Sub-Optimality Approximations

Russell Bent[1], Irit Katriel[2], and Pascal Van Hentenryck[1]

[1] Brown University, Box 1910 Providence, RI 02912, USA
[2] Max-Plank-Institut für Informatik, Saarbrücken, Germany

Abstract. The sub-optimality approximation problem considers an optimization problem $\mathcal{O}$, its optimal solution σ^*, and a variable x with domain $\{d_1, \ldots, d_m\}$ and returns approximations to $\mathcal{O}[x \leftarrow d_1], \ldots, \mathcal{O}[x \leftarrow d_m]$, where $\mathcal{O}[x \leftarrow d_1]$ denotes the problem $\mathcal{O}$ with x assigned to d_i. The sub-optimality approximation problem is at the core of online stochastic optimization algorithms and it can also be used for solution repair and approximate filtering of optimization constraints. This paper formalizes the problem and presents sub-optimality approximation algorithms for metric TSPs, packet scheduling, and metric k-medians that run faster than the optimal or approximation algorithms. It also presents results on the hardness/easiness of sub-optimality approximations.

1 Introduction

In an increasing dynamic, interconnected, and real-time world, optimization technology faces new challenges and opportunities. Indeed, on many applications, it is no longer sufficient to produce optimal, or near-optimal, solutions offline. Optimization software should adapt dynamically to uncertainties, update existing solutions to accommodate new requests and events, and produce high-quality decisions under severe time constraints.

This paper introduces the sub-optimality approximation problem, which is at the core of many online and dynamic applications. Given an optimization problem $\mathcal{O}$ and an optimal solution σ^* to $\mathcal{O}$, the sub-optimality approximation problem consists of approximating the problems $\mathcal{O}[x \leftarrow d_1], \ldots, \mathcal{O}[x \leftarrow d_m]$, where x is a decision variable, $\{d_1, \ldots, d_m\}$ are its possible values, and $O[x \leftarrow d]$ denotes the problem $\mathcal{O}$ where x is assigned to d. The key property of the sub-optimality approximation problem is the availability of the optimal solution σ^*. Since each problem $\mathcal{O}[x \leftarrow d_i]$ is closely related to $\mathcal{O}$, the optimal solution is, in general, of tremendous help for the sub-optimality approximations. However, to be useful, a sub-optimality approximation algorithm should be faster than traditional approximation algorithms. This performance requirement is formalized by the concept of amortized sub-optimality algorithm that finds approximations to $\mathcal{O}[x \leftarrow d_1], \ldots, \mathcal{O}[x \leftarrow d_m]$ in the time it takes to solve $\mathcal{O}$ optimally or approximately. The sub-optimality approximation problem is a critical component of the regret algorithm for online stochastic optimization [3, 6]. It can also be used for solution repair, for evaluating the robustness of solutions, and for approximate filtering of optimization constraints to name a few of its applications.

This paper makes three contributions. First, it identifies and formalizes the sub-optimality approximation problem and demonstrates its relevance for a number of applications. Second, it presents amortized sub-optimality approximation algorithms for three problems: the metric TSP, packet scheduling in networks, and the k-median problem. The proof techniques are interesting in the sense they reason about the optimal solutions to both $\mathcal{O}$ and $\mathcal{O}[x \leftarrow d]$. Third, it presents results on the hardness and easiness of sub-optimality approximations, showing that some "hard" problems become "easy", while others remain "hard".

This paper is organized as follows. Sections 2 and 3 formalize the problem and discuss its applications. Section 4, 5, and 6 present sub-optimality approximations for the metric TSP, the packet scheduling, and the metric k-median problem. Section 7 presents the hardness results.

2 Amortized Sub-Optimality Approximation Algorithms

This section formalizes sub-optimality approximation problems and algorithms. The formalization uses the definition of CSPs from [15], where the set of constraints is abstracted by a Boolean function which holds if all the constraints are satisfied (since we are not interested in the constraint structure). Solutions are also represented as functions (assignments) from variables to their sets of values.

Definition 1. *A* CSP *is a triplet $\langle V, D, C\rangle$, where V denotes the set of variables, D denotes the set of possible values for these variables, and $C : (V \rightarrow D) \rightarrow Bool$ is a constraint which specifies which assignments of values to the variables are solutions. A solution to a CSP $\mathcal{P} = \langle V, D, C\rangle$ is a function $\sigma : V \rightarrow D$ such that $C(\sigma) = true$. The set of solutions to a CSP $\mathcal{P}$ is denoted by $Sol(\mathcal{P})$.*

Constraint Optimization Problems (COPs) are CSPs with an objective function.

Definition 2. *A* COP *is a pair $\langle \mathcal{P}, f\rangle$, where $\mathcal{P} = \langle V, D, C\rangle$ is a CSP and $f : V \rightarrow \mathcal{N}$ is an objective function. A solution to a COP $\mathcal{O} = \langle \mathcal{P}, f\rangle$ is a solution to $\mathcal{P}$. An optimal solution to $\mathcal{O}$ is a solution of $\mathcal{O}$ that minimizes f. The sets of solutions and optimal solutions to a COP $\mathcal{O}$ are denoted by $Sol(\mathcal{O})$ and $OptSol(\mathcal{O})$ respectively.*

Given a CSP $\mathcal{P} = \langle V, D, C\rangle$, $var(\mathcal{P})$ denotes the variables V and $dom(\mathcal{P})$ the domain D. Similar notations are used for COPs, the variables and values of a COP being those of its underlying CSP. Sub-optimality approximation problems consider sets of related CSPs where one variable is assigned different values. Given a CSP $\mathcal{P} = \langle V, D, C\rangle$, $\mathcal{P}[x \leftarrow d]$ ($x \in V$ & $d \in D$) denotes the CSP $\mathcal{P}$ where variable x is assigned the value d, i.e., the CSP $\langle V, D, C \wedge x = d\rangle$. Similarly, given a COP $\mathcal{O} = \langle \mathcal{P}, f\rangle$, $\mathcal{O}[x \leftarrow d]$ ($x \in V$ & $d \in D$) denotes the COP $\mathcal{O}$ where variable x is assigned the value d, i.e., the COP $\langle \mathcal{O}[x \leftarrow d], f\rangle$.

We are ready to specify the sub-optimality approximation problem. Informally, given an optimal solution σ^* to a COP $\mathcal{O}$, the sub-optimality approximation problem consists of finding constant factor approximations to the COPs $\mathcal{O}[x \leftarrow d_i]$ for all values $d_1, \ldots, d_m$ of variable x.

Definition 3 (The Sub-Optimality Approximation Problem). *A sub-optimality approximation problem receives as input a COP $\mathcal{O} = \langle \mathcal{P}, f\rangle$ with*

$dom(\mathcal{P}) = \{d_1, \ldots, d_m\}$, an optimal solution $\sigma^* \in OptSol(\mathcal{O})$, and a variable x in $var(\mathcal{P})$. Its output is a set of solutions

$$\tilde{\sigma}_i \in Sol(\mathcal{P}[x \leftarrow d_i]) \quad (1 \leq i \leq m)$$

satisfying

$$f(\tilde{\sigma}_i) \leq \beta f(\sigma_i^*)$$

for some constant β, where $\sigma_i^* \in OptSol(\mathcal{O}[x \leftarrow d_i])$.

The fundamental property in the sub-optimality approximation problem is the fact that the input contains an optimal solution to $\mathcal{O}$. This solution should of course be used by the sub-optimality approximation algorithm in order to solve the COPs $\mathcal{O}[x \leftarrow d_i]$ efficiently. Observe that the definition can easily be generalized to accommodate stronger or weaker approximation requirements.

To capture performance requirements of great benefit in practical applications, we introduce the concept of amortized sub-optimality approximation algorithms. The intuition here is that a sub-optimality algorithm is amortized if it approximates the solutions of $\mathcal{O}[x \leftarrow d_1], \ldots, \mathcal{O}[x \leftarrow d_m]$ in the time it takes to solve $\mathcal{O}$ optimally. It is strongly amortized if it approximates such solutions for all variables in the same time.

Definition 4 (Amortized Sub-Optimality Approximation). *Consider a class $\mathcal{C}$ of COPs, let $\mathcal{A}$ be an algorithm for solving $\mathcal{C}$ in time $O(g)$, and let $\tilde{\mathcal{A}}$ be a sub-optimality approximation algorithm for class $\mathcal{C}$ that runs in time $O(\tilde{g})$. Let $|dom(\mathcal{O})| = m$ and let $|var(\mathcal{O})| = n$. Algorithm $\tilde{\mathcal{A}}$ is amortized wrt $\mathcal{A}$ on class C if, for each COP $\mathcal{O} \in \mathcal{C}$ with $|dom(\mathcal{O})| = m$, we have that $m\,\tilde{g}$ is $O(g)$.. It is strongly amortized wrt $\mathcal{A}$ on class C if $nm\,\tilde{g}$ is $O(g)$.*

These definitions can be generalized to the important case where the COP is solved through an approximation algorithm with performance guarantees. This is especially significant in online optimization under strict time constraints where optimal solutions can rarely be obtained within the time limits.

Definition 5 (The Sub-Optimality (α,β)-Approximation Problem). *A sub-optimality (α, β)-approximation problem receives as input a COP $\mathcal{O} = \langle \mathcal{P}, f \rangle$ with $dom(\mathcal{P}) = \{d_1, \ldots, d_m\}$, an approximation $\tilde{\sigma}$ satisfying $f(\tilde{\sigma}) \leq \alpha f(\sigma^*)$ for $\sigma^* \in OptSol(\mathcal{O})$, and x in $var(\mathcal{P})$. Its output is the solutions $\tilde{\sigma}_i \in Sol(\mathcal{P}[x \leftarrow d_i]) \quad (1 \leq i \leq m)$ satisfying $f(\tilde{\sigma}_i) \leq \beta f(\sigma_i^*)$ and $\sigma_i^* \in OptSol(\mathcal{O}[x \leftarrow d_i])$.*

The concept of amortized sub-optimality (α, β)-approximation is similar to Definition 4, although its requirements are typically much stricter. However, as shown later, the same sub-optimality approximation may apply to both problems.

3 Applications

This section reviews a number of applications that benefit from sub-optimality approximation algorithms to demonstrate its relevance and applicability. The section does not aim to be comprehensive but to give some indication of where sub-optimality approaximations may be beneficial.

Online Stochastic Optimization Our primary motivation for sub-optimality optimization came from online stochastic optimization. Online optimization problems (e.g., [11]) is a class of applications where the data is revealed online during the execution of the decision-making process. In many of these applications [8, 3, 4], a distribution of the data, or an approximation thereof, is available to the algorithm for sampling. Alternatively, the data distribution can be learned during the algorithm execution [6]. A natural framework for online stochastic optimization was defined in [5, 3, 2] and only its most basic version is considered here for simplicity. The key idea behind the framework is to consider a time interval H and to allow a single request to be served at each time $t \in H$. The selected request (if any) is selected from the set of available requests R at time t. Each request r has a weight $w(r)$ that specifies how valuable it is. Which requests may be served is problem-specific and left unspecified in the framework. The framework simply assumes that the underlying algorithms have access to two black-boxes: an optimization algorithm that can find an optimal solution for a set of requests and a distribution that can be sampled to obtain scenarios reflecting the future (to some degree). The goal of the online algorithms is to a choose requests online to maximize the weighted sum of the serviced requests. More formally, the algorithms are all instantiations of the online schema:

```
onlineOptimization($H$)
1 $R \leftarrow \emptyset$;
2 $w \leftarrow 0$;
3 for $t \in H$
4     do $R \leftarrow$ availableRequests($R, t$) $\cup$ newRequests($t$);
5         $r \leftarrow$ chooseRequest($R, t$);
6         serveRequest($r, t$);
7         $w \leftarrow w + w(r)$;
8         $R \leftarrow R \setminus \{r\}$;
```

but they differ in how they implement function chooseRequest. The online optimization schema considers the set of available requests (i.e. those requests that may be served at time t without violating any constraints) and new requests at each time step. It chooses a request r which is then served and removed from the set of available requests. Function availableRequest(R, t) returns the set of requests available for service at time t and function serveRequest(r, t) simply serves r at time t (i.e., $\sigma(t) \leftarrow r$). To implement function chooseRequest, the algorithms have at their disposal two black-boxes:

1. A function optimalSolution($R, t, \mathit{\Delta}$) that, given a set R of requests, a time t, and a number $\mathit{\Delta}$, returns an optimal solution for R over $[t, t + \mathit{\Delta}]$;
2. A function getSample($[t_s, t_e]$) that returns a set of requests over the interval $[t_s, t_e]$ by sampling the arrival distribution.

Typically, the goal is to choose a request at time t that maximizes expectation. The exact computation of the expected value of servicing a request is often too computationally demanding and one of the traditional approaches approximates expectation by evaluating each decision with respect to samples from the distribution (Algorithm E) [8]. A simple implementation is as follows:

```
chooseRequest-E($R, t$)
1 for $r \in R$
2     do $f(r) \leftarrow 0$;
3 for $i \leftarrow 1 \ldots \mathcal{O}/|R|$
4     do $S \leftarrow R \cup$ getSample($[t+1, t+\Delta]$);
5         for $r \in R$
6             do $f(r) \leftarrow f(r) + (w(r) + W($optimalSolution$(S \setminus \{r\}, t+1)))$;
7 return $argmax(r \in R)\ f(r)$;
```

Lines 1-2 initialize the evaluation function $f(r)$ for each request r. The algorithm then generates a number of samples for future requests (line 3). For each such sample, it computes the set R of all available and sampled requests at time t (line 4). The algorithm then considers each available request r successively (line 5), it implicitly schedules r at time t, and applies the optimal offline algorithm using $S \setminus \{r\}$ and the time horizon. The evaluation of request r is updated in line 6 by incrementing it with its weight and the score of the corresponding optimal offline solution. All samples are evaluated for all available requests and the algorithm then returns the request $r \in R$ with the highest evaluation. Observe Line 3 of Algorithm E which distributes the available offline optimizations across all available requests. The expectation algorithm is typically too computationally demanding for an online setting as each evaluation of a request on a sample requires an optimization. A recent advance is the regret algorithm (R), where each sample is solved optimally once and sub-optimality approximations are used to evaluate the remaining requests [3]:

```
chooseRequest-R($R, t$)
1 for $r \in R$
2     do $f(r) \leftarrow 0$;
3 for $i \leftarrow 1 \ldots \mathcal{O}$
4     do $S \leftarrow R \cup$ getSample($[t+1, t+\Delta]$);
5         $\sigma^* \leftarrow$ optimalSolution($S, t$);
6         $f(\sigma^*(t)) \leftarrow f(\sigma^*(t)) + W(\sigma)$;
7         for $r \in R \setminus \{\sigma^*(t)\}$
8             do $f(r) \leftarrow f(r) + W($suboptimalityApproximation$(\sigma^*, r, S, t))$;
9 return $argmax(r \in R)\ f(r)$;
```

Algorithm R (lines 7-8) computes an approximation of the best solution of s serving r at time t, i.e., $W($suboptimalityApproximation$(\sigma^*, r, S, t))$. Hence, the value of scheduling each available request is approximated on every sample at time t for the cost of a single offline optimization (asymptotically). Observe that the regret algorithm solves the sub-optimality problem where variable $\sigma(t)$ is assigned the values $R \setminus \{\sigma(r)\}$ and that the optimal solution σ^*, or an approximation thereof, is naturally available since the sample is solved in line 5. By solving the sub-optimality problem, algorithm R enjoys essentially the same theoretical performance guarantees as algorithm E at a fraction of the cost [7].

Solution Repair Solution repair is another important application for sub-optimality approximations. For example, a catastrophic hub failure in a network may require a nearby hub to be opened quickly. Such applications are often modeled as dynamic facility location problems where one must quickly approximate the optimal solution to the problem in which a facility is forced to be closed or to be opened. Once again, the optimal solution, or an approximation thereof, is naturally available, and one is interested in solving the sub-optimality approximation problem for specific hubs. Solution repair is closely related to the issue of robustness. Sub-optimality approximations provide a computational method to evaluate the robustness of different optimal, or locally optimal, solutions. Solutions with small sub-optimality gaps may be preferred, since they entail smaller quality loss when some variables cannot be assigned some values. Again, optimal or approximated solutions are naturally available in these applications.

Partial or Approximate Filtering Sub-pOtimality approximations are also useful for partial or approximate filterings of optimization constraints (e.g., [12]). An optimization constraint captures a combinatorial substructure arising in many applications and can be specified as an optimization problem $\mathcal{O} = \langle \mathcal{P}, f \rangle$. During the search, an optimization constraint uses bounds on f to detect infeasibility and prunes the domains of the variables. Consider a minimization constraint $\mathcal{O} = \langle \mathcal{P}, f \rangle$ and assume that U is an upper bound on f. Typically, the minimization constraint searches for an optimal solution o^* to $\mathcal{O}$ to detect feasibility, i.e., $f(o^*) \leq U$. Once feasibility is established, the constraint filters the domains of the variables to remove all values that cannot appear in any solution not greater than U. Here sub-optimality approximation can be used to detect quickly values that can, or cannot, be filtered. Assume that the sub-optimality algorithm is a ρ-approximation and consider a variable x with domain D. The sub-optimality approximation problem provides a solution $\tilde{o}_d$ to $\mathcal{O}[x \leftarrow d]$ satisfying $f(\tilde{o}_d) \leq \rho f(o_d)$, where o_d is an optimal solution to $\mathcal{O}[x \leftarrow d]$. Hence, the value d cannot be filtered whenever $f(\tilde{o}_d) \leq U$ and must be filtered whenever $f(\tilde{o}_d) > \rho U$. Once again, observe that the optimal solution o^* is naturally available.

4 The Travelling Salesman Problem

The traveling salesman problem (TSP) is probably the most studied combinatorial optimization problem. It is also an important component in a wide variety of online applications, such as courier services.

The Sub-Optimality Approximation Problem The sub-optimality approximation problem consists of approximating the cost of assigning different successors to a vertex i. In other words, the variable under consideration is the successor of vertex i and the domains are all other vertices.

The Sub-Optimality Approximation Algorithm A simple relocation provides a sub-optimality approximation algorithm to bound the effect of traveling to customer j after i: remove j from the optimal solution and reinsert it after i. This amortized algorithm is a constant factor approximation for the Euclidean TSP.

Theorem 1. [Amortized Sub-Optimality for the Metric TSP] *The metric TSP has a strongly amortized sub-optimality (α,β)-approximation algorithm.*

Proof. Let σ^* be the optimal solution to a TSP and let σ_{ij} be the optimal solution when j must follow i. The optimal solution σ^* consists of a tour with length $C(\sigma) = c_{i,i+} + C_{i+,j-} + c_{j-,j} + c_{j,j+} + C_{j+,i}$, where $C_{i,j}$ (resp. $c_{i,j}$) denotes the cost of the path (resp. arc) between i and j in σ^*. The approximation solution consists of a tour with length $C(\tilde{\sigma}_{x_{ij}}) = c_{i,j} + c_{j,i+} + C_{i+,j-} + c_{j-,j+} + C_{j+,i}$. By the triangle inequality,

$$\begin{aligned}
C(\tilde{\sigma}_{ij}) &= c_{i,j} + c_{j,i+} + C_{i+,j-} + c_{j-,j+} + C_{j+,i} \\
&\leq c_{i,j} + c_{j,i+} + C_{i+,j-} + c_{j-,j} + c_{j,j+} + C_{j+,i} \\
&\leq c_{i,i+} + C_{i+,j-} + c_{j-,j} + c_{i+,j} + C_{i+,j-} + c_{j-,j} + c_{j,j+} + C_{j+,i} \\
&\leq c_{i,i+} + C_{i+,j-} + c_{j-,j} + C_{i+,j-} + c_{j-,j} + C_{i+,j-} + c_{j-,j} + c_{j,j+} + C_{j+,i} \\
&\leq c_{i,i+} + 3C_{i+,j-} + 3c_{j-,j} + c_{j,j+} + C_{j+,i} \\
&\leq 3\ C(\sigma) \\
&\leq 3\ C(\sigma_{ij}) \text{ by optimality of } \sigma.
\end{aligned}$$

Now assume that $\tilde{\sigma}$ is an α-approximation of σ^*. We have that $C(\tilde{\sigma}_{ij}) \leq 3\ C(\tilde{\sigma})$ by the above proof and hence $C(\tilde{\sigma}_{ij}) \leq 3\ \alpha\ C(\tilde{\sigma}) \leq 3\ \alpha\ C(\tilde{\sigma}_{ij})$. Each such approximation takes $O(1)$ time. The algorithm is strongly amortized wrt all approximation algorithms (which are $\Omega(|E|)$, where E is the set of arcs). □

5 Packet Scheduling

The Optimization Problem This section considers a simple scheduling problem used to model a variety of applications, including the packet scheduling problem from [8]. The problem is given as inputs a set R of tasks/requests for service and a time horizon $H = [\underline{H}, \overline{H}]$ during which requests must be scheduled. Each request r is characterized by a weight $w(r)$ and an arrival time $a(r)$, requires a single time unit to be processed, and must be scheduled in its time window $[a(r), a(r)+d]$. In other words, the request is lost if it is not served within its time window. In addition, no two requests can be scheduled at the same time. The goal is to find a schedule of maximal weight, i.e., a schedule which maximizes the sum of the weights of all scheduled requests. This is equivalent to minimizing weighted loss. More formally, assume for simplicity and without loss of generality, that there is a request scheduled at each time step. Under this assumption, a schedule is a function $\sigma : H \to R$ which assigns a request to each time in the schedule horizon. A schedule σ is feasible if it satisfies the constraints

$$\begin{aligned}
&\forall\ t_1, t_2 \in H : t_1 \neq t_2\ \ \to \sigma(t_1) \neq \sigma(t_2) \\
&\forall\ t \in H : a(\sigma(t)) \leq t \leq a(\sigma(t)) + d.
\end{aligned}$$

The weight of a schedule σ, denoted by $W(\sigma)$, is given by

$$W(\sigma) = \sum_{t \in H} w(\sigma(t)).$$

and the goal is to find a feasible schedule σ maximizing $W(\sigma)$. This offline problem can be solved in quadratic time $O(|R||H|)$ [8].

The Sub-Optimality Approximation Problem The sub-optimality approximation problem for packet scheduling is motivated by online stochastic optimization, where future packets are known in advance and are revealed online as the algorithm makes decision. As discussed in Section 3, it is highly beneficial in online optimization to use stochastic information and to evaluate many scenarios at a time t in order to select a good packet to schedule. However, due to severe time constraints, only a few optimizations can be executed and the regret algorithm uses the sub-optimality approximation algorithm to estimate the value of scheduling all packets on all scenarios for the cost of one optimization. As a consequence, the sub-optimality approximation problem is given a set of request R and an optimal solution σ^* for scheduling these requests in the time horizon $[t, \overline{H}]$. For each request $r \in R$ available at time t, it must approximate the optimal schedule σ_r that schedules request r at time t, i.e., $\sigma_r(t) = r$. The results also generalize to arbitrary times in H.

The packet scheduling problem is an interesting case study because its offline algorithm takes quadratic time and hence an amortized sub-optimality approximation must approximate $|R|$ schedules within the same time bounds.

The Amortized Sub-Optimality Approximation The sub-optimality approximation consists of swapping a constant number of requests in the optimal schedule σ^* at a time t and performs a case analysis on the properties of the request r.

If a request r is not scheduled (i.e., $r \notin \sigma^*$), the key idea is to try rescheduling the request $\sigma^*(t)$ instead of the request of smallest weight in the schedule σ^*. The value of the sub-optimality approximation becomes

$$W(\sigma^*) - \min(s \in [t, a(\sigma^*(t)) + d])\ w(\sigma^*(s)) - w(r),$$

since the replaced request is removed from σ^* and r is added to the schedule. In the worst case, the replaced request is $\sigma^*(t)$ and the approximation is $W(\sigma^*) - (w(\sigma^*(t)) - w(r))$.

If request r is scheduled at time t_r, the sub-optimality approximation first tries to swap r and $\sigma^*(t)$ in which case the approximation is $W(\sigma^*)$. If this is not possible, the approximation tries rescheduling $\sigma^*(t)$ instead of the request of smallest weight in σ^*. If $\sigma^*(t)$ cannot be rescheduled, the approximation simply selects the best possible unscheduled request which may be scheduled at t_r and the approximation is

$$W(\sigma^*) - (w(\sigma^*(t)) - \max(u \in U_r)\ w(u))$$

where $U_r = \{r \mid a(r) \leq t_r \leq a(r) + d \ \wedge\ r \notin \sigma^*\}$, If $\sigma^*(t)$ is rescheduled at time s, then the approximation concludes by selecting the best possible unscheduled request which may be scheduled at t_r and the approximation is

$$W(\sigma^*) - (w(\sigma^*(s)) - \max(u \in U_{r,s})\ w(u))$$

where $U_{r,s} = \{r \mid a(r) \leq t_r \leq a(r) + d \ \wedge\ (r \notin \sigma^* \ \vee\ r = \sigma^*(s))\}$. Each sub-optimality approximation takes $O(d)$ time and is performed at most $|R|$ times (typically much less than $|R|$ since only one request of the same class must be evaluated). Thus all the approximations take $O(d|R|)$ time, which is $O(|R||H|)$ and is negligible in practice for this application. Theorem 2 shows that this amortized algorithm produces a 2-approximation.

Theorem 2. *Packet scheduling has an amortized suboptimality approximation.*

Proof. Let $r \in R$ be a request that can be scheduled at time t and σ^* be an optimal solution. let σ_r be an optimal solution when r is scheduled at time t (i.e., $\sigma_r(t) = r$) and let $\tilde{\sigma_r}$ be the solution obtained by the sub-optimality approximation. This theorem shows that

$$\frac{w(\sigma_r)}{w(\tilde{\sigma_r})} \leq 2.$$

Most of the proof consists of showing that, for each lost request x, where x is typically $\sigma^*(t)$, there is another request in σ^* whose weight is at least $w(x)$ yielding a 2-approximation since $w(\sigma_r) \leq w(\sigma^*)$.

First observe that the result holds when $w(x) \leq w(r)$ since, in the worst case, the sub-optimality approximation only loses request x. So attention is restricted to $w(x) \geq w(r)$. If $x \in \tilde{\sigma_r}$, i.e., if the sub-optimality approximation swaps x with another request y (case 1), the result also holds since $w(y) \leq w(x)$. If $x \notin \tilde{\sigma_r}$ and x can be scheduled at a time other than t, it means that there exists a request y at each of these times satisfying $w(y) \geq w(x)$ and the result holds. It thus remains to consider the case where x can only be scheduled at time t and is thus lost in σ_r. If $r \notin \sigma^*$, the sub-optimality approximation is optimal, since otherwise r would be in the optimal schedule at a time other than t. Otherwise, it is necessary to reason about a collection of requests. Indeed,

$$w(\sigma^*) = w(x) + w(r) + w(S),$$

where $S = \{p \in \sigma^* \;\mid\; p \neq x \;\&\; p \neq y\}$. It is also known that $w(\tilde{\sigma_r}) \geq w(r) + w(S)$ since, in the worst case, the approximation loses request x. Finally, $w(\sigma_r) = w(r) + w(Z)$ where Z are the requests scheduled after time t. Since σ^* is optimal, we have $w(Z) \leq w(r) + w(S)$ and the result follows. □

Experimental Results Figure 1 (taken from [3]) shows the significance of the sub-optimality approximation problem for online stochastic optimization. The plot depicts the performance of various algorithms as a function of the number of optimizations available for each decision. It considers two oblivious algorithms: greedy (G) which always schedules the available packet of highest weight, local optimization (LO) which uses the result of the optimization on the known requests to select the packet to schedule at time t. It also considers two stochastic algorithms: expectation (E) which runs the optimization algorithm when each available request is scheduled at time t in each scenario, and the regret algorithm (R) which solves each scenario once and uses the sub-optimality approximation to evaluate each available request. The figure also displays the optimal, a posteriori, solution, i.e., the solution that would be obtained if all requests had been known in advance. As can be seen from this plot, the regret algorithm provides great benefits over all the other algorithms. In particular, it significantly outperforms E when few optimizations are available for decision making.

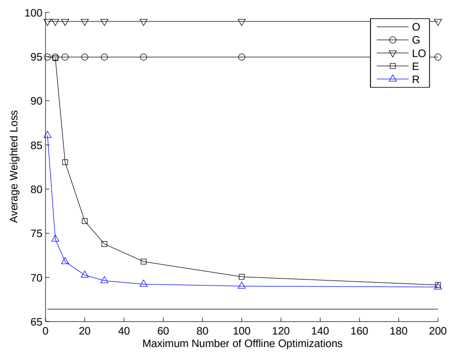

Fig. 1. Regret on Packet Scheduling

6 The k-Median Problem

The Optimization Problem This section studies k-median problems and presents an amortized sub-optimality approximation performing a single local move. A k-median problem receives, as inputs, a set F of facilities, a set of customers S, a function $c_{a,b}$ specifying the connection costs between a and b ($a, b \in F \cup S$), and an integer k. The goal is to find a subset A ($A \subseteq F$) of cardinality k to minimize the objective function

$$W_A(S) = \sum_{s \in S} \min_{a \in A} c_{s,a}.$$

The Sub-Optimality Approximation Problem The computational complexity in the k-median problem consists of choosing which k facilities to open. Once the facilities are open, it suffices to assign the customers to the cheapest facility. As a consequence, the decision are whether to open or close a warehouse and the sub-optimality approximation problem consists of approximating the optimal solution whenever a facility is forced to be open or forced to be closed. This section considers metric k-median problems, i.e., k-median problems where the costs are taken from a metric space. The k-median problem has applications in networking where servers or specialized routers may be modeled as facilities. When a server fails (closing a facility), it is important to choose a replacing router quickly. Similarly, in order to contain failure propagation, it may be important to start a server (opening a facility) at some node. The amortized sub-optimality algorithm presented here handles these two cases very quickly. The Internet is typically not a metric space. However, recent research [10] has shown that it can conveniently be embedded in a metric space.

The Sub-Optimality Approximation Algorithm The sub-optimality approximation algorithm consists of performing the best swap of the considered facility. In other words, when a facility x must be closed (resp. open), the algorithm opens (resp. closes) the warehouse y that increases the cost the least. We now

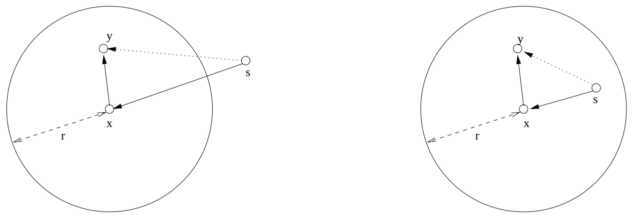

Fig. 2. Closing a Facility: Facility y is Inside the Circle.

show that this local move is a constant approximation by showing a constant approximation for some swaps.

Proposition 1. [Closing] *Let A be an optimal solution to a metric k-median problem and B be an optimal solution when facility x is closed. There exists a facility $y \neq x$ such that $\tilde{B} = A \setminus \{x\} \cup \{y\}$ is a 5-approximation of B.*

Proof. Denote by S_x the set of customers assigned to x in A. Define a circle C centered at x of radius r and define *Inner* as the set of customers in S_x lying inside C, *Outer* as the set of customers in S_x lying outside C and *Other* the remaining customers $S \setminus S_x$. Moreover, choose r such $|Inner| = |Outer|$. We analyze the cost of the solution $\tilde{B}$ that opens the facility y nearest to x and assigns all customers in S_x to y.

First consider the case where y lies in C (see Figure 2). For each customer $s \in Outer$, we have by the triangular inequality

$$c_{s,y} \leq c_{s,x} + c_{x,y} \leq c_{s,x} + r \leq 2c_{s,x}$$

and it follows that $W_{\tilde{B}}(Outer) \leq 2W_A(Outer)$. For each customer $s \in Inner$, we have by the triangular inequality

$$c_{s,y} \leq c_{s,x} + c_{x,y} \leq c_{s,x} + r.$$

Since $r \times |Outer| \leq W_A(Outer)$ and $|Inner| = |Outer|$ it follows that $W_{\tilde{B}}(Inner) \leq W_A(Inner) + W_A(Outer)$. Hence

$$\begin{aligned} W_{\tilde{B}}(S) &= W_{\tilde{B}}(Other) + W_{\tilde{B}}(Inner) + W_{\tilde{B}}(Outer) \\ &\leq W_A(Other) + W_A(Inner) + 3W_A(Outer) \leq 3W_A(S) \leq 3W_B(S). \end{aligned}$$

Consider now the case in which y is outside C and assume that y at a distance $r+d$ of x (Figure 3). Consider a customer $s \in Inner$. By the triangular inequality, $c_{s,y} \leq c_{s,x} + (r+d)$ and thus

$$W_{\tilde{B}}(Inner) \leq W_A(Inner) + (r+d) \times |Inner|$$

Since $r \times |Outer| \leq W_A(Outer)$ and $|Inner| = |Outer|$, it follows that $r \times |Inner| \leq W_A(Outer)$. By definition of y, each *Inner* customer must pay at

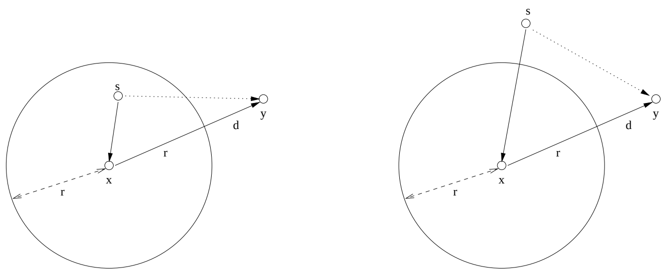

Fig. 3. Closing a Facility: Facility y is Outside the Circle.

least d to get to a facility in the optimal solution in which x is closed. Hence $d \times |Inner| \leq W_B(Inner)$ and $W_{\tilde{B}}(Inner) \leq W_A(Inner) + W_A(Outer) + W_B(Inner)$. Consider now a customer $s \in Outer$. By the triangular inequality, $c_{s,y} \leq c_{s,x} + (r + d)$ giving

$$W_{\tilde{B}}(Outer) \leq W_A(Outer) + (r + d) \times |Outer|.$$

Once again, $r \times |Outer| \leq W_A(Outer)$ and $d \times |Outer| \leq W_B(Inner)$, since $|Inner| = |Outer|$. It follows that

$$W_{\tilde{B}}(Outer) \leq W_A(Outer) + W_A(Outer) + W_B(Inner).$$

Hence $W_{\tilde{B}}(S) \leq W_A(Other) + 3W_A(Outer) + 2W_B(Inner) + W_A(Inner)$ and

$$W_{\tilde{B}}(S) \leq 3W_A(S) + 2W_B(S) \tag{1}$$

and, by optimality of A, it follows that $W_{\tilde{B}}(S) \leq 5W_B(S)$. $\square$

We now consider the case where a facility x is forced to be open. The following proposition indicates that swapping x in the optimal solution provides a constant approximation. The proof adapts some of the proof techniques from online k-median algorithms (Fact 1 in [9]).

Proposition 2. [Opening] *Let A be an optimal solution to a metric k-median problem and let B be the optimal solution where facility x must be open. There exists a facility $y \neq x$ such that $\tilde{B} = A \setminus \{y\} \cup \{x\}$ satisfies $W_{\tilde{B}}(S) \leq 3W_B(S)$.*

Proof. Let $B = B' \cup \{x\}$. Define A' as the set of facilities obtained by considering each facility w in B' and selecting its nearest facility in A and define $\tilde{B}$ as $A' \cup \{x\}$. The proof shows that $W_{\tilde{B}}(S) \leq 3W_B(S)$. Since $|B'| = k - 1$, $|A'| \leq k - 1$ and $\tilde{B}$ can be viewed as swapping x with one of the non-selected facility y of A and the results follows.

To bound the cost of $\tilde{B}$, partition S into S_x and S_o, where S_x are all the customers allocated to facility x in B. The bound on $\tilde{B}$ is obtained by assigning all the customers in S_x are assigned to x. Consider now a customer $s \in S_o$ and let a be its closest facility in A, b is closest facility in B, and let b' be the facility

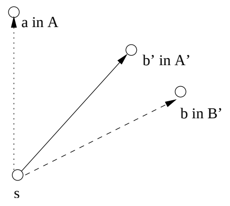

Fig. 4. Opening a Facility: Customer s is Assigned to b'.

in A' nearest to b. The bound on $\tilde{B}$ is obtained by assigning s to b', giving the inequality (see Figure 4)

$$\begin{aligned} c_{s,b'} &\leq c_{s,b} + c_{b,b'} && \text{by the triangular inequality} \\ &\leq c_{s,b} + c_{b,a} && \text{since } c_{b,b'} \leq c_{b,a} \\ &\leq c_{s,b} + c_{b,s} + c_{s,a} && \text{by the triangular inequality} \\ &\leq 2c_{s,b} + c_{s,a} \end{aligned}$$

Summing on all vertices in S_o, we obtain $W_{\tilde{B}}(S_o) \leq 2W_B(S_o) + W_A(S_o)$. By the allocation of S_x, we obtain

$$W_{\tilde{B}}(S) \leq 2W_B(S) + W_A(S) \tag{2}$$

and, by optimality of A, it follows that $W_{\tilde{B}}(S) \leq 3W_B(S)$. □

Theorem 3. Amortized Sub-Optimality for the Metric k-Medians. *The metric k-median problem has an amortized sub-optimality approximation algorithm that runs in time $O(|S| \log |F|)$.*

Proof. Propositions 1 and 2 show that a single swap to open or close the considered facility x produces a 5-approximation of the optimal solution in the worst case. The best swap to open or close x is thus a constant approximation and it can be computed in $O(|S| \log |F|)$ using the data structures of [14]. □

The result continues to hold even when only an α-approximation of the k-median is available to the sub-optimality approximation algorithm. Indeed, the propositions only rely on the optimality of A in the last steps of their proofs (after Equations 1 and 2) and the same proof technique as in Theorem 1 can be used. Such α-approximations can be obtained by local search for instance [1]. The algorithm is amortized since it consists of a single swap and it is strongly amortized if one assumes that a local search performs at least $|F|$ swaps.

Theorem 4. Amortized Sub-Optimality for the Metric k-Medians. *The metric k-median problem has an amortized sub-optimality (α, β)-approximation algorithm that runs in time $O(|S| \log |F|)$.*

The k-median is closely related to (uncapacitated and capacitated) facility location problems. The results described here apply directly to uncapacitated facility location with uniform fixed costs. It would be interesting to study whether they also apply when the costs are not uniform.

7 Hardness/Easiness of Sub-Optimality Approximations

In general, the availability of an optimal solution σ^* is a significant advantage for sub-optimality approximation algorithms. In fact, some difficult problems become trivial when σ^* is available. Consider, for instance, the graph-coloring problem which consists of finding the chromatic number of a graph. No constant factor approximation for graph coloring likely exists (unless $P = NP$) [13], yet the suboptimality problem can be solved exactly in polynomial time.

Lemma 1. *The sub-optimality problem can be solved exactly in polynomial time for graph coloring.*

Proof. Let O be a graph-coloring problem with optimal solution σ^* and let c_x be the color of x in σ^*. The suboptimality problem $O[x \leftarrow c]$ can be solved optimally by swapping the colors c_x and c in σ^*. $\square$

Some polynomial algorithms also enjoy simple sub-optimality approximation algorithms. Consider the problem of finding the shortest path from a source to a sink and the sub-optimality problem that consists of studying the choice of various successors to the source. This problem arises in online stochastic planning and can be solved (optimally) by two shortest paths: one from the source to the sink and one from the sink to the source (reverting all arcs).

One may thus think that suboptimality approximations are inherently simpler than the original problems. This is not case unfortunately: there are problems for which suboptimality approximation is as hard as the problem itself. One such problem is maximum satisfiability (MAX-SAT): given a CNF formula ϕ, find a truth assignment that satisfies the maximum number of clauses.

Lemma 2. *Suboptimality of MAX-SAT is as hard as MAX-SAT.*

Proof. Assume that there exists a polynomial-time (exact or approximate) sub-optimality algorithm $\mathcal{A}$ for MAX-SAT. We can construct an algorithm $\mathcal{A}'$ that solves MAX-SAT (exactly or approximately) as follows. Given a CNF folmula $\phi = (C_1 \wedge \ldots \wedge C_k)$ where each C_i is a clause, $\mathcal{A}'$ constructs a formula $\phi = (C'_1 \wedge \ldots \wedge C'_k)$, where $C'_i = (C_i \vee x)$ $(1 \leq i \leq k)$ and x is a brand new variable. Obviously, any truth assugnment in which x is *true* is an optimal solution. $\mathcal{A}'$ now calls A on the formula ϕ', variable x, and any such optimal assignment. Since $\mathcal{A}'$ returns the optimal solution for the case in which x is assigned *false*, $\mathcal{A}'$ returns an optimal solution for the original formula ϕ. $\square$

The above proof uses the following scheme: It transforms the input by a small change into an instance for which computing an optimal solution is trivial. Then, the modified input with its optimum is given to a suboptimality algorithm, which faces the original problem. The method can also be applied to minimization problems. For example, an instance of minimum hitting set can be transformed by selecting an item e which does not appear in any set, and adding it to each of the sets. Now, the set $\{e\}$ is an optimal solution. A suboptimality algorithm can then be asked to compute (or approximate) the optimum when e is forbidden from belonging to the hitting set. Clearly, the solution solves (or approximates) the original minimum hitting set instance.

8 Conclusion

This paper introduced the sub-optimality approximation problem and the concept of amortized sub-optimality approximation algorithms, and discussed its applications to online stochastic optimization, solution repair, and approximate filtering of optimization constraints. The paper also presented amortized sub-optimality (α, β)-approximations for metric TSP, packet scheduling, and metric k-median problems, as well as some hardness (and easiness) results on the sub-optimality approximation problems. There are many avenues of further research. Paramount among them is the need to understand the nature of problems that admit (amortized) sub-optimality approximations.

Acknowledgments Special thanks to Claire Kenyon and Neal Young for suggesting the proof of Proposition 2.

References

1. Arya, V. and Garg, N. and Khandekar, R. and Pandit, V. Local search heuristics for k-median and facility location problems. In *Proceedings of the 33rd ACM Symposium on the Theory of Computing (STOC 2001)*, 2001.
2. R. Bent and P. Van Hentenryck. Online Stochastic and Robust Optimization. In *ASIAN'04*, Chiang Mai University, Thailand, December 2004.
3. R. Bent and P. Van Hentenryck. Regrets Only. Online Stochastic Optimization under Time Constraints. In *AAAI'04*, San Jose, CA, July 2004.
4. R. Bent and P. Van Hentenryck. Scenario Based Planning for Partially Dynamic Vehicle Routing Problems with Stochastic Customers. *Operations Research*, 52(6), 2004.
5. R. Bent and P. Van Hentenryck. The Value of Consensus in Online Stochastic Scheduling. In *ICAPS 2004*, Whistler, British Columbia, Canada, 2004.
6. R. Bent and P. Van Hentenryck. Online Stochastic Optimization without Distributions . In *ICAPS 2005*, Monterey, CA, 2005.
7. R. Bent and P. Van Hentenryck and Eli Ufval. Online Stochastic Optimization Under Time Constraints. Working Paper, 2005.
8. H. Chang, R. Givan, and E. Chong. On-line Scheduling Via Sampling. *Artificial Intelligence Planning and Scheduling (AIPS'00)*, pages 62–71, 2000.
9. Chrobak, M. and Kenyon, C. and Young, N. . The reverse greedy algorithm for the metric k-median problem. In *The Eleventh International Computing and Combinatorics Conference (Cocoon 2005)*, Kunming, Yunnan, 2005.
10. F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A Decentralized Network Coordinate System. In *SIGCOMM 2004*, August 2004.
11. A. Fiat and G. Woeginger. *Online Algorithms: The State of the Art.* 1998.
12. F. Focacci, A. Lodi, and M. Milano. Optimization-Oriented Global Constraints. *Constraints*, 7(3-4):351–365, 2002.
13. C. Lund and M. Yannakakis. On the Hardness of Approximating Minimization Problems. In *STOC-93*, New York, NY, 1993.
14. L. Michel and P. Van Hentenryck. A Simple Tabu Search for Warehouse Location. *European Journal of Operational Research*, 157(3):576–591, 2004.
15. P. Van Hentenryck, P. Flener, J. Pearson, and M. Ågren. Tractable symmetry breaking for csps with interchangeable values. *IJCAI'03.*