

Discrete Optimization

Quentin Louveaux

ULg - Institut Montefiore

2016

Heuristics and approximation algorithms

Sometimes, an exact approach is **very difficult** to finalize or **computationally too expensive**.

Heuristics

A **heuristic** is an algorithm whose running time is reasonable and that is likely to give a **good feasible solution** most of the time for a given class of problems.

A **meta-heuristic** is a heuristic that is generic and only some parts of the heuristic must be defined in a **problem-specific** way.

Approximation algorithms

An **approximation algorithm** is a problem-specific algorithm that runs in **polynomial time** and such that its outcome has a worst-case **guarantee** compared to the optimal solution.

Heuristics and approximation algorithms

Sometimes, an exact approach is **very difficult** to finalize or **computationally too expensive**.

Heuristics

A **heuristic** is an algorithm whose running time is reasonable and that is likely to give a **good feasible solution** most of the time for a given class of problems.

A **meta-heuristic** is a heuristic that is generic and only some parts of the heuristic must be defined in a **problem-specific** way.

Approximation algorithms

An **approximation algorithm** is a problem-specific algorithm that runs in **polynomial time** and such that its outcome has a worst-case **guarantee** compared to the optimal solution.

Simple greedy heuristics for the TSP

- **Nearest neighbor** : Pick a city, take the **closest remaining city** in the tour
- **Greedy** (Kruskal-like) : Sort the edge costs, pick all edges in **nondecreasing order** without creating a subtour or a degree 3 node
- **Nearest insertion** : Start from the cheapest edge, and insert in the tour the **closest node** to the tour.
- **Farthest insertion** : Same except that we insert the **farthest** node in the **cheapest** way.

Simple greedy heuristics for the TSP

- **Nearest neighbor** : Pick a city, take the **closest remaining city** in the tour
- **Greedy** (Kruskal-like) : Sort the edge costs, pick all edges in **nondecreasing order** without creating a subtour or a degree 3 node
- **Nearest insertion** : Start from the cheapest edge, and insert in the tour the **closest node** to the tour.
- **Farthest insertion** : Same except that we insert the **farthest** node in the **cheapest** way.

Simple greedy heuristics for the TSP

- **Nearest neighbor** : Pick a city, take the **closest remaining city** in the tour
- **Greedy** (Kruskal-like) : Sort the edge costs, pick all edges in **nondecreasing order** without creating a subtour or a degree 3 node
- **Nearest insertion** : Start from the cheapest edge, and insert in the tour the **closest node** to the tour.
- **Farthest insertion** : Same except that we insert the **farthest** node in the **cheapest** way.

Simple greedy heuristics for the TSP

- **Nearest neighbor** : Pick a city, take the **closest remaining city** in the tour
- **Greedy** (Kruskal-like) : Sort the edge costs, pick all edges in **nondecreasing order** without creating a subtour or a degree 3 node
- **Nearest insertion** : Start from the cheapest edge, and insert in the tour the **closest node** to the tour.
- **Farthest insertion** : Same except that we insert the **farthest** node in the **cheapest** way.

Local search

Consider the problem

$$\begin{array}{ll}\min & c(x) \\ \text{subject to} & x \in \mathcal{F}\end{array}$$

Idea :

Start from a given **feasible solution** x .

Define a **neighborhood** $N(x)$ of x .

Pick $y \in N(x)$.

If $c(y) < c(x)$, **then** $x := y$ is the new incumbent.

Repeat until some convergence criterion is reached.

Local search

Fundamental question : define a **neighborhood**.

Examples : TSP : 2-opt, 3-opt, matching neighborhood
 k -opt for binary optimization.

Issue : One is often trapped in a local minimum.

Very-large neighborhood search

In this framework, the idea is to define a **very large neighborhood** and to rely on generic solver or known well-solved problems to deal with it.

Example : If we want to solve an MIP

$$\begin{aligned} \min \quad & c^x \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \{0, 1\}^n \end{aligned}$$

From a given solution x^* , we can define a large neighborhood by splitting the variable set into two and **fixing alternatively** each set of variables.

Each subproblem is then solved using a branch-and-bound algorithm.

This is also very popular to combine with **constraint programming**.

In all these methods, it is important to know the **structure** of the problem for the heuristic to make sense.

Feasibility pump

This is an MIP-based method to find good **feasible solutions**.

We want to solve

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \{0, 1\}^n \end{aligned}$$

1. Solve the LP relaxation. Let x^* be an optimal solution.
2. For T iterations, do the sequence (2. - 3.)
Round x^* componentwise to the nearest integer : $\tilde{x}_j = \lfloor x_j^* \rfloor$.
3. Solve minimize $\sum_{j=1}^n |x_j - \tilde{x}_j|$ subject to $x \in LP$.

Feasibility pump

This is an MIP-based method to find good **feasible solutions**.

We want to solve

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \{0, 1\}^n \end{aligned}$$

1. Solve the LP relaxation. Let x^* be an optimal solution.
2. For T iterations, do the sequence (2. - 3.)
Round x^* componentwise to the nearest integer : $\tilde{x}_j = \lfloor x_j^* \rfloor$.
3. Solve minimize $\sum_{j=1}^n |x_j - \tilde{x}_j|$ subject to $x \in LP$.

Feasibility pump

This is an MIP-based method to find good **feasible solutions**.

We want to solve

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \{0, 1\}^n \end{aligned}$$

1. Solve the LP relaxation. Let x^* be an optimal solution.
2. For T iterations, do the sequence (2. - 3.)
Round x^* componentwise to the nearest integer : $\tilde{x}_j = \lfloor x_j^* \rfloor$.
3. Solve minimize $\sum_{j=1}^n |x_j - \tilde{x}_j|$ subject to $x \in LP$.

Simulated annealing

Metaheuristic based on local search that tries to escape from local minima.
Inspired by physical analogy.

Simulated annealing algorithm

- **Start** with $x \in \mathcal{F}$ and an initial temperature T
- **Repeat** until some convergence criterion is reached
- **Pick** randomly $y \in N(X)$
- **If** $c(y) < c(x)$, **then** $x := y$
- **If** $c(y) > c(x)$, **then accept** $x := y$ with probability $e^{(c(x)-c(y))/T}$
- **Decrease** the temperature T

Simulated annealing

Let $A = \sum_{z \in \mathcal{F}} e^{-c(z)/T}$ and

$$\pi(x) = \frac{e^{-c(x)/T}}{A}.$$

Theorem

Assume that the Markov chain $x(t)$ is irreducible, then the unique steady distribution of the Markov chain is the vector with components $\pi(x), x \in \mathcal{F}$.

3/2-approximation algorithm for TSP

- Compute a **minimum spanning tree** T .
- Let S be the set of nodes with **odd degree** in T .
Compute a min cost matching M of the nodes in S
- All nodes in $T \cup M$ have even degree, create a closed walk using all edges in $T \cup M$.
Shortcut the closed walk to form a tour.

Approximation algorithms

Scheduling with precedence constraints

We have n jobs with processing times p_i and weights w_i .

We want to schedule the jobs on a **single machine** and respect the **precedence constraints** while minimizing

$$\sum_i w_i C_i$$

where C_i is the **completion time** of job i .

Formulation

$$\min \sum_i w_i C_i$$

$$\text{s.t. } \sum_{i \in S} p_i C_i \geq \frac{1}{2} \sum_{i \in S} p_i^2 + \frac{1}{2} \left(\sum_{i \in S} p_i \right)^2 \quad S \subset V$$

$$C_j \geq C_i + p_j \quad \text{if } i \text{ precedes } j$$

Approximation algorithms

Scheduling with precedence constraints

We have n jobs with processing times p_i and weights w_i .

We want to schedule the jobs on a **single machine** and respect the **precedence constraints** while minimizing

$$\sum_i w_i C_i$$

where C_i is the **completion time** of job i .

Formulation

$$\min \sum_i w_i C_i$$

$$\text{s.t. } \sum_{i \in S} p_i C_i \geq \frac{1}{2} \sum_{i \in S} p_i^2 + \frac{1}{2} \left(\sum_{i \in S} p_i \right)^2 \quad S \subset V$$

$$C_j \geq C_i + p_j \quad \text{if } i \text{ precedes } j$$

Rounding algorithm

Deterministic rounding algorithm

- **Solve** the linear programming relaxation
Obtain C^*
- **Sort** the C_i^* in increasing order and create a feasible schedule by using the **same order**.
(The precedence constraints will be automatically satisfied)

Theorem

Let Z_H be the weighted completion time of the schedule produced by the algorithm, and Z_{LP} the value of the relaxation. Then

$$\frac{Z_H}{Z_{LP}} \leq 2.$$

Rounding algorithm

Deterministic rounding algorithm

- **Solve** the linear programming relaxation
Obtain C^*
- **Sort** the C_i^* in increasing order and create a feasible schedule by using the **same order**.
(The precedence constraints will be automatically satisfied)

Theorem

Let Z_H be the weighted completion time of the schedule produced by the algorithm, and Z_{LP} the value of the relaxation. Then

$$\frac{Z_H}{Z_{LP}} \leq 2.$$

Proof : Let us assume $C_1^* \leq C_2^* \leq \dots \leq C_n^*$.

The completion time of job j is $\bar{C}_j = \sum_{k=1}^j p_k$.

$$\begin{aligned} C_j^* \sum_{k=1}^j p_k &\geq \sum_{k=1}^j p_k C_k^* \text{ since } C_j^* \geq \dots \geq C_1^* \\ &\geq \frac{1}{2} \sum_{k=1}^j p_k^2 + \frac{1}{2} \left(\sum_{k=1}^j p_k \right)^2 \\ &\geq \frac{1}{2} \left(\sum_{k=1}^j p_k \right)^2 \end{aligned}$$

and thus

$$C_j^* \geq \frac{1}{2} \sum_{k=1}^j p_k = \frac{1}{2} \bar{C}_j.$$

Approximation schemes

- A family of algorithms \mathcal{A}_ϵ is a **polynomial time approximation scheme (PTAS)** if for every $\epsilon > 0$, \mathcal{A}_ϵ is an ϵ -approximation algorithm and its running time is polynomial for fixed ϵ .
- If \mathcal{A}_ϵ is a PTAS and its running time is also polynomial in $1/\epsilon$, then it is a **fully polynomial time approximation scheme (FPTAS)**

Approximation schemes

- A family of algorithms \mathcal{A}_ϵ is a **polynomial time approximation scheme (PTAS)** if for every $\epsilon > 0$, \mathcal{A}_ϵ is an ϵ -approximation algorithm and its running time is polynomial for fixed ϵ .
- If \mathcal{A}_ϵ is a PTAS and its running time is also polynomial in $1/\epsilon$, then it is a **fully polynomial time approximation scheme (FPTAS)**

The bin packing problem

n items with sizes s_1, \dots, s_n .

Find the minimum number of bins needed to **pack them**.

The first fit algorithm

Let j be the **first bin** in which item i **fits**.

Place item i in bin j .

It is an asymptotic approximation scheme.

Proposition

$$z_H \leq 2Z_{IP} + 1$$

Proposition

If we can approximate the bin packing problem with approximation ratio $\alpha < 3/2$, then $P = NP$.

The bin packing problem

n items with sizes s_1, \dots, s_n .

Find the minimum number of bins needed to **pack them**.

The first fit algorithm

Let j be the **first bin** in which item i **fits**.

Place item i in bin j .

It is an asymptotic approximation scheme.

Proposition

$$z_H \leq 2Z_{IP} + 1$$

Proposition

If we can approximate the bin packing problem with approximation ratio $\alpha < 3/2$, then $P = NP$.

The bin packing problem

n items with sizes s_1, \dots, s_n .

Find the minimum number of bins needed to **pack them**.

The first fit algorithm

Let j be the **first bin** in which item i **fits**.

Place item i in bin j .

It is an asymptotic approximation scheme.

Proposition

$$z_H \leq 2Z_{IP} + 1$$

Proposition

If we can approximate the bin packing problem with approximation ratio $\alpha < 3/2$, then $P = NP$.