Fast and Scalable Algorithm For Sequencing Problems with Private Information

by

Ha Nguyen
August 18, 2014

SUPERVISORS
Marc Uetz
Ruben Hoeksma

Contents

1 Introduction

Consider a small but busy airport where all planes land at the same time, for example at 6 am. Suppose that the airport has just 1 gate in which these landed planes can dock. Obviously at any point of time, at most one plane can dock at this gate. Also suppose that after being docked, it takes p_j minutes for the passengers to get out of plane j. We call p_j the processing time of plane j. The time between the landing time (6 am) and the docking time of plane j is the waiting time S_j of plane j. Each plane belongs to an airline. The economic cost per unit waiting time is not the same for all airline. We call the cost per unit waiting time for the airline that owns plane j, weight w_j. Each airline is the only one that knows exactly its own processing time and weight. The probability distributions of these information are, however, publicly known. Moreover, the airlines get compensated for their waiting times. The question is how do we decide the order in which the planes are sent to the gate, and the amount of money paid to each airline for its waiting time, such that the expected total payment is minimized?

The above example is a mechanism design problem for a scheduling problem with private information. This is the problem we study in this thesis.

1.1 Mechanism design

One can think of mechanism design as the reverse engineering of game theory. In game theory terminology, a mechanism induces a game whose equilibrium outcome is the objective that the mechanism designer has set [19].

For example, consider a seller who owns a house, and wants to sell it to a set of buyers. Each buyer has a value for the object, which is the utility of the house to the buyer. The seller wants to design a selling procedure, an auction for example, such that he gets the maximum possible price (revenue) by selling the house. If the seller knew the values of the buyers, then he would simply offer the house to the buyer with the highest value and give him a 'take-it-or-leave-it' offer at a price equal to that value. Clearly, the (highest value) buyer has no incentive to reject such an offer. Now, consider a situation where the seller is unaware of the values of the buyers. What selling procedure will give the seller the maximum possible revenue? A clear answer is impossible if the seller knows nothing about the values of the buyer. However, the seller may have some information about the values of the buyers. For example, the possible range of values, the probability of having these values etc. Given this information, is it possible to design a selling procedure that guarantees maximum (expected) revenue to the seller? In this example, the seller had a particular objective in mind, namely maximizing his revenue. Given this objective he wanted to design a selling procedure such that when buyers participate in the selling procedure and try

to maximize their own payoffs within the rules of the selling procedure, the seller will maximize his expected revenue over all such selling procedures.

The study of mechanism design looks at such issues. A mechanism designer needs to design a mechanism (a selling procedure in the above example) where strategic agents can interact. The interactions of agents result in an outcome. While there are several possible ways to design the rules of the mechanism, the designer has a particular objective in mind. Depending on the objective, the mechanism needs to be designed in a manner such that when strategic agents interact, the resulting outcome gives the desired objective, where the resulting outcome is an equilibrium of the game amongst the agents, that is induced by the mechanism.

1.2 Sequencing problem with private information

Back to our study, the setting comprises a single server who can handle one job at a time, and job-agents $j \in J = \{1, \ldots, n\}$ that compete for being processed. No job can be interrupted once started, and each job j is characterized by its processing time p_j and weight w_j. The weight w_j represents job j's disutility for waiting one unit of time. In addition, we assume that the service provider compensates jobs for waiting, while the job data are private and known to the job-agents themselves. The probability distributions of the private job data, however, are assumed to be publicly known. The problem is introduced in more detail in Chapter 2. This problem is an abstraction of economic situations where clients queue for a single scarce resource, for example, a specialized operation theater, while the information on the urgency and duration to treat each client is private, yet known probabilistically [1]. A concrete example for the latter are waiting lists for medical treatments in the Netherlands (Kenis 2006) [13].

1.3 Research objective

The effect of the private information setting is that the job-agents may have incentives to be dishonest, they may want to pretend to have higher waiting cost and/or higher processing time to get higher compensation payments. Thus, we here study the design of algorithms under the presence of 'selfish' job-agents who hold the inputs to the algorithm. The agents are selfish in the sense that they are interested in maximizing their own utility, and instead of revealing the true input, they may declare any false input that will increase their utilities. We aim to design a fast algorithm that works well with respect to the true input, although this information is not publicly known. More precisely, the goal of our mechanism design problem is to find an algorithm that computes for each instance (set of data) an appropriate scheduling rule and a payment scheme that incentivize the players to reveal their true inputs and at the same time minimizing the expected total payment that is made to the jobs.

1.4 Research method

The one-dimensional case, where the processing times of jobs are publicly known and the weights are the only private information, is solved, in polynomial time, by a version of Smith's rule [2, 6]. Smith's rule is explained at the end of Chapter 2. It also has been shown that when both waiting cost and processing times are private, optimal mechanisms generally do not satisfy an independence condition known as IIA, and that a closed form for optimal mechanisms is generally not conceivable [2, 6].Though, it is known that optimal mechanisms can be computed in polynomial time if the scheduling rule is allowed to be randomized [1].

We here aim at constructing an algorithm, that computes mechanisms fast. More precisely, we want that in reasonable computation times the algorithm produces mechanisms of good quality. To do that, we use the insight gained from the related literature. The quality of the algorithm, i.e. the quality of the output mechanisms and the speed of the algorithm are then proven through empirical research. The algorithm will be called Algorithm *F&E*, which stands for fast and effective.

1.5 Outline

In Chapter 2 we briefly discuss the related literature. Chapter 3 looks at the mechanism design setting. The notations and concepts, needed, are discussed. In Chapter 4, the formal problem formulation can be found alongside the research questions. With the concepts from Chapter 3, in Chapter 5 we are ready to discuss the related literature, mentioned in Chapter 2, in details. Chapter 6 covers the answers to the research questions, raised in Chapter 4. One of the answers describes our initial solution (algorithm) for the problem. We show that the initial algorithm does not work well. We also explain what may cause its poor performance, this provides us new insight into the problem. The newly gained knowledge together with intensive experimental research leads us to an alternative algorithm, which is described in Chapter 7 as Algorithm *F&E*. It is then empirically validated. The experiment results can be found in Chapter 8. We close the thesis with the conclusion chapter, Chapter 9.

2 Background information and related literature

The optimal mechanism design problem we study here, was introduced by Heydenreich et al.(2008) [2]. Mechanism design has important applications in economics (for example, design of voting procedures, markets, auctions), and more recently finds applications in networked-systems (for example, Internet interdomain routing, design of sponsored search auctions) [23]. Myerson among others led the development of mechanism design theory. Myerson (1981) determined the revenue-maximizing auction with risk-neutral bidders and independent private information. The trick in his analysis was the use of the revelation principle [22]. The revelation principle was introduced by Gibbard(1973) [7]. Later this principle was extended to the broader solution concept of Bayesian equilibrium by Dasgupta, Hammond and Maskin (1979) [4], Holmstrom (1977) [12], and Myerson (1979) [21], Harris and Townsend(1981) [9], and Rosenthal(1978) [27]. The principle states that any auction can be equivalently replaced by a direct mechanism in which bidders simultaneously report their private information and then the mechanism determines assignments and payments based on the vector of reports. For any equilibrium of any auction mechanism, there is an equivalent direct mechanism in which bidders truthfully report their types and agree to participate. Hence, without loss of generality we can restrict ourselves to incentive compatible, individually rational direct revelation mechanisms to understand properties of all auction games. The concepts of incentive compatibility and individually rationality will be explained in detail in Chapter 3.

The problem as presented by Heydenreich et al.(2008) [2] is divided in two cases. In the single dimensional case, the processing times p_j are public information and only the weights w_j are private. In the two-dimensional case, both the weights w_j and the processing times p_j are private. The private information of a job is called a job's type. In either of the two cases, they assumed the type spaces of the jobs to be discrete. This deviates from the traditional literature on auctions but the choice is not uncommon. There are also some economic situations in which discrete type spaces will be the natural ones to consider. For example, firms (agents) competing for contracts in a procurement auction may be able to make pre-auction investments to increase their competitiveness. This has been studied in the context of R&D investments in preparation for a competition for government contracts (cf. Dasgupta 1990 [5], Tan 1992 [28] , Piccione and Tan 1996 [25]), and capacity investments in competition for health care contracts (Li 2005) [16]. The discrete type space assumption allows one to formulate the problem of finding an expense-minimizing Bayesian incentive compatible mechanism as an integer linear program, while 'nothing of qualitative significance is lost in moving from a continuous to a discrete type space' [29].

For the single dimensional case, Heydenreich et al. [2,6] followed earlier work in auction theory. A graph theoretic interpretation of the so-called incentive compatibility constraints was used as by Rochet [26], Malakhov and Vohra [17], Müller, Perea, and Wolf [20], Lavi and Swamy [15], Heydenreich et al. [11]. A closed formula for the optimal mechanism was obtained. It has been shown that serving the jobs in the order

of non-increasing ratios of 'virtual' weights $\overline{w_j}$ over service times p_j (Smith's rule with respect to the virtual weights $\overline{w_j}$) is optimal for the service provider, as long as a certain regularity condition is fulfilled. This result is in agreement with known results for single dimensional mechanism design, e.g. Hartline and Karlin [10]. This mechanism also minimizes expected total payments in the standard Bayes-Nash setting and it can be implemented in dominant strategies [6]. Because we also use Smith's rule a lot in this thesis, an explanation about Smith's rule is given at the end of this chapter.

For the two-dimensional case, Heydenreich et al. [2,6] proved that the optimal mechanisms do not, in general, satisfy an independence condition which is known as '**independence of irrelevant alternatives (IIA)**' by giving a concrete instance (example), which has been found with integer linear programming techniques. Broadly speaking, an allocation rule f satisfies IIA if the relative order of any two jobs j_1 and j_2 depends only on the data of these two jobs (and hence this relative order is independent of the data of jobs other than j_1 and j_2). Their conclusion, drawn from the analysis of the mentioned instance, reads as follows: 'the optimal mechanism cannot be expressed in terms of the virtual weights along the lines of the single dimensional case. In fact, any kind of priority based scheduling algorithm, for example, scheduling using Smith's rule with respect to the virtual weight, where the virtual weights of a job depend only on the characteristics of that job itself, cannot be an optimal mechanism in general.' So for the two-dimensional case the next inequality holds.

$$\begin{aligned} & \textit{Expected total payments optimal deterministic mechanism} \\ \leq\ & \textit{Expected total payments optimal deterministic, IIA mechanism} \end{aligned} \tag{2.1}$$

and there exist instances where inequality (2.1) is strict.

Recently, Hoeksma and Uetz [1] proved that the optimal mechanism design problem can be solved in polynomial time. They proposed a solution based on linear programming techniques, that results in optimal randomized mechanisms. A linear relaxation of the optimal mechanism design problem is obtained by dropping the triangle inequalities and integrality constraints from a linear ordering ILP formulation of the problem. Yet the affine image of the linear ordering variables in their model, still yields a feasible point in the so-called scheduling polytope. By moving from the ILP formulation to its LP relaxation, one in fact move from deterministic scheduling rules to randomized ones. Even though it is a linear relaxation, it is still an exponential size formulation in general, as it depends on the size of the type space. The crucial ingredient to get rid of this dimensionality problem was the so-called LP compactification, in which the number of variables and constraints were reduced drastically, resulting in an LP formulation of polynomial size in the input. The LP computes the expected payments and a so-called interim schedule which represents the expected waiting times for the jobs. After that, the computed solution of the LP relaxation, specifically the computed interim schedule is translated into a randomized schedule $f(t)$ for any given reported type vector $t \in T$. In addition, they showed that randomized Bayes-Nash mechanisms perform better than deterministic Bayes-Nash mechanisms in terms

of optimal expected total payment. So together with inequality (2.1) we have

$$\begin{aligned} & \textit{Expected total payments optimal randomized mechanism} \\ \leq\ & \textit{Expected total payments optimal deterministic mechanism} \\ \leq\ & \textit{Expected total payments optimal deterministic, IIA mechanism} \end{aligned} \tag{2.2}$$

and there exist instances where the above inequalities are strict.

Note that in the single dimensional setting the two inequalities above are equalities instead. For the two-dimensional case, the complexity to find an optimal deterministic mechanism, with or without IIA-condition, remains open, and it is not even clear if the decision problem is contained in NP.

As mentioned earlier, we explain here **Smith's rule** because we make use of it a lot in this thesis. Suppose that 1 machine has to process n jobs $j \in J$. Each job j has positive processing time p_j and weight w_j, these information are publicly known. Figure 2.1 is an example of a schedule for 3 jobs, where S_j is the waiting/starting time of job j.

Figure 2.1: Example of a schedule for 3 jobs on 1 machine

We want to schedule the jobs so that the sum of the weighted starting times, $\sum_j w_j S_j$ is minimized. Smith's rule schedules the jobs in the order of non-increasing ratios w_j/p_j. That Smith's rule leads to an optimal schedule can be shown by a very simple interchange argument as follows.

We first prove that having non-increasing ratios is a necessary condition for a schedule to be optimal. Let f be a schedule in which the jobs are not in ratio order. Then, in f, there is a job γ that immediately precedes a job j and yet $w_\gamma/p_\gamma < w_j/p_j$. If job γ starts at time S_γ, then job j starts at time $S_\gamma + P_\gamma$. If we interchange these two jobs, this affects only their starting times, not those of other jobs. The result is a strict decrease in total cost, by

$$\begin{aligned} \Big[w_\gamma S_\gamma + w_j(S_\gamma + P_\gamma)\Big] - \Big[w_j S_\gamma + w_\gamma(S_\gamma + P_j)\Big] &= w_j P_\gamma - w_\gamma P_j \\ &= P_\gamma P_j \left(\frac{w_j}{p_j} - \frac{w_\gamma}{P_\gamma}\right) > 0 \end{aligned}$$

from which it follows that f is not optimal. Conversely, we now prove that having non-increasing ratios is a sufficient condition for a schedule to be optimal. Let f be

a schedule in which the jobs are in ratio order and let f^* be an optimal schedule. If $f \neq f^*$ then in f^* there is a job γ immediately preceding a job j, where j precedes γ in f. But then $w_j/p_j \geq w_\gamma/p_\gamma$ (because in f the jobs are processed in the order of non- increasing ratios) and $w_\gamma/p_\gamma \geq w_j/p_j$ (by the first part of this proof and because f^* is optimal) and, therefore, $w_\gamma/p_\gamma = w_j/p_j$. Interchanging the order of job γ and j in f^* creates a new schedule of equal cost. A finite number of such interchanges converts f^* to f, demonstrating that f is optimal.□

We first describe the elements of our mechanism design problem.

3.1 Job-agents

A finite group of individuals (job-agents) interact. This set is denoted by $J = \{1, \ldots, n\}$ with index j. Here, each agent j has a job with weight w_j and processing time p_j. We identify the jobs with the agents.

3.2 Private information

Job-agents hold private information. In the two-dimensional case both weight and processing time are private information, known only by the agent that owns the job. Job j's information is represented by a type $t_j = (w_j, p_j)$, which lies in a set $T_j := \{t_j^1, \ldots, t_j^{m_j}\}$. Formally, $T_j := W_j \times P_j$, where $W_j := \{w_j^1, \ldots, w_j^{m_j}\}$ with $w_j^1 < \ldots < w_j^{m_j}$ and $P_j = \{p_j^1, \ldots, p_j^{q_j}\}$ with $p_j^1 < \ldots < p_j^{q_j}$.

Let $\varphi_j(t_j)$ denote the probability for job j having type t_j. Probability distributions $\varphi_j(t_j)$ and type spaces T_j are public information. Further, we assume that types are independent across jobs. Let us denote by $T := T_1 \times \ldots \times T_n$ the set of all type profiles. Let φ be the joint probability distribution of $t = (t_1, \ldots, t_n) \in T$. Then by independence, $\varphi(t) = \Pi_{j=1}^n \varphi_j(t_j)$.

Moreover, (t_j, t_{-j}) denotes a type vector where t_j is the type of job j and t_{-j} are the types of all jobs except j, that is $t_{-j} \in T_{-j} = \times_{k \neq j} T_k$. For given $t \in T$ and $K \subseteq J$ we also define the notation $\varphi(t_K) := \prod_{k \in K} \varphi_k(t_k)$, particularly $\varphi(t_{-j}) := \prod_{k \neq j} \varphi_k(t_k)$.

3.3 Direct revelation mechanism

As mentioned in Chapter 2, because of the revelation principle, we can restrict ourselves to incentive compatible, individually rational direct revelation mechanisms. In a direct revelation mechanism, the only actions available to the job-agents are to reveal their types.

Further, a direct revelation mechanism consist of two components:

1. an allocation rule f,

2. a payment scheme π.

After job-agents report their types $t = (t_1, \ldots, t_n)$, and depending on those reported types, the allocation rule is nothing but a schedule $f(t)$ that determines the order in which the jobs are executed. Note that $f(t)$ is a non-preemptive scheduling, because no job can be interrupted once started. Next to the allocation rule, there is a vector of payments $\pi(t)$ that assigns a payment to every job-agent in order to compensate them for their waiting.

In other words, for each type profile $t := (t_1, \ldots, t_n) \in T := T_1 \times \ldots T_n$, the payment scheme is the vector $\pi(t) = (\pi_1(t), \ldots, \pi_n(t))$, where $\pi_j(t)$ is the amount paid to job j and the allocation rule f is represented by a permutation of the set $\{1, \ldots, n\}$. Such allocation rule representation is illustrated in the next example.

Example 1. Suppose we have an instance with 3 jobs. Job 1 has a type space containing only one type, type $(w_1, p_1) = (4, 1)$. For job 2, let $W_2 = \{2, 3\}$ and $P_2 = \{1\}$, so job 2 has two types namely $(w_2^1, p_2) = (2, 1)$ and $(w_2^2, p_2) = (3, 1)$, and let the corresponding probabilities be $\varphi((w_2^1, p_2)) = \varphi((w_2^2, p_2)) = 0.5$. Job 3 has the type space $(w_3, p_3^1) = (3, 2)$ and $(w_3, p_3^2) = (3, 3)$, with $\varphi((w_3, p_3^1)) = 0.8$ and $\varphi((w_3, p_3^2)) = 0.2$

So there are in total $1 \times 2 \times 2 = 4$ type profiles. As an example consider allocation rule f, assigning the following schedules to reported type profiles

$$[(4,1),(2,1),(3,2)] \to 132 \qquad [(4,1),(3,1),(3,2)] \to 123$$
$$[(4,1),(2,1),(3,3)] \to 231 \qquad [(4,1),(3,1),(3,3)] \to 312.$$

Consider the type profile $[(4,1),(2,1),(3,2)]$, agent 1 has no choice but reporting type $(4,1)$, if agent 2 reports type $(2,1)$ and agent 3 reports type $(3,2)$, then job 1 is processed first, job 3 secondly and job 2 lastly. For the other three type profiles, the representation of allocation rule f can be read in the same way.

3.4 Bayes-Nash incentive compatibility

We will be interested in the so-called Bayes-Nash setting. That is, given a mechanism (f, π), in the Bayes-Nash model each job's goal is to maximize its expected utility. Thus we next discuss the computation of the expected utilities.

3.4.1 Expected utility

Each job-agent j has to report a type $t_j \in T_j$. We assume that a job can only report a processing time that is not lower than the true processing time, and that a job is processed for his reported processing time, because reporting a shorter processing

time, can be easily punished by preempting the job after the declared processing time, before it is actually finished. This restriction on agent's responses is sometimes referred to as verifiability of information. In general, organizations function by giving their members some discretionary power. As a check on their freedom of choice, each agent is often held accountable for his decisions. Individual decisions may be made to suit objectives other than that of the organization as a whole, but gross neglect of responsibility to the organization can be detected and punished [8, 24].

Let D_j denote the set of decisions of agent j, in other words D_j contains the types $t_j \in T_j$ that job-agent j can report. Depending on the true type (w_j^i, p_j^k) of job agent j, D_j is the set contains all types $(w_j^{i'}, p_j^{k'})$ such that $p_j^{k'} \geq p_j^k$. Job-agents have preferences over decisions that are represented by a utility function $u_j : D_j \times T_j \to \mathbb{R}$. So $u_j(t_j^{i'}, t_j^i)$ is the benefit that job-agent j having type t_j^i receives from reporting type $t_j^{i'}$. Clearly, if $u_j(t_j^{i_1}, t_j^i) > u_j(t_j^{i_2}, t_j^i)$ then agent j, with true type t_j^i as the true type, prefers reporting type $t_j^{i_1}$ to reporting type $t_j^{i_2}$.

If in a given schedule, job j has waiting time S_j and the actual type (w_j, p_j), then the actual cost for waiting of job j is $w_j S_j$. But as compensation, job j receives payment π_j, and therefore its total utility equals $\pi_j - w_j S_j$.

As stated in the beginning of Section 3.4, in the Bayes-Nash setting, each job's goal is to maximize its expected utility. For that reason, we need to express the expected valuation of a job when reporting to be of type t_j, which is determined by the expected waiting time when reporting to be of type t_j. Formally, the expected waiting time of job j if it reports type t_j, and if the allocation rule f is in place, equals

$$ES_j(f, t_j) = \sum_{t_{-j} \in T_{-j}} S_j(f(t_j, t_{-j}))\varphi_{-j}(t_{-j})$$

and if $t_j^i = (w_j^i, p_j^i)$ and $t_j^k = (w_j^k, p_j^k)$ then

$$Eu_j(t_j^k, t_j^i) = E\pi_j(t_j^k) - w_j^i ES_j(f, t_j^k). \tag{3.1}$$

3.4.2 Bayes-Nash incentive compatibility

We restrict ourselves to direct revelation mechanisms, that are incentive compatible, individually rational in the Bayes-Nash setting. An incentive compatible mechanism is a direct revelation mechanism in which the agents report truthfully their information in equilibrium. Incentive compatibility captures the essence of designing a mechanism to overcome the self-interest of the agents, in an incentive-compatible mechanism an agent will choose to report its private information truthfully out of its own interest.

As mentioned above, we want our (output) mechanisms (f, π) to be **Bayes-Nash incentive compatible (BIC)**. Formally, mechanism (f, π) is BIC if truth telling is a

weakly dominant strategy in expectation, so for every job j and every two types $t_j^i = (w_j^i, p_j^i)$, $t_j^k = (w_j^k, p_j^k) \in T_j$:

$$\begin{aligned} Eu_j(t_j^i, t_j^i) &\geq Eu_j(t_j^k, t_j^i) \\ \iff E\pi_j(t_j^i) - w_j^i ES_j(f, t_j^i) &\geq E\pi_j(t_j^k) - w_j^i ES_j(f, t_j^k) \end{aligned} \tag{3.2}$$

The expectation is taken under the assumption that all agents apart from j report truthfully. The inequality in (3.2) just expresses the Bayes-Nash equilibrium concept: under the assumption that no other job deviates from being truthful, job j's expected utility is maximal when being truthful, too.

If for a given allocation rule f there exists a payment scheme π such that (f, π) is BIC, then f is called Bayes-Nash implementable. The payment scheme π is referred to as an incentive compatible payment scheme.

3.5 Individual rationality

Recall that next to Bayes-Nash incentive compatible, we require the mechanisms to be individually rational. Here, we explain what 'individually rational' means. Typically, there is an objective function that the designer wants to maximize or minimize. One common objective is to maximize the social welfare (the sum of the agents' utilities), but there are many others. For example, the designer may wish to maximize revenue. However, there are certain constraints on what the designer can do. For example, it would not be reasonable for the designer to specify that a losing bidder in an auction should pay a large amount of money: if so, the bidder would simply not participate in the auction. To prevent such scenario, individual rationality (IR) constraints are used. IR constraints sometimes known as 'voluntary participation' constraints, which follows from the idea that an agent is often not forced to participate but can decide whether or not to participate. Essentially, individual rationality places constraints on the level of expected utility that an agent receives from participation.

So next to Bayes-Nash incentive compatibility, we impose individual rationality constraints as well. Unlike in auctions, where participation is optional, we assume a priori that all jobs must be scheduled. Mathematically, individual rationality makes sure that the optimal mechanism design problem is bounded [6].

A mechanism (f, π) is (interim) **individually rational (IR)** if for every agent j and every type $t_j^i \in T_j$

$$Eu(t_j^i, t_j^i) = E\pi_j(t_j^i) - w_j^i ES_j(f, t_j^i) \geq 0. \tag{3.3}$$

Note that individual rationality only makes a claim about the expected utilities of job-agents that truthfully report their types.

Dummy type We introduce a dummy type t_j^d for each job j, with probability $\varphi_j(t_j^d) = 0$, and making sure that the dummy type gives zero utility to the job, by defining $ES_j(f, t_j^d) := 0$ and $E\pi_j(t_j^d) := 0$ for all jobs $j \in J$. Now, we impose the constraints in (3.2) also for $k = d$, which then implies the inequalities in (3.3). Therefore, the dummy types together with the mentioned assumptions guarantee that individual rationality is satisfied along with the incentive compatibility constraints.

For a job j having $|W_j| = m_j$ and $|P_j| = q_j$, that is job j has m_j weights and q_j processing times, it will also be convenient to identify the dummy type with $(w_j^{m_j+1}, p_j^h)$ for any $h = 1, \ldots, q_j$.

4 Problem formulation

With the mechanism's setting described in Chapter 3, for each instance we seek to find a mechanism that fulfill Bayes-Nash incentive compatible constraints (3.2) and individual rationality constraints (3.3), and among all such mechanisms minimizes the expected total payment that has to be made to the jobs. However, in earlier works it has been proved that while the single dimensional problem can be efficiently solved, in agreement with Myerson's results for single item auctions, the two-dimensional case of this mechanism design problem seems to be considerably more complicated than the two-dimensional case in the traditional auction models. In addition, it is conceivable that a closed form solution does not exist in general [6]. It is not yet known whether or not the two-dimensional problem here is hard computationally. Also recall that the problem can be solved as a LP, that results in optimal randomized mechanisms. From a practical point of view, we want the mechanisms, obtained from our algorithm, to be simple in the sense that they are deterministic and we want the algorithm, used to compute the mechanisms, to be fast in the sense that it is faster than the LP-model for large size instances.

Bearing in mind the complexity of the problem, we do not require the resulted expected total payment to be optimal but it should be a reasonable result, in any case not higher than the expected total payment given by Smith's rule. Thus the **main goal** of this study is to find an algorithm that for each instance (input data set) in small computation time, computes a mechanism that

- is Bayes-Nash incentive compatible and (interim) individually rational,
- gives a 'reasonable result', i.e. the expected total payment that belongs to the output mechanism is relatively small in comparison with the expected total payment that belongs to the optimal mechanism.

Recall that a mechanism consists of an allocation rule f and a payment scheme π. For each type profile $t := (t_1, \ldots, t_n) \in T := T_1 \times \ldots T_n$, the payment scheme is the vector $\pi(t) = (\pi_1(t), \ldots, \pi_n(t))$, where $\pi_j(t)$ is the amount paid to job j and the allocation rule f is represented by a permutation of the set $\{1, \ldots, n\}$. But as can be seen later, for a mechanism it suffices to give the allocation rule only, because for any implementable allocation rule f we can compute the minimal, incentive compatible, (interim) individually rational payment scheme using the graph theoretic approach (Lemma 2 Chapter 5). Further an allocation rule can be represented by list of all types of the jobs. In this list the types are arranged in the order of non-decreasing importance, see Section 6.4.

Research questions

The main goal of this thesis, automatically, gives rise to the next research questions:

(I) As mentioned in Chapter 2, optimal mechanisms generally do not satisfy IIA condition in the two-dimensional setting. As a result, the same graph theoretic approach, as used in the one-dimensional setting, must fail to determine an optimal mechanism for the scheduling problem. But using the insight from the graph theoretic approach, how do we (approximately) compute deterministic optimal-IIA mechanisms. Note that 'optimal-IIA mechanism' means one of the best IIA mechanisms, that is an mechanism that satisfies IIA condition, and compared to other IIA mechanisms it gives the lowest expected total payment. Also recall inequality (2.1):

$$\begin{aligned} & \textit{Expected total payments optimal deterministic mechanism} \\ \leq\ & \textit{Expected total payments optimal deterministic, IIA mechanism} \end{aligned}$$

(II) Secondly, how should we evaluate the performance of our designed algorithm? When can we say that the computation time of the algorithm is short and that the corresponding (output) mechanism is a fairly good mechanism? Thus, experimental inspections need to be carried out. More specifically, we aim to create one or more control algorithms, such that a comparison between the results from our algorithm and those from the control algorithms, can be made.

Sub-questions

Then, for research question (I) we have the next sub-questions:

1. (I-1) How do we exploit and embed the idea that leads to optimal mechanism in the one-dimensional case? Roughly speaking, the idea in one-dimensional case runs as follows.

   As described in the paper of Heydenreich et al. [2, 6] and in Chapter 5 of this thesis, when using the graph theoretic approach, the minimal incentive compatible payments **for a given allocation rule f** can easily be computed via the shortest paths (from every node to the dummy node) in the so-called reduced type graphs $T_j(f)$. Please do not confuse $T_j(f)$ with T_j: $T_j(f)$ denotes the type graph of job j given allocation rule f and T_j denotes the type space of job j.

   The question remains, which allocation rule f is optimal? In the one-dimensional setting, the optimal allocation rule is just Smith's rule with respect to modified weights, the so-called virtual weights. The virtual weights are actually computed using the shortest paths in the type graphs. However, in the single dimensional case all shortest paths are fixed, independent of the allocation rule f, which is the reason why a closed form solution is conceivable there.

2. (I-2) In constrast to the single dimensional case, the shortest paths depend on the allocation rule f in the two-dimensional case. Although several examples of small, two dimensional instances were given [2, 6], it has not been discussed how to find and select the shortest paths in the two dimensional type graphs.

So in order to exploit the graph theoretic approach, we need to implement an algorithm that finds and selects the shortest paths in the reduced type graphs efficiently.

3. (I-3) Analogous to sub-question (I-2), how do we compute the virtual weights for the two-dimensional instances?

For research question (II) we have the next sub-questions:

(II-1) How do we evaluate the quality of the (output) mechanisms?

(II-2) How do we evaluate the computation times of our algorithm (used to compute a mechanism for each instance)?

Observe that by answering all the sub-questions we already answer the research questions. As mentioned in Section 1.5, the answers to the sub-questions raised in this chapter, can be found in Chapter 6. However, to make the thesis more readable, sub-question (I-3) is answered in Chapter 5. Further, the answer to (I-2) precedes the answer to (I-1) in Chapter 6.

Now with the concepts from Chapter 3, we are ready to discuss important results from the analysis done by Heydenreich et al [2, 6], that we use to develop our own algorithm for the two-dimensional case.

5.1 The type graph

The type graph $T_j(f)$ of job j has nodes $T_j = W_j \times P_j$ and dummy node $t_j^d = t_j^{m_j+1}$. Remind that $W_j = \{w_j^1, \ldots, w_j^{m_j}\}$ with $w_j^1 < \ldots < w_j^{m_j}$ and $P_j = \{p_j^1, \ldots, p_j^{q_j}\}$ with $p_j^1 < \ldots < p_j^{q_j}$ and that only larger than the true processing times can be reported by any job. The type graph contains an arc from any node $(w_j^{i_1}, p_j^{k_1})$ to every other node $(w_j^{i_2}, p_j^{k_2})$ with $k_1 < k_2$. This way we have arcs only in direction of increasing processing times, since jobs can only overstate their processing time. Furthermore, every node has an arc to the dummy node, but there are no outgoing arcs from the dummy node.

The lengths of the arcs are computed by the next formula

$$\ell_{(i_1k_1,i_2k_2)} = w_j^{i_1}[ES_j(f, w_j^{i_2}, p_j^{k_2}) - ES_j(f, w_j^{i_1}, p_j^{k_1})] \tag{5.1}$$

$\ell_{(i_1k_1,i_2k_2)}$ represents the gain in expected waiting-cost for agent j by truthfully reporting type $(w_j^{i_1}, p_j^{k_1})$ of job j instead of lying type $(w_j^{i_2}, p_j^{k_2})$, it can be both positive or negative.

In terms of the arc's lengths, the incentive constraints for a BIC mechanism (f, π) of job j in (3.2) can then be written as

$$\begin{aligned}
u_j((w_j^{i_1}, p_j^{k_1}), (w_j^{i_1}, p_j^{k_1})) &\geq u_j((w_j^{i_2}, p_j^{k_2}), (w_j^{i_1}, p_j^{k_1})) \\
\Leftrightarrow E\pi_j(w_j^{i_1}, p_j^{k_1}) - w_j^{i_1} ES_j(f, w_j^{i_1}, p_j^{k_1}) &\geq E\pi_j(w_j^{i_2}, p_j^{k_2}) - w_j^{i_1} ES_j(f, w_j^{i_2}, p_j^{k_2}) \\
\Leftrightarrow E\pi_j(w_j^{i_1}, p_j^{k_1}) - w_j^{i_1} ES_j(f, w_j^{i_1}, p_j^{k_1}) + w_j^{i_1} ES_j(f, w_j^{i_2}, p_j^{k_2}) &\geq E\pi_j(w_j^{i_2}, p_j^{k_2}) \\
\Leftrightarrow E\pi_j(w_j^{i_1}, p_j^{k_1}) + \underbrace{w_j^{i_1}[ES_j(f, w_j^{i_2}, p_j^{k_2}) - ES_j(f, w_j^{i_1}, p_j^{k_1})]}_{\ell_{(i_1k_1,i_2k_2)}} &\geq E\pi_j(w_j^{i_2}, p_j^{k_2}) \\
\Leftrightarrow E\pi_j(w_j^{i_1}, p_j^{k_1}) + \ell_{(i_1k_1,i_2k_2)} &\geq E\pi_j(w_j^{i_2}, p_j^{k_2})
\end{aligned} \tag{5.2}$$

Inequality (5.2) implies that the expected payments $E\pi_j(\cdot)$ constitute a node potential in directed graph $T_j(f)$, and therefore Bayes-Nash implementability of an allocation rule f is equivalent to $T_j(f)$ having no negative length directed cycle. Consider a

directed cycle consisting of only two arcs, its length equals

$$\begin{aligned}\ell_{(i_1k_1,i_2k_2)}+\ell_{(i_2k_2,i_1k_1)} &= w_j^{i_1}[ES_j(f,w_j^{i_2},p_j^{k_2})-ES_j(f,w_j^{i_1},p_j^{k_1})]\\ &+ w_j^{i_2}[ES_j(f,w_j^{i_1},p_j^{k_1})-ES_j(f,w_j^{i_2},p_j^{k_2})]\\ &= (w_j^{i_1}-w_j^{i_2})[ES_j(f,w_j^{i_2},p_j^{k_2})-ES_j(f,w_j^{i_1},p_j^{k_1})]. \quad (5.3)\end{aligned}$$

Remind that we have arcs only in direction of increasing processing times, so there are no cycles in the vertical direction. From equation (5.3) it is easy to see that the length of any two-cycle is non-negative for all jobs j if and only if f fulfills the following monotonicity condition.

Monotonicity with respect to weights An allocation rule f satisfies monotonicity with respect to weights if for every job j and fixed $p_j \in P_j$, $w_j^i < w_j^k$ implies that $ES_j(f,w_j^i,p_j) \geq ES_j(f,w_j^k,p_j)$. In other words, for any two types from the same type space T_j of job j, with the same processing time, the type with larger weight can not have larger expected starting time.

The previous observation together with the fact that any cycle can be decomposed into two-cycles leads to an important theorem that links the Bayes-Nash implement-ability of an allocation rule to the monotonicity.

Theorem 1. *[2, 6] Allocation rule f is Bayes-Nash implementable if and only if it satisfies monotonicity with respect to weights.*

Note that we aim for mechanisms that satisfy BIC, from the previous theorem, the allocation rule in our mechanism need to satisfy monotonicity. But then a technical but useful lemma has been formulated, which states that in case of monotonicity the set of necessary incentive constraints can be reduced. Consequently, we can omit some arcs in the type graph, since the corresponding incentive constraints are implied by others. The resulted graph is called the reduced type graph.

Lemma 1. *[2, 6] Let f be an allocation rule satisfying monotonicity with respect to weights. For any job j, the following constraints imply all other incentive constraints*

$$\begin{aligned}E\pi_j(w_j^i,p_j^k)-w_j^iES_j(f,w_j^i,p_j^k) &\geq E\pi_j(w_j^{i+1},p_j^k)-w_j^iES_j(f,w_j^{i+1},p_j^k) \quad \forall i,k\\ E\pi_j(w_j^{i+1},p_j^k)-w_j^{i+1}ES_j(f,w_j^{i+1},p_j^k) &\geq E\pi_j(w_j^i,p_j^k)-w_j^{i+1}ES_j(f,w_j^i,p_j^k) \quad \forall i,k\\ E\pi_j(w_j^i,p_j^k)-w_j^iES_j(f,w_j^i,p_j^k) &\geq E\pi_j(w_j^i,p_j^{k+1})-w_j^iES_j(f,w_j^i,p_j^{k+1}) \quad \forall i,k\end{aligned}$$

Hence the **reduced type graph** contains only the next arcs:

- arcs from type (w_j^i,p_j^k) to (w_j^{i+1},p_j^k) for all $i \in \{1,\dots,m_j\}$ and $k \in \{1,\dots,q_j\}$,
- arcs from type (w_j^i,p_j^k) to (w_j^{i-1},p_j^k) for all $i \in \{2,\dots,m_j\}$ and $k \in \{1,\dots,q_j\}$,

- arcs from type (w_j^i, p_j^k) to (w_j^i, p_j^{k+1}) for all $i \in \{1, \ldots, m_j\}$ and $k \in \{1, \ldots, q_j - 1\}$.

A sketch of the reduced type graph is given in Figure (5.1).

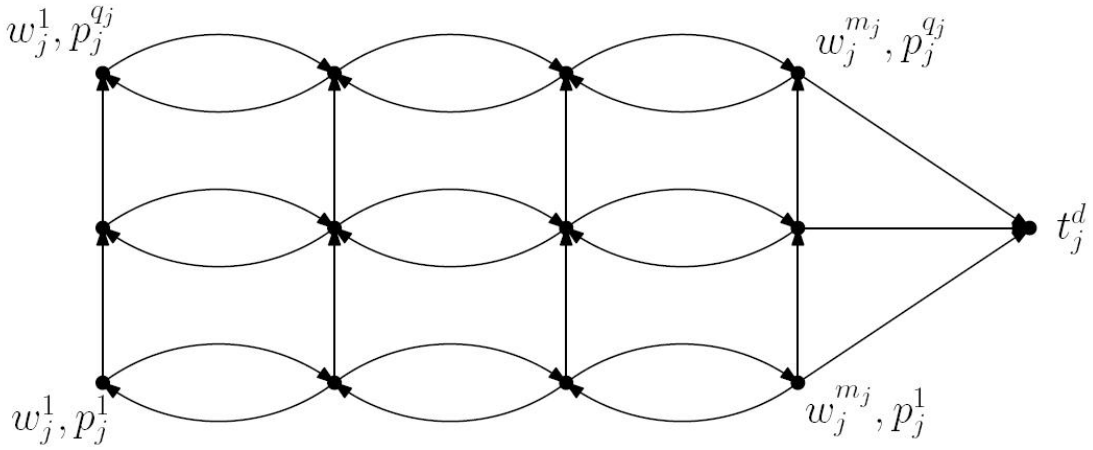


Figure 5.1: Sketch of a reduced type graph

Therefore from this point forward, by 'type graph' we mean 'reduced type graph'.

Minimal expected incentive compatible payments for implementable allocation rule f Depending on the scheduling rule f, the lengths of the arcs in the reduced type graph can be computed. After that , a lower bound for the expected payment $E\pi_j(w_j^i, p_j^k)$ for type (w_j^i, p_j^k) is found by taking the negative of the shortest path length from node (w_j^i, p_j^k) to dummy node (w_j^d, p_j) in the type graph $T_j(f)$. To verify this, let

$$P = \left[(w_j^i, p_j^k) = a_0, a_1, \ldots, a_m = (w_j^d, p_j)\right]$$

denote some directed path from $(w_j^{i_1}, p_j^{k_1})$ to the dummy node in the graph $T_j(f)$ for job j. Denote by $\ell(P)$ its length. Let (f, π) be a Bayes-Nash incentive compatible mechanism. From inequality (5.2) we have

$$E\pi_j(a_i) \leq E\pi_j(a_{i-1}) + \ell_{a_{i-1}a_i} \quad \text{for } i = 1, \ldots, m$$

Adding up the m incentive constraints above yields

$$E\pi_j(w_j^d, p_j) \leq E\pi_j(w_j^i, p_j^k) + \ell(P)$$

By definition, we have $E\pi_j(w_j^d, p_j) = 0$, thus

$$\begin{aligned} 0 &\leq E\pi_j(w_j^i, p_j^k) + \ell(P) \\ \Leftrightarrow E\pi_j(w_j^i, p_j^k) &\geq -\ell(P) \end{aligned} \tag{5.4}$$

Recall that for any implementable allocation rule f, $T_j(f)$ has no negative length directed cycle. Consequently, shortest paths exist for all nodes in the graph. In light of inequality (5.4), the next lemma has been proven

Lemma 2. *[2, 6] For any implementable allocation rule f, the payment scheme defined by*

- $\pi_j(t_j^d) = \pi_j(w_j^{m_j+1}, p_j) = 0$ *and*
- *for $i = 1, \ldots, m_j$ and for $k = 1, \ldots, q_j$, the payment $\pi_j^f(w_j^i, p_j^k)$ equals the negative of shortest path's length from (w_j^i, p_j^k) to the dummy node in the reduced type graph $T_j(f)$*

is incentive compatible, (interim) individually rational and minimizes the expected total payment made to the jobs.

The lengths of the shortest paths depend on the expected starting times of the types and the expected starting times of the types depend on the public data of all jobs, i.e. the weights, the processing times, and the corresponding distributions of all jobs. However, the payments as defined in Lemma 2 are independent of the reported type profile. That is, job j receives one and the same payment when reporting type t_j independent of what other agents report, so that $\pi(t_j, t_{-j}) = \pi(t_j) = E\pi(t_j)$. Let us call this property **'independence of reported types'**.

Because our algorithm draw ideas from the one-dimensional case, we briefly discuss the optimal payments for a given allocation rule f in one-dimensional setting. In the single dimensional case, the shortest paths in the type graphs $T_j(f)$ are independent of the (implementable) allocation rule f. Due to the implementability of f, there are no negative cycles in the graph, so all shortest paths must be simple paths. Finding oneself in any node, one always takes the arcs that goes to the node on the right until the dummy node is reached, see Figure 5.2.

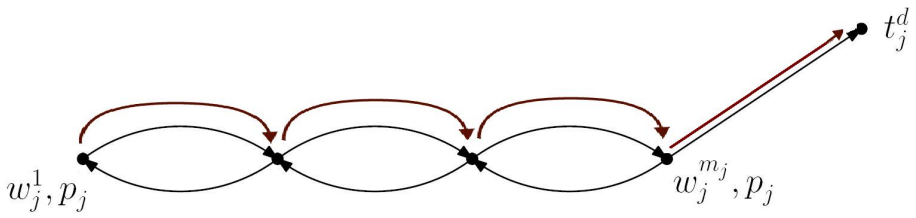


Figure 5.2: Single dimension reduced type graph

The above observation together with Lemma 2 brings us the next result. The payment scheme

$$\pi_j(w_j^{m_j+1}) = 0 \tag{5.5}$$

$$\pi_j(w_j^i) = \sum_{h=i}^{m_j} w_j^h \left[ES_j(f, w_j^h) - ES_j(f, w_j^{h+1}) \right] \quad \forall i = 1, \ldots, m_j \tag{5.6}$$

is incentive compatible, (interim) individually rational and minimizes the expected total payment made to the jobs in the single dimensional case.

5.2 Virtual weights

In this section, we answer sub-question (I-3). Recall that we want to use the graph theoretic approach to compute (approximately) optimal-IIA mechanisms. Therefore, we need to know how to compute the virtual weights for the two-dimensional instances.

In the works of Heydenreich et al. [2,6] the computation of the virtual weights is only discussed for single dimensional case. However, mimicking their arguments we easily obtain the virtual weights in the two-dimensional case as follows.

For the moment, assume that if there are different shortest paths found from one node to the dummy node, we randomly choose one. Given an implementable allocation rule f, the minimal expected incentive compatible payments can be computed using Lemma 2. Let $\mathscr{L}(w_j^i, p_j^k)$ denotes the shortest path's length of from (w_j^i, p_j^k) to the dummy node, the corresponding expected payment to job-agent j is

$$\pi_j^f = \sum_{i=1}^{m_j} \sum_{k=1}^{q_j} \varphi(w_j^i, p_j^k)(-\mathscr{L}(w_j^i, p_j^k)) \tag{5.7}$$

But $\mathscr{L}(w_j^i, p_j^k)$ is just the sum of arc's lengths of all arcs lying on the chosen shortest path from (w_j^i, p_j^k) to the dummy node. Let

$$P = \left[(w_j^i, p_j^k) = a_0, a_1, \ldots, a_m = (w_j^d, p_j) \right]$$

denotes the chosen shortest path from $(w_j^{i_1}, p_j^{k_1})$ to the dummy node in the type graph $T_j(f)$ of job j. Further, let $w_j^{a_i}$ denotes the weight belonging to node a_i and $ES(a_i)$ the expected starting time of the type corresponding with node a_i. Then

$$\begin{aligned}
\mathscr{L}(a_0) &= \ell_{a_0 a_1} + \ldots + \ell_{a_{m-1} a_m} \\
&= w_j^{a_0}(ES(a_1) - ES(a_0)) \ldots + w_j^{a_{m-1}}(ES(a_m) - ES(a_{m-1})) \\
\Rightarrow -\mathscr{L}(a_0) &= w_j^{a_0}(ES(a_0) - ES(a_1)) \ldots + w_j^{a_{m-1}}(ES(a_{m-1}) - ES(a_m)) \\
&= w_j^{a_0} ES(a_0) + \ldots + (w_j^{a_{m-1}} - w_j^{a_{m-2}}) ES(a_{m-1}) - w_j^{a_{m-1}} ES(a_m)
\end{aligned}$$

by definition the expected starting time of the dummy node a_m is zero so the last term in the above expression is zero and

$$
\begin{aligned}
&-\mathscr{L}(a_0) = w_j^{a_0} ES(a_0) + \ldots + (w_j^{a_{m-1}} - w_j^{a_{m-2}}) ES(a_{m-1}) \qquad (5.8)\\
&\Rightarrow \varphi(a_0)(-\mathscr{L}(a_0)) = \varphi(a_0) w_j^{a_0} ES(a_0) + \ldots + \varphi(a_0)(w_j^{a_{m-1}} - w_j^{a_{m-2}}) ES(a_{m-1})\\
&= \varphi(a_0) \underbrace{w_j^{a_0}} ES(a_0) + \ldots + \varphi(a_{m-1}) \underbrace{\frac{\varphi(a_0)(w_j^{a_{m-1}} - w_j^{a_{m-2}})}{\varphi(a_{m-1})}} ES(a_{m-1})\\
&= \varphi(a_0)\xi^P(a_0) ES(a_0) + \ldots + \varphi(a_{m-1}) \qquad \xi^P(a_{m-1}) \qquad ES(a_{m-1})\\
&= \sum_{i=0}^{m-1} \varphi(a_i)\xi^P(a_i) ES(a_i). \qquad (5.9)
\end{aligned}
$$

Now let call the term $\xi^P(a_i)$ in the last equation, the virtual weight contribution by shortest path P to node a_i. For each node (w_j^i, p_j^k) we find all shortest paths passing it, including shortest paths with starting node other than (w_j^i, p_j^k) itself. Next define the virtual weight of type (w_j^i, p_j^k), denoted by $\overline{w_j}^{(i,k)}$, to be the sum of the virtual weight contributions by all shortest paths passing node (w_j^i, p_j^k). From, equation 5.9, it is easy to see that equation (5.7) can be rewritten as

$$
\pi_j^f = \sum_{i=1}^{m_j} \sum_{k=1}^{q_j} \varphi(w_j^i, p_j^k) \overline{w_j}^{(i,k)} ES(f, w_j^i, p_j^k) \qquad (5.10)
$$

and the expected total payment is given by

$$
P^{\min}(f) = \sum_{j \in J} \sum_{i,k} \varphi(w_j^i, p_j^k) \overline{w_j}^{(i,k)} ES(f, w_j^i, p_j^k) \qquad (5.11)
$$

For every (implementable) allocation rule f, we now know how to compute the payments such that the expected total payment is minimized. The question remains which allocation rule f minimizes $P^{\min}(f)$ among all Bayes-Nash implementable and individually rational allocation rules.

Using independence of reported types, $P^{\min}(f)$ can be rewritten as

$$
\begin{aligned}
P^{\min}(f) &= \sum_{j \in J} \sum_{i,k} \varphi(w_j^i, p_j^k) \overline{w_j}^{(i,k)} ES(f, w_j^i, p_j^k)\\
&= \sum_{j \in J} \sum_{t_j \in T_j} \varphi(t_j) \overline{w_j}^{t_j} ES(f, t_j)\\
&= \sum_{j \in J} \sum_{t_j \in T_j} \varphi(t_j) \overline{w_j}^{t_j} \sum_{t_{-j} \in T_{-j}} S_j(f(t_j, t_{-j})) \varphi_{-j}(t_{-j})\\
&= \sum_{j \in J} \sum_{(t_j, t_{-j}) \in T} \varphi(t_j, t_{-j}) \overline{w_j}^{t_j} S_j(f(t_j, t_{-j}))\\
&= \sum_{t \in T} \varphi(t) \sum_{j \in J} \overline{w_j}^{t_j} S_j(f(t)) \qquad (5.12)
\end{aligned}
$$

For the time being, it is convenient to impose the following regularity condition.

Regularity is satisfied if for every job j, for every $i = 1; \ldots, m_j$ and $k = 1, \ldots, q_j$, we have $\overline{w_j}^{(i,k)} < \overline{w_j}^{(i',k)}$ whenever $w_j^{(i,k)} < w_j^{(i',k)}$.

So regularity only makes a claim about weights from types with equal processing time. Note that if f is the allocation rule that schedules jobs in non-increasing order of ratios $\overline{w_j}/p_j$ and if regularity holds, then f satisfies monotonicity with respect to weights and is thus Bayes-Nash implementable, see Theorem 1.

Next, observe that the virtual weights of job j are determined by the shortest paths in $T_j(f)$, the original weights and the probabilities of the types of j. In the single dimensional case, the shortest paths are fixed, independent of the allocation rule f. Consequently, the virtual weights are also fixed and can be computed by the formula in (5.14) below. Remind that in the single dimensional case, the processing times are publicly known, job agents reports the weights. Because of this, we can drop the subscripts for processing time in the single dimensional case.

$$\begin{aligned} \overline{w_j}^1 &= w_j^1 && (5.13) \\ \overline{w_j}^i &= w_j^i + (w_j^i - w_j^{i-1}) \frac{\sum_{h=1}^{i-1} \varphi(w_j^h)}{\varphi(w_j^i)} \\ &= w_j^i + (w_j^i - w_j^{i-1}) \frac{\Phi(w_j^{i-1})}{\varphi(w_j^i)} \quad \text{for } i = 2, \ldots, m_j && (5.14) \end{aligned}$$

Consider the expected total payment, formula (5.12) rewritten for single dimensional case:

$$P^{\min}(f) = \sum_{w \in W} \varphi(w) \sum_{j \in J} \overline{w_j} S_j(f(w)) \tag{5.15}$$

In the single dimensional case, we have $\overline{w_j}^i$ fixed. Hence $P^{\min}(f)$ can be minimized by point wise minimizing $\sum_{j \in J} \overline{w_j} S_j(f(w))$ for every reported type profile w. This is achieved by scheduling the jobs in order of non-increasing ratios $\overline{w_j}/p_j$ (ties can be broken arbitrarily). But we also require that f is Bayes-Nash implementable, and as stated earlier if f is the allocation rule that schedules jobs in non-increasing order of ratios $\overline{w_j}/p_j$ and if regularity holds, then is Bayes-Nash implementable.

Thus the next theorem holds for single dimensional case.

Theorem 2. *[2, 6] With the virtual weights computed as in (5.13),(5.14) and the payments as in (5.5), (5.6), if f is the allocation rule that schedules jobs in non-increasing order of ratios $\overline{w_j}/p_j$ and if regularity holds, then (f, π) is a mechanism that minimizes the expected total costs among all individual rational, Bayes-Nash implementable mechanisms.*

Note that Theorem 2 only applies to single dimensional case. In contrast to the single dimensional case, the shortest paths in the type graph depend on the allocation rule

f in the two-dimensional case. Hence, neither the minimum payments nor the virtual weights can be expressed in a closed formula. Another point is, the allocation rule f as described in Theorem 2 satisfies independence of irrelevant alternatives (IIA), that is the relative order of any two jobs j_1 and j_2 is the same in the schedules $f(s)$ and $f(t)$ for any two type profiles s, $t \in T$ that differ only in the types of jobs from $J \backslash \{j_1, j_2\}$. With the aid of an example it has been shown that the allocation rule of the optimal mechanism for the two dimensional setting does not generally satisfy IIA. For more information see the proof of Theorem 7 in [6].

Ironing Note that the allocation rule of Theorem 2 need to be monotone otherwise it is not Bayes-Nash implementable, see Theorem 1. In general, when computing the virtual weights with formulas (5.13),(5.14), regularity will not be satisfied. To fix this a standard procedure known as 'ironing' can be carried out. Recall that the regularity condition only makes a claim about the virtual weights from types with equal processing time from the same job agent. For the sake of easier notations, the following explanation concerns the single dimensional setting. We will use the same ironing procedure in the two dimensional setting, but it is not sure whether or not the ironing procedure described below is the best way to iron the virtual weights in the two-dimensional cases. For the one-dimensional setting, we will show that it is the best way because it leads to the optimal outcomes.

Consider any non-monotone subsequence of the virtual weights $I_j^{qr} := \{\overline{w_j}^q, \overline{w_j}^{q+1}, \ldots, \overline{w_j}^r\}$, where $\overline{w_j}^q \geq \overline{w_j}^{q+1} \geq \ldots \geq \overline{w_j}^r$ and $\overline{w_j}^q > \overline{w_j}^r$. Let

$$\overline{w_j}^{qr} = \frac{\sum_{i=q}^{r} \varphi(w_j^i)\overline{w_j}^i}{\sum_{i=q}^{r} \varphi(w_j^i)} \tag{5.16}$$

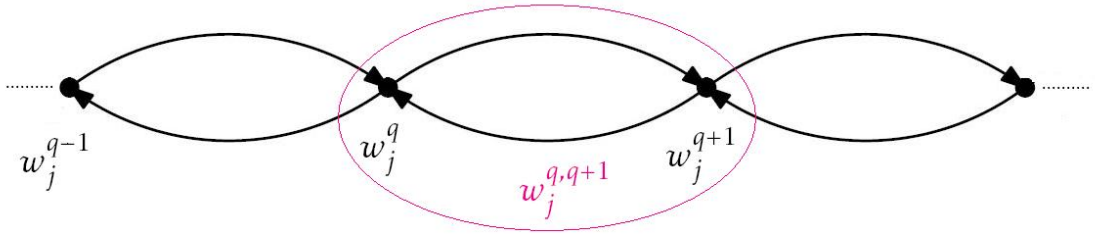
be a new virtual weight that replaces all the virtual weights $\overline{w_j}^i \in I_j^{qr}$. This way we obtain a new monotone allocation rule. In the single dimensional setting, Theorem 2 remains valid after the ironing procedure. To see why, suppose that the minimized expected total payment problem is solved, while the BIC constraints are ignored and as part of our solution we have the next virtual weights for job j

$$\overline{w_j}^1 \leq \ldots \leq \overline{w_j}^q > \overline{w_j}^{q+1} \leq \ldots \leq \overline{w_j}^{m_j}$$

Without loss of generality, assume that the virtual weights of all other jobs (apart from j) satisfy regularity. If not, for each job, whose virtual weights do not satisfy regularity, we just have to repeat the proof below.

If f is the allocation rule that schedules the jobs in the non-increasing order of ratios $\overline{w_j}/p_j$ then the expected starting time of type w_j^q is less than or equal to that of type w_j^{q+1}. If $ES_j(w_j^q) < ES(w_j^{q+1})$, then allocation rule f is not monotone. Adding the BIC constraints will cause the feasible region to shrink and these constraints will be tight. That means $ES_j(w_j^q)$ must equal $ES(w_j^{q+1})$. But then by replacing the two types w_j^q and w_j^{q+1} of job j by one type, say type $w_j^{q,q+1}$ with probability $\varphi(w_j^{q,q+1}) =$

$\varphi(w_j^q)+\varphi(w_j^{q+1})$ and weight w_j^{q+1}, we obtain an adapted problem, which without BIC constraints has the same feasible region as the original problem with BIC constraints.



Using formula (5.14), for the virtual weight of the merged node, we get

$$\overline{w_j}^{q,q+1} = w_j^{q+1} + (w_j^{q+1} - w_j^{q-1})\frac{\Phi(w_j^{q-1})}{\varphi(w_j^q)+\varphi(w_j^{q+1})} \tag{5.17}$$

On the other hand, let us consider the original problem with ironing procedure, we have

$$\overline{w_j}^q = w_j^q + (w_j^q - w_j^{q-1})\frac{\Phi(w_j^{q-1})}{\varphi(w_j^q)} \tag{5.18}$$

$$\overline{w_j}^{q+1} = w_j^{q+1} + (w_j^{q+1} - w_j^q)\frac{\Phi(w_j^{q-1})+\varphi(w_j^q)}{\varphi(w_j^{q+1})} \tag{5.19}$$

Substitute (5.18) and (5.19) in (5.16), we see that after ironing $\overline{w_j}^q$ and $\overline{w_j}^{q+1}$ are replaced by

$$\begin{aligned}
&\frac{\varphi(w_j^q)\overline{w_j}^q + \varphi(w_j^{q+1})\overline{w_j}^{q+1}}{\varphi(w_j^q)+\varphi(w_j^{q+1})} \\
= \;& \frac{\varphi(w_j^q)w_j^q + \varphi(w_j^{q+1})}{\varphi(w_j^q)+\varphi(w_j^{q+1})} + \frac{\cancel{\varphi(w_j^q)}(w_j^q - w_j^{q-1})\Phi(w_j^{q-1})}{\cancel{\varphi(w_j^q)}\left(\varphi(w_j^q)+\varphi(w_j^{q+1})\right)} \\
&+\frac{\cancel{\varphi(w_j^{q+1})}(w_j^{q+1} - w_j^q)\left(\Phi(w_j^{q-1})+\varphi(w_j^q)\right)}{\cancel{\varphi(w_j^{q+1})}\left(\varphi(w_j^q)+\varphi(w_j^{q+1})\right)} \\
= \;& \frac{\varphi(w_j^q)w_j^q + \varphi(w_j^{q+1})}{\varphi(w_j^q)+\varphi(w_j^{q+1})} + \left(w_j^{q+1} - \cancel{w_j^q} + \cancel{w_j^q} - w_j^{q-1}\right)\frac{\Phi(w_j^{q-1})}{\varphi(w_j^q)+\varphi(w_j^{q+1})} \\
&+\frac{\varphi(w_j^q)(w - j^{q+1} - w_j^q)}{\varphi(w_j^q)+\varphi(w_j^{q+1})} \\
= \;& \frac{\cancel{\varphi(w_j^q)+\varphi(w_j^{q+1})}}{\cancel{\varphi(w_j^q)+\varphi(w_j^{q+1})}}w_j^{q+1} + (w_j^{q+1} - w_j^{q-1})\frac{\Phi(w_j^{q-1})}{\varphi(w_j^q)+\varphi(w_j^{q+1})}
\end{aligned} \tag{5.20}$$

which is exactly the expression in (5.17), so Smith's rule based on the ironed virtual weights yields again the optimal mechanism in the single dimensional setting.

In this chapter we answer the sub-questions that have been raised in Chapter 4. In order to exploit and embed the idea that leads to optimal mechanisms in the one-dimensional case, we first need to know how to find and select the shortest paths (Sub-question (I-2)), as well as how to compute the virtual weights (Sub-question (I-3)) for instances in the two-dimensional setting. Sub-question (I-3) was already answered in Section 5.2. Therefore, we here discuss Sub-question (I-2) first.

6.1 Sub-question (I-2) Find and select the shortest paths in the two dimensional setting

6.1.1 Finding the shortest paths

Ignoring the dummy node, the reduced type graph in the two dimensional setting looks like a grid, see Figure 6.1. Let (r, c) denotes the position of the node on row r and column c. So the types with highest processing time are lying in row 1, the types with second largest processing time are lying in row 2, etc.

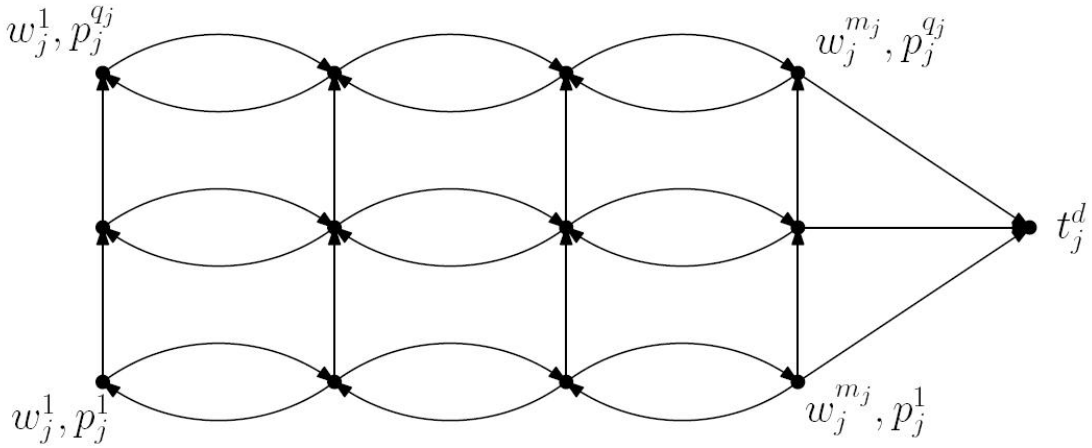


Figure 6.1: Sketch of a reduced type graph

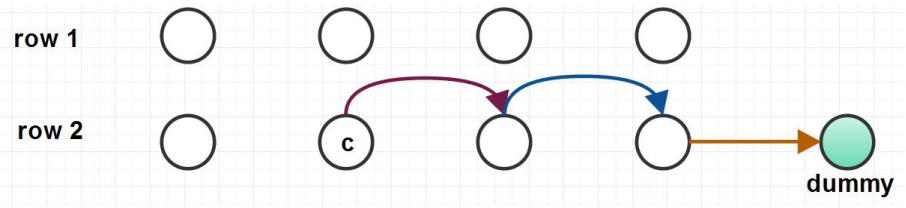
The only vertical arcs are those that go from row r to row $r-1$, with $r = 2, \ldots, q_j$. Looking for shortest paths from a node in row r, we only have to consider the part of the type graph that consists of row 1 to row r. To take advantage of this property, we use dynamic programming for our shortest paths problem.

Recall that if the allocation rule is implementable then there are no negative cycles in the type graph. Thus from every node, there is at least one shortest path to the

dummy node. Beginning with row 1 all shortest paths has the next form: starting in a node, one always takes the arc to the right until the dummy node is reached, just as in the single dimensional setting. For each node on row 1, the corresponding shortest path and its length are saved. Now consider a start node at position $(2, c)$, to go to the dummy node one can either take

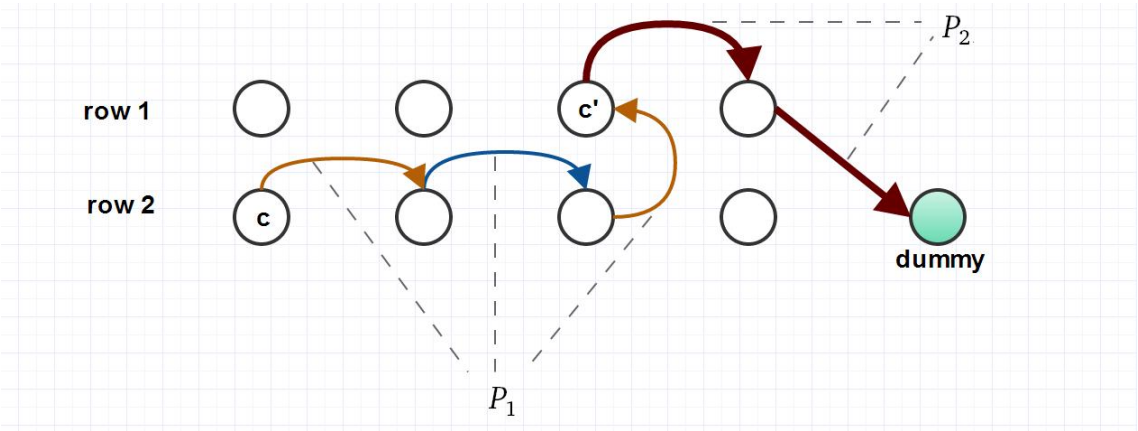
- the path that passes through only nodes in row 2 as in Figure 6.2

Figure 6.2



- or a path P that also passes through nodes in row 1, see Figure 6.3. Let $(1, c')$ be the first node in row 1 that P enters and let divide P into two parts P_1 and P_2, with P_1 the path segment from $(2, c)$ to $(1, c')$ and P_2 the path segment from $(1, c')$ to the dummy node. We want the length of P to be as small as possible, clearly P_2 should coincide with the shortest path from $(1, c')$ to the dummy node, which has been saved earlier. Obviously, shortest paths contain no cycle, i.e. they are simple paths. Because there is only one simple path from $(2, c)$ to $(1, c')$, this path must be taken to be P_1. So the length of P can be computed as the sum of $\ell(P_1)$ and $\ell(P_2)$.

Figure 6.3



Note that $c' \in \{1, \ldots, m_j\}$ so there are m_j such paths P, that should be examined.

Comparing the path's lengths of all paths mentioned above, we obtain for $(2, c)$ the shortest path(s) to the dummy node. These paths and the corresponding length are

then saved. Obviously, the process should be executed for all $c = 1, \ldots, m_j$ and repeated for all $r = 3, \ldots, q_j$. With a start node, lying in row r, length of the path that passes through only nodes in row r and lengths of paths that also passes through nodes in row $r-1$ are being compared. The computation time for each type graph $T_j(f)$ has order $O(|W_j|.|P_j|^2)$.

6.1.2 Selecting the shortest paths

In the single dimensional case, from each node there is only one shortest path to the dummy node . This does not hold for the two-dimensional case, there can be more than one shortest paths from a node to the dummy node. The question is how should we deal with more shortest paths from one node, which path should be chosen? Through an example we explain how the virtual weights are computed when there are several shortest paths with the same start node. Consider the type graph in Figure 6.4, to keep a clear overview only arcs from the shortest paths between a_0 and a_{10} are displayed.

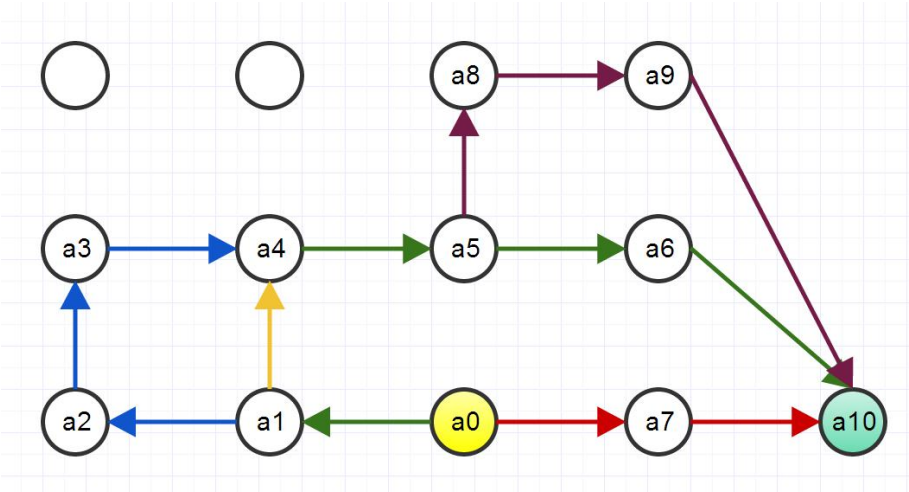


Figure 6.4: Several shortest paths from node a_0 to dummy node a_{10}

As can be seen, there are 5 shortest paths, namely

- $P_1 = [a_0, a_7, a_{10}]$,
- $P_2 = [a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_{10}]$,
- $P_3 = [a_0, a_1, a_4, a_5, a_6, a_{10}]$,
- $P_4 = [a_0, a_1, a_2, a_3, a_4, a_5, a_8, a_9, a_{10}]$,
- $P_5 = [a_0, a_1, a_4, a_5, a_8, a_9, a_{10}]$.

Method 1 The first obvious solution that came to mind is randomly choose one of the five paths and compute the virtual weights contributions as explained before in Section 5.2.

Method 2 Another option is taking into account all five paths, the virtual weights contributions will then be convex combinations of contribution by each individual path. More precisely, let $\alpha_k \in (0,1)$ with $\sum_k \alpha_k = 1$ for $k = 1, \ldots, 5$, then the virtual weights contribution to node a node a in the type graph by these five paths, is

$$\sum_{k=1}^{5} \alpha_k \xi^{P_k}(a) \tag{6.1}$$

Note that the virtual weight contributions computed this way, produce virtual weights that still satisfy the formula for computing expected total payment (5.11):

$$P^{\min}(f) = \sum_{j \in J} \sum_{t_j} \varphi(t_j) \overline{w_j}^{t_j} ES(f, t_j)$$

It remains to determine the values of α_k. Arcs departing from the same junction node can be taken with equal probabilities. For example, starting at node a_0, we take the red arc to a_7 with probability 0.5 and the green arc to a_1 with probability 0.5. Repeating the same argument for junction nodes a_1 and a_5 we obtain the next values

$$\alpha_1 = 1/2, \qquad \alpha_2 = \alpha_3 = \alpha_4 = \alpha_5 = 1/8.$$

Method 3 A slightly different version of the previous method reads as follows. Assume that the dummy node does not belong to any row in the grid of the type graph. Starting in node a_0, there are three ways to leave row 3, namely

- $[a_0, a_7, a_{10}]$,
- $[a_0, a_1, a_4]$,
- $[a_0, a_1, a_2, a_3]$.

we assign a probability of $1/3$ to each of the possible way. Being in node a_5 there are two ways to leave row 2, $[a_5, a_6]$ and $[a_5, a_8, a_{10}]$. We similarly assign probability $1/2$ to each of them. The next values are then obtained.

$$\alpha_1 = 1/3, \qquad \alpha_2 = \alpha_3 = \alpha_4 = \alpha_5 = 1/6.$$

It turns out that for the expected total payment, the three given methods provide similar results. To illustrate this, we give some examples in the next section. Further, it seems more natural to taking into account all shortest paths instead of randomly choosing one. Although all three methods have been implemented, the last one was the most easy to implement, thus in our own algorithm in Chapter 7 the last method will be used.

6.2 Sub-question (I-1) Graph theoretic approach and optimal-IIA mechanism

In this section, we discuss our initial idea for the algorithm. We actually try to mimick the optimal mechanism of the one-dimensional setting. Recall that in the single dimensional setting, the shortest paths are independent of the allocation rule f, so are the virtual weights. Because of this the virtual weights can be expressed in a closed formulae and the expression

$$P^{\min}(f) = \sum_{w \in W} \varphi(w) \sum_{j \in J} \overline{w_j} S_j(f(w))$$

can be minimized by point wise minimizing $\sum_{j \in J} \overline{w_j} S_j(f(w))$ for every reported type profile w. This is done by using Smith's rule with respect to the virtual weights, i.e. the jobs are scheduled in the order of non-increasing $\overline{w_j}/p_j$. In contrast to the single dimensional case, the shortest paths depend the allocation rule f in the two dimensional setting. We therefore need an (Bayes-Nash implementable) initial allocation rule f_0 to begin with. Using Lemma 2 and the answer to Sub-question (I-2), we find the shortest paths and compute the incentive compatible, individually rational payment scheme π^{f_0} that minimizes the expected total payment. After that, we use the answer to Sub-question (I-3) to compute the virtual weights. Next, we schedule the job according to Smith's rule with respects to the ironed virtual weights. The result is a new Bayes-Nash implementable allocation rule f_1 then we repeat the steps (we did for f_0) for the new allocation rule f_1. So we get a sequence of mechanisms as output, with the corresponding allocation rules f_0, f_1, f_2, etc. The question then arises: 'Does this process converge? If so, does it converge into a (local) optimum?' The answer, unfortunately, is 'no'. The process, in general, does not converge, but in all cases it goes into a vicious cycle, that is it alternates between a finite number of mechanisms. Moreover, for the most instances, the corresponding optimal-IIA mechanism is not in the sequence of mechanisms that is output by the algorithm. In the rest of this section we explain our initial idea, described above, in details. We show that it is not a good algorithm and we also give an explanation for its poor performance. Though it is not a good algorithm, it provides us new insight into the problem. Taking into account the new insight, we construct an alternative algorithm, called Algorithm *F&E* that does work well.

The initial algorithm is implemented as follows.

Step 1, Choose an initial allocation rule f_0, f_0 must be implementable, Smith's rule makes a good candidate. So we let f_0 be the allocation rule based on Smith's rule with respect to the original weights w_j. In case of ties, schedule the job with the smaller index first.

Step 2, On the basis of f_0, determine the expected starting times $ES_j(w_j^i, p_j^k)$ and the arc's lengths in the type graphs, see formula (5.1).

Step 3, After that, the shortest paths to the dummy node can be found as described in Section 6.1.1 and the virtual weights can be computed using one of the three proposed methods in Section 6.1.2.

Step 4, The expected payments to job j is then given by formula 5.7. The expected total payment can be computed either as the sum of the expected payments to all job j or with formula 5.11.

Step 5, Next, schedule the jobs according to Smith's rule based on the ironed virtual weights. Let call this scheduling f_1. Clearly, f_1 is implementable.

Step 6, Now, apply step 2 to 5 to allocation rule f_1, the allocation rule we then get is called f_2. Repeat the process for f_2, etc.

In the above description, a stopping criterion is missing. However in our actual implementation we observe that after a finite number of iterations, say k iterations, we get allocation rule f_k, where the corresponding virtual weights are identical to the virtual weights of allocation rule f_m with $m < k$. The value of k depends on the instances, especially the maximum number of types a job has, with other words the size of the largest type graph. If the method, used in step 3 for computing the virtual weights, takes into account all shortest paths (method 2 or method 3) then after iteration k we get into a cycle: f_k coincides with f_m, f_{k+1} coincides with f_{m+1}, ..., $f_{k+(k-m)}$ coincides with f_k, etc. To illustrate all that has been said, we give the next example.

Example 2. We have 3 jobs with $W_1 = \{7, 8, 28\}$, $P_1 = \{8\}$ and $\varphi_1((7,8)) = 0.11844$, $\varphi_1((8,8)) = 0.76385$, $\varphi_1((28,8)) = 0.11771$. In Figure 6.5, job 1's data are shown.

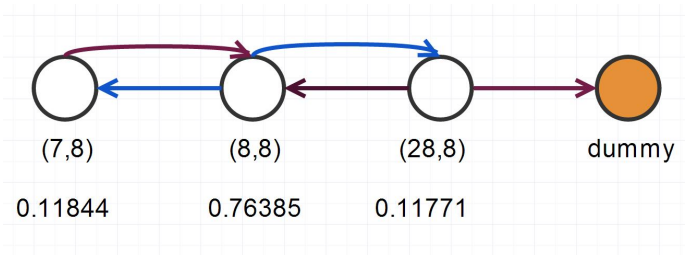
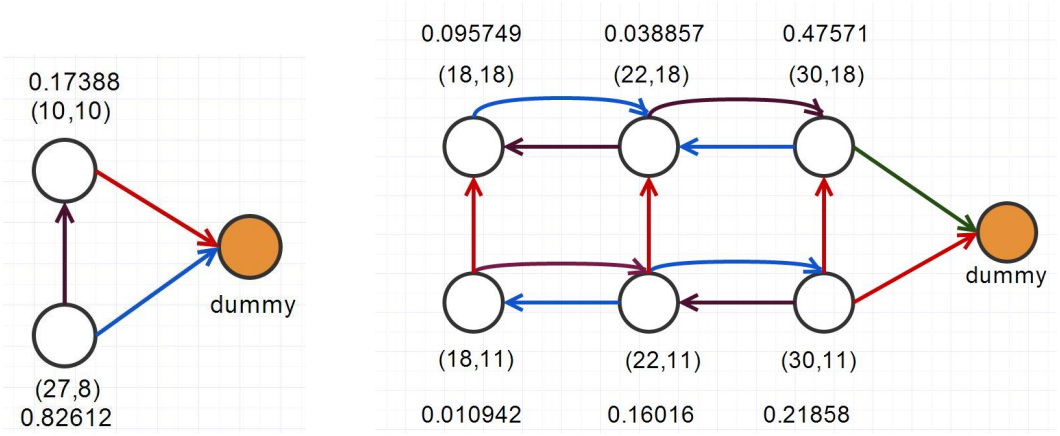


Figure 6.5: Job 1's data

We display the data of job 2 and 3 also in the corresponding type graphs, see Figure 6.6.



(a) Job 2 **(b)** Job 3

Figure 6.6: Job's data

Using Smith's rule as initial allocation rule f_0 we get $P^{min}(f_0) = 366.55$ as expected total payment. We get $P^{min}(f_1) = 374.68$ independent of which method 1, 2 or 3 from Section 6.1.2 is used to compute the virtual weights in step 3. Now if we replace $W_2 = \{10\}$ by $W_2 = \{27\}$, and if other data are unchanged, we have $P^{min}(f_0) = 470.51$ and $P^{min}(f_1) = 470.51$ when method 1 is used, $P^{min}(f_1) = 471.05$ when method 2 is used, $P^{min}(f_1) = 471.05$ when method 3 is used. As can be seen, it does not matter very much which method we use, the resulted expected total payments are of the same orders of magnitude. From now on, we use method 3.

The expected total payments when using method 3 and $W_2 = \{10\}$ are

$$P^{min}(f_0) = 366.55 \quad P^{min}(f_1) = 374.68$$
$$P^{min}(f_2) = 353.85 \quad P^{min}(f_3) = 354.83$$
$$P^{min}(f_4) = 353.85 \quad P^{min}(f_5) = 354.83$$
$$\ldots$$

The virtual weights that determine f_4 are identical to the virtual weights that determine f_2. So we enter the cycle at iteration four and we can let the algorithm stop there. The computation time of these four iterations is 0.14189 seconds. In view of the results, one should process the jobs according to allocation rule f_2.

Recall the LP-model by Hoeksma and Uetz in Chapter 2, in view of inequality 2.2, we can see the expected total payment obtained by the LP-model as a lower bound. Thus we can compare the result from our own algorithm to the lower bound. The relative difference from these two values is an indication of the performance of our algorithm. Besides the LP-model, Hoeksma [1] has implemented an ILP-model that computes the optimal IIA solution for small instances. Using his ILP, we found that the expected total payment provided by the optimal IIA mechanism, equals 353.85.

For this instance, the proposed algorithm seems to work very well, it is fast and it even found the optimal IIA solution.

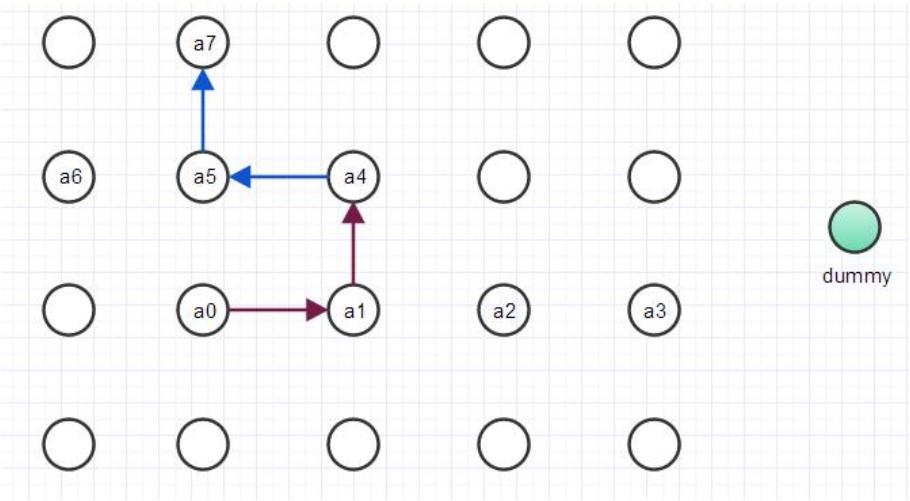
Example 3. In this example, there are 3 jobs as well. Job 1 has 4 types, job 2 has 40 types and job 3 has 42 types, for details see Section 10.1 in the appendix. The expected total payments are then

$$P^{min}(f_0) = 527.76 \quad P^{min}(f_1) = 601.31$$
$$P^{min}(f_2) = 641.14 \quad P^{min}(f_3) = 790.8$$
$$P^{min}(f_4) = 759.06 \quad P^{min}(f_5) = 790.8.$$

The instance in this example is bigger than that from Example 2, in total there are 86 types while in the previous example there are 11 types in total. The computation time is 1.8771 seconds, so the computation time is not a problem. However, the proposed algorithm does not work well in this case. As can be seen, Smith's rule gives the best result. The cause of the poor performance is not the larger number of types. Independent of the number of jobs, from all examples that we have seen, where the type graphs are not bigger than a 2×3 grid, the proposed algorithm works very well:

- the number of iterations is always less than 10,
- the output of the algorithm is a sequence of mechanisms. We take the best mechanism in this sequence, let π^* denotes the corresponding expected total payment. Further, let *Obj(LP)* be the expected total payment computed by the LP-model. Recall that *Obj(LP)* is an lower bound for the optimal expected total payment. The ratio π^* / *Obj(LP)* is less than 1.02 for all instances we have seen, that is the expected total payment found with the algorithm is just 2% more than the lower bound. So in some cases it might be the optimal mechanism that the algorithm has found.

The poor performance is actually related to the ironing process. To see why, consider the next type graph



In constrast to the single dimensional setting, shortest paths do not always go to the right. Suppose that shortest paths passing a_1 always go up to a_4. Note that the weight

corresponding to node a_1 is greater than that of node a_0, so all shortest paths passing the segments $a_0 - a_1 - a_4$ cause the virtual weight of a_1 to increase, as can be seen from the computations in Section 5.2. This way the virtual weights of a_1 can become larger than that of nodes a_2 and a_3, an as a consequence ironing must be carried out.

Similarly, when there are many shortest paths that go through $a_4 - a_5 - a_7$ the virtual weights of node a_5 can be very small, even negative because the weight corresponding to node a_4 is larger than that of a_5. The chance is great that the virtual weight of a_6 is larger than the virtual weight of a_5. Again ironing is necessary.

The larger the type graph the more such scenarios happen. When ironing is carried out to a large extent, many types with the same processing time will get the same expected starting time assigned, independent of weight. This apparently does not make a good allocation rule.

About research question (I), hitherto, we can only say that mimicking the point-wise-minimizing-algorithm of the single dimensional case does not yield a solution that is close to the optimal-IIA mechanism, at least not for reasonably sized instances. For instances where job agents has many types, the proposed algorithm even gives worse result than Smith's rule. What we have learned from this is, in the search for a good mechanism using graph theoretic approach, we should try to avoid ironing as much as possible.

6.3 Sub-question (II-1) Quality test

In the Chapter 7, we will formulate our own algorithm, called Algorithm *F&E* (fast and effective). In Algorithm *F&E* we still make use of the graph theoretic interpretation and the concept of virtual weights, where we take into account the potential unfavorable effect of ironing in the two-dimensional setting. We also use Smith's rule as initial allocation rule. In order to check whether or not the corresponding expected total payment is a good result, we can compare it to the LP-result (or ILP-result for really small instances) just as we have done earlier in Examples 2 and 3 in the previous section. However, when the total number of types becomes really large, more than 1000, then even the LP-model can not handle the problem. In that case, we use the improvement in relation to Smith's rule as performance's measure.

6.4 Sub-question (II-2) Speed test

There is another question to answer, namely 'when can we say that Algorithm *F&E* is fast?'. To answer this, we implement various 'control algorithms', where no virtual weights are involved and the graph theoretic interpretation is used purely for the computation of the payments.

First observe that for a solution (a mechanism) it suffices to give the allocation rule only, because the corresponding payments can be computed using Lemma 2. **The control algorithms** are standard algorithms based on a solution representation that equals to a list of all types of the jobs. In this list the types are arranged in the order of non-decreasing importance. We illustrate such a representation in the next example.

Example 4. Suppose we have an instance with 2 jobs. Job 1 has a type space containing two types: t_1^1 and t_1^2. Job 2 has a type space containing two types: t_2^1 and t_2^2. So in total there 4 types. As an example consider the solution that is represented by the list below

$$\{t_2^2, t_1^1, t_1^2, t_2^1\}$$

if job 1 reports type t_1^2 and job 2 reports type t_2^2 then job 1 is processed first because the types were arranged in the order of non-decreasing importance. Thus the corresponding allocation rule, assigns the following schedules to reported type profiles

$$\begin{array}{ll} [t_1^1, t_2^1] \to 21 & [t_1^1, t_2^2] \to 12 \\ [t_1^2, t_2^1] \to 21 & [t_1^2, t_2^2] \to 12. \end{array}$$

Also note that the list $\{t_2^2, t_1^2, t_1^1, t_2^1\}$, where the order of type t_1^1 and type t_1^2 are different from the earlier list, leads to the same allocation rule. Thus there can be several (list) representations that represent the same allocation rule, therefore equivalently the same mechanism.

We implemented several of such control algorithms: Simple local search, Simulates annealing, Global search. In order to make fair comparisons, we use the next stopping criterion for all control algorithms: 'stop when a expected total payment is found, that is less than or equal to the one we found with Algorithm *F&E*'. So we can compare the computation times needed by the algorithms to find more or less the same result. It might happen that the control algorithms get stuck in a local minimum, that lies above the result of Algorithm *F&E*. For very large instances, it also might happen that a control algorithm does not get stuck in a local minimum, but it is still busy (still finding better results) after a ridiculously long time, but the result is not as good as that of Algorithm *F&E*. For this reason, time limits are being set for the control algorithms(we set the time limits to be 120 seconds for all control algorithms and all instances).

Needless to say, we try to create the best possible control algorithms. For the sake of fairness, Smith's rule is used as initial allocation rule for all algorithms we implemented.

6.4.1 Simple local search

As mentioned above we start with Smith's rule, the corresponding allocation rule is then represented as a list of all types of the jobs. We save this list as the current solution. The goal of the algorithm is to improve the current solution in each iteration. To do that we change the current solution slightly by swapping the order of two types in the current solution. Note that we can not just swap any two types, because the allocation rule, corresponding to the list after the swapping, may no longer be Bayes-Nash implementable. For this reason, we swap only consecutive types that are either types owned by two different job-agents or types with different processing time (if both types are owned by the same agent). If the swapping does not worsen the result, i.e. the expected total payment after the swapping is not greater than the expected total payment of the current solution, then we accept the swapping and update the current solution. As can be seen, we also accept zero-improvements, i.e. swaps that let the expected total payment unchanged. Recall that different lists can represent the same mechanism, so by accepting zero-improvements we will have a broader neighborhood search, which may lead to better results.

Below we decribe the steps of the Simple local search in details.

Step 1, Schedule the jobs according to Smith's rule. Compute the corresponding expected total payment $P^{min}(f_0)$. Set current expected total payment to be $P^{min}(f_c) = P^{min}(f_0)$ and the current allocation rule $f_c = f_0$.

Step 2, Let τ denotes the total number of types, that is $\sum_j |T_j| = \tau$ and let $Z = \{1, 2, \ldots, \tau\}$. Further, $g : T \to Z$ is a one-to-one function, that assigns to each type a number between 1 and τ according to the ratio w_j/p_j. Type with the largest ratio gets number τ assigned, type with the second largest ratio gets $\tau - 1$, etc. In case of ties, breaking ties arbitrary if the types are from the same job, otherwise give the job with smaller index the larger number. The number a type gets assigned, is actually its rank in the list representation (the solution representation in the form of a list explained earlier in this section).

Note that if we now schedule the jobs according to the assigned numbers (types with greater numbers come first), then it is just Smith's rule again, where ties are broken in favor of the job having the smaller index.

Let the blacklist be an empty list. The blacklist helps us avoid unnecessary repetitions. If we know of two types, say A and B, that the corresponding swap is not accepted because it worsen the current expected total payment then in following iterations we do not need to consider the swap of A and B again, until an other swap is accepted and therefore a new mechanism is obtained. In the new mechanism we have to consider the swap of A and B, because we do not know the effect of this swap in the new current solution yet. That is why the blacklist will be emptied when a new mechanism is accepted.

Step 3, Consider all pair of types that satisfy the next conditions

- the types get consecutive numbers assigned,
- the types are from different jobs or from the same job but having different processing time.
- the pair is not in the blacklist.

These first two conditions ensure that regularity is preserved if we exchange the numbers of the types. If no such pair of types exist, stop. Otherwise, we randomly chose such a pair and exchange the corresponding numbers.

Step 4, Schedule the jobs according to the assigned numbers, the type with greater number takes precedence.

- If the corresponding expected total payment is less than or equal to the current value, update f_c and $P^{min}(f_c)$. Empty the blacklist and go to step 5.
- Otherwise, reverse the exchange in the previous step and add the corresponding pair of types in the blacklist. Go to step 5.

Step 5, As long as $P^{min}(f_c)$ is greater than the expected total payment found with Algorithm *F&E* or if the time limit has not been reached yet. Repeat step 3-5.

Steepest descent local search Instead of randomly choosing pair of types in step 3, we can choose a pair that improves the payment the most. Thus for all pairs of types found in step 3, we have to calculate the expected total payment after the corresponding exchange (that is really time consuming however). One exchange in the steepest descent method produces (usually slightly) greater improvement than one exchange in the Simple local search. However with the same amount of time needed for one exchange in the steepest descent method, the Simpel local search can carry out several exchanges and thus several smaller improvements. Considering the net effect, Simple local search is a better choice. Moreover, we have seen in many experiments that steepest descent method, after just a few iterations, gets stuck in some local minimum, that is way above the expected total payment found with Algorithm *F&E*.

6.4.2 Simulated annealing

Due to the fact that the Simple local search never accepts worse solutions, it might happen that the algorithm (shortly after the start) gets stuck in some local minimum. This problem can be overcome by using the simulated annealing search framework. Simulated annealing is a probabilistic method proposed in Kirkpatrick, Gelett and Vecchi (1983) [14] and Cerny (1985) [3] for finding the global minimum of a cost function that may possess several local minima. The method derives its name from the annealing process used to recrystallize metals. Initially when the metal is heated to high temperatures, the atoms have lots of space to move around. Slowly when the temperature is reduced the movement of free atoms are slowly reduced and finally

the metals crystallize themselves. As the metal cools its new structure becomes fixed, consequently causing the metal to retain its newly obtained properties. Simulated annealing is analogous to this annealing process. We keep a temperature variable to simulate the heating process. We initially set it high and then allow it to slowly cool as the algorithm runs. While this temperature variable is high the algorithm will be allowed, with more frequency, to accept solutions that are worse than our current solution. This gives the algorithm the ability to jump out of any local optimum at the initial stage of the execution. As the temperature is reduced so is the chance of accepting worse solutions, therefore allowing the algorithm to gradually focus in on an area of the search space in which hopefully, a close to optimum solution can be found. This gradual cooling process is what makes the Simulated annealing algorithm remarkably effective at finding a close to optimum solution when dealing with large problems which contain numerous local optima.

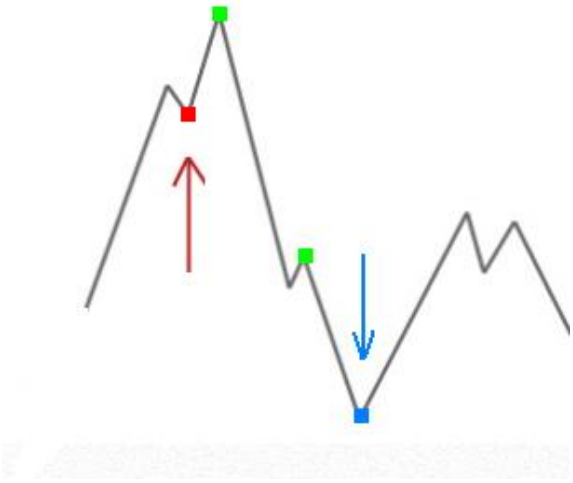


Figure 6.7: Simulated annealing

To help better understand, consider Figure 6.7. Suppose the red dot represents our current solution. Using the Simpel local search as described in Section 6.4.1, we will stay in this solution. In this example we can clearly see that it is stuck in a local optimum. In the mechanism design problem (or in any other real world problems), we would not know how the search space looks. So unfortunately we would not be able to tell whether this solution is anywhere close to a global optimum. By occasionally accepting worse solutions, Simulated annealing helps us to jump out of local optima we might have otherwise got stuck in. In this example Simulated annealing may helps us pass through the local minima, corresponding to the green dots and reach a better solution, corresponding to the blue dot.

The implementation of Simulated annealing is for the most part the same as the implementation of the Simple local search algorithm. The only differences are

- Simulated annealing has an additional cooling schedule,

- and the content of step 4, where a potential new solution is accepted.

The cooling schedule is a non-increasing function $T : \mathbb{N} \to (0, \infty)$, $T(i)$ is called the temperature at the i^{th}-iteration. The temperature in our implementation decrease per 100 iterations, each time it decreases by 5%.

Step 4 is modified as follows: schedule the jobs according to the numbers, assigned to the types.

- If the corresponding expected total payment $P^{min}(f_{new})$ is less than or equal to the current value, update f_c and $P^{min}(f_c) = P^{min}(f_{new})$. Empty the blacklist and go to step 5.
- Otherwise, compute the difference $\Delta = P^{min}(f_{new}) - P^{min}(f_c)$. With probability ρ, see formula 6.2, accept the exchange, update f_c and $P^{min}(f_c) = P^{min}(f_{new})$. Empty the blacklist and go to step 5. With probability $1-\rho$ reverse the exchange and add the corresponding pair of types in the blacklist. Go to step 5.

$$\rho = \exp^{-\Delta/T}, \text{ where } T \text{ is the temperature at that moment.} \tag{6.2}$$

It remains to determine the initial temperature.

- Schedule the jobs according to Smith's rule. Compute the corresponding expected total payment $P^{min}(f_0)$. Set current expected total payment to be $P^{min}(f_c) = P^{min}(f_0)$.
- Carry out randomly 10 exchanges, recall that τ is the total number of types. For each exchange, compute the corresponding expected total payment $P^{min}(f_{new})$ and the difference $\Delta = P^{min}(f_{new}) - P^{min}(f_c)$. After that, we have 10 Δ-values. These values do not differ very much from each others. This is not surprising because by exchanging the (consecutive) numbers of just one pair of types, the change in allocation rule is minimal. Due to the small differences, we choose for just 10 random exchanges. For the moment, suppose that there is at least a positive Δ. Initial temperature is chosen such that if after Smith's rule, the exchange with the most positive Δ, say Δ^*, happens, we accept the exchange with probability 0.5 and reverse (reject) the exchange with probability 0.5. That is $T(0) = \Delta^* / \ln(2)$.

  We have assumed above that there is at least a positive Δ. In all our experiments this was also the case but for the completeness, return an error message otherwise.

6.4.3 Global search

Recall the swapping procedure in the Simple local search algorithm, after each iteration at most two types have swapped places in the solution representation list. So

if we consider two mechanisms that are the outputs of two consecutive iterations, there will not be a big difference between these two mechanisms with respect to the expected total payments. Hence, if any strict improvement occurs after an iteration, it is small. The same arguments hold for the Simulated annealing algorithm. Further, swapping costs almost no computation time, most of the work is the computation of the effect of a swap, i.e. the computation of the expected total payment after two types swap places in the solution representation list. So the it may be advantageous to adjust the Simple local search algorithm as follows. We repeat the swapping procedure several times within an iteration, to obtain a very different new (Bayes-Nash implementable) allocation rule. We do not care about the effect of individual swap but the net effect of all swaps. If all swaps together do not cause the current expected total payment to increase then we accept all the swaps at once. We call this adjusted version of Simple local search, Global search.

The implementation of the Global search algorithm is for the most part the same as the implementation of the Simple local search algorithm. The only differences are

- we leave out the blacklist, because the probability that the same group of swaps happen at different iterations is nil.
- step 3 is modified as follows: let τ denotes the total number of types, repeat $\tau/2$ times the process below.

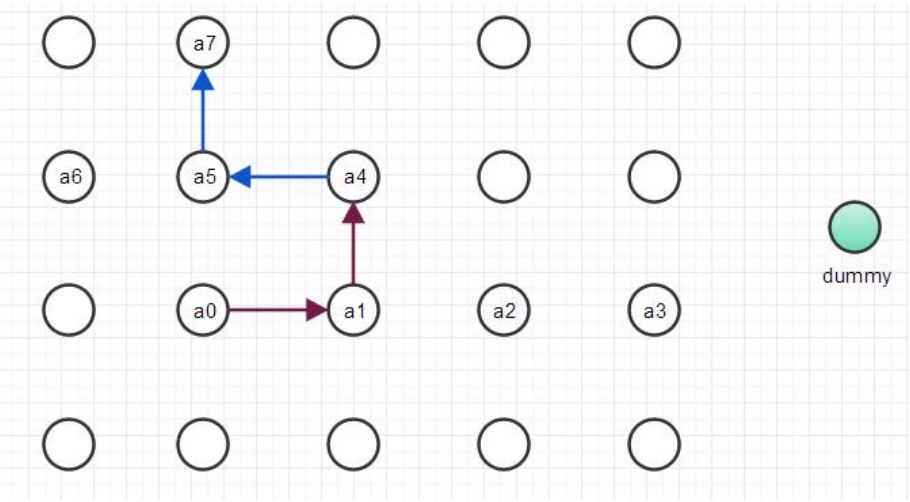
  Consider all pair of types that satisfy the next conditions

  - the types get consecutive numbers assigned.
  - the types are from different jobs or from the same job but having different processing time.

  If no such pair of types exist, stop. Otherwise, we randomly chose such a pair and exchange the corresponding numbers.

First of all, recall our initial idea discussed in Section 6.2 , where we start wit the allocation rule f_0, which is based on Smith's rule with respect to the original weights w_j. Then the incentive compatible, individually rational payment scheme π^{f_0} that minimizes the expected total payment, can be computed using the shortest paths in the type graphs $T_j(f_0)$, $j \in J$. The virtual weights are also computed using these shortest paths. Next, we schedule the job according to Smith's rule with respects to the ironed virtual weights, the corresponding allocation rule is f_1. The arc's lengths in the type graphs $T_j(f_1)$ are in general not the same as the arc's lengths in the type graphs $T_j(f_0)$. Therefore, shortest paths in $T_j(f_1)$ are different from the ones found in $T_j(f_0)$. That the shortest paths change, cause the virtual weights to change too. So we get new allocation rule f_2, etc. This process results in a sequence of mechanisms with corresponding allocation rules f_0, f_1, f_2, ... From the implementation we observed that the sequence of output mechanisms consists of a finite number of different mechanisms, namely the algorithm eventually goes into a vicious cycle, i.e. it alternates between a few mechanisms. From all output mechanisms, we of course choose the best one. It turned out that the initial mechanism with allocation rule based on Smith's rule with respect to the original weights w_j is the best mechanism (amongst all mechanisms that have been output) most of the time, for reasonably seized instances. For these instances, it is pointless to run the algorithm, it takes time and the result by Smith's rule is not improved. This poor performance can be attributed to the potential unfavorable effect of ironing (of the virtual weights) in the two-dimensional setting. Which in turn is caused by the fact that the shortest paths in the two-dimensional type graphs do not always go to the right until the dummy node is reached as in the one-dimensional setting, as stated in Section 6.2. For example, the path segment $a_0 - a_1 - a_4 - a_5 - a_7$ in Figure 7.1 does not has the configuration 'only go to the right until the dummy node'.

Figure 7.1



This leads us to the next idea. For the computing of the virtual weights, we consider only shortest paths that go to the right until the dummy node is reached (other shortest paths are ignored). In the implement of this new idea, we observe that in the sequence of output mechanisms, there is at least one mechanism that is significantly better than the initial mechanism f_0, for most of the instances. This outcome gives rise to the question 'what if we pretend as if all shortest paths has the same configuration, they all go to the right until the dummy node is reached and we compute the virtual weights based on these imaginary shortest paths?'. The answer is the idea of imaginary shortest paths gives even better results than the previous idea, where we ignored shortest paths that do not have the configuration 'only go to the right until the dummy node'. Because the imaginary shortest paths are fixed, the corresponding virtual weights are also fixed and are given by

$$\widetilde{w_j}^{(1,k)} = w_j^{(1,k)} \quad \text{for } k = 1, \ldots, q_j \tag{7.1}$$

$$\widetilde{w_j}^{(i,k)} = w_j^{(i,k)} + \left(w_j^{(i,k)} - w_j^{(i-1,k)}\right) \frac{\sum_{h=1}^{i-1} \varphi((w_j^h, p_j^k))}{\varphi((w_j^i, p_j^k))} \tag{7.2}$$
$$\text{for } i = 2, \ldots, m_j \text{ and } k = 1, \ldots, q_j.$$

In Example 5 at the end of this chapter, we give an intuitive explanation why this works so well.

Now, we can use $\widetilde{w_j}^{(i,k)}$ as virtual weights and schedule the jobs according to Smith's rule based on ironed virtual weights $\widetilde{w_j}^{(i,k)}$. This does produce results that are significantly better than the results by Smith's rule. However, through experimental research we have seen that the algorithm can even be improved further by using not only the imaginary shortest paths (which lead to fixed $\widetilde{w_j}^{(i,k)}$) but using a combination of the imaginary shortest paths and the real shortest paths (found in the type graphs). The Algorithm *F&E* then reads as follows.

Algorithm *F&E*

Step 1, Compute $\widetilde{w_j}^{(i,k)}$ using the formulas in 7.1 and 7.2.

Step 2, Let f be allocation rule based on Smith's rule based with respect to the the original weights w_j, find the shortest paths in the the corresponding type graphs.

Step 3, Get rid of all shortest paths that only go to right until the dummy node is reached, the contributions to the virtual weights by these paths are already included in $\widetilde{w_j}^{(i,k)}$.

Step 3, With the remaining shortest paths, compute the corresponding $\overline{w_j}^{(i,k)}$ as explained in Section 5.2 and let

$$\widehat{w_j}^{(i,k)} = \widetilde{w_j}^{(i,k)} + \alpha.\overline{w_j}^{(i,k)} \quad \text{for } i = 2, \ldots, m_j \text{ and } k = 1, \ldots, q_j. \tag{7.3}$$

We wish to schedule the jobs according to Smith's rule based on the ironed weights $\widehat{w_j}^{(i,k)}$. To do so, we need to determine the value of α. Observe that step 3 ensures that for the single dimensional instances, we always get the optimal mechanisms, independent of α. For the two-dimensional instances, we know that α should not be too large due to the potential unfavorable effect of ironing.

To get a feeling what value α should be, we compute the expected total payment that corresponds to the allocation rule based on Smith's rule with respected to the ironed weights $\widehat{w_j}^{(i,k)}$, where $\widehat{w_j}^{(i,k)}$ are computed using different values of α: α between 0 and 1, with a step size of 0.1 for several instances. The expected total payments are then plotted against α, see appendix Section 10.2 for the plots. Considering those plots, we reduce the range of α to $[0.2, 0.6]$. We let the step size be 0.05. It makes little sense to make the step size much smaller, because doing so we obtain many sets of virtual weights that slightly differ from each other but the chance is great that several sets produce the same allocation rule.

In short, for each $\alpha \in \{0.2, 0.25, 0.3, \ldots, 0.55, 0.6\}$, Algorithm *F&E* computes the corresponding $\widehat{w_j}^{(i,k)}$ with formula 7.3, and it also computes the corresponding expected total payment using Lemma 2. Thus for each value of α we have a corresponding expected total payment. From the obtained results, the best value of α and is chosen and the corresponding mechanism will be output.

In the example below, we give an intuitive explanation why the idea of imaginary shortest paths that all has the form 'go to the right until the dummy node', works so well.

Example 5. Suppose there are 2 job agents, with $W_1 = \{3, 12\}$, $W_2\{14\}$, $P_1 = P_2 = \{6\}$ and $\varphi((3,6)) = 0.9$, $\varphi((12,6)) = 0.1$, $\varphi((14,6)) = 1$.

Smith's rule	Optimal rule
	$\overline{w_1}^{(3,6)} = 3$ $\overline{w_1}^{(12,6)} = 12 + (12-3) * (0.9/0.1) = 93$ $\overline{w_2}^{(14,6)} = 14$
$ES_1((3,6)) = 6$ $ES_1((12,6)) = 6$ $ES_2((14,6)) = 0.6$	$ES_1((3,6)) = 6$ $ES_1((12,6)) = 0$ $ES_2((14,6)) = 0$
See Figure 7.2 for the type graphs $\pi_1((3,6)) = 72$ $\pi_1((12,6)) = 72$ $\pi_2((14,6)) = 0$	See Figure 7.3 for the type graphs $\pi_1((3,6)) = 18$ $\pi_1((12,6)) = 0$ $\pi_2((14,6)) = 8.4$
$E\pi_1 = 72$ $E\pi_2 = 0$ $P^{min}(f) = 72$	$E\pi_1 = 18 \times 0.9 + 0 \times 0.1 = 16.2$ $E\pi_2 = 8.4$ $P^{min}(f) = 24.6$

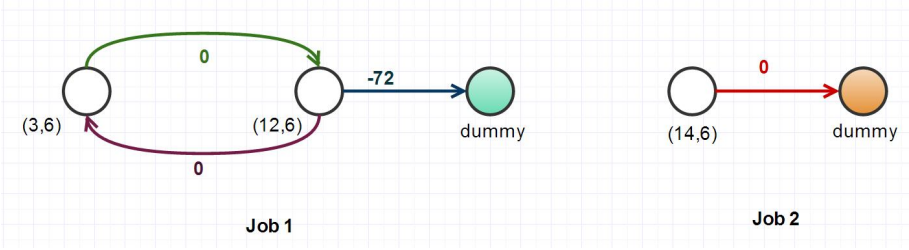


Figure 7.2: Type graphs by Smith's rule

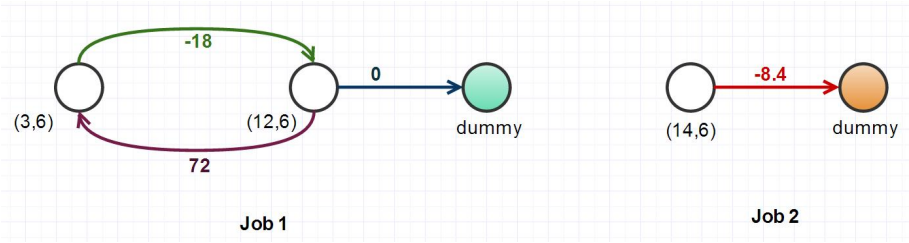


Figure 7.3: Type graphs by optimal rule

Comparing the optimal allocation rule to Smith's rule, we see that the expected starting time of type $(12,6)$ of job 1, is decreased by 6 at the expense of the expected starting time of type $(14,6)$ of job 2. Due to the small probability of type $(12,6)$, the expected starting time of type $(14,6)$ is increased by just 0.6. This is the reason why the decline in expected payment to job 1 is much more than the increase in expected payment to job 2. If the probability of type $(12,6)$ was greater than or equal to $9/11$ then the virtual weight of $(12,6)$ will be less than or equal to 14. That means optimal rule coincides with Smith's rule.

It therefore seems beneficial to process types with small probability first, especially types that are lying on the 'right-hand side' of the type graphs because regularity still needs to be satisfied. Scheduling based on Smith's rule with respect to the weights $\widetilde{w}_j^{(i,k)}$, as defined in formulas 7.1, 7.2, create the mentioned (desired) effect.

8 Results and discussion

This chapter presents the results and discussion of Algorithm *F&E* described in the previous chapter. Chapter sections include first the description of test instances (Section 8.1), that are used for the validation of Algorithm *F&E*. The experiments are divided into three classes based on the sizes of the instances, the class of small instances (Section 8.2), the class of large instances (Section 8.3) and the class huge instances (Section 8.4).

Further, the validation is divided into two parts: the quality part and the speed part. That is for each instance we check whether or not the solution (mechanism) found is a good mechanism and whether or not Algorithm *F&E* found the solution fast?

For the quality validation, we compare the result of *F&E* to

- the ILP-result for really small instances.
- the LP-result for large instances. The instances are now too large for the ILP-model to handle, the model will not terminate in acceptable time.
- the result by Smith's rule for huge instances. Theoretically, the LP-model can solve the problem in polynomial time. However the descriptions of the instances become so large such that so much memory is needed. As a result, the solver returns a memory error.

Other relevant information are: both the ILP [1] and LP-model [1] are implemented using programming language Python(version 3.2) and the solver Gurobi(version 5.2.6). Algorithm *F&E* is implemented in Matlab(version 2013b). All experiments were carried out on the system with

Operating system: Microsoft Windows XP.
Processor: Intel(R) Core(TM)2 Duo CPU T9300 @ 2.50GHz(2CPUs).
Memory: 3072MB RAM.

For the speed validation, we compare the computation times of Algorithm *F&E* to the computation times of the three control algorithms described earlier in Section 6.4, namely Simple local search, Simulated annealing and Global search. These comparisons of computation times are done for small and large instances. For the huge instances, we only report the computation times of Algorithm *F&E* because after the experiments with small and large instances, it can clearly be seen that all three control algorithms are no match for Algorithm *F&E*.

8.1 Description of instances

As mentioned before, we divide our experiments into three classes:

1. experiments for small scale instances;
2. experiments for large scale instances;
3. experiments for huge instances.

For the first class of experiments we have:

- 20 instances, each with 5 jobs, each jobs can have at most 3 different weights and at most 3 different processing times.
- 20 instances, each with 10 jobs, each jobs can have at most 3 different weights and at most 3 different processing times.

For the second class of experiments we generate

- 20 instances, each with 5 jobs, each jobs can have at most 10 different weights and at most 10 different processing times.
- 20 instances, each with 10 jobs, each jobs can have at most 10 different weights and at most 10 different processing times.
- 20 instances, each with 20 jobs, each jobs can have at most 10 different weights and at most 10 different processing times.

Finally, we have for the huge instances

- 20 instances, each with 100 jobs, each jobs can have at most 10 different weights and at most 10 different processing times.

Recall that the type of a job is represented by $t_j := (w_j, p_j) \in T_j := W_j \times P_j$, where $W_j := \{w_j^1, \ldots, w_j^{m_j}\}$ with $w_j^1 < \ldots < w_j^{m_j}$, and $P_j = \{p_j^1, \ldots, p_j^{q_j}\}$ with $p_j^1 < \ldots < p_j^{q_j}$. We generate the types by generating set W_j and P_j as follows.

Let $\mathscr{W}_j$ denotes the cardinality of W_j and $\mathscr{P}_j$ denotes the cardinality of P_j. For the instances, where each jobs can have at most 3 different weights and at most 3 different processing times, $\mathscr{W}_j$ and $\mathscr{P}_j$ are random integers from the interval $[1,3]$. For the remaining instances, where each jobs can have at most 10 different weights and at most 10 different processing times, $\mathscr{W}_j$ and $\mathscr{P}_j$ are random integers from the interval $[1,10]$.

For W_j we generate a vector containing $\mathscr{W}_j$ unique integers selected randomly from 1 to 20 inclusive, then we sort these unique integers in the increasing order. Analogously, for P_j we generate a vector containing $\mathscr{P}_j$ unique integers selected randomly from 1 to 20 inclusive, then these unique integers are sorted in the increasing order.

The probabilities of types, from a job, is uniform distributed in the sense that it has the conditional probability distribution of a uniform distribution, given that the sum of the probabilities is 1.

8.2 Experiments for small scale instances

As mentioned before, we have two sorts of test: quality tests and speed tests. For the quality tests, we compare the expected total payment computed by Algorithm *F&E* to the IIA-optimal expected total payment computed by the ILP-model [1]. More precisely, we compute the ratio of the two mentioned values:

$$\frac{Obj(F\&E)}{Obj(ILP)} \tag{8.1}$$

where *Obj(F&E)* is the objective value found by Algorithm *F&E* and *Obj(ILP)* is the objective value found by the ILP-model. Recall that Algorithm *F&E* output a mechanism as solution, Obj(F&E) is the expected total payment corresponds to that mechanism. *Obj(ILP)* is defined similarly.

As regards the speed test, we compare the computation time of Algorithm *F&E* with the computation times of the three control algorithms: Simple local search, Simulated annealing and Global search. In order to make fair comparisons, we want to compare the computation times needed by the algorithms to find more or less the same result. Therefore, we let the control algorithms stop at the moment a expected total payment is found that is less than or equal to the objective value of Algorithm *F&E*. An alternative stopping criterion could be: stop if a expected total payment is found that differs at most, for example 2% from the objective value of Algorithm *F&E*. But it does not matter much which criterion we used, so we choose the first criterion.

Note that it might happen that one or more control algorithms fail to find a expected total payment that is less than or equal to the objective value of Algorithm *F&E*. From the experiments, we observe that the Simple local search algorithm is usually the one that fails. The next instance is an example where the local search get stuck in a local minimum, that 'lies above' the objective value of Algorithm *F&E*.

Table 8.1: Example 6

Instance data		Algorithms	Obj value	Running time
Number of jobs	10	Smith	4391.976	
Max size	3x3	ILP	3653.256	
Number of types	43	LP	3653.256	
		F&E	3740.225	0.808511
		Global Search	3735.992	14.46086
		Local Search	3815.187	120.0216
		Simulated Annealing	3732.465	30.42383

Example 6. This is an instance of 10 jobs, where $|W_j| \leq 3$ and $|P_j| \leq 3$ for all $j = 1, \ldots, 10$. In total the jobs have 43 types together, that is $\sum_j |T_j| = 43$.

While the Global search finds the desired expected payment (and the corresponding mechanism) after 14.46 seconds and Simulated annealing after 30.42 seconds, the

Simple local search terminates after 120.02 seconds because the time limit of 120 seconds is exceeded. Considering the big differences in finish times of the local search and the other two control algorithms, it is highly likely that the expected total payment of 3815.187 belongs to a local minimum. We then increase the time limit of 120 seconds to 240 seconds, and let the Simple local search continue from the last found solution of 3815.187. We see that the objective value does not change after the extra time and because of the small size of the instance, we conclude that 3815.187 is a local minimum.

In order to be able to compare, we use only instances where all three control algorithms manage to find a expected total payment that is less than or equal to the objective value of Algorithm *F&E*, within the time limit of 120 seconds. Instances as in Example 6 will be discarded.

In Example 7 we give a typical small scale instance (used in this class of experiments) and the corresponding results.

Table 8.2: Example 7

Instance data		Algorithms	Obj value	Running time
Number of jobs	10	Smith	3017.512	
Max size	3x3	ILP	2300.84	
Number of types	37	LP	2300.678	
		F&E	2321.172	0.706006
		GS	2320.2	17.34417
		LS	2320.944	20.77391
		SA	2321.028	15.6872

Example 7. The objective value of the ILP-model is the expected total payment of an optimal-IIA mechanism. The objective value of the LP-model is the expected total payment of an optimal randomized mechanism. As stated earlier, the objective value of the LP-model is less than equal to that of the ILP-model. In this example, the mentioned inequality happens to be strict.

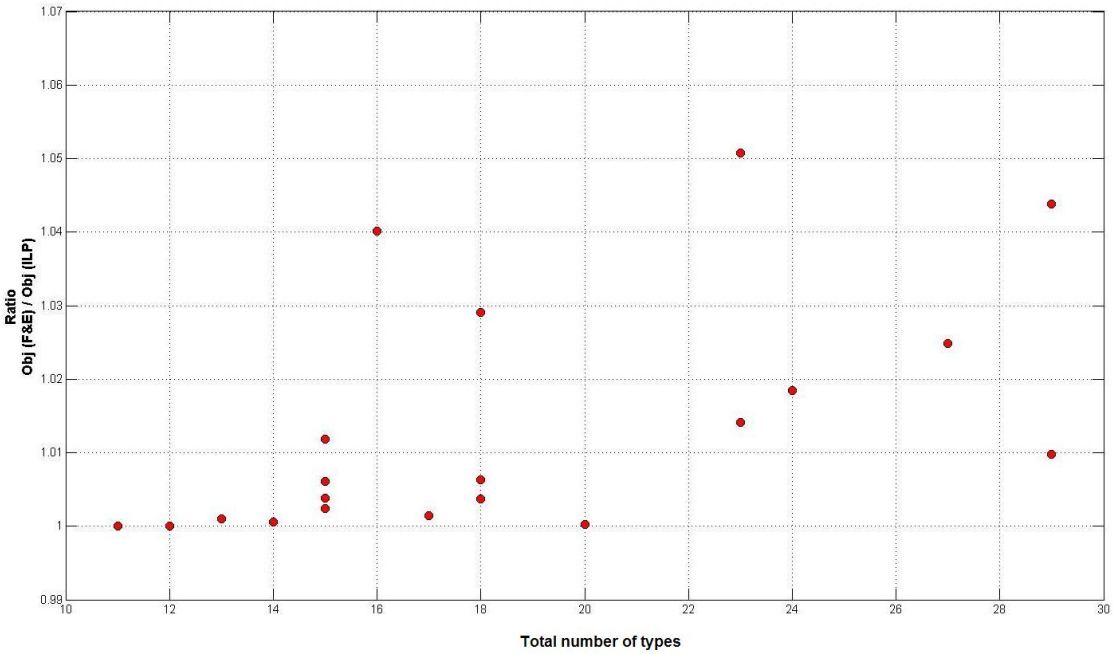
Because all mechanisms obtained by Algorithm *F&E* are deterministic and satisfy IIA condition, the ILP-model gives us tighter lower bound. In this class of experiments, we therefore use the results provided by the ILP-model.

8.2.1 Quality test

We test the quality of the outcome of Algorithm *F&E* by computing the ratio in formula 8.1 for each test instance. To give a clear view of the results, the ratios are then plotted against the corresponding total number of types. Below we describe these steps more precisely.

Recall that we have 20 instances, each with 5 jobs, each jobs can have at most 3 different weights and at most 3 different processing times. We arrange the instances in the order of increasing total number of types. In figure 8.1 the ratios computed by formula (8.1) are plotted against the total number of types. Take for example the point with coordinates $(20, 1)$, this belongs to an instance where $\sum_{j=1}^{5}|T_j| = 20$ and ratio 1 means Algorithm *F&E* finds the optimal-IIA mechanism.

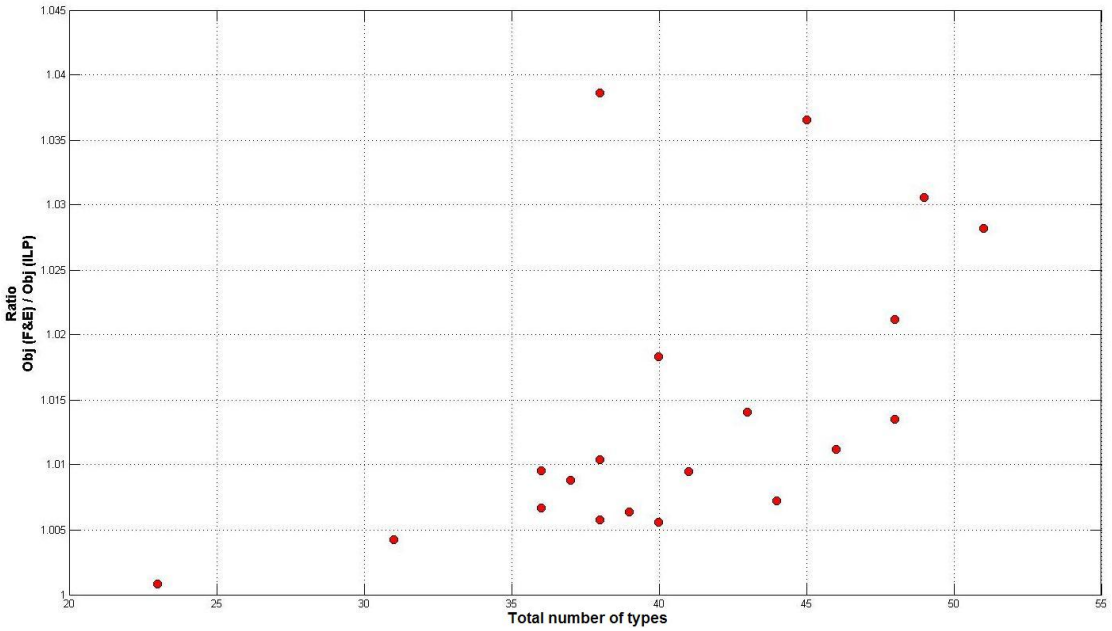
Figure 8.1: Ratios *F&E*/ optimal IIA of 20 small scale instances, each with 5 jobs



So, for the instances with 11, 12 and 20 types Algorithm *F&E* finds the optimal-IIA mechanisms. For 8 other instances the ratios are less than 1.01. So for those instances the expected total payments by Algorithm *F&E* are not more than 1% above the corresponding optimal-IIA expected total payments. The instance with the highest ratio has in total 23 types and the ratio is 1.051, which is still a very good result because the expected total payments by Algorithm *F&E* is just 5.1% above the optimal-IIA .

Analogously, we obtain figure 8.2 for the other 20 instances, each with 10 jobs, each jobs can have at most 3 different weights and at most 3 different processing times.

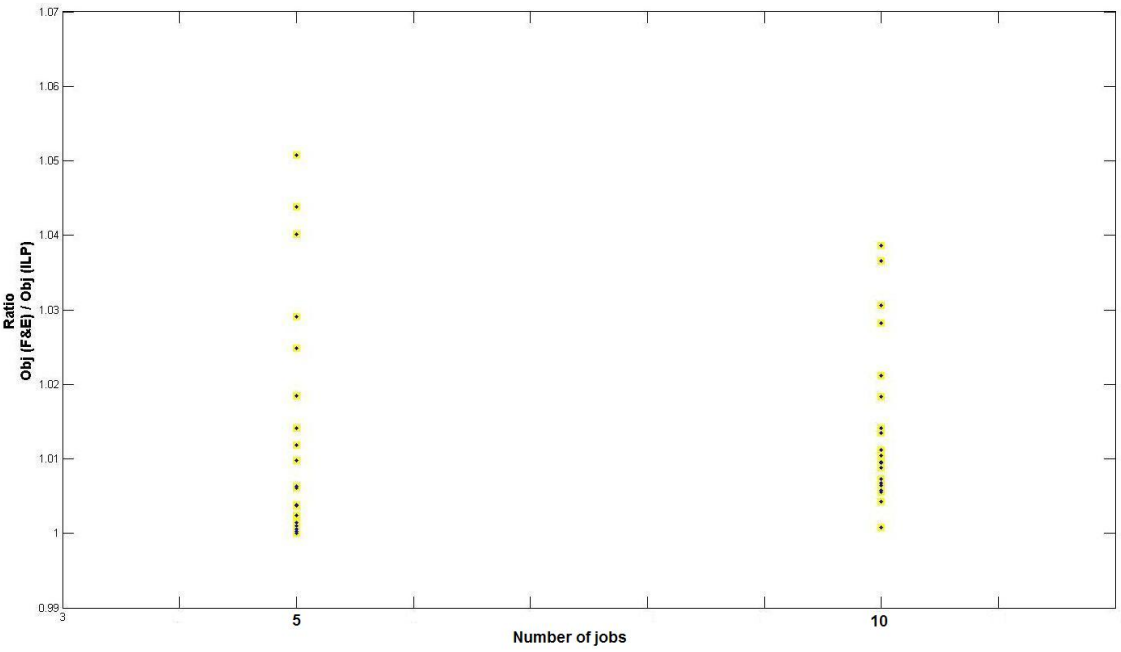
Figure 8.2: Ratio *F&E*/ optimal IIA of 20 small scale instances, each with 10 jobs



For 10 instances (out of the 20 instances) the corresponding ratios are less than 1.01, so the expected total payments by Algorithm *F&E* are not more than 1% above the corresponding optimal-IIA expected total payments for half of the instances. The instance with the highest ratio has in total 38 types and the ratio is 1.038.

Finally, we put the ratios of all instances with the same number of jobs under each other in a one-dimensional plot, see figure 8.3

Figure 8.3: Ratio *Obj(F&E)*/ optimal IIA



As can be seen, in the worst case, the payment given by Algorithm *F&E* is about 5% higher than the optimal IIA payment. Approximately for 50% of the 40 instances, Algorithm *F&E* gives a payment that is at most 1% higher than the optimal IIA payment. In short, Algorithm *F&E* gives solutions of excellent quality for small test instances.

8.2.2 Speed test

To check whether Algorithm *F&E* is a fast algorithm. We compare the computation times needed by the algorithms to find more or less the same result. The control algorithms stop at the moment a expected total payment is found that is less than or equal to the objective value of Algorithm *F&E*. As in the quality test, we arrange the instances in the order of increasing total number of types.

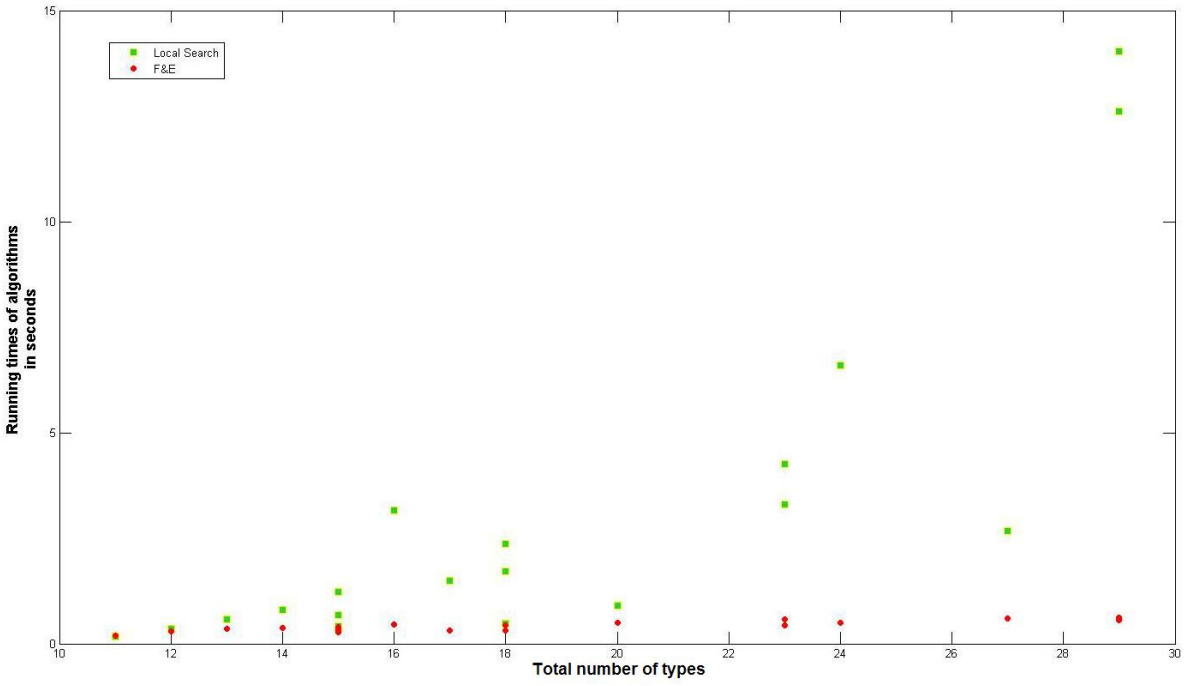
For instances with 5 jobs:

- the computation times of Simple local search and Algorithm *F*&*E* are plotted against the total number of types in figure 8.4. The green squares represent the computation times of Simple local search, the red dots represent the computatuon times of Algorithm *F*&*E*.
- The computation times of Simulated annealing and Algorithm *F*&*E* are plotted against the total number of types in figure 8.5. The cyan triangles represent

the times of Simulated annealing, the red dots represent the times of Algorithm $F\&E$.

- the computation times of Global search and Algorithm $F\&E$ are plotted against the total number of types in figure 8.6. The black asterisks represent the times of Global search, the red dots represent the times of Algorithm $F\&E$.

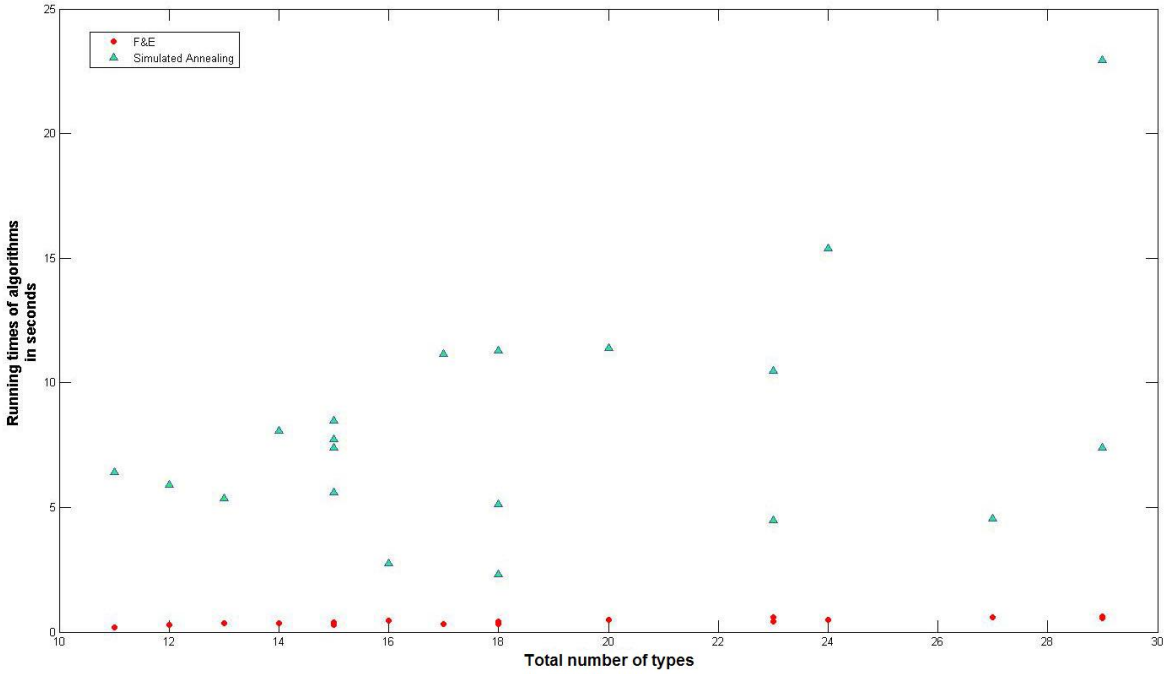
Figure 8.4: Computation times of Simple local search and $F\&E$ for instances with 5 jobs



Because there are 20 instances with 5 jobs, there are 20 red dots and 20 green squares in the plot. Take for example the points with coordinates $(27, 0.6)$ and $(27, 2.7)$. These two points belong to an instance that has in total 27 types, the computation time of Algorithm *F&E* is 0.6 seconds. With a computation time of 2.7 seconds Simple local search finds a expected total payment that is less than or equal the expected total payment given by Algorithm *F&E*.

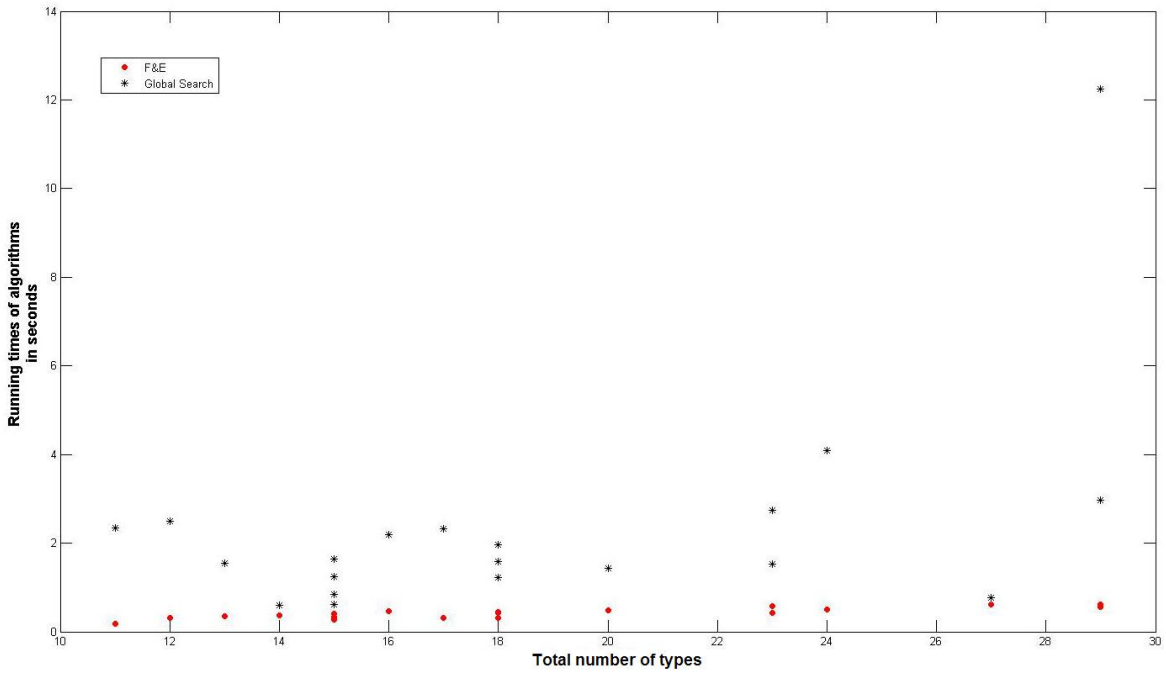
For all instances Algorithm *F&E* has smaller computation times. As the number of types grows, the differences between the computation times of Algorithm *F&E* and the computation times of Simple local search grows strongly. For the instance with 11 types, the computation times of both algorithms are more or less the same. However for the two largest instances, both have in total 29 types, for one instance the computation time of Simple local search is 22.5 times more than the the computation time of Algorithm *F&E*, for the other instance the computation time of Simple local search is 22.7 times more than the computation time of Algorithm *F&E*.

Figure 8.5: Computation times of simulated annealing and *F&E* for instances with 5 jobs



In the Figure 8.5, we see that Algorithm *F&E* is clearly much faster than Simulated annealing. For the instance with 11 types, the computation time of Algorithm *F&E* is 0.18 seconds and the computation time of Simulated annealing is 6.4 seconds, that is approximately 36 times more than the computation time of Algorithm *F&E*. The worst case takes place at an instance with 29 types, at this instance the computation time of Simulated annealing is 41 times more than the computation time of Algorithm *F&E*.

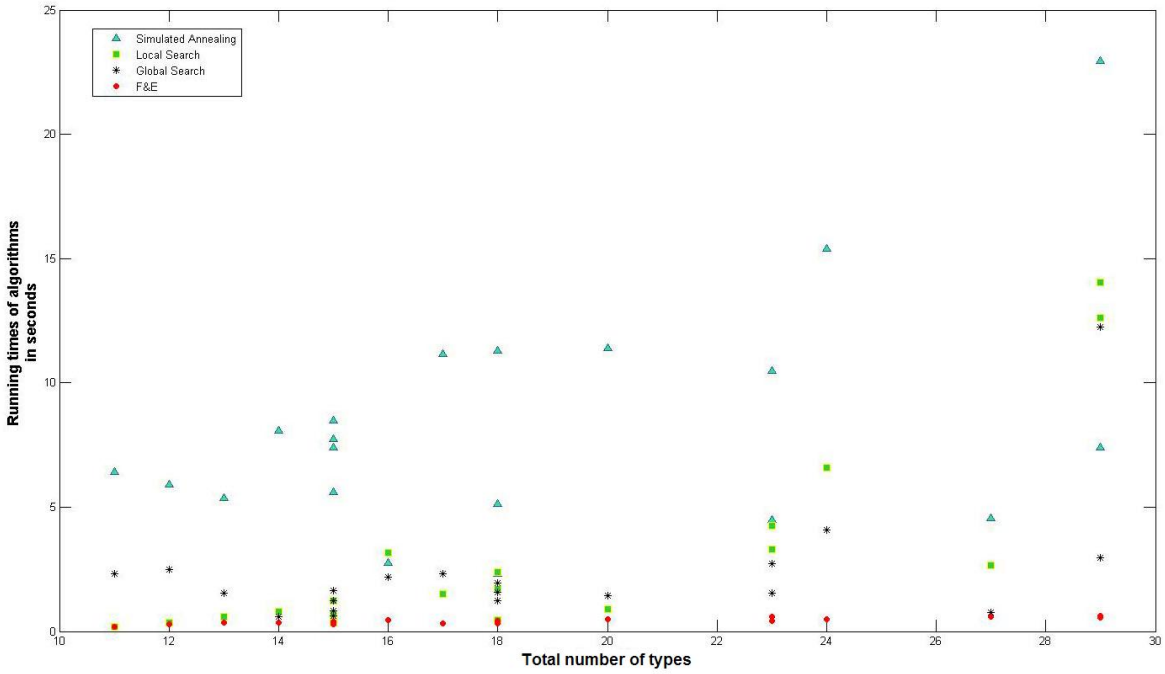
Figure 8.6: Computation times of all algorithms for instances with 5 jobs



It can not be seen from the figures but using the test data, we have that the averaged computation time of Global search for 20 instances (each with 5 jobs) is 2.3 seconds. For the same 20 instances the averaged computation times of Simple local search and Simulated annealing are respectively 11.7 seconds and 8.3 seconds. Though, the averaged computation time of Global search is much smaller than the averaged computation times of the other two control algorithms, Algorithm *F&E* defeat Global search at every instances. Moreover, the averaged computation time of Algorithm *F&E* is 0.4 seconds.

Merging Figure 8.4 , Figure 8.5 and Figure 8.6 we get Figure 8.7

Figure 8.7: Computation times of the global search and *F&E* for instances with 5 jobs



Analogously for instances with 10 jobs the computation times of different algorithms are plotted against the total number of types in figures 8.8, 8.9, 8.10, 8.11. Now we have 10 jobs per instance instead of 5 jobs, the range of the total number of types shift to the right from $[11, 29]$ to $[23, 51]$. As the number of types increases, the computation times of Algorithm *F&E* stay under 1 second, while the computations times of the other algorithms become so large that at an instance with 40 types, the computation time of Simulated annealing is 60 times more than the computation time of Algorithm *F&E*. That Algorithm *F&E* is much faster than all three control algorithms, is so clearly justified by Figure 8.4 , Figure 8.5 and Figure 8.6, that they need no further comment.

Figure 8.8: Computation times of the local search and $F\&E$ for instances with 10 jobs

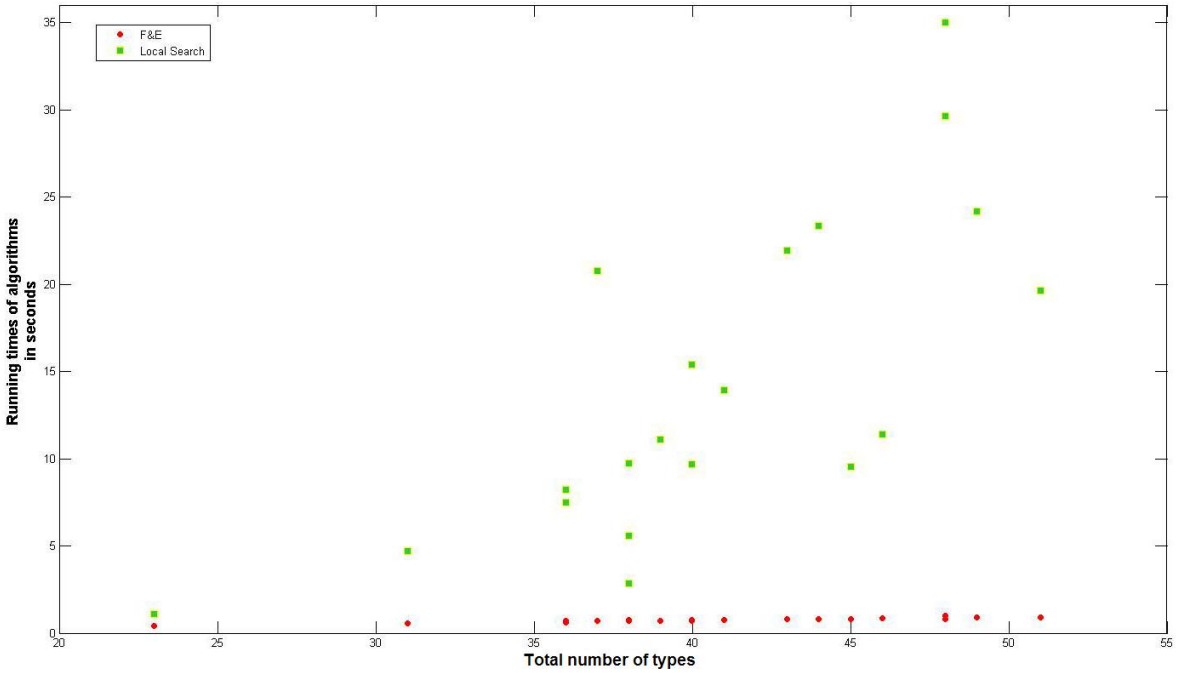


Figure 8.10: Computation times of the global search and $F\&E$ for instances with 10 jobs

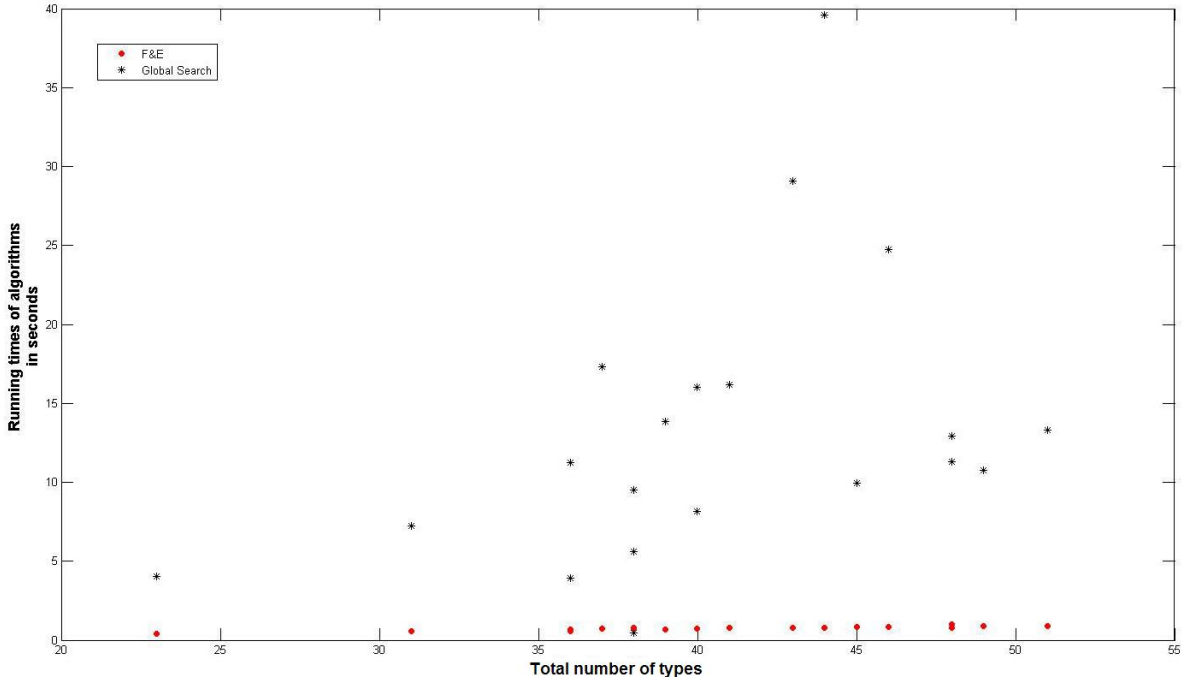


Figure 8.9: Computation times of simulated annealing and *F&E* for instances with 10 jobs

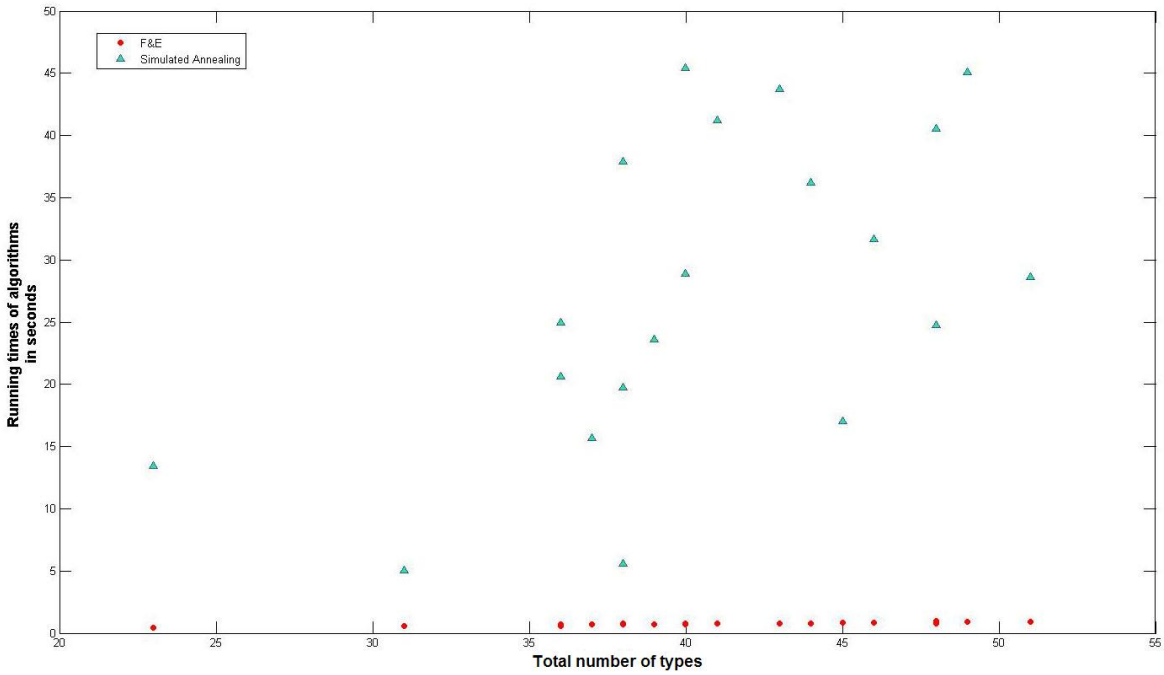
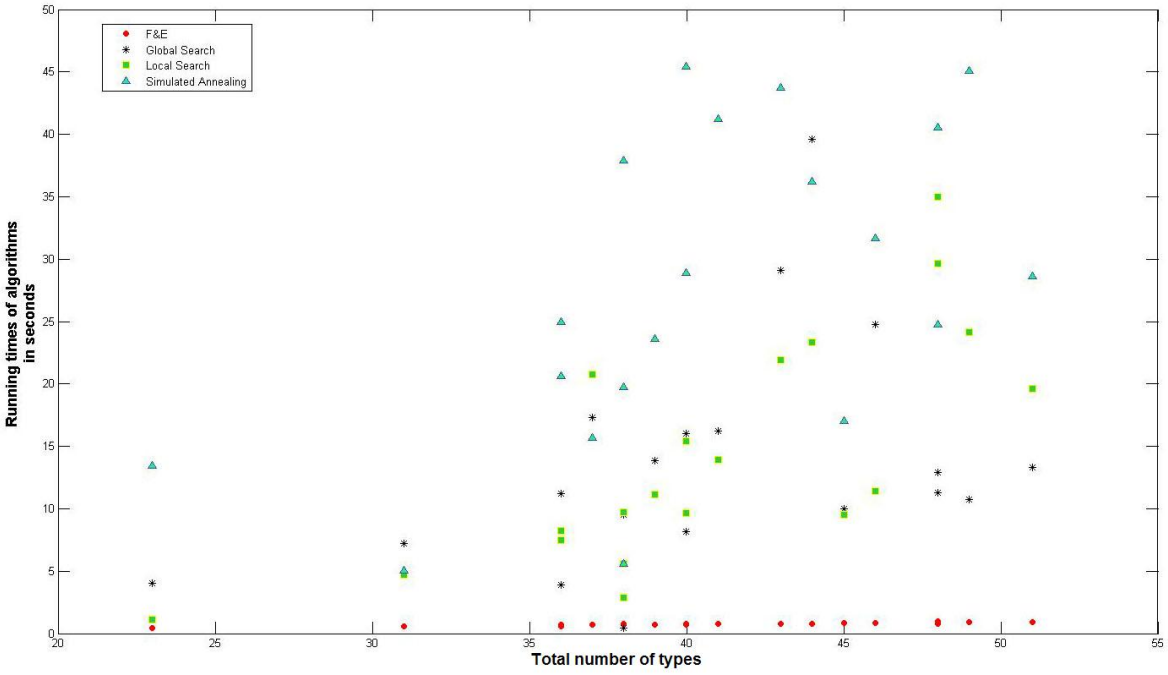


Figure 8.11: Computation times of all algorithms for instances with 10 jobs



As can be seen, Algorithm *F&E* is faster than all three control algorithms. When the

total number of types slightly increases, the computation time of Algorithm *F&E* remains more of less the same, while the control algorithms clearly needs more time, especially Simulated annealing and Simple local search. For all 40 small scale instances the running time of Algorithm *F&E* is less than 1 second. The average computation time of Algorithm *F&E* for 20 instances, each with 10 jobs, is 0.8 seconds. For the same instances, the average computation times of Simple local search, Simulated annealing and Global search are respectively 14.3 seconds, 27.5 seconds and 13.3 seconds.

Generally, Simulated annealing takes more time than the other two control algorithms. This can be clearly seen in figures 8.7 and 8.11. Simulated annealing is known to be an algorithmic framework with good performance, but the drawback of requiring high running times in order to converge to high quality solutions. This is indeed confirmed by our experiments.

In summary, Algorithm *F&E* provides solution of excellent quality, in less than 1 second for every small instance tested here.

8.3 Experiments for large scale instances

In this class of experiments we also have two sorts of test: quality tests and speed tests.

In the experiments for small scale instances we use the ratios $\frac{Obj(F\&E)}{Obj(ILP)}$ as quality indicators. Recall that in the large scale instances, $|W_j| \leq 10$ and $|P_j| \leq 10$, so each job can have at most 100 different types. In the small scale instances, each job can have up to 9 different types. Due to the larger size of the instances, the ILP-model is no longer able to handle most of the generated problems. Therefore, we have to use the LP-model to obtain lower bounds for expected total payments. Hence, we use the ratios

$$\frac{Obj(F\&E)}{Obj(LP)} \tag{8.2}$$

as quality indicators.

For the speed test, we earlier let the control algorithms stop at the moment a expected total payment is found that is less than or equal to the objective value of Algorithm *F&E*. Here, this is also unrealizable because of the size of the large scale instances. Even if we set time limit to be 60 minutes, for the most instances all three control algorithms fail to find an expected total payment that is less than or equal to the expected total payment found by Algorithm *F&E*. So here we run Algorithm *F&E*, the computation time of Algorithm *F&E* is then used as the time limit for the control algorithms. After that we compute for the next algorithms

- algorithm where Smith's rule is used as allocation rule,
- local search,

- simulated annealing,
- global search,
- F&E.

the ratio of the algorithm's objective value over the LP-model's objective value. Because all algorithms start from the Smith's rule, the idea is to compare how much the result by Smith's rule is improved by the control algorithms and by Algorithm *F&E* in the same amount of time.

For Simulated annealing, we will use the best found result and not the last found result. In example 8 we give a typical small scale instance (used in this class of experiments) and the corresponding results.

Table 8.3: Example 8

Instance data		Algorithms	Obj value	Running time
Number of jobs	20	Smith	25138.11	
Max size	10x10	LP	19140.27	
Number of types	445	F&E	21481.78	17.51188
		GS	25109.35	18.66481
		LS	25128.1	18.11799
		SA	25137.14	17.94663

Example 8. In this example we see that the control algorithms hardly improve the result of Smith's rule with a time limit of 17.51188 seconds. As can be seen later in this section, Global search performs significantly better than the other two control algorithms. Even so Algorithm *F&E* outshines all three control algorithms.

8.3.1 Quality test

Just as earlier, we arrange the instances in the order of increasing total number of types. For instances of 5 jobs, 10 jobs and 20 jobs, we compute the ratios *Obj(algorithm)*/ *Obj(LP)* described above and plot these ratios against the total number of types figures 8.12, 8.13 and 8.14 respectively.

Figure 8.12: Ratio expected total payment given by algorithms / expected total payment given by the LP-model for instances with 5 jobs

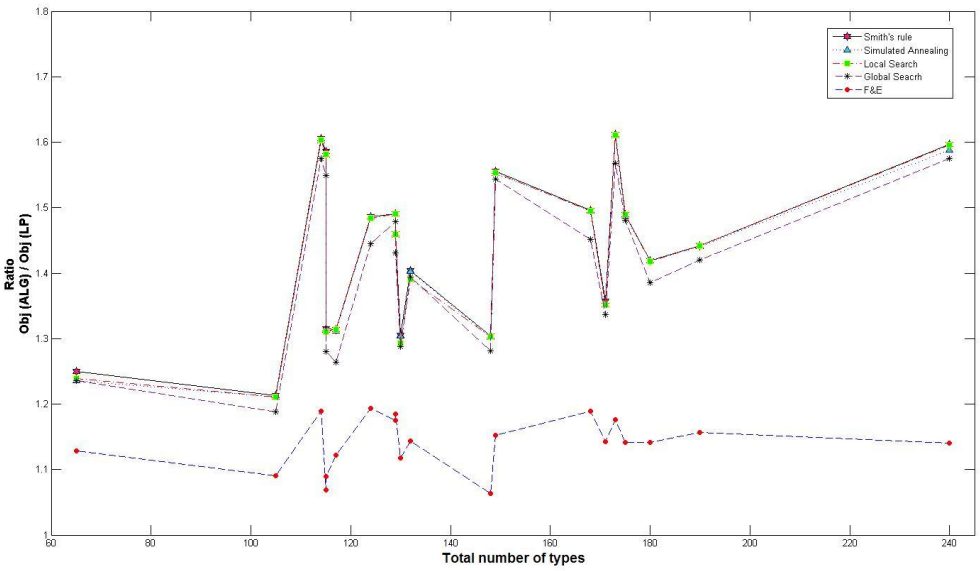


Figure 8.13: Ratio expected total payment given by algorithms / expected total payment given by the LP-model for instances with 10 jobs

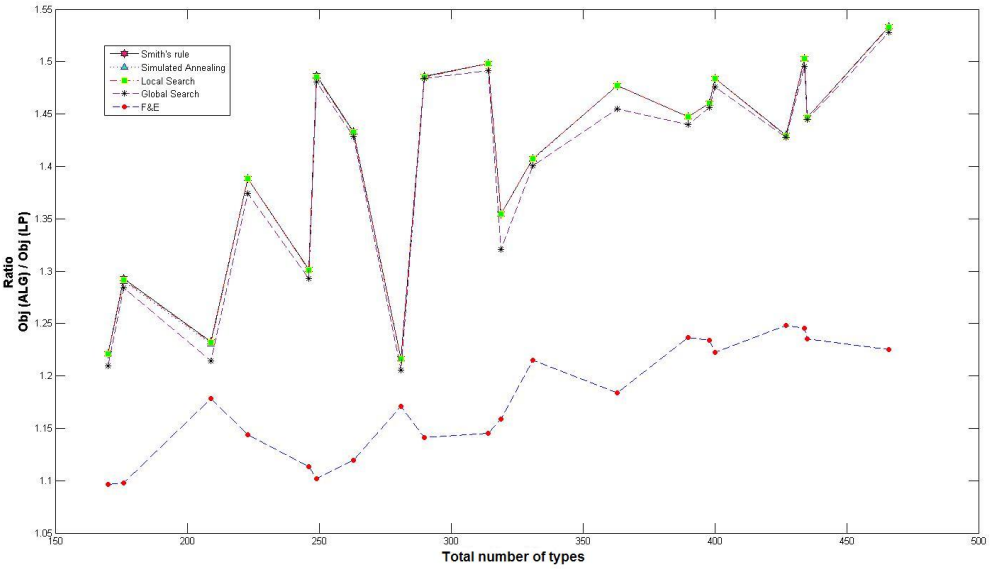
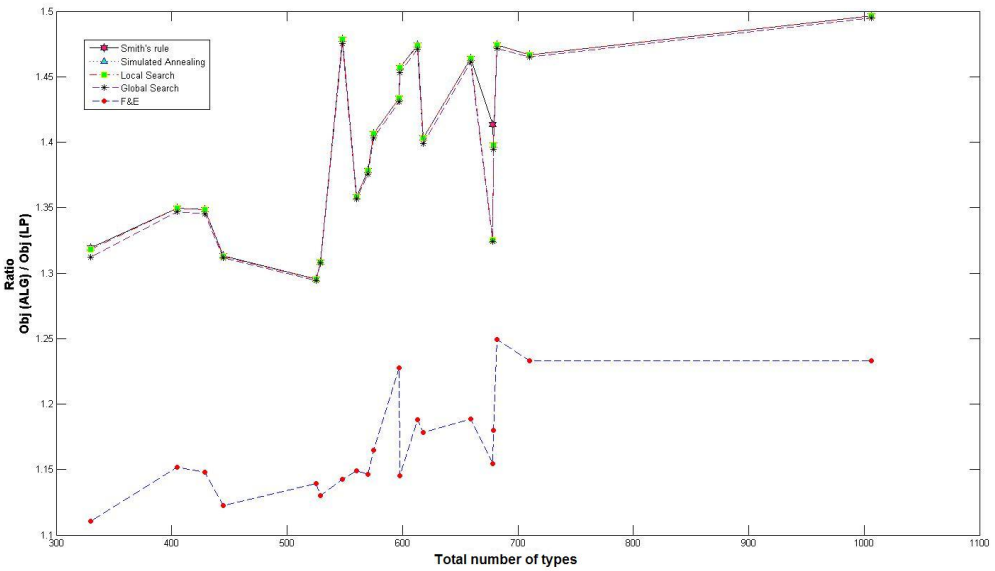


Figure 8.14: Ratio expected total payment given by algorithms / expected total payment given by the LP-model for instances with 20 jobs



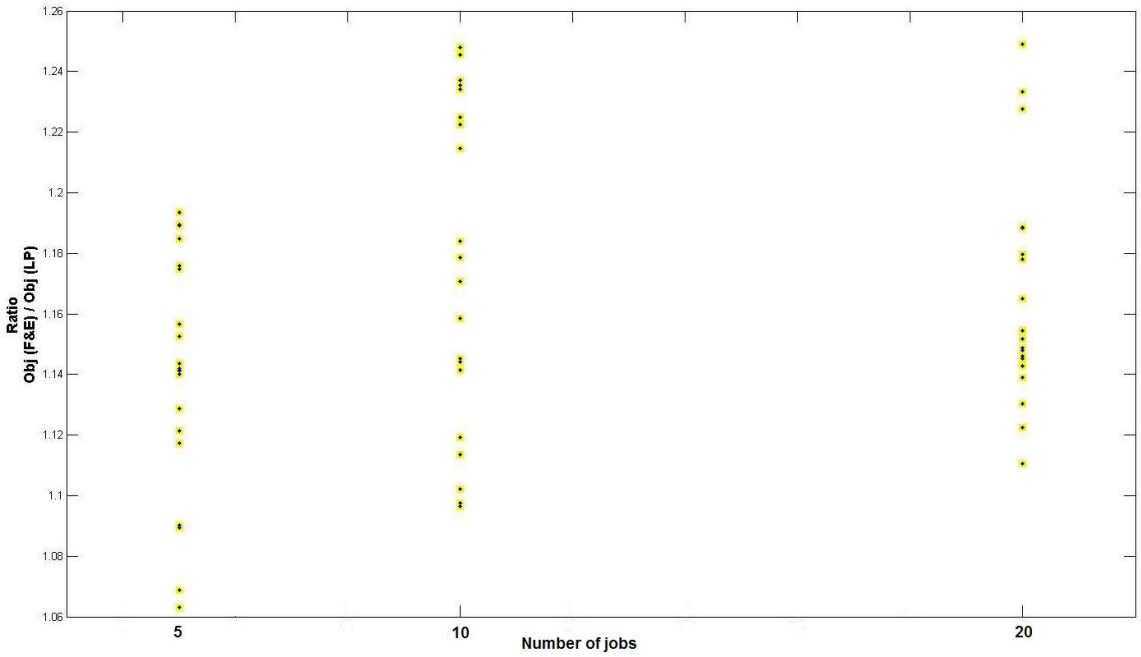
For the quality test, we are only interested in the bottom most curves in the plots above, the curves that belong to Algorithm *F&E*. For the moment, ignore other curves.

In the last two Figure 8.13 and Figure 8.14 we see that the two bottom most curves show the tendency to increase as the number of types increases. This suggest that the growth in total number of types may have a negative effect on the performance of Algorithm *F&E*, with other words on the quality of the result of Algorithm *F&E*.

Now, if we compare the tail of the curve in figure 8.12, more precisely the part of the curve that involve instances with the total number of types lying between 320 en 470, to the begin of the curve in figure 8.13, the part of the curve that involve instances with the total number of types lying between 330 en 550. The most ratios belonging to the begin of the curve in figure 8.13 are noticeably smaller than those belonging to the tail of the curve in figure 8.12, despite the fact that the instances in figure 8.13 have the same or large total number of types. Note that if instance A en B are both having the same total number of types, but instance A has fewer jobs, then on average instance A has more number of types per job. This seems to suggest that not only the total number of types affects negatively the performance of Algorithm *F&E* but also the number of type per job. However, recall that in one-dimensional setting, no matter how many types we have per jobs Algorithm *F&E* returns the optimal IIA result. Thus it has also to do with how the type graphs look like, that is the configurations of the type graphs.

Next, we put the ratios of all instances with the same number of jobs under each other in a one-dimensional plot, see figure 8.15

Figure 8.15: Ratio *Ojb(F&E)/Obj(LP)*



From Figure 8.15 we see that the ratio-range for large scale instances runs from 1.07 to 1.25. Recall that in the experiments for small scale instances we have a ratio-range from 1 to 1.05, see figure 8.3. This major shift in ratio-range can be explained by:

- the maximum size of the type graph was changed from 3×3 to 10×10. That means in the small instances, each job can have at most 9 types, now in the large instances, each job can have at most 100 types.
- a larger total number of types. The range of number of types was $[11, 51]$ and for large instances the range is $[65, 1006]$.
- a looser lower bounds by using the LP-model instead of the ILP-model.

Now consider also the other corresponding curves that belong to Smith's rule, Simple local search, Simulated annealing and Global search in Figure 8.12, Figure8.13 and Figure8.14. The curves of Smith's rule are a little bit difficult to see because they largely overlap with the curves of Simple local search and Simulated annealing. These curves are far above the corresponding curves of Algorithm *F&E*. That means the results given by Algorithm *F&E* are much better than the results given by other algorithms.

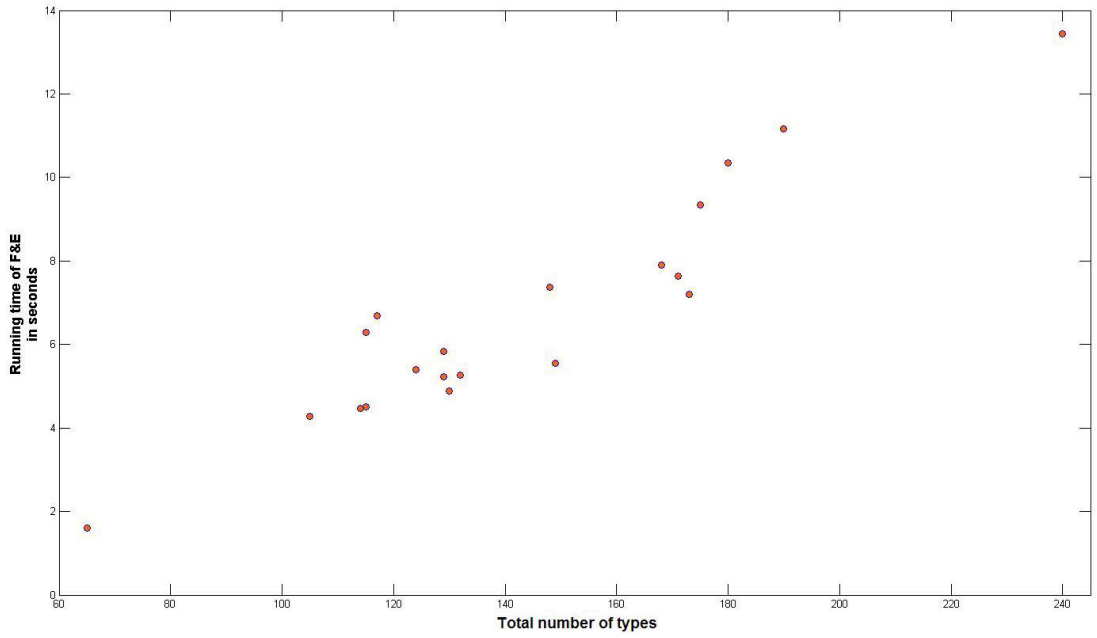
In view of the strong growth in instance's sizes, Algorithm *F&E* gives reasonable results for large instances.

8.3.2 Speed test

For the speed test, we compare how much the result by Smith's rule is improved by the control algorithms and Algorithm *F&E* in the same amount of time. Consider again figures 8.12, 8.13 and 8.14 now with all curves. Clearly, the improvements by Simple local search and Simulated annealing are negligible. Global search delivers better results, but still these improvements by Global search are insignificant compared with the improvements by Algorithm *F&E*.

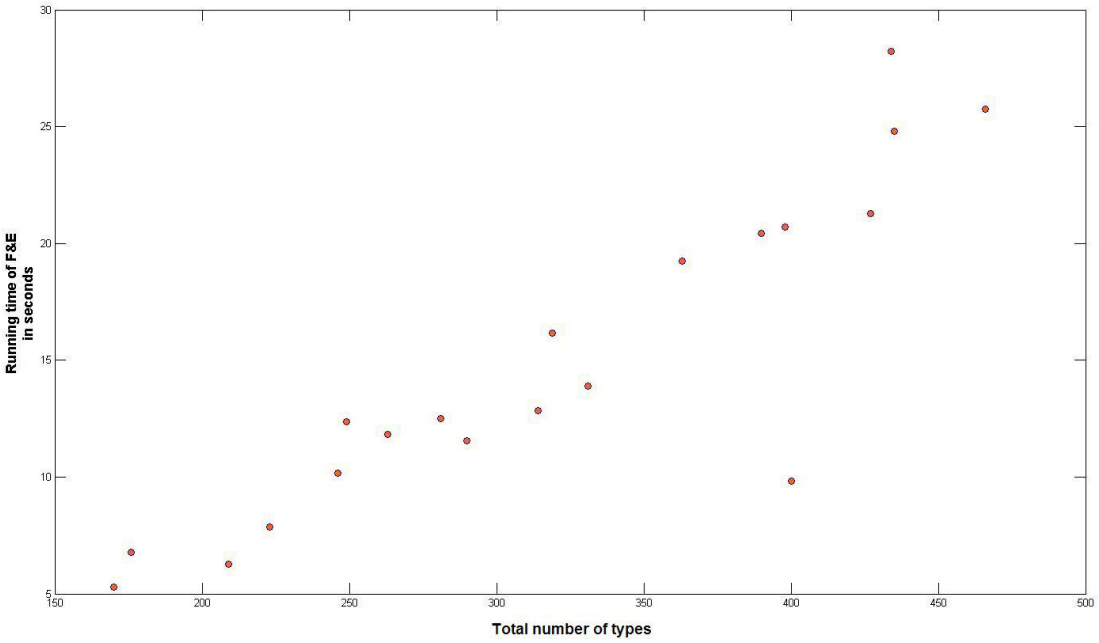
To give the readers some sense of the computation time of Algorithm *F&E*, we plot these times against the total number of types in figures for instances with 5 jobs, 10 jobs and 20 jobs respectively in Figure 8.16, Figure 8.17 and Figure 8.18.

Figure 8.16: Computation time of Algorithm *F&E* for instances with 5 jobs



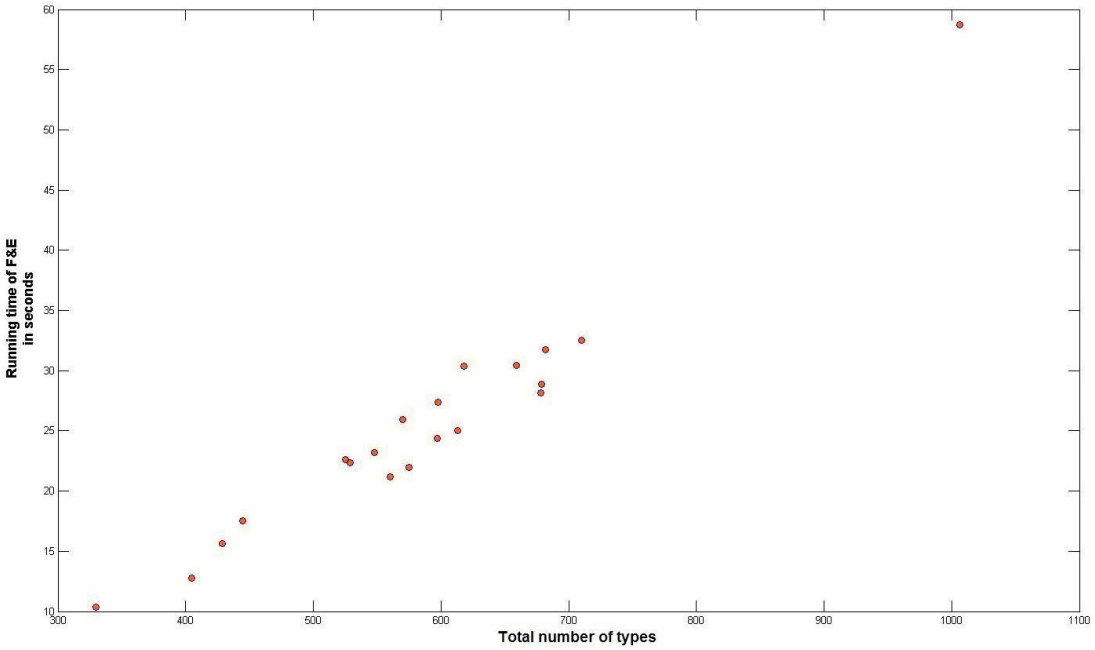
For test instances with up to 240 types in total, Algorithm *F&E* takes less than 14 seconds.

Figure 8.17: Computation time of Algorithm *F&E* for instances with 10 jobs



For test instances with up to 466 types in total, Algorithm *F&E* takes less than 30 seconds.

Figure 8.18: Computation time of Algorithm *F&E* for instances with 20 jobs



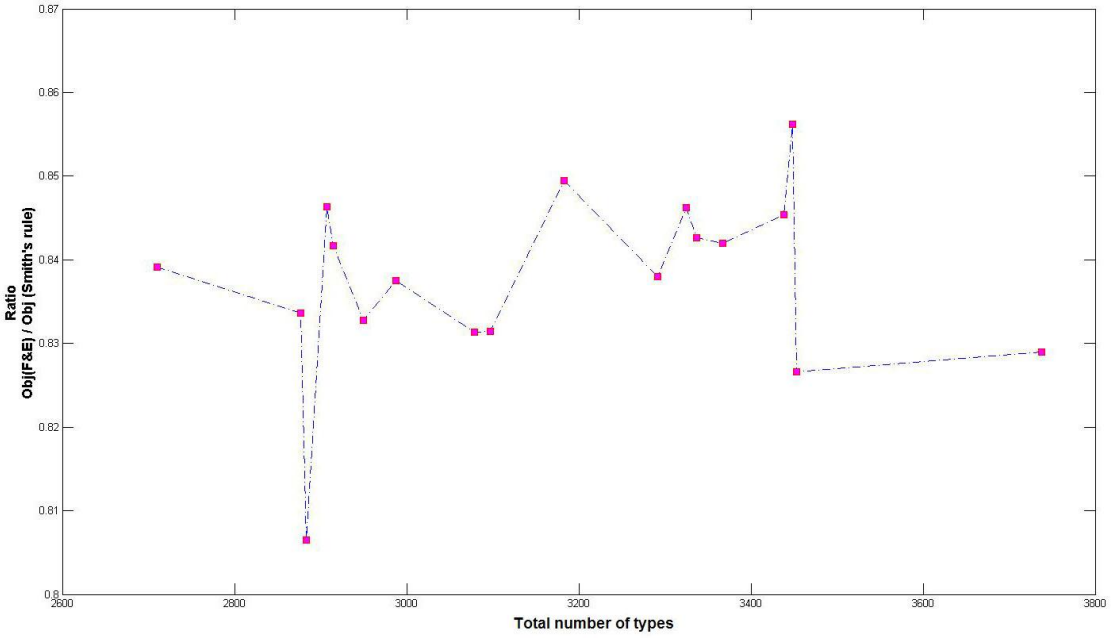
So for test instances with up to 1000 in total, Algorithm *F&E* takes less than 1 minute.

The computation times of Algorithm *F&E* for large instances are in the same order of magnitude as the computation times of the LP-model. Therefore we are satisfied with the computation times of Algorithm *F&E* for large instances.

8.4 Huge instances

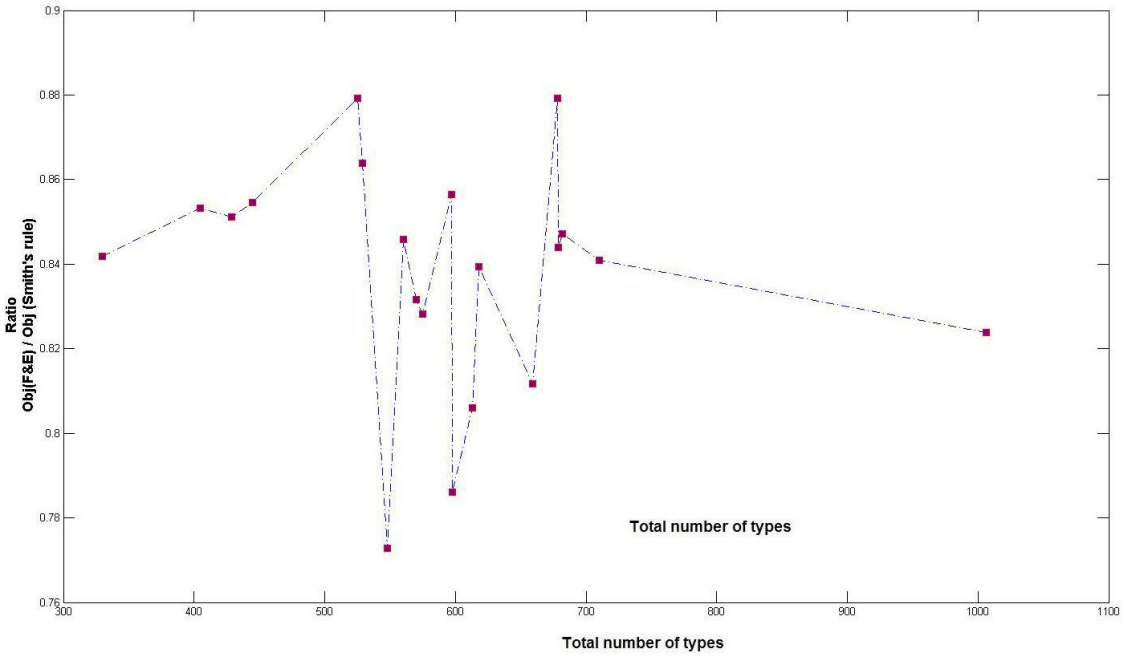
As mentioned earlier, the LP-solver is not able to handle the huge instances, instances with 100 jobs. Hence for the huge instances we no longer have lower bounds for the expected total payments. We compute the ratios $\frac{Obj(F\&E)}{Obj(Smith's\ rule)}$ in order to be able to say something about the quality of Algorithm *F&E*.

Figure 8.19: Ratio *Ojb(F&E)/Obj(Smith's rule)* at huge instances



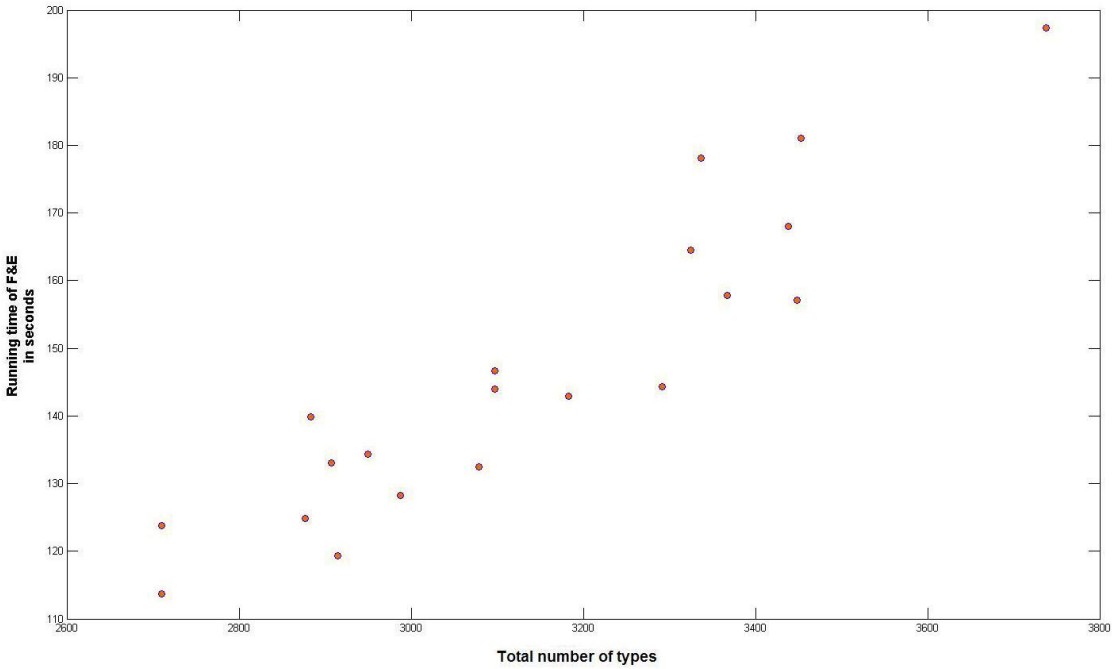
From Figure 8.19, we observe that the results by Algorithm *F&E* are 14% to 19% better than the results by Smith's. Doing the same for large instances with 20 jobs (in the previous section) we get the plot in Figure 8.20. The ratio-range at huge instances is more or less the same as the range at large instances.

Figure 8.20: Ratio *Ojb(F&E)/Obj(Smith's rule)* at large instances with 20 jobs



As regards the speed of Algorithm *F&E*, we plot the computation times of Algorithm *F&E* against the total number of types.

Figure 8.21: Computation time of Algorithm *F&E* at huge instances



In summary, the results presented in this chapter show that Algorithm *F&E* is a very good algorithm, in the sense that with Algorithm *F&E* we have achieved the main goal of this thesis, namely, to find an algorithm that for each instance in small computation time, computes a mechanism that

- is Bayes-Nash incentive compatible and (interim) individually rational,
- gives a 'reasonable result'.

9 Conclusion

In the first part of the thesis, we explore the graph theoretic approach from related literature [2, 6]. It has been proved by Heydenreich et al. that

- the graph theoretic approach yields a closed form solution for the optimal mechanism in the single dimensional case,
- in contrast to the one-dimensional case optimal mechanisms generally do not satisfy IIA condition in the two dimensional setting. As a result, the same graph theoretic approach, as used in the one dimensional setting, must fail to determine an optimal mechanism for the scheduling problem.

This outcome gives rise to the question: 'Can we then (approximately) compute deterministic optimal-IIA mechanisms using the graph theoretic approach?' which in turn leads to our initial implementation, explained in Section 6.2. In the initial implementation, the payments, the virtual weights are computed analogously to the one-dimensional setting. For each instance the algorithm outputs a finite sequence of mechanisms. The best mechanism is then selected as the solution.

Though it is a very fast algorithm, it turned out that for most of the reasonably sized instances, the algorithm resulted in solutions (mechanisms) where the corresponding allocation rules are just Smith's rule with respect to the original weights. In spite of the fact that it is not a good algorithm, it does provide us new insight into the problem. That is, the ironing procedure as used in one-dimensional setting, may have potential unfavorable effects on the two-dimensional cases. It is not known yet whether there is a 'better' ironing procedure for the two-dimensional cases. The newly gained insight , together with intensive experimental research has led us to the alternative Algorithm *F&E*. The validation of *F&E* was divided into two parts: the quality validation and the speed validation. Both were done by applying Algorithm *F&E* to 120 instances, with various seizes from small to huge.

As regards the quality, Algorithm *F&E* performed really well for all tested instances:

- For small instances it found results that are at most 5% from the IIA-optimal results found by the ILP-model.
- For large instances it found results that are at most 25% from the lower bounds (for optimal results) found by the LP-model.
- For huge instances it improved the results by Smith's rule by at least 12% and at most 23%.

The quality of Algorithm *F&E* may be affected by the size of the instances. However the unfavorable influence on the quality is only to see by a very large growth in the instance's size, especially the maximum size of type graphs.

Concerning the speed, it is impressive how fast Algorithm *F&E* is. Even though, we have constructed several control algorithms and we have made them as good as possible, Algorithm *F&E* outshines all three control algorithms at almost all instances. At really small instances, where *F&E* does not 'outshine' the control algorithms, the computation times of Algorithm *F&E*, are still noticeably smaller than those of the control algorithms. Moreover, Algorithm *F&E* are able to solve instances that are so huge that they are unmanageable for the LP-solver.

The results of this thesis point to several interesting directions for future work:

- In this thesis, an allocation rule (equivalently, a mechanism) is represented as a linear ordering of the types. By changing the ordering, we jump from mechanism to mechanism. But can we do this in a different way? Although the results presented in Chapter 8 have demonstrated the effectiveness of Algorithm *F&E*, mechanisms found by Algorithm *F&E* always satisfy IIA-condition and optimal mechanisms in general do not satisfy IIA-condition. So maybe with another representation we are no longer limited to IIA-mechanisms.
- The work described in this thesis has been concerned with the mechanism design problem for the sequencing problem with private information. For the problem we have found a method that was shown to be fast and effective. Is it possible to transfer ideas to other mechanism design problems?
- What is the worst case provable performance guarantees for Algorithm *F&E*? In other words, what is the theoretical (upper) bound for the ratio expected total payment given by Algorithm *F&E* over the optimal-IIA expected total payment?

10 Appendix

10.1 Example 3: job's data

there are also 3 jobs, with the next data

Prt \ W	3	9	10	24
11	0.5472	0.0175	0.3880	0.0473

Table 10.1: Job 1

The table can be read as follows: $W_1 = \{3, 9, 10, 24\}$ en $P = \{11\}$, $\varphi((3, 11)) = 0.5472$, etc.

Prt \ W	3	8	10	15	20	24	25	26
14	0.0280	0.0149	0.0116	0.0329	0.0249	0.0613	0.0084	0.0207
10	0.0203	0.0404	0.0397	0.0137	0.0082	0.0240	0.0154	0.0153
9	0.0150	0.0045	0.0262	0.0226	0.0184	0.0369	0.0317	0.0252
7	0.0420	0.2009	0.0027	0.0215	0.0090	0.0112	0.0129	0.0080
5	0.0012	0.0013	0.0032	0.0455	0.0211	0.0436	0.0024	0.0132

Table 10.2: Job 2

Prt \ W	4	11	17	22	26	27	28
17	0.0075	0.0068	0.0567	0.0493	0.0217	0.0109	0.0378
14	0.0111	0.0069	0.0549	0.0356	0.0241	0.0424	0.0026
11	0.0070	0.0019	0.0622	0.0043	0.0143	0.0042	0.0462
8	0.0194	0.0141	0.0858	0.0398	0.0226	0.0025	0.0483
5	0.0198	0.0715	0.0155	0.0172	0.0284	0.0144	0.0345
2	0.0191	0.0092	0.0003	0.0010	0.0160	0.0106	0.0019

Table 10.3: Job 3

10.2 Determine the range for α

The first plot belongs to an instance of 10 jobs, de second one belongs to an instance of 20 jobs,..., the last plot is from an instance of 100 jobs. In any instance, each job can have at most 10 different weights, 10 different processing times.

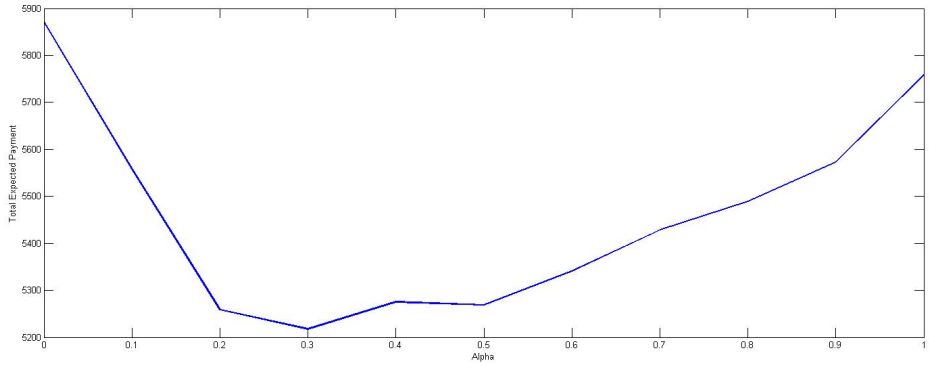


Figure 10.1: 10 jobs

The optimal α-value in Figure 10.1 is 0.3.

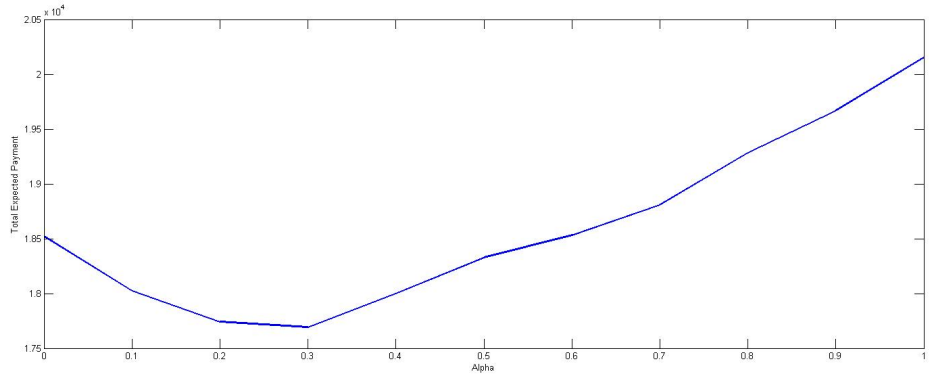


Figure 10.2: 20 jobs

The optimal α-value in Figure 10.2 is 0.3.

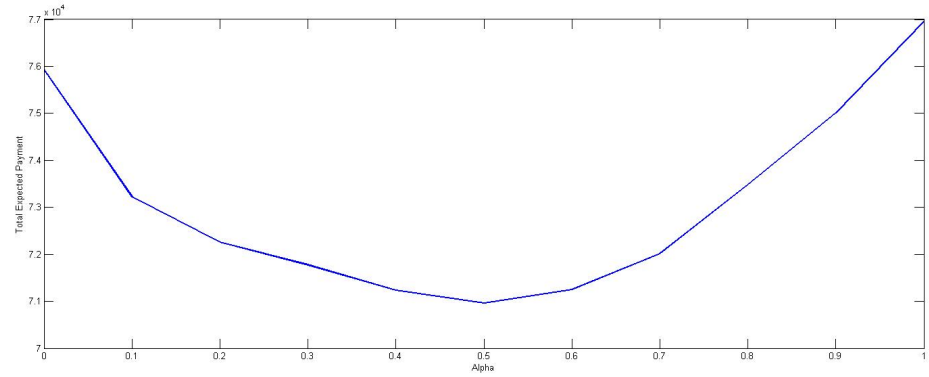


Figure 10.3: 30 jobs

The optimal α-value in Figure 10.3is 0.5.

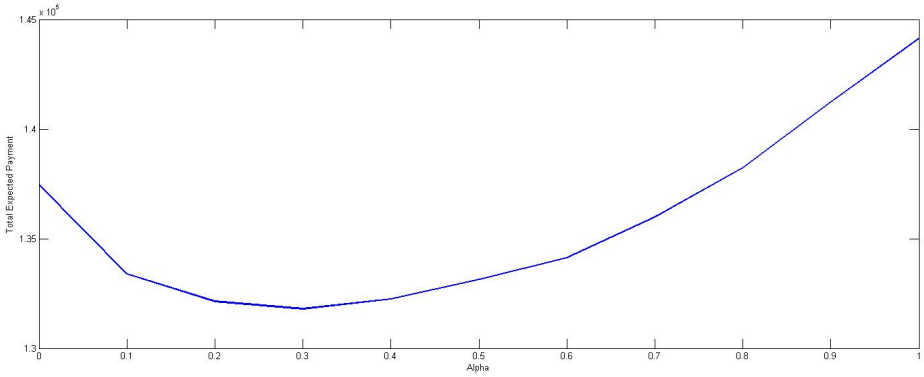


Figure 10.4: 40 jobs

The optimal α-value in Figure 10.4 is 0.3.

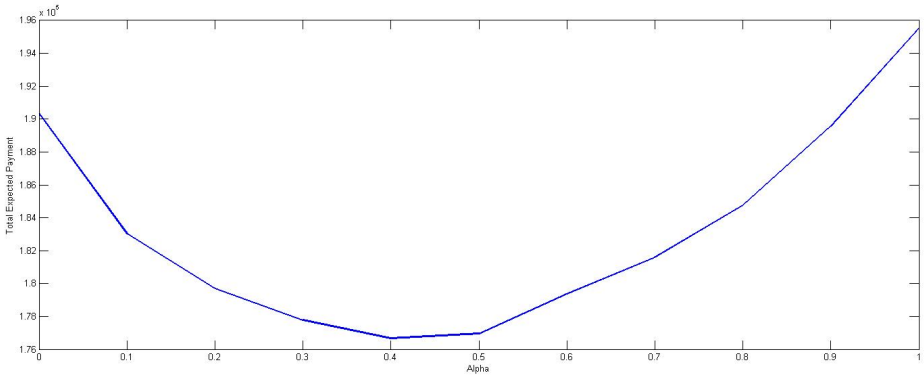


Figure 10.5: 50 jobs

The optimal α-value in Figure 10.5 is 0.4.

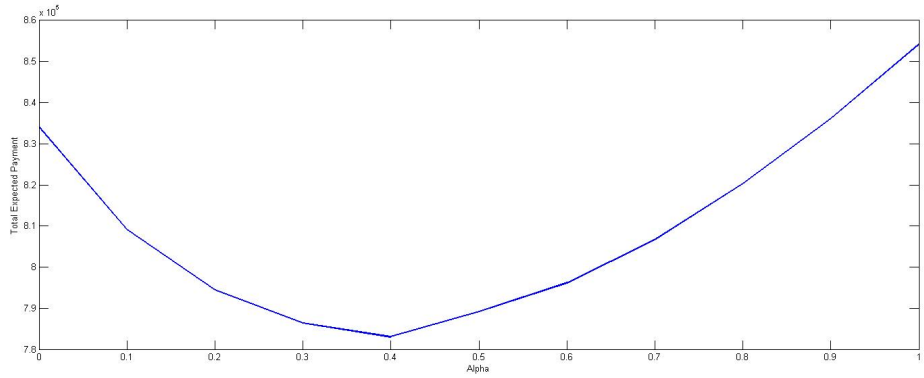


Figure 10.6: 100 jobs

The optimal α-value in Figure 10.6 is 0.4.

Bibliography

[1] R. Hoeksma and M. Uetz. Two dimensional optimal mechanism design for a sequencing problem. In: M. Goemans and J. Corréa, editors, *Integer Programming and Combinatorial Optimization*, volume 7801 of *Lecture Notes in Computer Science*, pages 242–253. Springer, 2013.

[2] B. Heydenreich, D. Mishra, R. Müller, and M. Uetz. Optimal Mechanisms for Single Machine Scheduling. In: C. Papadimitriou and S. Zhang, editors, *Internet and Network Economics*, volume 5385 of Lecture Notes in Computer Science, pages 414–425. Springer, 2008.

[3] Vladimír Černỳ. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications*,pages 41–51, 1985.

[4] Partha Dasgupta, Peter Hammond, and Eric Maskin. The implementation of social choice rules: Some general results on incentive compatibility. *The Review of Economic Studies*, pages 185–216, 1979.

[5] Sudipto Dasgupta. Competition for procurement contracts and underinvestment. *International Economic Review*, pages 841–865, 1990.

[6] Jelle Duives, Birgit Heydenreich, Debasis Mishra, Rudolf MÃijller, and Marc Uetz. On optimal mechanism design for a sequencing problem. *Journal of Scheduling*, pages 1–15, 2014.

[7] Allan Gibbard. Manipulation of voting schemes: a general result. *Econometrica: journal of the Econometric Society*, pages 587–601, 1973.

[8] Jerry R Green and Jean-Jacques Laffont. Partially verifiable information and mechanism design. *The Review of Economic Studies*, pages 447–456, 1986.

[9] Milton Harris and Robert M Townsend. Resource allocation under asymmetric information. *Econometrica: Journal of the Econometric Society*, pages 33–64, 1981.

[10] Jason Hartline and Anna Karlin. Profit maximization in mechanism design. *Algorithmic Game Theory*, pages 331–362, 2007.

[11] Birgit Heydenreich, Rudolf Müller, Marc Uetz, and Rakesh V Vohra. Characterization of revenue equivalence. *Econometrica: Journal of the Econometric Society*, pages 307–316, 2009.

[12] Bengt Robert Holmström. *On incentives and control in organizations*. Stanford University, 1977.

[13] Patrick Kenis. Waiting lists in dutch health care: an analysis from an organization theoretical perspective. *Journal of health organization and management*, pages 294–308, 2006.

[14] Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi, et al. Optimization by simmulated annealing. *science*, pages 671–680, 1983.

[15] Ron Lavi and Chaitanya Swamy. Truthful mechanism design for multidimensional scheduling via cycle monotonicity. *Games and Economic Behavior*, pages 99–124, 2009.

[16] Y. Li. Hospital capacity investment: Theory and practice. *Doctoral Dissertation, University of Michigan, Ann Arbor, MI*, 2005.

[17] Alexey Malakhov and Rakesh V Vohra. An optimal auction for capacity constrained bidders: a network perspective. *Economic Theory*, pages 113–128, 2009.

[18] Yusufcan Masatlioglu and Neslihan Uler. Behavioral differences between direct and indirect mechanisms: Evidence from first price auctions. Technical report, Citeseer, 2006.

[19] Debasis Mishra. *Lecture Notes of Game Theory II*. Indian Statistical Institute, 2013.

[20] Rudolf Müller, Andrés Perea, and Sascha Wolf. Weak monotonicity and bayes–nash incentive compatibility. *Games and Economic Behavior*, pages 344–358, 2007.

[21] Roger B Myerson. Incentive compatibility and the bargaining problem. *Econometrica: journal of the Econometric Society*, pages 61–73, 1979.

[22] Roger B Myerson. Optimal auction design. *Mathematics of operations research*, pages 58–73, 1981.

[23] Asu Ozdaglar. *Lecture Notes of Game Theory with Engineering Applications*. Massachusetts institute of technology, 2010.

[24] Paolo Penna and Carmine Ventre. Collusion-resistant mechanisms with verification yielding optimal solutions. *ACM Transactions on Computation Theory (TOCT)*, page 6, 2012.

[25] Michele Piccione and Guofu Tan. Cost-reducing investment, optimal procurement and implementation by auctions. *International Economic Review*, pages 663–685, 1996.

[26] Jean-Charles Rochet. A necessary and sufficient condition for rationalizability in a quasi-linear context. *Journal of Mathematical Economics*, pages 191–200, 1987.

[27] Robert W Rosenthal. Arbitration of two-party disputes under uncertainty. *The Review of Economic Studies*, pages 595–604, 1978.

[28] Guofu Tan. Entry and r & d in procurement contracting. *Journal of Economic Theory*, pages 41–60, 1992.

[29] Rakesh V Vohra. Optimization and mechanism design. *Mathematical programming*, pages 283–303, 2012.