Local Search Techniques for Multi-Agent Constraint Optimization Problems

THÈSE N^{O} 4074 (2008)

PRÉSENTÉE LE 30 MAI 2008

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS

Laboratoire d'intelligence artificielle

SECTION D'INFORMATIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Quang Huy NGUYEN

ingénieur diplômé de l'Institut polytechnique de Hanoi, Viêt-Nam
et de nationalité vietnamienne

acceptée sur proposition du jury:

Prof. A. Wegmann, président du jury
Prof. B. Faltings , directeur de thèse
Dr J.-A. Rodriguez Aguilar, rapporteur
Dr A. Roli, rapporteur
Prof. M. A. Shokrollahi, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Lausanne, EPFL
2008

Abstract

Combinatorial optimization problems in the real world are ubiquitous. Practical applications include planning, scheduling, distributed control, resource allocation, etc. These problems often involve multiple agents and can be formulated as a *Multi-agent Constraint Optimization Problem* (MCOP). A major challenge in such systems is the agent coordination, such that a global optimal outcome is achieved.

This thesis is devoted to two major issues that arise in MCOP: efficient optimization algorithms and manipulations from self-interested agents. We introduce a new randomized local search optimization algorithm called Random Subset Optimization (RSO). The RSO algorithm is tested on various applications and demonstrated to converge faster than other local search techniques while achieving competitive performance. For self-interested agents, we define a new form of incentive compatibility called *size-limited incentive compatibility* and show that RSO algorithm can be used to prevent agents' manipulations.

Keywords: artificial intelligence, constraint optimization, local search, incentive-compatibility, mechanism design

Résumé

Les problèmes d'optimisations combinatoires dans le monde réel sont omniprésents. Les applications pratiques incluent la planification, l'ordonnancement, les systèmes de contrôle distribués, ou l'attribution de ressources. Ces problèmes impliquent souvent de multiples agents et ils peuvent être formulés par des *problèmes d'optimisation multi-agent sous contraintes* (MCOP). Un défi important dans de tels systèmes est la coordination des agents, afin que des solutions globalement optimales soient obtenues.

Cette thèse est consacrée à deux questions principales qui se posent dans MCOP: l'efficacité des algorithmes d'optimisation et les manipulations par des agents stratégiques. Nous introduisons une nouvelle algorithme de recherche locale qui s'appelle l'optimisation par des sous-ensembles aléatoires (RSO). L'algorithme RSO est testée sur des applications diverses et a démontré qu'elle converge plus rapidement que d'autres techniques de recherche locale, tout en réalisant des performances compétitives. Pour des agents stratégiques, nous proposons un nouveau concept de *incentive-compatibility de taille limitée*. Nous montrons que l'algorithme RSO peut être utilisée pour empêcher des manipulations des agents stratégiques.

Mots-clés : intelligence artificielle, optimisation sous contraintes, recherche locale, incentive-compatibility, mechanism design

Acknowledgements

I wish to address my sincere thanks to my advisor, Professor Boi Faltings, who gave me the opportunity to work at the AI-Lab of the Swiss Federal Institute of Technology in Lausanne, for giving me much freedom in my work and for inspiring and motivating me tremendously during my time in Lausanne. Without his supports, this thesis would never be completed.

I would like to thank Professor Alain Wegmann for chairing my thesis committee. I am also grateful to Professor Amin Shokrollahi, Dr. Andrea Roli, and Dr. Juan-Antionio Rodriguez, for accepting the invitation to be in my thesis committee. I appreciate all their efforts in reading my thesis and bringing valued comments, which helped me considerably improve the quality of this thesis.

A special thank goes to Dr. Andrea Roli for insightful feedbacks and being a great mentor at the CP2005 conference. I am also indebted Dr. Jean-Cédric Chappelier for his useful suggestions on information theory. Thanks Vincent Schickel-Zuber and Martin Vesely for helping me accessing the computation resources at the LIA. I am also grateful to Luu Vinh Toan and Brammert Ottens for giving me their comments on my thesis presentation.

I wish to thank my colleagues at the LIA for their friendly help and enthusiastic collaboration, especially Dr. Sam-Haroud Djamila, Radu Jurca, Adrian Petcu, Paolo Viappiani, Michael Schumacher, Miroslav Melichar, Marita Ailomaa, Thomas Léauté, Santiago Macho Gonzalez, David Portabella, and Arda Alp. I am also grateful to Mrs. Marie Decrauzat for her administrative supports.

My Vietnamese friends from Lausanne (apologies to the ones I forgot, they're equally dear to me): Vu xuan Ha, Nguyen Tuan Viet, Bui Huu Trung, Ho Quoc Bang, Le Lam Son, Mai Tuan Anh, Nguyen Ngoc Anh Vu, Nguyen Xuan Hung, Nguyen Thanh Tung, Nguyen Vu Hieu, Vo Duc Duy, Pham Minh Hai, Pham Van Thai, To Huy Cuong, Phan Van Anh, Do Lenh Hung Son, Nguyen Minh Thu, Nguyen Hoang Minh, Nghiem Quynh Nga, and Vu Le Hung. Thanks to them, nearly one thousand stressful hours in writing up this thesis have been harmoniously relaxed by many enjoyable hours.

From the bottom of my heart, I am extremely thankful to my wife, my beloved daughter, my parents and my family for so many priceless things and so much great love they have ever given to me.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1. Multi-agent constraint optimization

Combinatorial optimization problems in the real world are ubiquitous. Examples in everyday life include logic systems, transportation, scheduling, resource allocation, combinatorial auction, and many others. In most of the cases, these optimization tasks involve multiple agents and constraints that have to be satisfied. These problems consist finding an assignment of discrete values of variables such that the solution is optimal with respect to some criterion. We call these problems as *Multi-agent Constraint Optimization problems* (MCOP).

There are two main tasks that have to be addressed in MCOP: the optimization task and the incentive task. The optimization is a classic task in which we want to maximize or minimize a global *objective function*, whereas the incentive, receiving an increasing attention in recent years, deals with agents' preferences.

The optimization is, however, a hard task due to the dimension of these systems and to the presence of many complex constraints. In the theory of computational complexity, the hardness of such problems corresponds to the fact that they are *NP-hard*. The techniques available for solving MCOP fall into two main classes: *exact* and *approximate*. Exact methods guarantee to find an optimal solution and to prove its optimality for every finite size instance of a MCOP within an instance-dependent run time. In the worst case, their time complexity function is exponential. Typically, exact methods are limited to very small problems and are not practical for large scale real world problems. Approximate heuristic methods are often used instead of exact methods to overcome these limitations: they provide a solution to a problem in polynomial time with a certain performance guarantee. They are particularly appealing because of their flexibility, ease of implementation, and capability of attaining satisfactory performance on many large optimization problems.

Local search methods are heuristic methods for MCOP developed in Operations Research and Artificial Intelligence. The search process proceeds by trial and error:

starting from an initial solution, the search makes a sequence of local changes, called *moves*, chosen to improve the quality of the current solution as much as possible. This process is an informed search process as it exploits problem-specific knowledge. From a theoretical standpoint, local search algorithms sacrifice solution quality for performance. They may fail to find optimal solutions. However they usually isolate optimal or near-optimal solutions within very reasonable time constraints. For many problems, local search methods are recognized as state-of-the-art methods. Examples are traveling salesman problem ([Lin and Kernighan 1973]), propositional satisfiability problem (SAT) ([McAllester et al. 1997], [Selman et al. 1992], [Selman et al. 1994]), Vehicle Routing problem ([Gendreau et al. 2001]), and the rapidly growing bioinformatics area (see, e.g., [Fogel et al. 2002]).

Another issue in MCOP is the incentives of agents participating in the problem. In this context, the algorithm for solving MCOP also serves as a protocol for coordinating among agents. Agents are *autonomous* in the sense that they have control over their own subproblems, and can choose their actions freely. They are *intelligent*, in the sense that they can reason about the state of the world, the possible consequences of their actions, and the utility they would extract from each possible outcome. However, they are *resource-bounded* agents, in that they only have a limited computation capacity and a limited time for decision making. In real world situations, agents usually have conflicting preferences and they are *self-interested*, i.e., they want to maximize their own benefit while participating in the protocol. Therefore, agents will act strategically using all means available to them to maximize their profits. This strategic behavior will make the optimization problem difficult as it might severely damage the efficiency of the system and prevent it to accomplish the purposes that it was designed for.

1.2. Motivations and Objectives

Local search is widely used for solving large optimization problems in practice. Many different local search methods have been proposed in the last 30 years. In settings with multiple agents, it naturally adapts to local computation and synchronization through message exchange. In recent years there has been an increasing attention on the incentive aspects of the optimization algorithms in settings with multiple agents. However, the incentive aspects of local search algorithms remain open.

This thesis will aim at designing local search algorithms that take into account both optimization and incentives aspects. Our work will focus on satisfactory and sub-optimal outcomes rather than an optimal outcome. The thesis builds on the idea of using computational complexity as a barrier to prevent manipulations from self-interested agents.

We investigate two following scientific questions:

1. Can the local search techniques that are successful in SAT, which are based on a partial optimization function, be generalized to constraint optimization?
2. Can these techniques be adapted to prevent manipulation by self-interested agents in a multi-agent setting?

Contribution

The main contributions of this thesis can be summarized as follows.

Local search optimization algorithms

We introduce a new local search algorithm for MCOP called Random Subset Optimization (RSO) algorithm. The RSO algorithm is based on the idea of using randomization to guide the local search. The idea of the algorithm is to randomly choose a part of the optimization function, and make a move that is optimal with respect to this part only. It turns out that this results in a more focussed optimization strategy while retaining the power to escape from local minima. Our algorithms converge faster than other local search techniques while achieving a very competitive performance. Several variants and improvements of the RSO algorithm are also proposed in the thesis. The RSO algorithm can also be combined with other local search algorithms as a general *metaheuristics* for escaping from local optima.

Comparison of local search algorithms

We re-implement and compare with our algorithms some of the renowned local search algorithms including the Min-Conflicts (or Hill-Climbing) heuristics, Randomized Iterated Improvement, Tabu Search, and Simulated Annealing algorithm. The applications for testing are: network resource allocation problems, graph coloring with fixed number of colors (we call this problem as *soft graph coloring*), and classical graph coloring with minimum number of colors (we call *graph coloring*). All the applications are first transformed into the MCOP framework, and then solved by local search methods mentioned above.

In this thesis, we aim at an "unbiased" comparison, that is, a comparison in which all algorithms are allowed to use the same computational resources and work under the same conditions (same data structures and equal effort for tuning). We compare the algorithms with respect to performance criteria, convergence time criteria, and solution recovery criteria.

Size-limited incentive compatibility

In classical mechanism design, the concepts of incentive compatibility are defined based on the *equilibriums* in games. However, when problems move towards reality,

decision situations will become real-time and the agents will be bounded-rational. The concepts of incentive compatibility in classical mechanism design are hard to apply due to complexity and time constraints.

We define a new concept of incentive compatibility for bounded rational agents settings that takes into account their limited computational capacities. We show that RSO algorithms can be applied to have this property. Although a complete theoretical proof is difficult to achieve in average-case complexity, our experimental analysis results can provide intuitive answers.

1.3. Organization of the thesis

The thesis continues in Chapter 2 with background and related work in local search methods and classical mechanism design. The first part of this chapter introduces the multi-agent constraint optimization problem frame work and local search methods for solving it. The second part of the chapter provides an overview of the state of the art of the classical mechanism design.

In Chapter 3, we describe the RSO algorithm and its extensions. The comparison of the RSO algorithm and other local search algorithms is carried out on several applications: network resource allocation problems, soft graph coloring, classical graph coloring, and combinatorial auctions. For each application, we provide the method for transforming it into a multi-agent constraint optimization framework.

Chapter 4 is concerned with self-interested agents. We defined the new concept of *size-limited incentive compatibility* for bounded-rational agents. The RSO algorithm can be applied with a proper payment scheme so that it achieves the size-limited incentive compatibility property and budget balanced property at the same time. We give experimental evidence to show that this indeed makes sense.

Finally, Chapter 5 closes the thesis with a summary of the main contributions and directions for further research.

Chapter 2

Preliminaries and Background

The first two parts of this chapter introduce the Multi-agent Constraint Optimization framework and the local search techniques available to solve them. The classical mechanism design is introduced in the part three of the chapter.

2.1. Multi-agent Constraint Optimization Problems

In recent years, Constraint Optimization Problems (COP) ([Bertele and Brioschi 1972], [Schiex et al. 1995]) has emerged as the most successful paradigm for problem-solving. We begin this chapter by introducing the definition of a COP. Formally,

Definition 2.1. (COP) A discrete *constraint optimization problem* (COP) is a tuple $\langle X, D, R \rangle$ such that:

- $X = \{x_1, ..., x_n\}$ is a set of variables.
- $D = \{d_1, ..., d_n\}$ is a set of discrete, finite variable domains
- $R = \{r_1, ..., r_m\}$ is a set of utility functions, where each r_i is a function $r_i : d_{i_1} \times ... \times d_{i_k} \rightarrow \mathbb{R}$. Such a function assigns a utility (reward) to each possible combination of values of the variables. Negative utilities mean costs. Hard constraints (which forbid certain value combinations) are a special case of utility functions, which assign 0 to feasible tuples, and $-\infty$ to infeasible ones;

The goal of a COP is to find a complete assignment X^* for the variables x_i that maximizes (or minimizes) the sum of utilities of individual utility functions. Formally,

$$X^* = argmax_X(\sum_{r_i \in R} r_i(X)) \tag{2.1}$$

where the values of r_i are their corresponding values for the particular assignment X.

In settings with multiple agents, we extend the standard definition of constraint optimization as follows:

Definition 2.2. A discrete *multi-agent constraint optimization problem* (MCOP) is a tuple $\langle A, X, D, C, R \rangle$ where:

- $A = \{A_1, .., A_m\}$ is a set of agents.
- $X = \{x_1, .., x_n\}$ is a set of variables.
- $D = \{d_1, .., d_n\}$ is a set of domains of the variables, each given as a finite set of possible values.
- $C = \{c_1, .., c_p\}$ is a set of hard constraints, where a constraint c_i is a function $d_{i1} \times .. \times d_{il} \rightarrow \{0, 1\}$ that returns 1 if the value combination is allowed and 0 if it is not.
- $R = \{r_1, .., r_o\}$ is a set of relations (soft constraints), where a relation r_i is a function $d_{i1} \times .. \times d_{il} \rightarrow \mathbb{R}$ giving the utility of choosing each combination of values.
- R_{A_i} is the subset of R that gives the relations associated with agent A_i.

The goal of a MCOP is to maximize a global *objective function* which is an aggregate function f of the agents utilities

$$f(X) = \sum_{a \in A} \sum_{r_l \in R_a} r_l(X) \tag{2.2}$$

Definition 2.3. The solution to a MCOP is an assignment X^* of values to all variables x_i that satisfies all hard constraints and maximizes the sum of agent utilities as expressed by their relations

$$X^* = argmax_X(f(X)) \tag{2.3}$$

subject to

$$\forall i = 1, ..., p : C_i(X) = 1$$

We assume that variables, domains, constraints are common and agreed upon knowledge among the agents. On the other hand, relations are specified by the individual agents, and they do not necessarily have to report them correctly. For the first part of this thesis (Chapter 2 and Chapter 3), we assume that agents are expected to work cooperatively towards finding the best solution to the optimization problem, by following the steps the algorithm as prescribed. In Chapter 4 we relax the assumption that the agents are cooperative, and discuss algorithms with self-interested agents.

Many combinatorial optimization problems can be formulated as a MCOP. For example, the meeting scheduling problem ([Kaplansky and Meisels 2005]) can be formulated by a variable for the starting time of each meeting, constraints between any two meetings involving the same participants ruling out start times that would make them overlap, and relations that express each agent's preferences for meeting times. Additional constraints can be used to express precedence constraints between meetings or external constraints on their times.

Another example is allocating capacity in a public network, for example a train or pipeline network. The network is a graph of connections, and only one agent can use any one connection at a given time. This can be represented by having one variable per link and time interval whose domain ranges over the set of agents. Constraints would enforce for example that successive links and times are assigned to the same agent. Agents serve customers' transportation demands with different efficiency by using different combinations of links. Thus, each agent has utilities for being able to use certain combinations of links, and reports these as relations.

Agents want to find a combined assignment that maximizes the sum of their utilities. Such combinatorial optimization is NP-complete and thus can be solved exactly only for small problems. For large instances, in many cases only local search methods can be implemented. They can provide no optimality guarantees, but with high probability will find a solution that is very close to optimal.

2.2. Local Search Methods for Multi-Agent Constraint Optimization Problems

2.2.1. A General Framework for Local Search

The use of local search in combinatorial optimization reaches back to the late 1950s and early 1960s. It was first used for the traveling salesman problem and since then it has been applied to a very broad range of problems ([Aarts and Lenstra 1997]).

Generally, local search algorithms start from an initial solution and iteratively move from one solution to a neighboring solution in hope of improving the objective function. The decision in each step is based on local information only. The main

operation of a local search algorithm is a *move* from a solution v to one of its neighbors. The set of neighboring solutions of v, denoted by $N(v)$, is called the neighborhood of v.

The local search framework we assume in this thesis is depicted in Algorithm 2.1.

Algorithm 2.1: *Local Search algorithm for MCOP*

procedure *LocalSearch(A,X,D,C,R)*
  $v \leftarrow SelectInitialSolution()$
  repeat
    $v^{old} \leftarrow v$
    $N \leftarrow ChooseNeighbors(v^{old}, X, D, C)$
    $(v, \underline{pay}) \leftarrow LocalChoice(N, R)$
    agents make/receive payments according to $\underline{pay}$
  until termination condition met
  return v
end procedure

The algorithm manipulates a complete assignment of values to all variables, represented as a vector v. It is initially set by function *SelectInitialSolution* to an assignment that satisfies all constraints and could be random.

Search then proceeds iteratively by local improvements. Function *ChooseNeighbors* provides a set of candidate assignments that are close to the current one and could possibly improve it. As an implementation example, they are generated by randomly selecting a variable $x_i \in X$ and generating all assignments that are equal to v^{old} but assign to x_i different values in the domain of x_i that are consistent with the rest of v_{old} and the constraints in C.

In the second step of the iteration, the assignment v is updated using the function *LocalChoice*. It chooses a new assignment to optimize the combined utility according to the relations in R. It also computes a vector of payments $\underline{pay}$ that agents must make or receive in the third step of the iteration. The payments sum up to zero and the way they are derived is described in detail in Section 3.

The iteration continues until a termination condition is met, for example when there is no further improvement in the utility of all agents for some number of steps, or when the number of iterations has reached a maximum number allowed.

A *local optimum* solution is a solution which is better than all its neighboring solutions. The local search algorithm always ends at a local optimum solution. Notice that a global optimum solution is always a local optimum, but the converse is in general not true. To avoid getting stuck in local optima, local search algorithms often use metaheuristics that hopefully drive the search toward global optimality.

2.2.2. Metaheuristics for Escaping from Local Minima

Metaheuristics aim at escaping from local optima and at directing the search toward the global optimal solution.

Random Restart

Random restart is the simplest metaheuristic that iterates a specific local search from different starting points in order to sample various regions of the search space and to avoid returning to low quality local optima. This metaheuristic can be composed with other metaheuristics for local search.

The random restart metaheuristic is depicted in the algorithm 2.2. It start from an initial solution and then performs a number of iterations. Each iteration consists of a local search and the generation of a new starting point by perturbing the current local optimum or by generating a new initial solution.

Algorithm 2.2: *Random Restart Local Search algorithm*

```
procedure RR(A,X,D,C,R)
  v ← SelectInitialSolution()
  v* ← v
  for i = 1 : m do
    v' ← LocalSearch(A, X, D, C, R, v)
    if f(v') > f(v*) then
      v* ← v'
    end if
    v ← GenerateNewSolution(v)
  end for
  return v*
end procedure
```

Randomized Iterative Improvement

The idea of the randomized iterative improvement is simply to accept side walk steps. Side walk steps are moves that lead to candidate solutions whose objective function can be worse than the current solution. The problem in doing this is that if worsening solutions are accepted deterministically, there will be a possible cycling behavior. A way to avoid this drawback is to determine when a worsening step has to be performed by using a probabilistic criterion. The simplest algorithm which does this is Randomized Iterative Improvement (RII). In RII, a parameter $p \in [0, 1]$, called walk probability, is used to probabilistically determine whether a worsening

step or an improving step will be performed. With probability p, a candidate solution from the neighborhood $N(s)$ is chosen randomly, otherwise an improving candidate solution is taken. The RII algorithm is given in Algorithm 2.3.

Algorithm 2.3: *Randomized Iterative Improvement algorithm*

```
procedure RII(A,X,D,C,R,p)
  v ← SelectInitialSolution()
  v* ← v
  repeat
    N ← ChooseNeighbors(v, X, D, C)
    rp ← random([0, 1])
    if rp < p then
      v ← ChooseRandom(N)
    else
      v ← LocalChoive(N, R)
    end if
    if f(v) > f(v*) then
      v* ← v
    end if
  until termination condition met
  return v*
end procedure
```

Purely random moves have the disadvantage that they are not guided towards actually breaking out of a local optimum and thus they are rarely applied in practice. An exception are algorithms for SAT problems, in particular Walksat and its variants ([Selman et al. 1994], [McAllester et al. 1997], [Selman et al. 1992]) are among the state-of-the-art algorithms for SAT, in which only moves that satisfy at least one currently unsatisfied clause are considered. RII algorithms are also applied successfully to distributed sensor networks problems ([Zhang et al. 2005]).

An alternative idea to randomized iterative improvement, proposed by Kauffman *et al.* ([Macready et al. 1996], [Kauffman and Macready 1995]), is to parallelize the local search. As observed by the authors, optimization performance initially improves when parallelism increased, but then abruptly degrades to no better to random walk beyond a certain point.

Simulated Annealing

Simulated Annealing (SA) ([Kirkpatrick et al. 1983], [Cerny 1985]) is a popular metaheuristics based on the Metropolis heuristic ([Metropolis et al. 1953]). In this

heuristic, inspired by statistical mechanics, a degrading move is accepted with a probability given by the Boltzman probability distribution:

$$p_{accept}(T, v, v') = \begin{cases} 1 & \text{if } f(v) \geq f(v') \\ exp(\frac{f(v)-f(v')}{T}) & \text{otherwise.} \end{cases} \quad (2.4)$$

where s is the current solution, $s' \in N(s)$, T is the *temperature* parameter of the heuristic.

The temperature is set relatively high at the beginning of the search and is then decreased according to a cooling schedule. The exact characterization of the temperature over the whole search depends on the implementation.

The SA algorithm is depicted in Algorithm 2.4. The parameter T_0 is the initial temperature. The cooling schedule is performed by the *UpdateTemperature* function.

Algorithm 2.4: *Simulated Annealing algorithm*

```
procedure SA(A,X,D,C,R,T_0)
  v ← SelectInitialSolution()
  v* ← v
  T ← T_0
  repeat
    N ← ChooseNeighbors(v, X, D, C)
    v' ← ChooseRandom(N)
    v ← v' with probability p_accept(T, v, v')
    if f(v) > f(v*) then
      v* ← v
    end if
    UpdateTemperature(T)
  until termination condition met
  return v*
end procedure
```

Tabu Search

Tabu search (TS) is another popular and effective metaheuristics introduced independently by Glover ([Glover 1989], [Glover 1990]) and Hansen and Jaumard ([Hansen and Jaumard 1990]). An outline of the TS alorithm is shown in Algorithm 2.5.

In the TS algorithm, a short-term memory TT is used to avoid the search to return to recently visited solutions. A parameter tt, called *tabu tenure*, determines the duration of the search steps in which the re-insertion or removal of the solutions

Algorithm 2.5: *Tabu Search algorithm*

procedure *SimulatedAnnealing(A,X,D,C,R,tt)*
  $v \leftarrow SelectInitialSolution()$
  $v^* \leftarrow v$
  $InitTabuList(TT)$
  repeat
    $N \leftarrow ChooseNeighborsTabu(v, X, D, C, TT)$
    $v' \leftarrow LocalChoice(N)$
    if $f(v) > f(v^*)$ **then**
      $v^* \leftarrow v$
    end if
    $v \leftarrow v'$
    UpdateTabuList(TT)
  until termination condition met
  return v^*
end procedure

is forbidden. In practice, only a part of each solution is stored instead of the whole solution in order to save memory space and comparison time.

During the search, it may happen that some attractive solutions are forbidden. This situation can be avoided by using the *aspiration criteria*, which specifies conditions under which the tabu status of a candidate solution may be overridden.

Many refinements and variants have been proposed in the literature. Taillard ([Taillard 1991]) improves the robustness of the TS performance by choosing tt randomly from an interval $[tt_{min}, tt_{max}]$. In [Battiti and Protasi 2001], the authors propose a reactive mechanism to dynamically adjust the tabu tenure during the search based on the detection of trajectory repetitions. Glover and Laguna ([Glover and Laguna 1997])) report several ideas to include a long-term memory and extend the neighborhood set $N(s)$ through the inclusion of elite candidate solutions. They focus on the trade off between *intensification* and *diversification*, i.e., between exploiting the experience accumulated during the search and exploring new regions of the search space.

The tabu search and its improvements are still among the state-of-the-art for the graph coloring problem ([Hertz and de Werra 1987], [Fleurent and Ferland 1996], [Galinier and Hao 1999]).

Dynamic Local Search

Dynamic local search is another approach for escaping from local optima. The idea is to modify the objective function during the search in such a way that further

improvement steps become possible. This can be done by associating penalty weights to individual solution components which have an impact on the objective function. Whenever the search stucks in a local optimum, the penalties of some solution components present in the solution are changed and the objective function is updated with the new weights. The modified evaluation function is usually expressed in the form:

$$f'(v) = f(v) + \lambda \sum_{i=1}^{n} w_i I_i(v)$$

where w_i is the penalty weight of solution component i and $I_i(v)$ is an indicator function which returns 1 if the solution component i is present in v and 0 otherwise. The parameter λ is used to control the relative weight of the penalties on the evaluation function.

The penalties are initially set to zero and subsequently updated after each new iterative improvement run. The update may involve multiple solution components present in locally optimal solution. Variants of dynamic local search may differ in the update scheme.

In *Guided Local Search* (GLS) ([Mills and Tsang 2000]), solution component i is used to estimate the utility u_i of increasing the penalty weight of component i: $u_i = f_i(v)/(1 + w_i)$. Only solution components with maximal utility values are updated by setting $w_i = w_i+1$. Thus solution components with high negative impact in the solution should increase their penalties during the search. GLS procedure is depicted in Algorithm 2.6.

In constraint optimization problems, GLS may associate weights with constraints and components of the objective function. The weights of the violated constraints can be increased each time the local search produces a new solution. By increasing the weight of a constraint c, GLS makes it more likely to explore solutions satisfying c.

Evolutionary Algorithms

Evolutionary algorithms (EA) are approaches inspired by natural evolution. Recommended books for EA in combinatorial optimization are [Holland 1992], [Goldberg 1989], and [Michalewicz and Fogel 2004].

In EA algorithms, a population of candidate solutions is maintained and a series of genetic operators are repeatedly applied to replace, partially or totally, the population with the next one. Typically, two operators are used to modify a solution: a *mutation* operator which modifies an individual of the population by random changes, and a *recombination* operator which generates one or more individuals by combining information from two or more other individuals. Finally, a selection criterion chooses the solutions for the next generation based on their fitness (usually

Algorithm 2.6: *Guided Local Search algorithm*

procedure *GLS(A,X,D,C,R)*
  $v \leftarrow SelectInitialSolution()$
  $v^* \leftarrow v$
  Initialize penalty weights w
  repeat
    Update objective function $f'(v) = f(v) + \lambda \sum_{i=1}^{n} w_i I_i(v)$
    $v' \leftarrow LocalSearch(A, X, D, C, R, v, f')$
    if $f(v') > f(v^*)$ **then**
      $v^* \leftarrow v'$
    end if
    for all $i \in argmax_i(u_i(v))$ **do**
      $w_i = w_i + 1$
    end for
  until termination condition met
  return v^*
end procedure

associated with the value of the objective function). Individuals with higher fitness have higher probability of being selected.

There are two popular variants of EA in combinatorial optimization: *Genetic algorithms* and *Memetic algorithms*. Genetic algorithms represent a solution based on bit strings of equal length. Memetic algorithms are genetic algorithms in which a local search procedure is applied after the mutation and recombination operators. Memetic algorithms typically achieve better performance than Genetic algorithms in most of the cases.

An outline of a general memetic algorithm is given in the algorithm 2.7.

An example of application of the memetic algorithms is graph coloring problem ([Fleurent and Ferland 1996], [Dorne and Hao 1998], [Galinier and Hao 1999]).

Ant Colony Optimization

Ant Colony Optimization (ACO) ([Dorigo et al. 1999])is a nature-inspired, population-based metaheuristic. The main idea of ACO, loosely inspired by the behavior of real ants, is that of a parallel search over several constructive computational threads based on local problem data and on a dynamic memory structure containing information on the quality of previously obtained result.

In each iteration of ACO, a population of k candidate solutions is generated by a construction procedure that uses probabilistic decisions. Next, the solutions are improved by an iterative improvement procedure, resulting in a population of locally

Algorithm 2.7: *Memetic algorithm*

procedure *Memetic(A,X,D,C,R)*
  $pv \leftarrow GenerateInitialPopulation()$
  $v^* \leftarrow argmin_{v \in pv'}(f(v))$
  repeat
    $pv' \leftarrow Recombination(pv)$
    $pv'' \leftarrow Mutation(pv' \bigcup pv)$
    $pv' \leftarrow LocalSearch(pv' \bigcup pv'')$
    if $\min_{v \in pv'} f(v) > f(v^*)$ **then**
      $v^* \leftarrow argmin_{v \in pv'}(f(v))$
    end if
    $pv \leftarrow Selection(pv')$
  until termination condition met
  return v^*
end procedure

optimal solutions. Finally, they are updated in such a way that it bias the search toward components found in high-quality solutions. The ACO procedure is outlined in Algorithm 2.8.

Earlier version of ACO, called Ant System ([Dorigo 1992]), achieved only poor results. A more successful refinement of ACO is MAX-MIN Ant System which has a peculiar update procedure ([Stützle and Hoos 2000]). For a comprehensive coverage of Ant Colony Optimization we refer the reader to Dorigo and Stützle ([Dorigo and Stützle 2004]).

2.3. Incentive Compatibility

There exists a huge literature in the classical mechanism design and computational mechanism design which we can not mention in full generality within the scope of this thesis. In this section, we will give some of the most important results in the mechanism design literature that related with the thesis.

2.3.1. Important concepts in mechanism design

In mechanism design, we consider settings in which each agent has its own preferences over different outcomes. The agents are *self interested*, in the sense that each one would like to obtain the decision that maximizes its own utility. A function that choose an outcome given agents' preferences is called *social choice function*. Let R_i be the set of possible preferences of agent i, X is the set of possible outcomes

Algorithm 2.8: *Ant Colony Optimization algorithm*

procedure *ACO(A,X,D,C,R)*
  Initialize s^*;
  repeat
    Construct a population of solutions S
    for all $s \in S$ **do**
      $s \leftarrow LocalSearch(s)$
    end for
    if $\min_{s \in S} f(s) < f(v^*)$ **then**
      $s^* \leftarrow argmin_{s \in S}(f(s))$
    end if
    $Update(s^*, S)$
  until termination condition met
  return s^*
end procedure

Definition 2.4. A *social choice function* $f : R_1 \times ... \times R_m \rightarrow X$ chooses an outcome $f(r) \in X$, given preferences $r = (r_1, ..., r_m)$.

In our context of multi-agent constraint optimization problems, the preferences of an agent a_i are its set of relations R_{a_i}, and the outcomes are possible assignments of values to all variables.

A *mechanism* (or *protocol*) defines the strategies available for agents and the method used to select the outcome based on agent strategies. We say that a mechanism implements social choice function f if the outcome computed with equilibrium agent strategies is a solution to the social choice function for all possible agent preferences.

The mechanism design problem is to define possible strategies and the method used to select an outcome based on agent strategies, to implement the solution to the social choice function despite agent's self-interest.

The concept of *incentive compatibility* (IC) was first introduced by Hurwicz in his seminal paper [Hurwicz 1972] and followed by other authors [d'Aspremont and Gérard-Varet 1979], [Green and Laffont 1977], [Jackson 1991]. The strongest form of IC is strategy-proof, where truth telling is dominant strategy for agents regardless of others' strategies. Another form of IC is *Bayesian incentive compatible*, where truth telling is in Bayesian-Nash equilibrium for agents. This solution concept requires assumptions about information of prior distributions of agents' preferences, which is difficult in real world problems.

Definition 2.5. (Incentive compatible) We say that a mechanism M is *incentive-compatible* if truth-telling is an equilibrium of the game induced by M.

Depending on the protocol, incentive-compatibility often means that each agent is reporting its relations truthfully, thus one also speaks of *truthful* protocols. Clearly, incentive-compatibility is important to obtain a meaningful solution to the MCOP. It is often achieved through tax or auction mechanisms such as the VCG mechanism ([Clarke 1971; Groves 1973; Vickrey 1961]).

Other important solution concepts in mechanism design are individual rationality, efficiency, and budget-balance.

Definition 2.6. (Individual rationality) We say that a mechanism is *individually rational* (IR) if if the expected utility that each agent gets when it participates in the protocol is at least as high as non-participation.

In other words, a mechanism is individually rational it is in best interest of each agent to participate in the protocol. The IR property is important because otherwise, agents may choose not to participate in the protocol.

Definition 2.7. (Efficiency) We say that a solution to a mechanism is *allocatively-efficient* if it maximizes the total utility over all agents.

An efficient solution is the optimal solution to the protocol.

Definition 2.8. (Budget-balance) We say that a mechanism is *budget-balanced* (BB) if the total payments from all agents to the mechanism is zero.

In other words, there are no net transfers out of the protocol or into the protocol.

Definition 2.9. (Weak budget-balance) We say that a protocol is *weak budget-balanced* if and only if the total payments from all agents is non-negative.

In other words, there can be a net payment made from agents to the protocol, but no net payment from the protocol to the agents.

2.3.2. Revelation principle

One important result called *revelation principle* ([Gibbard 1973], [Green and Laffont 1977], [Myerson 1979]) states that under weak conditions any mechanism can be transformed into an equivalent incentive-compatible mechanism. This means that if a mechanism satisfies certain properties (e.g., Individual rational, budget-balanced), the IC property can also be satisfied using an equivalent direct-revelation truthful mechanism. However, it is usually difficult to design direct-revelation mechanisms but it can be easier for indirect mechanisms ([Parkes and Ungar 2000], [Conen and Sandholm 2002]).

2.3.3. Impossibility results

The revelation principle allows the derivation of a number of impossibility results that outline the combinations of properties that no mechanism can achieve in particular types of environments. These results assume direct-revelation and incentive-compatibility, express the desired properties of an outcome rule as a set of mathematical conditions, and then show a conflict across the conditions.

Fundamental impossibility results ([Green and Laffont 1977], [Green and Laffont 1979], [Myerson and Satterthwaite 1983]) show that in general settings, it is impossible to achieve mechanisms that satisfy budget-balance, Pareto-efficiency, and incentive compatibility at the same time. Similar impossibility results in the voting environment include [Arrow 1963], [Gibbard 1973], and [Satterthwaite 1975]. These impossibility results are starting points for further research. There are two general approaches to overcome these results: the first one is to relax one of the properties when designing mechanisms; the second one is to relax the general setting assumption, designing mechanisms in particular and restricted settings.

2.3.4. Directions to achieve incentive compatibility

In the first approach, the most important result are the VCG mechanisms ([Clarke 1971; Groves 1973; Vickrey 1961]). These mechanisms are incentive compatible, efficient, and individual rational (Clarke mechanism), but they are not budget-balanced. However, VCG mechanisms are computationally intractable for both the mechanism designer to compute the optimal outcome and for the agents to compute their complete preferences. Moreover, combining VCG tax schemes and a non-optimal choice function breaks the IC property [Nisan and Ronen 2000]. Some researchers based on the VCG tax schemes to design mechanisms that impose budget-balance and satisfy only approximate efficiency and incentive compatibility ([Parkes et al. 2001]).

Another direction in the first approach is to relax the efficiency and design mechanisms that satisfy other properties (e.g., incentive compatible, budget-balanced, individual rational), this includes the works of [Myerson and Satterthwaite 1983], [McAfee 1990], [Barberà and Jackson 1995]. Other works in [Lehmann et al. 2002], [Mu'alem and Nisan 2002] followed this direction but they assume only restricted settings.

Using weaker forms of the incentive compatibility can also extend implementable mechanisms. d'Aspremont and Grard-Varet showed that it is possible to achieve efficiency and budget-balance with Bayesian incentive compatibility in the dAGVA mechanism ([d'Aspremont & Grard-Varet 79]). However it requires information about prior distribution of agents' preferences, so it is unrealistic in real world settings.

A more recent direction in the first approach is to use approximation in the incentive compatibility concept. Schummer ([Schummer 1999]) proposed *almost truthful* concept, where agents cannot gain too much from manipulation. Other researchers consider the effects of intractability on bounded-rational agent. In [Nisan and Ronen 2000], the authors proposed feasibly truthful mechanisms in which agents cannot manipulate without solving a NP-hard problem. They also proposed a *feasibly truthful* mechanism which is a variant of VCG-based mechanisms called second-chance mechanism. However, their mechanism is not realistic for bounded-rational agents as the appeal function requires extra knowledge of the algorithm from agents. Some researchers recently characterized incentive compatible mechanisms even with inefficient outcome ([Bikhchandani et al. 2003]). However, their characterization is only for single-item multi-unit auctions. For other settings it is still an open question.

In the second approach, researchers design incentive compatible mechanisms by assuming that the domain of admissible preferences is restricted. For example, in the quasi-linear utility environment, some authors developed approximation mechanisms for *single-minded* agents ([Lehmann et al. 2002], [Mu'alem and Nisan 2002]) and *single-parameter* agents ([Archer and Tardos 2001]). In the voting environment, Moulin shows that if the preferences are assumed to be *single-peaked* ([Moulin 1980]), then the median voter rule provides appropriate incentives for all agents to be truthful. Bogomolnaia and Moulin ([Bogomolnaia and Moulin 2001]) also show that it is possible to design randomized truthful mechanisms for dichotomous preferences. Sandholm and Conitzer ([Conitzer and Sandholm 2002], [Sandholm 2003]) show that if the preferences are explicitly given or structured, then it is possible to search for mechanisms that satisfy certain properties by automated mechanism design. However, this direction only works with finite preference spaces and it requires the mechanism designer to have prior knowledge about agents preferences.

2.3.5. Vickrey-Clarke-Groves (VCG) mechanism

The Vickrey-Clarke-Groves (VCG) mechanism is a widely-used mechanism in mechanism design. In seminal papers Vickrey ([Clarke 1971; Groves 1973; Vickrey 1961]), proposed the Vickrey-Clarke-Groves family of mechanisms (also called Groves mechanism) for problems in which agents have quasi-linear preferences. The VCG mechanisms are allocatively-efficient and strategy-proof direct-revelation mechanisms. In special cases there is a Groves mechanism that is also IR and satisfies weak BB, for example, the VCG mechanism for a combinatorial auction.

The special case of VCG mechanism for the allocation of a single item is the familiar second-price sealed-bid auction (auction with private bids), or Vickrey auction [Vickrey, 1961]. In this case, the item will be awarded to the agent with highest bid at the second highest price. The Vickrey mechanism satisfies IR, strategy-proofness and Pareto efficiency.

In VCG mechanism, given reported preferences $r = (r_1, ..., r_m)$, the choice rule must compute the efficient outcome that maximizes the total reported value over all agents:

$$v^* = \arg\max_v \sum_{r \in R} r(v)$$

The payment rule for VCG mechanism is computed as follows: for an agent a_i, the VCG payment is the "damage" it does the others, i.e., the decrease in utility gain its presence causes to the remaining agents:

$$VCGTax(a_i) = \sum_{r \in R \setminus \{r_i\}} r(v^*_{A \setminus \{a_i\}}) - \sum_{r \in R \setminus \{r_i\}} r(v^*_A) \tag{2.5}$$

where v^*_A is the optimal outcome with respect to the set of agent A, $v^*_{A \setminus \{a_i\}}$ is the optimal outcome when agent a_i is not participated in the protocol.

Note that since $v^*_{A \setminus \{a_i\}}$ is optimized for $A \setminus \{a_i\}$, the sum of its utilities for these agents will always be at least as large as that for v^*_A and thus the $VCGtax$ is never negative. Thus, the payments of all agents together leave a positive budget surplus.

The VCG payment is a special case of the Groves taxes:

$$GrovesTax(a_i) = h_i(.) - \sum_{r \in R \setminus \{r_i\}} r(v^*_A) \tag{2.6}$$

where $h_i : R_{-i} \rightarrow \mathbb{R}$ is an arbitrary function on the relations of all agents except a_i. This fredom in selecting $h_i(.)$ loeds to a family of mechanisms. Different choices of $h_i(.)$ make different tradeoffs across budget-balance and individual-rationality.

In [Green & Laffont, 1977], it is shown that the VCG family of mechanisms are the *only* mechanisms that are allocatively-efficient and strategy-proof among direct-revelation mechanisms. Krishna and Perry [Krishna & Perry, 1998], and Williams [Williams, 1999] have recently proved the uniqueness of VCG mechanisms among efficient and Bayesian-Nash mechanisms.

VCG mechanisms demonstrate that it is possible to implement allocatively-efficient (but not budget-balanced) strategy-proof mechanisms in quasi-linear domains. However, the impossibility results of Green and Laffont [Green & Laffont, 1977] show that they are not efficient and budget-balanced.

The seminal work of Ephrati and Rosenschein ([Ephrati and Rosenschein 1991]) was the first attempt to propose applying VCG mechanisms to agent coordination in settings with multiple agents. For constraint optimization, game theory has shown that the only practical mechanism for incentive-compatibility in MCOP is of the form of a VCG mechanism ([Green and Laffont 1977]). However, it has also been shown that VCG mechanisms require finding the provably optimal solution ([Nisan and Ronen 1999]). Many practical settings of optimization problems are too large

for complete optimization algorithms. Local search algorithms can quickly find a good approximation to an optimal solution. However, VCG mechanisms cannot be used directly with local search algorithms. In this thesis we thus introduce a weaker concept of incentive-compatibility, called *size-limited incentive compatibility*, where manipulation is hard through computational complexity. The uncertainty created by randomized local search makes it computationally intractable to evaluate the outcome of an untruthful behavior, thus rendering it uninteresting to agents.

Chapter 3

Random Subset Local Search Optimization

In this chapter, we introduce the Random Subset Optimization algorithms. We then present some extensions of the algorithm in different ways. We compare the performance of our algorithm with other local search techniques in several benchmark problems including Network Resource Allocation, Soft Graph Coloring, and Graph Coloring. Our empirical analysis gives a detailed view of the algorithm's performance and suggests the problem domains where it performs the best.

3.1. Random Subset Optimization algorithm

The objective function of the MCOP described in the previous chapter is the total of soft constraints and it can be decomposed into parts. The *random subset optimization* (RSO) ([Faltings and Nguyen 2006], [Nguyen and Faltings 2005], [Faltings and Nguyen 2005b]) algorithm comes from the idea that when the local search is stuck in a local minimum, only some part of the objective function is maximized locally and other parts of it need to be optimized elsewhere. The RSO algorithm randomly chooses a part of the optimization function, and makes a move that is optimal with respect to this part only. It turns out that this strategy results in a more focussed optimization strategy while retaining the power to escape from local minima. A similar randomized technique for escaping from local minima is the Walksat algorithm ([Selman et al. 1994]), which is one of the most successful algorithms for SAT. However, Walksat applies randomization in a slightly different way: it randomly picks a literal that occurs in an unsatisfied clause and flipping it, thus satisfying that clause. This kind of random walk is more directed towards actually improving some aspect of the solution.

The RSO algorithm is based on the generic local search framework presented earlier in the Chapter 2. It is useful to depict this framework again in Algorithm

3.1. We will leave out the payment elements in the algorithm as we only focus on the optimization part in this chapter.

Algorithm 3.1: *Local Search algorithm for MCOP*

procedure *LocalSearch(A,X,D,C,R)*
  $v \leftarrow SelectInitialSolution()$
  repeat
    $v^{old} \leftarrow v$
    $N \leftarrow ChooseNeighbors(v^{old}, X, D, C)$
    $(v, \underline{pay}) \leftarrow LocalChoice(N, R)$
    agents make/receive payments according to $\underline{pay}$
  until termination condition met
  return v
end procedure

Algorithm RSO, as shown in the algorithm 3.2, is a development of Algorithm 3.1 and introduces two random elements.

First, the neighborhood set N is generated by randomly choosing a variable x_k and considering as neighbors all assignments that differ in the value given to that variable. Function $ChooseRandom(X)$ returns the x_k for considering in the current iteration. The variable x_k is chosen randomly from the set of variables which are present in an unsatisfied hard constraint. If the set of hard constraints is empty, x_k is chosen from the set of variables X.

Second, when the algorithm reaches a local optimum, the algorithm chooses with probability p to optimize only for a subset $\tilde{R}$ of the constraints making up the objective function, and with probability $(1 - p)$ to choose a random neighbor regardless of whether it improves the objective function or not. The subset $\tilde{R}$ is chosen by function $ChooseRandomSet$ that takes as an additional parameter d the number of soft constraints that are dropped from the set R. The algorithm then chooses a value for x_k in the neighborhood set N that optimizes for the subset $\tilde{R}$ only. In this way, the search avoids moves that only degrade the solution and rapidly lead back to the original optimum, and focusses on moves that actually have a chance to break out of it. This technique can be applied whenever the objective function can be decomposed into parts, as for example when it is given as a set of soft constraints.

The RSO algorithm terminates when a termination condition is met, for example, when there is no improvement after a certain number of steps, or when a maximum number of steps is reached. In the experiments we report later, it appears that the best performance is reached when d is small and p is close to 1.

Algorithm 3.2: *RSO algorithm for MCOP*

procedure *RSO(X,D,C,R,p,d)*

1: $v \leftarrow SelectInitialSolution()$
2: $v^* \leftarrow v$
3: **repeat**
4: $v^{old} \leftarrow v$
5: $x_k \leftarrow ChooseRandom(X)$
6: $N \leftarrow ChooseNeighbors(x_k, v^{old}, X, D, C, R)$
7: **if** $LocalOptimum(v)$ **then**
8: **if** $random(0..1) \leq p$ **then**
9: $\tilde{R} \leftarrow ChooseRandomSet(R, d)$
10: $v \Leftarrow LocalChoice(N, \tilde{R})$
11: **else**
12: $v \Leftarrow ChooseRandom(N)$
13: **end if**
14: **else**
15: $v \Leftarrow LocalChoice(N, R)$
16: **end if**
17: **if** $f(v) > f(v^*)$ **then**
18: $v^* \leftarrow v$
19: **end if**
20: **until** termination condition met
21: return v^*

end procedure

3.1.1. Parameters of RSO

In each local search step of the RSO algorithm, one variable x_k is chosen randomly for changing value. However this is not limited to single variable-neighborhood, but the RSO algorithm can be applied to any kind of neighborhood. For example, in the experiments for the graph coloring problem we report later in the next section, we also implemented RSO algorithm for the path-exchange and s-impasse neighborhood structures.

Size of the subset

The parameter d is the number of soft constraints that are taken out from the set R. If d is large, the search performs similar to random walk. In an extreme case, only one agent is considered for optimization in each iteration. This will drive the search in the direction starting from the agent being considered in the set $\tilde{R}$ at the current iteration. If d if small, the search behaves similar to the hill-climbing algorithm. When the search is close to the global optimum, the value of d should be small in order to avoid the search jump out out the global optimum region.

Probability of random walk

In the RSO algorithm, p is the probability that the search will perform a random subset optimization in each iteration. With a small probability $1-p$, RSO algorithm allows a random move in order to escape from the current local optimum. It helps the search when the RSO heuristics keep staying in the current local optimum region. If the probability $1-p$ is large, the search behaves similar to to the random walk, thus have more chance to reach the global optimum region. However the convergence time of the algorithm is longer. When the search close to the global optimum region, the probability $1-p$ should be close to 0. In our implementation, we always set the the probability p to a value close to 1.

In Section 3.4, we will give further experiments on how the behavior of RSO algorithms depend on parameters d and p.

3.1.2. Completeness

The outcome of randomized algorithms cannot be predicted with certainty, and so the notion of completeness cannot be applied directly here. However, Hoos [Hoos 1999] has defined the notion of *probabilistically approximately complete* (PAC):

Definition 3.1. A randomized search algorithm is probabilistically approximately complete (PAC) if for all solvable problem instances, the probability $p(t)$ that it finds the optimal solution in time less than t goes to 1 as t goes to infinity.

Pure random walk strategies, such as picking random values without regard for whether they improve or degrade solution quality, make a local search algorithm PAC: starting with any initial state, there is a sequence of moves that sets each variable x_i to the value it has in an optimal assignment v_i^*, and this sequence of moves has a non-zero probability.

The RSO strategy by itself, reached when the parameter p of Algorithm 3.2 is set to 1, is clearly not PAC, as there are problems where the optimal assignment $\mathbf{v}^*$ is not reachable by the algorithm at all. This is the case when the optimal assignment is a compromise among all relations, i.e. there is no subset of the relations for which $\mathbf{v}^*$ is also optimal. When $p < 1$, the algorithm also performs a random walk with some probability and does become PAC. However, the probability is very small, in fact equal to the probability of picking the optimal assignment at random among all possible assignments, so that the guarantee is theoretical at best.

We first have the following result for mixed-RSO algorithm as an immediate result from the Markov chain theory:

Theorem 3.2. The RSO algorithm is PAC for any $p < 1$.

Proof. The corresponding Markov chain V_t of the RSO algorithm is an ergodic Markov chain as the transition probability from one state to any other state is positive if $p < 1$. Thus starting from any assignment, the algorithm will eventually reach the optimal assignment if the algorithm is run long enough. ■

We can consider the subclass of decomposable COP:

Definition 3.3. Let $\mathbf{v}^* = (v_1^*, v_2^*, ..., v_n^*)$ be an optimal assignment to the variables X of a COP. We call the COP *decomposable* if and only if for every subset of variables $Y \subseteq X$ and any consistent assignment of values to variables in $X - Y$, there is some subset of relations $S \subset R$ such that $\mathbf{v}_Y^*$, the subset of $\mathbf{v}^*$ restricted to variables in Y, is an optimal assignment for Y, given the values of the other variables.

and we can show:

Theorem 3.4. The RSO algorithm is PAC for decomposable COP.

Proof. We give a constructive proof by showing a sequence of moves that constructs the optimal assignment $\mathbf{v}^*$ starting from any initial state and has a non-zero probability. The sequence starts with $Y = \{x_1\}$ and chooses to change x_1 by optimizing the subset of soft constraints that results in $x_1 = v_1^*$. In the following steps, it chooses x_k and the subset of soft constraints that makes $x_1 = v_1^*, ..., x_k = v_k^*$ optimal. After n moves, the entire assignment is constructed. The sequence has non-zero probability as each move is among those randomly considered at each step, so it will eventually be carried out if the algorithm is run long enough. ■

An example of a decomposable constraint optimization problem is MAX-CSP, where the objective is to find a variable assignment that satisfies a maximum number of constraints. Let $Q \subseteq R$ be the constraints that are satisfied in the optimal solution $\mathbf{v}^*$. For a subset Y of variables, let $S \subseteq Q$ be a subset that only refers to variables in Y. The values that variables in Y take in the optimal solution $\mathbf{v}^*_Y$ satisfy all constraints in S, so this assignment is optimal for this subproblem.

As a consequence, MAX-SAT and Soft-coloring are also decomposable problems, and Theorem 3.4 is in line with the completeness results for the GWSAT algorithm reported in [Hoos 1999]).

However, not all optimization problems are decomposable, for example Graph Coloring Optimization - coloring a graph with a minimal number of colors - is not decomposable.

This can be seen on the following example: consider a graph of 4 nodes x_1 through x_4 and three edges (x_1, x_2), (x_2, x_3), (x_3, x_4). An optimal coloring is $x_1 = blue$, $x_2 = red$, $x_3 = blue$, $x_4 = red$. Consider the set $Y = \{x_1, x_4\}$ and let $x_2 = green$ and $x_3 = blue$. Then, $x_1 = blue$, $x_4 = red$ is no longer an optimal coloring for Y under any subset of the relations, since it can be colored with just one color.

In decomposable problems, optimization can be localized and they can thus be solved by solving a set of smaller subproblems, provided that the right subproblems are identified. Several researchers have proposed techniques for distributed optimization that work well for decomposable problems. For example, the DCOP algorithm [Mailler and Lesser 2004] composes an optimal solution from solutions that are optimal for subproblems. In general, we can expect non-decomposable problems to be hard for local search algorithms.

3.2. Extensions of RSO

The RSO algorithm can be improved by changing the way its parameters are set and by combining with other local search algorithms.

Adaptive RSO (ARSO)

In each step of the RSO algorithm, we choose randomly a variable, a subset of constraints and optimize it for the chosen random subset. Thus the later steps can break the local optimized region constructed in the previous steps. The idea of the adaptive choice of the subset is to continue to optimize for the local optimized regions constructed in the previous steps. In this approach, we continue to choose the subset to optimize in later steps in such a way that the chosen subset is a subset of the subset chosen in the previous steps for several steps or until there is no improvement for the subset for several steps.

The Adaptive RSO (ARSO) algorithm is shown in the algorithm 3.3. The difference between the ARSO and RSO algorithms is the loop FOR in whenever the search stucks at a local minimum. The ARSO then will perform several RSO steps with the subset chosen by the function *ChooseRandomSet*. This strategy results in a stronger informative way to guide the search to escape from the current local optimum.

Algorithm 3.3: *Adaptive RSO algorithm*

procedure *ARSO(X,D,C,R,p,d)*

$v \leftarrow SelectInitialSolution()$
$v^* \leftarrow v$
repeat
  $v^{old} \leftarrow v$
  $x_k \leftarrow ChooseRandom(X)$
  $N \leftarrow ChooseNeighbors(x_k, v^{old}, X, D, C, R)$
  $\tilde{R} \leftarrow ChooseRandomSet(R, d)$
  if $LocalOptimum(v)$ **then**
    $\tilde{R} \leftarrow R$
    for i = 1 : MAXTRIES **do**
      $\tilde{R}' \leftarrow ChooseRandomSet(\tilde{R}, d)$
      $x_k \leftarrow ChooseRandom(X)$
      $N \leftarrow ChooseNeighbors(x_k, v^{old}, X, D, C, \tilde{R}')$
      $v \Leftarrow LocalChoice(N, \tilde{R}')$
      if $f(v) > f(v^*)$ **then**
        $v^* \leftarrow v$
      end if
    end for
  else
    $v \Leftarrow LocalChoice(N, R)$
    if $f(v) > f(v^*)$ **then**
      $v^* \leftarrow v$
    end if
  end if
until termination condition met
return v^*

end procedure

Annealing-RSO (SA-RSO)

Our experiments shown that it is interesting to combine RSO with the simulated annealing heuristics. In the SA-RSO, a cooling strategy is used to adaptively change the number d of soft constraints to leave out when attempting to escape from local minima. We let the parameter d start with a high value d_0 and decrease it over time until it reaches zero. At each value of d, the algorithm SA-RSO accept a degrading move with a probability given by the Boltzman probability distribution in the equation 2.4:

$$p_{accept}(d, v, v') = \begin{cases} 1 & \text{if } f(v) \geq f(v') \\ exp(\frac{f(v)-f(v')}{d}) & \text{otherwise.} \end{cases} \quad (3.1)$$

The SA-RSO algorithm is depicted in the algorithm 3.4.

Algorithm 3.4: *Annealing RSO algorithm*

procedure *SA-RSO(X,D,C,R,p,d_0)*

1: $v \leftarrow SelectInitialSolution()$
2: $v^* \leftarrow v$
3: $d \leftarrow d_0$
4: **repeat**
5: $v^{old} \leftarrow v$
6: $x_k \leftarrow ChooseRandom(X)$
7: $N \leftarrow ChooseNeighbors(x_k, v^{old}, X, D, C, R)$
8: $\tilde{R} \leftarrow ChooseRandomSet(R, d)$
9: $v' \leftarrow LocalChoive(N, \tilde{R})$
10: $v \leftarrow v'$ with probability $p_{accept}(d, v, v')$
11: **if** $f(v) > f(v^*)$ **then**
12: $v^* \leftarrow v$
13: **end if**
14: UpdateTemperature(d)
15: **until** termination condition met
16: return v^*

end procedure

RSO-tabu

Tabu search technique can combine with RSO in several ways: we can use tabu lists to avoid recent visited solutions or recent selected subsets during the search. The tabu list length can be chosen randomly from an interval as proposed in [Taillard 1991].

In the RSO-tabu algorithm depicted in the algorithm 3.5, the function *ChooseRandomSet* is replaced by *ChooseRandomSet_tabu* with the tabu tenure parameter tt.

Algorithm 3.5: *RSO-tabu algorithm*

procedure *RSO(X,D,C,R,p,d)*

1: $\underline{v} \leftarrow SelectInitialSolution(X, D, C)$
2: **repeat**
3: $\underline{v^{old}} \leftarrow \underline{v}$
4: $x_k \leftarrow ChooseRandom(X)$
5: $N \leftarrow ChooseNeighbors(x_k, \underline{v^{old}}, X, D, C, R)$
6: **if** $LocalOptimum(\underline{v})$ **then**
7: **if** $random(0..1) \leq p$ **then**
8: $\tilde{R} \leftarrow ChooseRandomSet\_tabu(R, d, tt)$
9: $\underline{v} \Leftarrow LocalChoice(N, \tilde{R})$
10: **else**
11: $\underline{v} \Leftarrow ChooseRandom(N)$
12: **end if**
13: **else**
14: $\underline{v} \Leftarrow LocalChoice(N, R)$
15: **end if**
16: **until** termination condition met
17: return $\underline{v}$

end procedure

3.3. Distributed, asynchronous implementation

We define the distributed constraint optimization problem as a constraint optimization problem where each variable is controlled by an agent that is responsible for setting its value. We further define the neighborhood of a variable x_i to consist of all other variables that share a constraint with x_i. We assume that the agent controlling a variable knows all soft and hard constraints that involve the variable, and communicates with the agents controlling all variables in the neighborhood so that it always knows the values currently assigned to these neighboring variables.

Since the impact of a variable on the total utility or cost is completely determined by the set of soft constraints it participates in, it can be evaluated based only on the neighborhood. Consequently, a local search algorithm can be implemented in parallel where each agent decides whether to change its variable based only on

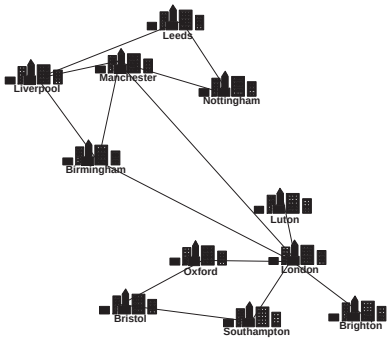

Figure 3.1. *Network of transputers and the structure of individual processes*

communication with the agents controlling its neighbors. Such an algorithm is given for example by Zhang ([Zhang et al. 2005]).

Changes to different variables that are not neighbors of each other can take place in parallel and even asynchronously, as their impact is independent of each other. In order to ensure that neighbors do not change at the same time and thus invalidate the local choice rule, a synchronization mechanism is needed. We use a mechanism where each variable compares the utility improvement that can be obtained by changing it with the potential improvement for each of its neighbors. Only the variable with the highest improvement is allowed to change value.

We implemented distributed random subset optimization using this mechanism and compared its performance with that of the centralized version. As agents need to compare the improvements for different variables, we ensured that for all variables, the same soft constraints are left out by randomly choosing for each soft constraint whether it should be left out or not.

We observed that such distribution does not seem to affect the quality of the final solution that can be obtained with random subset optimization. This shows that it can be applied to distributed scenarios without difficulty.

3.4. Applications

3.4.1. Network Resource Allocation

The network resource allocation problem consists of allocating tracks in the train network shown in Figure 3.1 to different operators. To avoid collisions, each arc in the graph can only be allocated to one operator who can then run trains on it. Operators transport goods between nodes in the network. Each transportation task can take one of 3 feasible routes and generates one variable whose domain is the agent and route assigned to it. For example, if task 3 $(London, Manchester)$ is assigned to agent a_1 on the route $(London \rightarrow Birmingham \rightarrow Manchester)$, the corresponding

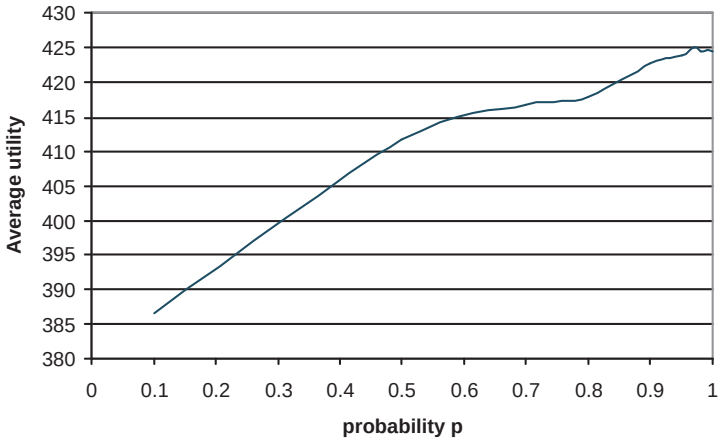

Figure 3.2. *Variation of the average cost of the random subset optimization algorithm on network resource allocation problem, for different values of p.*

variable x_3 is assigned the value $(a_1, London \rightarrow Birmingham \rightarrow Manchester)$. The problem is representative of a large class of resource allocation problems, for example in communication networks or in chemical plants.

The network capacity is modelled by binary constraints between any pair of tasks whose routes share at least one arc. The constraint rules out assigning it to such overlapping routes but different agents. Each operator has a different and randomly generated profit margin for being assigned a demand/route combination, and declares these through its relations. These relations are modelled as soft constraints, and the objective is to maximize their sum.

We randomly generated tasks and routes and simulated the problem starting from a situation where no task is assigned to any agent. The number of agents is 30. A set of 100 tasks is generated randomly; for each task, we generate up to 3 different paths for carrying it. We always report the average performance over 100 runs.

In the first experiment, we want to see how the level of randomness affects the performance of the RSO algorithms. Figure 3.2 shows the average total profit obtained as we varied parameter p from 0 to 1 and kept the number of relations to be left out d always at 1. The shape of this curve does not seem to depend significantly on the value of d. It clearly shows that the performance is best with p close to 1. Thus, in the following experiments, we always set p close to 1.

Next, we are interested in what the best value of parameter d would be for the network resource allocation problem. We run the RSO algorithms with the parameter p set to 1 and with different number of relations to leave out at a local minimum. Figure 3.3 shows the average solution quality reached in 100 runs. The

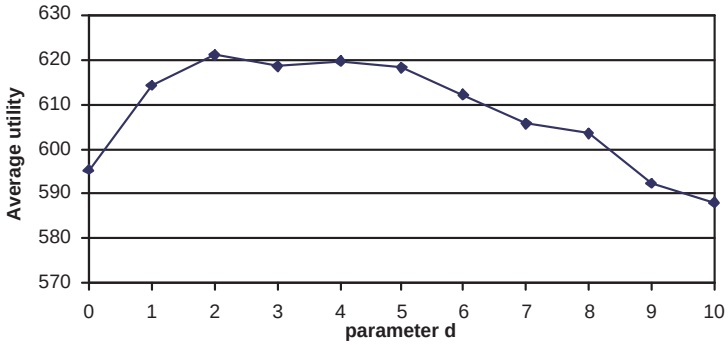

Figure 3.3. *Variation of the average cost of the random subset optimization algorithm on network resource allocation problem, for different values of d.*

results show that the best results are obtained for small d. Thus in the following experiments, we always set the parameter p to 1 and d to 2.

To determine whether the RSO algorithms would be efficient for the network resource allocation problems, we compare the performance of two versions of the RSO algorithms, RSO for Algorithm 3.2 with $p = 1, d = 2$ and ARSO for Adaptive RSO, with the following known local search techniques:

- HC: hill-climbing without randomization (or iterative improvement),included as a baseline comparison. To achieve better results for hill-climbing algorithm, we allow equal moves in the hill-climbing process: the algorithm stops when there is no further improvement for several iterations.

- RHC: hill-climbing with random restarts, similar to the GSAT algorithm [Selman et al. 1992]. For a fair comparison, we set the parameters so that the total number of iterations is similar to the other algorithms. The hill-climbing stops when there is no improvement for several iterations or the number of iterations exceeds 5 times of the number of variables and is then followed by a restart. The total number of iterations is limited to 2000 as for the other algorithms.

- SA: simulated annealing [Kirkpatrick et al. 1983]. The temperature is initialized such that the acceptance rate at the beginning if 97%, and updated according to a geometric cooling schedule with $\alpha = 0.97$; for each temperature value, $n(n-1)$ search steps are performed, where n is the number of variables. The search is terminated when there is no improvement of the evaluation function for five consecutive temperature values, and the acceptance rate of new solutions falls below 1%.

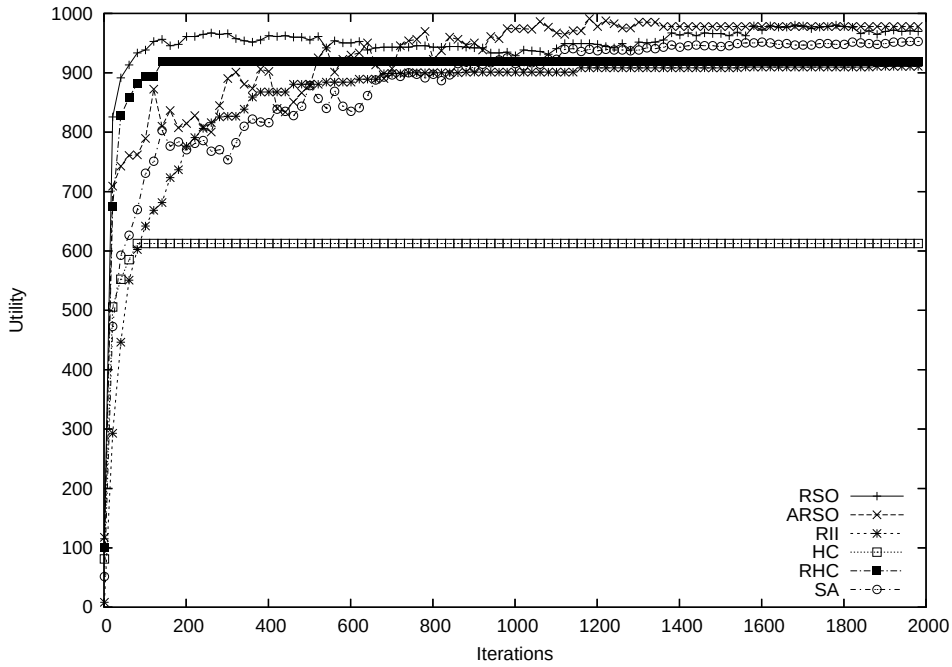

Figure 3.4. *Average utility gain of different local search algorithms on the networks resource allocation problem as a function of the number of steps.*

- RII: Randomized Iterative Improvement ([Zhang et al. 2005]). We tested different versions of the RII algorithm on the network resource allocation problem and took the best one (corresponding to DSA-B in [Zhang et al. 2005]) with probability p set to 0.3.

All the algorithms start from an empty assignment, i.e., all the tasks are unassigned. In Figure 3.4, we show the gain in utility as a function of the number of steps. We observe that ARSO clearly outperforms the other algorithms, while the SA algorithm is slightly better than RSO. However, RSO converges much faster than all other algorithms, and seems to be a clear winner for this application.

3.4.2. Soft Graph Coloring Problem

In a soft graph coloring problem, a weighted graph $G = (V, E)$ is given where V is the set of vertexes, E is the set of weighted edges. Each edge $(u, v) \in E$ is associated with a real value weight $w(u, v)$. Each vertex of the graph can be colored by one of K different colors, let $c(u)$ be the color assigned to the vertex u. An edge (u, v) is violated if u and v have the same color. Given a number K, the goal is to minimize

the total weight of violated edges:

$$\sum_{(u,v)\in E:c(u)=c(v)} w(u,v) \rightarrow min$$

Modeling soft graph coloring in MCOP

A soft graph coloring can be modeled as a MCOP as follows. Each vertex is represented as a variable whose domain is the set of available colors. The set of constraints is empty. One agent a_i is associated with one variable x_i so that it competes with other agents for the values of variables. Agent a_i's relation is defined by:

$$r_{a_i} = \sum_{(x_i,x_j)\in E:c(x_i)\neq c(x_j)} w(x_i,x_j)$$

In other words, each agent a_i's utility is proportional with the sum of all satisfied edges associated with x_i. In this way, maximizing the sum of agent utilities equals to minimizing the total weight of violated edges because:

$$\sum_{a_i} r_{a_i} = 2 \sum_{(u,v)\in E:c(u)\neq c(v)} w(u,v) = 2(\sum_{(u,v)\in E} w(u,v) - \sum_{(u,v)\in E:c(u)=c(v)} w(u,v))$$

In this MCOP the agents do not have to declare their relations, as they depend only on the assignment of values.

As an example, let G is the graph in figure 3.5 with 7 vertices and 2 colors available. The corresponding CMOP will has 7 variables, 7 agents, and the relations are given in the tables 3.1. The example in figure 3.5 shows a local minimum that the SLS can be easily trapped into local minima: variables 2 to 7 will not change their values because that will increase the total violated weight. The only variable that can change value is variable 1, which will not improve the solution quality. On the other hand, the RSO scheme can eventually escape from this local minima if variable 3 is left out of the optimization. In this case, variables 6 and 7 can change their values. However, it is very likely happened that in the next iteration it will go back to the same local minima unless the variable 3 is left out and variable 7 is considered.

Another example is given in the figure 3.6

Experimental results

We run experiments on randomly generated k-coloring problems with 100 vertices and the degree of connectivity $\approx k + 1$ where k is the number of colors used. We let the algorithms run for a large number of steps until either the overall quality does not improve for $10 * n$ steps, or a timeout limit of l steps is reached, where

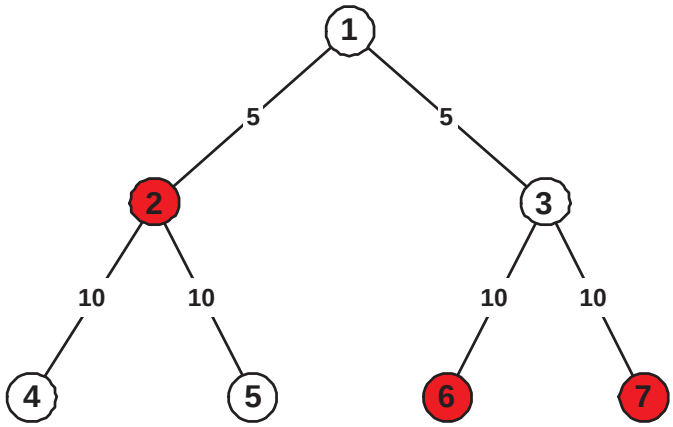

Figure 3.5. *A satisfiable graph with 2 color*

(x_1, x_2)	utility	(x_1, x_3)	utility
(0,0)	0	(0,0)	0
(1,1)	0	(1,1)	0
(0,1)	5	(0,1)	5
(1,0)	5	(1,0)	5

Table 3.1. Utility of agent 1

(x_2, x_1)	utility	(x_2, x_4)	utility	(x_2, x_5)	utility
(0,0)	0	(0,0)	0	(0,0)	0
(1,1)	0	(1,1)	0	(1,1)	0
(0,1)	5	(0,1)	10	(0,1)	10
(1,0)	5	(1,0)	10	(1,0)	10

Table 3.2. Utility of agent 2

(x_3, x_1)	utility	(x_3, x_6)	utility	(x_3, x_7)	utility
(0,0)	0	(0,0)	0	(0,0)	0
(1,1)	0	(1,1)	0	(1,1)	0
(0,1)	5	(0,1)	10	(0,1)	10
(1,0)	5	(1,0)	10	(1,0)	10

Table 3.3. Utility of agent 3

(x_4, x_2)	utility
(0,0)	0
(1,1)	0
(0,1)	10
(1,0)	10

(x_5, x_2)	utility
(0,0)	0
(1,1)	0
(0,1)	10
(1,0)	10

(x_6, x_3)	utility
(0,0)	0
(1,1)	0
(0,1)	10
(1,0)	10

(x_7, x_3)	utility
(0,0)	0
(1,1)	0
(0,1)	10
(1,0)	10

Table 3.4. Utility of agents 4, 5, 6, and 7

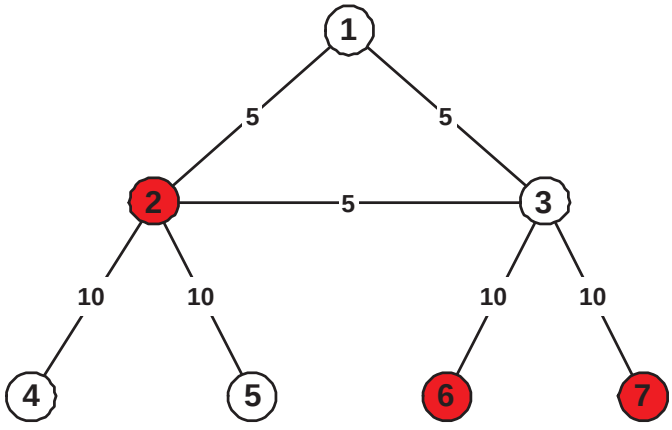

Figure 3.6. *A non-satisfiable graph with 2 color*

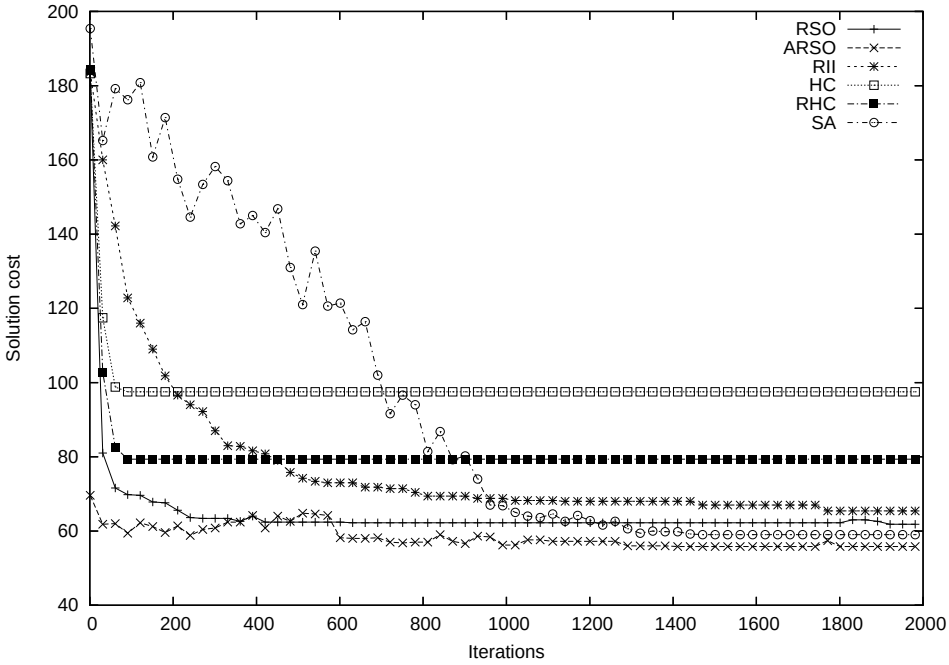

Figure 3.7. *Average solution cost of various optimization algorithms on the soft-k-coloring problem as a function of the number of steps.*

l is variable to indicate the performance of the algorithms. We always report the average performance over 100 runs.

Figure 3.7 shows the results after 2000 iterations. We can observe that SA and RSO eventually reach about the same solution quality, and again ARSO is significantly better than the others. However, RSO converges more than 10 times faster than the RII and simulated annealing algorithms.

3.4.3. Graph Coloring

In a graph coloring problem we are given a weighted graph $G = (V, E)$ where V is the set of vertexes and E the set of edges. The goal is to find a minimum number of colors to color the vertices of the graph so that no two vertices connected by an edge have the same color.

This problem can be modeled as a MCOP where each vertex is represented by a variable whose domain is the set of available colors, and there is an extra variable x_c that represents a bound on the number of colors used. The hard constraints are such that value of every variable must be smaller than x_c; There are two types of soft constraints:

- Soft constraint on x_c: the smaller value the better ($cost = x_c$)

- Soft constraints on edges: two adjacent vertices must have different colors or else there is a penalty of ($cost = 10 * w$). These are modelled as a soft constraint so that local moves that change only one variable can easily move between consistent assignments.

The graph classes we examined in this paper are the following:

- Random graph $G_{n,p}$ or fixed density graph: has n vertices and for each possible pair of vertices, an edge is created independently with probability p. Random graphs are difficult to color optimally. No *a priori* result on the chromatic number or on lower bounds is given for this type of graph except for the case $p = 0.5$ [Johri and Matula 1982].

- DIMACS benchmark suite of graphs [1]: Leighton graphs, k-partite graphs, and class scheduling graphs. These are structured graphs generated with a known chromatic number.

Neighborhood structure

We use the one-exchange neighborhood for comparing all the algorithms in this paper. This neighborhood is used in state-of-the-art algorithms for graph coloring such as Tabu search [Hertz and de Werra 1987] and the hybrid genetic local search in [Fleurent and Ferland 1996]. Two colorings are said to be neighbors if they differ only in the color of a single vertex. In our implementation for COP, a neighboring assignment is generated by randomly selecting a variable involved in a conflict and assigning a random value to it.

Initial solution

In our experiments, all the algorithms begin with the same initial solution which is constructed similar to the procedure used in [Hertz and de Werra 1987]: Given an initial number of colors k and a parameter q, we construct consecutively color sets by greedily computing maximal independent sets $C_1, C_2, ...C_k$ until there are at most q vertices left. The remaining vertices are assigned random values from 1 to k. In the experiments we set the parameter q to $|V|/2$.

In our experiments, we will compare the following algorithms:

- RSO algorithms: our RSO algorithms for graph coloring are based on Algorithm 3.2 and have the following variants:

[1] M. Trick. "COLOR02/03/04: Graph Coloring and its Generalizations" August 2001. http://mat.gsia.cmu.edu/COLOR04

- **RSO-1ex** : the *ChooseNeighbors* function in the algorithm 3.2 is implemented using a one-exchange neighborhood. First, a random variable x_k is chosen among variables involved in conflicts. Then the set of neighbors is the set of assignments which differ with the current assignment in only the value of the variable x_k. The *ChooseRandomSubset()* function chooses a single random variable x_j to leave out of the optimization among the variables sharing a soft constraint with the variable x_k (i.e. x_k and x_j are two adjacent vertices in the graph).
- **RSO-tabu** : This algorithm is a combination of RSO-1ex and tabu search. We use two tabu lists to avoid past choices in both value of the variable and the choice of the soft constraint to leave out. The tabu tenures are chosen randomly with a uniform distribution in [3..10].
- **ARSO-1ex** : This algorithm uses one-exchange neighborhod to implement the *ChooseNeighbors* function in the ARSO algorithm described in the previous section. The decrease of the parameter d during search follows a standard geometric cooling schedule: $d_{i+1} = \rho.d_i$. In our experiment we set $\rho = 0.9$.
- **ARSO-tabu** : in this variant, we combine ARSO with tabu search as described above. Two tabu lists are used to avoid past choices in value of variables and the choice of the subset of soft constraints to leave out. The tabu tenures are set as in RSO-tabu agorithm.

- Tabu search (**Tabu**): The first algorithm we want to compare with our algorithms is the well-known tabu search algorithm ([Glover 1989], [Glover 1990]), ([Hansen and Jaumard 1990]), which is successfully applied to graph coloring by Hertz and de Werra ([Hertz and de Werra 1987]). Some improvements of tabu search with one-exchange neighborhood is still among the state-of-the-art algorithms for graph coloring ([Fleurent and Ferland 1996], [Galinier and Hao 1999]).

  The tabu search algorithm works similar to the framework given in Algorithm 3.1, where the set of neighbors is generated using a one-exchange neighborhood. In each local choice step, the neighbor with fewest conflicts is chosen to be the new solution. The last tt moves are kept in a tabu list to prevent the search from cycling through a small set of suboptimal points. The tabu tenure tt is set proportional to the size of the neighborhood as $\alpha + \delta|N|$ where α is a random number in $[0, 10]$, δ is set to 1 and N is the set of neighbors, similar to the dynamic tabu tenure used in [Taillard 1991]. In our implementation, we use some improvements as proposed in [Hertz and de Werra 1987]: an aspiration function where a tabu move can be allowed if it improves the best

solution found; and use a type of brute force when the number of conflicts is small, checking for a solution with zero-cost with a maximum of three moves.

- Min-conflict heuristic: The second algorithm we compare is the min-conflict heuristic [Minton et al. 1992], implemented within the generic framework given of Algorithm 1, called **MC**. In each local choice step, a random variable is chosen among the conflicting ones and assigned the value having the best improvement of the objective function; ties are broken randomly. The algorithm stops when there are no improvements for a number of consecutive iterations.

- Randomized Iterative Improvement (so called Stochastic local search (**SLS**) in [Zhang et al. 2005]): This algorithm is basically similar to the algorithm 3.1 where in each local choice step a random variable is chosen among the conflicting ones and is assigned a random value. The local move is stochastic in that it is always accepted if the objective function is strictly improved, and is accepted with probability p if the objective function is not improved. We tested different versions of the SLS algorithm on graph coloring problems and took the best one (corresponding to DSA-B in [Zhang et al. 2005]) with the probability p set to 0.35.

- Simulated annealing (**SA**) [Kirkpatrick et al. 1983]: We also implement a simulated annealing algorithm for graph coloring using one-exchange neighborhood. The temperature is initialized such that the acceptance rate at the beginning is 97%, and updated according to a geometric cooling schedule with $\alpha = 0.97$; for each temperature value, γn search steps are performed, where n is the number of variables. In our implementation γ is set to $n/2$ where n is number of variables. The search is terminated when there is no improvement of the evaluation function for five consecutive temperature values, and the acceptance rate of new solutions falls below 1%.

Results

Our experiments have been carried out on a normal Pentium IV PC with 512Mb RAM. For a fair comparison, we let all the algorithms terminate after the same number of iterations. In order to compare the convergence speed of the algorithms, we limit the searches to a relatively small number of iterations for each problem instance.

Random graphs $G_{n,p}$

Tables 3.5 through 3.8 compare the performance of various algorithms on the random graph instances. We see that hill-climbing using the min-conflict heuristic converges quickly but to a solution of low quality as shown in figure 3.8. Tabu search and simulated-annealing eventually converge to optimal or close to optimal quality,

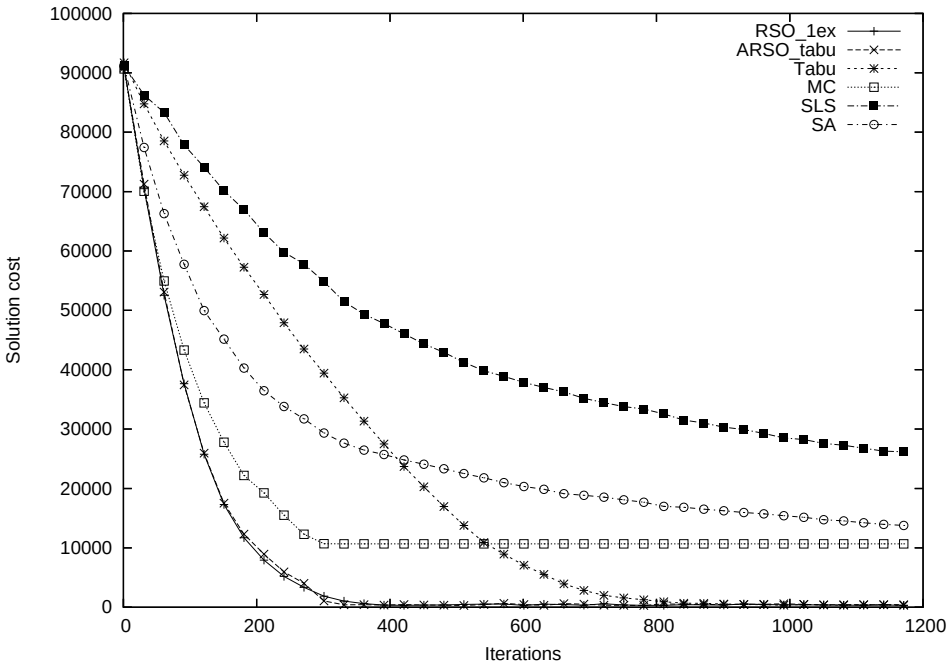

Figure 3.8. *Convergence over time of different heuristics on $G_{250,0.5}$ graph*

iterations	RSO-1ex	ARSO-1ex	RSO-tabu	ARSO-tabu	Tabu	MC
200	24	23	23	22	29	29
500	22	21	21	20	22	29

Table 3.5. Coloring results for $G_{125,0.5}$ graph

but takes quite long to do so. On the large instances of 500 and 1000 nodes (Tables 3.7 and 3.8), RSO reaches a close to optimal solution within 2000-5000 iterations while the other heuristics have made little progress beyond the initial state. Figures 3.8 and 3.9 show the convergence over time of different heuristics, both in the cost of constraint violations (Figure 3.8) and the number of colors used. It illustrates the faster convergence of RSO to results close to the optimal solution.

Structured graphs

Tables 3.9, 3.10, and 3.11 compare the performance of various search algorithms for the different instances of Leighton graphs, k-partite graphs and a scheduling

iterations	RSO-1ex	ARSO-1ex	RSO-tabu	ARSO-tabu	Tabu	MC
500	38	37	38	37	37	40
1000	35	35	36	36	35	39

Table 3.6. Coloring results for $G_{250,0.5}$ graph

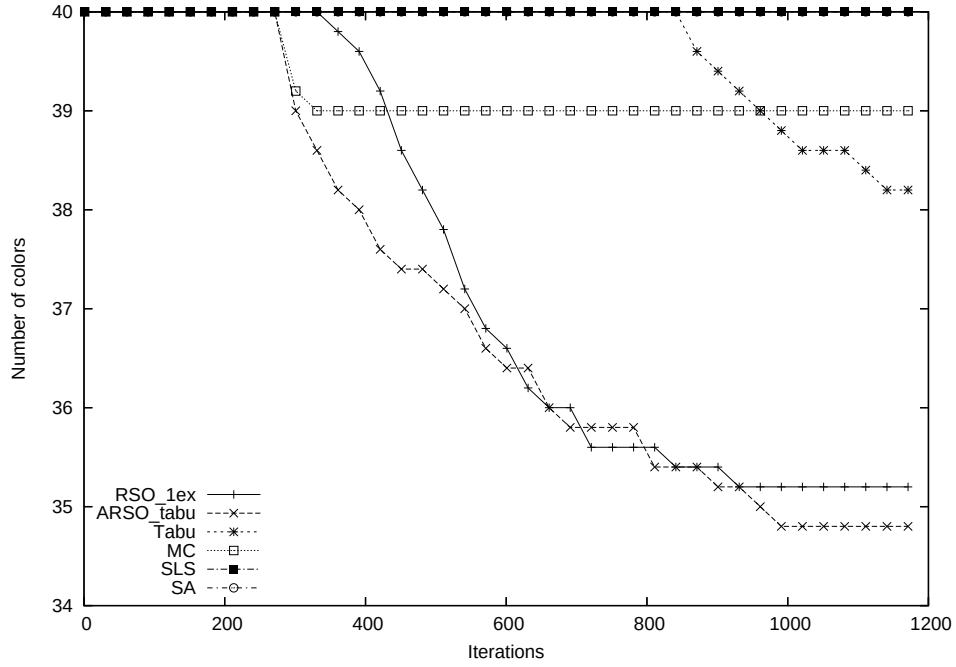

Figure 3.9. *Convergence over time of different heuristics on $G_{250,0.5}$ graph*

iterations	RSO-1ex	ARSO-1ex	RSO-tabu	ARSO-tabu	Tabu	MC
1000	65	64	65	63	70	70
3000	62	62	63	61	68	69

Table 3.7. Coloring results for $G_{500,0.5}$ graph

iterations	RSO-1ex	ARSO-1ex	RSO-tabu	ARSO-tabu	Tabu	MC
5000	107	107	108	106	116	118

Table 3.8. Coloring results for $G_{1000,0.5}$ graph

Graph	iter.	RSO-1ex	ARSO-1ex	RSO-tabu	ARSO-tabu	Tabu	MC
le450_15a.col	2000	19	19	19	18	21	24
le450_15b.col	2000	20	19	19	19	22	24
le450_15c.col	2000	21	21	20	18	25	28
le450_15d.col	2000	21	22	19	19	26	29
le450_25a.col	2000	29	28	27	27	30	37
le450_25b.col	2000	32	31	30	29	31	38
le450_25c.col	2000	33	33	32	31	33	39
le450_25d.col	2000	33	34	32	32	34	39

Table 3.9. Coloring results for Leighton graphs

Graph	iter.	RSO-1ex	ARSO-1ex	RSO-tabu	ARSO-tabu	Tabu	MC
flat300_20_0.col.b	2000	23	22	22	21	26	35
flat300_26_0.col.b	2000	35	35	35	34	36	39
flat300_28_0.col.b	2000	35	36	35	34	36	39
flat1000_50_0.col.b	5000	101	101	100	99	102	110

Table 3.10. Coloring results for k-partite graphs

graph from the DIMACS benchmark collection. Again, we can see that RSO consistently converges faster than Tabu search.

Recovery Performance

In this section, we want to compare the recovery ability of local search algorithms when only a small part of the current solution is corrupted. We performed the test in the following way: a small part (with given ratio) the solution will be randomly initialized whenever a good solution is found. The convergence speed and and solution quality of the algorithms when the ratio is set to 0.1 are shown in figure 3.10. It can be seen that RSO algorithm clearly outperforms other algorithms in the recovery ability. The only comparable algorithm with respect to RSO is the Tabu search.

Graph	iter.	RSO-1ex	ARSO-1ex	RSO-tabu	ARSO-tabu	Tabu	MC
school.col	1000	17	16	16	15	21	26

Table 3.11. Coloring results for class scheduling graphs

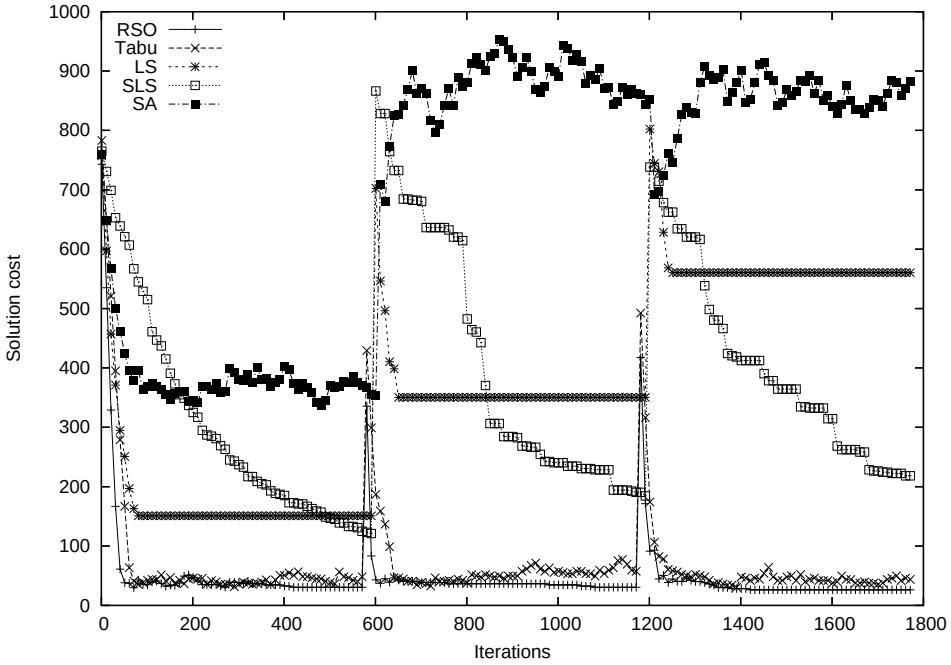

Figure 3.10. *Recovery performance results on* $G_{500,0.5}$ *graphs*

Chapter 4

Size-limited Incentive Compatibility

This chapter introduces the concept of *size-limited incentive-compatibility*, in which the manipulation is made impossible with high probability through computational complexity. Randomization schemes in a local search algorithm can make the prediction of outcomes hard and thus make the local search algorithm size-limited incentive-compatible.

4.1. Definition

The local search procedure can only work correctly if agents accurately report their utilities R. Using side payments, we can create an *incentive-compatible* mechanism where agents are motivated to truthfully report these valuations. Well-known results in game theory ([Green and Laffont 1977]) have shown that all mechanisms for MCOP that are incentive-compatible, individually rational and select the optimal solution must be a kind of VCG mechanism. Thus, there is no mechanism that makes local search incentive-compatible while maintaining individual rationality. Furthermore, Nisan and Ronen [Nisan and Ronen 1999] have shown that a VCG mechanism requires a provably optimal solution. However many practical settings of optimization problems are too large for complete optimization algorithms. Local search algorithms can quickly find a good approximation to an optimal solution. Unfortunately VCG mechanisms cannot be used directly with local search algorithms.

In this thesis I introduce a weaker concept of *size-limited incentive-compatibility* where manipulation is hard through computational complexity. We first introduced a similar solution concept in [Faltings and Nguyen 2005a]. The uncertainty created by randomized local search makes it computationally intractable to evaluate the outcome of an untruthful behavior, thus rendering it uninteresting to agents.

The term *bounded-rationality* is first introduced by Herbert Simon ([Simon 1957, 1982]). He pointed out that in reality, bounded rational agents experience limits in formulating and solving complex problems and in processing information.

We define the concept of a *bounded-rational agent* as follows:

Definition 4.1. (*Bounded-rational agent*) An agent is called bounded rational if it can examine at most C states of the local search before declaring its utilities R_i.

In other words, a bounded-rational agent has only limited computational capacity and it can only compute a limited polynomial number of states of the local search before participating and declaring its utilities to the algorithm.

With this concept we can define the a size-limited incentive-compatible algorithm. The equilibrium concept undefined in the general definition.

Definition 4.2. (*Size-limited incentive-compatibility*)Let p_k be a bound on the probability that a bounded rational agent can predict whether it has an expected utility gain sufficiently large to offset the expected loss from this misdeclaration. A mechanism is *size-limited incentive-compatibile* if for any bound b, there is a k_b such that $p_k < b$ for every $k > k_b$.

In other words, in a size-limited incentive-compatible algorithm, the probability that a bounded-rational agent can predict a beneficial manipulation can be made arbitrarily low given that the problem size is sufficiently large.

Earlier work, such as ([Conitzer and Sandholm 2003]), has proposed using NP-hardness as a protection against manipulation. However, our definition goes further as it requires *in almost all cases*, manipulation requires an amount of computation that is beyond the means of a bounded-rational agent. Any real computational agent is bounded rational for a sufficiently high C.

To apply this concept to local search, we consider both the local search steps (function *LocalChoice*) and the sequence of choices. For local choices, we assume that the space is small enough so that a complete optimization method can be applied. Thus, we ensure incentive-compatibility using a VCG mechanism. This leaves the possibility to use the outcome of a local choice to influence the situation for later choices. Such a manipulation would require the agent to predict the outcome of the search when a manipulation is applied. When search is randomized, we show that with high probability it is computationally hard to predict the outcome of any manipulation of the local search step. we show that when the optimization problem itself is sufficiently complex, we can guarantee an exponential lower bound on the amount of computation required to predict a beneficial manipulation outcome, with probability arbitrarily close to 1. Consequently, the local search mechanism is size-limited incentive compatible.

4.2. Payment budget balance and individual rationality

One problem with the VCG mechanism is that agents generate a surplus of taxes that cannot be returned to them without violating the incentive-compatibility properties. This not only reduces their net utility gain, but also creates incentives for whatever third party receives this gain.

The randomization allows us to make the VCG payment scheme budget balanced by simply paying the payment surplus to the agent a_e that was excluded from the optimization step. Let $VCGtax(a_i)$ be the tax that agent a_i has to pay to the mechanism. Each agent a_i other than a_e pays to a_e the following tax:

$$VCGtax_{-a_e}(a_i) = \sum_{r \in R \setminus (r_i \cup r_e)} \left[r(\overline{v_{A \setminus (a_i \cup a_e)}}) - r(\overline{v_{A \setminus a_e}}) \right]$$

where $\overline{v_{A \setminus (a_i \cup a_e)}}$ is the solution when agents a_i and a_e are left out, $\overline{v_{A \setminus a_e}}$ is the solution when agent a_e is left out.

This can be seen as compensating the agent for the loss of utility it is likely to incur as a consequence of having been left out of the optimization, and does not affect the incentive-compatibility properties:

- for agents other than a_e, it is still best to report their utilities truthfully since they follow a VCG mechanism in a world where a_e does not exist.
- for a_e, its declarations have no effect on the outcome or payments so any declaration is equally good. However, it does not know in advance that it will be excluded, so it still has an interest to make a truthful declaration.

This mechanism is similar to the proposal in [Ephrati and Rosenschein 1991], who proposed giving the surplus to agents that have no interest in the variable being considered. We call such agents *uninterested* agents. The mechanism proposed here applies even when no uninterested agent exists. When there are uninterested agents, optimization can be improved by selecting these to be chosen as excluded agents. More details on the mechanism can be found in [Faltings 2004].

In certain cases the sum of the taxes could be less than the utility loss of the excluded agent, and thus it would not be individually rational for the agent to participate. In fact, no matter what payment scheme is used, whenever the local search step leads to a reduction in total agent utility, there must be at least one agent for which individual rationality is violated. Any randomized local search algorithm will occasionally make such moves, for otherwise it would be susceptible to getting stuck in local optima. Thus, no scheme can guarantee individual rationality at every randomized local choice step.

As the algorithm on the whole improves utility for the community of agents, this does not mean that the local search process as a whole is not individually rational. No agent is systematically disadvantaged by the randomization, and so in expectation the scheme is individually rational for all agents. This is confirmed in our simulations, where individual rationality was almost always satisfied for all agents over the entire optimization. However, no hard guarantee can be given for this.

Let there be m agents and let Δ_{-i} be the utility gain of the agents except i in an optimization step where agent a_i is left out, and δ_i the true utility loss for agent a_i through the choice of the new value $\overline{v_{A\setminus a_i}}$ over the current value $\tilde{v}$. Then, the utility loss for a_i is equal to:

$$\delta_i + (n-2)\Delta_{-i}$$

Then, there also exist $n-1$ equally likely scenarios where the initial value is $\overline{v_{A\setminus a_i}}$ and another agent j is excluded from the optimization. In this case, the utility gain for a_i will be at least δ_i. Now, there are two cases:

- $\delta_i \geq \Delta_{-i}$: in this case, agent a_i's loss is $\leq (n-1)\delta_i$ more than compensated by the expected gain $(n-1)\delta_i$ from the other scenarios.
- $\delta_i < \Delta_{-i}$: in this case, if overall $\Delta_{-i} = \Delta - j = \Delta$, i.e. the surplus is the same no matter what agent is excluded, then agent a_i gets more than compensated by its expected gain of $(n-1)\Delta = (n-1)\Delta_i > \delta_i + (n-2)\Delta_{-i}$

Thus, there is no systematic bias in losses and gains, a fact which is also shown in the simulations.

4.3. Computing VCG taxes

Up to this point, we have assumed that our protocol uses RSO local search algorithm to compute outcome of the mechanism and agents' payments are based on the VCG taxes scheme. The VCG payments in RSO algorithm are computed as follows: each agent pays the utility difference it creates for each relation whenever a variable involved in that relation being optimized. Payments for each relation are only with respect to the values chosen for the last variable involved that relation.

Whenever a variable x_k is chosen for optimizing in a *Localchoice* step, each agent pays for all their relations involved in this variable. Moreover, for each relation, the agent pays only the last time that a variable in the relation being optimized. The excluded agent and its relations are not taken into account in the computation of taxes for the current iteration.

Let tax_0 be a data structure to store local payments and updated every time whenever a variable is optimized, c^{old} and c^{new} are respectively old and new value

assignment for a relation c that involved in the variable x_k. The local payments for agent a_i are computed as follows

$$tax_0(a_i, c) = \sum_{a_j \neq a_i} r_{a_j}(c^{old}) - \sum_{a_j \neq a_i} r_{a_j}(c^{new}) \tag{4.1}$$

The VCG taxes are applied in the last round

$$VCGtax(a_i) = \sum_{c \in \overline{v}} tax_0(a_i, c) \tag{4.2}$$

where $\overline{v}$ is the solution in the last round of RSO algorithm.
We have the following result:

Proposition 4.3. If RSO algorithm found the optimal solution and all relations have been considered during the search, then the taxes computed are Groves taxes.

Proof. If RSO algorithm found the optimal solution v^* and all relations have been considered during the search, then the second part of the Groves taxes in equation 2.6 is computed as

$$\sum_{a_j \neq a_i} r_{a_j}(v^*) \approx \sum_{a_j \neq a_i} r_{a_j}(c^{new})$$

We define the function $h_{-i}(.)$ in the Groves taxes is

$$h_{-i}(.) = \sum_{a_j \neq a_i} r_{a_j}(c^{old})$$

$h_{-i}(.)$ is clearly not depend on the valuations of agent a_i ■

Obviously the optimality assumption in proposition 4.3 is not always hold. However the solution of local search algorithms is often close to the optimal solution (without any guarantee, though). Deviation from RSO protocol is hard to predict: any agent want to manipulate the protocol must estimate how much is the expected gain from the deviation. Thus the computation needed for a possible beneficial manipulation will exceed the computational capacity of a bounded-rational agent.

4.4. Hardness of manipulation

A local search algorithm is in general incomplete and not guaranteed to find a particular optimal solution. Thus, as pointed out by [Nisan and Ronen 2000], non-truthful declarations can drive the local search algorithm to a solution that gives a

manipulating agent a better utility than the truthful declaration. However, effectively using such manipulation requires that the manipulating agent is capable of correctly predicting the effect of a non-truthful utility declaration on the outcome, and compare it against the utility loss it incurs by carrying out the manipulation in one or several local choice steps. We now show that in a randomized local search algorithm and a sufficiently large problem, with high probability (arbitrarily close to 1), such prediction would require an amount of computation that is beyond the capabilities of a bounded-rational agent.

We have the following proposition

Proposition 4.4. Let agent a be a bounded rational agent whose best algorithm for computing the average utility of a set of states with a confidence p requires sampling at least a polynomial fraction of the final states, and let the set of states grow exponentially with the size as observed in the experiments. Then random subset optimization with the VCG tax scheme is a size-limited incentive compatible algorithm for this agent.

Proof. The size of the space of final states grows exponentially with the size of the problem. Thus, the number of final states that the agent has to examine also grows exponentially with the size of the problem. Thus, for any probability p there will be a size k such that this fraction exceeds the agents' computation capacity, and the agent can no longer predict the outcome of a manipulation with sufficient a probability greater than p. ■

To obtain a worst-case result, we assume that a manipulating agent has complete and accurate knowledge of the relations declared by all other agents. Furthermore, we assume that is has access to an oracle that provides it with the most promising manipulation.

The remaining task of the manipulating agent is then to show that the manipulation actually produces a better utility than truthful behavior. As the local search algorithm is randomized, the manipulating agent can only predict an *expected* utility, obtained by considering the probability of certain states and the utilities that the agent would obtain in each of them. The key idea of our argument is that with high probability, the number of states that need to be considered in this calculation will grow exponentially with the size of the problem. Thus, for a certain problem size it will exceed the computational capacity of a bounded rational agent.

To show this result, we first argue that the manipulating agent has to examine a significant fraction of the probability mass of the states to ensure success of the manipulation. This fraction depends on two factors:

- the utility distribution of the problem: if only few states give a significant utility, or if there are strong symmetries so that the state space can be factored, it could be sufficient to sample only a small subset of the states, and

- the desired confidence of the prediction: since a manipulation will mean a certain utility loss to the agent in the search step where it is applied, the manipulation needs to succeed with a certain minimal probability in order to give an increase in expected utility. Depending on the utility distribution, this translates to a certain fraction of the state probability mass that will need to be examined.

Note that both parameters are independent of the size of the search space. Thus, we can assume that the manipulating agent will have to examine a minimal number of states that corresponds to some fraction α of the probability mass of the entire search space.

Next, we show that with high probability, this probability mass is distributed over a number of states that grows exponentially with problem size.

4.5. Average case analysis

In the LocalChoice process (Algorithm 3.2), the utilities associated with each choice of a variable x_i depend on the current assignment v^{old} to the remaining variables. Certain assignments v^{old} could be more favorable to an agent a_j in that it can obtain its favored value with a lower payment. a_j thus has an incentive to manipulate this assignment by making non-truthful utility declarations in earlier search steps.

In order to carry out such manipulation, a_j has to predict the outcome of the local search computation given a manipulation and compare it with the outcome that would be obtained otherwise. We assume that it has perfect knowledge of the utility declarations of all other agents. Obviously, this is the best possible situation for a manipulator so it gives us worst-case guarantees.

To determine whether a certain manipulation is profitable, an agent needs to determine the expected utility it obtains from the entire local search process when it applies the manipulation. To do this, it can either adopt a steady-state view, where it considers the search as a stationary Markov process and determines its optimal policy, or it can explicitly simulate the behavior from the current step forward. We assume that the optimization problem is sufficiently complex that the number of reachable states and transitions is well beyond the capacity C of a bounded rational agent, so that only the simulation of the solving process remains as an option. This is made difficult by the fact that for each random decision, every branch should be considered.

We assume that the utility that an agent can get from any one state is bounded to an interval $[0, D]$. We further assume that a manipulator can bound the utility of the states that have not been simulated to the interval $[0, \beta D]$, with $0 \leq \beta \leq 1$. Let α be the total probability mass of the states that the agent has simulated, i.e. with probability α the actual course of the search is part of simulation. Then

the manipulator can can guarantee success of the manipulation in the best case if all simulated states have utility D for the manipulated case and 0 for the non-manipulated case, and $\alpha D > (1-\alpha)\beta D \Rightarrow \alpha > \frac{\beta}{1+\beta}$ so that $\alpha > \alpha_0 = \frac{\beta}{1+\beta}$. In well-behaved domains without extreme utility variations, we can assume that β does not have an extremely small value. For example, $\beta = 0.1$ would lead to a threshold of $\alpha_0 = 0.1$.

The local search algorithm contains several possibilities for randomization that make manipulation hard:

- random choice of neighbourhood,
- random choice of excluded agent a_e,
- random choice of one among several equivalent local choices.

To make the effect of randomization easy to analyze, I consider only randomizations whose outcomes can be regarded as independent. This is not the case of the random choice of neighbourhood, as it will often be the case that choosing n_1 and then n_2 will lead to the same states as choosing n_2 and then n_1, thus cancelling the randomization effect. Also, local search has to ensure that all neighbourhoods are considered, placing limits on the amount of randomization that can be allowed.

Fortunately, for the choice of excluded agent as well as the choice among equivalent solutions, it does appear reasonable to assume independence of subsequent random choices. We thus make the simplifying assumption that the state space is a tree, i.e. we do not reach the same state through several paths in the simulation. A good local search algorithm that discovers the global optimum with significant probability must be able to access a large fraction of the possible state space. When the problem is sufficiently large so that this total space is big, the algorithm thus can only rarely revisit the same states.

Consider now a simulation of a sequence of states $s_1, ..., s_i$, where s_1 is the starting state of the search. Because of the random decisions each local choice step are independent, the probability of reaching s_i is equal to:

$$p(s_i) = p(s_i|s_{i-1}) \cdots p(s_2|s_1)$$

Let e be the event that in a local choice step, the most common outcome will be chosen at most $1/m$ of the time, i.e. $p(s_j|s_{j-1}) \leq 1/m$. Let p_m be the probability that this event happens at a local search step, and let k be the number of times e happens in a local search of i steps. Then we can give the following bound k_i as a function of the depth i such that $k < k_i$ with probability at most p_t:

$$\begin{aligned} p(k < k_i) &= \sum_{j=0}^{k_i-1} \left(\begin{array}{c} i \\ j \end{array} \right) p_m^j (1-p_m)^{i-j} \\ &\leq (1-p_m)^i \sum_{j=0}^{k_i-1} i^j \left(\frac{p_m}{1-p_m} \right)^j \\ &= (1-p_m)^i \sum_{j=0}^{k_i-1} \left(\frac{ip_m}{1-p_m} \right)^j \end{aligned}$$

Assuming that i is sufficiently large so that $\frac{ip_m}{1-p_m} > 2$:

$$p(k < k_i) \leq (1-p_m)^i \left(\frac{ip_m}{1-p_m} \right)^{k_i} < p_t$$

so that

$$k_i \leq \frac{\log p_t - i \log(1-p_m)}{\log i + \log p_m - \log(1-p_m)}$$

Thus, k_i is $O(i/logi)$ and we can make it arbitrarily large for a sufficiently large i. For example, for $p_m = 0.5$, $i = 1000$ and $p_t = 10^{-9}$, we have

$$k_{1000} \leq \frac{-30 + i}{log_2 i} = \frac{970}{10} = 97$$

In our experiments later in the next section, we have observed $p_4 \simeq 0.8$ (see below) so that 0.5 is a conservative estimate for this example. Note that the size of the problem must be sufficiently large to allow a sufficiently high value for i.

Then we can bound $p(s_i)$ by replacing all k factors that satisfy this bound by $1/m$, and all others by 1:

$$p(s_i) \leq m^{-k}$$

Assume now that we can guarantee that for any state s_i at depth i in the simulation, $k \geq k_i$, the number of states that would have to be generated to obtain at least a probability mass of α would be:

$$n(i) \geq \alpha m^{k_i}$$

Note that even if the simulation runs beyond depth i, it needs to examine at least this number of states at depth i in order to generate the states corresponding to at least α probability mass at the next time instant.

Assuming that at each step, the probability of satisfying the termination condition and stopping is p_s, the probability of a run that reaches depth at least i is $(1-p_s)^{i-1}$. We assume, again as a simplification to obtain a bound, that for all searches that stop in less than i steps, the outcome can be examined without computational cost. Then, in a local search with timeout T, a manipulator would have to examine at least $t(T)$ states:

$$t(T) = max_{i=0}^{T} \left[max(\alpha - (1-p_s)^{i-1}, 0) m^{k_i} \right]$$

For example, if $m = 4$, $p_s = 0.0001$ (expected number of cycles = 10'000) and $\alpha = 0.1$ and $T = 1000$, we have $k_i = 97$ and $t(1000) \geq 10^{54}$ which is certainly out of reach of any computational agent today. Thus, the probability that an agent would have to examine less than 10^{54} states is bounded by $p_t < 10^{-9}$. This is certainly well beyond the capability of any computational agent today.

The key condition to make the manipulation hard is that the problem is sufficiently large. Only in a large problem can we expect the stopping probability p_s to be sufficiently small and the timeout T to be sufficiently large to allow a large i.

4.6. Experiments

To find the global optimum, a local search algorithm has to be able to reach the entire search space. However, eventually it will come close to the global optimum and then remain within a much smaller subspace of nearly optimal states. While it is possible to give a theoretical analysis that shows that with arbitrarily high probability, the probability of reaching any given state is bounded by an exponentially decreasing value, such an analysis requires many independence assumptions that may not hold in practice. In this section, I present the following experimental measurements, obtained from experiments on the network resource allocation problem.

4.6.1. States space of RSO algorithm

Figure 4.1 shows the number of new states discovered in successive cycles of a simulated randomized local search. It initially grows exponentially, but eventually search stabilizes on certain optimal outcomes and thus fails to discover new states. Importantly, however, the total number of states discovered, shown in Figure 4.2, still grows exponentially with problem size: in this example, it muliplies with a factor of about 3 whenever the size increases by 1. Thus, the total number of states has exponential growth with problem size, even though it does not reach the total state space because of the convergence of the algorithm.

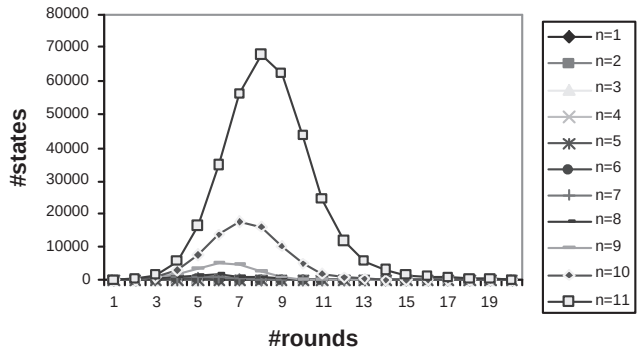

Figure 4.1. *New states discovered in successive cycles of a simulation of local search, for several problem sizes*

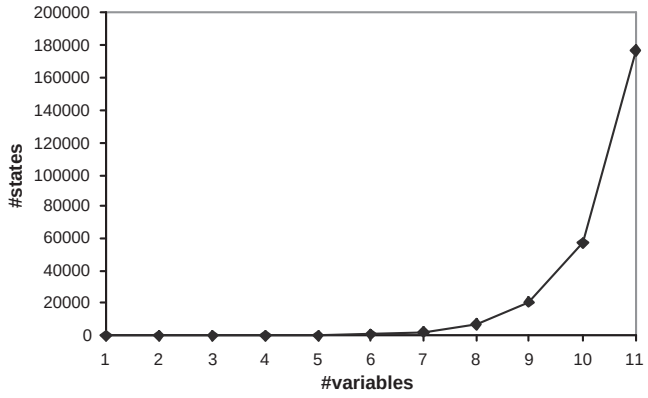

Figure 4.2. *Growth of the total number of states involved in a local search simulation as a function of the problem size*

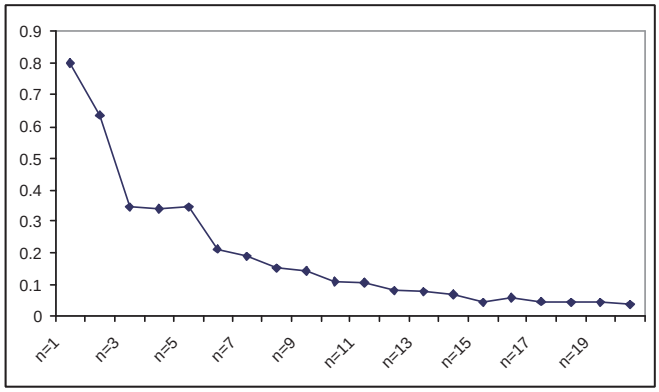

Figure 4.3. *Maximum repetition probability of states in the local search algorithm with RSO scheme*

4.6.2. Branching factors

Another aspect that needs to be shown is that the manipulating agent cannot limit its consideration to only certain states in this space, i.e. that the probability mass is distributed over a large subset of the states. We show this by considering the probabilities of the resulting states at each randomized step. Let p_m denote the probability that at a random branch, each of the branches is taken with probability at most $1/m$. I have measured p_m experimentally (see later section) and have obtained for example for $p_4 \simeq 0.908$, showing that the search process has a significant branching factor. Thus, with high probability the probability mass is distributed among a large number of states.

Next, we want see how often a state is repeated during a local search path. Intuitively if every state is not repeated much in the local search, then the protocol is hard to predict and manipulate. We run experiments on the network resource allocation problem to compute the maximum repetition probability of states during search. Figure 4.3 shows the maximum repetition probability of a state when the number of agents is fixed (5 in this case) and the problem size varies. It can be seen that the repetition probability decreases exponentially stabilizes when the problem size increases.

I also run simulations to estimate the average probability p_m that a Localchoice generates no branch with probability mass larger than $1/m$. We took a histogram over 1000 iterations of the number $r(m)$ that this condition is satisfied for m. Table 4.1 shows the result for $m \leq 10$. From the table, we can estimate for example $p_4 \simeq 0.9$.

m	1	2	3	4	5	6	7	8	9	≥ 10
$r(m)$	0	0	12	88	123	134	138	105	104	296

Table 4.1. Computational results for p_m

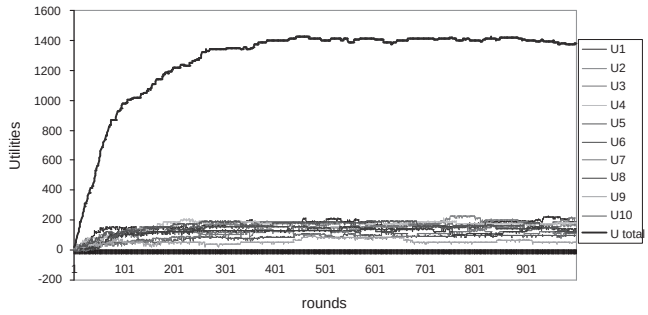

Figure 4.4. *Utilities of agents during local search*

4.6.3. Individual rationality

We are now interested in the actual utilities for each agent, and in particular whether we can guarantee individual rationality. Figure 4.4 shows the utilities of agents during the local search process. In this experiment, we run the local search on random problems with 10 agents and 100 tasks for 1000 rounds. It can be seen that the agents' net utilities are positive and stable when the number of rounds increases. We may want to see that in average, the algorithm is indeed individually rational. We thus run the algorithm for a large number of times and count the number of times that an agent ends up with negative utility. Table 4.2 shows the results after 100 runs. Figure 4.5 shows the final utilities of agents in 100 runs. We can see that there is only 1 time in 100 runs agent 4 had small negative utility. Thus the agents are individually rational *with high probability.*

While we have so far only analyzed relatively simple models, it seems clear that in general the probability mass is very likely to be spread among a large set of states as the size of the problem increases, and thus the method will be size-limited incentive compatible with the parameter being the problem size.

agent	1	2	3	4	5	6	7	8	9	10
times	0	0	0	1	0	0	0	0	0	0

Table 4.2. Number of negative utilities in 100 runs

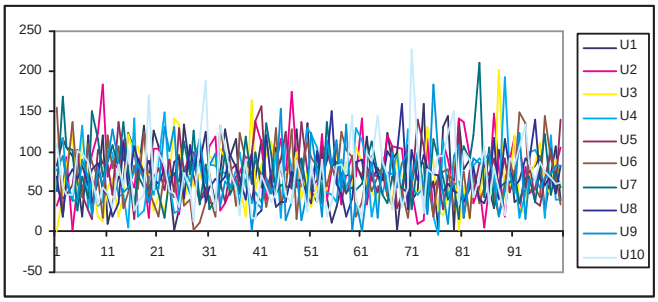

Figure 4.5. *Utilities of agents in 100 runs*

Chapter 5

Conclusions

This thesis deals with Multi-agent Constraint Optimization Problems, in particular, we aim at designing local search algorithms that take into account both optimization and incentives aspects. Our work focuses on satisfactory and sub-optimal outcomes rather than an optimal outcome.

Local search algorithms are often the only possibility to solve large optimization problems. In earlier work, analysis has often focussed only on the quality of the final solution that can be obtained. However, in our experience for some of the benchmark problems, the best algorithms take days of computation to converge. For general optimization problems, simulated annealing often achieves the best results, but has the drawback that it converges very slowly. There are many applications where the low quality of simple hill-climbing may not be acceptable, but the such slow convergence is not acceptable either. We attribute this slow convergence to the fact that the randomization techniques used "throw out the baby with the bathwater": they impose steps that most often lead away from the optimal solution rather than varying the direction of search in a more measured way.

Randomization has been shown to be key to obtaining results that are close to the optimal solution. This strategy has been particularly successful for SAT problems. By drawing an analogy with techniques that have been successful for SAT problems, we developed a new technique of random subset optimization that empirically converges much faster than known algorithms while obtaining solutions of equal or better quality.

Finding an optimal coordination between multiple self-interested agents is a problem that occurs frequently in practice. Incentive-compatibility is an essential property to ensure meaningful results of such an optimization. Previous work has shown the applicability of VCG mechanisms to such problems. However, it requires provably optimal solutions to the NP-hard optimization problem and thus cannot be applied to large problems.

Our work is based on the observation that in real life, the potential for manip-

ulation is limited by uncertainty and risk. This uncertainty makes it difficult for a manipulator to predict the consequences of his manipulation and thus makes attempts at manipulating it uninteresting. Similar uncertainty exists in local search algorithms where randomization is necessary to escape local optima. We have analyzed a scheme for randomization and shown that in sufficiently large problems, it creates a large amount of uncertainty so that simulating a sufficient part of the possible outcomes quickly surpasses the computational capacity of any real computational agent. Problems that are too small for this result to apply can likely be addressed by VCG mechanisms with complete optimization.

In the following we present our major contributions of this dissertation, and we conclude with some final remarks.

1. RSO algorithms (chapter 3): RSO is interesting for multi-agent problems, fast convergence can be exploited for problems with time constraints. It can also be corporated with other local search methods as a general metaheuristics. It is surprising that this was not discovered before. Several extensions of the RSO algorithm is proposed (section 3.2): Adaptive RSO , SA-RSO, and RSO-tabu.

2. Comparison of RSO algorithms with other local search methods on several applications: Network Resource Allocation, Soft Graph Coloring, and Graph Coloring (section 3.4). The comparison is performed with respect to performance criteria, convergence time criteria, and solution recovery criteria.

3. Size-limited incentive compatibility (chapter 4): We define a new concept of incentive compatibility for bounded rational agents settings that takes into account their limited computational capacities. We show that RSO algorithms can be applied to have this property. Though we do not give a complete theoretical proof, the experimental analysis results can provide intuitive answers.

For further research, there are still lot of works need to be done. We are now conducting further experiments to determine the influence of different parameters and variations on the performance of the RSO algorithms.

Size-limited incentive-compatibility requires further work. Complexity theory does not provide useful tools are there is little analysis of best- or average-case complexity. The worst case is not helpful for protecting against manipulation. We consider that experimental analysis as we have presented can be a reasonable alternative. However, we cannot guarantee that algorithms to manipulate such a scheme do not exist. This situation is similar to that of NP-completeness and is not easy to resolve.

While no randomized local search algorithm can guarantee individual rationality, we found that it seems to be satisfied with high probability. It would be interesting to analyze the individual rationality properties of local search schemes to obtain probabilistic guarantees similar to those for non-manipulability.

The most important weakness of the current scheme is that the parameter that needs to be varied to guarantee size-limited incentive-compatibility is the size of the problem. It would be much better if we had a mechanism that could guarantee high manipulation complexity even for small problems through suitable randomization of this choice, similarly to certain cryptographic hash functions.

Bibliography

E. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Inc., New York, NY, USA, 1997. pages 21

A. Archer and E. Tardos. Truthful mechanisms for one-parameter agents. In *IEEE Symposium on Foundations of Computer Science*, pages 482–491, 2001. pages 33

K. J. Arrow. *Social Choice and Individual Values*. New Haven, Yale University, 2nd edition, 1963. pages 32

S. Barberà and M. O. Jackson. Strategy-proof exchange. *Econometrica*, 63(1):51–87, 1995. pages 32

R. Battiti and M. Protasi. Reactive local search for the maximum clique problem. *Algorithmica*, 29(4):610 – 637, 2001. pages 26

U. Bertele and F. Brioschi. *Nonserial Dynamic Programming*. Academic Press, Inc., Orlando, FL, USA, 1972. ISBN 0120934507. pages 19

S. Bikhchandani, S. Chatterjee, and A. Sen. Incentive compatibility in multi-unit auctions, 2003. pages 33

A. Bogomolnaia and H. Moulin. A new solution to the random assignment problem. *Journal of Economic Theory*, 100(2):295–328, October 2001. pages 33

V. Cerny. Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm. *Journal of Optimization, Theory and Applications*, 45(1):41–51, 1985. pages 24

E. H. Clarke. Multipart pricing of public goods. *Public Choice*, 11:17–33, 1971. pages 31, 32, 33

W. Conen and T. Sandholm. Partial-revelation VCG mechanism for combinatorial auctions. In *Proceedings of AAAI-02*, pages 367–372, Edmonton, Canada, 2002. pages 31

V. Conitzer and T. Sandholm. Complexity of mechanism design. In *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI)*, Edmonton, Canada, August 1-4 2002. pages 33

V. Conitzer and T. Sandholm. Universal voting protocol tweaks to make manipulation hard, 2003. pages 62

C. d'Aspremont and L.-A. Gérard-Varet. Incentives and incomplete information. *J. of Public Economics*, 11:25–45, 1979. pages 30

M. Dorigo. Optimization, learning and natural algorithms, 1992. pages 29

M. Dorigo, G. D. Caro, and L. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172, 1999. pages 28

M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, USA, 2004. pages 29

R. Dorne and J.-K. Hao. A new genetic local search algorithm for graph coloring. *Lecture Notes in Computer Science*, 1498:745–754, 1998. pages 28

E. Ephrati and J. Rosenschein. The clarke tax as a consensus mechanism among automated agents. In *Proceedings of the National Conference on Artificial Intelligence*, pages 173–178, San Jose, California, July 1991. pages 34, 63

B. Faltings. A budget-balanced, incentive-compatible scheme for social choice. In *Agent-mediated E-commerce (AMEC) VI*, 2004. pages 63

B. Faltings and Q. H. Nguyen. Multi-agent coordination using local search. In *Proceedings of IJCAI05*, pages 953–958, Edinburgh, Scotland, Aug 2005a. pages 61

B. Faltings and Q. H. Nguyen. Random subset optimization. In *Proceedings of Second International Workshop on Local Search Techniques in Constraint Satisfaction LSCS-05*, pages 32–45, Barcelona, Spain, Oct 2005b. pages 37

B. Faltings and Q. H. Nguyen. Random subset optimization. In *Proceedings of ECAI06*, pages 88–92, 2006. pages 37

C. Fleurent and J. A. Ferland. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63:437–461, 1996. pages 26, 28, 54, 55

G. B. Fogel, W. V. Porto, D. G. Weekes, D. B. Fogel, R. H. Griffey, J. A. Mcneil, E. Lesnik, D. J. Ecker, and R. Sampath. Discovery of RNA structural elements using evolutionary computation. *Nucl. Acids Res.*, 30(23):5310–5317, December 2002. pages 16

P. Galinier and J. K. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, (3):379–397, 1999. ISSN 0305-0548. pages 26, 28, 55

M. Gendreau, G. Laporte, and J.-Y. Potvin. *The Vehicle Routing Problem*, volume 9, chapter Metaheuristics for the vehicle routing problem, pages 129—-154. SIAM Series on Discrete Mathematics and Applications, 2001. pages 16

A. Gibbard. Manipulation of voting schemes: A general result. *Econometrica*, 41 (4):587–601, July 1973. pages 31, 32

F. Glover. Tabu search—part I. *ORSA Journal on Computing*, 1(3):190–206, Summer 1989. pages 25, 55

F. Glover. Tabu search– part II. *ORSA Journal on Computing*, 2(1):4–32, 1990. pages 25, 55

F. Glover and M. Laguna. *Tabu Search*. Kluwer, Norwell, MA., 1997. pages 26

D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989. ISBN 0201157675. pages 27

J. Green and J.-J. Laffont. Characterization of satisfactory mechanisms for the revelation of preferences for public goods. *Econometrica*, 45(2):427–38, March 1977. pages 30, 31, 32, 34, 61

J. Green and J.-J. Laffont. On coalition incentive compatibility. *Review of Economic Studies*, 46(2):243–54, April 1979. pages 32

T. Groves. Incentives in teams. *Econometrica*, 41(4):617–31, July 1973. pages 31, 32, 33

P. Hansen and B. Jaumard. Algorithms for the maximum satisfiability problem. *Computing*, 44:279–303, 1990. pages 25, 55

A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4):345–351, 1987. pages 26, 54, 55

J. Holland. *Adaptation in Nature and Artificial Systems*. Univ. of Michigan Press, reprinted by MIT Press, 1992. pages 27

H. H. Hoos. On the run-time behaviour of stochastic local search algorithms for SAT. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI'99)*, pages 661–666, Orlando, Florida, 1999. pages 40, 42

L. Hurwicz. *On Informationally Decentralized Systems*, pages 297–336. 1972. pages 30

M. O. Jackson. Bayesian implementation. *Econometrica*, 59(2):461–77, March 1991. pages 30

A. Johri and D. W. Matula. Probabilistic bounds and heuristic algorithms for coloring large random graphs. Technical report, Southern Methodist University, Dallas, Texas, 1982. pages 54

E. Kaplansky and A. Meisels. Distributed personnel scheduling - negotiation among scheduling agents. *Annals of Operations Research*, 2005. pages 21

S. A. Kauffman and W. G. Macready. Technological evolution and adaptive organizations. *Complexity*, 26(2):26–43, March 1995. pages 24

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983. pages 24, 48, 56

D. Lehmann, L. I. O'Callaghan, and Y. Shoham. Truth revelation in approximately efficient combinatorial auctions. *J. ACM*, 49(5):577–602, 2002. ISSN 0004-5411. pages 32, 33

S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973. pages 16

W. Macready, A. Siapas, and S. Kauffman. Criticality and parallelism in combinatorial optimization. *Science*, 271:56, 1996. pages 24

R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 438–445, Washington, DC, USA, 2004. IEEE Computer Society. pages 42

R. P. McAfee. A dominant strategy double auction. Working Papers 734, California Institute of Technology, Division of the Humanities and Social Sciences, May 1990. pages 32

D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *In Proceedings of the 14th National Conference on Artificial Intelligence*, pages 321–326. AAAI Press / The MIT Press, 1997. pages 16, 24

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, June 1953. pages 24

Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer, December 2004. pages 27

P. Mills and E. Tsang. Guided local search for solving sat and weighted max-sat problems. *J. Autom. Reason.*, 24(1-2):205–223, 2000. ISSN 0168-7433. pages 27

S. Minton, M. D. Johnston, A. B. Philips, and P. Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artif. Intell.*, 58(1-3):161–205, 1992. ISSN 0004-3702. pages 56

H. Moulin. On strategy-proofness and single peakedness. *Public Choice*, 35:437–455, 1980. pages 33

A. Mu'alem and N. Nisan. Truthful approximation mechanisms for restricted combinatorial auctions: extended abstract. In *Proceedings of the Eighteenth national conference on Artificial intelligence*, pages 379–384. American Association for Artificial Intelligence, 2002. ISBN 0-262-51129-0. pages 32, 33

R. B. Myerson. Incentive compatibility and the bargaining problem. *Econometrica*, 47(1):61–73, Jan. 1979. pages 31

R. B. Myerson and M. A. Satterthwaite. Efficient mechanisms for bilateral trading. *Journal of Economic Theory*, 29(2):265–281, April 1983. pages 32

Q. H. Nguyen and B. Faltings. Randomization for multi-agent constraint optimization. In *Proceedings of CP2005*, pages 864–864, Sitges, Spain, Oct 2005. pages 37

N. Nisan and A. Ronen. Algorithmic mechanism design (extended abstract). In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 129–140. ACM Press, 1999. ISBN 1-58113-067-8. pages 34, 61

N. Nisan and A. Ronen. Computationally feasible VCG mechanisms. In *Proceedings of the 2nd ACM conference on Electronic commerce*, pages 242–252. ACM Press, 2000. ISBN 1-58113-272-7. pages 32, 33, 65

D. C. Parkes, J. Kalagnanam, and M. Eso. Achieving Budget-Balance with Vickrey-Based Payment Schemes in Exchanges. In *Proceedings of IJCAI-01*, pages 1161–1168, 2001. pages 32

D. C. Parkes and L. H. Ungar. Iterative combinatorial auctions: Theory and practice. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 74–81. AAAI Press / The MIT Press, 2000. ISBN 0-262-51112-6. pages 31

T. Sandholm. Automated mechanism design: A new application area for search algorithms. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP-2003)*, pages 19–36, 2003. pages 33

M. Satterthwaite. Strategy-proofness and arrow's conditions: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10:187–217, 1975. pages 32

T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In C. Mellish, editor, *IJCAI'95: Proceedings International Joint Conference on Artificial Intelligence*, Montreal, 1995. pages 19

J. Schummer. Almost-dominant strategy implementation. Discussion Papers 1278, Northwestern University, Center for Mathematical Studies in Economics and Management Science, Nov 1999. pages 33

B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *In Proceedings of the 12th National Conference on Artificial Intelligence*, pages 337–343, Menlo Park, CA, USA, 1994. pages 16, 24, 37

B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *In Proceedings of the 10th National Conference on Artificial Intelligence*, pages 440–446, Menlo Park, CA, USA, 1992. pages 16, 24, 48

H. A. Simon. *Models of Man, Social and Rational: Mathematical Essays on Rational Human Behavior in a Social Setting*, chapter A Behavioral Model of Rational Choice. New York: Wiley, 1957. pages 62

H. A. Simon. *Models of Bounded Rationality*, volume 1 and 2, chapter A Behavioral Model of Rational Choice, pages 239—-258. Cambridge: MIT Press, 1982. pages 62

T. Stützle and H. Hoos. Max–min ant system. *Future Generation Computer Systems*, 16(8):889—-914, 2000. pages 29

E. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17(4-5):443–455, 1991. pages 26, 44, 55

W. Vickrey. Counterspeculation, auctions and competitive sealed tenders. *Journal of Finance*, 16(1):8–37, 1961. pages 31, 32, 33

W. Zhang, G. Wang, Z. Xing, and L. Wittenburg. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artif. Intell.*, 161(1-2):55–87, 2005. pages 24, 46, 49, 56

Curriculum Vitae

Quang-Huy Nguyen
Artificial Intelligence Laboratory (LIA)
School of Computer and Communication Sciences (IC)
Swiss Federal Institute of Technology in Lausanne (EPFL)
CH-1015, Ecublens, Lausanne, Switzerland
Email: quanghuy.nguyen@epfl.ch
Web: http://liawww.epfl.ch/~huy/

Education

2004 – 05/2008	**PhD Candidate** at Artificial Intelligence Laboratory, School of Computer and Communication Sciences, Swiss Federal Institute of Technology in Lausanne (EPFL), Switzerland.
2003 – 2004	**Doctoral School** in Computer, Communication and Information Sciences, EPFL, Switzerland.
1999 – 2001	**M.S.** in Computer Science, Francophone Institute of Computer Science (IFI), Hanoi, Vietnam.
1994 – 1999	**B.S.** in Computer Science (minimum: 5 years), Hanoi University of Technology (HUT), Hanoi, Vietnam.

Awards and Honors

2002 – 2003	**Swiss Federal Scholarship** for foreign students to do research in Switzerland.
2001	**2nd top graduate M.S. student**, IFI, Vietnam.
1999	**Award** from the Young Researchers Symposium, HUT, Vietnam.
1999	**3rd prize** in the National Olympiad in Informatics for students, Vietnam.
1993,1994	**Prizes** in the National Informatics Olympiads for high school students, Vietnam.

Publications

Refereed Conference/Workshop Papers

1. B. Faltings and Q. H. Nguyen. Random Subset Optimization. *Proceedings of ECAI-06*, Riva del Garda, Italy, August 2006, pp. 88-92.

2. Q. H. Nguyen and B. Faltings. Randomization for Multi-agent Constraint Optimization. *Proceedings of CP-2005*, Sitges, Spain, October 2005, pp. 864 - 864.

3. B. Faltings and Q. H. Nguyen. Multi-agent Coordination using Local Search. *Proceedings of IJCAI-05*, Edinburgh, Scotland, August 2005, pp. 953-958.

4. B. Faltings and Q. H. Nguyen. Random Subset Optimization. *Proceedings of LSCS-05*, Barcelona, Oct, 2005, pp. 32-45.

5. T. T. Nguyen, Q. H. Nguyen and H. D. Nguyen. Counter-propagation neural network and its application in color classifying in industry. *Proceeding of the Conference of Math Application*, Hanoi, December 1999 (in Vietnamese).

6. T. T. Nguyen, H. D. Nguyen, Q. H. Nguyen, P. H. Cung and C. N. Mai. ENDODIAG - Un experimental system for disease diagnosis base on images. *Proceeding of the International Symposium on Medical Informatics And Fuzzy Technology*, pp. 257-262, Hanoi, August 1999.

7. T. T. Nguyen, Q. H. Nguyen, D. H. Le. Fuzzy multilayer neural network for Optical Character Recognition. *Proceeding of the Vietnam-Japan Bilateral Symposium on Fuzzy Systems and Applications (VJFUZZY'98)*, Halong bay, 1998.

Books & chapters

1. T. T. Nguyen and Q. H. Nguyen. *Programming exercises in C.* Science & Technique Press, 1999 (in Vietnamese).

2. T. T. Nguyen, H. D. Nguyen, Q. H. Nguyen, D. L. Anh. *Introduction to the Linux operating system.* Science & Technique Press, 2000 (in Vietnamese).

Personal Information

Date of birth:	November, 29th 1975
Nationality:	Vietnamese
Sex:	Male
Marital Status:	Married, 1 child

Languages

English:	Very good knowledge (level C1 in the European Language Scale)
French:	Very good knowledge (level C1 in the European Language Scale)
Vietnamese:	Mother tongue

Lausanne, May 09, 2008

✠