Approximation algorithms

Heuristic Algorithms

Giovanni Righini

University of Milan

Department of Computer Science (Crema)

Università degli Studi di Milano

Approximation algorithms

Approximation algorithms are heuristic algorithms providing guarantees on the approximation.

To evaluate how good approximation algorithms are, it is necessary to measure the approximation they guarantee a priori.

Measuring the approximation

Given an instance I of a problem P, we want to measure the approximation guaranteed by an algorithm A. We call

- z_I^A the value of the heuristic solution, computed by the algorithm
- z_I^* the value of the optimal solution.

Now we can consider

- the absolute difference $|z_I^A - z_I^*|$; this is not recommendable because it depends on the scale, which is arbitrary;
- the relative difference $\frac{|z_I^A - z_I^*|}{|z_I^*|}$, often used in experimental analysis;
- the approximation ratio $\max\{\frac{z_I^A}{z_I^*}, \frac{z_I^*}{z_I^A}\}$ which is always ≥ 1.

Worst-case approximation

We do not want to evaluate algorithms for single instances, but for any instance of a given problem.

Hence, as with the computational complexity analysis, we consider the worst-case, i.e. the maximum value that the approximation ratio can have among all possible instances of the problem.

$$\max_I \left\{ \frac{z_I^A}{z_I^*} \right\}.$$

$g(N)$-approximation

If for any instance I of P we have

$$\frac{|z_I^A - z_I^*|}{|z_I^*|} \leq g(n)$$

where $g(n)$ is a polynomial in the instance size n, then algorithm A is $g(n)$-approximating for problem P.

In this case the approximation bound depends on n, i.e. on the size of the instance.

When the approximation bounds depend on the data of the instance (not only on its size), they are called data-dependent bounds.

Constant factor approximation

If for any instance I of P we have

$$\frac{|z_I^A - z_I^*|}{|z_I^*|} \leq K$$

where K is a constant, then algorithm A is K-approximating for problem P.

In this case the approximation bound depends neither on the size of the instance nor on the values of the data.

Approximation schemes

If for any instance I of P we have

$$\frac{|z_I^A - z_I^*|}{|z_I^*|} \leq \epsilon$$

where $\epsilon \geq 0$ is an arbitrarily chosen parameter, then algorithm A is an approximation scheme for problem P.

In this case we can tune the trade-off between accuracy and computing time, by a suitable choice of ϵ.

An approximation scheme is polynomial (PTAS: polynomial time approximation scheme) if, for each fixed ϵ, its computational complexity is polynomial in n.
An approximation scheme is fully polynomial (FPTAS: fully polynomial time approximation scheme) if, its computational complexity is polynomial in n and $1/\epsilon$.

Constructive algorithms

Approximation properties are usually proven for constructive algorithms.

The general technique to prove these properties consists of establishing a relationship between a lower bound to z_I^* and an upper bound to z_I^A (for minimization problems).

A constructive algorithm is an algorithm where decisions are taken sequentially, so that a partial solution (initially empty) is iteratively added new elements until a complete solution is produced and the algorithm stops.

In a partial solution some variables have already been assigned a value, while others are still to be decided.

A 2-approximation algorithm for the *Vertex Cover Problem*

Given a graph $G = (V, E)$ we search for a minimum cardinality vertex set covering all the edges.

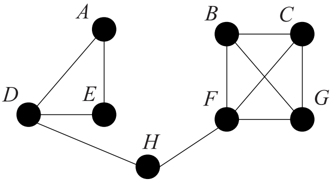
A matching is a set of non-adjacent edges.

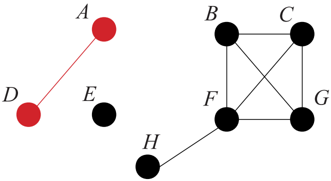
A maximal matching is a matching such that all the other edges of the graph are adjacent to it.

Matching algorithm:

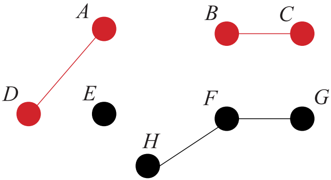
1. Compute a maximal matching $M \subseteq E$;
2. The solution is the set of the endpoints of M.

An example

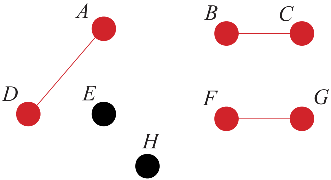




$z^A = 2 \cdot |M| = 2$



$z^A = 2 \cdot |M| = 4$



$z^A = 2 \cdot |M| = 6$

The optimal solution value is $z^* = 5$ (many optimal solutions).

Proof

The matching algorithm is 2-approximating.

1. The cardinality of M is a lower bound $LB(I)$:
   - the cardinality of an optimal covering for any subset of edges $E' \subseteq E$ does not exceed that of an optimal covering for the whole edge set E.

$$|x^*_{E'}| \leq |x^*_E|$$

   - the optimal covering for any matching M has cardinality $|M|$, because one vertex is necessary and sufficient for each edge of M.
2. Including both endpoints of each edge of M we get
   - a value equal to $2LB(I)$ (two endpoints for each edge)
   - an upper bound $UB(I)$ (because it covers M and the edges adjacent to it)
3. The matching algorithm produces solutions whose value is $z^A(I) = UB(I)$.

Therefore $z^A(I) \leq 2z^*(I)$ for each $I \in \mathcal{I}$, i.e. $\max_I\{\frac{z^A(I)}{z^*(I)}\} = 2$.

Tightness

In a K-approximating algorithm, the factor K does not link $z^A(I)$ with $z^*(I)$, but $UB(I)$ with $LB(I)$.

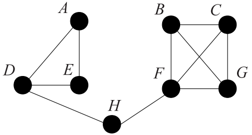
The approximation achieved by the algorithm in practice is often much better (tighter) than K.

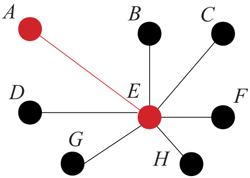
An interesting question is:
Are there instances $\bar{I}$ such that $z_A(\bar{I}) = Kz^*(\bar{I})$?
What characteristics do they have?

Studying these instances is useful

- to understand whether they are frequent or rare;
- to modify the algorithm.

A VCP instance for which the bound is tight.





Inapproximability

For some $\mathcal{NP}$-hard optimization problems it is not possible to find a K-approximation algorithm for any fixed K, unless $\mathcal{P} = \mathcal{NP}$.

For instance, if there exist an instance $\bar{I}$ whose optimal solution has null cost

$$\begin{cases} z^*(\bar{I}) = 0 \\ z^A(I) \leq Kz^*(I) \; \forall I \in \mathcal{I} \end{cases} \quad \Rightarrow z^A(\bar{I}) \leq Kz^*(\bar{I}) = 0 \Rightarrow z^A(\bar{I}) = 0$$

Example: given a digraph G, give null cost to its arcs and complete the digraph with arcs of cost 1. The ATSP has a zero-cost solution if and only if G contains a Hamiltonian circuit. $\Rightarrow$ *The ATSP admits a polynomial-time approximation algorithm if and only if* $\mathcal{P} = \mathcal{NP}$.

Special cases

A non-approximable problem may contain approximable special cases.

Consider the *TSP* with the following additional assumptions:

- the graph $G = (N, A)$ is complete;
- the cost function c is symmetric and satisfies the triangle inequality:

$$c_{ij} = c_{ji} \quad \forall i, j \in N \qquad \text{and} \qquad c_{ij} + c_{jk} \geq c_{ik} \quad \forall i, j, k \in N$$

The double spanning tree algorithm

Double spanning tree algorithm for the TSP with triangle inequality.

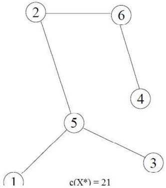
1. Given $G = (V, E)$, compute a minimum cost spanning tree $T^* = (V, X^*)$;
2. Duplicate each edge $[i, j]$ of X^* into two opposite arcs (i, j) and (j, i); Let D be the resulting arc set; it forms a circuit visiting each vertex at least once.
3. Compute the Euler Tour Representation (ETR) of D as follows:
   - Sort the arcs of D in lexicographic order.
   - Compute an adjacency list for each vertex (called *next*) and a map from vertices to the first entries of the adjacency lists (called *first*):
   - For each arc (u, v) in the sorted list:
     - if the previous arc (u', v') has $u' = u$ then set $next(u', v') := (u, v)$ else set $first(u) := (u, v)$.
   - Compute the (circular) arc list (called *succ*) as follows:

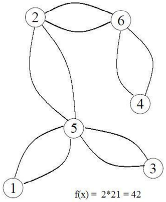
$$succ(u, v) := \begin{cases} first(v) & next(v, u) = nil \text{(advance arc)} \\ next(v, u) & \text{otherwise (retreat arc).} \end{cases}$$

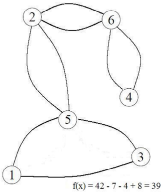
4. Follow the Euler tour defined by *succ* and every time a vertex occurs after the first time, shortcut it.

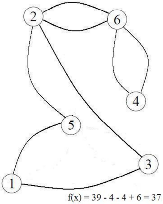
Example

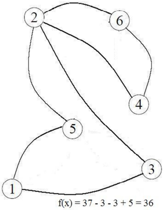
	1	2	3	4	5	6
1	-	9	8	9	7	9
2	9	-	6	5	4	3
3	8	6	-	9	4	6
4	9	5	9	-	8	3
5	7	4	4	8	-	5
6	9	3	6	3	5	-

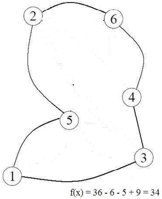












Proving the approximation ratio

The double spanning tree algorithm is 2-approximating.

1. the cost of T^* is a lower bound because:
   - deleting an arc from a Hamiltonian cycle, we obtain a Hamiltonian path with a non-larger cost;
   - a Hamiltonian path is a special case of a spanning tree.
2. when we replace each edge with two arcs we obtain D whose cost is twice the cost of T^*;
3. when we shortcut some arcs, we obtain a final solution whose cost is not larger than that of D (for the triangle inequality).

Therefore red $z^A(I) \leq 2z^*(I)$ for every instance $I \in \mathcal{I}$.

The repeated assignment algorithm for the ATSP

Given a strongly connected digraph $D = (N, A)$ and a cost function $c : A \mapsto \Re_+$, we want to compute a minimum cost Hamiltonian circuit.

We assume that the costs satisfy the asymmetric triangle inequality:

$$c_{ij} + c_{jk} \geq c_{ik} \quad \forall i, j, k \in N$$

and the digraph is complete.

Let us define a bipartite graph $B = (T, H, E)$, where T and H are the set of tails and heads of the arcs in A and the cost of each edge $[i, j]$ with $i \in T$ and $j \in H$ is c_{ij}.

Since there are no self-loops in D and D is complete, B has all edges except those of the form $[i, i]$.

The repeated assignment algorithm for the ATSP

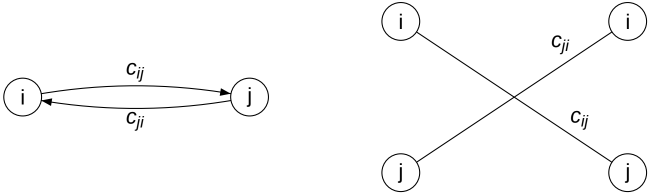


Figure: Correspondence between weighted arcs in *D* and weighted edges in *B*.

The repeated assignment algorithm for the ATSP

Property 1. A perfect bipartite matching on B corresponds to a set of arcs in A such that each node in N has in-degree and out-degree equal to 1. In general this is a set of sub-tours in D.

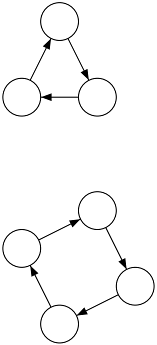


Figure: A set of sub-tours in D.

The repeated assignment algorithm for the ATSP

Property 2. A Hamiltonian circuit is a special case. Hence a minimum cost perfect bipartite matching M^* on B provides a lower bound to the minimum cost Hamiltonian tour H^* on D.

$$M^* \leq H^*.$$

The same property holds if we consider any subset $\overline{N} \subseteq N$.

A minimum cost perfect bipartite matching $\overline{M}^*$ on $\overline{N}$ provides a lower bound to the minimum cost Hamiltonian tour $\overline{H}^*$ visiting $\overline{N}$.

$$\overline{M}^* \leq \overline{H}^*.$$

The repeated assignment algorithm for the ATSP

But

$$\overline{H}^* \leq \overline{H^*} \leq H^*$$

where $\overline{H^*}$ is the sub-tour obtained from H^* short-cutting all nodes not in $\overline{N}$.

The first inequality comes from the optimality of $\overline{H}^*$.
The second inequality comes from the triangle inequality.
Therefore

$$\overline{M}^* \leq H^* \ \ \forall \overline{N} \subseteq N.$$

This idea is applied iteratively.

The repeated assignment algorithm for the ATSP

Step 1. Set $k = 1$ and define the initial bipartite graph B_1.

Step 2. Compute a minimum cost complete matching M_k^* on bipartite graph B_k.

Step 3. If the number of sub-tours corresponding to M_k^* is equal to 1, then go to Step 5a.

Step 4. Set $k := k + 1$. In each sub-tour choose a node at random as a representative and define the bipartite graph B_k induced by the representatives. Go back to Step 2.

The repeated assignment algorithm for the ATSP

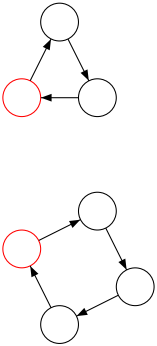


Figure: Iteration 1: 4 sub-tours and their representatives (red).

The repeated assignment algorithm for the ATSP

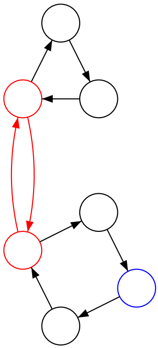


Figure: Iteration 2: 2 sub-tours and their representatives (blue).

The repeated assignment algorithm for the ATSP

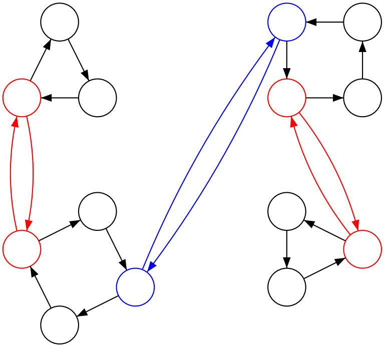


Figure: Iteration 3: a single sub-tour. The result is an Euler digraph.

The repeated assignment algorithm for the ATSP

Property 3. The digraph $D' = (N, \bigcup_k M_k^*)$ resulting from these steps is strongly connected, because each sub-tour is a s.c.c. and all of them can be reached from one another following the arcs between the representatives.

Property 4. The resulting digraph D' is also an Euler graph, because at each iteration both the in-degree and the out-degree of the representatives are increased by 1 while the degrees of the other nodes remain unchanged. So the in-degree and the out-degree of each node are always equal.

The repeated assignment algorithm for the ATSP

Step 5a. Find an Euler tour T in D'. By definition its cost is the same as the cost of all arcs in D': $T = \sum_k M_k^*$.
Step 5b. Obtain a Hamiltonian tour H from the Euler tour T by repeated shortcuts.

Property 5. The cost of the final Hamiltonian tour H is not larger than the cost of the Euler tour T, because of the asymmetric triangle inequality: $H \leq T$.

Property 6. The number of sub-tours (and representatives) is at least halved at each iteration, because self-loops are not allowed. Hence the number K of iterations is bounded by $\log n$.

The repeated assignment algorithm for the ATSP

Approximation. Combining the inequalities above, we have:

- $H \leq T$;
- $T = \sum_{k=1}^{K} M_k^*$;
- $M_k^* \leq H^* \ \ \forall k = 1, \ldots, K$;
- $K \leq \log_2 n$.

Therefore

$$H \leq \log_2 n \, H^*.$$

Complexity. Since every minimum cost bipartite matching M_k^* can be computed in polynomial time ($O(n^3)$ with the Hungarian algorithm) and the number of iterations is bounded by $\log_2 n$, the algorithm is polynomial time.

The repeated assignment algorithm (Frieze, Galbiati, Maffioli, 1975) is $\log n$-approximating for the ATSP with triangle inequality.

Christofides' algorithm (1976)

Christofides' algorithm is a constant-factor approximation algorithm for the TSP with triangle inequality. It runs in three steps.

Step 1. On the assigned graph $G = (V, E)$, compute a minimum cost spanning 1-tree T^*.

Property 1. A minimum cost spanning 1-tree can be computed in polynomial time in this way:

- compute a minimum cost spanning tree (Prim, Kruskal, Boruvka,...);
- find the minimum cost edge not in it and add it to the spanning tree.

Property 2. A Hamiltonian cycle is a special case of a spanning 1-tree. Hence $T^* \leq H^*$, where H^* is the minimum cost Hamiltonian cycle on G.

Christofides' algorithm (1976)

Step 2. Consider the subset V_o of vertices with odd degree in T^*. Compute a minimum cost perfect matching M^* between them.

Property 3. The cardinality of V_o is even, because the sum of the degrees of all nodes in a graph is always an even number. Therefore a perfect matching on the vertices in V_o exists.

Property 4. The perfect matching in a graph with an even umber of vertices can be computed in polynomial time (Edmonds' algorithm).

Christofides' algorithm (1976)

Property 5. Every Hamiltonian cycle with an even number of edges is the union of two perfect matchings. In particular $M^* \leq \frac{1}{2}H_o^*$, where H_o^* is the cost of the minimum Hamiltonian cycle through the vertices of V_o.

Property 6. Let H^* be the optimal Hamiltonian cycle and let $(H^*)_o$ the cycle obtained from it by short-cutting all vertices not in V_o. Since the triangle inequality holds, $(H^*)_o \leq H^*$. Since H_o^* is the minimum Hamiltonian cycle in the subgraph induced by V_o, $H_o^* \leq (H^*)_o$.

Therefore $M^* \leq \frac{1}{2}H^*$.

Christofides' algorithm (1976)

Step 3. Consider the graph with the edges of T^* and the edges of M^*. Traverse it, skipping already visited vertices and produce a Hamiltonian cycle H.

Property 7. All nodes with even degree in T^* have degree 0 in M^*. All nodes with odd degree in T^* have degree 1 in M^*. Hence all nodes in $T^* \cup M^*$ have even degree and the graph is connected (because it contains a spanning tree). Therefore the resulting graph with the edges of T^* and those of M^* is an Euler graph.

Property 8. An Euler tour E in an Euler graph can be found in polynomial time with a simple greedy algorithm: *Start from a random vertex and traverse-and-delete one of the edges without disconnecting the resulting graph*.

Christofides' algorithm (1976)

Property 9. A Hamiltonian tour H can be produced from an Euler tour E by successive shortcuts in polynomial time with another simple greedy algorithm: *Follow the Euler tour and apply a shortcut every time the next vertex has already been visited*.

Property 10. Because of the triangle inequality, shortcuts cannot increase the cost of the tour. Hence $H \leq E$.

Now we can combine the inequalities obtained from each step:

- From Step 1: $T^* \leq H^*$
- From Step 2: $M^* \leq \frac{1}{2}H^*$
- From Step 3: $H \leq E = T^* + M^*$.

Therefore

$$H \leq \frac{3}{2}H^*.$$

The nearest neighbor algorithm for the TSP

Nearest neighbor (NN) algorithm: start from a vertex at random and go to the closest vertex among those not yet visited.

If the triangle inequality holds, this algorithm has

$$\frac{z^A(I)}{z^*(I)} \leq \frac{1}{2} + \frac{1}{2}\lceil \log n \rceil$$

i.e. a $g(n)$-approximation guarantee.

Insertion algorithms for the TSP

Insertion Algorithms: Start from a partial tour including only two vertices and iteratively include in it one more vertex among those out of it.

The insertion always occurs in the cheapest position along the tour.

According to the selection criterion of the next vertex to insert we have different algorithms with different approximation properties:
Nearest Insertion and *Cheapest Insertion* have $\frac{z^A(I)}{z^*(I)} \leq 2$;
Farthest Insertion and *Furthest Insertion* have $\frac{z^A(I)}{z^*(I)} \leq 1 + \lceil \log n \rceil$.

Remark: Those with worse theoretical approximation properties provide better experimental results (!).

Insertion algorithms for the TSP with TI

Given a graph $G = (V, E)$ and a cost function c on the edges, satisfying the triangle inequality,

- select the two closest vertices u and v;
- arbitrarily select an insertion order of the other vertices.

This defines a heuristic solution of cost H.

We indicate by δ_i the increase in the solution cost when vertex i is inserted.

We set:

$$\delta_u = 0$$

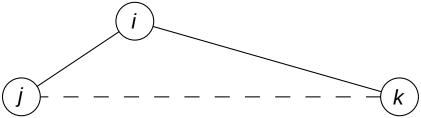
$$\delta_v = 2c_{uv}.$$

The cost of the heuristic solution is

$$H = \sum_{i \in V} \delta_i.$$

Insertion algorithms for the TSP with TI

After the first two vertices, every time a vertex i is inserted between a vertex j and a vertex k:



the increase in the tour length is $\delta_i = c_{ij} + c_{ik} - c_{jk}$.
Owing to the triangle inequality:

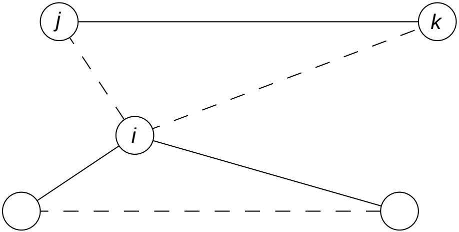
$$c_{ik} - c_{jk} \leq c_{ij} \quad \text{and} \quad c_{ij} - c_{jk} \leq c_{ik}$$

$$\delta_i \leq 2c_{ij} \quad \text{and} \quad \delta_i \leq 2c_{ik}$$

if i is inserted *adjacent to* j or k (after them).

Insertion algorithms for the TSP with TI

Insertions are always done in the cheapest position.



$$\delta_i \leq c_{ij} + c_{ik} - c_{jk}$$

Owing to the triangle inequality:

$$c_{ik} - c_{jk} \leq c_{ij} \quad \text{and} \quad c_{ij} - c_{jk} \leq c_{ik}$$

$$\delta_i \leq 2c_{ij} \quad \text{and} \quad \delta_i \leq 2c_{ik}$$

if i is inserted *not adjacent to* j or k (after them).

Insertion algorithms for the TSP with TI

So, the triangle inequality and the choice of the optimal insertion position guarantee that

$$\delta_i \leq 2c_{ij}$$

whenever a vertex i is inserted into the tour and a vertex j is already in the tour.

This property holds also for the first two vertices u and v, because $c_{uv} = \min_{[i,j] \in E}\{c_{ij}\}$.

Now consider any pair of vertices i and j.
If j is inserted after i then $\delta_j \leq 2c_{ij}$.
If i is inserted after j then $\delta_i \leq 2c_{ij}$.

Hence

Lemma 1 : $\min\{\delta_i, \delta_j\} \leq 2c_{ij} \quad \forall i, j \in V.$

Insertion algorithms for the TSP with TI

At iteration $k = 1$, consider the whole set of vertices $V_1 = V$ and the corresponding optimal solution $X_1^* = X^*$.

At each iteration k:

- Assign each edge $[i, j]$ in X_k^* a weight $w_{ij} = \min\{\delta_i, \delta_j\}$. The endpoint with minimum value of δ is the leading vertex for edge $[i, j]$.
- Compute a minimum cost matching M_k with respect to the original cost function c using the edges of X_k^*. Let L_k be the set of the leading vertices of the edges of M_k.
- Shortcut the vertices in L_k from X_k^* and obtain X_{k+1}^*.
- Delete the vertices in L_k from V_k and obtain V_{k+1}.
- Stop when only a vertex $\bar{v}$ is left.

Insertion algorithms for the TSP with TI

At each iteration $\lfloor \frac{|V_k|}{2} \rfloor$ vertices are deleted.

The number of iterations is $K = \lceil \log_2 n \rceil$.

All vertices but $\bar{v}$ are leading vertices of an edge in the matchings exactly once.

Hence

$$V = \bigcup_{k=1}^{K} L_k \cup \bar{v}$$

and

$$\sum_{k=1}^{K} \sum_{i \in L_k} \delta_i = \sum_{i \in V \setminus \{\bar{v}\}} \delta_i = H - \delta_{\bar{v}}.$$

Insertion algorithms for the TSP with TI

At each iteration k:

- by definition of leading vertex

$$\sum_{i \in L_k} \delta_i = \sum_{[i,j] \in M_k} w_{ij}$$

- for the definition of the weight function w and Lemma 1

$$w_{ij} \leq 2c_{ij} \;\; \forall [i,j] \in M_k \quad \text{and} \quad \sum_{[i,j] \in M_k} w_{ij} \leq 2 \sum_{[i,j] \in M_k} c_{ij}$$

- since $X_k^* \setminus M_k$ contains a matching and M_k is optimal

$$\sum_{[i,j] \in M_k} c_{ij} \leq \frac{1}{2} X_k^*$$

- for the triangle inequality

$$X_k^* \leq X^*$$

Insertion algorithms for the TSP with TI

Recalling that:

$$\sum_{k=1}^{K} \sum_{i \in L_k} \delta_i = H - \delta_{\bar{v}}$$

$$K = \lceil \log_2 n \rceil$$

and observing that $\delta_{\bar{v}} \leq H^*$ (trivial), we get:

$$H = \sum_{k=1}^{K} \sum_{i \in L_k} \delta_i + \delta_{\bar{v}} = \sum_{k=1}^{K} \sum_{[i,j] \in M_k} w_{ij} + \delta_{\bar{v}} \leq \sum_{k=1}^{K} (2 \sum_{[i,j] \in M_k} c_{ij}) + \delta_{\bar{v}} \leq$$

$$\leq \sum_{k=1}^{K} 2 \frac{1}{2} X_k^* + \delta_{\bar{v}} \leq \sum_{k=1}^{K} X^* + \delta_{\bar{v}} \leq (\lceil \log_2 n \rceil) X^* + X^* = (1 + \lceil \log_2 n \rceil) X^*.$$

Data-dependent bounds for the ATSP

Given a complete digraph $D(N,A)$ with a cost function c on the arcs, satisfying the asymmetric triangle inequality, we define a complete graph $G = (N,E)$ with weights w on the edges, such that $w_{ij} = c_{ij} + c_{ji}$ for each edge $[i,j] \in E$.

Each circuit in D has a corresponding anti-circuit $\hat{C}$, i.e. $(i,j) \in C \Leftrightarrow (j,i) \in \hat{C}$.

Each pair $(C,\hat{C})$ in D corresponds to a cycle in G, whose cost is the sum of the costs of C and $\hat{C}$.
Let C^* be the minimum cost Hamiltonian circuit in D, let $\hat{C}^*$ be its anti-circuit and let $\overline{S}$ be the Hamiltonian cycle in G corresponding to the pair $(C^*,\hat{C}^*)$.

It is easy to prove that the triangle inequality also holds on G, as a consequence of the triangle inequality on D.

Data-dependent bounds for the ATSP

Running Christofides' algorithm on G, we get a Hamiltonian cycle $S \leq \frac{3}{2}S^*$, where S^* is the minimum Hamiltonian cycle on G.

Let $(C_S, \hat{C_S})$ the circuit-anti-circuit pair in D that corresponds to S, so that $S = (C_S + \hat{C_S})$.
Let H be the shortest Hamiltonian circuit among C_S and $\hat{C_S}$. Then $H \leq \frac{1}{2}(C_S + \hat{C_S})$.

Combining the above inequalities we obtain:

- $H \leq \frac{1}{2}(C_S + \hat{C_S})$;
- $S = (C_S + \hat{C_S})$;
- $S \leq \frac{3}{2}S^*$;
- $S^* \leq \overline{S}$;
- $\overline{S} = (C^* + \hat{C}^*)$.

Hence $H \leq \frac{3}{4}(C^* + \hat{C}^*)$, i.e. $\frac{H}{C^*} \leq \frac{3}{4}(1 + \frac{\hat{C}^*}{C^*})$.

Data-dependent bounds for the ATSP

Now we define a measure of the asymmetry of the digraph D:

$$\alpha = \max_{(i,j)\in A}\{\frac{c_{ij}}{c_{ji}}\}$$

and we obtain

$$\frac{H}{C^*} \leq \frac{3}{4}(1+\alpha).$$

When α tends to 1 (symmetric costs), this bound tends to that of Christofides' algorithm, i.e. $\frac{3}{2}$.

Therefore this algorithm (Righini, Trubian, 1995) provides data-dependent bounds for the ATSP with triangle inequality.
In cases like this different instances of a same problem can be classified according to their approximability.

Combination of bounds: the Stacker-Crane Problem

We are given a weighted mixed graph $G = (N, A, E)$, where N is the set of nodes, A is a set of oriented arcs, E is a set of un-oriented edges and the following properties hold:

- each node either the tail or the head of exactly one arc (hence $|N| = 2|A|$);
- the cost of the edges linking the endpoints of an arc has the same cost as the arc;
- E contains all possible edges and the triangle inequality holds for their costs;

The objective is to find a minimum cost Hamiltonian tour on G that traverses all arcs in the right direction (from the tail to the head).

Every feasible solution is made by $N/2$ arcs and $N/2$ edges alternating.
We indicate by E^* the edges in the optimal solution H^*.

The problem is NP-hard.

Large-Arcs algorithm

Step 1. Define a complete bipartite graph $B = (T, H, E)$, where T and H are the set of tails and heads of the arcs in A. Compute a minimum cost matching M^* in B. The graph made by A and M^* is a set S of sub-tours.

Step 2. Define an auxiliary graph $L = (S, W)$, with one vertex for each sub-tour. The cost of each edge $[s', s'']$ in W is the minimum edge cost among all the edges of E connecting the two sub-tours s' and s''. Compute a minimum cost spanning tree T^* in L.

Step 3. Consider the multi-graph U made by the arcs A, the edges in M^* and two copies of each edge in T^*. It is an Euler graph. Find an Euler tour along it and transform it into a Hamiltonian tour $H_{LargeArcs}$, by repeated shortcuts on pairs of consecutive edges.

All steps can be done in polynomial time.

Large-Arcs algorithm

The following inequalities hold:

- $M^* \leq E^*$
- $T^* \leq E^*$
- $U = A + M^* + 2T^*$ by construction;
- $H_{LargeArcs} \leq U$, for the triangle inequality;
- $H^* = A + E^*$.

Therefore $H_{LargeArcs} \leq U = A + M^* + 2T^* \leq A + 3E^* = 3H^* - 2A$, i.e.:

$$\frac{H_{LargeArcs}}{H^*} * \leq 3 - 2\frac{A}{H^*}.$$

The larger is A, the better is the approximation bound.

Small-Arcs algorithm

Step 1. Consider an auxiliary graph $L = (V, W)$ with a vertex in V for each arc in A and such that the cost of each edge $[i, j]$ in W is the minimum among the costs of the four edges in E linking the endpoints of arc i with those of arc j. Compute a minimum cost spanning tree T^* in L and report its edges back to the original graph G.

Step 2. Consider the subset of nodes of L that have odd degree in T^*. Compute a minimum cost perfect matching M^* on them and report its edges back to the original graph G.

Small-Arcs algorithm

Step 3. Consider the graph made by the arcs A, the edges in T^* and the edges in M^*. The in-degree and out-degree of the endpoints of all arcs are either both even or both odd. Define *odd arcs* and *even arcs* accordingly. For each odd arc, insert a copy of its parallel edge; the cost for this is A_{odd}. For each sub-tour containing even arcs, consider its two possible orientations and choose the one in which the arcs traversed in the wrong direction have minimum cost. According to the chosen orientation, insert two copies of the parallel edges for each arc traversed in the wrong direction. The cost for this is at most $2\frac{A_{even}}{2}$. The resulting mixed graph U is an Euler graph.

Step 4. Find an Euler tour along U and transform it into a Hamiltonian tour $H_{SmallAarcs}$, by repeated shortcuts on pairs of consecutive edges.

All steps can be done in polynomial time.

Small-Arcs algorithm

The following inequalities hold (the first two steps are the same as those of Christofides' heuristics):

- $T^* \leq E^*$;
- $M^* \leq \frac{1}{2}E^*$;
- $U \leq A + M^* + T^* + A_{odd} + 2\frac{A_{even}}{2}$ by construction;
- $A = A_{odd} + A_{even}$;
- $H_{SmallArcs} \leq U$ for the triangle inequality;
- $H^* = A + E^*$.

Therefore $H_{SmallArcs} \leq U \leq 2A + M^* + T^* \leq 2A + \frac{3}{2}E^* = 2A + \frac{3}{2}(H^* - A) = \frac{3}{2}H^* + \frac{1}{2}A$, i.e.:

$$\frac{H_{SmallArcs}}{H^*} \leq \frac{3}{2} + \frac{A}{2H^*}.$$

The smaller is A, the better is the approximation bound.

Combining the two bounds

Both Large-Arcs and Small-Arcs provide an approximation bound that depends on A, i.e. a data-dependent bound.

But one of the bounds increases with A, the other one decreases with A. Hence, their combination provides a constant approximation bound.
When we run both algorithm and we select the best solution H we get an approximation equal to:

$$\frac{H}{H^*} \leq \min\{H_{LargeArcs}/H^*, H_{SmallArcs}/H^*\} = \min\{3 - 2\frac{A}{H^*}, \frac{3}{2} + \frac{A}{2H^*}\}.$$

The worst-case occurs when $3 - 2\frac{A}{H^*} = \frac{3}{2} + \frac{A}{2H^*}$, i.e. for $\frac{A}{2H^*} = \frac{3}{5}$ and the corresponding bound is $\frac{H}{H^*} \leq \frac{9}{5}$.

Therefore the combination of Large-Arcs and Small-Arcs (Frederickson, Hecht and Kim, 1978) provides a $\frac{9}{5}$-approximation for the Stacker Crane Problem.

Greedy algorithms: example 1

Example 1: the Knapsack Problem. Consider the following greedy algorithm: *Choose the next item of maximum value among those that fit into the residual capacity, until no more item can be inserted*.

Consider this instance:

- item 1 has volume b and value v;
- items $2, \ldots, b+1$ have volume 1 and value $u < v$;
- the capacity of the knapsack is b.

The greedy algorithm selects item 1 and then stops: $x_A = [1, 0, \ldots, 0]$ and $z(A) = v$.

But there exists a feasible solution $x^* = [0, 1, \ldots, 1]$ of value $z^* = bu$.

The approximation factor $\frac{z^* - z(A)}{z^*} = \frac{bu - v}{bu}$ can be arbitrarily close to 1 for very large values for b.

Greedy algorithms: example 2

Example 2: the Knapsack Problem (again). A slightly different greedy algorithm for the KP is the following:
Choose the next item of maximum efficiency among those that fit into the residual capacity, until no more item can be inserted.

The efficiency of an item is the ratio between its value and its volume, i.e. $\frac{c_j}{a_j}$.

This yields an approximation scheme for the KP.

Approximation algorithm for the Knapsack problem

The binary knapsack problem is:

$$
\begin{aligned}
KP)\ \text{maximize} \quad & z = \sum_{j \in \mathcal{N}} c_j x_j \\
\text{s.t.} \quad & \sum_{j \in \mathcal{N}} a_j x_j \leq b \\
& x \in \mathcal{B}^n
\end{aligned}
$$

It can be solved to optimality with a dynamic programming algorithm based on the following recursion:

$$
\begin{cases}
f(j, 0) = 0 \\
f(j, i) = \min\{f(j-1, i-c_j) + a_j, f(j-1, i)\} \quad \forall i = 1, \ldots, z^*
\end{cases}
$$

where $f(j, i)$ is the minimum capacity needed to achieve profit i using the first j elements of $\mathcal{N}$.

The algorithms takes $O(nz^*)$ time.

Bounds on z^*

Observation. Denoting the largest value of the profit vector c by $\overline{c} = \max_{j \in \mathcal{N}}\{c_j\}$, we have:

$$\overline{c} \leq z^* \leq n\overline{c}$$

The first inequality is true under the obvious assumption that all solutions with only one item are feasible (items with $a_j > b$ can be identified and discarded at pre-processing time in $O(n)$).

The second inequality is true because $z^* = \sum_{j \in \mathcal{N}} c_j x_j^* \leq \sum_{j \in \mathcal{N}} c_j \leq \sum_{j \in \mathcal{N}} \overline{c} = n\overline{c}$.

Scaling

We choose a scale factor k and we define modified costs $c'_j = \lfloor \frac{c_j}{k} \rfloor$.
The scaled problem is

$$\begin{aligned} KP') \text{ maximize} \quad & z' = \sum_{j \in \mathcal{N}} c'_j x_j \\ \text{s.t.} \quad & \sum_{j \in \mathcal{N}} a_j x_j \leq b \\ & x \in \mathcal{B}^n \end{aligned}$$

This is still a binary knapsack problem and it can be solved to optimality with the same D.P. algorithm in $O(nz^{*'})$ time.

Observation. Denoting the largest value of the scaled profit vector c' by $\overline{c'} = \max_{j \in \mathcal{N}}\{c'_j\}$, we have:

$$\overline{c'} \leq z^{*'} \leq n\overline{c'}.$$

Therefore the time complexity of the algorithm for KP' is $O(n^2 \frac{\overline{c}}{k})$.

Relationship between z^* and $z^{*'}$

Let $X^{*'}$ be the set of items with $x = 1$ in the optimal solution of KP'.
Let X^* be the set of items with $x = 1$ in the optimal solution of KP.

We can now establish a relationship between $z(X^*)$ and $z(X^{*'})$.

$$\begin{aligned} z(X^{*'}) &= \sum_{j \in X^{*'}} c_j \geq \sum_{j \in X^{*'}} k \lfloor \frac{c_j}{k} \rfloor \geq \sum_{j \in X^*} k \lfloor \frac{c_j}{k} \rfloor \\ &\geq \sum_{j \in X^*} (c_j - k) = \sum_{j \in X^*} c_j - k|X^*| \geq z(X^*) - kn \end{aligned}$$

The absolute error is bounded by kn.
The relative error is bounded by $\frac{kn}{z(X^*)}$, i.e. by $\frac{kn}{\overline{c}}$.

Relationship between z^* and $z^{*'}$

So, if we solve the scaled problem KP' instead of the original problem KP,

- we need $O(n^2 \frac{\overline{c}}{k})$ computing time;
- we achieve an approximation factor $\epsilon = \frac{kn}{\overline{c}}$.

Therefore the computational complexity of the approximation algorithm is

$$O(\frac{n^3}{\epsilon}).$$

This provides a *fully polynomial time approximation scheme (FPTAS)* for problem KP.

Truncated branch-and-bound

Another way to obtain approximated solutions is to truncate the search in branch-and-bound algorithms.

Trivially stopping the algorithm as soon as a given time-out expires does not provide any approximation guarantee.

We can modify the fathoming test, so that we have guarantees on the approximation (but not on computing time).

In case of minimization:

Normal test (optimality):	Modified test ($1/\alpha$ approximation):
if $(LB(P) \geq UB)$ then $Fathom(P)$	if $(LB(P) \geq \alpha UB)$ then $Fathom(P)$