A Technique for Large Automated Mechanism Design Problems

Frederick Asselin
Dept. of Computer Science
and Software Engineering
Université Laval
Québec (QC) G1K 7P4
Canada
fasselin@damas.ift.ulaval.ca

Brigitte Jaumard
Institute for Information
Systems Engineering
Concordia University
Montréal (QC) H3G 1M8
Canada
bjaumard@ciise.concordia.ca

Antoine Nongaillard
Dept. of Computer Science
and Software Engineering
Concordia University
Montréal (QC) H3G 1M8
Canada
a_nongai@encs.concordia.ca

Abstract

*Automated mechanism design (*AMD*) seeks to find, using algorithms, the optimal rules of interaction (a mechanism) between selfish and rational agents in order to get the best outcome. Here optimal is defined by the objective function of the designer of the mechanism where the function has usually some desirable properties (e.g. Pareto optimal). A difficulty with* AMD *lies in the size of the optimization problem that one needs to solve in order to select the best mechanism: there is a huge number of variables (and constraints but to a lesser extent) even for* AMD *instances of relatively small size. We study how to adapt the column generation techniques in order to solve the linear programming* LP *formulation of the* AMD *problem and compare its efficiency with the classical simplex algorithm for linear programs, on a bartering of goods example. We show that the resulting column generation algorithm is very quickly faster than the simplex algorithm for a fixed number of types (i.e., preference relations) on the goods as the number of goods increases, and then for a fixed number of goods as the number of types increases. Moreover, we show that, as the number of goods increases, the percentage of variables that need to be explicitly considered by the column generation techniques comes down very fast while the simplex algorithm must always consider explicitly all variables.*

1 Introduction

Mechanism design seeks to define the rules of interaction between selfish and rational agents in order to attain an objective not necessarily shared by all agents or even any one of them. In order to meet his objective, the designer of the mechanism relies on information privately owned by each agent. Because individual agents are selfish and rational, some agents may try to manipulate the mechanism by giving false information in order to obtain a better outcome associated with their own objective. Therefore, a good mechanism must be built in such a way that it must be immune to such manipulation and the agents must be encouraged to behave as expected by the designer of the mechanism, i.e., by revealing to the mechanism the true private information. We often call this information "type" and the goal of the mechanism is to incite each agent to reveal his true type while trying to meet his objective. Agents are autonomous with respect to the type each reveals to the mechanism. The mechanism simply makes agents reveal their true type despite their autonomy.

Studied in economics for a long time, mechanism design has recently become a tool also used by researchers in computer science and operations research as distributed systems, e.g., electronic market design or quality of service in IP networks, have many characteristics of an economy [6]. Recent research studies tackled this problem as an optimization problem and sought automatic design tools with artificial intelligence search algorithms under different designer's constraints. These studies resulted in the so-called *automated mechanism design* [3].

At the outset of a mechanism design process, the designer knows only the list of possible types for each agent, he does not know which one is the true type of each agent. Under these conditions, the designer must commit to a mechanism. When the mechanism is announced to the agents, each agent analyzes the mechanism and reveals his most advantageous type, i.e., the type giving the largest profit according to his own objective. Indeed, the agents can either reveal their true type, or lie by revealing another type which will possibly give them a larger utility.

An application of mechanism design is the problem of bartering where each agent owns a set of goods and wishes to exchange some of his goods for goods owned by other agents in the absence of a currency system. Owners of net-

works could use barter to exchange resources usage. The design of the bartering mechanism must satisfy a series of constraints. First, all agents should gain from participating in the barter. By that, we do not mean that each agent will exchange some of its goods but that if an exchange is made, then it will be beneficial to the agents involved in the exchange. In short, no agent will loose any utility in taking part in the barter (at least on average) as opposed to the utility of its initial set of goods. If the designer creates a mechanism which satisfies such property, we can say that the *individual rationality* constraints, denoted by *IR constraints*, are satisfied and all agents will take part in the bartering process.

A second set of constraints is necessary in order to incite each agent to reveal its true type. If an agent reveals a false type, the mechanism incurs a loss because its outcome is based on false information. It is necessary that no agent has a gain to lie. When the designer offers a mechanism, the agent looks at which type revelation would give him the greatest utility. If the agent realizes that by lying instead of revealing his true type he obtains a greater utility, then he won't reveal his true type because all agents are selfish. Thus, if revealing its true type gives an agent a greater or equal utility than by lying, then the *incentive compatibility* constraints, denoted by *IC constraints*, are satisfied.

Automated mechanism design can be formulated as an optimization problem under constraints. Unfortunately, the number of variables in automated mechanism design can be huge even for a moderate size problem. Indeed, the number of variables increases exponentially as a function of the number of agents. Even for one or two agents, the number of outcomes to the mechanism could be quite large, as in the bartering problem, leading very quickly to a huge number of variables that is difficult to manage.

However, in operations research, tools exist to handle optimization problems with a huge number of variables, namely column generation techniques [5] when one has to deal with linear programs with an exponential number of variables. We show that this technique is applicable to mechanism design and can help to reduce the number of variables to be explicitly considered. Moreover, it is much faster than the classical simplex algorithm for solving linear programs.

In this paper, we will focus on the bartering problem to show how column generation techniques can be applied for mechanism design. Experimental results will then illustrate the efficiency of column generation and in particular that column generation is very quickly faster than the simplex LP algorithm for a fixed number of types as the number of goods increases, and then for a fixed number of goods as the number of types increases. This efficiency mainly lies in the reduction of the number of variables to be explicitly considered, while the implicit set of variables may be of exponential size. We will illustrate this point, with statistics on the percentage of variables that need to be considered explicitly as the number of goods increases.

The next section offers a brief review of the relevant literature. In section 3, we state formerly the automated mechanism design problem. In Section 4, we outline the main features of column generation techniques and their application to automated mechanism design. Experimental results are presented in Section 5. Conclusions are drawn in Section 6.

2 Literature Review

Viewing the design of mechanisms as a computational problem is relatively recent. Conitzer and Sandholm [3] were the first to our knowledge to view mechanism design as an optimization problem over the space of incentive compatible mechanisms. Sethuraman, Piaw and Vohra [18, 19] used integer programming to characterize preference domains (utility functions) for agents that permit mechanisms implementing a particular objective (the Arrovian social welfare function). They were able to find both new and old results with this method. Malakhov and Vohra [13] also used linear programming to find the optimal auction mechanism when the type of agents has more than one dimension. Hsu [7] studies problems related to the finiteness of both the type set of each agent and the outcome set of the problem. Jameson *et al.* [9] empirically evaluated mechanisms generated automatically. Blumberg and Shelat [1] automatically designed mechanisms for agents using fictitious play instead of playing the mechanism in the ideal rational way. Hyafil and Boutilier [8] used automated mechanism design to construct mechanisms for a new solution concept, minimax-regret equilibria, when the probability distribution on the type set of each agent is not available to the designer of the mechanism.

The other known way to automate mechanism design beside optimization is by learning the best mechanism from the behaviour of the agents [10, 14]. Genetic algorithms were used to find a good double auction, a type of mechanism, for software agents [2]. A similar approach used genetic programming to evolve the rules of an auction at the same time that the strategies of the agents changed to adapt to the new rules [15]. In another way, the mechanism's parameters could be configured via search and learning techniques in order to find the effect of the parameters' values on the outcome of the mechanism [20].

Automated mechanism design has already been applied and proved itself useful. Likhodedov and Sandholm [11] characterized a particular type of auctions which has a parameter's value that is optimally set by a mechanism design algorithm for the particular setting at hand. No analytical solution exists for this problem [16]. The same idea was

also used to get a greater revenue for the auctioneer in a combinatorial auction [12]. Finally, automated mechanism design was used to find a optimal sequence of "take-it-or-leave-it" offers [17].

All of these approaches suffer from the huge number of variables problem. We show that the adaptation of column generation to automated mechanism design can help reduce this problem.

3 Statement of the AMD problem

Before stating formerly the AMD problem in the context of the bartering problem, let us first introduce the following notations:

- a finite set of K outcomes: $O = \{o_1, o_2, \ldots, o_k\}$, where o_{init} denotes the initial partition of goods among the agents;
- a finite set $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ of n agents such that, for each agent $a \in \mathcal{A}$, we associate:
  (i) a set $\Theta_a = \{\theta_a^1, \theta_a^2, \ldots, \theta_a^{m_a}\}$ of m_a types where a type represents, e.g., the preferences on the set O of outcomes. Under some conditions, let us assume that the preferences can be represented by a utility function;
  (ii) a probability distribution P_a on Θ_a: for each $\theta_a^i \in \Theta_a$, p_a^i is the probability that the agent a has the type θ_a^i;
  (iii) a utility function $u^a : \Theta_a \times O \rightarrow \mathbb{R}$. If the agent is the mechanism designer, then his utility function is defined as $d : O \rightarrow \mathbb{R}$ in the bartering problem;
- an objective function, e.g., the social welfare F_{WELFARE} defined by the sum of all agents utilities, or the utility of a particular agent, F_{DESIGNER};
- the desired type of the mechanism (deterministic or randomized);
- the use of payments (authorized or not, influences the form of the utility function);
- a set of individual rationality constraints;
- a set of incentive compatibility constraints.

These characteristics form the constraints of the problem, and the mechanism's goal (F_{WELFARE} or F_{DESIGNER}) constitutes the objective function of the optimization problem defined over variables g_i^k (for a case with one type revealing agent) where a particular g_i^k is the probability of choosing the outcome o_k when the (single) agent reveals type θ^i. It is an integer program (deterministic mechanism) with $g_i^k \in \{0, 1\}$ or a linear program (randomized mechanism) with $g_i^k \in [0, 1]$. The fundamental difference between a deterministic mechanism and a randomized mechanism is that a deterministic one always returns the same outcome for a given vector of revealed types by the agents but a randomized mechanism returns a probability distribution on the set of outcomes. We will concentrate on randomized mechanisms since they include deterministic mechanisms as a special case.

Mainly, we are interested by the problem of bartering, so there is no payment possible, but only the exchange of goods. For the individual rationality constraints, we used such constraints in the *interim* and *ex post* variants. We will present these concepts for a case with only one type revealing agent. We could extend this case to multiple type revealing agents by adding appropriate variables and constraints, but the principles remain the same. Also, we could assume that the designer knows everything except the precise type of each agent or we could include unknown information into the type space of the agents making it bigger. This last point is part of relevant future research because the designer does not know the initial set of goods of each agent.

interim: this concept is the following, after the agent has knowledge of her type but before she knows the outcome chosen by the mechanism (because the mechanism can be randomized), her expected utility is greater or equal on average than her previous utility at the beginning. This kind of constraints incites the agent to take part in the mechanism. Indeed, the mechanism promises a greater or equal utility, but there is a part of luck: the agent can obtain less than her utility at the beginning. In any case, the agent will take part in the mechanism because it guarantees a higher utility on average than she had at the beginning. If we apply this concept for one type revealing agent (say agent 2, agent 1 being the mechanism designer) and for each agent's types, then the *IR constraints* will look like this:

$$\forall \theta_2^i \in \Theta_2 \qquad \sum_{o_k \in O} u^2(\theta_2^i, o_k) \cdot g_i^k \geq u^2(\theta_2^i, o_{\text{init}})$$

ex post: the agent must obtain an equal or greater utility than she had before the exchange after she knows the outcome chosen by the mechanism. For a case with only one agent, if an outcome o_k and a type θ_2^i of an agent 2 exists such that for that type, the agent receives less utility from the outcome than at the beginning, then we must fix all variables g_i^k to 0. That way, there is no chance the agent will receive less utility from the outcome of the mechanism than at the beginning. Of course, that could make the problem overconstrained and therefore, no solution is possible. We can apply this kind of *IR constraints* on the agent, and on the designer when the objective function is the designer's utility function.

$\forall \theta_2^i \in \Theta_2, \forall o_k \in O$ if $u^2(\theta_2^i, o_k) < u^2(\theta_2^i, o_{\text{init}})$ for a $o_k \in O$ then $g_i^k = 0$.

Likewise for the mechanism designer, if there exist $o_k \in O$ such that $d(o_k) < d(o_{\text{init}})$, then $\forall \theta_2^i \in \Theta_2$, $g_i^k = 0$.

For the set of incentive compatibility constraints, we can use Bayes-Nash equilibrium. That means that revealing the true type gives as much utility or more as lying given the other agents reveal their true type. Another equilibrium that could been use is the dominant strategy equilibrium where telling the truth gives as much or even more utility than lying even if the other agents do not reveal their true type. But the constraints associated with the dominant strategy equilibrium are stronger than the ones associated with the Bayes-Nash equilibrium. Therefore the best mechanism in dominant strategy equilibrium has often a much lower objective function value that the best mechanism in Bayes-Nash equilibrium. However, in cases where there are only one type revealing agent, Bayes-Nash equilibrium and dominant strategy equilibrium are the same [4]. For one type revealing agent (again agent 2), the associated constraints are $\forall \theta_2^i, \theta_2^l \in \Theta_2, i \neq l$:

$$\sum_{o_k \in O} u^2(\theta_2^i, o_k) \cdot g_i^k \geq \sum_{o_k \in O} u^2(\theta_2^i, o_k) \cdot g_l^k$$

We will focus on one particular instance of the automated mechanism design applied to the barter problem. In this instance, two agents want to exchange some goods. The first agent acts as the mechanism designer so the objective function is to maximize the expected utility of the first agent. The incentive compatible constraints use the Bayes-Nash equilibrium (but since there is only only type revealing agent, it is the same as the dominant strategy equilibrium). The individual rationality constraints for the mechanism designer are of the *ex post* variant while they are *interim* for the lone agent. The designer's utility function $d(o_k)$ only depends on the outcome o_k. The mechanism is randomized, therefore the mechanism must output a probability distribution on the set of outcomes O. Some constraints are added to make sure the output is a valid probability distribution. Finally, payments are prohibited in the mechanism. So, we have the following linear program:

Objective function (maximize the expected utility of the mechanism designer):

$$\sum_{p^i \in P} p^i \cdot \sum_{o_k \in O} d(o_k) \cdot g_i^k$$

IC constraints:

$$\forall \theta^i, \theta^l \in \Theta, i \neq l \quad \sum_{o_k \in O} u(\theta^i, o_k) \cdot g_i^k \geq \sum_{o_k \in O} u(\theta^i, o_k) \cdot g_l^k$$

***Ex post* IR constraints for the mechanism designer:** If $\exists\, o_k$ such that $d(o_k) < d(o_{\text{init}})$, then $\forall \theta^i \in \Theta$, $g_i^k = 0$

***Interim* IR constraints for the lone agent:**

$$\forall \theta^i \in \Theta \quad \sum_{o_k \in O} u(\theta^i, o_k) \cdot g_i^k \geq u(\theta^i, o_{\text{init}})$$

Probability distribution constraints:

$$\forall \theta^i \in \Theta \quad \sum_{o_k \in O} g_i^k = 1$$

$$\forall \theta^i \in \Theta, \forall o_k \in O \quad g_i^k \geq 0$$

4 An Overview of Column Generation

Column generation techniques offer solution methods for linear programs with a very large number of variables (e.g., exponential) where constraints can be expressed implicitly with respect to the variables. Let us consider the following linear program with a huge number of variables (say n) and m constraints:

$$(\text{LP}) \qquad \max cx$$
$$\text{subject to:} \quad \begin{cases} Ax \leq b \\ 0 \leq x \leq 1. \end{cases}$$

The variables x of LP form the columns of the matrix A. So we will use indifferently both terms (variables and columns). Usually, linear programs are solved iteratively where in each iteration, the matrix A and vectors b and c of LP get rewritten. The new value of c is $\overline{c}$ and is called *reduced costs* of the variables. These costs indicate the potential for a particular variable x_j to increase the objective value of the function we want to optimize.

Column generation techniques rely on a decomposition of the initial linear program into a restricted linear program, called the *master problem*, and a pricing problem. The master problem corresponds to a linear program associated with a restricted matrix $A' \subseteq A$ with $\geq m$ columns. His guiding principle is the following: an optimal solution of the master program may be optimal for the initial linear program depending of the reduced costs of the missing columns, i.e, those that are not explicitly considered in LP.

Let us proceed as follows. Consider a submatrix of A, i.e., $A^1 = A'$ and solve the corresponding linear program. Let x^1_{LP} be an optimal vector. Note that c^1 is a subvector of c, associated with the columns of A^1.

$$(\text{LP}_1) = \left\{ \begin{array}{c} \max\ c^1 x \\ A^1 x \leq b \\ 0 \leq x \leq 1 \end{array} \right. \xrightarrow{\text{LP SOLVER}} \quad x^1_{\text{LP}}$$

We next have the following question: does there exist a column $a^j \in A \setminus A^1$ such that $\overline{c}_j > 0$; i.e., can we find a column of the matrix $A \setminus A^1$ for which the reduced cost is positive ? Two possible answers:

- NO: then the optimal solution x^1_{LP} of the system (LP_1) is optimal for (LP);
- YES: we must add some columns to the matrix A^1 in order to find the optimal solution, i.e., solve

$$(\text{LP}_2) = \left\{ \begin{array}{c} \max\ cx \\ A^2 x \leq b \\ 0 \leq x \leq 1 \end{array} \right. \overset{\text{LP SOLVER}}{\longrightarrow} \quad x^2_{\text{LP}}$$

  with $A^2 = A^1 \cup \{a^j\}$ such that $\overline{c}_j > 0$.

We must iterate this process as long as we are no more able to find j such as $\overline{c}_j \geq 0$ (a column for which the reduced cost is positive). In such a case, we conclude that we have obtained an optimal solution for (LP). But there remains a question: how to find if there exists a column which has a positive reduced cost? This problem is called *pricing problem* and is defined as follows. Consider the current master problem, e.g., (LP_1). Let us assume that we want to find if there exists a column a^j with a positive reduced cost. We must solve the pricing problem (PP) according to:

$$(\text{PP}) = \left\{ \begin{array}{l} \max \overline{c}(a^j) \\ \text{with constraints on the component of } a^j \\ \text{to guarantee that } A^2 \subseteq A, \end{array} \right.$$

where A^2 is the concatenation of A^1 and a^j.

For the LP problem, the reduced cost is defined as follows: $\overline{c} = c - vA$ (matrix form)
where v is the vector of dual variables given after solving (LP_1). For the LP-restricted problem, we get: $\overline{c}_j = c_j - v^1 a^j = c_j(a^j) - v^1 a^j$ for column j, where $\overline{c}_j$ is the reduced cost of the column j, v^1 is the vector of dual variables obtained by solving LP_1.

It is a linear expressions where the only unknowns are the m components of the vector a^j. The constraints on a^j come from the definition of the matrix, which must be a submatrix of A. The constraints are induced by the columns omitted at the time of the restriction of the problem

Solving the pricing problem is therefore equivalent to solving: $\overline{c}^\star = \max_{a^j \notin A'} \overline{c}_j(a^j) = \max_{a^j \notin A'} c_j(a^j) - v^1 a^j$
subject to some constraints on the component of a^j.

The pricing problem (PP) is very often a combinatorial problem difficult to solve because it is often $\mathcal{NP}$-complete. However, it is not necessary to solve it exactly at each iteration: as long as one can find a column a^j with a positive reduced cost, we can iterate. Let *Algo H* be a heuristic that solves PP.

If *Algo H* returns $\tilde{c}$ such that $\tilde{c} > 0$, we iterate. Otherwise, if *Algo H* returns $\tilde{c}$ such that $\tilde{c} \leq 0$, we need to solve PP using an exact algorithm, denoted by *Algo E*. Let $\hat{c}$ be the optimal reduced cost provided by *Algo E*. Three cases must be distinguished:

- $\hat{c} \geq \tilde{c} > 0 \Rightarrow$ we iterate with the column found by *Algo H*;
- $\hat{c} > 0 \geq \tilde{c} \Rightarrow$ we iterate with the column found by *Algo E*;
- $0 \geq \hat{c} \geq \tilde{c} \Rightarrow$ there exists no more column a^j with a positive reduced cost and therefore we have reached the optimal solution of (LP).

If one has a very efficient heuristic that is able to reach the optimal solution, *Algo E* needs to be called only once. In the next section, we will show that even with a naive algorithm, column generation is very quickly faster than classical linear programming (the simplex algorithm implementation in CPLEX that we call SC), as it requires to consider explicitly only a (very) small fraction of the problem's variables. This therefore leads to save memory space and to faster solution of larger instances that the simplex algorithm cannot even solve.

5 Experimental results

The results of this section were obtained for the specific case of automated mechanism design described by the LP formulation at the end of section 3. The naive algorithm used for the pricing problem is a linear search among the variables that stops as soon as a positive reduced cost has been found. Conitzer and Sandholm [4] have done a similar experimentation for their algorithm to search a deterministic mechanism. They have shown that as the size of the instances of the bartering problem gets bigger, the (integer) linear programming implementation of CPLEX gets competitive results compare to the other algorithms they evaluated. They also indicate that for big cases as ours, this implementation is the best compared to the algorithms they evaluated. Therefore, we have done a similar experimentation for column generation as compared to the CPLEX implementation of the simplex algorithm (SC) in searching for a randomized mechanism.

In order to generate random cases, we must only generate a random type for agent 1 (the designer of the mechanism) and as many random types for agent 2 as specified. For a specific case made of a particular number of goods and a particular number of types for agent 2, we have generated 100 such cases in a random fashion. The types of both agents was generated by choosing from a uniform distribution on the set of utility $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$

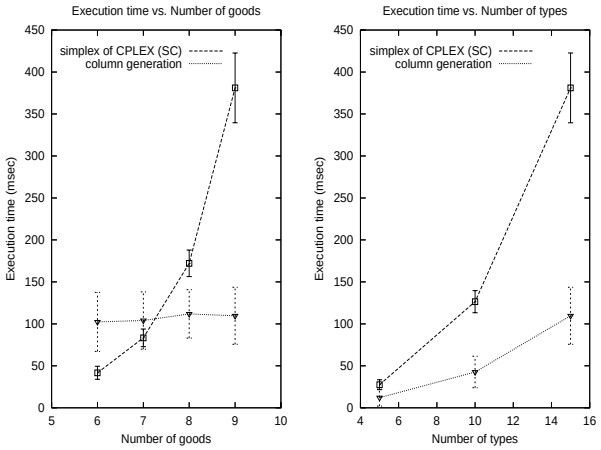

Figure 1. Mean execution time and standard deviation of SC and column generation algorithms depending on the number of goods between the agents for 15 types (left) and the number of possible types of the agents for 9 goods (right).

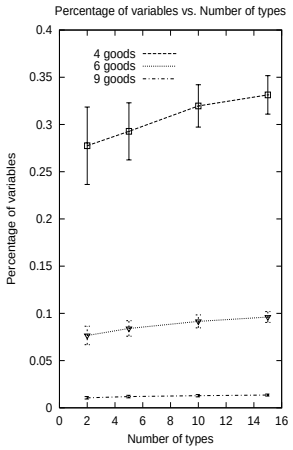

Figure 2. Mean percentage of variables used by column generation with standard deviation.

the utility of each good independently. The utility of an outcome for an agent is then the sum of the utility of the goods the agent owns in that outcome. The probability distribution on the set of types of agent 2, P^2, was fixed to be also uniform. As a convention, the goods with an odd index were initially owned by agent 1 and goods with an even index were initially owned by agent 2 to set the initial outcome (o_{init}).

Results related to mean execution time are summarized in figure 1. The leftmost graphic shows that beyond 8 goods for 15 types, column generation becomes faster than SC. While the execution time of SC seems to increase exponentially as a function of the number of goods, it remains stable from 6 to 9 goods for the column generation algorithm. This can be explained by the fact that the number of goods has a significant impact on the number of variables and column generation is pretty good at selecting the best variables to be explicitly considered. The number of partitions of goods (or equivalently the number of outcomes $o_k \in O$) between two agents is exponential in the number of goods (exactly $2^{\#\text{goods}}$). The number of variables in the problem is the number of outcomes times the number of types. So the number of goods influences exponentially the number of variables. This could explain the exponential growth of the execution time of SC. But column generation, even with a naive algorithm for solving the pricing problem is able to find the relatively few variables that receive strictly positive value (> 0) corresponding to the few partitions of goods that receive strictly positive probability of being selected for a given type revelation by the agent in the optimal solution (or mechanism) and in the intermediate solutions leading to the optimal one. On the other end, the standard deviation of the results for column generation is larger than the one for SC but as the number of goods increases, the standard deviation of both algorithms becomes similar as the standard deviation of the execution time of both algorithms for 9 goods indicates.

The rightmost graphic in figure 1, shows the growth of execution time for a fixed number of goods (here 9 goods) and for varying number of types. We can observe that both algorithms are sensible to the number of types. In the studied case, the number of types influences linearly the number of variables but quadratically the number of constraints (for t types, there are $t^2 + t$ constraints excluding the non-negative constraints, $g_i^k \geq 0$, for each variable). Because the number of goods of 9 is past the threshold of 8 goods, column generation is faster than SC. This shows that once past this threshold, the number of types does not determine which of the two algorithms is faster, column generation is still faster. In fact, the gap in execution time seems to grow as the number of types increases. Also, the standard deviation on the execution time seems to be equal with 15 types even if in smaller cases of 5 and 10 types, the standard deviation associated with SC is smaller.

Figure 2 shows the percentage of the total number of variables in the problem used by the column generation techniques. This parameter in important because the main drawback of automated mechanism design is its huge number of variables. So any technique that use only a small percentage of all variables helps to apply automated mechanism design to bigger problems relevant to practice. Since

SC basically uses all variables, this figure also shows the memory space saving of column generation. We can see that as the number of goods increases, the percentage of variables used by column generation drops to a small value (around 1% for 9 goods). Relative to SC, this means that column generation could solve problems with one hundred times more variables with the same memory space in principle. Even if the number of variables increases exponentially as a function of the number of goods, this small percentage is significative for the absolute performance of column generation. From the number of types point of view, since this parameter does not have as great an influence as the number of goods on the number of variables, as explained before, the percentage of variables used is stable for different number of types for the case with a greater number of goods (9 goods in Figure 2). Also the standard deviation in the percentage of variables used seems to decrease with a greater number of goods. The error bars of the standard deviation in the case with 9 goods is small enough to be confused with the line in the graphic.

6 Conclusion

While we have shown that the column generation techniques, even with a simple algorithm for solving the pricing problem, are better than the simplex algorithm for the bartering problem studied in this paper, we plan to improve the solution of the pricing problem using some heuristic in order to avoid the partial enumeration of the current linear search. If we do not go on until the optimal solution is found, we could also find heuristic solutions that would be satisfactory as compared to mechanisms in use in practice. Finally, we could study how to include unknown information by the designer, like each agent set of goods, into the agents' type space without increasing its size too much.

References

[1] A. J. Blumberg and A. Shelat. Searching for Stable Mechanisms: Automated Design for Imperfect Players. In *Proc. of the 19th National Conference on Artificial Intelligence*, pages 8–13. AAAI Press, 2004.

[2] D. Cliff. Evolution of Market Mechanism Through a Continuous Space of Auction-Types. In *Proceedings of the 2002 Congress on Evolutionary Computation*, volume 2, pages 2029–2034, 2002.

[3] V. Conitzer and T. Sandholm. Complexity of Mechanism Design. In *Proc. of the 18th Annual Conference on Uncertainty in Artificial Intelligence*, pages 103–110, 2002.

[4] V. Conitzer and T. Sandholm. An algorithm for automatically designing deterministic mechanisms without payments. In *Third International Joint Conference on Autonomous Agents and Multi Agent Systems*, volume 1, pages 128–135, 2004.

[5] J. Desrosiers and M. E. Lübbecke. *Column Generation*, chapter A primer in column generation, pages 1–32. GERAD 25th Anniversary Series. Springer, 2005.

[6] J. Feigenbaum and S. Shenker. Distributed algorithmic mechanism design: Recent results and future directions. In *Proceedings of the 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 1–13, New York, 2002. ACM Press.

[7] E. Hsu. Automated Mechanism Design: Type Space and Exponential Auction. In *Proc. of Game Theoretic and Decision Theoretic Agents Workshop at AAMAS'03*, Melbourne, Australia, 2003.

[8] N. Hyafil and C. Boutilier. Regret Minimizing Equilibria and Mechanisms for Games with Strict Type Uncertainty. In *Proc. of the 20th Annual Conference on Uncertainty in Artificial Intelligence*, pages 268–277, 2004.

[9] A. Jameson, C. Hackl, and T. Kleinbauer. Evaluation of Automatically Designed Mechanisms. In *Proc. of Bayesian Modeling Applications Workshop at UAI'03*, 2003.

[10] J. Kalagnanam and D. C. Parkes. *Supply Chain Analysis in the eBusiness Era*, chapter Auctions, Bidding and Exchange Design, pages 140–213. International Series on Operations Research and Management Science. Kluwer, 2003.

[11] A. Likhodedov and T. Sandholm. Auction Mechanism for Optimally Trading Off Revenue and Efficiency. In *Proc. of the Agent Mediated Electronic Commerce Workshop at AAMAS'03*, 2003.

[12] A. Likhodedov and T. Sandholm. Methods for Boosting Revenue in Combinatorial Auctions. In *Proc. of the 19th National Conference on Artificial Intelligence*, 2004.

[13] A. Malakhov and R. V. Vohra. Single and multi-dimensional optimal auctions. CMS-EMS discussion paper 1397, Northwestern University, 2004.

[14] D. Pardoe and P. Stone. Developing Adaptive Auction Mechanisms. In *SIGecom Exchanges*, volume 5, pages 1–10. ACM Press, 2005.

[15] S. Phelps, P. McBurney, S. Parsons, and E. Sklar. Coevolutionary Auction Mechanism Design: A Preliminary Report. In *Proc. of Agent-Mediated Electronic Commerce Workshop at AAMAS'02*, volume 2531 of *Lecture Notes in Computer Science*, pages 123–142. Springer, 2002.

[16] T. Sandholm. Automated Mechanism Design: A New Application Area for Search Algorithms. In *Proc. of the 9th International Conference on Principles and Practice of Constraint Programming*, volume 2833 of *Lecture Notes in Computer Science*, pages 19–36. Springer, 2003.

[17] T. Sandholm and A. Gilpin. Sequences of Take-It-or-Leave-It Offers: Near-Optimal Auctions Without Full Valuation Revelation. In *Proceedings of the Agent Mediated Electronic Commerce Workshop at AAMAS'03*, 2003.

[18] J. Sethuraman, T. C. Piaw, and R. V. Vohra. Integer programming and arrovian social welfare functions. *Mathematics of Operations Research*, 28(2):309–326, 2003.

[19] J. Sethuraman, T. C. Piaw, and R. V. Vohra. Anonymous monotonic social welfare functions. *Journal of Economic Theory*, 128(1):232–254, May 2006.

[20] Y. Vorobeychik, C. Kiekintveld, and M. P. Wellman. Empirical Mechanism Design: Methods, with Application to a Supply Chain Scenario. In *Proc. of the 7th ACM Conference on Electronic Commerce*. ACM Press, 2006.