC) problem in polynomial time.

If there is no HC in G, then any TSP tour should use at least one of the edges of length αn. The other edges on the tour have length at least 1. The total length of the optimal TSP tour is at least $\alpha n + (n-1) \geq \alpha n + 1$ (for $n \geq 2$). Hence,

$$\begin{array}{lcll} G \text{ has a HC} & \Rightarrow & \textsc{Opt}_{TSP} = n & \Rightarrow \quad \textsc{Alg} \leq \alpha n. \\ G \text{ has no HC} & \Rightarrow & \textsc{Opt}_{TSP} \geq \alpha n + 1 & \Rightarrow \quad \textsc{Alg} \geq \alpha n + 1. \end{array}$$

We see that G has a HC if and only if the value Alg of the solution given by the algorithm is at most αn. Now assume we want to find a HC in a given graph. If we would have an α-approximation algorithm for the non-metric TSP, then (by the reduction above) we could use it to find a HC in G. That means, HC reduces to non-metric TSP. □

Three TSP algorithms

Algorithm 1 (Double tree).
- Find a minimum spanning tree T.
- Double all the edge of the tree. (See Figure 10).
- Find an Euler tour in the double tree.
- Apply shortcutting in order to turn the Euler tour into a Hamiltonian cycle.

Algorithm 2 (Nearest addition).
- Pick an arbitrary point, say i_1, as the first point.
- Let i_2 be the point nearest to i_1. Make a directed tour from i_1 to i_2 and back to i_1. Let $S = \{i_1, i_2\}$.
- Repeat the following until a feasible tour is found:
  - Find a pair $i \in S, j \notin S$ with minimum cost c_{ij}. (In other words, find the point j that is nearest to the already chosen set S.) Insert j in the tour after i. Add j to S.

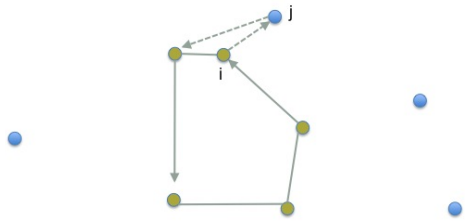


Figure 9: *Iteration of the nearest addition algorithm.*

Algorithm 3 (Christofides' algorithm).
- Find a minimum spanning tree T. Let O be the vertices of odd degree in T.
- Find a minimum cost perfect matching of the vertices in O. Denote the edges in this matching by M.
- Find an Euler tour in the graph $T + M$.
- Apply shortcutting in order to turn the Euler tour into a Hamiltonian cycle.

Note that a perfect matching on O exists since $|O|$ is even. (Any graph contains an even number of odd-degree points.) Also note that the cheapest perfect matching can be found in polynomial time. (Not for this course.)

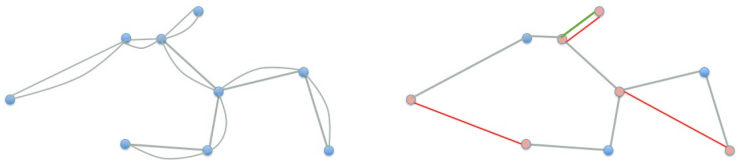


Figure 10: *The double tree (left) (for Algorithm 1) and the MST plus a matching of the odd-degree nodes (right) (for Algorithm 3).*

Analysis of the algorithms Let T be a minimum spanning tree. Denote by *Cost*(T) the cost of the MST (i.e., the sum of the edge lengths). Let Opt be the length of the shortest TSP tour.

Lemma 1. Cost(T) $\leq$ Opt.

Proof. Removing an arbitrary edge from the optimal TSP tour gives a path that connects all points. Note that this is also spanning tree. Hence, the cost of the MST (the cheapest spanning tree) is no more than Opt. □

Theorem 7. *Double tree is a 2-approximation algorithm.*

Proof. The length of the tour before shortcutting is exactly twice the length of the MST, which is at most twice Opt (by Lemma 1) The shortcutting step at the end does not increase the length of the tour (since the triangle inequality holds). □

Theorem 8. *Nearest addition is a 2-approximation algorithm.*

Proof. The edges (i, j) with $i \in S, j \notin S$ that are selected in each iteration are exactly the same edges that are selected by Prim's MST algorithm. (See an example on teh slides.)
Now consider an arbitrary iteration of the algorithm. Assume that pair (i, j) was selected in that step. That means, j was the point nearest to S and this distance is c_{ij} with $i \in S$. Let k be the point that follows i in the tour constructed so far. By the triangle inequality $c_{jk} \leq c_{ji} + c_{ik} \Rightarrow c_{jk} - c_{ik} \leq c_{ji}$. Inserting j in the tour increases its length (cost) by at most

$$c_{ij} + c_{jk} - c_{ik} \leq 2c_{ij}.$$

We conclude that the length of the tour is at most twice the length of the minimum spanning tree. Now the proof follows from Lemma 1 □

Theorem 9. *Christofides' algorithm is a* 3/2*-approximation algorithm.*

Proof. Consider an optimal tour. Now shortcut this tour to get a tour on O. The length of this tour is at most Opt (by the triangle inequality). This tour is composed of exactly two perfect matchings on O. Since the algorithm computes the cheapest perfect matching, its cost is no more than Opt/2.

$$Cost(T) + Cost(M) \leq \frac{3}{2}\textsc{Opt}.$$

The shortcutting step at the end of the algorithm does not increase the length of the tour (since the triangle inequality holds). □

Christofides' algorithm is the best approximation algorithm known so far (in terms of approximation ratio) for the metric TSP.

Exercises

Exercise 1 Show by an example that Algorithm A is not a 2-approximation algorithm for the *weighted* vertex cover problem.

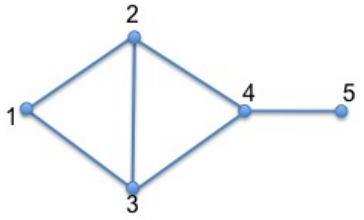
Exercise 2

(a) Give an optimal vertex cover for the graph below.

(b) Write down the ILP for this vertex cover instance.

(c) Write down the LP-relaxation for this vertex cover instance.

(d) For this instance, give a solution to the LP-relaxation which has a value strictly smaller than the optimal value (given in (a)).



Exercise 3 Show by an example that the Greedy algorithm for k-clustering is not better than a 2-approximation algorithm. That means, given an example for which the value of the algorithm's solution is twice the optimal value.

Exercise 4 Exercise 9.4 of Dasgupta.

Exercise 5 Recall the k-SPANNING TREE problem of Exercise 8.12 of Dasgupta (week 2). We proved that the problem is NP-complete by a reduction from Rudrata Path. Now use the same reduction to show that for $\alpha < 1.5$ there is no α-approximation algorithm for k-SPANNING TREE (assuming that $\mathcal{P} \neq \mathcal{NP}$).

Exercise 6 Exercise 9.6 of Dasgupta.
Hint: Argue that the cost of an optimal TSP tour on the terminals V' is at most twice the cost of an optimal Steiner Tree (ST) on V':
$\text{OPT}_{TSP}(V') \leq 2\text{OPT}_{ST}(V')$.