Sur les aspects théoriques et pratiques des compromis dans les problèmes d'allocation des ressources

Abhinav Srivastav

Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministérial : 25 mai 2016

Présentée par

Abhinav Srivastav

Thèse dirigée par **Oded Maler, Directeur de recherche, CNRS-Verimag**
et codirigée par **Denis Trystram, Professeur, Grenoble INP**

préparée au sein **Laboratoire Verimag**
et de **L' École Doctorale Mathématiques, Science et technologies de l'information, Informatique**

Sur les aspects théoriques et pratiques des compromis dans les problèmes d'allocation des ressources

Thèse soutenue publiquement le **16 Février 2017**,
devant le jury composé de :

Monsieur Oded Maler
Directeur de Recherche, CNRS-Verimag, Directeur de thèse
Monsieur Denis Trystram
Professeur, Grenoble INP, Grenoble, Co-Directeur de thèse
Madame Marie-Christine Rousset
Professeur, Université Grenoble Alpes, Présidente
Monsieur Adi Rosén
Directeur de Recherche, CNRS et Université Paris Diderot, Rapporteur
Monsieur Monaldo Mastrolilli
Professeur, SUPSI-Suisse, Rapporteur
Monsieur Seffi Naor
Professeur, Institut de Technologie D'Israël, Examinateur
Monsieur Daniel Vanderpooten
Professeur, Université Paris-Dauphine, Examinateur
Monsieur Lothar Thiele
Professeur, ETH-Zürich - Suisse, Examinateur

Résumé

Cette thèse est divisée en deux parties. La première partie de cette thèse porte sur l'étude d'approches heuristiques pour approximer des fronts de Pareto. Nous proposons un nouvel algorithme de recherche locale pour résoudre des problèmes d'optimisation combinatoire. Cette technique est intégrée dans un modèle opérationnel générique où l'algorithme évolue vers de nouvelles solutions formées en combinant des solutions trouvées dans les étapes précédentes. Cette méthode améliore les algorithmes de recherche locale existants pour résoudre le problème d'assignation quadratique bi- et tri-objectifs.

La seconde partie se focalise sur les algorithmes d'ordonnancement dans un contexte non-préemptif. Plus précisément, nous étudions le problème de la minimisation du stretch maximum sur une seule machine pour une exécution online. Nous présentons des résultats positifs et négatifs, puis nous donnons une solution optimale semi-online. Nous étudions ensuite le problème de minimisation du stretch sur une seule machine dans le modèle récent de la réjection. Nous montrons qu'il existe un rapport d'approximation en $O(1)$ pour minimiser le stretch moyen. Nous montrons également qu'il existe un résultat identique pour la minimisation du flot moyen sur une machine. Enfin, nous étudions le problème de la minimisation du somme des flots pondérés dans un contexte online.

Abstract

This thesis is divided into two parts. The first part of the thesis deals with the study of heuristic based approaches for approximating Pareto fronts. We propose a new Double Archive Pareto local search algorithm for solving multi-objective combinatorial optimization problems. We embed our technique into a genetic framework where our algorithm restarts with the set of new solutions formed by the recombination and the mutation of solutions found in the previous run. This method improves upon the existing Pareto local search algorithm for multiple instances of bi-objective and tri-objective quadratic assignment problem.

In the second part of the thesis, we focus on non-preemptive scheduling algorithms. Here, we study the online problem of minimizing maximum stretch on a single machine. We present both positive and negative theoretical results. Then, we provide an optimally competitive semi-online algorithm. Furthermore, we study the problem of minimizing average stretch on a single machine in a recently proposed rejection model. We show that there exists a polynomial time $O(1)$-approximation algorithm for minimizing the average stretch and average flow time on a single machine. Essentially, our algorithm converts a preemptive schedule into a non-preemptive schedule by rejecting a small fraction of jobs such that their total weights are bounded. Lastly, we study the weighted average flow time minimization problem in online settings. We present a mathematical programming based framework that unifies multiple resource augmentations. Then, we developed a scheduling algorithm based on the concept of duality. We showed that our algorithm is $O(1)$-competitive for solving the weighted average flow time problem on a set of unrelated machines. Furthermore, we showed that our algorithm when applied to the problem of minimizing ℓ_k norms of weighted flow problem on a set of unrelated machines, is $O(k)$-competitive.

CONTENTS

LIST OF FIGURES

LIST OF ALGORITHMS

1

INTRODUCTION

In many real-life problems, a decision maker searches for an *optimal* choice amongst a set of alternatives. For example, a driver interested in choosing among the set of alternative routes or a buyer interested in choosing among different brands. Quite often, each solution can be evaluated according to various cost or utility functions. Hence these searches are based on optimizing several, possibly conflicting, objective functions. For instance, each route may be assessed with respect to fuel consumption, its travel time and its touristic value while the value of a product such as a smartphone or a laptop can be estimated by its price, quality, and size. Consequently, a solution which performs better according to one objective function may have the worse performance with respect to another. This gives rise to a set of *incomparable* solutions. Without any prior subjective preference about the decision maker's choice, all such solutions are considered equally good. Problems like these frequently arise in diverse areas like telecommunication, engineering, computer science, operation research, medicine, finance, physics, etc.

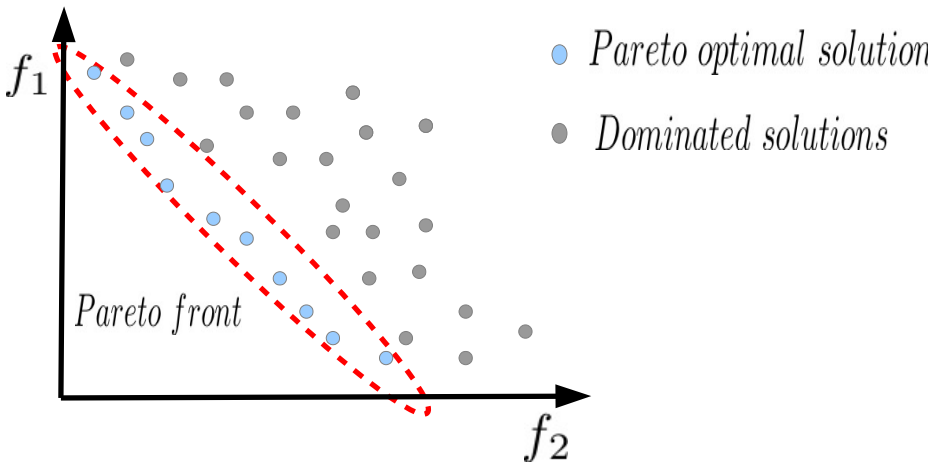


Figure 1.1. – An example showing trade-offs in the cost space for a bi-objective minimization problem

In multi-objective optimization, there is typically not a single optimal solution but rather a set of incomparable optimal solutions where no solution can be improved with respect to one objective function without worsening its performance with respect to some

other objective function(s). Such solutions are called Pareto optimal solutions while a set consisting of all such solutions is known as the *Pareto front*. Figure 1.1 illustrates the notion of Pareto front for a minimization problem with two objective functions f_1 and f_2. Thus, the notion of "solving a multi-objective optimization problem" refers to finding the Pareto front.

Unfortunately, many real-world problems are combinatorial in nature and thus, finding the sets of optimal solutions are generally hard, *i.e.* such problems cannot be solved within a reasonable amount of time limit unless $P = NP$. For hard single-objective problems, the most common approach is to relax the notion of optimality so as to find a good enough solution within a reasonable amount of time. But for multi-objective problems, the meaning of a relaxed notion of optimality is not self-evident. For instance, does it refer to finding a single solution or a set of solutions approximating the entire Pareto front? This problem becomes pronounced when the number of optimal solutions in the Pareto front is large. Many of the classical multi-objective combinatorial optimization problems such as knapsack [Hug07], shortest path [Han80], s-t cut [Hug07], spanning tree [HR94], TSP [EP09] to name a few, are known to have exponential (in the size of the input) number of optimal solutions, even for the bi-objective case. In such cases, the focus of an algorithm designer shifts towards finding the approximate Pareto optimal solutions. By far the most common approach is to approximate the Pareto front by a set of mutually incomparable solutions. Numerous methods, both practical and theoretical have been proposed in the literature which can efficiently produce such a set of solutions.

Local search algorithms are one such type of heuristics which have been applied to a variety of single-objective problems. They have also been extended to multi-objective case under the name of Pareto local search (PLS) techniques. Essentially, these techniques are iterative improvement algorithms that start with a random feasible solution and repeatedly apply local changes with the goal of improving its quality. Figure 1.2 illustrates one such iteration where the solution s has four neighbors: s_1, s_2, s_3 and s_4 which are represented in the cost space. Note that s_1 dominates s_2 with respect to both objective functions. , s_3 dominates s_4. Thus, PLS discards s_2 and s_4 and selects either s_1 and s_3 for the next step in the local search. These techniques are known to produce high-quality solutions. The quality of these solutions can be improved further when they are coupled with other heuristics. For instance, some of the best performing algorithms for solving multi-objective travelling salesman problems have PLS as a crucial component. Various other problems such as bi-objective permutation flow-shop problem and multi-dimensional knapsack have been efficiently solved using the PLS techniques.

Another widely used heuristic is genetic algorithms (GA), which are population-based metaheuristic optimization algorithms inspired by a simplified view of the evolution. GAs are search algorithms where solutions are represented as bit strings such that each bit represents a *gene*. Once such a string is created, it is assigned a fitness function

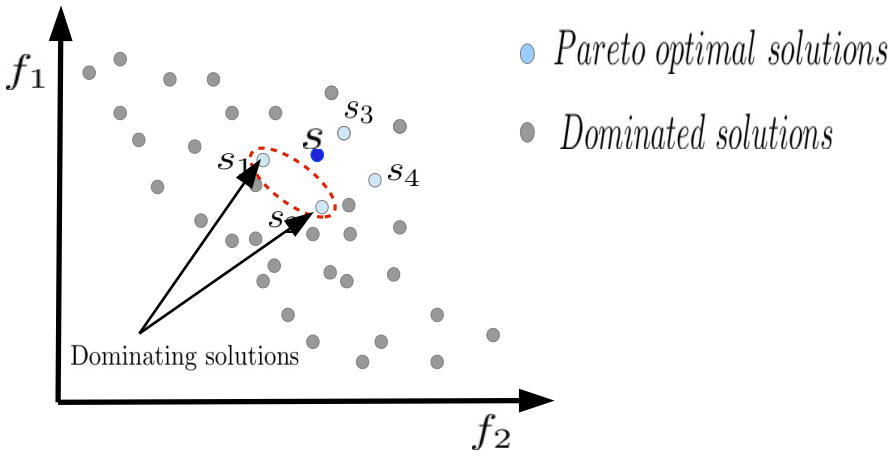


Figure 1.2. – An example showing concept of dominance in local search for bi-objective problem ($\min f_1, \min f_2$).
50100054794869

according to solution's objective function value. The fitness function determines the chances of survival of the candidate solutions. In each generation, the reproduction step builds the mating pool with promising candidate solutions. Next, the crossover operator is applied to strings in the mating pool. For this, two individuals are randomly picked from the pool and some part of their strings are exchanged to generate new individuals. These offsprings are then mutated with some probability to provide a better diversity in the population. The above procedure is repeated until some stopping criteria are met. Since GAs maintain a set of individuals as a population, they have been easily extended to multi-objective optimization problems. Quite often in practice, multi-objective genetic algorithms are coupled with PLS methods. This helps in improving the running time of genetic algorithms and provides a better set of incomparable solutions.

Although these heuristics in practice may provide good sets of solutions, they generally do not provide any theoretical guarantee on the quality of the solutions with respect to the optimal Pareto front. However, such guarantees are often required to understand the underlying complexity of the problem. Moreover, they provide us with insights on the types of instances for which a particular heuristic will perform well.

A widely accepted notion of a theoretical guarantee of an algorithm for single-objective problem is its *approximation ratio*. For a minimization (maximization) problem, the approximation ratio of an algorithm is the largest (smallest) ratio between the result obtained by the algorithm and the optimal solution. For multi-objective problems, the term *approximation ratio* refers to a set of ratios, where each ratio corresponds to the worst case performance of the algorithm with respect to different objective functions. For instance, consider a case of bi-objective problem where the

term approximation ratio refers to a set (ρ_1, ρ_2), where ρ_1 is the factor by which the algorithm performs with respect to f_1 and ρ_2 is the factor by which the algorithm performs with respect to f_2. Figure 1.3 represents the notion of (ρ_1, ρ_2)-approximation for a bi-objective problem. Papadimitriou et al. [PY00] proposed a generalized notion of ρ-approximate Pareto curve which consists of a set of incomparable solutions such that each optimal Pareto solution is within the factor of ρ from some solution in the set.

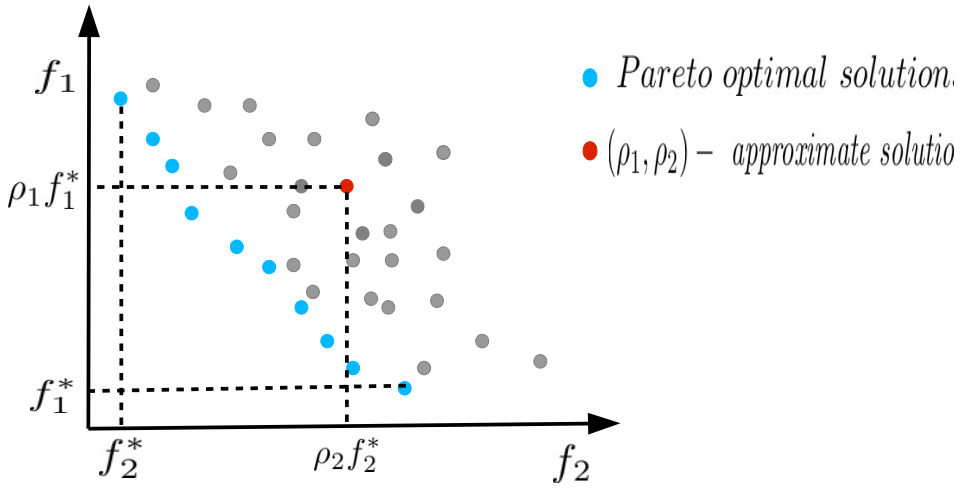


Figure 1.3. – An example showing (ρ_1, ρ_2)-approximation solution for bi-objective problem. Note that the solution f_1^* and f_2^* are the best values with respect to f_1 and f_2 objective function.

For many multi-objective combinatorial problems, such approximate schemes have been developed. In fact, several works on the theoretical study of multi-objective optimization problems focus either on finding a single solution which can provide a vector of approximation ratios or on finding a set of incomparable points which can approximate the entire Pareto front within some constant bound. Our focus is on the study of combinatorial problems arising in the context of scheduling. Scheduling is a combinatorial optimization problem that plays an important role in different disciplines. It is generally concerned with the allocation of limited resources to the tasks with the aim of optimizing one or more performance measures. Such problems arise in a variety of domains such as vehicle routing, transportation, manufacturing, etc. Our motivation comes from the study of the client-server model where tasks are modelled as jobs and resources are modelled as machines. Then, the general aim is to assign the sequence in which jobs are processed on a set of machines. The typical example of such systems includes operating systems, high-performance platforms, web-servers, database servers, etc.

One of the most common measure for the quality of service delivered to a job is the amount of time it spends in a system. Mathematically, this can be modelled as the *flow time* of the job, which is defined as the difference between the completion time and the arrival time of the job. Historically, such problems have been studied in the preemptive settings where jobs are allowed to be interrupted during the execution and resumed later for the completion. In general, preemptive flow time minimization problems are known to have strong lower bounds [KTW95, LSV08, BMR03, BCM98]. Therefore, these problems are often explored under a relaxed notion of performance known as *resource augmentation* [KP00].

Resource augmentation refers to a method of algorithm analysis, where the algorithm is allowed more resources than the optimal algorithm. In a client-server model, extra resources can often be thought as having faster machines or having more machines in comparison to the optimal algorithm. Then, performance guarantees are stated with respect to a fixed resource augmentation *i.e.* the performance of the algorithm with extra resources is compared to the optimal performance without any extra resources. In the last decade or so, several intuitive explanations have been proposed in the support of using resource augmentation for the analysis of algorithms. For instance, Kalyanasundaram *et al.* [KP00] first proposed this model for online problems where they explained the usage of extra resources as the cost of buying clairvoyance. They claimed that the comparison with a weaker optimal suggests a practical way to combat the lack of knowledge of the future. Another interesting explanation is based on the "sensitivity" analysis of hard problems [BCK$^+$07]. Here, the existence of a constant approximation algorithm under resource augmentation model is seen as the evidence that hard instances are rare in practice, *i.e.* if hard instances are slightly perturbed, then the lower bound for the problem under the consideration falls apart. This also suggests why resource augmentations are able to identify algorithms that perform well in practice. In multi-objective perspective, we can view the resource augmentation as a trade-off between the usage of extra resources and the performance of the algorithm.

Another important criterion for the client-server model is the uninterrupted (non-preemptive) execution of a job. This is particularly important because preemptive scheduling usually have a huge management overhead which can affect the overall performance of the system. However, in spite of the importance, most of the existing work is focused on the preemptive setting. This is mainly due to the fact that non-preemptive problems have strong lower bounds even with speed and machine augmentation. In contrast, we focus on non-preemptive problems in the recently proposed model of rejection where the algorithm is allowed to reject a small fraction of the instance of the problem [CDK15, CDGK15]. For example in the case of scheduling, an algorithm is allowed to reject a small fraction of the total number of jobs. Then, the performance of the algorithm on non-rejected instance is compared to the optimal value for the entire instance. This is a very natural setting in the client-server model, where the server is allowed to reject a small fraction of requests from clients to have a good overall

performance. Moreover, the rejection model gives us insight on the types of instances for which a particular scheduling algorithm will have a good performance.

1.1 Our Contributions

Our work is divided into two separate parts. The first part deals with the study of heuristic based approaches for approximation Pareto fronts. Specifically, we study Pareto local search algorithm (PLS) and Queued Pareto local search algorithm (QPLS). We show some major drawbacks in the design of these algorithms. Then, we present a new Double Archive Pareto local search algorithm (DAPLS) which takes these drawbacks into consideration. Essentially, our effort is focused on designing a new method to maintain the set of candidate solutions whose neighbours are unexplored during the iteration. Moreover we show that like PLS and QPLS, our algorithm also terminates in a local optimum. This is one of the basic design requirement for any local search algorithm. To escape local optima, we embed our technique into a genetic framework where DAPLS restarts with the set of new solutions formed by the recombination and the mutation of solutions found in the previous run of DAPLS. Empirically, we show that our genetic DAPLS outperforms genetic PLS and genetic QPLS on bi-objective and tri-objective instances of the quadratic assignment problems [MS16].

In the second part of the thesis, we focus on non-preemptive scheduling algorithms. Unlike the above heuristic approach, our algorithms produce a single solution whose performance guarantee is bounded. First, we study the problem of minimizing maximum stretch on a single machine. Stretch is a variant of the weighted flow time problem where the weight of each job is inversely proportional to its processing time. This is a natural performance measure where the importance is given to "fair" scheduling of jobs *i.e.* for instances where the job preferences are not made explicit, one can assume that jobs with a large processing time should wait more than jobs with a smaller processing time. We present both positive and negative theoretical results. We show the existence of a strong lower bound on the problem of minimizing maximum stretch on a single machine. Then we provide a *semi-online* algorithm which achieves this lower bound [DSST16].

In general, it has been shown that the offline problem of minimizing stretch on a single machine is NP-hard. Therefore, we study this problem under the notion of resource augmentation. Specifically, we show that there exists an $O(1)$-approximation ratio for the problem of minimizing average stretch such that the algorithm is allowed to reject at most an ϵ-fraction of the total weights of the jobs. Essentially, our algorithm converts a preemptive schedule into a non-preemptive schedule by rejecting a small fraction of jobs such that their total weights are bounded. Using a similar idea, we also show that there exists an $O(1)$-approximation ratio for the problem of minimizing average flow time on a single machine. Moreover, our algorithm has polynomial running time. This

answers an open question on the existence of a polynomial time $O(1)$-approximation algorithm [ILMT15]. Our algorithm can be seen as a trade-off between the optimal performance and the number of rejected jobs [LST16].

Lastly, we study the weighted average flow time minimization problem in online settings. We present a mathematical programming based framework that unifies multiple resource augmentations. Then, we developed a scheduling algorithm based on the concept of duality. We showed that there exists an $O(1)$-competitive algorithm for solving the weighted average flow time problem on a set of unrelated machines where the algorithm is allowed to reject at most ϵ-fraction of the total weights of jobs. Furthermore, we showed that this idea can be extended to the problem of minimizing weighted ℓ_k norms of flow time on a set of unrelated machines [LNST16].

2

MULTI-OBJECTIVE OPTIMIZATION

The field of *multi-objective optimization* deals with the problem of finding a set of *optimal* trade-offs among a set of solutions. Such problems are characterized by a set of objective functions defined over a solution space. Many problems with several, possibly conflicting, objectives can be modelled in these terms. Depending on the problem, the objectives can either be defined explicitly for each criteria or they may be formulated as a set of constraints. These problems are generally addressed by providing the set of all non-dominated solutions. Specifically a solution s is said to *dominate* solution s' only if it is at least as good as s' with respect to all objective and is strictly better with respect to at least one of the objective function. The set consisting of all such non-dominated optimal solutions is known as the *Pareto front*.

2.1 Formalization

Mathematically, a multi-objective optimization problem can be viewed as a tuple $\varphi = (\mathcal{S}, \mathcal{C}, \mathcal{F})$ where $\mathcal{S}$ is the set of feasible solutions, $\mathcal{C} \subseteq \mathbb{R}^d$ is the cost (objective) space and $\mathcal{F} : \mathcal{S} \rightarrow \mathcal{C}$ is a set of d objective functions, i.e. $\mathcal{F} = \{f_1, f_2, ..., f_d\}$. Figure 2.1 illustrates the relationship between a solution space and its corresponding cost space for a bi-objective problem. Without the loss of generality, we assume that all the objective functions in $\mathcal{F}$ are to be minimized. The duality principle, defined in the context of optimization, suggests that we can convert a maximization problem into a minimization problem [BV04]. This has made the task of handling maximization and mixed maximization/minimization problem much easier. Since our aim is to find a set with minimal costs in $\mathcal{C} \subseteq \mathbb{R}^d$, we provide a partial order on elements of $\mathcal{C}$.

Definition 2.1. *For two solution $s, s' \in \mathcal{S}$, we say that*

1. *s weakly dominates s', denoted as $\mathcal{F}(s) \preceq \mathcal{F}(s')$, iff $\forall i \in \{1, ..., d\} : f_i(s) \leq f_i(s')$.*
2. *s strictly dominates s', denoted as $\mathcal{F}(s) \prec \mathcal{F}(s')$, iff $\forall i \in \{1, ..., d\} : f_i(s) \leq f_i(s')$ and $\exists j \in \{1, ..., d\} : f_j(s) < f_j(s')$.*

Using Definition 2.1, we now define the notion of *incomparable* solutions.

Definition 2.2. *Two solution $s, s' \in \mathcal{S}$ are said to be incomparable iff $\mathcal{F}(s) \not\prec \mathcal{F}(s')$ and $\mathcal{F}(s') \not\prec \mathcal{F}(s)$. We denote this fact by $\mathcal{F}(s) || \mathcal{F}(s')$.*

It follows that if s and s' are incomparable, then there exists $i, j \in \{1, ..., d\}$ such that $i \neq j$ and $f_i(s) < f_i(s')$ and $f_j(s') < f_j(s')$. The notion of domination (illustrated in

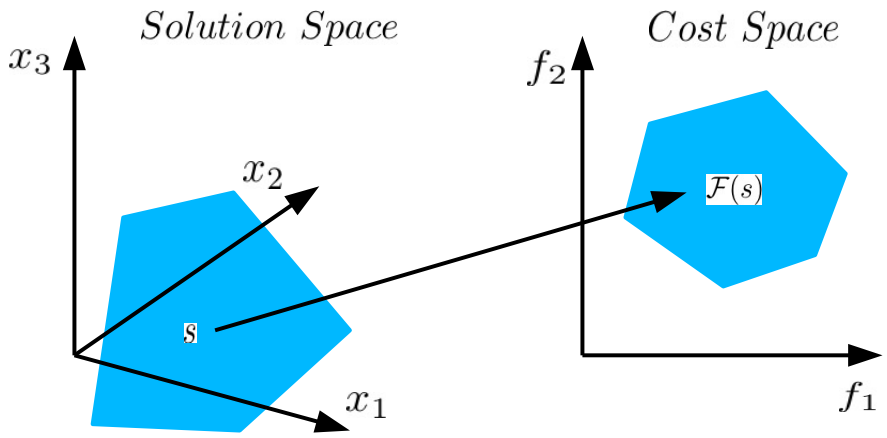


Figure 2.1. – The solution space and the corresponding cost space

Figure 2.2) and incomparability are central to the field of multi-objective optimization. A solution s which is strictly dominated by some s', is generally discarded, as s' has better objective values in comparison to s in all dimensions. A solution is considered optimal when its objective values are not dominated by any other solution in $\mathcal{S}$.

Definition 2.3. *A Pareto front is a set $P \subseteq \mathcal{S}$ consisting of all Pareto optimal solutions where a solution $s \in \mathcal{S}$ is said to be Pareto Optimal iff $\forall s' \in \mathcal{S} : \mathcal{F}(s') \not\prec \mathcal{F}(s)$.*

2.2 Bounds on the Pareto front

In this section, we define *ideal* and *nadir* points as the lower and the upper bounds on the cost of Pareto optimal solutions. Informally, these points indicate the range of the values which non-dominated solution can attain in $\mathcal{C}$.

2.2.1 Ideal point

For each of the d-objectives, there exists an optimal solution. A cost vector consisting of these individual optimal solutions is known as the ideal point.

Definition 2.4. *A point $z^* \in \mathcal{C}$ is called the ideal point of the multi-objective optimization problem iff $\forall i \in \{1, .., d\}, z_i^* = \min_{\mathcal{S}} f_i$.*

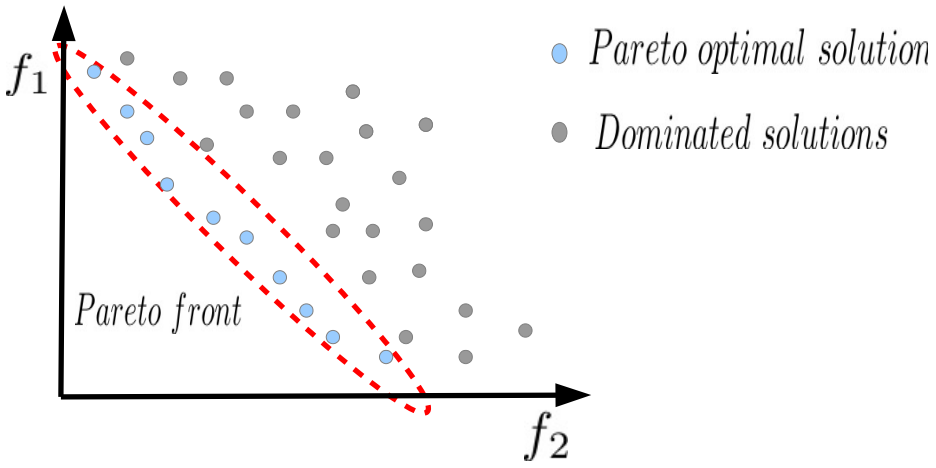


Figure 2.2. – An example showing trade-offs in cost space for a bi-objective minimization problem

Although the ideal point is typically infeasible (for conflicting objectives), it can serve as a reference solution. Furthermore, it can be used by many algorithms to normalize objective values into a common range.

2.2.1 Nadir points

Nadir points represent the upper bound on the cost of non-dominated solutions. Note that the nadir point is not a vector of costs with worst objective values for each objective.

Definition 2.5. *A point in $z^{**} \in \mathcal{C}$ is said to be the nadir point of the multi-objective optimization problem iff $\forall i \in \{1, ..., d\}, z_i^{**} = \max_P f_i$, where P is the Pareto front of the problem under consideration.*

Note that the nadir point may represent an existent or non-existent solution, depending upon the convexity of the Pareto optimal set. Although the nadir point is easy to compute, the solution corresponding to the nadir point is much harder to find in practice. However, for some problems like linear multi-objective optimization problems, the nadir point can be derived from the ideal point using the *payoff table* mentioned in Miettinen [Mie99]. Figure 2.3 show the ideal and the nadir points for a bi-objective minimization problem.

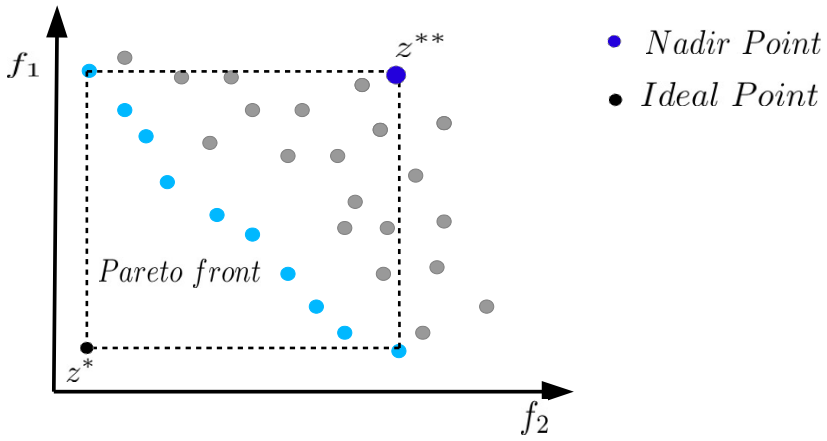


Figure 2.3. – The notion of ideal and nadir points in a bi-objective minimization

2.3 Solving Multi-objective optimization problem

Unfortunately, many of the discrete multi-objective problems of interest are NP-hard even in the single-objective case. Furthermore, in a multi-objective scenario there exists multiple Pareto solutions and if their number is huge, enumerating all of them is impossible within a reasonable amount of time. Such problems are known as *intractable*.

Definition 2.6. *A multi-objective optimization problem is said to be* intractable *if there are instances for which the number of solutions in the Pareto set is super-polynomial in the size of the input.*

In the case of intractable or NP-hard problems, one has to be content with an approximation of the Pareto front. Below we define the notion of approximate Pareto front.

Definition 2.7. *An approximation of a Pareto front in a feasible cost space $\mathcal{S}$ is a set $A \subseteq \mathcal{S}$ such that $\forall s, s' \in A : \mathcal{F}(s)||(F)s'$. We denote the set of all such approximations as $\mathcal{A}$.*

Numerous multi-objective optimizers have been proposed in the literature which can efficiently generate approximations of the Pareto front. We postpone their study in detail until the next chapter. Now we turn our attention on the quality assessment of the approximate Pareto front.

2.3.1 Quality of Approximate Pareto front

One of the key questions pertaining multi-objective optimization is how to evaluate the performance of several different multi-objective optimizers and compare the quality of their respective outputs. Note that the definition of the approximate Pareto front does not comprise of any notion of quality. However, we can make statements about the quality of approximation sets in comparison to other approximation sets using dominance relations.

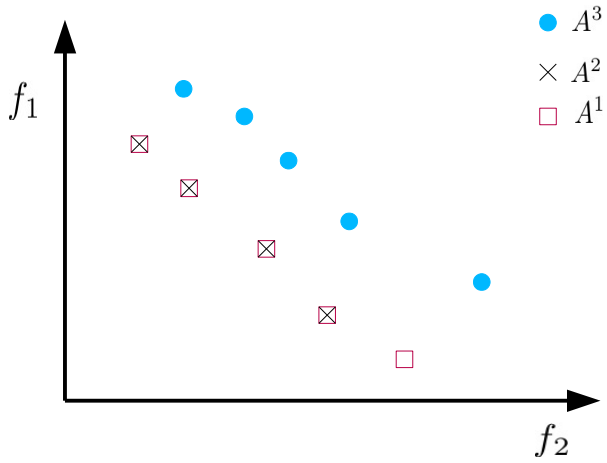


Figure 2.4. – Outcomes of three hypothetical algorithms for a bi-objective minimization problem. The corresponding approximation sets are denoted as $A^1, A^2,$ and A^3

Consider a scenario (borrowed from [ZTL+03]) where outcomes of three different hypothetical optimizers are depicted in Fig 2.4. Based on the Pareto dominance relationship, A^1 and A^2 dominate A^3. Moreover, the A^1 provide better quality of approximation as it contains all solutions in A^2 and one more solution not in A^2. Although such a simple comparison can check whether an optimizer is better than another with respect to dominance relation, it is difficult to make precise quantitative comparison *i.e.* how much is one optimizer is better than another ? or if no optimizer can be said better than the other (mutually incomparable solutions) then in what aspect certain optimizer is better than others ?

In general, one expects that solutions returned should be close to Pareto optimal solutions. But this alone does not guarantee a good quality of solutions as all solutions returned by an optimizer may be concentrated in one region of the cost space. Therefore, there is further need of finding the solutions that are spread evenly in the cost space (diversity). This will provide a good understanding of trade-offs available to a decision

maker. As pointed out above, the dominance criteria alone cannot determine the quality of the approximate solution. We need indicators that can provide some kind of total order on the outcome of different optimizers. Below we define the notion of quality indicator for an approximation of the Pareto front.

Definition 2.8. *A quality indicator is a function* $\mathcal{I} : \mathcal{A} \rightarrow \mathbb{R}$ *which assigns real numbers to approximate Pareto fronts.*

Without the loss of generality, we assume that as the value of $\mathcal{I}$ increases, the quality of approximation improves. Several quality indicators like inverted generalized distance [CC05], R-metrics [HJ98], epsilon indicator [ZTL$^+$03], etc have been introduced to compare outcomes of different optimizers. Below we present two such quality indicators which are used later in this thesis.

Hypervolume Indicator

The most common indicator used in the literature is hypervolume or L-measure which is popularized by the work of Zizler et.al [ZTL$^+$03].

Definition 2.9. *Let* $r \in \mathcal{C}$ *be a reference point and* $A \in \mathcal{A}$ *be an approximation set, then we define dominated region (denoted* $B^+(A,r)$*) as*

$$B^+(A,r) = \{c \in \mathcal{C}, \exists s \in A : \mathcal{F}(s) \preceq c \preceq r\}$$

.

The shaded area in Figure 2.5 represents the region dominated by approximating solutions (in dark blue color) with respect to a reference point (in black color).

Definition 2.10. *Given* r *and* A*, the hypervolume indicator, denoted by* $I_H(A,r)$*, is the volume of the region* $B^+(A,r)$.

This measure is Pareto compliance, in the sense that if $A_1, A_2 \in \mathcal{A}$ and $A_1 \preceq A_2$, then $I_H(A_1,r) > I_H(A_2,r)$. Unfortunately, this measure is biased towards the convex inner portions of the objective space. Later Zitzler et al. [ZBT07] proposed a weighted variant of the hypervolume measure.

The above presented performance measure is used for deterministic algorithms *i.e* algorithms which produce a single approximate Pareto front. Many multi-objective optimizers, including evolutionary algorithms, are stochastic in nature. Therefore, each run of these algorithms can produce different outcomes. In order to compare algorithms, one needs to perform a statistical analysis on the quality of each output with an indicator. For example, consider a scenario where two multi-objective optimizers P and Q are compared using the hypervolume quality indicator. Both P and Q are run multiple times on the same instance of the problem. The final quality indicator used in comparing P and Q, is the mean and/or variance of the hypervolume.

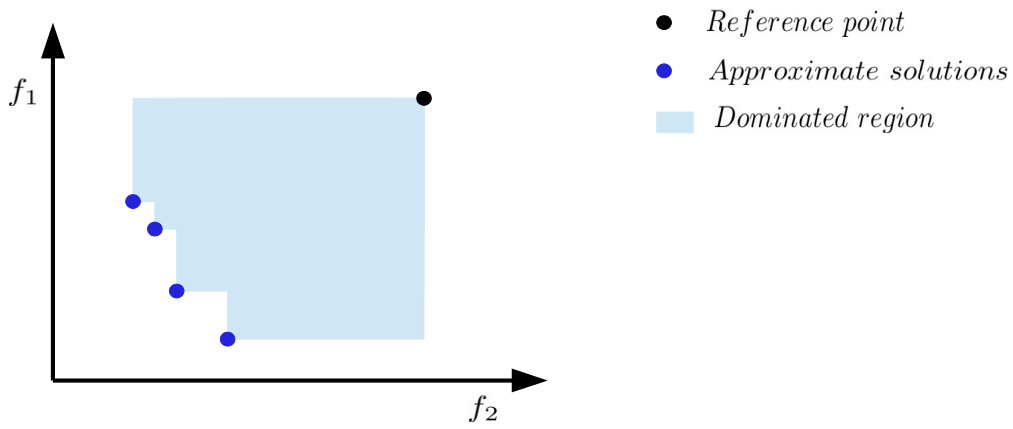


Figure 2.5. – The shaded region of the cost space corresponds to the hypervolume indicator

Empirical Attainment Function

In a stochastic multi-objective optimizer, each outcome can be seen as the realization of a random non-dominated point set. Therefore, they can be studied under the distribution of a random set. In particular, the attainment function is defined as the probability of an approximate set A attaining an arbitrary point $c \in \mathcal{C}$, *i.e.*, $Pr(\exists s \in A : \mathcal{F}(s) \preceq c)$. For stochastic optimizers, the attainment function can be estimated from the experimental data of several different runs. Such an estimated is called the *empirical attainment function* (EAF).

The EAF can be seen as a distribution of the solution quality after running an algorithm for a specific amount of time. In particular, EAFs are used as a graphical tool for comparing algorithms in two or three dimensions. In such cases, boundaries of the regions of the cost space where EAF takes some constant value are usually of interest. Based on this idea, Fonseca *et al.* [FGLIP11] proposed the notion of *attainment surface*, which corresponds to a boundary which separates the objective space into two regions: first that are attained by outcomes of the stochastic algorithm and second that are not. Formally, this boundary is known as a k%-attainment surface, which is a set of hyperplanes separating the objective space attained by at least k percent of runs of an algorithm. For example, the median attainment surface corresponds to the region in the cost space attained by 50 percent of the runs. Similarly, the worst attainment surface (100%-attainment surface) corresponds to the region attained by all runs, whereas the best attainment surface corresponds to the limit between the region attained by at least one run and the region never attained. Fonseca et al. [FGLIP11] also gave efficient algorithms for computing EAF in two and three dimensions using dimension sweep algorithms.

Part I.

ON MULTI-OBJECTIVE OPTIMIZATION

3

MULTI-OBJECTIVE OPTIMIZERS

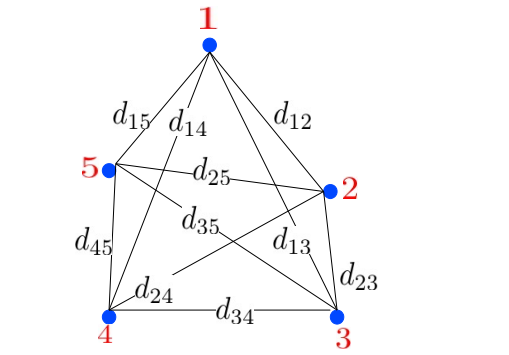
In this chapter, we review some of the popular heuristics used for solving multi-objective optimization problems. Specifically, we focus on the study of local search and evolutionary algorithms. Both these techniques have been quite effective in solving hard single-objective combinatorial optimization problems. In fact, many of the best-known solutions for travelling salesman problem and quadratic assignment problem are based on these methods or use them as crucial components. Several variants of local search and evolutionary algorithm have been defined in the context of multi-objective optimization. In next section, we discuss the general schema of local search algorithm for single-objective optimization problems.

3.1 Local Search for Single-Objective Optimization

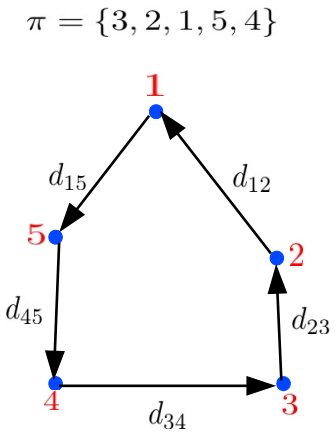
A single-objective optimization problem (SOP) can be viewed as a tuple $\varphi = (\mathcal{S}, \mathcal{C}, \mathcal{F})$, where S is the feasible solutions, $C \in \mathbb{R}$ is the cost (objective) space and $\mathcal{F} : \mathcal{S} \rightarrow \mathcal{C}$ is a function that maps the set of feasible solutions to costs. We assume, without the loss of generality, that $\mathcal{F}$ is to be minimized. Furthermore, we assume that the solution space is discrete and finite. The goal is to find a solution $s^* \in \mathcal{S}$ such that $\mathcal{F}(s^*)$ is optimal, *i.e.* $\mathcal{F}(s^*)$ is the minimum feasible value in the cost space.

Local search belongs to a family of search methods that explores a small subset $S' \subset \mathcal{S}$ of the solution space and returns the optimal solution obtained in this subset. The key issue in designing a local search method is how to come up with a good S'. The first ingredient of a local search algorithm lies in defining the term "local", that is defining a metric on $\mathcal{S}$. Each element of $\mathcal{S}$ is defined by values assigned to a finite number of discrete variables. We illustrate local search and its ingredient for an instance of the travelling salesman problem (TSP): given a list of n cities and distance d_{ij} for each pair of cities i and j, find the cheapest *path* such that each city is visited exactly once and the path ends at a city where it starts. A feasible solution for a TSP problem corresponds to an order in which each city is visited. Thus, each solution of $\mathcal{S}$ can be represented by a permutation $\pi = \{\pi_1, \pi_2, ..., \pi_n\}$ where π_i corresponds to some city in n such that $\forall i \neq j, \pi_i \neq \pi_j$ and if $i < j$ then city π_i is visited before city π_j. Figure 3.1 illustrates one element of solution space for a travelling salesman problem with 5 cities.

On such a discrete representation of the solution, a *local operator* $\mathcal{L} : \mathcal{S} \rightarrow \mathcal{S}$ is defined that transforms one solution to another solution by making a small change in the representation. For the above example, one can define a simple local operator



Complete graph with 5 vertices where a vertice corresponds to a city and an edge corresponds to the distance between two city



A feasible solution (a tour) corresponding to a permutation π

Figure 3.1. – An instance and a feasible solution for TSP with 5 cities.

which swaps the order in which two adjacent cities are visited (refer to Figure 3.2). The distance between s and s' is defined as the smallest number of changes required to transform s into s'. The next ingredient of a local search is the neighborhood of a solution. Formally, the neighborhood of a solution, denoted by $N(s)$, consists of the solutions whose distance from s is some constant (usually 1). For instance, in the above example, the neighborhood of a solution consists of all solutions which have two adjacent cities interchanged. Now we present the general scheme of a local search method in Algorithm 3.1.

Algorithm 3.1 Local Search Algorithm for single-objective optimization problem

1: $s :=$ generate a random solution
2: $B := \mathcal{F}(s)$
3: **while** (stopping criteria = FALSE) **do**
4: generate neighborhood $N(s)$ of the solution s
5: $s'' := argmin_{s' \in N(s)} \mathcal{F}(s')$
6: **if** $\mathcal{F}(s'') < B$ **then**
7: $P := \mathcal{F}(s'')$
8: **end if**
9: $s := select(N(s))$
10: **end while**

Observe that in each iteration, local search explores the neighborhood of the current solution s and computes their objective value. The best objective value B is updated when a neighboring solution with a better objective value is found. The next point in the

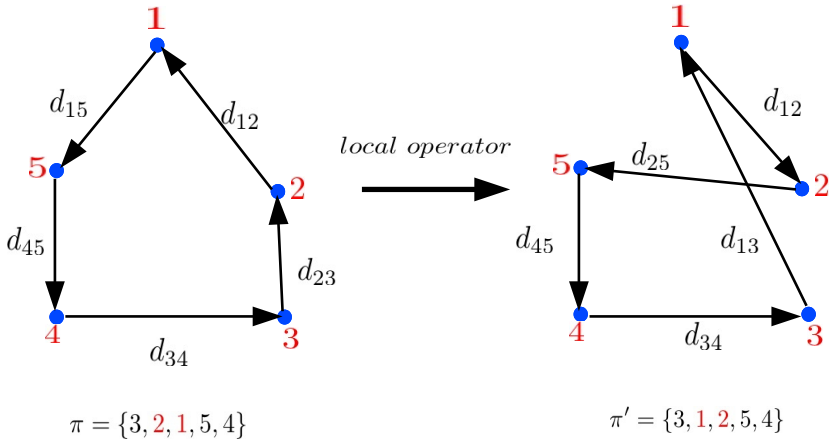


Figure 3.2. – Example of local operator for above TSP problem with 5 cities.

search is determined by the *select* procedure which can be deterministic or probabilistic, depending upon the variant of the local search algorithm.

The performance of any local search algorithm depends significantly on the underlying neighborhood relation and the *select* procedure. Generally, larger neighborhoods contain more and potentially better solutions and hence they offer better chances for improved solutions. But at the same time, the time complexity for computing objective values of a larger neighborhood is much higher. One attractive idea for improving the time complexity of the cost computation is to examine neighbors that are likely to yield better solutions. The neighborhood pruning techniques identify neighbors that provably cannot lead to improved solutions based on insights into the combinatorial structure of a given problem.

Another method for improving local search is to design the *select* procedure more efficiently. The most widely used selection strategy in iterative improvement algorithms is the *best improvement strategy* and the *first improvement strategy*. The best improvement strategy is based on selecting the best neighboring solutions. Formally, the *select* procedure, denoted by g, for the best improvement strategy can be defined as $g := argmin\{\mathcal{F}(s')|s' \in N(s)\}$. Note that the best improvement strategy requires evaluating all the neighbors in each iteration. Thus, if the size of the neighborhood is large, one needs to design an efficient algorithm to compute the cost of neighboring solutions. In the literature, this strategy is also known as *greedy hill climbing* or *discrete gradient descent*.

The first improvement strategy, on the other hand, tries to avoid evaluating the entire neighborhood by selecting the first improving solution encountered during the neighborhood generation. At each iteration, this strategy evaluates the neighboring solution $s' \in N(s)$ in a particular fixed order and the first solution s' for which $\mathcal{F}(s') < \mathcal{F}(s)$, is selected. In general, the neighborhood cost in the first improvement

algorithms can often be computed more efficiently than in the best improvement strategy, since in the former case, typically a small part of the local neighborhood is evaluated, especially as long as there are multiple improving neighbors from the current solution. However, the overall rate of improvement by the first improvement is typically smaller and therefore more iterations have to be performed in order to reach the same quality of solutions. Note that both of the above strategies search for better solutions in order to replace the current solution. Such neighborhood searches are often iterated until no more improvements can be found and the algorithm stops in what is known as *local optimum*. We present below a more formal definition of the local optimum solution.

Definition 3.1. *Let $N : \mathcal{S} \rightarrow 2^{\mathcal{S}}$ be a neighborhood function that associates a set of neighboring solutions $N(s)$ to every feasible solution s. A solution s is said to be* local optimum *with respect to N iff there is no $s' \in N(s)$ such that $f(s') < f(s)$.*

To avoid getting stuck in a local optimum, one can modify the *select* procedure to perform non-improving steps. The simplest technique for achieving this is to use a restart strategy that re-intializes the local search algorithm whenever it gets stuck in some local optimum. An interesting alternative is to design a *select* procedure that probabilistically determines in each iteration whether to choose an improving or randomly selection solution from the neighborhood. Typically this is done by introducing a *noise parameter*, that corresponds to the probability of selecting a random solution rather than an improving solution. The resulting algorithm is called *Randomized iterative improvement*.

Another commonly used probabilistic local search algorithm is *Simulated Annealing* (SA) which is inspired by the physical annealing process. Like randomized local search, SA starts with a random initial solution and performs randomized local search iterations wherein each iteration, a neighbor is chosen as the next point in the search based on the temperature parameter T. Throughout the run of SA, T is adjusted according to a given annealing schedule [KGV83]. Theoretically, under certain conditions, SA algorithms are known to converge to a globally optimal solution [GG84].

A fundamentally different approach for escaping local minima is to utilize the memory (visited solutions) in designing the *select* procedure. The resulting algorithms are known as *Tabu search* (TS) [Glo86]. TS typically uses the best improvement strategy which in local minima corresponds to a selecting a worsening solution. To prevent the algorithm from immediately returning to visited solutions and to avoid cycling, TS forbids selection of recently visited solutions by explicitly maintaining the list of recently visited solutions. Apart from above-mentioned techniques, there exist numerous other methods such as Variable Neighborhood search [MH97], Variable Depth Search [KL70], Dynasearch [PVdV70], etc all dedicated to escaping local minima.

3.2 Local Search for Multi-Objective Optimization

The adaptation of Algorithm 3.1 to multi-objective setting involves several issues. Note that unlike single-objective optimization, the cost space C of a multi-objective problem is a subset of $\mathbb{R}^d$, where d is the number of objective functions in $\mathcal{F}$. Therefore, the outcome B is not a singleton, but rather a set of incomparable costs encountered so far. Moreover, there may exist a set of solutions with incomparable cost in the neighborhood $N(s)$ of a solution s. This further complicates the design of an efficient *select* procedure.

Several adaptations of local search algorithms to the multi-objective settings have been proposed. Serafini [Ser94] and Ulungu *et al.* [UTF95] presented simulated annealing based algorithms for multi-objective combinatorial problems. They proposed the usage of the weighted norm in the design of *select* procedure. Essentially, if the s' and $s'' \in N(s)$ are incomparable, then the *select* procedure was based on local aggregation of all objective values using L^p-norm, where p is either 1 or ∞. The final output consisted of non-dominated solutions formed with the outcome of several runs of the algorithm with different sets of weight vectors. Later, Czyzżak *et al.* [CJ98] proposed another variant of these algorithms where weights of each objective are tuned in each iteration in order to assure a uniform distribution of the solutions. The concept of Tabu search has also been extended to multi-objective problems. Dahl *et al.* [DL95] generated a set of non-dominated solutions by solving the family of objective functions where each objective corresponds to the weighted summation of the objective functions with different weight vectors. Hertz *et al.* [HJRFF94] proposed the method of solving a sequence of single-objective problems considering, in turn, each objective $f_i \in \mathcal{F}$ with a penalty term. Later on, several variants of tabu search algorithm were proposed in the multi-objective settings [GMF97, GMU97, Han97, BAKC99, GF00].

Observe that all of the above techniques are based on scalarization techniques that basically convert a multi-objective problem into a single objective problem. A fundamentally different approach for designing a local search algorithm is inspired by the notion of Pareto optimality: a solution in the neighbourhood is optimal solution if it is not dominated by all non-dominated solutions found so far in the search. The resulting local search algorithm is known as *Pareto Local Search* (PLS) [PCS04]. In [PSS07], Paquete *et al.* presented soundness and completeness proofs for the PLS algorithm. Previous works have shown that PLS can find good approximation of the Pareto front in many combinatorial problems if it runs until completion [PS06, LT10a].

Many variations of PLS has been proposed in the literature, each having small differences in their neighborhood exploration strategy. Angel *et al.* [ABG04] proposed to use of *Dynasearch* and dynamic programming to explore the neighborhood of the bi-objective travelling salesman problem. They also proposed the idea of the bounded archive that accepts neighboring solutions whose objective values do not lie in the same partition of the objective space and a restart version of PLS. Liefooghe *et al.* [LHM$^+$12]

performed several experiments on the instances of the bi-objective travelling salesman and scheduling problems so as to compare various PLS algorithms using different parameters. To improve the performance of PLS, Lust *et al.* proposed a two-phase PLS algorithm [LT10b]. In the first phase of the algorithm, an initial population of the solutions was generated with single-objective solvers. Then PLS algorithm was applied in the second phase to each solution of the initial, so as to generate the solutions not found in the first phase. Later, Dubois-Lacoste *et al.* used this combination to solve the bi-objective flow-shop scheduling problems of minimizing the pairwise combination of makespan, the sum of completion times and weighted tardiness of all jobs [DLLIS11].

Alsheddy *et al.* proposed a guided Pareto local search (GPLS) that essentially first performs PLS and then tries to escape from a Pareto local optimal set by adjusting weights (penalties) of different objectives [AT09]. Another interesting way of escaping local optima is to use stochastic operators. Iterated PLS (SPLS) uses such stochastic operators that restart PLS from the solutions generated by applying path guided mutation to the currently found Pareto local optimal set [DT10]. In [DT12], Drugan *et al.* discuss and analyze different Pareto neighborhood exploration strategies with respect to PLS and proposed a deactivation mechanism that restarts PLS from a set of solutions in order to avoid the exploration of already visited regions. They applied two perturbation strategies: path-guided mutation and q-mutation exchange to escape a Pareto local optimal set (formally defined in next chapter).

3.3 Evolutionary Algorithms for Single-Objective Optimization

In section 3.1, we discussed search paradigms that manipulate one single solution of a given problem instance in each iteration. Another interesting extension is to consider procedures where several individual solutions are simultaneously maintained: this idea leads to population-based meta-heuristics. Perhaps the most prominent population-based meta-heuristics is the class of evolutionary algorithms (EA) where the candidate solutions in the population interact with each other directly. These algorithms are inspired by models of the natural evolution of biological species [B$\ddot{9}$6, Mit98]. EAs are iterative methods that start with a set of solutions and repeatedly apply a series of operators such as *selection, mutation, and recombination*. In each iteration of an EA, the set of current individuals are replaced by a new set of individuals using these operators. As an analogy to biological evolution, the population in each iteration of the algorithm is referred to as a *generation*.

In this thesis, we specifically focus on a special and most common class of evolutionary algorithms known as *genetic algorithms* (GA). We will describe the principles of GA's operation. To illustrate the working of GAs, we will focus on a 0-1 knapsack problem: given a list of n items where each item i admits a profit v_i and a weight w_i, find the subset of items that maximizes the total profit provided that the total weight of selected items is at most W. The solution space $\mathcal{S}$ of the knapsack problem con-

sists of all the subset of n items. Each solution of $\mathcal{S}$ can be represented by a vector $X = \{x_1, x_2, ..., x_n\}$ where x_i is 1 if item i is present in the subset, otherwise $x_i = 0$. Figure 3.3 illustrates one feasible solution for a knapsack problem with 10 items. This encoding of the decision variables in a binary string is primarily used to achieve a "pseudo chromosomal" representation of a solution. After choosing a binary representation scheme of a solution, GAs create a population of such strings at random on which *genetic operators* are iteratively applied. However, before we explain the notion of genetic operators on such strings, we will describe an intermediate step of assigning a measure to each solution.

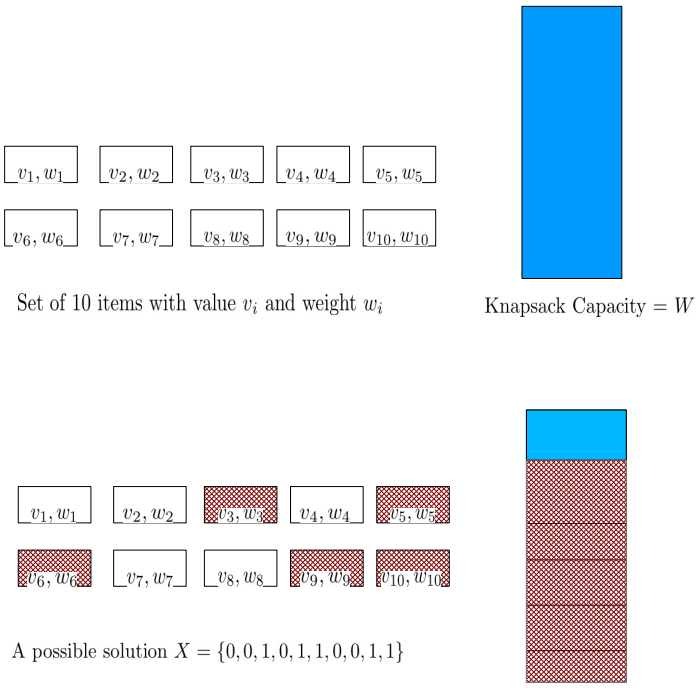


Figure 3.3. – Example of a feasible solution for knapsack problem with 10 items

It is necessary to evaluate the solution in the context of objective functions and constraints. The *fitness* of a string is a function that assigns a value which is a function of the solution's objective value and constraint violation. In the case of above knapsack example, the fitness of a solution can be evaluated as $\sum_{i=1}^{n} v_i x_i - \max\{0, (\sum_{i=1}^{n} w_i x_i - W)\}$. Since the knapsack problem is a maximization problem, solutions with larger fitness values are considered important. In general, the evaluation of a solution means calculating the objective value and checking constraint violations. Note that for infeasible solutions, the fitness value is the total profit minus the penalty in proportion to the constraint violation. Next, we explain operators commonly used in the design of a genetic algorithm.

Selection Operator: The general purpose of this operator is to make a large copy of "good" solutions so as to build a mating pool of most promising candidates. This, in turn, eliminates bad solutions from the population. This is achieved by a probabilistic filter that selects good solutions with a higher probability. The most common methods are *tournament selection* [MG96], *proportionate selection* [B$\ddot{9}$6], and *ranking selection* [GD91]. In the tournament selection, a tournament is repeatedly played between two solutions of the population and the better solution (according to fitness evaluation) is added to the mating pool. It has been shown that if carried out systematically, the tournament selection has similar convergence and computation time complexity properties as other operators [GD91]. In the proportionate selection method, the solutions are copied proportionate to their fitness values. If the average fitness of the population is f_a then a solution s_i with fitness f_i get an expected f_i / f_a number of copies. Since this method requires the computation of average fitness value, the proportionate selection method is slower compared to the tournament selection method. Furthermore, proportionate selection method has scaling issues as the outcome is dependent on the true values of the fitness.

Crossover Operator: Once a mating pool is generated using the selection operator, a crossover operator is applied. Typically, a crossover consists of creating one or two new individual solutions. There exist numerous ways to define a crossover operator [Spe98, VMC95, TYH99, OKK03], but in almost all of them, two individuals are randomly picked from the mating pool and some portion of their string are exchanged [Spe98]. For example in a one-point crossover operator, a crossing site along the string is chosen at random and all the bits on the right side of two solutions are exchanged. Figure 3.4 illustrates the application of one-point crossover operator to the Knapsack problem. Note that a crossover operator can lead to both better and worse solutions. Therefore, a crossover probability is used such that some portion of the population is preserved. The above concept of exchanging partial bits between two strings can also be generalized to n-point crossover operator, where an n cross-site are chosen at random.

Mutation Operator: After the application of the crossover operator, the mutation operator is applied to the newly generated individuals. A bit-wise mutation operator changes a single bit of a string with some mutation probability. This operator brings better diversity in the population than the crossover operator since the latter is just a recombination of existing solutions. Goldberg *et al.* [GD91] suggested a clock mutation operator where after a bit is mutated, the location of the next mutated bit is determined by an exponential distribution. This operator has also been used in [DA99].

Now that we have discussed all the ingredients, we present the pseudocode in Algorithm 3.2 illustrating the working principle of a genetic algorithm. The primary objective of genetic algorithms for combinatorial problem is to mature the population such that the promising regions of the search space are covered efficiently, thus resulting in high-quality solutions for the given problem. However, pure genetic algorithms

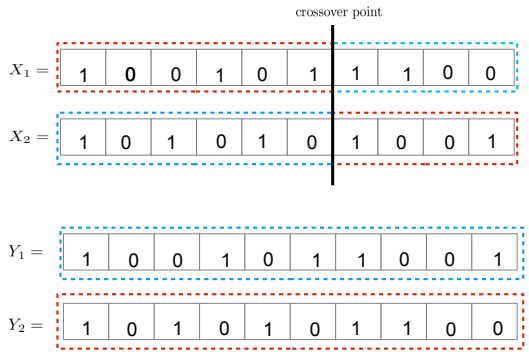


Figure 3.4. – One-point crossover for knapsack problem with 10 items where X_1 and X_2 correspond to two individuals picked randomly from a mating pool. Y_1 and Y_2 are two new solutions formed after the application of one-point crossover

Algorithm 3.2 Genetic Algorithm for single-objective optimization problem

1: Initialize Population
2: $gen := 0$
3: **while** (gen $\leq$ Required number of generation) **do**
4: Evaluate population and assign fitness values
5: Apply selection operator to build a mating pool
6: Apply crossover operator
7: Apply mutation operator
8: $gen := gen + 1$
9: **end while**

have been often criticized on their capability of search intensification. Therefore, in many practical applications, the performance of genetic algorithms can be significantly improved by combining them with a local search phase after mutation and recombination [Bra85, UAB+91, MF00] or by incorporating a local search process into the recombination operator [NK13, KP94, MF97].

3.4 Evolutionary Algorithms for Multi-Objective Optimization

A clear difference between a local search optimization method and a genetic algorithm is that the latter maintains a set of solutions and processes them iteratively. This gives GA an advantage while solving multi-objective optimization problems. GA's population approach can be exploited to maintain a set of non-dominated solutions. The first implementation of multi-objective evolutionary algorithm VEGA was suggested in the work of Schaffer [Sch85], who proposed an independent selection according to each objective. VEGA is the simplest and rather straightforward extension of single-objective

GA to the multi-objective case. In each generation, the population is randomly divided into d equal subpopulations. Each subpopulation is assigned a fitness function based on a different objective function. A separate mating pool using proportionate selection is generated for each subpopulation which is eventually combined together to make a single mating pool. Finally, crossover and mutation are performed on this combined mating pool. Observe that each solution in VEGA is evaluated with respect to only one objective function. This, in turn, limits the diversity of VEGA and it converges to a set of individual solutions that perform well with respect to individual objective functions.

Fonesca *et al.* [FF93] introduced a multi-objective genetic algorithm (MOGA) which used ranking based fitness assignment to each solution in the population. Each solution in MOGA is checked for its domination in the population. The rank of a solution s_i is set to one plus the number of solutions that dominate s_i is the population. Thus, the non-dominated solutions are ranked 1 and the maximum rank of a solution is no more than the total number of solutions in the population. Once the ranking is completed, a fitness value is assigned to solutions based on their ranking. This way, non-dominated solution are emphasized in a population. In order to maintain diversity (the spread of solutions in cost space), Fonseca *et al.* have introduced niching among the solutions of each rank. For this, they first calculated the normalized euclidean distance between any two solutions having the same rank and then computed a niche count using the sharing function mentioned in [GR87]. A shared fitness value is computed by dividing the fitness of a solution with its niche count. This, in turn, produces a large selection pressure on the solutions that reside in a less crowded region. Thereafter, a selection, a single point crossover, and bit-wise mutation operators are applied to create a new population.

Srinivas *et al.* implemented the concept of non-dominated sorting in genetic algorithms (NSGA) [SD94]. Like MOGA, NSGA also uses fitness assignment scheme based on non-domination criteria and a sharing strategy that preserves diversity. The solutions in the population are sorted according to non-domination into several equivalence classes where any two solutions in the same class are mutually incomparable. The first class of solutions corresponds to the best non-dominated set in the population, while the second best solutions belong to the second set and so on. The fitness assignment is performed from the first class (best non-dominated sets) and iteratively proceeds to other classes. The fitness values for all solutions in the first class are initialized to the size of the population. Further, these fitness values are degraded to maintain the diversity. That is, for each solution in the first class, its fitness value scaled down by its niche count, where the sharing function is applied to the solution of first class only. In order to proceed to the next front, the fitness values of the solutions in the second class are initialized to a value slightly smaller than the minimum shared fitness of the first class. This procedure is iteratively continued until all solutions in the population are assigned fitness values. NSGA applied stochastic roulette wheel operator [GDK89]

for selecting solution into the mating pool. The crossover and mutation operators are applied to the whole population.

Deb *et al.* proposed the idea of *elitist* non-dominated sorting genetic algorithm (NSGA-II) [DAPM00] where the offsprings population are first created using the crowded tournament selection, crossover and mutation operators. Then the offspring population and the parent population are combined together to form a merged population. A non-dominated sorting is used to classify the solutions of the merged population into multiple equivalence classes. The final population is generated by selecting solutions from the different classes according to their non-dominating ranks. That is, the filling starts with the best non-dominated solution and continues with the solutions from the second best non-dominated front and so on. All fronts which can not be accommodated into the fixed size population are deleted. The solutions from last allowed front are selected based on the niching strategy which prefers the solutions residing in the least crowded region in that front. Since the population size is fixed, NSGA-II can resort to cycle before converging to a well-distributed set of solutions.

Another interesting work in the domain of elitist evolutionary algorithm is the strength Pareto evolutionary algorithm (SPEA) [ZT98, ZT99] where elitism is introduced by maintaining an external population. This population stores a fixed number of non-dominated solutions that are found until so far. As the size of the non-dominated population can go beyond the fixed size of the external population, SPEA uses hierarchical clustering methods which preserve the less crowded elite solutions. Moreover, SPEA uses elite solutions for genetic operators in order to maintain the population in the good regions of the search space. In fact, SPEA assigns to a solution in external population, a smaller fitness value (known as strength) than those assigned to the current population. This method of fitness suggests that a solution with smaller fitness value is considered better. Finally, a binary tournament selection, a crossover and mutation operators are applied to create a new population. Later, Zitzler *et al.* proposed an improved version, namely SPEA-2, which incorporates a better fine-grained fitness assignment strategy, a density estimation technique, and an enhanced archive truncation method [ZLT01]. In particular, the fitness value assigned to a solution s_i is the sum of its strength (number of solutions s_i dominates in current and external population) and its density (inverse of the distance to the k-th nearest solution). Apart from above mentioned, many other evolutionary algorithms such as PAES [KC00], HypE [BZ11], ParEGO [Kno06], approximation guided EA [BFNW11], to name a few, have been proposed for solving the multi-objective optimization problems.

4

DOUBLE ARCHIVE PARETO LOCAL SEARCH

In this chapter, we discuss in detail Pareto local search algorithms that have been quite useful in solving multi-objective combinatorial problems. We show that under certain conditions these algorithms can converge prematurely to a local optimum. In contrast, we propose a new algorithm which improves upon these algorithms in terms of convergence to the Pareto front and spread of solutions in the cost space.

4.1 Pareto Local Search Algorithm

In this section, we revisit the Pareto local search introduced in [PCS04]. Algorithm 4.1 presents the pseudo-code of the algorithm. PLS starts with randomly generated solutions S_0. Since the neighbors of S_0 are not explored, solutions in S_0 are marked as unvisited. The algorithm randomly selects an unvisited solution s and explores its neighborhood $N(s)$. Line 11 attempts to insert $s' \in N(s)$ to the archive P using a min operator which is realized using a Pareto filter: a procedure that takes as input a set K and a solution r and returns a set consisting of the non-dominated solutions in $K \cup \{r\}$. The algorithm stops when all the solutions in the set P are visited. The pseudo-code for the min operator is given below in Algorithm 4.2.

Algorithm 4.1 Pareto Local Search $PLS(S_0, \mathcal{F})$

```
1:  Input: An initial set S_0 of incomparable solutions
2:  Output: A set P consisting of mutually incomparable solutions
3:  P := S_0
4:  for each s' ∈ P do
5:     visited(s') := False
6:  end for
7:  repeat
8:     s := select randomly a solution from P.
9:     for each s' ∈ N(s) do
10:       visited(s') := False
11:       P := min(P ∪ {s'})
12:    end for
13:    visited(s) := true
14: until ∀s' ∈ P : visited(s') = True
```

Algorithm 4.2 Pareto filter $min(K \cup r)$

Input : a set of mutually incomparable solutions K and a solution r
Output : a set of mutually incomparable solutions
$K' := \{s' \mid s' \in K \text{ and } \mathcal{F}(r) \prec \mathcal{F}(s')\}$
$K := K \setminus K'$
if $(\exists s' \in K : \mathcal{F}(r) || \mathcal{F}(s'))$ **then**
  $K := K \cup \{r\}$
end if
Return K

Observe that PLS applies the best improvement strategy that iteratively moves from a set of current solutions to a neighboring solution that improves upon them. The algorithm stops in a *Pareto local optimal set*: a set of solutions that have no improving solutions in their neighborhood. We formalize these notions below.

Definition 4.1. *[PCS04] Let $N : \mathcal{S} \rightarrow 2^{\mathcal{S}}$ be a neighborhood function that relates a subset of $\mathcal{S}$ to every feasible solution s. A solution s is said to be a* Pareto local optimum *with respect to N if and only if there is no $s' \in N(s)$ such that $\mathcal{F}(s') \prec \mathcal{F}(s)$. Moreover, a set P is a* Pareto local optimum set *with respect to N if and only if it consists of Pareto local optimum solutions with respect to N.*

To improve upon PLS performance, Drugan *et al.* [DT12] proposed a stochastic Pareto local search algorithm which aims to escape from local optimal sets by using a combination of mutation and recombination genetic operators. Algorithm 4.3 gives the pseudo-code of the Genetic Pareto local search (GPLS) algorithm.

GPLS first runs the PLS algorithm in order to construct an initial Pareto set P. Afterward, a solution s is randomly selected from P. A new solution s' is generated by mutating s (with probability α) or by recombining s (with probability $1 - \alpha$) with another solution s'' randomly selected from P. The solutions in P are *deactivated* using Algorithm 4.4 and the resultant Pareto set P' is provided as an initial input to the PLS algorithm. The archive P is finally updated with the outcome of the PLS run. This process is repeated until some stopping criterion is met. The GPLS algorithm also returns a *Pareto local optimum set*.

PLS has major advantages over prior works. First. it maintains only a set of non-dominated solutions, hence it provides fast convergence to a *Pareto local optimum set*. Furthermore, the size of the archive P is unlimited. This helps in providing a large number of incomparable solutions. However, one of its major drawbacks is that good candidate solutions are removed from the archive if dominated by other solutions. This premature deletion may limit the diversity of future exploration [IKdW$^+$14].

Algorithm 4.3 Genetic Pareto local search $GPLS(S_0, \alpha, \mathcal{F})$

Input: An initial set of incomparable solutions S_0
Output: A set P consisting of mutually incomparable solutions
$P :=$ PLS $(S_0, \mathcal{F})$
while NOT Termination **do**
  select s uniform randomly from P
  if $\alpha >$ rand$(0,1)$ or $|P| < 2$ **then**
    $s' :=$ Mutate(s)
  else
    Select $s'' \neq s$ from P
    $s' :=$ Recombine(s'', s)
  end if
  $P' :=$ Deactivate$(P, \{s'\})$
  $P := \min(P \cup \text{DAPLS}(P', f))$
end while
return P

Algorithm 4.4 Deactivation $Deactivate(r, K)$

Input: A Pareto Set K and a solution r
Output: A set K' consisting of mutually incomparable solutions
$K' := \{r\}$
for $\forall s' \in K$ **do**
  if $\mathcal{F}(r) || \mathcal{F}(s')$ **then**
    $K' := \min(K' \cup \{s'\})$
  end if
end for
return K'

4.2 Queued Pareto Local Search Algorithm

Inja et al. [IKdW+14] introduced a queue based Pareto local search (QPLS) which prevents the premature deletion of promising candidate solutions by maintaining a queue of solutions, which leads to a more diverse Pareto archive. The pseudo code for QPLS is presented in Algorithm 4.5.

QPLS starts with an initial queue Q of solutions and an empty Pareto archive P. A candidate solution s is popped from the queue and a recursive Pareto improvement function (PI) is applied. It improves the solution by repeatedly selecting a dominating solution from the neighborhood until no such improvements are possible. After a solution s is found that is not weakly dominated by any of its neighbors, it is compared to P. If no solution in the current archive dominates s, then solution s is added to P using the min operator. Following this, a set of k new incomparable candidate solutions

Algorithm 4.5 Queued Pareto Local Search $QPLS(Q, \mathcal{F})$

Input: An initial queue Q
Output: A set P consisting of mutually incomparable solutions
$P := \emptyset$
while Q is not empty **do**
  $s :=$ pop an element from Q
  $s :=$ PI$(s, \mathcal{F})$
  if $\exists p \in P : f(s) \not\prec f(p) \wedge f(s) \neq f(p)$ **then**
    $P = \min(P \cup \{s\})$
    $N := \{s' \in N(s) : f(s) \not\prec f(s')\}$
    Q.addK(N, k)
  end if
end while
return P

are randomly selected from the neighborhood of s and are added to the queue. This entire procedure is repeated until Q is empty.

Inja *et al.* proved that in a finite solution space, QPLS terminates in a finite number of steps for any finite initial queue. To avoid converging to a locally optimal set, QPLS is combined with a genetic framework that mutates and recombines the entire Pareto archive from the previous runs. The resultant algorithm is known as Genetic queued Pareto local search (GQPLS). The pseudocode for GPLS is presented in Algorithm 4.6.

GQPLS first finds a locally optimal Pareto achieve P by running QPLS on the initial Q. A new set of solutions is generated by mutating and recombining the solutions from the previous run. For each solution in P, either a mutation is performed with probability α or recombination with some other solution in P is performed with probability $(1 - \alpha)$ to generate a new solution. The newly generated solution is added to the queue. Finally, QPLS is called on the set of newly generated individuals. This process is repeated until some stopping criterion is met.

QPLS has a major advantage over PLS as the solutions which have not been explored are stored in a queue. This prevents the premature deletions of solutions. However, a drawback of the QPLS algorithm is that it applies a recursive Pareto improvement strategy. PI improves upon a solution by repeatedly selecting a single dominating solution from the neighborhood. Note that a neighborhood of a solution may consist of a set of non-dominated solutions which are mutually incomparable. In this case, QPLS selects only one such solution while discarding the rest. We present a new algorithm in the next section that does not delete candidate solutions prematurely.

Algorithm 4.6 Genetic Queued Pareto local search GQPLS($Q, \alpha, \mathcal{F}$)

1: **Input:** An initial queue Q
2: P :=QPLS $(Q, \mathcal{F})$
3: **while** NOT Termination **do**
4: Initialize Q to be an empty queue
5: **for each** $s \in P$ **do**
6: **if** $\alpha >$ rand$(0,1)$ or $|P| < 2$ **then**
7: $s' :=$ mutate(s)
8: **else**
9: Select $s'' \neq s$ from P
10: $s' :=$ Recombine(s'', s)
11: **end if**
12: Q.add(s')
13: **end for**
14: $P := \min(P \cup QPLS(Q, \mathcal{F}))$
15: **end while**
16: **return** P

4.3 Double Archive Pareto Local Search Algorithm

In this section, we present our Double archive Pareto local search algorithm that maintains an additional archive L which is maintained as a queue. Algorithm 4.7 depicts the general scheme of DAPLS.

Both archives P and L are initialized to a set S_0 of mutually incomparable solutions. While L is not empty, a solution s is selected and its entire (or partial) neighborhood $N(s)$ is generated. The current Pareto archive P is updated with solutions from $N(s)$ using a min operator. If a solution $s' \in N(s)$ is present in the updated Pareto archive, it is added to L in line 12 and remains there even if it is later removed from P. This procedure is repeated for all the solutions in neighborhood of s (lines 10-13).

As in previous work of SPLS [DT12], one can distinguish between two neighborhood generation strategies: best improvement and first improvement implementation. Observe that, DAPLS applies improvement once per iteration and saves all dominating points from P in L. Moreover, we maintain L as a queue. Hence, DAPLS explores the neighbors of $s \in L$ before the neighbors of $s' \in L$ if and only if s is added to L earlier than s'. This provides *fair* chance to the solutions for exploration and prevents premature convergence. Informally, DAPLS can also be seen as a *breadth-first* exploration of the search space using the Pareto dominance criteria.

DAPLS admits some natural properties mentioned below.

Property 1. *DAPLS is an iterative improvement algorithm with respect to its neighbours and strict non-dominance relation.*

Algorithm 4.7 DAPLS($S_0, \mathcal{F}$)

1: **Input:** An initial set of incomparable solutions S_0
2: **Output:** A set P consisting of mutually incomparable solutions
3: $P := S_0$
4: **for each** $s \in S_0$ **do**
5: L.add(s)
6: **end for**
7: **while** L is not empty **do**
8: $s :=$ pop a solution from L
9: $P := \min(P \cup N(s))$
10: **for each** $s' \in N(s)$ **do**
11: **if** $s' \in P$ **then**
12: L.add(s')
13: **end if**
14: **end for**
15: **end while**
16: **return** P

Property 2. *Let P_i and L_i denote the archives P and L, respectively at end of iteration i, then $\forall s \in L_i, \exists i' \leq i : s \in P_{i'}$.*

Property 3. *DAPLS terminates with a Pareto local optimum set.*

Property 1 holds since P is updated using the min operator. Thus at all times, P consists of incomparable solutions. The essence of our approach lies in Property 2. Essentially all the solutions which are inserted to P at iteration i are also inserted to archive L (line 12). At a later iteration i', it may happen that a new solution is removed from P in which case, L protects the unvisited solution. Recall that a solution from L can only be removed after its neighborhood has been explored. DAPLS protects all the dominating incomparable solutions from the neighborhood by inserting them in L.

Next, we present a simple genetic variation of DAPLS, depicted in Algorithm 4.8. Our genetic DAPLS escapes local optima by mutating and recombining the entire Pareto archive. It starts with executing DAPLS on an initial set of solutions. Once, a locally optimal Pareto archive P is obtained. It mutates (with probability α) or recombines (with probability $1 - \alpha$) all the solutions in P. Essentially, the skeleton of Genetic DAPLS is similar to Genetic QPLS where QPLS algorithm is replaced with DAPLS algorithm and updates to S are realized using non-dominance relation. This has a similar effect to that of deactivation scheme applied to the GPLS algorithm and helps DAPLS to explore only a small number of candidate solutions. Note that genetic DAPLS also runs until some stopping criterion is met.

Algorithm 4.8 Genetic DAPLS($S_0, \alpha, \mathcal{F}$)

Input: An initial set S_0 of incomparable solutions
Output: A set P consisting of mutually incomparable solutions
$P :=$ DAPLS (S_0, f)
while NOT Termination **do**
  $S := \emptyset$
  for each $s \in P$ **do**
    if $\alpha >$ rand$(0,1)$ or $|P| < 2$ **then**
      $s' :=$ mutate(s)
    else
      Select $s'' \neq s$ from P
      $s' :=$ Recombine(s'', s)
    end if
    $S := \min(S \cup \{s'\})$
  end for
  $P := \min(P \cup \text{DAPLS}(S, \mathcal{F}))$
end while
return P

4.4 Experimental Results

We compare DAPLS to QPLS and PLS on the *multi-objective quadratic assignment problem* (MQAP) [KC03].

Single objective QAPs are NP-hard combinatorial optimization problems that model many real-world situations like the layout of electrical circuits in computer aided design, scheduling, vehicle routing, etc. In fact, travelling salesman problem which is one of the most challenging problems in combinatorial optimization is a special case of QAP. Intuitively, QAPs can be described as the assignment of n facilities to n locations where the distance between each pair of locations is given and for each pair of facilities, the amount of flow (or materials) transported is specified. The aim is to find an optimal assignment of facilities to locations that minimizes the sum of products between distance and flows.

In this work, we consider the multi-objective version of QAPs (MQAP) introduced by Knowles *et al.*, where the flows between each pair of facilities are multi-dimensional values [KC03]. The values in flow matrices are correlated with factor ρ. If ρ is strongly positive then the Pareto front is small and is closer to being convex. This makes the problem harder. On the other hand, if the value of ρ is small or negative, then there exists a large number of Pareto optimal solution which is evenly spread out in the cost space.

Specifically, we are given n facilities and n locations such that d_{pq} denotes the distance between location p and location q. Moreover, we are provided with d flow

matrices $F^1, \cdots, F^d$, where F^i_{jk} denotes the flow from facility j to facility k in the i-th dimension. The aim is to minimize:

$$C^i(\pi) = \sum_{a=1}^{n} \sum_{b=1}^{n} F^i_{ab}.d_{\pi(a),\pi(b)}, \quad \forall i \in \{1, \cdots, d\}$$

where $\pi(.)$ is a permutation from set of all permutations $\Pi(n)$ of $\{1, 2, \cdots, n\}$. Given a permutation π, it takes $O(n^2)$ to compute the above cost functions.

Below we present a description of the neighborhood relation for QAPs. Furthermore, we also define a mutation and a recombination operators applied in the genetic version of the algorithms. Lastly, we present simple time-based stopping criteria so as to have a fair comparison of the performance of the algorithms.

Neighborhood Relation: MQAPs are permutation problems where a suitable neighborhood operator is the q-exchange operator that swaps the locations of q-facilities. In this work, we the use a 2-exchange operator that swaps the location of two difference facilities. It has two major advantages: the neighborhood size $\binom{n}{2}$, is relatively small and the time complexity of computing the incremental change in cost is linear [PS06].

Mutation Operator: We use the q-exchange mutation operator described in [DT10]. The q-exchange mutation randomly selects $q > 2$ locations $\{l_1, \cdots, l_q\}$, without replacement from a solution. A new solution is generated by exchanging these locations from left to right or from right to left with equal probability. For example, when exchanges are made right to left, the facility at l_i is shifted to location l_{i-1} where $i > 2$ and facility at location l_1 is shifted to location l_q. Note that a new solution is q-swaps apart from the original solution. Since our neighborhood operator is 2 exchange operator, we use $q > 2$ exchange mutation to escape from the local optima.

Recombination Operator: Drugan et al. [DT10] also introduced the idea of the *path-guided mutation* for QAP problems where two solutions s and s' are selected from the current Pareto archive such that the distance is at least q. An offspring s'' is generated by copying the solution s. The set of common cycles for two solutions, s'' and s' are identified. A cycle is a minimal subset of locations such that the set of their facilities is the same in both parent solutions. For example, in Figure 4.1 there are two cycles between s' and s'': $\{2, 7, 5, 3\}$ and $\{6, 8, 4, 1\}$. Then a cycle c is randomly chosen. For $q-1$ times, choose at random a location i in the cycle c from solution s'', where $s''[i] = s'[j]$ and $i \neq j$. Exchange the facilities of $s''[i]$ and $s'[j]$. Thus, the distance between s'' and its first parent s, is increased by 1 and the distance between s'' and the second parent s', is decreased by 1. If the size of c is smaller or equal to q, a second cycle is chosen. This process of randomly selecting a cycle and swap locations is repeated until the distance between s'' and s is q. If there are no parent solutions at distance larger or equal to q, we generate a solution with the mutation operator.

Stopping Criteria: In the majority of the MOEA literature, the algorithms are compared using the number of fitness function evaluation rather than the execution time. On

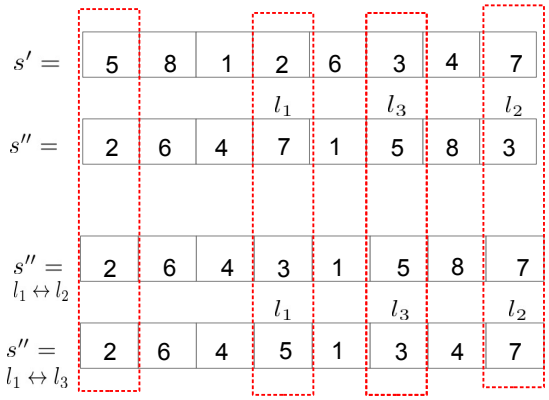


Figure 4.1. – An example of 3-exchange path mutation borrowed from [DT10]. The solution s' and s'' form two cycles from which cycle $\{2,3,5,7\}$ is randomly chosen. The three positions l_1, l_2 and l_3 in cycle are used to decrease the distance between s' and s''.

the other hand, neighborhood generating operators do not perform full fitness evaluations. Instead they perturb many small changes to the solutions, which can be evaluated in a fractional amount of time. Therefore, we measure the outcome of different algorithm as a function of time. We run each algorithm for the same fixed amount of time for each instance.

Comparison Methodology

We generated multiple QAPs problem instances with different correlation factors and facilities. For bi-objective QAP problem, we choose a large number of facilities, $n \in \{50,75\}$ with correlation factors $\rho \in \{-0.25,-0.75,0.25,0.75\}$. For tri-objective QAP, since the number of non-dominated solutions in the Pareto front increases exponentially with dimension, we restricted our choice to rather small number of facilities $n \in \{20,25\}$ with the combination of correlation factor $\rho \in \{0.25.0.75\}$. To have a fair comparison, we kept the runtime for each instance across all the algorithms a constant. For example, for a small bi-objective instance with 50 facilities and $\rho = 0.25$, all algorithms were executed for 20 minutes. Due to stochasticity of the algorithms, each experiment was executed 20 times for each instance. The comparison of performance is carried out using a hypervolume unary indicator [ZT99]. This indicator measures the volume of the cost space which is weakly dominated by an approximating set.

Bi-qap	Volume		
n, ρ	GSPLS	GQPLS	DAPLS
50, -0.75	0.92 ± 0.008	0.93 ± 0.002	$\mathbf{0.94 \pm 0.001}$
50, -0.25	0.94 ± 0.010	0.97 ± 0.011	$\mathbf{0.99 \pm 0.008}$
50, $+0.25$	0.93 ± 0.015	0.95 ± 0.002	$\mathbf{0.98 \pm 0.005}$
50, $+0.75$	0.84 ± 0.012	0.82 ± 0.015	$\mathbf{0.88 \pm 0.006}$
75, -0.75	0.80 ± 0.007	0.83 ± 0.010	$\mathbf{0.86 \pm 0.002}$
75, -0.25	0.75 ± 0.001	0.79 ± 0.007	$\mathbf{0.81 \pm 0.009}$
75, $+0.25$	0.81 ± 0.006	0.83 ± 0.001	$\mathbf{0.86 \pm 0.001}$
75, $+0.75$	0.77 ± 0.001	0.77 ± 0.013	$\mathbf{0.83 \pm 0.014}$

Table 1. – Performance of GSPLS, GQPLS and Genetic DAPLS on 8 large bi-objective instances of QAP in terms of normalized hypervolume

Table 1 shows the average performance, in terms of normalized hypervolume [1] for bi-objective QAP instances. For all instances, genetic DAPLS outperforms both algorithms. Moreover, we observe that for small and negative correlation factors the difference between genetic DAPLS and other methods is almost negligible. This can be explained by the fact that for such instances the solutions are evenly spread out in the cost space and all algorithms can find them easily. On the other hand, for strongly positive correlation factors, the number of solutions is small and rather restricted to some part of the cost space. This in turn increases the complexity of finding solutions approximating the Pareto front for such instances. Our method improves upon the previous algorithms and finds a much better approximate Pareto front for these hard instances.

Similarly, Table 2 shows the average performance for tri-objective QAP instances. Here, ρ_1 represents the correlation factor between objectives 1 and 2 while ρ_2 presents the correlation between objectives 1 and 3. Like the bi-objective case, genetic DAPLS outperforms all other methods in terms of hypervolume. We observe that especially for large correlation factors ($\rho_1 = 0.75$ and $\rho_2 = 0.75$), DAPLS finds better approximate solutions.

For comparison, we also use visualization of EAFs from outcomes of multiple runs of each algorithm. An approximating set is k%-approximation set if it weakly dominates exactly those solutions that have been attained in at least k percent of runs. We show 50%-approximation set of EAFs for bi-objective instances with positive correlation. These plots are generated using the $\mathcal{R}$-statistical tool with the library *EAF*. The details of the generating algorithm can be found in [LIPS10].

1. We used hypervolume generating tool from *http://lopez-ibanez.eu/hypervolume* for comparison.

Tri-qap	Volume		
n, ρ_1, ρ_2	GSPLS	GQPLS	DAPLS
$20, 0.25, 0.25$	0.81 ± 0.027	0.84 ± 0.003	$\mathbf{0.86 \pm 0.001}$
$20, 0.25, 0.75$	0.79 ± 0.001	0.81 ± 0.001	$\mathbf{0.83 \pm 0.003}$
$20, 0.75, 0.25$	0.78 ± 0.013	0.80 ± 0.001	$\mathbf{0.82 \pm 0.002}$
$20, 0.75, 0.75$	0.73 ± 0.011	0.75 ± 0.002	$\mathbf{0.82 \pm 0.001}$
$25, 0.25, 0.25$	0.90 ± 0.019	0.92 ± 0.003	$\mathbf{0.96 \pm 0.002}$
$25, 0.25, 0.75$	0.82 ± 0.012	0.84 ± 0.006	$\mathbf{0.87 \pm 0.001}$
$25, 0.75, 0.25$	0.80 ± 0.014	0.82 ± 0.008	$\mathbf{0.84 \pm 0.001}$
$25, 0.75, 0.75$	0.78 ± 0.012	0.82 ± 0.023	$\mathbf{0.87 \pm 0.001}$

Table 2. – Performance of GSPLS, GQPLS and Genetic DAPLS on 8 tri-objective instances of QAP in terms of normalized hypervolume

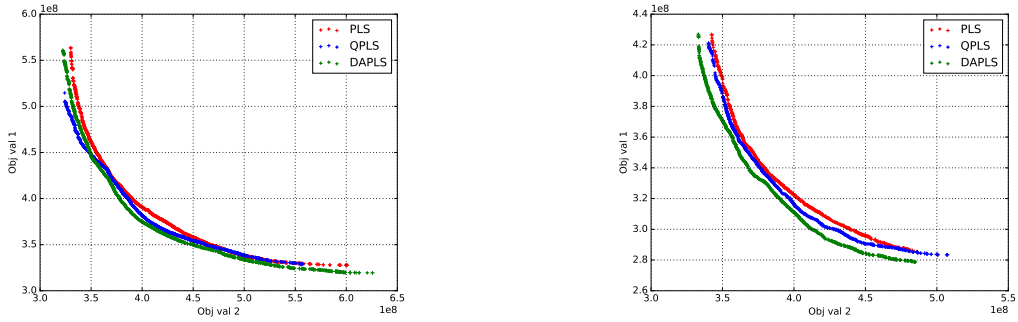


Figure 4.2. – Median attainment surfaces for $n = 50$ with $\rho = 0.25$ (on left) and $\rho = 0.75$ (on right)

Figure 4.2 shows the median attainment surfaces for bi-objective instance of 50-facilities with correlation factors 0.25 and 0.75. Similarly, Figure 4.3 shows the median attainment surface for 75 facilities for ρ equal to 0.25 and 0.75. For instances with correlation factor 0.25, it is clearly visible that genetic DAPLS achieves better spread of solutions (diversity) in the cost space than GSPLS and GQPLS. Similar improvements are also observed for the instances with correlation factor 0.75, where genetic DAPLS not only achieves better diversification but also provides solutions which are closer to the Pareto front.

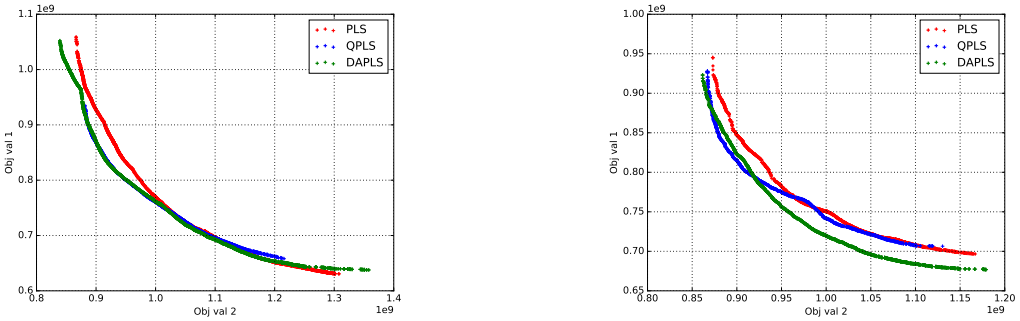


Figure 4.3. – Median attainment surfaces for $n = 75$ with $\rho = 0.25$ (on left) and $\rho = 0.75$ (on right)

4.5 Conclusion and Future work

We developed a new local search algorithm for approximating Pareto fronts. Our algorithm is based on the principle that the neighbors of solutions that were non-dominated at some stage of the search process should be explored before they are discarded. To escape the local optimal set, we embedded DAPLS in a genetic framework. We showed empirically that genetic DAPLS outperforms GSPLS and GQPLS algorithms on several instances of bi-objective and tri-objective quadratic assignment problems.

In the future, we would like to study DAPLS for higher dimensional cost spaces where the size of the Pareto front is typically huge. This, in turn, causes an increase in the runtime of the inner loop of genetic DAPLS. Our next aim would be to use some heuristics to limit the size of the neighborhood. We would also like to see how DAPLS performs on other multi-objective problems like the knapsack, coordination graphs, etc.

Part II.

ON NON-PREEMPTIVE SCHEDULING

5

INTRODUCTION TO SCHEDULING

Scheduling is a combinatorial problem of allocating resources to a set of requests so as to optimize some performance objectives. Such problems have been studied in various disciplines such as management science, computer science, finance, economics, etc. This has lead to a virtually unlimited number of models for scheduling problems. In this thesis, we focus on the client-server model where resources are represented as machines or processors, whereas requests correspond to jobs/tasks that arrive over time. The general aim of a scheduler is to assign the sequence in which the jobs are executed on the machines. The typical example of such systems includes operating systems, high-performance platforms, web-servers, database server, etc.

One of the most common and natural measure for the quality of service delivered to a job is the amount of time it spends in a system. Mathematically, this can be modelled as the *flow time* of a job, which is defined as the amount of time a job remains in the system until it is fully served. Informally, the flow time can be related to the waiting time of a job in a system. Several variants of the flow time problem have been investigated in different settings. We specifically focus on *non-preemptive* settings where a job is executed uninterruptedly to its completion time. This is particularly important because scheduling with interruptions (preemptions) usually have a huge management overhead, which can impact the overall performance of the system.

5.1 Preliminaries

A scheduling problem generally consists of a set of independent jobs $\mathcal{J} = \{J_1, J_2, ..., J_n\}$ and a set of machines $\mathcal{M} = \{1, 2, ..., m\}$. Each job $J_j \in \mathcal{J}$ is characterized by its release date r_j and a set of processing times $p_{ij}, \forall i \in \{1, 2, ...m\}$. Depending upon the machine type, scheduling problems are generally studied in four different settings.

1. **Single machine**: This is the simplest and most commonly studied setting where all jobs in $\mathcal{J}$ are scheduled on a single machine *i.e.* $m = 1$. Moreover, each job $J_j \in \mathcal{J}$ has a processing time p_j.
2. **Identical machines**: This setting is the natural generalization of single machine case where each job $J_j \in \mathcal{J}$ are scheduled on a set of similar parallel machines $\mathcal{M}$ such that $\forall i \in \mathcal{M} : p_{ij} = p_j$.
3. **Related machines**: Here, each job $J_j \in \mathcal{J}$ has a processing requirement of p_j and can be scheduled on a set of machines $\mathcal{M}$ where machine $i \in \mathcal{M}$ executes jobs at a speed s_i . Thus, a job j requires $\frac{p_j}{s_i}$ time units on machine i.

4. **Unrelated machines**: This is most general setting where each job $J_j \in \mathcal{J}$ has a processing time of p_{ij} on the machine $i \in \mathcal{M}$.

In addition to above machine models, we will also consider problems where each job is given a degree of importance. The importance of a job J_j is represented by its weight w_j. The total number of total jobs in an instance is denoted by n. Given a schedule, the completion time C_j of job J_j is defined as the time at which J_j is fully served with respect to its resource requirement. Then, the flow time (or response time) F_j is defined as the time J_j spends in the system *i.e.* $F_j = C_j - r_j$.

Another related measure to the flow time objective is the *stretch*, which is defined as the factor by which a job is slowed down with respect to it's processing time. Stretch is often used in fair scheduling where users are willing to wait longer for large jobs as opposed to small jobs. In fact, stretch is a special case of the weighted flow time measure where the weight w_j of a job is equal to the inverse of it's processing time *i.e* $S_{ij} = \frac{F_j}{p_{ij}}$ where J_j is entirely scheduled on machine i [BCM98].

Typically, most of the scheduling problems are studied under two separate scenarios: *offline* and *online* settings. In case of offline settings, the entire instance $\mathcal{J}$ is known in advance where the aim is to design efficient algorithms that achieve optimal performance guarantees. Unfortunately, many such scheduling problems are NP-hard *i.e.* there exists no polynomial time algorithm to find an optimal solution unless $P = NP$. Thus, a widely accepted approach is to relax the notion of optimality and find a polynomial time solution with a provable good performance guarantee. The performance guarantees of offline problems are usually expressed in the terms of approximation ratio. Below we define more formally the notion of the approximation algorithm.

Definition 5.1. *Let $\mathcal{A}$ and OPT denote an algorithm and some fixed optimal solution, respectively. Then $\mathcal{A}$ is said to be an α-approximation algorithm iff for all instances of the optimization problem*

$$\mathcal{F}(\mathcal{A}) \leq \alpha.\mathcal{F}(OPT)$$

where $\mathcal{F}$ denotes the objective function.

Informally, the idea here is to find a solution whose objective value is close to the optimal solution. Moreover, the running time of the algorithm generating such a solution should be small (polynomial in the size of input instance). Note that for most of the offline scheduling problems, one can find an optimal solution by simply enumerating all the permutations of the jobs. But the time required for such an enumeration is exponential in the size of input instance. Thus, the approximation algorithm can be seen as a trade-off between the quality of a solution and the search time.

The online setting differs from the offline setting in the sense that the existence of a job is revealed at the time of its arrival. Thus, an *online algorithm* has to make decisions based on the partial knowledge of the instance that has arrived until that time. In this thesis, we focus on problems arising in the clairvoyant scheduling, where the set of

processing times p_{ij} of a job j is revealed on its arrival time r_j. As in the offline setting, for most the online scheduling problems finding an optimal solution is infeasible in a small amount of time. Moreover, the partial knowledge of the instance complicates this problem further as the notion of optimal is not very clear. In such a case, the performances of online algorithms are given in terms of the *competitive ratio* where the cost of the online algorithm is compared to the cost of the offline optimal solution. Below we formally define the notion of competitive ratio.

Definition 5.2. *Let $\mathcal{A}$ and OPT denote an online algorithm and some fixed offline optimal solution, respectively. Then $\mathcal{A}$ is said to be an ρ-competitive algorithm iff for all instances of the optimization problem*

$$\mathcal{F}(\mathcal{A}) \leq \rho.\mathcal{F}(OPT)$$

where $\mathcal{F}$ denotes the objective function.

Note that in above definition, we assume that the objective function is a minimization problem. The competitive ratio for maximization problems can be defined similarly. Moreover, the performance of an online algorithm is compared to the optimal offline solution *i.e.* where the optimal algorithm knows the entire instance in advance whereas the online algorithm is made aware of the instance as jobs arrive.

Both of the above definitions can be extended to multi-objective analysis, where an algorithm's performance is measured using two or more objective functions. For instance in a bi-objective problem, the performance of an online algorithm is measured terms of (ρ_1, ρ_2)-competitive ratio which is defined as:

Definition 5.3. *An online algorithm is said to be (ρ_1, ρ_2)-competitive iff for all instances of the online optimization problem, the algorithm produces a solution that is simultaneously ρ_1-competitive for the first objective and ρ_2-competitive for the second objective.*

Note that in the above definition, the performance of an online algorithm is compared to two separate offline algorithms. In terms of general multi-objective problems, this can be seen as comparing the cost of a solution with an ideal point.

5.2 Resource Augmentation

The above performance measures are based on the worst case analysis of the algorithm *i.e.* the performance of an algorithm is measured on the most difficult instances of the problem. However, in practice, such instances are rare. In fact, many algorithms which have arbitrarily huge performance guarantees with respect to the optimal algorithm, perform quite well under practical settings. Therefore, the standard performance measures like approximation ratio or competitive ratio turns out to be overly pessimistic.

In such cases, a different form of analysis known as *resource augmentation* has been proven useful [KP00, PSTW97].

Resource augmentation refers to the relaxed notion of analysis where the algorithm's performance is computed on the set of slightly extra resources in comparison to the optimal solution [KP00]. For example, consider a case of speed augmentation where the algorithm is allowed to execute the jobs $(1+\epsilon)$ times faster than the optimal algorithm, where $\epsilon > 0$. In other words, the work that the optimal algorithm can do in an 1 unit of time will require $\left(\frac{1}{1+\epsilon}\right)$ unit of time in the algorithm. The power of resource augmentation lies in the fact that it has successfully provided theoretical evidences for many scheduling algorithms with good performance in practice.

Below we formally define the notion of resource augmentation in terms extra speed and extra machine.

Definition 5.4. *An online algorithm $\mathcal{A}$ is said to be s-speed c-competitive iff for all instances of the problem $\mathcal{A}$ is c-competitive and executes job s times faster than the offline optimal policy.*

Definition 5.5. *An online algorithm $\mathcal{A}$ is said to be m-machine c-competitive iff for all instances of the problem $\mathcal{A}$ is c-competitive and the number of machines available to $\mathcal{A}$ is m times the number of machines available to the offline optimal algorithm.*

The speed and the machine augmentation have been extensively applied to preemptive flow minimization problems. In fact, many scheduling algorithms which behave poorly under the classical models, are shown to have $O(1)$-competitive ratio under resource augmentation [KP00, PSTW97].

In contrast, we specifically focus on non-preemptive flow time minimization problems. In general settings, non-preemptive flow minimization problems are much harder and therefore had very little success in above resource augmentation model. We study such problems in a recently proposed model of rejection. Unlike previous resource augmentation model, the rejection model does not provide any extra resources; instead it allows an algorithm to discard a small fraction of the jobs. Then, the idea is to find an algorithm which can provide a good performance guarantee for the remaining set of jobs. Below we formally define the definition of algorithm's performance in terms of rejection model.

Definition 5.6. *Under rejection model, an online algorithm $\mathcal{A}$ is said to be f-rejection c-competitive iff for all instances of the problem $\mathcal{A}$ is c-competitive and is allowed to reject at most f-fraction of total number of jobs.*

The performance of the algorithm is computed on the non-rejected set of jobs and compared to the offline optimal solution for all jobs. Intuitively, the rejection can be seen as a resource augmentation where a small fraction of the jobs have very huge number of resources at its disposal. Above definitions can be similarly extended to offline

algorithms in terms of approximation ratios. Many preemptive problems with strong lower bound in the resource augmentation model have been solved efficiently in the rejection model [CDK15, CDGK15]. In contrast, we focus on study of non-preemptive flow time minimization problems. Such problems are much harder to solve than their preemptive counterpart. In fact, there is a limited literature focusing on the study of non-preemptive problems. In the following chapters, we will present some new results for these problems in different settings.

5.3 Related Works

5.3.1 Results without resource augmentation

The stretch metric was originally introduced to study the fairness for HTTP requests arriving at web servers [BCM98]. Bender *et al.* [BCM98] showed that the problem of optimizing *max-stretch i.e* minimizing the maximum stretch attained by a set of jobs, in a non-preemptive offline setting cannot be approximated within a factor of $\Omega(n^{1-\epsilon})$, unless $P = NP$. To show this, they reduced an arbitrary instance of 3-partition problem to an instance of the non-preemptive max-stretch problem. Then using the adversary technique for online algorithms, they presented a lower bound of $\Omega(\Delta^{\frac{1}{3}})$ for the max-stretch problem on a single machine. Finally, they provided an online preemptive algorithm using a variant of classical EDF strategy (*earliest deadline first*) and showed that it is $O(\sqrt{\Delta})$-competitive, where Δ is the ratio of largest to smallest processing time.

Later, Legrand *et al.* showed that First-Come First-Served algorithm (FCFS) is Δ-competitive for the *max-stretch* problem on a single machine [LSV08]. Since the preemption is not used in FCFS, above bound is also valid for the non-preemptive case. They also showed that the problem of optimizing *max-stretch* on a single machine with preemption cannot be approximated within a factor of $\frac{1}{2}\Delta^{\sqrt{2}-1}$. Saule *et al.* showed that all approximation algorithms optimizing max-stretch cannot have a competitive ratio better than $\frac{1+\Delta}{2}$ on a single machine [SBÇ12].

For the problem of minimizing average stretch, Muthukrishnan *et al.* [MRSG99] showed that the classical preemptive strategy of scheduling jobs with shortest remaining processing time (SRPT) is 2-competitive on a single machine. Their analysis relies on a careful comparison of the set of unfinished jobs in SRPT with the set of unfinished jobs in any other scheduling algorithm. Later, Bender *et al.* presented a polynomial time approximation scheme with a time complexity of $O(n^{poly(\frac{1}{\epsilon})})$ [BMR03]. A more general problem of minimizing average stretch is the problem of minimizing the average weighted flow time ($\sum w_i F_i$). In preemptive settings, there is no known online algorithm with a constant competitive ratio for the average of weighted flow time on a single machine. The best known guarantee is given by a randomized algorithm which achieves an approximation ratio of $O(\log\log\Delta)$ [BP14]. In online setting, Chekuri *et*

al. [CKZ01] and Bansal *et al.* [BD07] presented online algorithms with $O(\log W)$ and $O(\log^2 \Delta)$ competitive ratio, respectively, where W is the ratio of the largest weight over the smallest weight. Furthermore, Bansal *et al.* showed that any algorithm should have a competitive ratio of at least $\Omega(\min\{\sqrt{\frac{\log W}{\log\log W}}, \sqrt{\frac{\log\log \Delta}{\log\log\log \Delta}}\})$ [BC09]. In terms of instance dependent parameter, Tao *et al.* [TL13] proved that the weighted SPT is $(\Delta+1)$-competitive for the total weighted flow-time objective.

In case when weights are equal to 1 and preemptions are allowed, a well-known online strategy of scheduling jobs with shortest remaining processing time (SRPT) provides an optimal solution for minimizing the average flow time on a single machine [BT09]. Whereas in the non-preemptive setting, Kellerer *et al.* [KTW95] showed that there exists a strong lower bound of $O(n^{\frac{1}{2}-\epsilon})$ on the inapproximability of the total flow time minimization problem and presented an algorithm which achieves $O(\sqrt{n})$-approximation for the flow time problem on single machine. In the online setting, Chekuri *et al.* [CKZ01] showed that any algorithm minimizing the non-preemptive total flow should have a competitive ratio $\Omega(n)$. Bunde [Bun04] later proved that the online strategy of scheduling jobs with shortest processing time (SPT) strategy is $\frac{\Delta+1}{2}$-competitive for the average flow-time minimization problem. He also showed that this is the best achievable bound for optimizing the online average flow time on a single machine.

In the identical parallel machines setting, SRPT is a 14-competitive algorithm for the preemptive average stretch minimization problem [MRSG99] and $O(\log(\min\{\frac{n}{m}, \Delta\}))$-competitive algorithm for the preemptive problem of minimizing average flow-time [LR07]. Leonardi *et al.* further showed that the competitive ratio of any randomized online algorithm is $\Omega(\log \frac{n}{m})$ and $\Omega(\log \Delta)$. In non-preemptive case, a general technique for the total flow-time minimization problem has been presented in [LR07], which transforms any preemptive schedule into a non-preemptive schedule by loosing a factor of $O(\sqrt{\frac{n}{m}})$. Thus, this technique yields an $O(\sqrt{\frac{n}{m}}\log(\min\{\frac{n}{m}, \Delta\}))$-approximation algorithm for minimizing the non-preemptive flow time on a set of parallel machines. Lastly, they also presented a lower bound of $\Omega(n^{\frac{1}{3}})$ for this problem.

5.3.2 Results with resource augmentation

Bechetti *et al.* [BLMSP06] showed that the online preemptive scheduling policy of weighted shortest remaining time first is $(1+\epsilon)$-speed $O(\frac{1}{\epsilon})$-competitive for the average weighted flow-time minimization problem. In the rejection model, Choudhary *et al.* [CDK15] presented an $O(\frac{1}{\epsilon^{12}})$-competitive algorithm for the same problem where the algorithm rejects at most an ϵ-fraction of total weights of the job. Moreover, they showed that their result also holds on the set of identical machines. If preemptions are not allowed, Bansal *et al.* [BCK$^+$07] showed that there exist algorithms that is 12-speed $(2+\epsilon)$-approximation algorithm for the total flow-time objective and a 12-speed 4-approximation algorithm for the total weighted flow-time objectives. First,

they proposed a new integer programming formulation whose the relaxation is close to an optimum. Then, they transformed the schedule obtained from solving relaxed LP into a feasible non-preemptive schedule. Using a different approach of dynamic programming, Im *et al.* [ILMT15] proposed a quasi-polynomial time framework that achieves $(1+\epsilon)$-speed and $(1+\epsilon)$-approximate solution for the total weighted flow-time minimization problem and a $(1+\epsilon)$-speed and $O(1)$-approximate solution for the total flow-time minimization problem. Moreover, they showed that their framework can be easily extended to a set of parallel machines.

In [PSTW97], an $O(\log \Delta)$-machine 1-competitive algorithm has been proposed for the total weighted flow-time objective even for identical machines. For the unweighted version, the authors also proposed an $O(\log n)$-machine $(1+o(1))$-competitive algorithm and an $O(\log n)$-machine $(1+o(1))$-speed 1-competitive algorithm. Note that algorithms in [PSTW97] work in the online setting but they need to know the minimum and the maximum processing times in advance. Moreover, a m-machine $(1+\Delta^{1/m})$-competitive algorithm was presented in [EVS01] for the total flow-time minimization problem, if Δ is known a priori to the algorithm. The authors also provided a lower bound which shows that their algorithm is optimal up to a constant factor for any constant m.

Another general problem related to weighted flow time is the problem of minimizing the weighted ℓ_k-norm of flow time. Bansal *et al.* [BP03] first studied this problem in preemptive settings and showed that the policy of scheduling jobs with highest density first (HDF) is $(1+\epsilon)$-speed, $\frac{1}{\epsilon}$-competitive where the density of a job is equal to the weight of the job divided by its processing time. Anand *et al.* [AGK12] improved this bound and presented a primal-dual based algorithm that is $(1+\epsilon)$-speed, $O(k/\epsilon^{2+1/k})$-competitive. Thang [Ngu13] further improved this bound and provided an algorithm that is $(1+\epsilon)$-speed, $O(k/\epsilon^{1+1/k})$ and does not need to know ϵ a priori. To the best of our knowledge, no competitive algorithm is known for non-preemptive problem of weighted ℓ_k-norm of flow time.

5.4 Our Results

The main contribution of this part of the thesis are summarized below.

1. In Chapter 6, we study the online problem of minimizing max-stretch on a single machine. We show an improved lower bound of $\alpha\Delta$, where $\alpha = \frac{\sqrt{5}-1}{2}$. Then, we design a new *semi-online* algorithm which asymptotically achieves the same performance guarantee. At last, we show that our algorithm achieves the competitive ratio of Δ^2 for the problem of minimizing the average stretch.
2. In Chapter 7, we study the problem of minimizing average flow and average stretch under the rejection model. We present a *polynomial time* algorithm that converts a preemptive SRPT schedule into a feasible non-preemptive schedule with a performance guarantee of $O(\frac{1}{\epsilon})$ and rejects at most ϵ-fraction of total

number (weights) of jobs for the problem of minimizing the average flow (the average stretch) on a single machine.

3. In Chapter 8, we study the online problem of minimizing average weighted flow time on unrelated machine. Here we first present a generalized framework that unifies multiple variants of resource augmentation. Then, we design a primal-dual algorithm that is $(1+\epsilon_s)$-speed $O\left(\frac{1}{\epsilon_r.\epsilon_s}\right)$-competitive ratio for average weighted flow time problem and rejects at most ϵ_r-fraction of the total weight of the jobs.
4. In Chapter 9, we extended the analysis to ℓ_p-norm of the weighted flow and showed that the same algorithm (as in Chapter 8) achieves $(1+\epsilon_s)$-speed $O\left(\frac{k^{(k+3)/k}}{\epsilon_r^{1/k}.\epsilon_s^{(k+2)/k}}\right)$-competitive and rejects at most ϵ_r-fraction of the total weight of the jobs.

6

SCHEDULING TO MINIMIZE MAX-STRETCH ON A SINGLE MACHINE

6.1 Introduction

In this chapter, we consider basic single processor scheduling scenario where jobs arrive over time and the aim of a scheduler is to optimize some function that measures the performance or quality of service delivered to jobs. One of the most relevant performance measures in job scheduling is the *fair* amount of time that jobs spend in the system. This includes both the waiting time due to processing some other jobs and the actual processing time of the job itself. We consider *stretch* as an objective for *fair* scheduling of jobs.

Here, we are interested in scheduling a stream of jobs to minimize the maximum stretch (*max-stretch*) on a single machine. This problem is denoted as $1|r_i, online|S_{max}$ in the classical 3-fields notation of scheduling problems [LLRKS93]. The problem of max-stretch admits no constant approximation in the *offline setting* unless $P = NP$ [BCM98]. Though interesting results can be derived by introducing an instance-dependent parameter Δ: the ratio between the largest and the smallest processing time in the instance.

6.2 Problem Definition

We study the problem of scheduling a set of n independent jobs on a single machine where jobs arrive over time and their processing time is known only at their release times. A scheduling instance is specified by the set of jobs J. Without loss of generality, we assume that the smallest and largest processing times are equal to 1 and Δ, respectively. In this work, we consider the *semi-online* version of the problem where the value of Δ is known a priori.

In a given schedule σ_i, C_i and S_i denote the start time, completion time and stretch of job i, respectively where $S_i = \frac{C_i - r_i}{p_i}$. We are interested in minimizing $S_{max} = \max_{j \in J} S_j$.

6.3 Lower Bounds on Competitive Ratios

Definition 6.1. *A scheduling algorithm $\mathcal{A}$ is said to be* greedy *iff there does not exist a time t where the machine is idle while $\mathcal{A}$ has a set of unfinished jobs.*

Note that the lower bounds mentioned below are expressed in the term of competitiveness defined in Chapter 6. According to the general definition of competitiveness [BEY98], these bounds hold true iff Δ is not a constant.

Observation 6.1. *Any greedy algorithm for scheduling jobs on a single machine has a competitive ratio of at least Δ for minimizing the max-stretch objective.*

Proof. We present a simple proof using the adversary technique. At time 0 a job of processing time Δ arrives. Any greedy algorithm schedules it immediately on the processor. At time ϵ, a small job of processing time 1 is released. Since preemption is not allowed, the greedy algorithm can only schedule the small job at time $t = \Delta$ and thus attaining $S_{max} \approx \Delta$. On the contrary, the optimal algorithm finishes the small job first and hence has a stretch close to 1 for both jobs; more precisely of $S^* = \frac{\Delta+\epsilon}{\Delta}$. □

Hence, for an improved bound any algorithm should incorporate some waiting time strategy. We show below a lower bound on the competitive ratio of such algorithms using a similar adversary technique.

Theorem 6.1. *There exists no ρ-competitive non-preemptive algorithm for minimizing* max-stretch *for any fixed $\rho < \frac{\sqrt{5}-1}{2}\Delta$.*

Proof. Let $\mathcal{A}$ be any scheduling algorithm. At time 0, a job x of size Δ is released. Now, consider the following two behaviors of the adversary.

— **Case 1**:
If algorithm $\mathcal{A}$ schedules job x at time t such that $0 \leq t \leq \frac{\sqrt{5}-1}{2}\Delta$, then the adversary releases a job y of size 1 at time $t + \epsilon$ where $0 < \epsilon \ll 1$. The algorithm $\mathcal{A}$ schedules job y after the completion of job x. Therefore, job x and job y achieve a stretch of at least $\frac{t+\Delta}{\Delta}$ and $\Delta + 1$, respectively. On other hand, the adversary first waits for $t + \epsilon$ amount of time and then schedules job y before job x. In which case, the stretch of job x and job y is at most $\frac{t+1}{\Delta} + 1$ and 1, respectively. Hence, the competitive ratio of $\mathcal{A}$ is greater than $\frac{\sqrt{5}-1}{2}\Delta$, for sufficiently large values of Δ.

— **Case 2**:
If $\mathcal{A}$ schedules job x at time t such that $t > \frac{\sqrt{5}-1}{2}\Delta$, then the adversary releases a job z of size 1 at time Δ. As in the above case, the algorithm $\mathcal{A}$ schedules job z after the completion of job x. Therefore, job x and job z achieve the stretch of at least $\frac{t+\Delta}{\Delta}$ and $t + 1$, respectively. Whereas, the adversary schedules both job x and z at their respective release times and therefore attains the stretch of 1 (for both x and z). The competitive ratio, in this case, is at least $\frac{\sqrt{5}-1}{2}\Delta$.

□

6.4 The Algorithm

We design a *semi-online* non-preemptive algorithm for optimizing max-stretch on a single machine. The algorithm is semi-online in that the parameters P_{min} and P_{max} are known in advance. To develop the intuition, we briefly consider the case where all jobs have been released before time t. Let $\mathcal{K}$ denote the set of such jobs. Our objective is to determine if there exists a schedule such that all jobs achieve the stretch of at most S. For each job $j \in \mathcal{K}$, we define a deadline $d_j = r_j + Sp_j$. Note that $r_j \leq t$. Then the above problem is transformed into a problem of scheduling jobs with their deadlines. Brucker [Bru01] showed that *Earliest Deadline First* (EDF) schedules all jobs before their deadlines if such a schedule exists.

Note that in above scenario, the problem can be efficiently solved since all the jobs have been released until time t. In the online setting, where the release times and processing times are not known in advance and jobs have to be scheduled non-preemptively, computing efficient deadline is not feasible. Moreover, Observation 6.1 states that any algorithm, which has to achieve better than the Δ-competitive ratio, must wait for some amount of time before it starts scheduling large jobs. This further complicates the problem of computing appropriate deadlines.

Waiting time strategies have been studied for problems of minimizing completion time [LSS03, NS04]. Here, we focus on the problem of minimizing max-stretch. To best of our knowledge, this is the first work which studies waiting time strategies in the context of the flow-time based objective. Our algorithm works in two steps: first, it forces the large jobs to wait for some time before they can be considered for scheduling and second, it maintains an *online estimate* of max-stretch for assigning deadlines to each job. In case, jobs cannot be scheduled with their current deadlines, our algorithm adjusts the online estimate to provide a new set of deadlines. The details of our algorithm are mentioned below.

Wait-Deadline Algorithm (WDA)

Based on the processing times, we classify the online arriving jobs into two sets, namely *large set* and *small set* (denoted by J_{large} and J_{small}, respectively). A job $i \in J_{small}$ if and only if $p_i \leq 1 + \alpha\Delta$, where $\alpha = \frac{\sqrt{5}-1}{2}$. Otherwise a job $i \in J_{large}$. Note that J_{large} consists of jobs which have processing time *strictly* greater than $1 + \alpha\Delta$.

WDA maintains two separate queues, the *Ready queue* (denoted by Q_R) which contains jobs that are *available* for scheduling and the *Wait queue* (denoted by Q_W) where jobs belonging to *large* set are waiting. More specifically, when a job i is released, it is placed directly into Q_R if $j \in J_{small}$. On the other hand, if the job $i \in J_{large}$, it is initially placed in Q_W where it waits for αp_i units of time before it is shifted to Q_R. WDA picks a job from Q_R and schedules them one by one in a specific order.

Based on the management of both queues, WDA is triggered at three separate events:

1. when a new job is released (denoted by E_1),
2. when some job completes its waiting time (denoted by E_2), and
3. when a job completes its execution (denoted by E_3).

On the occurrence of event E_1, WDA classifies the newly arrived job and updates Q_R or Q_W, correspondingly. In case if some job completes waiting, the event E_2 is triggered where WDA shifts this job from Q_W to Q_R. Lastly, if a job completes its execution (event E_3), WDA removes a job from Q_R and schedules it on the machine. The selection of a job from Q_R is performed using the pseudocode presented in Algorithm 6.1. In case, the multiple events occur at the same time, WDA prioritizes them in order of E_1 followed by E_2 which is followed by E_3. Moreover, after the end of each event, WDA checks the state of the machine. In case if the machine is idle, as in the event E_3, WDA removes a job from Q_R and schedule it on the machine. The entire procedure based on events is summarized in Algorithm 6.2.

Algorithm 6.1 Job selection in WDA

Input: Ready queue Q_R at time t and the previous *max-stretch* estimate $S(t')$, where $(t' \leq t)$
Order all jobs in Q_R according to their release time
Compute the *max-stretch* and set it as UB
$LB := \max\{S(t'), 1\}$
while $LB \neq UB$ **do**
  $M := (UB + LB)/2$
  $\forall j \in Q_R : d_j(t) = r_j + Mp_j$
  Schedule all jobs in Q_R according to the *Earliest Deadline First* policy
  if all jobs complete within their deadline **then**
    $UB := M$
  else
    $LB := M$
  end if
end while
$S(t) := UB$
Return the job of Q_R with the earliest deadline according to $S(t)$, where ties are broken according to the processing time of the job (the shortest job is returned)

Intuitively, we modify the release time of every job $i \in J_{large}$ to a new value $r_i + \alpha p_i$. Let t be the time at which the machine becomes idle. Then the input to Algorithm 6.1 is the set Q_R and the previous max-stretch estimate. WDA sets the deadline $d_i(t)$ for each job $i \in Q_R$ where $d_i(t) = r_i + S(t)p_i$, where $S(t)$ is the new estimated *max-stretch* such that all jobs in Q_R can be completed. Note that the deadline $d_i(t)$ uses the original release time r_i rather than the modified release date. For already released jobs, $S(t)$ can be computed in polynomial time using a binary search similarly to the technique used

Algorithm 6.2 Wait-Deadline algorithm

1: Initial State: Q_R and Q_W are empty sets
2: Wait for events to occur.
3: Let t be the time at which events occurred.
4: **while** At least one event occurring at time t has not been processed **do**
5: **switch** (Event)
6: **case** Job i has been released**:**
7: the new job is in J_{small}
8: Update Q_R.
9: Create a new event at time $t + \alpha p_i$ and update Q_W.
10: **case** Job i finished its waiting period**:**
11: Remove i from Q_W and add it to Q_R.
12: **case** Job i finished its execution**:**
13: Nothing special to do in this case for Q_R and Q_W.
14: **end switch**
15: **if** $Q_R \neq \emptyset$ and the machine is idle **then**
16: Select a new job to execute using Algorithm 6.1 and remove it from Q_R.
17: **end if**
18: **end while**
19: Return to the first line to wait for the next time instant when events occur.

in [BCM98]. The upper bound for the binary search can be derived from the FCFS schedule of all jobs in Q_R, while 1 is a natural lower bound at time $t = 0$. At any later time $t > 0$, whenever a job has to be selected, WDA uses the previous stretch estimate as a lower bound for the new binary search. Algorithm 6.1 outputs the job with the earliest deadline.

Before we start with the competitive analysis, remember that $\alpha = \frac{\sqrt{5}-1}{2}$. Indeed Theorem 6.1 suggests that for an instance of two jobs with size 1 and Δ, it is optimal to wait for $\alpha\Delta$ time units before the job of size Δ is scheduled. When the size of jobs can take any values between 1 and Δ, the partitioning of jobs in J_{small} and J_{large} ensures that small jobs can be scheduled as soon as they arrive while large jobs wait a fraction α of their processing time before they can be scheduled.

6.5 Analysis for Max-stretch

6.5.1 Some definition and properties related to WDA

Let WDA denote the schedule produced by our algorithm. We use r'_i to denote the modified released time of job i, that is $r'_i = r_i$ if job $i \in J_{small}$, otherwise $r'_i = r_i + \alpha p_i$. Moreover $d_i(t)$ denotes the *estimated deadline* of job i at time t *i.e*, $d_i(t) = r_i + S(t)p_i$.

Property 4. *Let t, t' denote the two times such that $t \leq t'$, then $S(t) \leq S(t')$.*

Let z be the job in WDA that attains the *max-stretch* among jobs in J. We remove all jobs from the instance J that are released after the start of job z without changing the S_z and without increasing the optimal stretch. Similarly, we also remove the set of jobs that are scheduled after the job z in WDA, without changing S_z and without increasing the optimal stretch. Therefore, we assume without loss of generality, that z is the latest job in J that is processed in WDA.

Property 5. *The machine is busy for during time interval $[r'_i, C_i), \forall i \in J$.*

Definition 6.2. *Let J_B denote the set of jobs that start and complete their execution during the interval $[r'_z, \sigma_z)$,* i.e,

$$J_B = \{i \in J : r'_z \leq \sigma_i < \sigma_z\}$$

Now we define the relationship between the deadlines of jobs in set J_B and deadline of z.

Consider a job $i \in J_B$. Since $r'_z \leq \sigma_i < \sigma_z$, both i and z are available in *Ready queue* at time σ_i. Then it must be the case that $d_i(\sigma_i) \leq d_z(\sigma_i)$. This relationship can be derived between each $i \in J_B$ and z. The next property formalizes this relationship.

Property 6. *For all $i \in J_B$, it holds that $d_i(\sigma_i) \leq d_z(\sigma_i)$.*

Observation 6.2. *There does not exist a job $i \in J$ such that the following two conditions are met simultaneously,*

— $p_i > p_z$
— $r_i > r_z$

Proof. We prove this by a simple contradiction. Suppose that there exists some job i such that $r_i > r_z$ and $p_i > p_z$. Since z is the latest job in WDA, it must be the case that job i belongs to set J_B. Then Property 6 implies that $S(\sigma_i) \leq \frac{r_z - r_i}{p_i - p_z} < 0$. This contradicts the fact that the lower bound on *online stretch* estimate at any time is at least 1. □

The completion time of job z in WDA can be formulated as:

$$C_z = r_z + S_z p_z \tag{6.1}$$

where S_z is the stretch of z attains in WDA.

6.5.2 Defining Optimal Schedule and its relation to WDA

Let OPT denote some fixed optimal schedule. For the rest of this analysis, a superscript of $*$ indicates that the quantities in question refer to OPT. Our general approach is to relate the stretch of job z with the stretch of some job in the optimal schedule.

In OPT, there exists a job which completes at or after time $C_z - \delta$, where $\delta \leq \alpha\Delta$. This can be explained by the fact that $\alpha\Delta$ is the *maximum* difference between the makespan of schedules WDA and OPT. In the rest of the analysis, let y denote one such job in OPT. Hence, the completion time of job y in OPT can be written as $C_y^* \geq C_z - \delta$. Without the loss of generality, we assume that $\sigma_y^* < C_z - \alpha\Delta$. Since y is schedule in OPT with stretch S_y^*, we also have $C_y^* = r_y + S_y^* p_y$. Combining this inequality with Equation 6.1, it implies that $r_y + S_y^* p_y \geq r_z + S_z p_z - \delta$.

Rearranging the terms in previous equation, we get:

$$S_z \leq S_y^* \left(\frac{p_y}{p_z}\right) + \frac{r_y - r_z}{p_z} + \frac{\delta}{p_z} \tag{6.2}$$

where $\delta \leq \alpha\Delta$.

Theorem 6.2. *WDA is $(1+\alpha\Delta)$-competitive for the problem of minimizing max-stretch non-preemptively.*

The proof is constructed on the case-by-case analysis of Equation 6.2. The main cases are considered in: Lemma 6.1, Lemma 6.2 and Lemma 6.5 where each lemma corresponds to different ratio of processing time of z and y. Specifically, Lemma 6.1 considers the case when $p_y \leq p_z$ while Lemma 6.2 and Lemma 6.5 consider the case when $p_z < p_y \leq (1+\alpha\Delta)p_z$ and $(1+\alpha\Delta)p_z < p_y$, respectively. Depending on the start time of y in WDA, we further divide these cases into sub-cases where a different lower bound on *max-stretch* is computed.

As aforementioned, we use the notation $S(t)$ to refer *online stretch* estimate at time t such that all jobs in the *Ready queue* can be scheduled within their respective deadlines. Also note that $S(\sigma_i) \geq S_i$ for all job $i \in J$.

6.5.3 Consider a scenario where $p_y \leq p_z$

Lemma 6.1. *If $p_y \leq p_z$, then it holds that $S_z \leq S_y^* + \alpha\Delta$.*

Proof. We consider the following two cases:

1. *Suppose $y \in J_B$.* Since $r'_z < \sigma_z < \sigma_y$, Property 4 implies that $S(\sigma_y) \leq S(\sigma_z)$. As job z attains the max-stretch we have $S(\sigma_y) \leq S(\sigma_z) \leq S_z$. Substituting, this inequality in Property 6 leads to $r_y - r_z \leq S_z(p_z - p_y)$. Consequently, Equation 6.2 can be formulated as:

$$
\begin{aligned}
S_z &\leq S_y^* \left(\frac{p_y}{p_z}\right) + \left(1 - \frac{p_y}{p_z}\right) S_z + \frac{\delta}{p_z} \\
S_z &\leq S_y^* + \frac{\alpha\Delta}{p_y} \qquad\qquad \text{since } p_y \leq p_z \\
S_z &\leq S_y^* + \alpha\Delta
\end{aligned}
$$

2. *Suppose* $y \notin J_B$. Let β be a binary variable such that

$$
\beta = \begin{cases} 1, & \text{if } z \in J_{large}. \\ 0, & \text{otherwise.} \end{cases} \tag{6.3}
$$

We re-formulated r'_z as $r'_z = r_z + \beta\alpha p_z$. Since $r_y < \sigma_y < r'_z$, it implies that $r_y - r_z < \beta\alpha p_z$. Applying this inequality to Equation 6.2 we get,

$$
S_z \leq S_y^* \left(\frac{p_y}{p_z}\right) + \frac{\delta}{p_z} + \beta\alpha \leq S_y^* + \frac{\alpha\Delta}{1 + \beta\alpha\Delta} + \beta\alpha \qquad \leq S_y^* + \alpha\Delta
$$

Note that the second inequality is due to the fact that the lower bound on the processing time of z can be expressed as $p_z \geq 1 + \beta\alpha\Delta$.

□

For the remaining cases, it follows that job z is processed before job y in OPT, $p_z < p_y$ and $r_y < r_z$.

6.5.4 Consider a scenario where $p_z < p_y \leq (1 + \alpha\Delta)p_z$

First we define the notion of *limiting* jobs which plays a crucial role.

Definition 6.3. *Let $i, j \in J$ such that the following conditions are true:*

1. $p_i > p_j$
2. $r'_i \leq r'_j$
3. $r'_j \leq \sigma_i < \sigma_j$
4. $r'_j \leq \sigma_j^* < \sigma_i^*$

*Then we say that job i **limits** job j.*

Property 7. *If i limits j then $S_i \geq 1 + \frac{p_i}{p_j} - \frac{p_j}{p_i}$.*

Proof. From Property 6, it follows that the estimated deadline of i is at most the estimated deadline of j at σ_i. This can be expressed as $r_i + S(\sigma_i)p_i \leq r_j + S(\sigma_i)p_j$. Since job i is scheduled earlier than job j, the estimated stretch $S(\sigma_i)$ is at least $\frac{p_i + p_j}{p_j}$.

Combining these two inequalities with the fact that $p_i > p_j$, we have $S_i = \frac{\sigma_i - r_i + p_i}{p_i} \geq \frac{r_j - r_i + p_i}{p_i} > 1 + \frac{p_i}{p_j} - \frac{p_j}{p_i}$. □

Now we show the competitive ratio for *max-stretch* when $p_y \leq (1 + \alpha\Delta)p_z$.

Lemma 6.2. *If $p_z < p_y$ and $p_y \leq (1 + \alpha\Delta)p_z$ then $S_z \leq S^*(1 + \alpha\Delta)$.*

Proof. We consider two following cases.

— **Suppose** $\delta \leq 0$: Here, the completion time of z in WDA is no later than the completion time of job y in OPT. Therefore, the Equation 6.2 can be simplified to

$$S_z \leq S_y^* \left(\frac{p_y}{p_z}\right) + \frac{r_y - r_z}{p_z}$$

From Observation 6.2, it follows that $r_y - r_z \leq 0$. Therefore above inequality can be expressed as $S_z \leq S_y^* \left(\frac{p_y}{p_z}\right) \leq S^*(1 + \alpha\Delta)$.

— **Suppose** $\delta > 0$: We split this particular cases into three subcases:

— **Assume that** $y \in J_B$. Then, Property 6 implies that $r_y - r_z \leq S(\sigma_y)(p_z - p_y)$. Since $p_y > p_z$ and $S(\sigma_y) \geq 1$, it implies that $r_y - r_z \leq p_z - p_y$. Putting this in Equality 6.2, we get:

$$\begin{aligned} S_z &\leq S_y^* \left(\frac{p_y}{p_z}\right) + \frac{p_z - p_y}{p_z} + \frac{\delta}{p_z} \\ &\leq (S_y^* - 1) \left(\frac{p_y}{p_z}\right) + \frac{1 + \delta}{p_z} \\ &\leq (S_y^* - 1)(1 + \alpha\Delta) + (1 + \alpha\Delta) = S_y^*(1 + \alpha\Delta) \end{aligned}$$

The second last inequality is due to the fact that $S_y^* \geq 1$ and the right hand side is maximized when $p_y \leq (1 + \alpha\Delta)p_z$.

— **Assume that** $y \notin J_B$ **and** $C_y \leq r'_z$. Then $r_y + S_y p_y \leq r_z + \beta\alpha p_z$, where $\beta = 0$ if $z \in J_{small}$, otherwise $\beta = 1$. If $z \in J_{large}$, then $y \in J_{large}$. In this case $S_y > 1 + \alpha$. On other hand, if $z \in J_{small}$, then $S_y \geq 1$. Therefore, the lower bound on S_y can be expressed as $1 + \beta\alpha$. Combining the lower bound on S_y with previous inequality, we have $r_y - r_z \leq \beta(\alpha p_z - \alpha p_y) - p_y$. Putting this in Equation 6.2, we get:

$$\begin{aligned} S_z &\leq S_y^* \left(\frac{p_y}{p_z}\right) + \frac{\beta(\alpha p_z - \alpha p_y) - p_y}{p_z} + \frac{\delta}{p_z} \\ &\leq (S_y^* - 1) \left(\frac{p_y}{p_z}\right) + \frac{\beta\alpha(p_z - p_y)}{p_z} + \frac{\delta}{p_z} \end{aligned}$$

$$\leq (S_y^* - 1)\left(\frac{p_y}{p_z}\right) + \frac{\alpha\Delta}{p_z} \qquad \text{since } p_z < p_y$$
$$\leq S_y^*(1+\alpha\Delta)$$

The last inequality is due to the fact that $S_y^* \geq 1$ and the right hand side is maximized when $p_y \leq (1+\alpha\Delta)p_z$.

— **Assume that** $y \notin J_B$ **and** $C_y > r'_z$. Observe that during the interval $I = [\sigma_y, C_z)$, the machine is completely busy in WDA. We claim that there exists a job k such that $[\sigma_k, C_k) \subseteq I$ and $[\sigma_k^*, C_k^*] \not\subseteq I$. Assume there exists no such job. Then OPT is busy during the entire interval I. But this is in contradiction to the fact that $\delta > 0$. Now, we consider two sub-cases:

— **Consider** $r_k \geq \sigma_y$. This implies that $C_z < C_k^*$. Furthermore, assume that $p_k \leq p_z$, then Lemma 6.1 implies that competitive ratio holds. On the contrary if $p_k > p_z$, then Observation 6.2 implies that k is released earlier than job z. Since both k and z are release during the processing time of y, we can bound $r_z - r_k$ by at most $p_y \leq (1+\alpha\Delta)p_z$. Moreover $k \in J_B$. Using Property 6, we have $r_k + S(\sigma_k)p_k \leq r_z + S(\sigma_k)p_z$ where $S(\sigma_k) \geq \frac{p_k+p_z}{p_z}$. This implies that $\frac{p_k^2-p_z^2}{p_z} \leq r_z - r_k$. Combining this inequality with the upper bound of $(1+\alpha\Delta)p_z$, we get $p_k \leq (\sqrt{2+\alpha\Delta})p_z$. As $C_k^* > C_z$ and $p_k \leq p_z(\sqrt{2+\alpha\Delta})$, we get $S_z \leq S_k^*(\sqrt{2+\alpha\Delta}) \leq S^*(1+\alpha\Delta)$.

— **Consider** $r_k < \sigma_y$ If $p_k \leq p_z$ then by Property 7, we have $S_y > 1 + \frac{p_y}{p_k} - \frac{p_k}{p_y} > 1 + \frac{p_y}{p_k} - \frac{p_z}{p_y}$. Since $r'_z \leq C_y$ and $y \notin J_B$, we have $r_y + S_y p_y - p_y < r_z$. Using both inequalities in Equation 6.2 proves that our bound holds in this case. Conversely suppose that $p_k > p_z$. Since $k \in J_B$, using Property 6 we have $r_k + S(\sigma_k)p_k \leq r_z + S(\sigma_k)p_z$. As intermediate stretch estimate is a non-decreasing function of time, $p_k > p_z$ and $\sigma_y \leq \sigma_k$, we have $r_k + S(\sigma_y)p_k < r_z + S(\sigma_y)p_z$. Hence $r_y + S(\sigma_y)p_y < r_k + S(\sigma_y)p_k < r_z + S(\sigma_y)p_z$. The above facts imply that $r_y - r_z < S(\sigma_y)(p_z - p_y) < p_z - p_y$ since $p_z - p_y < 0$. Substituting this inequality in Equation 6.2 gives $S_z \leq S_y^* \frac{p_y}{p_z} + 1 - \frac{p_y}{p_z} + \frac{\alpha\Delta}{p_z} \leq (S_y^* - 1)\frac{p_y}{p_z} + 1 + \alpha\Delta \leq S_y^*(1+\alpha\Delta)$.

□

6.5.5 Proving the bound when $(1+\alpha\Delta)p_z < p_y$

In this case, job z and job y belong to class J_{small} and J_{large}, respectively. To simplify the notations, from here on we will refer to r'_z as r_z.

Definition 6.4. *We define $J_U(t)$ as set of jobs that have not been scheduled until the time t, i.e. $J_U(t) = \{i \in J : r_i \leq t < \sigma_i\}$.*

Then the following lemma relates the stretch estimates $S(t)$ shortly after r_z with jobs in $J_U(r_z)$.

Lemma 6.3. *in WDA, let j be the first job that starts processing after r_z. Then for any time $t \geq \sigma_j$, the estimated stretch $S(t)$ is at least $\frac{\sum_{i \in J_U(r_z)} p_i + \sigma_j - r_z}{p_z}$.*

Proof. Let i^* be the some job in $J_U(r_z)$ which has latest deadline according to *online stretch* $S(\sigma_j)$. We claim that $p_{i^*} \leq p_z$. Thus, we have $S(\sigma_{i^*}) \geq \frac{\sigma_j - r_z + \sum_{i \in J_U(r_z)}}{p_{i^*}} \geq \frac{\sigma_j - r_z + \sum_{i \in J_U(r_z)}}{p_z}$. Now it remains to show that $p_{i^*} \leq p_z$. Assume that $p_{i^*} > p_z$. Since $i^* \in J_B$, we have $r_{i^*} + S(\sigma_{i^*})p_{i^*} \leq r_z + S(\sigma_{i^*})p_z$. This implies that $r_{i^*} - r_z < 0$. But this contradicts the fact that $i^* \in J_U(r_z)$. □

Before we proceed to the analysis of the last case in Lemma 6.5, we define two sets of jobs. Our aim is to relate the set of jobs in WDA and OPT that are executed after r_z. Informally, we first define a set consisting of jobs that were processed during the interval $[r_z, C^*_y)$ in OPT such that for each job, their processing time is at most the processing time of job z.

Definition 6.5. *We define a new set J_S. Let i be some job in J. Then $i \in J_S$ if and only if following conditions are met:*

- $\sigma^*_i \geq r_z$
- $p_i \leq p_z$ *or* $r_i + S^* p_i \leq r_z + S^* p_z$
- $C^*_i < C^*_y$

where S^* denotes the max-stretch *attained by some job in OPT.*

Observe that J_S is a non-empty set as job $z \in J_S$. Now we define the set of big jobs that were processed *consecutively* [1] just before job y.

Lemma 6.4. *If $y \in J_S$, then $S_z \leq S^*(1 + \alpha\Delta)$.*

Proof. Since $y \in J_S$, we have $r_y + S^* p_y \leq r_z + S^* p_z$. Using this in Equation 6.2, we get:

$$\begin{aligned} S_z &\leq S^*_y \left(\frac{p_y}{p_z}\right) + \frac{S^*(p_z - p_y)}{p_z} + \frac{\delta}{p_z} \\ &\leq S^* + \frac{\delta}{p_z} \qquad \leq S^*(1 + \alpha\Delta) \end{aligned}$$

□

1. Here we assume that the optimal schedule is non-lazy, that is all jobs are scheduled at the earliest available time.

For the rest of analysis, we assume that $y \notin J_S$.

Definition 6.6. *We define J_L as the set of jobs in schedule OPT that are executed between the completion time of latest job in set J_S and completion time of job y.*

$$J_L = \{i \in J : \sigma_i^* \in [C_k^*, C_y^*)\}$$

where $k \in J_S$ and $\sigma_k^* \geq \sigma_i^*, \forall i \in J_S$. Moreover, λ and $|J_L|$ denote the length of time interval $[C_k^*, C_y^*)$ and the number of jobs in J_L, respectively.

Note that job y belongs to J_L. Hence $\lambda \geq p_y$ and $\forall i \in J_L$, we have $p_i > p_z$ and $r_z + S^* p_z < r_i + S^* p_i$. We assume that $\sigma_y^* < C_z - \alpha\Delta$. This implies that $\delta < 0$.

Property 8. *If $p_z(1+\alpha\Delta) < p_y \leq \Delta$, then the total processing time of jobs scheduled after r_z in WDA is at least $\lambda - p_y + \alpha\Delta$.*

Proof. Let $J_{U\setminus B}$ is a set of jobs that are released on or after r_z. The set $J_{U\setminus B}$ include z. Then we have $C_z = w + r_z + \sum_{i \in J_{U\setminus B}} p_i + \sum_{i \in J_U(r_z)} p_i$, where w is the remaining processing time of job executing at r_z. Moreover all jobs in set $J_{U\setminus B}$ are scheduled after time r_z in OPT. Therefore $C_y^* \geq r_z + \lambda + J_{U\setminus B}$. Since $\sigma_y^* < C_z - \alpha\Delta$, we have $w + \sum_{i \in J_U(r_z)} p_i \geq \lambda - p_y + \alpha\Delta$. □

For the rest of the analysis, let j denote the first job that starts its execution after time r_z, that is $\sigma_j \leq \sigma_i : \forall i \in J_B$.

Corollary 6.1. $S(\sigma_j) \geq \frac{\lambda - p_y + \alpha\Delta + p_z}{p_z}$

Proof. The result is direct consequence of the Lemma 6.3 and Property 8. □

Now we have all the tools necessary to prove the following lemma.

Lemma 6.5. *If $p_z(1+\alpha\Delta) < p_y \leq \Delta$, then $S_z < S^*(1+\alpha\Delta)$, where S^* is the* max-stretch *of some job in OPT.*

Proof. Let k be the latest job in set J_S. More formally, $k \in J_S$ and $\forall i \in J_S : \sigma_i^* \leq \sigma_k^*$. From Definition 6.6, we have $C_k^* = C_y^* - \lambda$. We can re-write this equality in terms of the stretch of job y and k as $p_y S_y^* = p_k S_k^* + \lambda + r_k - r_y$. Substituting this expression in Equation 6.2, we get:

$$S_z \leq S_k^* \frac{p_k}{p_z} + \frac{r_k - r_z}{p_z} + \frac{\delta + \lambda}{p_z} \tag{6.4}$$

We consider two scenarios.

— **Case A:**
Suppose there exists a job $l \in J_L$ such that $\sigma_l \geq r_z$. From Property 6 it follows that $r_l + S(\sigma_l)p_l < r_z + S(\sigma_l)p_z$. Due to Property 4, we have $r_l + S(\sigma_j)p_l \leq r_z + S(\sigma_j)p_z$. Moreover we also have $r_z + S^* p_z < r_l + S^* p_l$. Combining these together, we get $S(\sigma_j) \leq S^*$. Now we make two simple sub-cases depending upon the release time of job k.

— Suppose $r_k - r_z \leq 0$. Either $p_k \leq p_z$ or $r_k - r_z \leq S^*(p_k - p_z)$. In both case Equation 6.4 can be simplified to:

$$\begin{aligned} S_z &\leq S_k^* + \frac{\delta + \lambda}{p_z} \\ \frac{S_z}{S_k^*} &\leq 1 + \frac{\lambda + \delta}{\lambda - p_y + p_z + \alpha\Delta} \\ &\leq 1 + \frac{\lambda + \alpha\Delta}{\lambda - p_y + p_z + \alpha\Delta} \\ &\leq 1 + \frac{\Delta + \alpha\Delta}{1 + \alpha\Delta} \qquad \leq 1 + \alpha\Delta \end{aligned}$$

— Suppose $r_k > r_z$. From Observation 6.2, we have $p_k < p_z$. Also job k belongs to J_B. From Property 6, we have the $r_k - r_z \leq S(\sigma_k)(p_z - p_k) \leq S_z(p_z - p_k)$. Then Equation 6.4 can be simplified to:

$$\begin{aligned} S_z &\leq S_k^* \frac{p_k}{p_z} + \frac{S_z(p_z - p_k)}{p_z} + \frac{\delta + \lambda}{p_z} \\ &\leq S_k^* + \frac{\delta + \lambda}{p_z} \\ \frac{S_z}{S_k^*} &\leq 1 + \frac{\lambda + \delta}{\lambda - p_y + p_z + \alpha\Delta} \\ &\leq 1 + \frac{\lambda + \alpha\Delta}{\lambda - p_y + p_z + \alpha\Delta} \\ &\leq 1 + \frac{\Delta + \alpha\Delta}{1 + \alpha\Delta} \qquad \leq 1 + \alpha\Delta \end{aligned}$$

— **Case B:**
Assume that $\forall l \in J_L : \sigma_l < r_z$.

— Assume that all job $\forall w \in J_B : \sigma_w^* \geq r_z$. We show by contradiction that there exists a job i such that $\sigma_i < r_z < C_w$. Assume that no such job exists. Then $J_B \subseteq J_S$. But this is in contradiction to fact that $\sigma_y^* < C_z - \alpha\Delta$. Moreover, we can infer that $p_i > p_i(r_z) > \alpha\Delta + p_z$, , where $p_i(r_z)$ denotes the remaining processing time of job i at r_z. Therefore $C_z = p_i(r_z) + \sum_{w \in J_B} p_w + p_z$. This implies that $\sigma_y^* < p_i(r_z) + \sum_{w \in J_B} p_w + p_z - \alpha\Delta$. Note that all jobs in J_B

are executed before job y in OPT. Therefore, $\lambda + \delta \leq p_i(r_z)$. Using this inequality in Equation 6.4, we get

$$S_z \leq S_k^* \frac{p_k}{p_z} + \frac{r_k - r_z}{p_z} + \frac{p_i(r_z)}{p_z}$$

Thus using the relation $S^* > 1 + \alpha$ and arguments mentioned in Case A, our bound holds.

— Now we assume that $\exists w \in J_B : \sigma_w^* < r_z$. Without the loss of generality, we assume that all jobs in J_L start before r_z in WDA *i.e.* $\forall l \in J_L : l \in J_B$. Let v be the smallest job in J_L. Now we partition proof into two sections.
  — Assume $p_v \leq p_w$. Since $\sigma_v < \sigma_w$, it implies that $r_v + S(\sigma_v)p_v \leq r_w + S(\sigma_v)p_w$. Using Property 4, we have $r_v + S(\sigma_j)p_v \leq r_w + S(\sigma_j)p_w \leq r_z + S(\sigma_j)p_z$. As $v \in J_L$, we have $r_z + S^*p_z \leq r_v + S^*p_v$. Combining this with Corollary 6.1, we get $S^* \geq \frac{\lambda - p_y + \alpha\Delta + p_z}{p_z}$. Using this in Equation 6.4, our bound holds.
  — Assume that $p_v > p_w$. Then there must be at least $|J_L|$ jobs in J_B such that their start times are no more than r_z in OPT. Furthermore, there exists a job in J_B such that it is delayed by at least λ time units. We overload the notation w to denote such a job. At σ_v, we have $r_v + S(\sigma_v)p_v \leq r_w + S(\sigma_w)p_w$. In OPT we have $r_w + S^*p_w \leq r_v + S^*p_v$. This implies that $S^* > S(\sigma_x) > \frac{\lambda + p_w}{p_w}$. Now we partition proof into two parts:
    — We assume that $p_w \leq 2p_z$ or $\frac{\lambda + p_w}{p_w} \geq (2|J_l| + 1)$. This implies that $S^* \geq \frac{\lambda + 2p_w}{2p_w}$ or $S^* \geq \frac{\lambda + p_w}{p_w} \geq (2|J_l| + 1)$. Applying this lower bound in Equation 6.4, our bound holds.
    — Assume that $p_w > 2p_z$ and $\frac{\lambda + p_w}{p_w} < (2|J_l| + 1)$. This implies that $p_w > \frac{\lambda}{2|J_L|} > \frac{p_v}{2}$ since v is the smallest job in J_L. Hence $p_w < p_v < 2p_w$. Moreover, $r_z + S^*p_z < r_v + S^*p_v < r_v + 2S^*p_w$. At time σ_v, we have $r_v + S(\sigma_v)p_v \leq r_w + S(\sigma_w)p_w$. Since $p_w > 2p_z > p_z$, we have $r_v < r_w$. Therefore $r_z + S^*p_z < r_w + 2S^*p_w$. On the other hand, in WDA we have $r_w + S(\sigma_w)p_w \leq r_z + S(\sigma_w)p_z$. Combining this both inequality, we get $S^* \geq S(\sigma_w)\frac{p_w - p_z}{2p_w - p_z} \geq S(\sigma_j)\left(\frac{p_w - p_z}{2p_w - p_z}\right)$. Using this inequality and Corollary 6.1 in Equation 6.4, our bound holds.

□

6.6 Concluding remarks

In this chapter, we investigated the non-preemptive problem of minimizing the maximum stretch on a single machine. We consider the scenario where jobs arrive

over time. We showed that no algorithm can achieve a competitive ratio better than the $\left(\frac{\sqrt{5}-1}{2}\Delta\right)$ for the maximum stretch objective. We proposed a new algorithm which delays the execution of large jobs and achieves the optimal competitive ratio for max-stretch objective. As a future work, we would like to explore the waiting time strategy for the more general problem of the weighted flow-time minimization problem.

7

SCHEDULING WITH REJECTION TO MINIMIZE STRETCH AND FLOW TIME

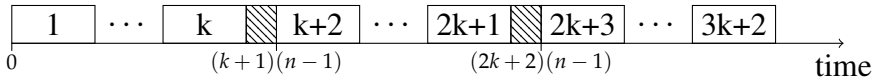
7.1 Introduction

In this chapter, we are interested in the design of efficient algorithms for scheduling set of independent jobs non-preemptively. We consider the offline problem of scheduling a set $\mathcal{J}$ of n jobs on a single machine where each job $j \in \mathcal{J}$ is characterized by its *processing time* p_j and its*release time* r_j. Given a schedule $\mathcal{S}$, $\sigma_j^{\mathcal{S}}$ and $C_j^{\mathcal{S}}$ denote the starting and completion times of j, respectively. The *flow time* of j is defined as the total time that j remains in the system *i.e.* $F_j^{\mathcal{S}} = C_j^{\mathcal{S}} - r_j$. Furthermore, the *stretch* of j in $\mathcal{S}$ is defined as $s_j^{\mathcal{S}} = \frac{F_j^{\mathcal{S}}}{p_j}$ i.e. the flow time of j is normalized with respect to its processing time. When there is no ambiguity, we will simplify the above notation by dropping $\mathcal{S}$. Our objective is to create a *non-preemptive* schedule that minimizes the average stretch of jobs in $\mathcal{J}$, i.e., $\sum_{j\in\mathcal{J}} s_j$.

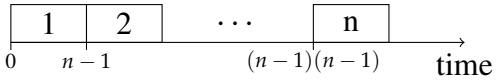
As aforementioned, the average stretch minimization problem is a special case of the *average weighted flow-time minimization* problem where each job $j \in \mathcal{J}$ is additionally characterized by a weight w_j and the objective is to minimize $\sum_{j\in\mathcal{J}} w_j F_j$. Another closely related problem is the *average flow-time minimization*. This is also a special case of the average weighted flow-time minimization problem where the weights of all jobs are equal. Although average flow-time and average stretch objectives do not have an immediate relation, the latter is generally considered to be a more difficult problem since w_j depends on the job's processing time and therefore pays a huge penalty when a large job is scheduled before a small job. Here, we study the average stretch and average flow minimization problems with respect to the rejection model.

7.2 Speed augmentation v/s Rejection model

In this chapter, we explore the relation between preemptive and non-preemptive schedules with respect both objectives subject to the rejection model. More specifically, we consider the SRPT policy for creating a preemptive schedule. Observe that the relation among the rejection model and other resource augmentation models is not clear. For example, in Fig. 7.1 we give an instance for which the best possible solution in the rejection model is worse than the best possible solution in the speed-augmentation model, when the same constant ϵ is selected for both models.



Scheduling using rejections (reject the jobs $k+1$ and $2k+2$)



Scheduling using speed augmentation (augmented processing times are equal to $n-1$)

Figure 7.1. – An instance of $n = 3k+2$ jobs with equal processing times $p_j = n$, equal weights $w_j = 1$, and release dates $r_j = (j-1)(n-1)$, where $1 \leq j \leq n$. By setting $\epsilon = \frac{1}{n-1}$, in the rejection model we are allowed to reject at most $\epsilon n \leq 2$ jobs, while in the speed augmentation model the processing time of each job becomes $\frac{p_j}{1+\epsilon} = n-1$. The sum flow time using rejections is $3\sum_{j=1}^{k}(n+j-1) = \frac{21}{2}k^2 + \frac{9}{2}k$, while the sum flow time using speed augmentation is $n(n-1) = 9k^2 + 9k + 2$ which is better for large enough k.

However, our results in this chapter show the strength of the rejection model, particularly in the non-preemptive context. Note that previous algorithms for solving the non-preemptive average flow time minimization problem either required quasi-polynomial time [ILMT15] or cannot achieve performance guarantee arbitrarily close to the optimum [BCK+07, PSTW97]. Before continuing, we give some additional notation which we use throughout the chapter.

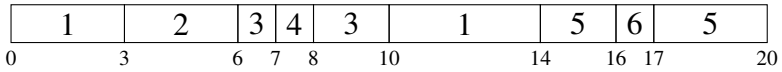
Notations. In what follows, for each job $j \in \mathcal{J}$ and schedule $\mathcal{S}$, we define the interval $[\sigma_j^{\mathcal{S}}, C_j^{\mathcal{S}}]$ to be the *active interval* of j in $\mathcal{S}$. In the case where preemptions are allowed, the active interval of j may have a length bigger than p_j. A job j is *available* at a time t if it is released but it is not yet completed, i.e., $r_j \leq t < C_j$. We call a schedule *compact* if it does not leave any idle time whenever there is a job available for execution.

7.3 Structure and Properties of SRPT and an Intermediate Schedule

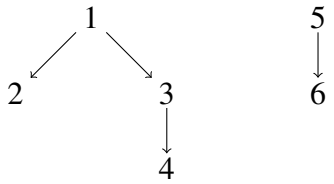
In this section we deal with the structure of a preemptive schedule created by the classical Shortest Remaining Processing Time (SRPT) policy and we give some useful properties that we will use in the following sections.

According to the SRPT policy, at any time, we schedule the unfinished job with the shortest remaining processing time. Since the remaining processing time of the executed job $j \in \mathcal{J}$ decreases over time, its execution may be interrupted only in the case where a new job $k \in \mathcal{J}$ is released and the processing time of k is smaller than the remaining processing time of j at r_k. Hence, the SRPT policy can be seen as an

jobs	1	2	3	4	5	6
r_j	0	3	6	7	14	16
p_j	7	3	3	1	5	1



Shortest Remaining Processing Time (SRPT) schedule



Collection of out-trees

Figure 7.2. – A schedule created by the SRPT policy and its corresponding collection of out-trees.

event-driven algorithm where at each time t when a job is released or completed, it schedules an unfinished jobs with the shortest remaining processing time. In case of ties, we assume without the loss of generality that SRPT resumes the partially executed job, if any, with the latest starting time; if all candidate jobs are not processed before, then it chooses among them the job with the earliest release time.

Kellerer *et al.* [KTW95] observed that in the schedule produced by the SRPT policy, for any two jobs j and k, their active intervals are either completely disjoint or the one contains the other. Moreover, there is no idle time during the active interval of any job. Based on the above observation, the execution of the jobs in the SRPT schedule has a tree-like structure. More specifically, we can create a graph which consists of a collection $\mathcal{T}$ of out-trees and corresponds to the SRPT schedule as follows (see Fig. 7.2): for each job $j \in \mathcal{J}$, we create a vertex u_j. For each pair of jobs $j,k \in \mathcal{J}$, we add an arc (u_j,u_k) if and only if $[\sigma_k,C_k] \subset [\sigma_j,C_j]$ and there is no other job $i \in \mathcal{J}$ so that $[\sigma_k,C_k] \subset [\sigma_i,C_i] \subset [\sigma_j,C_j]$.

In what follows, we denote by $root(T)$ the root of each out-tree $T \in \mathcal{T}$. Intuitively, each vertex $root(T)$ corresponds to a job for which at any time t during its execution there is no other job which has been partially executed at t. We denote also by $a(j)$ the parent of the vertex that corresponds to the job $j \in \mathcal{J}$ in T. Moreover, let $T(u_j)$ be the subtree of $T \in \mathcal{T}$ rooted at a vertex u_j in T. Note that, we may refer to a job j by its corresponding vertex u_j and vice versa.

In this chapter, we use the schedule created by the SRPT policy for the preemptive variant of our problem as a lower bound to the non-preemptive variant. The SRPT

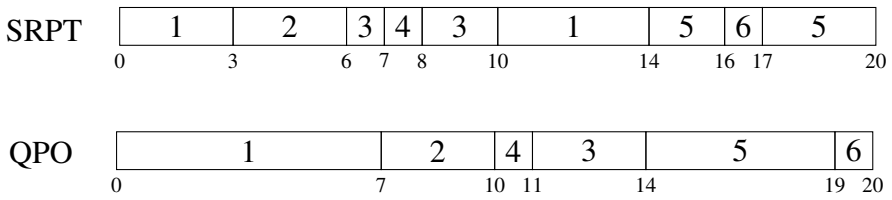


Figure 7.3. – Transformation from SRPT to QPO schedule

policy is known to be optimal [LR07] for the problem of minimizing $\sum_{j\in\mathcal{J}} F_j$ when preemptions of jobs are allowed. However, for the preemptive variant of the average stretch minimization problem, SRPT is a 2-approximation algorithm [MRSG99].

Consider now the collection of out-trees $\mathcal{T}$ obtained by an SRPT schedule and let $T(u_j)$ be the subtree rooted at any vertex u_j. We construct a non-preemptive schedule for the jobs in $T(u_j)$ as follows: during the interval $[\sigma_j, C_j]$, we run the jobs in $T(u_j)$ starting with j and then running the remaining jobs in order of increasing SRPT completion time as shown in Figure 7.3. This policy has been proposed in [Bun04] for the problem of minimizing the sum $\sum_{j\in\mathcal{J}} F_j$ and corresponds to a post order transversal of the subtree $T(u_j)$ excluding its root which is scheduled in the first position. We call the above policy as *Quasi Post Order* (QPO) and we will use it for the problem of minimizing $\sum_{j\in\mathcal{J}} s_j$. The following lemma presents several observations for the QPO policy.

Lemma 7.1. [Bun04] *Consider any subtree $T(u_k)$ which corresponds to a part of the schedule $\mathcal{SRPT}$ and let $\mathcal{QPO}$ be the non-preemptive schedule for the jobs on $T(u_k)$ created by applying the Quasi Post Order policy. Then,*

1. *all jobs in $\mathcal{QPO}$ are executed during the interval $[\sigma_k^{\mathcal{SRPT}}, C_k^{\mathcal{SRPT}}]$ without any idle period,*
2. *$\sigma_j^{\mathcal{QPO}} \geq r_j$ for each u_k in $T(u_k)$,*
3. *$C_j^{\mathcal{QPO}} \leq C_j^{\mathcal{SRPT}} + p_k$ for each u_j in $T(u_k)$ with $j \neq k$, and*
4. *$C_k^{\mathcal{QPO}} = C_k^{\mathcal{SRPT}} - \sum_{u_j\in T(u_k):j\neq k} p_j$.*

Note that the schedule created by the SRPT policy is a compact schedule, since it always execute a job if there is an available one. Therefore, by Lemma 7.1, the following directly holds.

Corollary 7.1. *The schedule created by the QPO policy is compact.*

7.4 The Rejection Model

In this section we consider the rejection model. More specifically, given an $\epsilon \in (0,1)$, we are allowed to reject any subset of jobs $\mathcal{R} \subset \mathcal{J}$ whose total weight does not exceed

an ϵ-fraction of the total weight of all jobs, i.e., $\sum_{j\in\mathcal{R}} w_j \leq \epsilon \sum_{j\in\mathcal{J}} w_j$. We will present our rejection policy for the more general problem of minimizing $\sum_{j\in\mathcal{J}} w_j F_j$.

Our algorithm is based on the tree-like structure of the SRPT schedule. Let us focus first on a single out-tree $T \in \mathcal{T}$. The main idea is to reject the jobs that appear in the higher levels of T (starting with its root) and run the remaining jobs using the QPO policy. The rejected jobs are, in general, long jobs which are preempted several times in the SRPT schedule and their flow time can be used as an upper bound for the flow time of the smaller jobs that are released and completed during the life interval of the longest jobs. In order to formalize this, for each job $j \in \mathcal{J}$ we introduce a charging variable x_j. In this variable we accumulate the weight of jobs whose flow time will be upper bounded by the flow time of job j in the SRPT schedule. At the end of the algorithm, this variable will be exactly equal to $\frac{1}{\epsilon}w_j$ for each rejected job $j \in \mathcal{R}$, while $x_j < \frac{1}{\epsilon}w_j$ for each non-rejected job $j \in \mathcal{J} \setminus \mathcal{R}$. In fact, for most of the non-rejected jobs this variable will be equal to zero at the end of the algorithm. Our algorithm considers the jobs in a bottom-up way and charges the weight of the current job to its ancestors in T which are closer to the root and their charging variable is not yet full; that is the vertices to be charged are selected in a top-down way. Note that, we may charge parts of the weight of a job to more than one of its ancestors.

Algorithm 7.1 describes formally the above procedure. For notational convenience, we consider a fictive vertex u_0 which corresponds to a fictive job with $w_0 = 0$. We connect u_0 with the vertex $root(T)$ of each out-tree $T \in \mathcal{T}$ in such a way that u_0 becomes the parent of all of them. Let T^* be the created tree with root u_0.

Algorithm 7.1 Non-preemptive Schedule for the jobs in $\mathcal{J} \setminus \mathcal{R}$

Create a preemptive schedule $\mathcal{SRPT}$ and the corresponding out-tree T^*
Initialization: $\mathcal{R} \leftarrow \emptyset$, $x_j \leftarrow w_j$ for each $j \in \mathcal{J}$, $x_0 \leftarrow 0$
for each vertex u_j of T^* with $x_j = w_j$ in post-order traversal **do**
  while $x_j \neq 0$ **and** $x_{a(j)} < \frac{1}{\epsilon}w_{a(j)}$ **do**
    Let u_k be a vertex in the path between u_0 and u_j such that $x_{a(k)} = \frac{1}{\epsilon}w_{a(k)}$ and $x_k < \frac{1}{\epsilon}w_k$
    Let $y \leftarrow \min\{x_j, \frac{1}{\epsilon}w_k - x_k\}$
    $x_j \leftarrow x_j - y$ and $x_k \leftarrow x_k + y$
  end while
end for
for each job $j \in \mathcal{J}$ **do**
  if $x_j = \frac{1}{\epsilon}w_j$ **then**
    Reject j, i.e., $\mathcal{R} \leftarrow \mathcal{R} \cup \{j\}$
  end if
end for
return S: the non-preemptive schedule for the jobs in $\mathcal{J} \setminus \mathcal{R}$ using QPO

Note that, the for-loop in Lines 3-7 of Algorithm 7.1 is not executed for all jobs. In fact, it is not applied to the jobs that will be rejected as well as to some children of it for which at the end of the algorithm it holds that $w_j < x_j < \frac{1}{\epsilon} w_j$. The weight of these jobs is charged to itself. Moreover, the while-loop in Lines 4-7 of Algorithm 7.1 terminates either if the whole weight of j is charged to its ancestors or if the parent of u_j is already fully charged, i.e., $x_{a(j)} = \frac{1}{\epsilon} w_{a(j)}$.

Theorem 7.1. *For the schedule $\mathcal{S}$ created by Algorithm 7.1 it holds that*

(i) $\sum_{j \in \mathcal{J} \setminus \mathcal{R}} w_j F_j^{\mathcal{S}} \leq \frac{1}{\epsilon} \sum_{j \in \mathcal{J}} w_j F_j^{\mathcal{SRPT}}$, *and*

(ii) $\sum_{j \in \mathcal{R}} w_j \leq \epsilon \sum_{j \in \mathcal{J}} w_j$.

Proof. Consider first any vertex u_k such that $k \in \mathcal{J} \setminus \mathcal{R}$ and $a(k) \in \mathcal{R}$. By the execution of the algorithm, all the jobs corresponding to vertices in the path from u_0 to $a(k)$ do not appear in $\mathcal{S}$. Hence, k starts in $\mathcal{S}$ at the same time as in $\mathcal{SRPT}$, i.e., $\sigma_k^{\mathcal{S}} = \sigma_k^{\mathcal{SRPT}}$. Thus, by Lemma 7.1, the jobs that correspond to the vertices of the subtree $T^*(u_k)$ are scheduled in $\mathcal{S}$ during the interval $[\sigma_k^{\mathcal{SRPT}}, C_k^{\mathcal{SRPT}}]$. In other words, for any job j in $T^*(u_k)$ it holds that $C_j^{\mathcal{S}} \leq C_k^{\mathcal{SRPT}}$, while by the construction of T^* we have that $\sigma_k^{\mathcal{SRPT}} < r_j$. Assume now that the weight of j is charged by Algorithm 7.1 to the jobs $j_1, j_2, \ldots, j_{q_j}$, where q_j is the number of these jobs. Let w_j^i be the weight of j charged to $j_i \in \{j_1, j_2, \ldots, j_{q_j}\}$; note that $w_j = \sum_{i=1}^{q_j} w_j^i$. By the definition of the algorithm, each $j_i \in \{j_1, j_2, \ldots, j_{q_j}\}$ is an ancestor of both k and j in T^* (one of them may coincides with k). Therefore, by the definition of T^*, it holds that $\sigma_{j_i}^{\mathcal{SRPT}} < r_j < C_j^{\mathcal{S}} \leq C_{j_i}^{\mathcal{SRPT}}$, for each $j_i \in \{j_1, j_2, \ldots, j_{q_j}\}$. Then, we have

$$\sum_{j \in \mathcal{J} \setminus \mathcal{R}} w_j F_j^{\mathcal{S}} \leq \sum_{j \in \mathcal{J} \setminus \mathcal{R}} \sum_{i=1}^{q_j} w_j^i F_{j_i}^{\mathcal{SRPT}} \leq \sum_{j \in \mathcal{J}} x_j F_j^{\mathcal{SRPT}} \leq \sum_{j \in \mathcal{J}} \frac{1}{\epsilon} w_j F_j^{\mathcal{SRPT}}$$

where the second inequality holds by regrouping the flow time of all appearances of the same job, and the last one by the fact that Algorithm 7.1 charges at each job j at most $(1 + \frac{1}{\epsilon}) w_j$. Finally, since the weight of each job is charged exactly once (probably to more than one other jobs) we have $\sum_{j \in \mathcal{J}} w_j \geq \frac{1}{\epsilon} \sum_{j \in \mathcal{R}} w_j$ and the theorem holds. □

Since SRPT creates an optimal preemptive schedule for the problem of minimizing $\sum_{j \in \mathcal{J}} F_j$ on a single machine and an optimal preemptive schedule is a lower bound for a non-preemptive one the following theorem holds.

Theorem 7.2. *Algorithm 7.1 is a $\frac{1}{\epsilon}$-approximation algorithm for the single-machine average flow-time minimization problem without preemptions if we are allowed to reject an ϵ-fraction of the jobs.*

By combining Theorem 7.1 and the fact that SRPT is a 2-approximation algorithm for the preemptive variant of the average stretch minimization problem [MRSG99], the following theorem holds.

Theorem 7.3. *Algorithm 7.1 is a $\frac{2}{\epsilon}$-approximation algorithm for the single-machine average stretch minimization problem without preemptions if we are allowed to reject a set of jobs whose total weight is no more than an ϵ-fraction of the total weight of all jobs.*

7.5 Conclusion

We studied the effects of applying resource augmentation in the transformation of a preemptive schedule to a non-preemptive one for the problem of minimizing total stretch on a single machine. Specifically, we show the power of the rejection model for scheduling without preemptions comparing with other resource augmentation models, by presenting an algorithm which has a performance arbitrarily close to optimal. So, an interesting question is to explore the general idea of transforming preemptive to non-preemptive schedules subject to the rejection model on parallel machines based on the above results.

8

ONLINE SCHEDULING TO MINIMIZE WEIGHTED FLOW TIME ON UNRELATED MACHINES

8.1 Introduction

In this chapter, we are interested in studying speed augmentation and rejection model, that compares online algorithms to a weaker adversary. The power of these models lies in the fact that many natural scheduling algorithms can be analyzed with respect to them. Furthermore, they have successfully provided theoretical evidences for heuristic scheduling algorithms with good performance in practice. Although these models give more power to online algorithms, the connection especially between the rejection model and the speed augmentation is unclear. This disconnection is emphasized by the fact that some algorithms have good performance in a model but have moderate behaviour in others. For instance, it has been shown that no algorithm can be constant competitive with respect to the speed augmentation for the problem of minimizing maximum flow time on the restricted machine settings, whereas in the rejection model, this problem is known to be scalable competitive [CDK15].

8.1.1 Mathematical Programming and Duality

Our approach for the systematic study of algorithms, is based on the principle of mathematical duality. This techniques have been extensively applied for the design of approximation algorithms [WS11] and online algorithms [BN09]. Specifically, Buchbinder *et al.* [BN09] gave a generalized framework for online covering/packing LPs that applies to several fundamental problems in the domain of online computation. However, this framework encounters several issues while designing competitive algorithms for online scheduling problems. Recently, Anand *et al.* [AGK12] have proposed the use of dual-fitting techniques to study scheduling problems in the speed augmentation model. After this seminal paper, the duality approaches in online scheduling have been extended to a variety of problems, and has rapidly become standard techniques. The duality approaches have also led to the development of newer techniques for analyzing algorithm (see for example [DH14, GKP13, IKM14, IKMP14, Ngu13]).

To see that the duality is particularly appropriate, we first explain the model and the approach intuitively. The weak duality in mathematical programming can be interpreted as a game between an algorithm and an adversary (the primal program against the dual one). The game is $L(x, \lambda)$, the standard Lagrangian function completely defined for a given problem, in which x and λ are primal and dual variables, respectively. The

primal and dual variables are controlled and correspond to the strategies of the adversary and the algorithm, respectively. The goal of the algorithm is to choose a strategy λ among its feasible sets so as to minimize $L(x, \lambda)$ for whatever feasible strategy x of the adversary.

8.1.2 Generalized Resource Augmentation

The resource augmentation models proposed in [KP95, PSTW97] consist in giving more power to the algorithm. This idea could be perfectly interpreted as a game between an algorithm and an adversary in which additional power for the algorithm is reflected by better choices over its feasible strategy set. Concretely, let us illustrate this idea for the speed augmentation and the rejection models. In several scheduling problems, a constraint originally states that the speed of a given machine is at most one. In the speed augmentation model, this constraint is relaxed such that the algorithm executes jobs at a slightly higher speed than that of the adversary. In the approach proposed in [AGK12], the key step relies on the construction of a dual feasible solution in such a way that its dual objective is up to some bounded factor from that of the algorithm. Then, the dual variables are carefully designed in order to encode the power of speed augmentation. Later on, Nguyen [Ngu13] explicitly formalized the comparison through the mean of Lagrangian functions between the algorithm and the adversary, with a tighter feasible domain due to speed augmentation. That point of view makes the framework in [AGK12] effective to study non-convex formulations. On the other hand, the relaxation is of a different nature in the rejection model. Specifically, there are usually constraints ensuring that all jobs should be completed. In the rejection model, the algorithm is allowed to systematically reject a fraction of constraints whereas the adversary should satisfy all of them.

In both models, the algorithm optimizes the objective over a feasible domain whereas the adversary optimizes the same objective over a sub-domain with respect to the algorithm. This naturally leads to a more general model of resource augmentation.

Definition 8.1 (**Generalized Resource Augmentation**). *Consider an optimization problem that can be formalized by a mathematical program. Let $\mathcal{P}$ be the set of feasible solutions of the program and let $\mathcal{Q}$ be a subset of $\mathcal{P}$. In* generalized resource augmentation*, the performance of an algorithm is measured by the worst ratio between its objective over $\mathcal{P}$ and that of a solution which is optimized over $\mathcal{Q}$.*

Based on the above definition, the polytope of the adversary in speed augmentation is a strict subset of the algorithm's polytope since the speed constraint for the adversary is tighter. In the rejection model, the polytope of the adversary is also a strict subset of the algorithm's one since it contains more constraints. In addition, the generalized model allows us to introduce different kind of relaxations to the set of feasible solutions – each corresponding to different type of models.

Together with the generalized model, we consider the following duality-based approach for the systematic design and analysis of algorithms. Let $\mathcal{P}$ and $\mathcal{Q}$ be the sets of feasible solutions for the algorithm and the adversary, respectively. Note that $\mathcal{Q} \subset \mathcal{P}$ for the both, resource augmentation and rejection model. In order to study the performance of an algorithm, we consider the dual of the mathematical program consisting of the objective function optimized over $\mathcal{Q}$. By weak duality, the dual is a lower bound for any solution. We bound the algorithm's cost by that of this dual. We exploit the model properties (relation between $\mathcal{P}$ and $\mathcal{Q}$) to derive effective bounds. Intuitively, one needs to take the advantage from these models so as to raise the dual as much as possible — an impossible procedure otherwise. As it has been shown in previous works, the duality approach is particularly appropriate to study problems with resource augmentation [GKP13, IKM14, AGK12, Ngu13].

8.1.3 Our approach

We apply the generalized model and the duality-based approach to scheduling problems, in which jobs arrive online and they have to be scheduled non-preemptively on a set of unrelated machines. The objective is to minimize the weighted average flow time of the jobs.

In this chapter, we present scheduling algorithms in a model which combines speed augmentation and the rejection model. The design and analysis of the algorithms follow the duality approach. At the release time of any job, the algorithm defines the dual variables associated with the job and assigns the job to some machine based on this definition. The value of the dual variables associated with a job are selected in order to satisfy two key properties:

1. comprise the marginal increase of the total weighted flow-time due to the arrival of the job — the property that has been observed [AGK12, Ngu13] and has become more and more popular in dual-fitting for online scheduling;
2. capture the information for a future decision of the algorithm whether this job will be completed or rejected — a *novel* point in the construction of dual variables to exploit the power of rejection.

Informally, to fulfill the second property, we introduce *prediction* terms to dual variables that at some point in the future will indicate whether the corresponding job would be rejected. Moreover, these terms are chosen so as to stabilize the schedule such that the properties of the assignment policy are always preserved (even with job rejections in the future). This allows us to maintain a non-migratory schedule.

Our algorithm dispatches jobs immediately at their release time — a desired property in scheduling. Besides, the algorithm processes jobs in the highest density first manner and interrupts a job only if it is rejected. In other words, no completed job has been interrupted during its execution. The algorithm is relatively simple, particularly for

a single machine setting as there is no assignment policy. Therefore, the analysis of the algorithm in the generalized resource augmentation could be considered as a first step toward the theoretical explanation for the well-known observation that simple scheduling algorithms usually behave well and are widely used in practice.

8.2 Problem Definition and Notations

We are given a set $\mathcal{M}$ of m unrelated machines. Let $\mathcal{J}$ denote the set of all jobs of our instance, which is not known a priori. Each job $j \in \mathcal{J}$ is characterized by its *release time* r_j, its *weight* w_j and if job j is executed on machine $i \in \mathcal{M}$ then it has a *processing time* p_{ij}. We study the *non-preemptive* setting. In this chapter, we consider a stronger non-preemptive model according to which we are only allowed to interrupt a job if we reject it, *i.e.*, we do not permit restarts. Moreover, each job has to be dispatched to one machine at its arrival and migration is not allowed.

Given a schedule $\mathcal{S}$, let C_j and F_j denote the completion time and flow time of the job j. Our objective is to create a non-preemptive schedule that minimizes the weighted average of flow-times of all jobs, i.e., $\sum_{j\in\mathcal{J}} w_j F_j$.

Let $\delta_{ij} = \frac{w_j}{p_{ij}}$ be the *density* of a job j on machine i. Moreover, let $q_{ij}(t)$ be the remaining processing time at time t of a job j which is dispatched at machine i. A job j is called *pending* at time t, if it is already released at t but not yet completed, i.e., $r_j \leq t < C_j$. Finally, let $P = \max_{j,j'\in\mathcal{J}}\{p_j/p_{j'}\}$ and $W = \max_{j,j'\in\mathcal{J}}\{w_j/w_{j'}\}$.

8.3 Lower Bound

Lemma 8.1. *For any speed augmentation $s \leq P^{1/10}$ (resp., $s < W^{1/6}$), every deterministic algorithm has competitive ratio of $\Omega(P^{1/10})$ (resp., $\Omega(W^{1/6})$) for the problem of minimizing weighted average flow time on a single machine problem.*

Proof. Let $s > 1$ be the speed of the machine; without loss of generality, we assume that the machine speed for the adversary is 1. Let $R > s^2$ be an arbitrary (large) constant. Let A be some fixed scheduling algorithm.

We consider the following instance. At time 1, a *long* job of processing time $2sR^3$ and weight 1 is released. A *short* job of processing time 1 and weight R is released at time R^3. In the phase 1 of the instance, one short job is released at every time interval aR^3 for all $2 \leq a \leq 2s-1$, if the algorithm does not processes the long jobs during the entire interval $[(a-1)R^3, aR^3]$, otherwise the adversary stops and instance halts. At time $2sR^3$, the phase 2 of the instance starts where new *small* job of processing time 1 and weight R^2 are released at every time interval aR^3 for all $2s \leq a \leq 2sR^2$ if the algorithm does not processes the long jobs during the entire interval $[(a-1)R^3, aR^3]$, otherwise the adversary stops and instance halts.

In the instance, we have at most $2s$ short jobs and $2sR^2$ small jobs. Observe that by using speed s, the algorithm cannot complete the long job between two consecutive release times of short or small jobs. We analyze the performance of the algorithm by considering different cases.

— **Case 1: the instance halts during phase 1.** In this case, there is a $a \in \{1, 2, \ldots, 2s-1\}$ for which the algorithm keeps processing the long job during the whole interval $[aR^3, (a+1)R^3]$ and hence the short job released at aR^3 is not processed during that time interval. Hence the short job's flow time is at least R^3. Therefore, the weighted flow-time of the short job is at least $R \cdot R^3$. However, the adversary can execute immediately all short jobs at their release times and process the long job in the end. The sum weighted flow-time of all short jobs is at most $2sR$. The long job would be started no later than the time where phase 1 terminates, which is $(2s-1)R^3 + 1$. So the weighted flow-time of the long job is at most $4sR^3$. Therefore, the competitive ratio is at least $\Omega(R/s)$.

— **Case 2: the instance halts during phase 2.** In this case, there is a $a \in \{2s, 2s+1, \ldots, 2sR^2\}$ for which the algorithm keeps processing the long job during the whole interval $[aR^3, (a+1)R^3]$ and hence the small job released at aR^3 is not processed during that time interval. We proceed similarly as in the previous case. The weighted flow-time of this small job is at least $R^2 \cdot R^3 = R^5$. Nevertheless, the adversary can process the long job during $[1, 2sR^3 + 1]$, execute small jobs at their release time (except the first one which starts 1 unit of time after its release time) and execute all short jobs during the interval $[2sR^3 + 2, 5sR^3]$ whenever a small job is not executed. This is a feasible schedule since the number of short jobs is $(2s-1) < 3R^3 - 5$ (note that there are 2 small jobs released during $[2sR^3 + 2, 5sR^3]$). By this strategy, the weighted flow-time of the long job is $2sR^3 + 1 < 2s^2R^4$. The total weighted flow-time of small jobs is at most $2sR^2 \cdot R^2 < 2s^2R^4$. The total weighted flow-time of short jobs is at most $2s \cdot R \cdot 5sR^3 = 10s^2R^4$. Hence, the cost of the adversary is at most $14s^2R^4$ and the competitive ratio is at least $\frac{R^5}{14s^2R^4} = \Omega(R/s^2)$.

— **Case 3: the instance halts at the end of phase 2.** The algorithm executes the long job after the end of phase 2 and hence this job is completed at later than $2sR^5$; so its weighted flow-time is at least $2sR^5$. The adversary can apply the same strategy as in Case 2 with total cost $14s^2R^4$. Therefore, the competitive ratio is at least $\Omega(R/s)$.

In summary, the competitive ratio is at least $\Omega(R/s^2)$. Recall that P and W are the largest ratio between processing times and that between weights, respectively. In this instance, $P = 2sR^3$ and $W = R^2$ respectively. By a simple estimation (setting $R = s^3$), for any speed $s \leq P^{1/10}$ the competitive ratio is at least $\Omega(P^{1/10})$; and for $s \leq W^{1/6}$, the competitive ratio is at least $\Omega(W^{1/6})$. □

8.4 Scheduling to Minimize Average Weighted flow time

In this section, we describe our primal-dual method for the online non-preemptive scheduling problem of minimizing the total weighted flow-time on unrelated machines. This problem admits no competitive algorithm even with speed augmentation as shown by the Lemma 8.1. In the following, we study the problem in the generalized resource augmentation model with speed augmentation and rejection.

8.4.1 Linear Programming Formulation

For each machine $i \in \mathcal{M}$, job $j \in \mathcal{J}$ and time $t \geq r_j$, we introduce a binary variable $x_{ij}(t)$ which indicates if j is processed on i at time t. We consider the following linear programming formulation. Note that the objective value of this linear program is at most twice that of the optimal preemptive schedule.

$$\min \sum_{i \in \mathcal{M}} \sum_{j \in \mathcal{J}} \int_{r_j}^{\infty} \delta_{ij} \left(t - r_j + p_{ij}\right) x_{ij}(t) dt$$

$$\sum_{i \in \mathcal{M}} \int_{r_j}^{\infty} \frac{x_{ij}(t)}{p_{ij}} dt \geq 1 \quad \forall j \in \mathcal{J} \tag{8.1}$$

$$\sum_{j \in \mathcal{J}} x_{ij}(t) \leq 1 \quad \forall i \in \mathcal{M}, t \tag{8.2}$$

$$x_{ij}(t) \in \{0,1\} \quad \forall i \in \mathcal{M}, j \in \mathcal{J}, t \geq r_j$$

After relaxing the integrality constraints, we get the following dual program.

$$\max \sum_{j \in \mathcal{J}} \lambda_j - \sum_{i \in \mathcal{M}} \int_0^{\infty} \gamma_i(t) dt$$

$$\frac{\lambda_j}{p_{ij}} - \gamma_i(t) \leq \delta_{ij} \left(t - r_j + p_{ij}\right) \quad \forall i \in \mathcal{M}, j \in \mathcal{J}, t \geq r_j \tag{8.3}$$

We will interpret the resource augmentation models in the above primal and dual programs as follows. In the speed augmentation model, we assume that all machines in the schedule of our algorithm run with speed 1, while in adversary's schedule they run at a speed $a < 1$. This can be interpreted in the primal linear program by modifying the constraint (8.2) to be $\sum_{j \in \mathcal{J}} x_{ij}(t) \leq a$. Intuitively, each machine in the adversary's schedule can execute jobs with speed at most a at each time t. The above modification in the primal program reflects to the objective of the dual program which becomes $\sum_{j \in \mathcal{J}} \lambda_j - a \sum_{i \in \mathcal{M}} \int_0^{\infty} \gamma_i(t) dt$. In the rejection model, we assume that the algorithm is allowed to reject some jobs. This can be interpreted in the primal linear program by summing up only on the set of the non rejected jobs, *i.e.*, the algorithm does not

have to satisfy the constraint (8.1) for rejected jobs. Hence the objective becomes $\sum_{i\in\mathcal{M}}\sum_{j\in\mathcal{J}\setminus\mathcal{R}}\int_{r_j}^{\infty}\delta_{ij}\left(t-r_j+p_{ij}\right)dt$. Concluding, our algorithm rejects a set $\mathcal{R}$ of jobs, uses machines with speed $1/a$ times faster than that of the adversary and, by using weak duality, has a competitive ratio at most

$$\frac{\sum_{i\in\mathcal{M}}\sum_{j\in\mathcal{J}\setminus\mathcal{R}}\int_{r_j}^{\infty}\delta_{ij}(t-r_j+p_{ij})dt}{\sum_{j\in\mathcal{J}}\lambda_j-a\sum_{i\in\mathcal{M}}\int_0^{\infty}\gamma_i(t)dt}.$$

8.4.2 Algorithm and Dual Variables

We describe next the scheduling, the rejection and the dispatching policies of our algorithm which we denote by $\mathcal{A}$. In parallel, we give the intuition about the definition of the dual variables in a primal-dual way. Let $\epsilon_s > 0$ and $0 < \epsilon_r < 1$ be constants arbitrarily small. Intuitively, ϵ_s and ϵ_r stand for the speed augmentation and the rejection fraction of our algorithm, respectively. In what follows, we assume that in the schedule created by $\mathcal{A}$ all machines run with speed 1, while in the adversary's schedule they run by speed $\frac{1}{1+\epsilon_s}$.

Each job is immediately dispatched to a machine upon its arrival. We denote by $Q_i(t)$ the set of pending jobs at time t dispatched to machine $i\in\mathcal{M}$, *i.e.*, the set of jobs dispatched to i that have been released but not yet completed and have not been rejected at t. Our *scheduling policy* for each machine $i\in\mathcal{M}$ is the following: at each time t when the machine i becomes idle or has just completed or interrupted some job, we start executing on i the job $j\in Q_i(t)$ such that j has the largest density in $Q_i(t)$, *i.e.*, $j=\operatorname{argmax}_{j'\in Q_i(t)}\{\delta_{ij'}\}$. In case of ties, we select the job that arrived earliest.

When a machine $i\in\mathcal{M}$ starts executing a job $k\in\mathcal{J}$, we introduce a counter v_k (associated to job k) which is initialized to 0. Each time when a job $j\in\mathcal{J}$ with $\delta_{ij}>\delta_{ik}$ is released during the execution of k and j is dispatched to i, we increase v_k by w_j. Then, the *rejection policy* is the following: we interrupt the execution of the job k and we reject it the first time where $v_k > \frac{w_k}{\epsilon_r}$.

Let Δ_{ij} be the increase in the total weighted flow-time occurred in the schedule of our algorithm if we assign a new job $j\in\mathcal{J}$ to machine i, following the above scheduling and rejection policies. Assuming that the job $k\in\mathcal{J}$ is executed on i at time r_j, we have that

$$\Delta_{ij}=\begin{cases} w_j\left(q_{ik}(r_j)+\sum\limits_{\substack{\ell\in Q_i(r_j)\setminus\{k\}:\\ \delta_{i\ell}\geq\delta_{ij}}}p_{i\ell}\right)+p_{ij}\sum\limits_{\substack{\ell\in Q_i(r_j)\setminus\{k\}:\\ \delta_{i\ell}<\delta_{ij}}}w_\ell & \text{if } v_k+w_j\leq\frac{w_k}{\epsilon_r},\\ w_j\sum\limits_{\substack{\ell\in Q_i(r_j):\\ \delta_{i\ell}\geq\delta_{ij}}}p_{i\ell}+\left(p_{ij}\sum\limits_{\substack{\ell\in Q_i(r_j):\\ \delta_{i\ell}<\delta_{ij}}}w_\ell-q_{ik}(r_j)\sum\limits_{\substack{\ell\in Q_i(r_j)\cup\{k\}:\\ \ell\neq j}}w_\ell\right) & \text{otherwise.}\end{cases}$$

where, in both cases, the first term corresponds to the weighted flow-time of the job j if it is dispatched to i and the second term corresponds to the change of the weighted flow-time for the jobs in $Q_i(r_j)$. Note that, the second case corresponds to the rejection of k and hence we remove the term $w_j q_{ik}(r_j)$ in the weighted flow-time of j, while the flow-time of each pending job is reduced by $q_{ik}(r_j)$.

In the definition of the dual variables, we aim to charge to job j the increase Δ_{ij} in the total weighted flow-time occurred by the dispatching of j in machine i, except from the quantity $w_j q_{ik}(r_j)$ which will be charged to job k, if $\delta_{ij} > \delta_{ik}$. However, we will use the dual variables (in the primal-dual sense) to guide the assignment policy. Hence the charges have to be distributed in a consistent manner to the assignment decisions of jobs to machines in the past. So in order to do the charging, we introduce a *prediction term*: at the arrival of each job j we charge to it an additional quantity of $\frac{w_j}{\epsilon_r} p_{ij}$. By doing this, the consistency is maintained by the rejection policy: if the charge from future jobs exceeds the prediction term of some job then the latter will be rejected.

Based on the above, we define

$$\lambda_{ij} = \begin{cases} \dfrac{w_j}{\epsilon_r} p_{ij} + w_j \displaystyle\sum_{\ell \in Q_i(r_j):\delta_{i\ell} \geq \delta_{ij}} p_{i\ell} + p_{ij} \sum_{\ell \in Q_i(r_j)\setminus\{k\}:\delta_{i\ell} < \delta_{ij}} w_\ell & \text{if } \delta_{ij} > \delta_{ik} \\ \dfrac{w_j}{\epsilon_r} p_{ij} + w_j \left(q_{ik}(r_j) + \displaystyle\sum_{\ell \in Q_i(r_j)\setminus\{k\}:\delta_{i\ell} \geq \delta_{ij}} p_{i\ell} \right) + p_{ij} \sum_{\ell \in Q_i(r_j):\delta_{i\ell} < \delta_{ij}} w_\ell & \text{otherwise} \end{cases}$$

which represents the total charge for job j if it is dispatched to machine i. Note that the only difference in the two cases of the definition of λ_{ij} is that we charge the quantity $w_j q_{ik}(r_j)$ to j only if $\delta_{ij} \leq \delta_{ik}$. Moreover, we do not consider the negative quantity that appears in the second case of Δ_{ij}. Intuitively, we do not decrease our estimation for the completion times of pending jobs when a job is rejected. The *dispatching policy* is the following: dispatch j to the machine $i^* = \text{argmin}_{i \in \mathcal{M}}\{\lambda_{ij}\}$. Intuitively, a part of Δ_{ij} may be charged to job k, and more specifically to the prediction part of λ_{ik}. However, we do not allow to exceed this prediction by applying rejection. In other words, the rejection policy can be re-stated informally as: we reject k just before we exceed the prediction charging part in λ_{ik}.

In order to keep track of the negative terms in Δ_{ij}, for each job $j \in \mathcal{J}$ we denote by D_j the set of jobs that are rejected by the algorithm after the release time of j and before its completion or rejection (including j in case it is rejected), that is the jobs that cause a decrease to the flow time of j. Moreover, we denote by j_k the job released at the moment we reject a job $k \in \mathcal{R}$. Then, we say that a job $j \in \mathcal{J}$ which is dispatched to machine $i \in \mathcal{M}$ is *definitively finished* $\sum_{k \in D_j} q_{ik}(r_{j_k})$ time after its completion or rejection. Let $U_i(t)$ be the set of jobs that are dispatched to machine $i \in \mathcal{M}$, they are already completed or rejected at a time before t, but they are not yet definitively finished at t.

It remains to formally define the dual variables. At the arrival of a job j, we set $\lambda_j = \frac{\epsilon_r}{1+\epsilon_r} \min_{i\in\mathcal{M}}\{\lambda_{ij}\}$ and we never change λ_j again. Let $W_i(t)$ be the total weight of jobs dispatched to machine $i \in \mathcal{M}$ in the schedule of $\mathcal{A}$, and either they are pending at t or they are not yet definitively finished at t, i.e., $W_i(t) = \sum_{\ell\in Q_i(t)\cup U_i(t)} w_\ell$. Then, we define $\gamma_i(t) = \frac{\epsilon_r}{1+\epsilon_r} W_i(t)$. Note that $\gamma_i(t)$ is updated during the execution of $\mathcal{A}$. Specifically, given any fixed time t, $\gamma_i(t)$ may increase if a new job j' arrives at any time $r_{j'} \in [r_j, t)$. However, $\gamma_i(t)$ does never decrease in the case of rejection since the jobs remain in $U_i(t)$ for a sufficient time after their completion or rejection.

8.4.3 Analysis

We first prove the following lemma which guarantees the feasibility of the dual constraint using the above definition of the dual variables.

Lemma 8.2. *For every machine $i \in \mathcal{M}$, job $j \in \mathcal{J}$ and time $t \geq r_j$, the dual constraint is feasible, that is $\frac{\lambda_j}{p_{ij}} - \gamma_i(t) - \delta_{ij}\left(t - r_j + p_{ij}\right) \leq 0$.*

Proof. Fix a machine i. We have observed above that, for any fixed time $t \geq r_j$, as long as new jobs arrive, the value of $\gamma_i(t)$ may only increase. Then, it is sufficient to prove the dual constraints for the job j using the values of $\gamma_i(t)$, $Q_i(t)$, $U_i(t)$ and $W_i(t)$ as these are defined at time r_j. Let k be the job executed in machine i at r_j.

We have the following cases.

CASE 1: $\delta_{ij} > \delta_{ik}$. In this case we may have rejected the job k at r_j. By the definitions of λ_j and λ_{ij}, we have

$$\begin{aligned}
\frac{\lambda_j}{p_{ij}} &\leq \frac{\epsilon_r}{(1+\epsilon_r)}\frac{\lambda_{ij}}{p_{ij}} = \frac{\epsilon_r}{1+\epsilon_r}\left(\frac{w_j}{\epsilon_r} + \delta_{ij}\sum_{\ell\in Q_i(r_j):\delta_{i\ell}\geq\delta_{ij}} p_{i\ell} + \sum_{\ell\in Q_i(r_j)\setminus\{k\}:\delta_{i\ell}<\delta_{ij}} w_\ell\right)\\
&= \frac{\epsilon_r}{1+\epsilon_r}\left(\frac{w_j}{\epsilon_r} + \delta_{ij}\sum_{\ell\in Q_i(r_j)\setminus\{j\}:\delta_{i\ell}\geq\delta_{ij}} p_{i\ell} + w_j + \sum_{\ell\in Q_i(r_j)\setminus\{k\}:\delta_{i\ell}<\delta_{ij}} w_\ell\right)\\
&= w_j + \frac{\epsilon_r}{1+\epsilon_r}\left(\delta_{ij}\sum_{\ell\in Q_i(r_j)\setminus\{j\}:\delta_{i\ell}\geq\delta_{ij}} p_{i\ell} + \sum_{\ell\in Q_i(r_j)\setminus\{k\}:\delta_{i\ell}<\delta_{ij}} w_\ell\right)
\end{aligned}$$

By the definition of $\gamma_i(t)$ we get

$$\begin{aligned}
\gamma_i(t) + \delta_{ij}(t - r_j + p_{ij}) &= \frac{\epsilon_r}{1+\epsilon_r}\sum_{\ell\in Q_i(t)\cup U_i(t)} w_\ell + \delta_{ij}(t - r_j) + w_j\\
&\geq \frac{\epsilon_r}{1+\epsilon_r}\left(\sum_{\ell\in Q_i(t)\cup U_i(t)} w_\ell + \delta_{ij}(t - r_j)\right) + w_j
\end{aligned}$$

Thus, it remains to show that

$$\delta_{ij} \cdot \sum_{\substack{\ell \in Q_i(r_j)\setminus\{j\}:\\ \delta_{i\ell} \geq \delta_{ij}}} p_{i\ell} \; + \sum_{\substack{\ell \in Q_i(r_j)\setminus\{k\}:\\ \delta_{i\ell} < \delta_{ij}}} w_\ell \leq \sum_{\ell \in Q_i(t) \cup U_i(t)} w_\ell \; + \delta_{ij}(t - r_j) \tag{8.4}$$

Let $\tilde{C}_j = r_j + \sum_{\ell \in Q_i(r_j):\delta_{i\ell} \geq \delta_{ij}} p_{i\ell}$ (if k is rejected) or $\tilde{C}_j = r_j + q_{ik}(r_j) + \sum_{\ell \in Q_i(r_j):\delta_{i\ell} \geq \delta_{ij}} p_{i\ell}$ (otherwise) be the estimated completion time of j at time r_j if it is dispatched to machine i.

Case 1.1: $t \leq \tilde{C}_j$**.** By the definition of $U_i(t)$, all jobs in $Q_i(r_j)$ with $\delta_{i\ell} < \delta_{ij}$ still exist in $Q_i(t) \cup U_i(t)$. Moreover, for every job $\ell \in Q_i(r_j) \setminus (Q_i(t) \cup U_i(t) \cup \{k\})$ it holds that $\delta_{i\ell} \geq \delta_{ij}$, since ℓ is processed before j by the algorithm. Then, by splitting the first term of the left-hand side of (8.4) we get

$$\begin{aligned}
&\delta_{ij} \cdot \sum_{\ell \in Q_i(r_j)\setminus\{j\}:\delta_{i\ell} \geq \delta_{ij}} p_{i\ell} \; + \sum_{\ell \in Q_i(r_j)\setminus\{k\}:\delta_{i\ell} < \delta_{ij}} w_\ell \\
&= \delta_{ij} \sum_{\ell \in Q_i(r_j)\setminus(Q_i(t)\cup U_i(t)\cup\{k\})} p_{i\ell} + \delta_{ij} \sum_{\substack{\ell \in (Q_i(r_j)\cap(Q_i(t)\cup U_i(t)))\setminus\{j\}:\\ \delta_{i\ell} \geq \delta_{ij}}} p_{i\ell} + \sum_{\substack{\ell \in (Q_i(r_j)\cap(Q_i(t)\cup U_i(t)))\setminus\{k\}:\\ \delta_{i\ell} < \delta_{ij}}} w_\ell \\
&\leq \delta_{ij} \sum_{\ell \in Q_i(r_j)\setminus(Q_i(t)\cup U_i(t)\cup\{k\})} p_{i\ell} + \sum_{\ell \in (Q_i(t)\cup U_i(t))\setminus\{j\}:\delta_{i\ell} \geq \delta_{ij}} w_\ell + \sum_{\ell \in (Q_i(t)\cup U_i(t))\setminus\{k\}:\delta_{i\ell} < \delta_{ij}} w_\ell \\
&\leq \delta_{ij}(t - r_j) + \sum_{\ell \in Q_i(t)\cup U_i(t)} w_\ell
\end{aligned}$$

where the first inequality is due to $\delta_{ij} p_{i\ell} \leq w_\ell$ for each $\ell \in Q_i(t) \cup U_i(t)$ with $\delta_{i\ell} \geq \delta_{ij}$, while the latter one holds since the set of jobs $Q_i(r_j) \setminus (Q_i(t) \cup U_i(t) \cup \{k\})$ corresponds to the set of pending jobs at r_j that start their execution after r_j and are definitively finished before t.

Case 1.2: $t > \tilde{C}_j$**.** By splitting the second term of the left-hand side of (8.4) we get

$$\begin{aligned}
&\delta_{ij} \cdot \sum_{\ell \in Q_i(r_j)\setminus\{j\}:\delta_{i\ell} \geq \delta_{ij}} p_{i\ell} \; + \sum_{\ell \in Q_i(r_j)\setminus\{k\}:\delta_{i\ell} < \delta_{ij}} w_\ell \\
&= \delta_{ij} \sum_{\ell \in Q_i(r_j)\setminus\{j\}:\delta_{i\ell} \geq \delta_{ij}} p_{i\ell} + \sum_{\ell \in Q_i(r_j)\setminus(Q_i(t)\cup U_i(t)\cup\{k\}):\delta_{i\ell} < \delta_{ij}} w_\ell + \sum_{\ell \in Q_i(r_j)\cap(Q_i(t)\cup U_i(t)):\delta_{i\ell} < \delta_{ij}} w_\ell \\
&\leq \delta_{ij}(\tilde{C}_j - r_j) + \delta_{ij} \sum_{\ell \in Q_i(r_j)\setminus(Q_i(t)\cup U_i(t)\cup\{k\}):\delta_{i\ell} < \delta_{ij}} p_{i\ell} + \sum_{\ell \in Q_i(t)\cup U_i(t)} w_\ell \\
&\leq \delta_{ij}(\tilde{C}_j - r_j) + \delta_{ij}(t - \tilde{C}_j) + \sum_{\ell \in Q_i(t)\cup U_i(t)} w_\ell
\end{aligned}$$

where the first inequality follows by the definition of $\tilde{C}_j$ and since $w_\ell < \delta_{ij} p_{i\ell}$ for each $\ell \in Q_i(r_j)$ with $\delta_{i\ell} < \delta_{ij}$, while the second inequality follows since the set of jobs in $Q_i(r_j) \setminus (Q_i(t) \cup U_i(t) \cup \{k\})$ with $\delta_{i\ell} < \delta_{ij}$ corresponds to the pending jobs at r_j which at time r_j have been scheduled to be executed during the interval $[\tilde{C}_j, t)$.

Case 2: $\delta_{ij} \leq \delta_{ik}$**.** In this case the job k is not rejected at the arrival of job j. By using the same arguments as in Case 1, we have

$$\frac{\lambda_j}{p_{ij}} \leq w_j + \frac{\epsilon_r}{1+\epsilon_r} \left(\delta_{ij} q_{ik}(r_j) + \delta_{ij} \sum_{\ell \in Q_i(r_j) \setminus \{k,j\} : \delta_{i\ell} \geq \delta_{ij}} p_{i\ell} + \sum_{\ell \in Q_i(r_j) : \delta_{i\ell} < \delta_{ij}} w_\ell \right)$$

Let $\tilde{C}_k = r_j + q_{ik}(r_j)$ be the estimated completion time of k at time r_j. We consider different scenarios.

Case 2.1: $t \leq \tilde{C}_k$**.** In this case, it holds that $w_k \geq \delta_{ij} p_k \geq \delta_{ij} q_{ik}(r_j)$. Then,

$$\begin{aligned}\gamma_i(t) + \delta_{ij}\left(t - r_j + p_{ij}\right) &\geq \frac{\epsilon_r}{1+\epsilon_r} \sum_{\ell \in Q_i(t) \cup U_i(t)} w_\ell + w_j \geq \frac{\epsilon_r}{1+\epsilon_r} \sum_{\ell \in Q_i(r_j)} w_\ell + w_j \\ &\geq \frac{\epsilon_r}{1+\epsilon_r} \left(\sum_{\ell \in Q_i(r_j) \setminus \{k\}} w_\ell + w_k \right) + w_j \geq \frac{\epsilon_r}{1+\epsilon_r} \left(\sum_{\ell \in Q_i(r_j) \setminus \{k\}} w_\ell + \delta_{ij} q_{ik}(r_j) \right) + w_j\end{aligned}$$

Hence, it remains to show

$$\delta_{ij} \sum_{\ell \in Q_i(r_j) \setminus \{k,j\} : \delta_{i\ell} \geq \delta_{ij}} p_{i\ell} + \sum_{\ell \in Q_i(r_j) : \delta_{i\ell} < \delta_{ij}} w_\ell - \sum_{\ell \in Q_i(r_j) \setminus \{k\}} w_\ell \leq 0$$

which directly holds as $\delta_{ij} p_{i\ell} \leq w_\ell$ for any job $j \in Q_i(r_j)$ with $\delta_{i\ell} \geq \delta_{ij}$.

Case 2.2: $t > \tilde{C}_k$**.** By the definition of $\gamma_i(t)$ we get

$$\gamma_i(t) + \delta_{ij}\left(t - r_j + p_{ij}\right) \geq \frac{\epsilon_r}{1+\epsilon_r} \left(\sum_{\ell \in Q_i(t) \cup U_i(t)} w_\ell + \delta_{ij} q_{ik}(r_j) + \delta_{ij}(t - r_j) \right) + w_j$$

Hence it suffices again to prove (8.4), which has been proved previously.

□

Lemma 8.3. *For the set $\mathcal{R}$ of jobs rejected by the algorithm $\mathcal{A}$ it holds that $\sum_{j \in \mathcal{R}} w_j \leq \epsilon_r \sum_{j \in \mathcal{J}} w_j$.*

Proof. Each job $j \in \mathcal{J}$ dispatched to machine $i \in \mathcal{M}$ may increase only the counter v_k of the job $k \in \mathcal{J}$ that was executed on i at r_j. In other words, each job j may be charged to at most one other job. Besides, we reject a job k the first time where $v_k > \frac{w_k}{\epsilon_r}$, meaning that the total weight of jobs charged to k is at least $\frac{w_k}{\epsilon_r}$. Hence, the total weight of rejected jobs is at most ϵ_r fraction of the total job weight in the instance. □

Theorem 8.1. *Given any $\epsilon_s > 0$ and $\epsilon_r \in (0,1)$, $\mathcal{A}$ is a $(1+\epsilon_s)$-speed $\frac{2(1+\epsilon_r)(1+\epsilon_s)}{\epsilon_r \epsilon_s}$-competitive algorithm that rejects jobs of total weight at most $\epsilon_r \sum_{j \in \mathcal{J}} w_j$.*

Proof. By Lemma 8.2, the proposed dual variables constitute a feasible solution for the dual program. By definition, the algorithm $\mathcal{A}$ uses for any machine at any time a factor of $1+\epsilon_s$ higher speed than that of the adversary. By Lemma 8.3, $\mathcal{A}$ rejects jobs of total weight at most $\epsilon_r \sum_{j \in \mathcal{J}} w_j$. Hence, it remains to give a lower bound for the dual objective.

We denote by $F_j^{\mathcal{A}}$ the flow-time of a job $j \in \mathcal{J} \setminus \mathcal{R}$ in the schedule of $\mathcal{A}$. By slightly abusing the notation, for a job $k \in \mathcal{R}$, we will also use $F_k^{\mathcal{A}}$ to denote the total time passed after r_k until deciding to reject a job k, that is, if k is rejected at the release of the job $j \in \mathcal{J}$ then $F_k^{\mathcal{A}} = r_j - r_k$. Denote by j_k the job released at the moment we decided to reject k, i.e., for the counter v_k before the arrival of job j_k we have that $w_k/\epsilon_r - w_{j_k} < v_k < w_k/\epsilon_r$.

Let Δ_j be the total increase in the flow-time caused by the arrival of the job $j \in \mathcal{J}$, i.e., $\Delta_j = \Delta_{ij}$, where $i \in \mathcal{M}$ is the machine to which j is dispatched by $\mathcal{A}$. By the definition of λ_j's, we have

$$\begin{aligned}\sum_{j \in \mathcal{J}} \lambda_j &\geq \frac{\epsilon_r}{1+\epsilon_r} \left(\sum_{j \in \mathcal{J}} \Delta_j + \sum_{k \in \mathcal{R}} \left(q_{ik}(r_{j_k}) \sum_{\ell \in Q_i(r_{j_k}) \cup \{k\} : \ell \neq j_k} w_\ell \right) \right) \\ &= \frac{\epsilon_r}{1+\epsilon_r} \left(\sum_{j \in \mathcal{J}} w_j F_j^{\mathcal{A}} + \sum_{j \in \mathcal{J}} \left(w_j \sum_{k \in D_j} q_{ik}(r_{j_k}) \right) \right)\end{aligned}$$

where the inequality comes from the fact that if $\delta_{ij} > \delta_{ik}$ then in the prediction part of the running job k at r_j we charge the quantity $w_j p_k$ instead of $w_j q_k(r_j)$ which is the real contribution of k to the weighted flow-time of job j. By the definition of $\gamma_i(t)$'s, we have

$$\begin{aligned}\sum_{i \in \mathcal{M}} \int_0^\infty \gamma_i(t) dt &= \frac{\epsilon_r}{1+\epsilon_r} \left(\sum_{i \in \mathcal{M}} \int_0^\infty \sum_{\ell \in Q_i(t)} w_\ell dt + \sum_{i \in \mathcal{M}} \int_0^\infty \sum_{\ell \in U_i(t)} w_\ell dt \right) \\ &= \frac{\epsilon_r}{1+\epsilon_r} \left(\sum_{j \in \mathcal{J}} w_j F_j^{\mathcal{A}} + \sum_{j \in \mathcal{J}} \left(w_j \sum_{k \in D_j} q_{ik}(r_{j_k}) \right) \right)\end{aligned}$$

since the set $Q_i(t)$ contains the pending jobs at time t dispatched on machine i, while each job $j \in \mathcal{J}$ appears by definition in $U_i(t)$ for $\sum_{k \in D_j} q_{ik}(r_{j_k})$ time after its completion or rejection.

Therefore, the proposed assignment for the dual variables leads to the following value of the dual objective

$$\sum_{j \in \mathcal{J}} \lambda_j - \frac{1}{1+\epsilon_s} \sum_{i \in \mathcal{M}} \int_0^\infty \gamma_i(t)dt \geq \frac{\epsilon_r \epsilon_s}{(1+\epsilon_r)(1+\epsilon_s)} \left(\sum_{j \in \mathcal{J}} w_j F_j^{\mathcal{A}} + \sum_{j \in \mathcal{J}} \left(w_j \sum_{k \in D_j} q_{ik}(r_{j_k}) \right) \right)$$
$$\geq \frac{\epsilon_r \epsilon_s}{(1+\epsilon_r)(1+\epsilon_s)} \sum_{j \in \mathcal{J}} w_j F_j^{\mathcal{A}} \geq \frac{\epsilon_r \epsilon_s}{(1+\epsilon_r)(1+\epsilon_s)} \sum_{j \in \mathcal{J} \setminus \mathcal{R}} w_j F_j^{\mathcal{A}}$$

Since the objective value of our linear program is at most twice the value of an optimal non-preemptive schedule, the theorem follows. □

8.5 Conclusion

In this chapter, we proposed a new generalized resource augmentation model for avoiding the pessimistic bounds arising from competitive analysis of online algorithms for non-preemptive scheduling problem. Then, we present a strong lower bound on the problem of minimizing weighted flow-time on unrelated machines. Finally, we show that this problem admits a scalable-competitive algorithms in a generalized resource augmentation model which combines the speed augmentation model and the rejection model. Our work suggests that the rejection can be quite powerful in dealing with non-preemptive settings and also suggests that for problems that remain stubborn even with some form of resource augmentation, one possible avenue is to consider allowing two or more types of augmentation.

9

SCHEDULING TO MINIMIZED WEIGHTED ℓ_K-NORMS OF FLOW TIME ON UNRELATED MACHINES

9.1 Introduction

Despite the common use of average flow time measure as a quality of service delivered to a set of jobs, there is a general concern regarding the unfair starving of some longer jobs. For instance, the classical policy of scheduling jobs with shortest remaining processing time is known to be optimal for minimizing average flow time on a single machine, whereas it has unbounded competitive ratio for minimizing maximum flow. Therefore, it is often required to have a quality measure that balances the trade-off between the average quality of service and maintaining *fairness* among the jobs. The most natural way to achieve this tradeoff is to use 2-norm of the flow time measure. In this chapter, we extend the ideas from previous chapter to the more general objective of minimizing the weighted ℓ_k-norm of flow-time of jobs on unrelated machines. The ℓ_k-norm captures the notion of fairness between jobs since it removes the extreme outliers and hence it is more appropriate to balance the difference among the flow-times of individual jobs than the average function, which corresponds to the ℓ_1-norm (see for example [MPS13]).

9.2 Problem Definition and Notations

We are given a set $\mathcal{M}$ of m unrelated machines. Let $\mathcal{J}$ denote the set of all jobs of our instance, which is not known a priori. Each job $j \in \mathcal{J}$ is characterized by its *release time* r_j, its *weight* w_j and if job j is executed on machine $i \in \mathcal{M}$ then it has a *processing time* p_{ij}. We study the *non-preemptive* setting. In this chapter, we consider a stronger non-preemptive model according to which we are only allowed to interrupt a job if we reject it, *i.e.*, we do not permit restarts. Moreover, each job has to be dispatched to one machine at its arrival and migration is not allowed.

Given a schedule $\mathcal{S}$, let C_j and F_j denote the completion time and flow time of the job j. Our objective is to create a non-preemptive schedule that minimizes the weighted ℓ_k-norm of the flow-time of all jobs, i.e., $(\sum_{j\in\mathcal{J}} w_j F_j^k)^{1/k}$, where $k \geq 1$.

Let $\delta_{ij} = \frac{w_j}{p_{ij}}$ be the *density* of a job j on machine i. Moreover, let $q_{ij}(t)$ be the remaining processing time at time t of a job j which is dispatched at machine i. A job j is called *pending* at time t, if it is already released at t but not yet completed, i.e., $r_j \leq t < C_j$.

9.3 Linear Programming Formulation

In this section, we study the objective of minimizing the weighted ℓ_k-norm of flow-times.

Let $\epsilon_s > 0$ and $0 < \epsilon_r < 1$ be the speed augmentation and the rejection fraction of our algorithm, respectively. For each machine $i \in \mathcal{M}$, job $j \in \mathcal{J}$ and time $t \geq r_j$, we introduce a binary variable $x_{ij}(t)$ which indicates if j is processed on i at time t. We consider the following linear programming formulation. Note that the optimal objective value of this linear program is at most $\frac{4(20k)^{k+3}}{\epsilon_s^{k+1}}$ times the total weighted k-power of flow-time of jobs in an optimal preemptive schedule.

$$\begin{aligned}
\min \sum_{i\in\mathcal{M}} \sum_{j\in\mathcal{J}} \int_{r_j}^{\infty} \frac{2(20k)^{k+3}}{\epsilon_s^{k+1}} \delta_{ij} \left[(t-r_j)^k + p_{ij}^k\right] x_{ij}(t)dt \\
\sum_{i\in\mathcal{M}} \int_{r_j}^{\infty} \frac{x_{ij}(t)}{p_{ij}} dt \geq 1 \quad \forall j \in \mathcal{J} \\
\sum_{j\in\mathcal{J}} x_{ij}(t) \leq 1 \quad \forall i \in \mathcal{M}, t \\
x_{ij}(t) \in \{0,1\} \quad \forall i \in \mathcal{M}, j \in \mathcal{J}, t \geq r_j
\end{aligned}$$

After relaxing the integrality constraints, we get the following dual program.

$$\begin{aligned}
&\max \sum_{j\in\mathcal{J}} \lambda_j - \sum_{i\in\mathcal{M}} \int_0^{\infty} \gamma_i(t)dt \\
&\frac{\lambda_j}{p_{ij}} - \gamma_i(t) \leq \frac{2(20k)^{k+3}}{\epsilon_s^{k+1}} \delta_{ij} \left[(t-r_j)^k + p_{ij}^k\right] \quad \forall i \in \mathcal{M}, j \in \mathcal{J}, t \geq r_j \\
&\lambda_j, \gamma_i(t) \geq 0 \quad \forall i \in \mathcal{M}, j \in \mathcal{J}, t
\end{aligned}$$

9.3.1 Algorithm and Dual Variables

The algorithm follows the same ideas of the one for the objective of minimizing the average weighted flow-time in previous chapter. Each job is immediately dispatched to a machine upon its arrival. We denote by $Q_i(t)$ the set of pending jobs at time t dispatched to machine $i \in \mathcal{M}$, i.e., the set of jobs dispatched to i that have been released but not yet completed and have not been rejected at t. Our *scheduling policy* for each machine $i \in \mathcal{M}$ is the following: at each time t when the machine i becomes idle or has just completed or interrupted some job, we start executing on i the job $j \in Q_i(t)$ of largest density, i.e., $j = \text{argmax}_{j' \in Q_i(t)}\{\delta_{ij'}\}$. In case of ties, we select the job that arrived the earliest.

When a machine $i \in \mathcal{M}$ starts executing a job $u \in \mathcal{J}$, we introduce a counter v_u (associated to job u) which is initially equal to zero. Each time when a job $j \in \mathcal{J}$ with $\delta_{ij} > \delta_{iu}$ is released during the execution of u and j is dispatched to i, we increase v_u by w_j. Then, the *rejection policy* is the following: we interrupt the execution of the job k and we mark it as rejected the first time where $v_u > \frac{w_u}{\epsilon_r}$. As before we define the set of rejected jobs D_j which causes a decrease to the flow time of j and we say that j is *definitively finished* $\sum_{u \in D_j} q_{iu}(r_{j_u})$ time after its completion or rejection. However, j does not appear to the set of pending jobs $Q_i(t)$ for any t after its completion or rejection. Recall that $U_i(t)$ is the set of jobs that have been marked finished at a time before t in machine i but they have not yet been definitively finished at t. For a job $j \in Q_i(t) \cup U_i(t)$, let $F_j(t)$ be the *remaining time* of j from t to to the moment it is definitively finished.

Let Δ_{ij} be the increase in the total weighted k-th power of flow-time occurred in the schedule of our algorithm if we assign a new job $j \in \mathcal{J}$ to machine i, following the above scheduling and rejection policies. Assuming that the job $u \in \mathcal{J}$ is executed on i at time r_j, we have that, if $v_u + w_j \leq \frac{w_u}{\epsilon_r}$ then

$$\Delta_{ij} = w_j \left(q_{iu}(r_j) + \sum_{\substack{a \in Q_i(r_j) \cup \{j\} \setminus \{u\}: \\ \delta_{ia} \geq \delta_{ij}}} p_{ia} \right)^k + \sum_{\substack{a \in Q_i(r_j) \setminus \{u\} \\ \delta_{ia} < \delta_{ij}}} w_a \left[\left(F_a(r_j) + p_{ij} \right)^k - F_a(r_j)^k \right],$$

otherwise,

$$\Delta_{ij} = w_j \left(\sum_{\substack{a \in Q_i(r_j) \cup \{j\}: \\ \delta_{ia} \geq \delta_{ij}}} p_{ia} \right)^k + \sum_{\substack{a \in Q_i(r_j) \setminus \{u\} \\ \delta_{ia} < \delta_{ij}}} w_a \left[\left(F_a(r_j) + p_{ij} - q_{iu}(r_j) \right)^k - F_a(r_j)^k \right],$$

where, in both cases, the first term corresponds to the weighted k-th power of the flow-time of job j if it is dispatched to i and the second term corresponds to the change of the weighted k-th power of flow-time for the jobs in $Q_i(r_j)$. Note that, the second case corresponds to the rejection of u and hence we do not have the term $q_{iu}(r_j)$ in the weighted flow-time of j, while the flow-time of each pending job is reduced by $q_{iu}(r_j)$.

Based on the above, we define λ_{ij} in the following. If $\delta_{ij} > \delta_{iu}$ then λ_{ij} equals to

$$\frac{2^k (10k)^k}{\epsilon_s^k} \frac{1+\epsilon_r}{\epsilon_r} w_j p_{ij}^k + \left(1 + \frac{\epsilon_s}{5}\right) w_j \left(\sum_{\substack{a \in Q_i(r_j) \cup \{j\} \setminus \{u\}: \\ \delta_{ia} \geq \delta_{ij}}} p_{ia} \right)^k + \sum_{\substack{a \in Q_i(r_j) \setminus \{u\} \\ \delta_{ia} < \delta_{ij}}} w_a \left[\left(F_a(r_j) + p_{ij} \right)^k - F_a(r_j)^k \right],$$

otherwise, λ_{ij} equals to

$$\frac{2^k(10k)^k}{\epsilon_s^k}\frac{1+\epsilon_r}{\epsilon_r}w_j p_{ij}^k + \left(1+\frac{\epsilon_s}{5}\right)w_j\left(q_{iu}(r_j) + \sum_{\substack{a\in Q_i(r_j)\cup\{j\}\setminus\{u\}:\\ \delta_{ia}\geq\delta_{ij}}} p_{ia}\right)^k + \sum_{\substack{a\in Q_i(r_j)\setminus\{u\}\\ \delta_{ia}<\delta_{ij}}} w_a\left[\left(F_a(r_j)+p_{ij}\right)^k - F_a(r_j)^k\right].$$

The value of λ_{ij} represents the total charge for job j if it is dispatched to machine i. Note that we do not consider the negative quantity that appears in the second case of Δ_{ij}. Moreover, the only difference in the two cases of the definition of λ_{ij} is in the second term. The coefficients of the terms in the formula of λ_{ij} are chosen in such a way that the total value of λ_{ij}'s can cover the total value of Δ_{ij}'s (more detail in Theorem 9.1). The *dispatching policy* is the following: dispatch j to the machine $i^* = \text{argmin}_{i\in\mathcal{M}}\{\lambda_{ij}\}$.

It remains to formally define the dual variables. At the arrival of a job $j \in \mathcal{J}$, we set $\lambda_j = \frac{\epsilon_r}{1+\epsilon_r}\min_{i\in\mathcal{M}}\{\lambda_{ij}\}$ and we will never change the value of λ_j again. Define $\gamma_i(t)$ as the following.

$$\gamma_i(t) = \frac{\epsilon_r}{1+\epsilon_r}\left(1+\frac{\epsilon_s}{2}\right)\left(1+\frac{\epsilon_s}{5}\right)\cdot k\sum_{a\in Q_i(t)\cup U_i(t)} w_a F_a(t)^{k-1}$$

Note that $\gamma_i(t)$ is updated during the execution of $\mathcal{A}$. More specifically, given any fixed time t, $\gamma_i(t)$ may increase if a new job j' arrives at any time $r_{j'} \in [r_j, t)$. However, $\gamma_i(t)$ does never decrease in the case of rejection since the jobs remain in $U_i(t)$ for a sufficient time after their completion or rejection.

9.3.2 Analysis

We prove the main technical lemma which guarantees the feasibility of the dual constraint using the above definition of the dual variables. Note that the inequality in Lemma 9.1 is stronger than the dual constraint.

Lemma 9.1. *For every machine $i \in \mathcal{M}$, job $j \in \mathcal{J}$ and time $t \geq r_j$, the dual constraint is feasible, that is*

$$\frac{\lambda_j}{p_{ij}} - \gamma_i(t) \leq \frac{(20k)^{k+3}}{\epsilon_s^{k+1}}\delta_{ij}\left[(t-r_j)^k + p_{ij}^k\right] + \frac{2^k(10k)^k}{\epsilon_s^k}p_{ij}^k$$

Proof. Fix a machine i and job j. For any fixed time $t \geq r_j$, as long as new jobs arrive, the value of $\gamma_i(t)$ may only increase. Hence, it is sufficient to prove the inequality assuming that no job will be released after r_j.

Let $Q_i^1(t) \subset Q_i(t)$ be the set of pending jobs u assigned to machine i and $\delta_{iu}(r_j) \geq \delta_{ij}$. Let $Q_i^2(t) = Q_i(t) \setminus Q_i^1(t)$. By definition of λ_{ij} and by convexity of function z^k,

$$\begin{aligned}
\frac{\lambda_{ij}}{p_{ij}} &\leq \frac{2^k(10k)^k}{\epsilon_s^k}\frac{1+\epsilon_r}{\epsilon_r}\delta_{ij}p_{ij}^k + k\sum_{\substack{a\in Q_i(r_j)\setminus\{u\}\\ \delta_{ia}<\delta_{ij}}} w_a\big(F_a(r_j)+p_{ij}\big)^{k-1} \\
&\qquad + \left(1+\frac{\epsilon_s}{5}\right)\begin{cases}
\delta_{ij}\left(\sum\limits_{\substack{a\in Q_i(r_j)\cup\{j\}\setminus\{u\}:\\ \delta_{ia}\geq\delta_{ij}}} p_{ia}\right)^k & \text{if } \delta_{ij}>\delta_{iu}\\
\delta_{ij}\left(q_{iu}(r_j)+\sum\limits_{\substack{a\in Q_i(r_j)\cup\{j\}\setminus\{u\}:\\ \delta_{ia}\geq\delta_{ij}}} p_{ia}\right)^k & \text{otherwise}
\end{cases} \\
&\leq \frac{2^k(10k)^k}{\epsilon_s^k}\frac{1+\epsilon_r}{\epsilon_r}\delta_{ij}p_{ij}^k + k\sum_{a\in Q_i^2(r_j)} w_a\big(F_a(r_j)+p_{ij}\big)^{k-1} \\
&\qquad + \left(1+\frac{\epsilon_s}{5}\right)\delta_{ij}\left(\sum_{a\in Q_i^1(r_j)} p_{ia}+p_{ij}\right)^k \qquad (9.1)
\end{aligned}$$

Note that in job u is included in the last term if $\delta_{iu} \geq \delta_{ij}$. Besides, the second inequality holds since $\sum_{a\in Q_i^1(r_j)} p_{ia} + p_{ij} \geq \sum_{a\in Q_i^1(r_j)\cup\{j\}} p_{ia}$ (and equality happens if j is not assigned to machine i).

Since $\lambda_j = \frac{\epsilon_r}{1+\epsilon_r}\min_i \lambda_{ij}$, the lemma inequality is a corollary of the following inequality.

$$\frac{\epsilon_r}{1+\epsilon_r}\frac{\lambda_{ij}}{p_{ij}} - \gamma_i(t) \leq \frac{(20k)^{k+3}}{\epsilon_s^{k+1}}\delta_{ij}\Big[(t-r_j)^k + p_{ij}^k\Big] + \frac{2^k(10k)^k}{\epsilon_s^k}\delta_{ij}p_{ij}^k$$

By (9.1) and by definition of $\gamma_i(t)$ and and $0 < \epsilon_r < 1$, in order to prove the above inequality, we will prove a stronger inequality

$$\begin{aligned}
&\left(1+\frac{\epsilon_s}{5}\right)\delta_{ij}\left(\sum_{a\in Q_i^1(r_j)} p_{ia}+p_{ij}\right)^k + k\sum_{a\in Q_i^2(r_j)} w_a\big(F_a(r_j)+p_{ij}\big)^{k-1} \\
&\qquad \leq k\left(1+\frac{\epsilon_s}{2}\right)\left(1+\frac{\epsilon_s}{5}\right)\sum_{a\in Q_i(t)} w_a F_a(t)^{k-1} + \frac{(20k)^{k+3}}{\epsilon_s^{k+1}}\delta_{ij}\Big[(t-r_j)^k + p_{ij}^k\Big] \qquad (9.2)
\end{aligned}$$

Let t_0 be the completion time of job j if j is scheduled in machine i by the algorithm. Inequality (9.2) follows Lemma 9.2 and Lemma 9.3 by choosing the parameter $\epsilon = \epsilon_s/(10k)$ in the latters.

□

In the analysis, we extensively use the following simple inequalities. For $a, b \geq 0$ and $\epsilon > 0$ small enough,

$$(a+b)^k \leq (1+\epsilon)^k a^k + \left(1+\frac{1}{\epsilon}\right)^k b^k \leq (1+2k\epsilon)a^k + \frac{2^k}{\epsilon^k} b^k \tag{9.3}$$

The proof of the first inequality is done by considering cases whether $b \leq \epsilon a$ or $b > \epsilon a$. In the former, the term $(1+\epsilon)^k a^k$ dominates $(a+b)^k$ while in the latter, $\left(1+\frac{1}{\epsilon}\right)^k b^k$ dominates $(a+b)^k$. The second inequality holds for ϵ small enough.

Lemma 9.2. *Fix a machine i and a job j and assume that no new job is released after r_j. Let t_0 be the completion time of job j if j is scheduled in machine i by the algorithm. Then, for every $r_j \leq t \leq t_0$ and for $\epsilon > 0$, it holds that*

$$(1+2k\epsilon)\delta_{ij}\left(\sum_{a\in Q_i^1(r_j)} p_{ia} + p_{ij}\right)^k + k\sum_{a\in Q_i^2(r_j)} w_a\big(F_a(r_j)+p_{ij}\big)^{k-1}$$
$$\leq k(1+8k\epsilon)\sum_{a\in Q_i(t)} w_a F_a(t)^{k-1} + k2^{k+3}\epsilon^{-(k+1)}\delta_{ij}\Big[(t-r_j)^k + p_{ij}^k\Big]$$

Proof. First, we prove the following claim.

Claim 1. *It holds that*

$$k\sum_{a\in Q_i^2(t)} w_a(F_a(r_j)+p_{ij})^{k-1}$$
$$\leq k(1+2k\epsilon)\sum_{a\in Q_i^2(t)} w_a F_a(t)^{k-1} + k2^k\epsilon^{-(k+1)}\delta_{ij}\left((t-r_j)^k + p_{ij}^k\right)$$

Proof of claim. Observe that $Q_i^2(r_j) = Q_i^2(t)$ and $q_{ia}(t) = q_{ia}(r_j)$ for $a \in Q_i^2(r_j)$ since no such job a is scheduled in interval $[r_j, t]$. Let V_1 be the set of jobs $a \in Q_i^2(t)$ such that $t - r_j \leq \epsilon(F_a(t) + p_{ij})$. We have

$$k\sum_{a\in Q_i^2(r_j)} w_a\left(F_a(r_j)+p_{ij}\right)^{k-1} \leq k\sum_{a\in Q_i^2(r_j)} w_a\left((t-r_j)+F_a(t)+p_{ij}\right)^{k-1}$$
$$= k\sum_{a\in V_1} w_a\left((t-r_j)+F_a(t)+p_{ij}\right)^{k-1} + k\sum_{a\in Q_i^2(t)\setminus V_1} w_a\left((t-r_j)+F_a(t)+p_{ij}\right)^{k-1}$$
$$\leq k(1+2k\epsilon)\sum_{a\in V_1} w_a\left(F_a(t)+p_{ij}\right)^{k-1} + k\sum_{a\in Q_i^2(t)\setminus V_1} w_a\left((t-r_j)+F_a(t)+p_{ij}\right)^{k-1}$$

$$
\begin{aligned}
&\leq k(1+2k\epsilon)\sum_{a\in V_1} w_a\Big(F_a(t)+p_{ij}\Big)^{k-1} \\
&\quad + \; k(1+2k\epsilon)\sum_{a\in Q_i^2(t)\setminus V_1} w_a\Big(F_a(t)+p_{ij}\Big)^{k-1} + \; k2^k\epsilon^{-k}\sum_{a\in Q_i^2(t)\setminus V_1} w_a(t-r_j)^{k-1} \\
&\leq k(1+2k\epsilon)\sum_{a\in Q_i^2} w_a\Big(F_a(t)+p_{ij}\Big)^{k-1} + k2^k\epsilon^{-k}\delta_{ij}\sum_{a\in Q_i^2(t)\setminus V_1} p_{ia}(t-r_j)^{k-1} \\
&\leq k(1+2k\epsilon)\sum_{a\in Q_i^2(t)} w_a\Big(F_a(t)+p_{ij}\Big)^{k-1} + k2^k\epsilon^{-(k+1)}\delta_{ij}(t-r_j)^k \qquad (9.4)
\end{aligned}
$$

The second inequality follows the definition of V_1. In the third inequality, we apply inequality (9.3). The fourth inequality follows because $\delta_{ij} \geq \delta_{ia}$ for all $a \in Q_i^2(t)$. The last inequality holds since $\sum_{a\in V_2} p_{ia} \leq \max_{a\in Q_i^2(t)\setminus V_1} F_a(t) \leq (t-r_j)/\epsilon$ (by definition of $Q_i^2(t)\setminus V_1$).

Let V_2 be the set of jobs $a \in Q_i^2(t)$ such that $p_{ij} \leq \epsilon F_a(t)$. Similarly, we have

$$
\begin{aligned}
k\sum_{a\in Q_i^2(t)} w_a(F_a(t)+p_{ij})^{k-1} &= k\sum_{a\in V_2} w_a(F_a(t)+p_{ij})^{k-1} + k\sum_{a\in Q_i^2(t)\setminus V_2} w_a(F_a(t)+p_{ij})^{k-1} \\
&\leq k(1+2k\epsilon)\sum_{a\in V_2} w_a F_a(t)^{k-1} + k\sum_{a\in Q_i^2(t)\setminus V_2} w_a(F_a(t)+p_{ij})^{k-1} \\
&\leq k(1+2k\epsilon)\sum_{a\in Q_i^2(t)} w_a F_a(t)^{k-1} + k2^k\epsilon^{-k}\sum_{a\in Q_i^2(t)\setminus V_2} w_a p_{ij}^{k-1} \\
&\leq k(1+2k\epsilon)\sum_{a\in Q_i^2(t)} w_a F_a(t)^{k-1} + k2^k\epsilon^{-k}p_{ij}^{k-1}\delta_{ij}\sum_{a\in Q_i^2(t)\setminus V_2} p_{ia} \\
&\leq k(1+2k\epsilon)\sum_{a\in Q_i^2(t)} w_a F_a(t)^{k-1} + k2^k\epsilon^{-(k+1)}\delta_{ij}p_{ij}^k \qquad (9.5)
\end{aligned}
$$

where the second inequality follows inequality (9.3) and rearranging terms; the third inequality holds since $\delta_{ia} \leq \delta_{ij}$ and $q_{ia}(t) = q_{ia}(r_j)$ for $a \in Q_i^2(t) = Q_i^2(r_j)$; the fourth inequality is due to the fact that $\sum_{a\in Q_i^2(t)\setminus V} p_{ia}$ is bounded by the maximal $F_a(t)$ for $a \in Q_i^2(t)\setminus V_2$, which is bounded by p_{ij}/ϵ (by definition of $Q_i^2(t)\setminus V_2$).

Combining (9.4) and (9.5), we get

$$
k\sum_{a\in Q_i^2(t)} w_a(F_a(r_j)+p_{ij})^{k-1} \leq k(1+2k\epsilon)\sum_{a\in Q_i^2(t)} w_a F_a(t)^{k-1} + k2^k\epsilon^{-(k+1)}\delta_{ij}\Big((t-r_j)^k + p_{ij}^k\Big)
$$

□

We are now proving Lemma 9.2. Observe that

$$\sum_{a\in Q_i^1(r_j)} q_{ia}(r_j) + p_{ij} = (t-r_j) + \sum_{a\in Q_i^1(t)} q_{ia}(t) + q_{ij}(t)$$

Therefore,

$$(1+2k\epsilon)\delta_{ij}\Big(\sum_{a\in Q_i^1(r_j)} q_{ia}(r_j) + p_{ij}\Big)^k = (1+2k\epsilon)\delta_{ij}\Big((t-r_j) + \sum_{a\in Q^1(t)} q_{ia}(t) + q_{ij}(t)\Big)^k$$

$$\le k2^{k+1}\epsilon^{-(k+1)}\delta_{ij}(t-r_j)^k + k(1+5k\epsilon)\delta_{ij}\Big(\sum_{a\in Q_i^1(t)} q_{ia}(t) + q_{ij}(t)\Big)^k$$

$$\le k2^{k+1}\epsilon^{-(k+1)}\delta_{ij}(t-r_j)^k + k(1+5k\epsilon)2^k\epsilon^{-(k+1)}q_{ij}^k(t) + k(1+5k\epsilon)(1+2k\epsilon)\delta_{ij}\Big(\sum_{a\in Q_i^1(t)} q_{ia}(t)\Big)^k$$

$$\le k2^{k+2}\epsilon^{-(k+1)}\delta_{ij}\Big[(t-r_j)^k + p_{ij}^k\Big] + k(1+8k\epsilon)\delta_{ij}\sum_{a\in Q_i^1(t)} q_{ia}(t)\Big(\sum_{b\in Q_i^1(t):\delta_{ib}\ge\delta_{ia}} q_{ib}(t)\Big)^{k-1}$$

$$\le k2^{k+2}\epsilon^{-(k+1)}\delta_{ij}\Big[(t-r_j)^k + p_{ij}^k\Big] + k(1+8k\epsilon)\sum_{a\in Q_i^1(t)} w_a F_a(t)^{k-1} \tag{9.6}$$

In the inequalities, we use (9.3) and estimations with ϵ sufficiently small. The last inequality is due to the fact that $\delta_{ij} \le \delta_{ia}$ for $a \in Q_i^1(t)$.

Hence, using Claim 1 and (9.6), we get

$$(1+2k\epsilon)\delta_{ij}\Big(\sum_{a\in Q_i^1(r_j)} p_{ia}\Big)^k + k\sum_{a\in Q_i^2(r_j)} w_a\big(F_a(r_j)+p_{ij}\big)^{k-1} \le k(1+8k\epsilon)\sum_{a\in Q_i(t)} w_a F_a(t)^{k-1} + k2^{k+3}\epsilon^{-(k+1)}\delta_{ij}\Big[(t-r_j)^k + p_{ij}^k\Big]$$

which is the lemma inequality. □

Lemma 9.3. *Fix a machine i and a job j and assume that no new job is released after r_j. Let t_0 be the completion time of job j if j is scheduled in machine i by the algorithm. Then, for every $t > t_0$ and for $\epsilon > 0$, it holds that*

$$(1+2k\epsilon)\delta_{ij}\Big(\sum_{a\in Q_i^1(r_j)} p_{ia} + p_{ij}\Big)^k + k\sum_{a\in Q_i^2(r_j)} w_a\big(F_a(r_j)+p_{ij}\big)^{k-1} \le k(1+8k\epsilon)\sum_{a\in Q_i(t)} w_a F_a(t)^{k-1} + k2^{k+2}\epsilon^{-(k+1)}\delta_{ij}\Big[(t-r_j)^k + p_{ij}^k\Big]$$

Proof. First we argue the following claim.

Claim 2. *It holds that*

$$k \sum_{a \in Q_i^2(r_j)} w_a F_a(r_j)^{k-1} \leq k\left(\frac{k+1}{k}\right)^{k-1} \sum_{a \in Q_i(t)} w_a F_a(t)^{k-1} + 3k^k \delta_{ij}(t-r_j)^k - k(t_0-r_j)^k$$

Proof of claim. Let $\{a_1, \ldots, a_h\} \subset Q_i^2(r_j)$ be the set of jobs processed by the algorithm in interval $[t_0, t]$ where all jobs in W but probably a_h have been completed. It means that at time t, the machine is processing job a_h or has just completed job a_{h-1}. Hence, $Q_i(t) = Q_i^2(r_j) \setminus \{a_1, \ldots, a_{h-1}\}$. Recall that $q_{ia}(t)$ is the remaining of job a at time t. We have

$$\begin{aligned}
&k \sum_{a \in Q_i^2(r_j)} w_a F_a(r_j)^{k-1} \\
&\leq k\delta_{ij} \sum_{b=1}^{h-1} p_{i,a_b} \left(t_0 - r_j + p_{i,a_1} + \ldots + p_{i,a_b}\right)^{k-1} \\
&\quad + k \sum_{a \in Q_i^2(r_j) \setminus \{a_1,\ldots,a_{h-1}\}} w_a \left(t - r_j + F_a(t)\right)^{k-1} \\
&\leq k(1+2k\epsilon)\delta_{ij} \left[\sum_{b=1}^{h-1} p_{i,a_b} (t_0 - r_j + p_{i,a_0} + \ldots + p_{i,a_{b-1}})^{k-1}\right] + k2^k \epsilon^{-k} \delta_{ij} \sum_{b=1}^{h-1} p_{i,a_b}^k \\
&\quad + k \sum_{a \in Q_i^2(t)} w_a \left(t - r_j + F_a(t)\right)^{k-1} \\
&\leq (1+2k\epsilon)\delta_{ij} \left[(t-r_j)^k - (t_0-r_j)^k\right] + k2^k \epsilon^{-k} \delta_{ij} (t-r_j)^k \\
&\quad + k \sum_{a \in Q_i(t)} w_a \left(t - r_j + F_a(t)\right)^{k-1} \qquad (9.7)
\end{aligned}$$

The first inequality follows the fact that $\delta_{ij} \geq \delta_{ia}$ for $a \in Q_i^2(r_j)$. In the second inequality, we use inequality (9.3) and conventionally $p_{i,a_0} = 0$. The last inequality holds because of the convexity of function z^k and $p_{i,a_1} + \ldots + p_{i,a_{h-1}} \leq t - t_0 \leq t - r_j$.

Let V_3 be the set of jobs $a \in Q_i(t)$ such that $t - r_j \leq \epsilon F_a(t)$. Let $V_4 = Q_i(t) \setminus V_3$. Then we have

$$\begin{aligned}
&k \sum_{a \in Q_i(t)} w_a (t - r_j + F_a(t))^{k-1} = k \sum_{a \in V_3} w_a (t - r_j + F_a(t))^{k-1} + k \sum_{a \in V_4} w_a (t - r_j + F_a(t))^{k-1} \\
&\leq k(1+\epsilon)^{k-1} \sum_{a \in V_3} w_a F_a(t)^{k-1} + k \sum_{a \in V_4} w_a (t - r_j + F_a(t))^{k-1}
\end{aligned}$$

$$\leq k(1+2k\epsilon)\sum_{a\in V_3} w_a F_a(t)^{k-1} + k(1+2k\epsilon)\sum_{a\in V_4} w_a F_a(t)^{k-1} + k2^k\epsilon^{-k}\sum_{a\in V_4} w_a(t-r_j)^{k-1}$$
$$\leq k(1+2k\epsilon)\sum_{a\in Q_i(t)} w_a F_a(t)^{k-1} + k2^k\epsilon^{-k}\sum_{a\in V_4}\delta_{ij}p_{i,a}(t-r_j)^{k-1}$$
$$\leq k(1+2k\epsilon)\sum_{a\in Q_i(t)} w_a F_a(t)^{k-1} + k2^k\epsilon^{-(k+1)}\delta_{ij}(t-r_j)^k \tag{9.8}$$

The first inequality follows the definition of V_3. The second one holds due to inequality (9.3) and for ϵ sufficiently small. The last inequality follows the observation that $\sum_{a\in V_4} p_{ia} \leq (t_0 - r_j) + \sum_{a\in V_4} q_{ia}(t) \leq (t-r_j) + \max_{a\in V_4} F_a(t) \leq (t-r_j)/\epsilon$ (by definition of V_4).

Using (9.7) and (9.8), we deduce that

$$k\sum_{a\in Q_i^2(r_j)} w_a F_a(r_j)^{k-1}$$
$$\leq k(1+2k\epsilon)\sum_{a\in Q_i(t)} w_a F_a(t)^{k-1} + k2^{k+1}\epsilon^{-(k+1)}\delta_{ij}(t-r_j)^k - \delta_{ij}(t_0-r_j)^k$$

which is the claim inequality. □

We are now proving the lemma. By exactly the same arguments to prove inequalities (9.5) and (9.8), we have

$$k\sum_{a\in Q_i^2(r_j)} w_a(F_a(r_j)+p_{ij})^{k-1} \leq k(1+2k\epsilon)\sum_{a\in Q_i^2(t)} w_a F_a(r_j)^{k-1} + k2^k\epsilon^{-(k+1)}\delta_{ij}p_{ij}^k \tag{9.9}$$

Therefore,

$$(1+2k\epsilon)w_j\left(\sum_{a\in Q_i^1(r_j)} p_{ia} + p_{ij}\right)^k + k\sum_{a\in Q_i^2(r_j)} w_a\big(F_a(r_j)+p_{ij}\big)^{k-1}$$
$$\leq (1+2k\epsilon)\delta_{ij}(t_0-r_j)^k + k\sum_{a\in Q_i^2(r_j)} w_a(F_a(r_j)+p_{ij})^{k-1}$$
$$\leq (1+2k\epsilon)\delta_{ij}(t_0-r_j)^k + k(1+2k\epsilon)\sum_{a\in Q_i^2(r_j)} w_a F_a(r_j)^{k-1} + k2^k\epsilon^{-(k+1)}\delta_{ij}p_{ij}^k$$
$$\leq k(1+5k\epsilon)\sum_{a\in Q_i(t)} w_a F_a(t)^{k-1} + k2^{k+2}\epsilon^{-(k+1)}\delta_{ij}\left[(t-r_j)^k + p_{ij}^k\right]$$

The first inequality holds since every job in Q_i^1 has been completed by t_0. The second inequality is due to inequality (9.9). The last inequality follows Claim 2. □

By the rejection policy, the algorithm rejects at most a small fraction of the total job weight. The proof of the following lemma is the same as Lemma 8.3.

Lemma 9.4. *For the set $\mathcal{R}$ of jobs rejected by the algorithm $\mathcal{A}$ it holds that $\sum_{j\in\mathcal{R}} w_j \leq \epsilon_r \sum_{j\in\mathcal{J}} w_j$.*

Theorem 9.1. *Given any $\epsilon_s > 0$ and $\epsilon_r \in (0,1)$, the algorithm is a $(1+\epsilon_s)$-speed*

$O\left(\frac{k^{(k+3)/k}}{\epsilon_r^{1/k}\epsilon_s^{(k+2)/k}}\right)$-competitive algorithm
that rejects jobs of total weight at most $\epsilon_r \sum_{j\in\mathcal{J}} w_j$.

Proof. By Lemma 8.2, the proposed dual variables constitute a feasible solution for the dual program. By definition, the algorithm uses for any machine at any time a factor of $1+\epsilon_s$ more speed with respect to the adversary. By Lemma 9.4, the algorithm rejects jobs of total weight at most $\epsilon_r \sum_{j\in\mathcal{J}} w_j$. Hence, it remains to give a lower bound for the dual objective based on the proposed dual variables.

We denote by F_j the flow-time of a job $j \in \mathcal{J}\setminus\mathcal{R}$ in the schedule of the algorithm. By slightly abusing the notation, for a job $k \in \mathcal{R}$, we will also use F_u to denote the total time passed after r_u until deciding to reject a job u. In other words, if the job u is rejected at the release of the job $j \in \mathcal{J}$ then $F_u = r_j + q_{iu}(r_j) - r_u$. Denote j_u the job released at the moment we decided to reject the job u, i.e., $w_u/\epsilon_r - w_{j_u} < v_u < w_u/\epsilon_r$ for the value of the counter v_u before the arrival of job j_u.

By the definition of λ_j's and as $0 < \epsilon_r < 1$, we have

$$\sum_{j\in\mathcal{J}} \lambda_j = \frac{\epsilon_r}{1+\epsilon_r} \sum_{j\in\mathcal{J}} \lambda_{ij}$$
$$\geq \frac{\epsilon_r}{1+\epsilon_r}\left(\sum_{j\in\mathcal{J}} w_j F_j^k + \sum_{j\in\mathcal{J}}\left(w_j \sum_{u\in D_j}\left(\left(F_j(r_{j_u}) + p_{ij_u}\right)^k - \left(F_j(r_{j_u}) + p_{ij_u} - q_{iu}(r_{j_u})\right)^k\right)\right)\right)$$

where the inequality follows the definition of λ_{ij}. It can be seen by decomposing the first term of Δ_{ij} (in case $\delta_{ij} > \delta_{iu}$) using inequality (9.3) as follows. For $\epsilon > 0$ sufficiently small,

$$w_j\left(q_{iu}(r_j) + \sum_{\substack{a\in Q_i(r_j)\cup\{j\}\setminus\{u\}:\\ \delta_{ia}\geq\delta_{ij}}} p_{ia}\right)^k \leq w_j(1+2k\epsilon)\left(\sum_{\substack{a\in Q_i(r_j)\cup\{j\}\setminus\{u\}:\\ \delta_{ia}\geq\delta_{ij}}} p_{ia}\right)^k + w_j 2^k \epsilon^{-k} p_{iu}^k$$

Then choose $\epsilon = \epsilon_s/(10k)$, the right-hand side of this inequality is captured by the first two terms in the definition of λ_{ij} (note that $0 < \epsilon_r < 1$). That also explains the (complex) coefficients in the definition of λ_{ij}.

By the definition of $\gamma_i(t)$'s, for ϵ_s sufficiently small,

$$\sum_{i\in\mathcal{M}} \int_0^\infty \gamma_i(t)dt = \sum_{i\in\mathcal{M}} \int_0^\infty \frac{\epsilon_r}{1+\epsilon_r}\left(1+\frac{\epsilon_s}{2}\right)\left(1+\frac{\epsilon_s}{5}\right)k \sum_{a\in Q_i(t)\cup U_i(t)} w_a F_a(t)^{k-1}dt$$

$$\leq \frac{\epsilon_r}{1+\epsilon_r}\left(1+\frac{3\epsilon_s}{4}\right)\left(\sum_{j\in\mathcal{J}} w_j F_j^k + \sum_{j\in\mathcal{J}}\left(w_j \sum_{u\in D_j}\left(\left(F_j(r_{j_u})+p_{ij_u}\right)^k - \left(F_j(r_{j_u})+p_{ij_u}-q_{iu}(r_{j_u})\right)^k\right.\right.$$

Therefore, the proposed assignment for the dual variables leads to the following value of the dual objective

$$\begin{aligned}\sum_{j\in\mathcal{J}}\lambda_j - \frac{1}{1+\epsilon_s}\sum_{i\in\mathcal{M}}\int_0^\infty \gamma_i(t)dt &\geq \frac{\epsilon_r}{(1+\epsilon_r)}\sum_{j\in\mathcal{J}} w_j F_j^k\left(1-\frac{1+3\epsilon_s/4}{1+\epsilon_s}\right)\\ &\geq \frac{\epsilon_r\epsilon_s}{4(1+\epsilon_r)(1+\epsilon_s)}\sum_{j\in\mathcal{J}\setminus\mathcal{R}} w_j F_j^k\end{aligned}$$

for ϵ_s sufficiently small.

Recall that the relaxation is at most $\frac{4(20k)^{k+3}}{\epsilon_s^{k+1}}$ times the total weighted k-power of flow-time of jobs in an optimal preemptive schedule. Therefore, we deduce the competitive ratio of the ℓ_k-norm objective (i.e., $\left(\sum_{j\in\mathcal{J}} w_j F_j^k\right)^{1/k}$) is at most $O\left(\frac{k^{(k+3)/k}}{\epsilon_r^{1/k}\epsilon_s^{(k+2)/k}}\right)$.

□

10

CONCLUSION

In this thesis, we dealt with trade-offs that often arise in the context of resource allocation problems. First, we formulated them as multi-objective optimization problems where the final outcome consists of a set of mutually incomparable solutions known as the Pareto front. Since most of the discrete problems are NP-hard, we concentrated our effort on finding an approximation of the Pareto front. For this purpose, we studied Pareto-based local search algorithms where we argued that previous algorithms had significant drawbacks in the sense that they removed candidate solutions prematurely. Taking this drawback into consideration, we designed a new double archive Pareto local search algorithm (DAPLS) that protects good candidate solutions. We showed that DAPLS terminates in a Pareto local optimal set. To escape the local optimum, we embedded DAPLS in a genetic framework. We showed empirically that genetic DAPLS outperforms state-of-the-art algorithms on several instances of bi-objective and tri-objective quadratic assignment problems.

It would be interesting to see how DAPLS performs for other kinds of multi-objective combinatorial problems such as the multi-objective versions of 0/1 knapsack, the travelling salesman problem, the scheduling problem, the graph coordination etc. In the future, we would like to study DAPLS for higher dimensional cost space where the size of the Pareto front is typically huge. Specifically, we would like to explore trade-offs between the time taken and the quality of solutions produced. Moreover, in such problems, DAPLS can take a large amount of time to reach a local Pareto optimal set. Our aim would be to use some heuristics to limit the size of the neighborhood.

In the second part of the thesis, we focused on the design of algorithms with theoretical guarantees in the context of scheduling. Specifically, we studied trade-offs in client-server systems where requests are modelled as a set of independent jobs with processing requirement and resources are modelled as a set of machines. Our aim was to create a feasible schedule with some degree of performance guarantee. One of the well-studied and important performance measures is the flow-time of a job. We studied several variants of this measure in different machine settings. First, we considered the problem of minimizing maximum weighted flow-time on a single machine where jobs arrive online and the weight of a job is inversely proportional to its processing time. We presented an improved lower bound of $\frac{\sqrt{5}-1}{2}\Delta$ where Δ is the ratio between the largest and the smallest processing times of the input instance. Then, we designed a *semi-online* algorithm that asymptotically achieves the performance guarantees. Our algorithm used the strategy of waiting-time where long jobs incur additional delay after their release time.

As it turns out, many of the flow-time related problems have strong lower bound. Therefore we explored these problems under a relaxed notion of resource augmentation. In this model, we first considered the problem of minimizing average flow on a single machine in the offline setting. In the classical model (without resource augmentation), there exists a strong lower bound of $O(\sqrt{n})$. Instead, we showed that there exists an $O(\frac{1}{\epsilon})$-approximation algorithm for average-flow problem where the algorithm is allowed to reject at most an ϵ-fraction of the total number of jobs. Our approach used the tree-based structure of the shortest remaining processing time policy to convert an optimal preemptive algorithm into a non-preemptive algorithm with the bounded number of rejections. We extended this idea to the offline problem of minimizing average stretch on a single machine.

In the last section of the thesis, we considered the problem of minimizing average weighted flow-time on a set of unrelated machines where jobs arrive online. Here, we showed the existence of a strong lower bound for the speed augmentation model. We then proposed a mathematical programming based framework that unifies the speed augmentation and the rejection model. Finally, we developed a scheduling algorithm that achieves $O(1)$-competitive ratio in this generalized resource augmentation model. We used the concept of dual-fitting for the analysis of our algorithm. Despite the widespread use of average flow, it is often criticized for being unfair in that an algorithm minimizing average flow may starve some jobs. Therefore, we considered the ℓ_k-norm of a flow-time which introduces some fairness among competing jobs. Using the above idea, we showed that there exists an $O(k)$-competitive algorithm for the problem of minimizing weighted ℓ_k-norm of flow-time on a set of unrelated machines.

Our generalized model opens up possibilities for different types of resource augmentation. As shown in this thesis, this model can be used to explain the competitiveness of algorithms for certain problems that were known to admit no performance guarantee even with speed augmentation. Besides in our model, one can benefit from the power of duality-based techniques for analyzing online algorithms. It would be interesting to consider how different problems can be explored under the new model.

BIBLIOGRAPHY

[ABG04] E. Angel, E. Bampis, and L. Gourvès. A *Dynasearch* neighborhood for the bicriteria traveling salesman problem. *Metaheuristics for Multiobjective Optimisation*, pages 153–176, 2004.

[AGK12] S. Anand, N. Garg, and A. Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1228–1241, 2012.

[AT09] A. Alsheddy and E. Tsang. Guided Pareto local search and its application to the $0\backslash 1$ multi-objective knapsack problems. In *International Conference on Metaheuristics*, 2009.

[BÖ6] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, 1996.

[BAKC99] F. Ben Abdelaziz, S. Krichen, and J. Chaouachi. *Meta-Heursitics - Advances and Trends in Local Search Paradigms for Optimization*. Kluwer Academic Publishers, 1999.

[BC09] N. Bansal and H. Chan. Weighted flow time does not admit $\mathrm{O}(1)$-competitive algorithms. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1238–1244, 2009.

[BCK+07] N. Bansal, H. Chan, R. Khandekar, K. Pruhs, C. Stein, and B. Schieber. Non-preemptive min-sum scheduling with resource augmentation. In *Proceedings of the IEEE Annual Symposium on Foundation of Computer Science*, pages 614–624, 2007.

[BCM98] M. Bender, S. Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 270–279, 1998.

[BD07] N. Bansal and K. Dhamdhere. Minimizing weighted flow time. *Journal of ACM Transactions on Algorithms*, 3, 2007.

[BEY98] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. 1998.

[BFNW11] K. Bringmann, T. Friedrich, F. Neumann, and M. Wagner. Approximation-guided evolutionary multi-objective optimization. In *Proceeding of the International Joint Conference on Artificial Intelligence*, pages 1198–1203, 2011.

[BLMSP06] L. Becchetti, S. Leonardi, A. Marchetti-Spaccamela, and K Pruhs. Online weighted flow time and deadline scheduling. *Journal of Discrete Algorithms*, 4:339 – 352, 2006.

[BMR03] M. A. Bender, S. Muthukrishnan, and R. Rajaraman. Approximation algorithms for average stretch scheduling. *Journal of Scheduling*, 7:195–222, 2003.

[BN09] N. Buchbinder and J. Naor. The design of competitive online algorithms via a primal: Dual approach. *Foundations and Trends Theoretical Computer Science*, 3:93–263, 2009.

[BP03] N. Bansal and K. Pruhs. Server scheduling in the ℓ_p norm: A rising tide lifts all boat. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 242–250, 2003.

[BP14] N. Bansal and K. Pruhs. The geometry of scheduling. In *SIAM Journal on Computing*, volume 43, pages 1684–1698, 2014.

[Bra85] R. M Brady. Optimization strategies gleaned from biological evolution. *Nature*, 317:804–806, 1985.

[Bru01] P. Brucker. *Scheduling Algorithms*. Springer-Verlag New York, Inc., 3rd edition, 2001.

[BT09] K. R. Baker and D. Trietsch. *Principles of Sequencing and Scheduling*. Wiley Publishing, 2009.

[Bun04] D. P. Bunde. SPT is optimally competitive for uniprocessor flow. *Information Processing Letters*, 90:233–238, 2004.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. 2004.

[BZ11] J. Bader and E. Zitzler. Hype: An algorithm for fast hypervolume-based many-objective optimization. *Journal of Evolutionary Computation*, 19:45–76, 2011.

[CC05] C. A. Coello and N. C. Cortés. Solving multiobjective optimization problems using an artificial immune system. *Genetic Programming and Evolvable Machines*, 6:163–190, 2005.

[CDGK15] A. R. Choudhury, S. Das, N. Garg, and A. Kumar. Rejecting jobs to minimize load and maximum flow-time. In *Proceedings of Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1114–1133, 2015.

[CDK15] A. R Choudhury, S. Das, and A. Kumar. Minimizing weighted l_p-norm of flow-time in the rejection model. In *Proceedings of the Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 25–37, 2015.

[CJ98] P. Czyzżak and A. Jaszkiewicz. Pareto simulated annealing: a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis*, 7:34–47, 1998.

[CKZ01] C. Chekuri, S. Khanna, and A. Zhu. Algorithms for minimizing weighted flow time. In *Proceeding of ACM Symposium on Theory of Computing*, pages 84–93, 2001.

[DA99] K. Deb and S. Agrawal. Understanding interactions among genetic algorithm parameters. In *Foundations of Genetic Algorithms*, pages 265–286, 1999.

[DAPM00] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Proceedings of the International Conference on Parallel Problem Solving from Nature*, pages 849–858, 2000.

[DH14] N. R. Devanur and Z. Huang. Primal dual gives almost optimal energy efficient online algorithms. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1123–1140, 2014.

[DL95] G. Dahl and A. Lokkentangen. A tabu search approach to the channel minimization problem. In *Proceedings of the International Conference on Optimization Techniques and Applications*, 1995.

[DLLIS11] J. Dubois-Lacoste, M. López-Ibánez, and T. Stützle. A hybrid TP+PLS algorithm for bi-objective flow-shop scheduling problems. *Computers and Operations Research*, 38:1219–1236, 2011.

[DSST16] P. F. Dutot, E. Saule, A. Srivastav, and D. Trystram. Online non-preemptive scheduling to optimize max stretch on a single machine. In *Proceedings of the International Conference on Computing and Combinatorics*, pages 483–495, 2016.

[DT10] M. M. Drugan and D. Thierens. Path-guided mutation for stochastic Pareto local search algorithms. In *Proceedings of the International Conference on Parallel Problem Solving from Nature*, pages 485–495, 2010.

[DT12] M. M. Drugan and D. Thierens. Stochastic Pareto local search: Pareto neighbourhood exploration and perturbation strategies. *Journal of Heuristics*, 18:727–766, 2012.

[EP09] V. A. Emelichev and V. A. Perepelitsa. On cardinality of the set of alternatives in discrete many-criterion problems. *Discrete Mathematics and Applications*, 2:461–472, 2009.

[EVS01] L. Epstein and R. Van Stee. Optimal online flow time with resource augmentation. In *Proceedings of the International Symposium on Fundamentals of Computation Theory*, pages 472–482, 2001.

[FF93] C. M. Fonseca and P. J. Fleming. Genetic algorithms for multiobjective optimization: Formulation discussion and generalization. In *Proceedings of the International Conference on Genetic Algorithms*, pages 416–423, 1993.

[FGLIP11] C. M. Fonseca, A. P. Guerreiro, M. López-Ibáñez, and L. Paquete. On the computation of the empirical attainment function. In *Proceedings of the International Conference on Evolutionary Multi-Criterion Optimization*, pages 106–120, 2011.

[GD91] D .E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Proceedings of Internationa conference on Foundations of Genetic Algorithms*, pages 69–93, 1991.

[GDK89] D. Goldberg, K. Deb, and B. Korb. Messy genetic algorithms: motivation, analysis, and first resutls. *Complex Systems*, 3:493–530, 1989.

[GF00] X. Gandibleux and A. Freville. Tabu search based procedure for solving the 0-1 multiobjective knapsack problem: The two objectives case. *Journal of Heuristics*, 6:361–383, 2000.

[GG84] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6:721–741, 1984.

[GKP13] A. Gupta, R. Krishnaswamy, and K. Pruhs. Online primal-dual for non-linear optimization with applications to speed scaling. In *Proceedings of the Annual International Workshop on Approximation and Online Algorithms*, pages 173–186, 2013.

[Glo86] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computer and Operations Research*, 13:533–549, 1986.

[GMF97] X. Gandibleux, N. Mezdaoui, and A. Fréville. A tabu search procedure to solve multiobjective combinatorial optimization problems. In *Proceedings of the Second International Conference on Multi-Objective Programming and Goal Programming*, pages 291–300, 1997.

[GMU97] X. Gandibleux, N. Mezdaoui, and E. L. B. Ulungu. Simulated annealing versus tabu search multi-objective approaches to the multiobjective knapsack problem. In *Proceeding of the International Conference on Multiple Criteria Decision-Making*, 1997.

[GR87] D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the International Conference on Genetic Algorithms and Their Application*, pages 41–49, 1987.

[Han80] P. Hansen. Bicriterion path problems. In *Proceedings of the International Conference on Multiple Criteria Decision Making Theory and Application*, pages 109–127, 1980.

[Han97] M. P. Hansen. Tabu search for multiobjective optimization: MOTS. In *Proceedings of the International Conference on Multiple Criteria Decision Making*, pages 6–10, 1997.

[HJ98] M.P. Hansen and A. Jaszkiewicz. *Evaluating the Quality of Approximations to the Non-dominated Set*. Technical report (IMM). Department of Mathematical Modelling, Technical University of Denmark, 1998.

[HJRFF94] A. Hertz, B. Jaumard, C. C. Ribeiro, and W. P. Formosinho Filho. A multi-criteria tabu search approach to cell formation problems in group technology with multiple objectives. *RAIRO - Operations Research - Recherche Opérationnelle*, 28:303–328, 1994.

[HR94] H. W. Hamacher and Günter Ruhe. On spanning tree problems with multiple objectives. *Annals of Operations Research*, 52:209–230, 1994.

[Hug07] H. Hugot. *Approximation et énumération des solutions efficaces dans les problèmes d'optimisatio combinatoire multi-objectifs*. Ph.D thesis - Université Paris-Dauphine, 2007.

[IKdW+14] M. Inja, C. Kooijman, M. de Waard, D. M. Roijers, and S. Whiteson. Queued Pareto local search for multi-objective optimization. In *Proceedings of the International Conference on Parallel Problem Solving from Nature*, pages 589–599, 2014.

[IKM14] S. Im, J. Kulkarni, and K. Munagala. Competitive algorithms from competitive equilibria: Non-clairvoyant scheduling under polyhedral constraints. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 313–322, 2014.

[IKMP14] S. Im, J. Kulkarni, K. Munagala, and K. Pruhs. Selfishmigrate: A scalable algorithm for non-clairvoyantly scheduling heterogeneous processors. In *Proceedings of the IEEE Annual Symposium on Foundations of Computer Science*, pages 531–540, 2014.

[ILMT15] S. Im, S. Li, B. Moseley, and E. Torng. A dynamic programming framework for non-preemptive scheduling problems on multiple machines. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1070–1086, 2015.

[KC00] J. D. Knowles and D. W. Corne. Approximating the nondominated front using the Pareto archived evolution strategy. *Journal of Evolutionary Computation*, 8:149–172, 2000.

[KC03] J. Knowles and D. Corne. Instance generators and test suites for the multiobjective quadratic assignment problem. *Evolutionary Multi-Criterion Optimization*, pages 295–310, 2003.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and C. D. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[KL70] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49:291–307, 1970.

[Kno06] J. Knowles. ParEGO: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10:50–66, 2006.

[KP94] A. Kolen and E. Pesch. Genetic local search in combinatorial optimization. *Discrete Applied Mathematics*, 48:273–284, 1994.

[KP95] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. In *Proceedings of Annual Symposium on Foundations of Computer Science*, pages 214–221, 1995.

[KP00] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of ACM*, 47:617–643, 2000.

[KTW95] H Kellerer, T. Tautenhahn, and G. J. Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single machine. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 418–426, 1995.

[LHM+12] A. Liefooghe, J. Humeau, S. Mesmoudi, L. Jourdan, and E. Talbi. On dominance-based multiobjective local search: Design, implementation and experimental analysis on scheduling and travelling salesman problems. *Journal of Heuristics*, 18:317–352, 2012.

[LIPS10] M. López-Ibáñez, L. Paquete, and T. Stützle. Experimental methods for the analysis of optimization algorithms. *Exploratory Analysis of Stochastic Local Search Algorithms in Biobjective Optimization*, pages 209–222, 2010.

[LLRKS93] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. Sequencing and scheduling: Algorithms and complexity. *Handbooks in operations research and management science*, 4:445–522, 1993.

[LNST16] G. Lucarelli, K. T Nguyen, A. Srivastav, and D. Trystram. Online nonpreemptive scheduling in a resource augmentation model based on duality. In *Proceedings of the Annual European Symposium on Algorithms*, pages 63:1–63:17, 2016.

[LR07] S. Leonardi and D. Raz. Approximating total flow time on parallel machines. *Journal of Computer and System Sciences*, 73(6):875 – 891, 2007.

[LSS03] X. Lu, R. A. Sitters, and L. Stougie. A class of on-line scheduling algorithms to minimize total completion time. *Operation Research Letters*, 31:232–236, 2003.

[LST16] G. Lucarelli, A. Srivastav, and D. Trystram. From preemptive to nonpreemptive scheduling using rejections. In *Proceedings of the International Conference on Computing and Combinatorics*, pages 510–519, 2016.

[LSV08] A. Legrand, A. Su, and F. Vivien. Minimizing the stretch when scheduling flows of divisible requests. *Journal of Scheduling*, 11:381–404, 2008.

[LT10a] T. Lust and J. Teghem. The multiobjective traveling salesman problem: A survey and a new approach. *Advances in Multi-Objective Nature Inspired Computing*, pages 119–141, 2010.

[LT10b] T. Lust and J. Teghem. Two-phase Pareto local search for the biobjective traveling salesman problem. *Journal of Heuristics*, 16:475–510, 2010.

[MF97] P. Merz and B. Freisleben. Genetic local search for the TSP: new results. In *Proceedings of the IEEE Conference on Evolutionary Computation*, pages 159–164, 1997.

[MF00] P. Merz and B. Freisleben. Fitness landscapes, memetic algorithms, and greedy operators for graph bipartitioning. *Journal on Evolutionary Computation*, 8:61–91, 2000.

[MG96] B. L. Miller and D. E. Goldberg. Genetic algorithms, selection schemes, and the varying effects of noise. *Journal on Evolutionary Computation*, 4:113–131, 1996.

[MH97] N. Mladenovica and P. Hansen. Variable neighborhood search. *Computers and Operations Research*, 24:1097–1100, 1997.

[Mie99] K. Miettinen. *Nonlinear Multiobjective optimization*. Springer US, 1999.

[Mit98] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1998.

[MPS13] B. Moseley, K. Pruhs, and C. Stein. The complexity of scheduling for p-norms of flow and stretch. In *Proceedings of the International conference on Integer Programming and Combinatorial Optimization*, pages 278–289, 2013.

[MRSG99] S. Muthukrishnan, R. Rajaraman, A. Shaheen, and J. E. Gehrke. Online scheduling to minimize average stretch. In *Proceedings of the Annual Symposium on Foundations of Computer Science*, pages 433–443, 1999.

[MS16] O. Maler and A. Srivastav. Double archive Pareto local search. In *Proceedings of the International Symposium on Computational Intelligence*, page To appear, 2016.

[Ngu13] K.T. Nguyen. Lagrangian duality in online scheduling with resource augmentation and speed scaling. In *Proceedings of the Annual European Symposium*, pages 755–766, 2013.

[NK13] Y. Nagata and S. Kobayashi. A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. *INFORMS Journal on Computing*, 25:346–363, 2013.

[NS04] N.Megow and A. S. Schulz. On-line scheduling to minimize average completion time revisited. *Operations Research Letters*, 32:485–490, 2004.

[OKK03] I. Ono, H. Kita, and S. Kobayashi. A real-coded genetic algorithm using the unimodal normal distribution crossover. *Advances in Evolutionary Computing: Theory and Applications*, pages 213–237, 2003.

[PCS04] L. Paquete, M. Chiarandini, and T. Stützle. Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. *Metaheuristics for Multiobjective Optimisation*, pages 177–199, 2004.

[PS06] L. Paquete and T. Stützle. Stochastic local search algorithms for multiobjective combinatorial optimization: Methods and analysis. *IRIDIA-Technical Report, Université de Bruxelles*, 2006.

[PSS07] L. Paquete, T. Schiavinotto, and T. Stützle. On local optima in multiobjective combinatorial optimization problems. *Annals of Operations Research*, 156:83–96, 2007.

[PSTW97] C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 140–149, 1997.

[PVdV70] C.N. Potts and S. Van de Velde. Dynasearch: Iterative local. *Technical Report LPOM-9511, Faculty of Mechanical Engineering, University of Twente, Enschede*, 49:291–307, 1970.

[PY00] C. H. Papadimitriou and M. Yannakakis. On the approximability of tradeoffs and optimal access of web sources. In *Proceedings of the Annual Symposium on Foundations of Computer Science*, page 86, 2000.

[SBÇ12] E. Saule, D. Bozdag, and Ü. V. Çatalyürek. Optimizing the stretch of independent tasks on a cluster: From sequential tasks to moldable tasks. *Journal of Parallel and Distributed Computing*, 2012.

[Sch85] J. D. Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of the International Conference on Genetic Algorithms*, pages 93–100, 1985.

[SD94] N. Srinivas and K. Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Journal on Evolutionary Computation*, 2:221–248, 1994.

[Ser94] P. Serafini. Simulated annealing for multi objective optimization problems. In *Proceedings of the International Conference on Multiple Criteria Decision Making*, pages 283–292, 1994.

[Spe98] W. M Spears. *The Role of Mutation and Recombination in Evolutionary Algorithms*. Ph.D Thesis, George Mason University, 1998.

[TL13] J. Tao and T. Liu. WSPT's competitive performance for minimizing the total weighted flow time: From single to parallel machines. *Mathematical Problems in Engineering*, 2013.

[TYH99] S. Tsutsui, M. Yamamura, and T. Higuchi. Multi-parent recombination with simplex crossover in real coded genetic algorithms. In *Proceedings of the International Conference on Genetic and Evolutionary Computation*, pages 657–664, 1999.

[UAB+91] N. L. J. Ulder, E. H. L. Aarts, H. J. Bandelt, P. J. M. Van Laarhoven, and E. Pesch. Genetic local search algorithms for the traveling salesman problem. In *Proceedings of International Conference on Parallel Problem Solving from Nature*, pages 109–116, 1991.

[UTF95] B. Ulungu, J. Teghem, and P. Fortemps. Heuristic for multi-objective combinatorial optimization problems by simulated annealing. *MCDM: Theory and Applications*, pages 229–238, 1995.

[VMC95] H. Michael Voigt, H. Mühlenbein, and D. Cvetkovic. Fuzzy recombination for the breeder genetic algorithm. In *Proceedings of the International Conference on Genetic Algorithms*, pages 104–113, 1995.

[WS11] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[ZBT07] E. Zitzler, D. Brockhoff, and L. Thiele. The hypervolume indicator revisited: On the design of Pareto-compliant indicators via weighted integration. In *Proceedings of the International Conference on Evolutionary Multi-Criterion Optimization*, pages 862–876, 2007.

[ZLT01] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength pareto evolutionary algorithm. *TIK-ETH, Technical Report*, 103, 2001.

[ZT98] E. Zitzler and L. Thiele. An evolutionary algorithm for multi-objective optimization: The strength pareto approach. *ETH, Technical Report*, 43, 1998.

[ZT99] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, 3:257–271, 1999.

[ZTL+03] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7:117–132, 2003.